GRLIB IP Core User’s Manual

Version 1.3.1 - B4135, August 2013

Table of contents

1 Introduction

1.1 Scope

This document describes specific IP cores provided with the GRLIB IP library. When applicable, the cores use the GRLIP plug&play configuration method as described in the 'GRLIB User's Manual'.

1.2 IP core overview

The tables below lists the provided IP cores and their AMBA plug&play device ID. The columns on the right indicate in which GRLIB distributions a core is available. GPL is the GRLIB GNU GPL (free) distribution, COM is the commercial distribution, FT the full fault-tolerant distribution and FT-FPGA is the GRLIB release targeted for raditation-tolerant programmable devices. Some cores can only be licensed separately or as additions to existing releases, this is marked in the *Notes* column. Contact Aeroflex Gaisler for licensing details.

Note: The open-source version of GRLIB includes only cores marked with "Yes" in the GPL column.

Note: IP core FT features are only supported in FT or FT-FPGA distributions.

Note: For encrypted RTL, contact Aeroflex Gaisler to ensure that your EDA tool is supported by GRLIB for encrypted RTL. Supported tools are listed in the GRLIB IP Library user's manual.

Table 1. Processors and support functions

Name	**Function**	**Vendor:Device**	**GPL**	**COM**	**FT**	**FT-FPGA**	**Notes**
LEON3	SPARC V8 32-bit processor	0x01 : 0x003	Yes	Yes	Yes	Yes	
LEON3FT	Fault-tolerant SPARC V8 32-bit Processor	0x01 : 0x053	No	No	Yes	Yes	2)
DSU3	Multi-processor Debug support unit (LEON3)	0x01 : 0x004	Yes	Yes	Yes	Yes	
LEON4	SPARC V8 32-bit processor	0x01 : 0x048	No	No	No	No	1)
L4STAT	LEON4 statistics unit	0x01 : 0x047	No	No	No	No	1), 3)
DSU4	Multi-processor Debug support unit (LEON4)	0x01 : 0x049	No	No	No	No	1)
LEON3/4 CLK2x	LEON processor double clocking (includes special LEON entity, interrupt controller and qualifier unit)	-	No	Yes	Yes	Yes	
CLKGEN	Clock generation	-	Yes	Yes	Yes	Yes	
DIV32	Divider module	-	Yes	Yes	Yes	Yes	
GPTIMER	General purpose timer unit	0x01 : 0x011	Yes	Yes	Yes	Yes	
GRCLKGATE	Clock gate unit	0x01 : 0x02C	No	Yes	Yes	Yes	
GRTIMER	General purpose timer unit	0x01 : 0x038	No	Yes	Yes	Yes	
GRFPU / GRFPC	High-performance IEEE-754 Floating-point unit with floating-point controller to interface LEON	-	No	No	No	No	1), 2)
GRFPU-Lite / GRFPC-lite	Low-area IEEE-754 Floating-point unit with floating point controller to interface LEON	-	No	No	No	No	1), 2)
IRQMP	Multi-processor Interrupt controller	0x01 : 0x00D	Yes	Yes	Yes	Yes	
IRQ(A)MP	Multi-processor Interrupt controller	0x01 : 0x00D	No	Yes	Yes	Yes	
MUL32	32x32 multiplier module	-	Yes	Yes	Yes	Yes	
MULTLIB	High-performance multipliers	-	Yes	Yes	Yes	Yes	

1) Available as separate package or as addition to existing releases.
2) Only available as netlist or encrypted RTL
3) Always included with LEON4 license
4) Requires PHY for selected target technology. Please see IP core documentation for supported technologies.

Table 2. Memory controllers and supporting cores

Name	Function	Vendor:Device	GPL	COM	FT	FT-FPGA	Note
DDRSPA	Single-port 16/32/64 bit DDR controller	0x01 : 0x025	Yes	Yes	Yes	Yes	4)
DDR2SPA	Single-port 16/32/64-bit DDR2 controller	0x01 : 0x02E	Yes	Yes	Yes	Yes	4)
MCTRL	8/16/32-bit PROM/SRAM/SDRAM controller	0x04 : 0x00F	Yes	Yes	Yes	Yes	
SDCTRL	32-bit PC133 SDRAM controller	0x01 : 0x009	Yes	Yes	Yes	Yes	
SRCTRL	8/32-bit PROM/SRAM controller	0x01 : 0x008	Yes	Yes	Yes	Yes	
SSRCTRL	32-bit Synchronous SRAM (SSRAM) controller	0x01 : 0x00A	No	Yes	Yes	Yes	
FTMCTRL	8//32-bit PROM/SRAM/SDRAM controller w. RS/ BCH EDAC	0x01 : 0x054	No	No	Yes	Yes	
FTSDCTRL	32/64-bit PC133 SDRAM Controller with EDAC	0x01 : 0x055	No	No	Yes	Yes	
FTSDCTRL64	64-bit PC133 SDRAM controller with EDAC	0x01 : 0x058	No	No	Yes	Yes	
FTSRCTRL	8/32-bit PROM/SRAM/IO Controller w. BCH EDAC	0x01 : 0x051	No	No	Yes	Yes	
FTSRCTRL8	8-bit SRAM / 16-bit IO Memory Controller with EDAC	0x01 : 0x056	No	No	Yes	Yes	
NANDFCTRL	NAND Flash memory controller	0x01 : 0x059	No	Yes	Yes	Yes	
SPIMCTRL	SPI Memory controller	0x01 : 0x045	Yes	Yes	Yes	Yes	
AHBSTAT	AHB status register	0x01 : 0x052	Yes	Yes	Yes	Yes	
MEMSCRUB	Memory scrubber	0x01 : 0x057	No	No	Yes	Yes	

1) Available as separate package or as addition to existing releases.
2) Only available as netlist or encrypted RTL
3) Always included with LEON4 license
4) Requires PHY for selected target technology. Please see IP core documentation for supported technologies.

Table 3. AMBA Bus control

Name	Function	Vendor:Device	GPL	COM	FT	FT-FPGA	Note
AHB2AHB	Uni-directional AHB/AHB Bridge	0x01 : 0x020	No	Yes	Yes	Yes	
AHBBRIDGE	Bi-directional AHB/AHB Bridge	0x01 : 0x020	No	Yes	Yes	Yes	
AHBCTRL	AMBA AHB bus controller with plug&play	-	Yes	Yes	Yes	Yes	
APBCTRL	AMBA APB Bridge with plug&play	0x01 : 0x006	Yes	Yes	Yes	Yes	
AHBTRACE	AMBA AHB Trace buffer	0x01 : 0x017	Yes	Yes	Yes	Yes	
GRIOMMU	I/O Memory management unit	0x01 : 0x04F	No	Yes	Yes	Yes	1)

1) Available as separate package or as addition to existing releases.
2) Only available as netlist or encrypted RTL
3) Always included with LEON4 license
4) Requires PHY for selected target technology. Please see IP core documentation for supported technologies.

Table 4. PCI interface

Name	Function	Vendor:Device	GPL	COM	FT	FT-FPGA	Note
GRPCI2	Advanced 32-bit PCI bridge	0x01 : 0x07C	No	Yes	Yes	Yes	
PCITARGET	32-bit target-only PCI interface	0x01 : 0x012	Yes	Yes	Yes	Yes	
PCIMTF/GRPCI	32-bit PCI master/target interface with FIFO	0x01 : 0x014	Yes	Yes	Yes	Yes	
PCITRACE	32-bit PCI trace buffer	0x01 : 0x015	Yes	Yes	Yes	Yes	
PCIDMA	DMA controller for PCIMTF	0x01 : 0x016	Yes	Yes	Yes	Yes	
PCIARB	PCI Bus arbiter	0x04 : 0x010	Yes	Yes	Yes	Yes	

Table 5. On-chip memory functions

Name	Function	Vendor:Device	GPL	COM	FT	FT-FPGA	Note
AHBRAM	Single-port RAM with AHB interface	0x01 : 0x00E	Yes	Yes	Yes	Yes	
AHBDPRAM	Dual-port RAM with AHB and user back-end interface	0x01 : 0x00F	Yes	Yes	Yes	Yes	
AHBROM	ROM generator with AHB interface	0x01 : 0x01B	Yes	Yes	Yes	Yes	
FTAHBRAM	RAM with AHB interface and EDAC protection	0x01 : 0x050	No	No	Yes	Yes	
L2CACHE	Level-2 cache controller	0x01 : 0x04B	No	No	No	No	1), 3)
REGFILE_3P	Parametrizable 3-port register file	-	Yes	Yes	Yes	Yes	
SYNCRAM	Parametrizable 1-port RAM	-	Yes	Yes	Yes	Yes	
SYNCRAM_2P	Parametrizable 2-port RAM	-	Yes	Yes	Yes	Yes	
SYNCRAM_DP	Parametrizable dual-port RAM	-	Yes	Yes	Yes	Yes	

1) Available as separate package or as addition to existing releases.
2) Only available as netlist or encrypted RTL
3) Always included with LEON4 license
4) Requires PHY for selected target technology. Please see IP core documentation for supported technologies.

Table 6. Serial communication

Name	Function	Vendor:Device	GPL	COM	FT	FT-FPGA	Note
AHBUART	Serial/AHB debug interface	0x01 : 0x007	Yes	Yes	Yes	Yes	
AHBJTAG	JTAG/AHB debug interface	0x01 : 0x01C	Yes	Yes	Yes	Yes	
APBPS2	PS/2 host controller with APB interface	0x01 : 0x060	Yes	Yes	Yes	Yes	
APBUART	Programmable UART with APB interface	0x01 : 0x00C	Yes	Yes	Yes	Yes	
CAN_OC	Opencores CAN 2.0 MAC with AHB interface	0x01 : 0x019	Yes	Yes	Yes	Yes	
GRCAN	CAN 2.0 Controller with DMA	0x01 : 0x03D	No	Yes	Yes	Yes	
GRSPW	SpaceWire link with RMAP and AHB interface	0x01 : 0x01F	No	No	Yes	Yes	1), 2)
GRSPW2	SpaceWire link with RMAP and AHB interface	0x01 : 0x029	No	No	Yes	Yes	1), 2)
I2C2AHB	I2C (slave) to AHB bridge	0x01 : 0x00B	Yes	Yes	Yes	Yes	
I2CMST	I2C Master with APB interface	0x01 : 0x028	Yes	Yes	Yes	Yes	
I2CSLV	I2C Slave with APB interface	0x01 : 0x03E	Yes	Yes	Yes	Yes	
SPI2AHB	SPI (slave) to AHB bridge	0x01 : 0x05C	Yes	Yes	Yes	Yes	
SPICTRL	SPI Controller with APB interface	0x01 : 0x02D	Yes	Yes	Yes	Yes	
TAP	JTAG TAP controller	-	No	Yes	Yes	Yes	

1) Available as separate package or as addition to existing releases.
2) Only available as netlist or encrypted RTL
3) Always included with LEON4 license
4) Requires PHY for selected target technology. Please see IP core documentation for supported technologies

Table 7. Ethernet interface

Name	Function	Vendor:Device	GPL	COM	FT	FT-FPGA	Note
GRETH	Aeroflex Gaisler 10/100 Mbit Ethernet MAC with AHB I/F	0x01 : 0x01D	Yes	Yes	Yes	Yes	
GRETH_GBIT	Aeroflex Gaisler 10/100/1000 Mbit Ethernet MAC with AHB	0x01 : 0x01D	No	Yes	Yes	Yes	

Table 8. USB interface

Name	Function	Vendor:Device	GPL	COM	FT	FT-FPGA	Note
GRUSBHC	USB-2.0 Host controller (UHCI/EHCI) with AHB I/F	0x01 : 0x027	No	No	No	No	1)
GRUSBDC / GRUSB_DCL	USB-2.0 device controller / AHB debug communication link	0x01 : 0x022	No	No	No	No	1)

1) Available as separate package or as addition to existing releases.
2) Only available as netlist or encrypted RTL
3) Always included with LEON4 license
4) Requires PHY for selected target technology. Please see IP core documentation for supported technologies.

Table 9. MIL-STD-1553 Bus interface

Name	Function	Device ID	GPL	COM	FT	FT-FPGA	Note
GR1553B	Advanced MIL-ST-1553B / AS15551 Interface	0x01 : 0x04D	No	No	No	No	1)
B1553BC	AHB interface for Actel B1553BC	0x01 : 0x070	No	No	Yes	Yes	
B1553RT	AHB interface for Actel B1553RT	0x01 : 0x071	No	No	Yes	Yes	
B1553BRM	AHB interface for Actel B1553BRM	0x01 : 0x072	No	No	Yes	Yes	

1) Available as separate package or as addition to existing releases.
2) Only available as netlist or encrypted RTL
3) Always included with LEON4 license
4) Requires PHY for selected target technology. Please see IP core documentation for supported technologies.

Table 10. Encryption

Name	Function	Vendor:Device	GPL	COM	FT	FT-FPGA	Note
GRAES	128-bit AES Encryption/Decryption Core	0x01 : 0x073	No	No	No	No	1)
GRAES_DMA	Advanced Encryption Standard with DMA	0x01 : 0x07B	No	No	No	No	1)
GRECC	Elliptic Curve Cryptography Core	0x01 : 0x074	No	No	No	No	1)

1) Available as separate package or as addition to existing releases.
2) Only available as netlist or encrypted RTL
3) Always included with LEON4 license
4) Requires PHY for selected target technology. Please see IP core documentation for supported technologies.

Table 11. Simulation and debugging

Name	Function	Vendor:Device	GPL	COM	FT	FT-FPGA	Note
SRAM	SRAM simulation model with srecord pre-load	-	Yes	Yes	Yes	Yes	
MT48LC16M16	Micron SDRAM model with srecord pre-load	-	Yes	Yes	Yes	Yes	
MT46V16M16	Micron DDR model	-	Yes	Yes	Yes	Yes	
CY7C1354B	Cypress ZBT SSRAM model with srecord pre-load	-	Yes	Yes	Yes	Yes	
AHBMSTEM	AHB master simulation model with scripting (deprecated)	0x01 : 0x040	Yes	Yes	Yes	Yes	
AHBSLVEM	AHB slave simulation model with scripting (deprecated)	0x01 : 0x041	Yes	Yes	Yes	Yes	
AMBAMON	AHB and APB protocol monitor	-	No	Yes	Yes	Yes	
ATF	AMBA test framework consisting of master, slave and arbiter.	0x01 : 0x068 - 0x06A	No	Yes	Yes	Yes	
LOGAN	On-chip Logic Analyzer	0x01 : 0x062	No	Yes	Yes	Yes	

Table 12. Graphics functions

Name	Function	Vendor:Device	GPL	COM	FT	FT-FPGA	Note
APBVGA	VGA controller with APB interface	0x01 : 0x061	Yes	Yes	Yes	Yes	
SVGACTRL	VGA controller core with DMA	0x01 : 0x063	Yes	Yes	Yes	Yes	

Table 13. Auxiliary functions

Name	Function	Vendor:Device	GPL	COM	FT	FT-FPGA	Note
GRACECTRL	AMBA SystemACE interface controller	0x01 : 0x067	Yes	Yes	Yes	Yes	
GRADCDAC	Combined ADC / DAC Interface	0x01 : 0x036	No	Yes	Yes	Yes	
GRFIFO	External FIFO Interface with DMA	0x01 : 0x035	No	Yes	Yes	Yes	
GRGPIO	General purpose I/O port	0x01 : 0x01A	Yes	Yes	Yes	Yes	
GRGPREG	General purpose Register	0x01 : 0x087	Yes	Yes	Yes	Yes	
GRPULSE	General purpose I/O with pulses	0x01 : 0x037	No	Yes	Yes	Yes	
GRPWM	PWM generator	0x01 : 0x04A	No	Yes	Yes	Yes	
GRSYSMON	AMBA Wrapper for Xilinx System Monitor	0x01 : 0x066	Yes	Yes	Yes	Yes	
GRVERSION	Version and revision register	0x01 : 0x03A	No	Yes	Yes	Yes	

Table 14. Error detection and correction functions

Name	Function	GPL	COM	FT	FT-FPGA	Note
RS(24, 16, 8, E=1)	16 bit data, 8 check bits, corrects 4-bit error in 1 nibble	No	No	Yes	Yes	
RS(40, 32, 8, E=1)	32 bit data, 8 check bits, corrects 4-bit error in 1 nibble	No	No	Yes	Yes	
RS(48, 32, 16, E=1+1)	32 bit data, 16 check bits, corrects 4-bit error in 2 nibbles	No	No	Yes	Yes	
RS(48, 32, 16, E=2)	32 bit data, 16 check bits, corrects 4-bit error in 2 nibbles	No	No	Yes	Yes	
GR(2^4)(68, 60, 8, T=1)	QEC/QED error correction code encoder/decoder	No	No	Yes	Yes	

1.3 Supported technologies

Technology support and instructions for extending GRLIB with support for additional technologies is documented in the ‘GRLIB User’s Manual’. The table below shows the technology maps available from Aeroflex Gaisler for GRLIB and in which GRLIB distributions these techology maps are included.

Vendor	**Technology**	**GPL**	**COM**	**FT**	**FT-FPGA**	**Comment**
Actel	ProASIC3, ProASIC3e, ProASIC3l, Axcelerator, Axcelerator DSP, Fusion	No	Yes	Yes	Yes	
Altera	Cyclone2 - 4, Statix - Stratix3	Yes	Yes	No	No	
Lattice	-	Yes	Yes	No	No	
Xilinx	Unisim (Virtex2 - Virtex7)	Yes	Yes	Yes	Yes	
Other ASIC	-	No	-	-	No	Contact Aeroflex Gaisler for details. See also *GRLIB IP Library User’s Manual*.

1.4 Implementation characteristics

The table below shows the approximate area for some of the GRLIP IP blocks mapped on Virtex2, Actel-AX and typical ASIC technologies. The area depends strongly on configuration options (generics), optimization constraints and used synthesis tools. The data in the table should therefore be seen as an indication only. The tools used to obtain the area was Synplify-8.1 for FPGA and Synopsys DC for ASIC. The LUT area for Altera Stratix devices is roughly the same as for Virtex2. Using XST instead of Synplify for Xilinx FPGAs gives typically 15% larger area.

Table 15. Approximate area consumption for some standard GRLIB IP cores

	Virtex2		**AX/RTAX**		**ASIC**
Block	**LUT**	**RAM16**	**Cells**	**RAM64**	**Gates**
AHBCTRL	200		500		1,000
AHBJTAG	120		350		1,000
AHBUART (DSU UART)	450		800		2,000
APBCTRL	150		200		800
APBPS2	450		800		2,000
APBUART	200		300		1,000
APBVGA	250	4	-		1,400
CAN_OC (CAN-2.0 core with AHB I/F)	1,600	2	2,800	2	8,000
GRCAN (CAN 2.0 Controller with DMA)	2,300		4,800		20,000
DDRCTRL	1,600	2	-		10,000
DDRSPA (32-bit)	900	2	-		-
DIV32 (64/32-bit iterative divider)	400		500		2,000

Table 15. Approximate area consumption for some standard GRLIB IP cores

	Virtex2		**AX/RTAX**		**ASIC**
Block	**LUT**	**RAM16**	**Cells**	**RAM64**	**Gates**
GPTIMER (16-bit scaler + 2x32-bit timers)	250		400		1,300
GRETH 10/100 Mbit Ethernet MAC	1,500		2,500	2	8,000
GRETH 10/100 Mbit Ethernet MAC with EDCL	2,600	1	4,000	4	15,000
GRFPU-Lite including LEON3 controller	4,000	6	7,000	4	35,000
GRFPU IEEE-754 floating-point unit	8,500	2	-		100,000
GRFPC for LEON3	5,000	4	-		25,000
GRGPIO, 16-bit configuration	100		150		800
GRSPW Spacewire link	1,900	3	2,800	3	15,000
GRSPW Spacewire link with RMAP	3,000	4	4,500	4	25,000
GRTC CCSDS telecommad decoder front-end	2,000		3,000		15,000
GRTM CCSDS telemetry Generator	4,500	2	6,000	4	30,000
I2CMST I2C Master	200		300		1,500
I2CSLV I2C Slave	150		250		1,000
IRQMP (1 processor)	300		350		1,500
LEON3, 8 + 8 Kbyte cache	4,300	12	6,500	40	20,000
LEON3, 8 + 8 Kbyte cache + DSU3	5,000	12	7,500	40	25,000
LOGAN, 32 channels, 1024 traces, 1 trigger	300	2	-		-
MCTRL	350		1,000		1,500
MCTRL including SDRAM support	600		1,400		2,000
MUL32 (32x32 multiplier, 4-cycle iterative)	200		1,400		5,500
PCI_TARGET, simple PCI target	150		500		800
PCI_MTF, master/target PCI with FIFO	1,100	4	2,000	4	6,000
PCIDMA, master/target PCI with FIFO/DMA	1,800	4	3,000	4	9,000
PCITRACE	300	2	600	4	1,400
SRCTRL	100		200		500
SDCTRL	300		600		1,200
SPICTRL	450		900		2,500
SPIMCTRL	300		600		1,200
SVGACTRL	1,200	2	1,600	2	8,000
USBDCL	2,000		-		12,000

Table 16. Approximate area consumption for some FT GRLIB IP cores

Block	**RTAX2000 (Cells)**	**ASIC (gates)**
GRFPU-Lite-FT including LEON3 controller	7,100 + 4 RAM64K36	36,000
GRFPCFT for LEON3	-	30,000 + RAM
LEON3FT, 8 + 4 Kbyte cache	7,500 + 40 RAM64K36	22,000 + RAM
LEON3FT, 8 + 4 Kbyte cache + DSU3	8,500 + 44 RAM64K36	27,000 + RAM
LEON3FT, 8 + 4 Kbyte cache with FPU + DSU3	16,000 + 48 RAM64K36	60,000 + RAM
FTSRCTRL	700	2,500
FTSRCTRL8	750	-
FTSDCTRL	1,000	3,500
FTAHBRAM (2 Kbyte with EDAC)	300 + 5 RAM64K36	2,000 + RAM

The table below show the area resources for some common FPGA devices. It can be used to quickly estimate if a certain GRLIB design will fit the target device.

Table 17. Area resources for some common FPGA devices

FPGA	**Logic**	**Memory**
Actel AX1000	18,144 Cells	32 RAM64K36
Actel AX2000	32,248 Cells	64 RAM64K36
Xilinx Spartan3-1500	33,248 LUT	64 RAMB16
Xilinx Virtex2-3000	28,672 LUT	96 RAMB16
Xilinx Virtex2-6000	67,584 LUT	144 RAMB16

2 AHB2AHB - Uni-directional AHB/AHB bridge

2.1 Overview

The uni-directional AHB/AHB bridge is used to connect two AMBA AHB buses clocked by synchronous clocks with any frequency ratio. The bridge is connected through a pair consisting of an AHB slave and an AHB master interface. AHB transfer forwarding is performed in one direction, where AHB transfers to the slave interface are forwarded to the master interface. Applications of the uni-directional bridge include system partitioning, clock domain partitioning and system expansion.

Features offered by the uni-directional AHB to AHB bridge are:

- Single and burst AHB transfers
- Data buffering in internal FIFOs
- Efficient bus utilization through (optional) use of SPLIT response and data prefetching
- Posted writes
- Read and write combining, improves bus utilization and allows connecting cores with differing AMBA access size restrictions.
- Deadlock detection logic enables use of two uni-directional bridges to build a bi-directional bridge (one example is the bi-directional AHB/AHB bridge core (AHBBRIDGE))

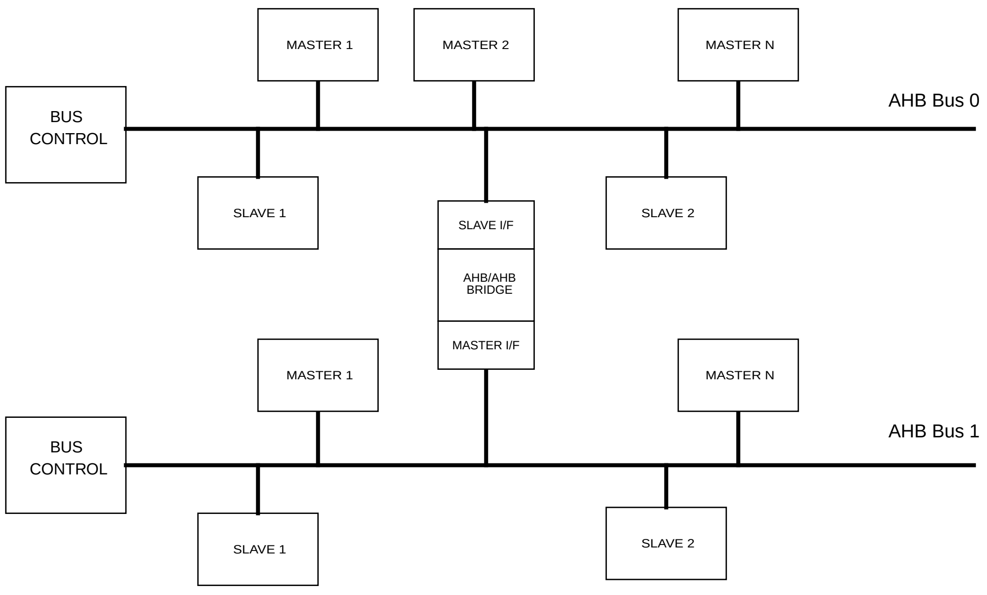


Figure 1. Two AHB buses connected with (uni-directional) AHB/AHB bridge

2.2 Operation

2.2.1 General

The address space occupied by the AHB/AHB bridge on the slave bus is configurable and determined by Bank Address Registers in the slave interface’s AHB Plug&Play configuration record.

The bridge is capable of handling single and burst transfers of all burst types. Supported transfer sizes (HSIZE) are BYTE, HALF-WORD, WORD, DWORD, 4WORD and 8WORD.

For AHB write transfers write data is always buffered in an internal FIFO implementing posted writes. For AHB read transfers the bridge uses GRLIB’s AMBA Plug&Play information to determine whether the read data will be prefetched and buffered in an internal FIFO. If the target address for an AHB read burst transfer is a prefetchable location the read data will be prefetched and buffered.

The bridge can be implemented to use SPLIT responses or to insert wait states when handling an access. With SPLIT responses enabled, an AHB master initiating a read transfer to the bridge is always splitted on the first transfer attempt to allow other masters to use the slave bus while the bridge performs read transfer on the master bus.The descriptions of operation in the sections below assume that the bridge has been implemented with support for AMBA SPLIT responses. The effects of disabling support for AMBA SPLIT responses are described in section 2.2.11.

If interrupt forwarding is enabled the interrupts on the slave bus interrupt lines will be forwarded to the master bus and vice versa.

2.2.2 AHB read transfers

When a read transfer is registered on the slave interface the bridge gives a SPLIT response. The master that initiated the transfer will be de-granted allowing other bus masters to use the slave bus while the bridge performs a read transfer on the master side. The master interface then requests the bus and starts the read transfer on the master side. Single transfers on the slave side are normally translated to single transfers with the same AHB address and control signals on the master side, however read combining can translate one access into several smaller accesses. Translation of burst transfers from the slave to the master side depends on the burst type, burst length, access size and the AHB/AHB bridge configuration.

If the read FIFO is enabled and the transfer is a burst transfer to a prefetchable location, the master interface will prefetch data in the internal read FIFO. If the splitted burst on the slave side was an incremental burst of unspecified length (INCR), the length of the burst is unknown. In this case the master interface performs an incremental burst up to a specified address boundary (determined by the VHDL generic *rburst).* The bridge can be configured to recognize an INCR read burst marked as instruction fetch (indicated on HPROT signal). In this case the prefetching on the master side is completed at the end of a cache line (the cache line size is configurable through the VHDL generic *iburst*). When the burst transfer is completed on the master side, the splitted master that initiated the transfer (on the slave side) is allowed in bus arbitration by asserting the appropriate HSPLIT signal to the AHB controller. The splitted master re-attempts the transfer and the bridge will return data with zero wait states.

If the read FIFO is disabled, or the burst is to non-prefetchable area, the burst transfer on the master side is performed using sequence of NONSEQ, BUSY and SEQ transfers. The first access in the burst on the master side is of NONSEQ type. Since the master interface can not decide whether the splitted burst will continue on the slave side or not, the master bus is held by performing BUSY transfers. On the slave side the splitted master that initiated the transfer is allowed in bus arbitration by asserting the HSPLIT signal to the AHB controller. The first access in the transfer is completed by returning read data. The next access in the transfer on the slave side is extended by asserting HREADY low. On the master side the next access is started by performing a SEQ transfer (and then holding the bus using BUSY transfers). This sequence is repeated until the transfer is ended on the slave side.

In case of an ERROR response on the master side the ERROR response will be given for the same access (address) on the slave side. SPLIT and RETRY responses on the master side are re-attempted until an OKAY or ERROR response is received.

2.2.3 AHB write transfers

The AHB/AHB bridge implements posted writes. During the AHB write transfer on the slave side the data is buffered in the internal write FIFO and the transfer is completed on the slave side by always giving an OKAY response. The master interface requests the bus and performs the write transfer when the master bus is granted. If the burst transfer crosses the write burst boundary (defined by VHDL generic *wburst*), a SPLIT response is given. When the bridge has written the contents of the

FIFO out on the master side, the bridge will allow the master on the slave side to perform the remaining accesses of the write burst transfer.

Writes are accepted with zero wait states if the bridge is idle and the incoming access is not locked. If the incoming access is locked, each access will have one wait state. If write combining is disabled a non-locked BUSY cycle will lead to a flush of the write FIFO. If write combining is enabled or if the incoming access is locked, the bridge will not flush the write FIFO during the BUSY cycle.

2.2.4 Deadlock conditions

When two bridges are used to form a bi-drectional bridge, a deadlock situation can occur if the bridges are simultaneously accessed from both buses. The bridge that has been configured as a slave contains deadlock detection logic which will resolve a deadlock condition by giving a RETRY response, or by issuing SPLIT complete followed by a new SPLIT response. When the core resolves a deadlock while prefetching data, any data in the prefetch buffer will be dropped when the core's slave interface issues the AMBA RETRY response. When the access is retried it may lead to the same memory locations being read twice.

Deadlock detection logic for bi-directional configurations may lead to deadlocks in other parts of the system. Consider the case where a processor on bus A on one side of the bidirectional bridge needs to perform an instruction fetch over the bridge before it can release a semaphore located in memory on bus A. Another processor on bus B, on the other side of the bridge, may spin on the semaphore wating for its release. In this scenario, the accesses from the processor on bus B could, depending on system configuration, continuously trigger a deadlock condition where the core will drop data in, or be prevented from initiating, the instruction fetch for the processor on bus A. Due to scenarios of this kind the bridge should not be used in bi-directional configurations where dependencies as the one described above exist between the buses connected by the bridge.

Other deadlock conditions exist with locked transfers, see section 2.2.5.

2.2.5 Locked transfers

The AHB/AHB bridge supports locked transfers. The master bus will be locked when the bus is granted and remain locked until the transfer completes on the slave side. Locked transfers can lead to deadlock conditions, the core's VHDL generic *lckdac* determines if and how the deadlock conditions are resolved.

With the VHDL generic *lckdac* set to 0, locked transfers may *not* be made after another read access which received SPLIT until the first read access has received split complete. This is because the bridge will return split complete for the first access first and wait for the first master to return. This will cause deadlock since the arbiter is not allowed to change master until a locked transfer has been completed. The AMBA specification requires that the locked transfer is handled before the previous transfer, which received a SPLIT response, is completed.

With *lckdac* set to 1, the core will respond with an AMBA ERROR response to locked access that is made while an ongoing read access has received a SPLIT response. With *lckdac* set to 2 the bridge will save state for the read access that received a SPLIT response, allow the locked access to complete, and then complete the first access. All non-locked accesses from other masters will receive SPLIT responses until the saved data has been read out.

If the core is used to create a bi-directional bridge there is one more deadlock condition that may arise when locked accesses are made simultaneously in both directions. If the VHDL generic *lckdac* is set to 0 the core will deadlock. If *lckdac* is set to a non-zero value the slave bridge will resolve the deadlock condition by issuing an AMBA ERROR response to the incoming locked access.

2.2.6 Read and write combining

Read and write combining allows the bridge to assemble or split AMBA accesses on the bridge's slave interface into one or several accesses on the master interface. This functionality can improve bus utilization and also allows cores that have differing AMBA access size restrictions to communicate with each other. The functionality attained by read and write combining depends on the VHDL generics *rdcomb* (defines type of read combining), *wrcomb* (defines type of write combining), *slvmstaccsz* (defines maximum AHB access size supported by the bridge's slave interface) and *mstmaccsz* (defines maximum AHB access size that can be used by bridge's master interface). These VHDL generics are described in section 2.6. The table below shows the effect of different settings. BYTE and HALF-WORD accesses are special cases. The table does not list illegal combinations, for instance *mstmaccsz* /= *slvmaccsz* requires that *wrcomb* /= 0 and *rdcomb* /= 0.

Table 18. Read and write combining

Access on slave interface	**Access size**	**wrcomb**	**rdcomb**	**Resulting access(es) on master interface**
BYTE or HALF-WORD single read access to any area	-	-	-	Single access of same size
BYTE or HALF-WORD read burst to prefetchable area	-	-	-	Incremental read burst of same access size as on slave interface, the length is the same as the number of 32-bit words in the read buffer, but will not cross the read burst boundary.
BYTE or HALF-WORD read burst to non-prefetchable area	-	-	-	Incremental read burst of same access size as on slave interface, the length is the same as the length of the incoming burst. The master interface will insert BUSY cycles between the sequential accesses.
BYTE or HALF-WORD single write	-	-	-	Single access of same size
BYTE or HALF-WORD write burst	-	-	-	Incremental write burst of same size and length, the maximum length is the number of 32-bit words in the write FIFO.
Single read access to any area	Access size <= mstmaccsz	-	-	Single access of same size
Single read access to any area	Access size > mstmaccsz	-	1	Sequence of single accesses of mstmaccsz. Number of accesses: (access size)/mstmaccsz
Single read access to any area	Access size > mstmaccsz	-	2	Burst of accesses of size mstmaccsz. Length of burst: (access size)/mstmaccsz
Read burst to prefetchable area	-	-	0	Burst of accesses of incoming access size up to address boundary defined by rburst.
Read burst to prefetchable area	-	-	1 or 2	Burst of accesses of size mstmaccsz up to address boundary defined by rburst.
Read burst to non-prefetchable area	Access size <= mstmaccsz	-	-	Incremental read burst of same access size as on slave interface, the length is the same as the length of the incoming burst. The master interface will insert BUSY cycles between the sequential accesses.
Read burst to non-prefetchable area	Access size > mstmaccsz	-	1 or 2	Burst of accesses of size mstmaccsz. Length of burst: (incoming burst length)*(access size)/mstmaccsz
Single write	Access size <= mstmaccsz	-	-	Single write access of same size
Single write	Access size > mstmaccsz	1	-	Sequence of single access of mstmaccsz. Number of accesses: (access size)/mstmaccsz.
Single write	Access size > mstmaccsz	2	-	Burst of accesses of mstmaccsz. Length of burst: (access size)/mstmaccsz.
Write burst	-	0	-	Burst of same size as incoming burst, up to address boundary defined by VHDL generic wburst.

Table 18. Read and write combining

Access on slave interface	Access size	wrcomb	rdcomb	Resulting access(es) on master interface
Write burst	-	1 or 2	-	Burst write of maximum possible size. The bridge will use the maximum size (up to mst-maccsz) that it can use to empty the writebuffer.

Read and write combining prevents the bridge from propagating fixed length bursts and wrapping bursts. See section 2.2.7 for a discussion on burst operation.

Read and write combining with VHDL generics *wrcomb/rdcomb* set to 1 cause the bridge to use single accesses when divding an incoming access into several smaller accesses. This means that another master on the bus may write or read parts of the memory area to be accessed by the bridge before the bridge has read or written all the data. In bi-directional configurations, an incoming access on the master bridge may cause a collision that aborts the operation on the slave bridge. This may cause the bridge to read the same memory locations twice. This is normally not a problem when accessing memory areas. The same issues apply when using an AHB arbiter that performs early burst termination. The standard GRLIB AHBCTRL core does not perform early burst termination.

To ensure that the bridge does not re-read an address, and that all data in an access from the bridge's slave interface is propagated out on the master interface without interruption the VHDL generics *rdcomb* and *wrcomb* should both be set to 0 or 2. In addition to this, the AHB arbiter may not perform early burst termination (early burst termination is not performed by the GRLIB AHBCTRL arbiter).

Read and write combining can be limited to specified address ranges. See description of the *combmask* VHDL generic for more information. Note that if the core is implemented with support for prefetch and read combining, it will not obey combmask for prefetch operations (burst read to prefetchable areas). Prefetch operations will always be performed with the maximum allowed size on the master interface.

2.2.7 Burst operation

The core can be configured to support all AMBA 2.0 burst types (single access, incrementing burst of unspecified length, fixed length incrementing bursts and wrapping bursts). Single accesses and incrementing bursts of unspecified length have previously been discussed in this document. An incoming single access will lead to one access, or multiple accesses for some cases with read/write combining, on the other side of the bridge. An incoming incrementing burst of unspecified length to a prefetchable area will lead to the prefetch buffer (if available) being filled using the same access size, or the maximum allowed access size if read/write combining is enabled, on the master interface.

If the core is used in a system where no fixed length bursts or incremental bursts will be used in accesses to the bridge, then set the *allbrst* generic to 0 and skip the remainder of this section.

The VHDL generic *allbrst* controls if the core will support fixed length and wrapping burst accesses. If *allbrst* is set to 0, the core will treat all burst accesses as incrementing of unspecified length. For fixed length and wrapping bursts this can lead to performance penalties and malfunctions. Support for fixed length and wrapping bursts is enabled by setting *allbrst* to 1 or 2. Table 19 describes how the core will handle different burst types depending on the setting of *allbrst*.

Table 19. Burst handling

Value of allbrst generic	**Access type***	**Undefined length incrementing burst INCR**	**Fixed length incrementing burst INCR{4,8,16}**	**Wrapping burst WRAP{4,8,16}**
0	Reads to non-prefetchable area	Incrementing burst with BUSY cycles inserted. Same behaviour with read and write combining.	Fixed length burst with BUSY cycles inserted. If the burst is short then the burst may end with a BUSY cycle. If access combining is used the HBURST signal will get incorrect values.	Malfunction. Not supported
	Reads to prefetchable area	Incrementing burst of maximum allowed size, filling prefetch buffer, starting at address boundary defined by prefetch buffer.		Malfunction. Not supported
	Write burst	Incrementing burst	Incrementing burst, if write combining is enabled, and triggered, the burst will be translated to an incrementing burst of undefined length. VHDL generic *wrcomb* should not be set to 1 (but to 0 or 2) in this case	Write combining is not supported. Same access size will be used on both sides of the bridge.
1	Reads to non-prefetchable area	Incrementing burst with BUSY cycles inserted. Same behaviour with read and write combining.	Same burst type with BUSY cycles inserted. If read combining is enabled, and triggered by the incoming access size, an incremental burst of unspecified length will be used. If the burst is short then the burst may end with a BUSY cycle.	Same burst type with BUSY cycles inserted. If read combining is enabled, and triggered by the incoming access size, an incremental burst of unspecified length will be used. This will cause AMBA violations if the wrapping burst does not start from offset 0.
	Reads to prefetchable area	Incrementing burst of maximum allowed size, filling prefetch buffer.	For reads, the core will perform full (or part that fits in prefetch buffer) fixed/wrapping burst on master interface and then respond with data. No BUSY cycles are inserted. If the access made to the slave interface is larger than the maximum supported access size on the master interface then a incrementing burst of unspecified length will be used to fill the prefetch buffer. This (read combining) is not supported for wrapping bursts.	
	Write burst	Same as for allbrst = 0		
2	Reads to non-prefetchable area	Incrementing burst with BUSY cycles inserted. Same behaviour with read and write combining.	Reads are treated as a prefetchable burst. See below.	
	Reads to prefetchable area	Incrementing burst of maximum allowed size, filling prefetch buffer, starting at address boundary defined by prefetch buffer.	Core will perform full (or part that fits in prefetch buffer) fixed/wrapping burst on master interface and then respond with data. No BUSY cycles are inserted. If the access made to the slave interface is larger than the maximum supported access size on the master interface then a incrementing burst of unspecified length will be used to fill the prefetch buffer. This (read combining) is not supported for wrapping bursts.	
	Write burst	Same as for allbrst = 0		

* Access to prefetchable area where the core's prefetch buffer is ised (VHDL generic pfen /= 0).

2.2.8 Transaction ordering, starvation and AMBA arbitration schemes

The bridge is configured at implementation to use one of two available schemes to handle incoming accesses. The bridge will issue SPLIT responses when it is busy and on incoming read accesses. If the bridge has been configured to use first-come, first-served ordering it will keep track of the order of incoming accesses and serve the requests in the same order. If first-come, first-served ordering is disabled the bridge will give some advantage to the master it has a response for and then allow all masters in to arbitration simultaneously, moving the decision on which master that should be allowed to access the bridge to the bus arbitration.

When designing a system containing a bridge the expected traffic patterns should be analyzed. The designer must be aware how SPLIT responses affect arbitration and how the selected transaction ordering in the bridge will affect the system. The two different schemes are further described in sections 2.2.9 and 2.2.10.

2.2.9 First-come, first-served ordering

First-come, first served ordering is used when the VHDL generic *fcfs* is non-zero.

With first-come, first-served ordering the bridge will keep track of the order of incoming accesses. The accesses will then be served in the same order. For instance, if master 0 initiates an access to the bridge, followed by master 3 and then master 5, the bridge will propagate the access from master 0 (and respond with SPLIT on a read access) and then respond with SPLIT to the other masters. When the bridge has a response for master 0, this master will be allowed in arbitration again by the bridge asserting HSPLIT. When the bridge has finished serving master 0 it will allow the next queued master in arbitration, in this case master 3. Other incoming masters will receive SPLIT responses and will not be allowed in arbitration until all previous masters have been served.

An incoming locked access will always be given precedence over any other masters in the queue.

A burst that has initiated a pre-fetch operation will receive SPLIT and be inserted last in the master queue if the burst is longer than the maximum burst length that the bridge has been configured for.

It should be noted that first-come, first-served ordering may not work well in systems where an AHB master needs to have higher priority compared to the other masters. The bridge will not prioritize any master, except for masters performing locked accesses.

2.2.10 Bus arbiter ordering

Bus arbiter ordering is used when VHDL generic *fcfs* is set to zero.

When several masters have received SPLIT and the bridge has a response for one of these masters, the master with the queued response will be allowed in to bus arbitration by the bridge asserting the corresponding HSPLIT signal. In the following clock cycle, all other masters that have received SPLIT responses will also be allowed in bus arbitration as the bridge asserts their HSPLIT signals simultaneously. By doing this the bridge defers the decision on the master to be granted next to the AHB arbiter. The bridge does not show any preference based on the order in which it issued SPLIT responses to masters, except to the master that initially started a read or write operation. Care has been taken so that the bridge shows a consistent behavior when issuing SPLIT responses. For instance, the bridge could be simplified if it could issue a SPLIT response just to be able to change state, and not initiate a new operation, to an access coming after an access that read out prefetched data. When the bridge entered its idle state it could then allow all masters in bus arbitration and resume normal operation. That solution could lead to starvation issues such as:

T0: Master 1 and Master 2 have received SPLIT responses, the bridge is prefetching data for Master 1

T1: Master 1 is allowed in bus arbitration by setting the corresponding HSPLIT

T2: Master 1 reads out prefetch data, Master 2 HSPLIT is asserted to let Master 2 in to bus arbitration

T3: Master 2 performs an access, receives SPLIT, however the bridge does not initiate an access, it just stalls in order to enter its idle state.

T4: Master 2 is allowed in to bus arbitration, Master 1 initiates an access that leads to a prefetch and Master 1 receives a SPLIT response

T5: Master 2 performs an access, receives SPLIT since the bridge is prefetching data for master 1

T6: Go back to T0

This pattern will repeat until Master 1 backs away from the bus and Master 2 is able to make an access that starts an operation over the bridge. In most systems it is unlikely that this behavior would introduce a bus lock. However, the case above could lead to an unexpectedly long time for Master 2 to complete its access. Please note that the example above is illustrative and the problem does not exist in the core as the core does not issue SPLIT responses to (non-locked) accesses in order to just change state but a similar pattern could appear as a result of decisions taken by the AHB arbiter if Master 1 is given higher priority than Master 2.

In the case of write operations the scenario is slightly different. The bridge will accept a write immediately and will not issue a SPLIT response. While the bridge is busy performing the write on the master side it will issue SPLIT responses to all incoming accesses. When the bridge has completed the write operation on the master side it will continue to issue SPLIT responses to any incoming access until there is a cycle where the bridge does not receive an access. In this cycle the bridge will assert HSPLIT for all masters that have received a SPLIT response and return to its idle state. The first master to access the bridge in the idle state will be able to start a new operation. This can lead to the following behavior:

T0: Master 1 performs a write operation, does NOT receive a SPLIT response

T1: Master 2 accesses the bridge and receives a SPLIT response

T2: The bridge now switches state to idle since the write completed and asserts HSPLIT for Master 2.

T3: Master 1 is before Master 2 in the arbitration order and we are back at T0.

In order to avoid this last pattern the bridge would have to keep track of the order in which it has issued SPLIT responses and then assert HSPLIT in the same order. This is done with first-come, first-served ordering described in section 2.2.9.

2.2.11 AMBA SPLIT support

Support for AMBA SPLIT responses is enabled/disabled through the VHDL generic *split*. SPLIT support should be enabled in most systems. The benefits of using SPLIT responses is that the bus on the bridge's slave interface side can be free while the bridge is performing an operation on the master side. This will allow other masters to access the bus and generally improve system performance. The use of SPLIT responses also allows First-come, first-served transaction ordering.

For configurations where the bridge is the only slave interface on a bus, it can be beneficial to implement the bridge without support for AMBA SPLIT responses. Removing support for SPLIT responses reduces the area used by the bridge and may also reduce the time required to perform accesses that traverse the bridge. It should be noted that building a bi-directional bridge without support for SPLIT responses will increase the risk of access collisions.

If SPLIT support is disabled the bridge will insert wait states where it would otherwise issue a SPLIT response to a master initiating an access. This means that the arbitration ordering will be left to the bus arbiter and the bridge cannot be implemented with the First-come, first-served transaction ordering scheme. The bridge will still issue RETRY responses to resolve dead lock conditions, to split up long burst and also when the bridge is busy emptying it's write buffer on the master side.

2.2.12 Core latency

The delay incurred when performing an access over the core depends on several parameters such as core configuration, the operating frequency of the AMBA buses, AMBA bus widths and memory access patterns. Table 20 below shows core behavior in a system where both AMBA buses are running at the same frequency and the core has been configured to use AMBA SPLIT responses. Table 21 further down shows core behavior in the same system without support for SPLIT responses.

Table 20. Example of single read with FFACT = 1, and SPLIT support

Clock cycle	Core slave side activity	Core master side activity
0	Discovers access and transitions from idle state	Idle
1	Slave side waits for master side, SPLIT response is given to incoming access, any new incoming accesses also receive SPLIT responses.	Discovers slave side transition. Master interface output signals are assigned.
2		If bus access is granted, perform address phase. Otherwise wait for bus grant.
3		Register read data and transition to data ready state.
4	Discovers that read data is ready, assign read data output and assign SPLIT complete	Idle
5	SPLIT complete output is HIGH	
6	Typically a wait cycle for the SPLIT:ed master to be allowed into arbitration. Core waits for master to return. Other masters receive SPLIT responses.	
7	Master has been allowed into arbitration and performs address phase. Core keeps HREADY high	
8	Access data phase. Core has returned to idle state.	

Table 21. Example of single read with FFACT = 1, without SPLIT support

Clock cycle	Core slave side activity	Core master side activity
0	Discovers access and transitions from idle state	Idle
1	Slave side waits for master side, wait states are inserted on the AMBA bus.	Discovers slave side transition. Master interface output signals are assigned.
2		Bus access is granted, perform address phase.
3		Register read data and transition to data ready state.
4	Discovers that read data is ready, assign HREADY output register and data output register.	Idle
5	HREADY is driven on AMBA bus. Core has returned to idle state	

While the transitions shown in tables 20 and 21 are simplified they give an accurate view of the core delay. If the master interface needs to wait for a bus grant or if the read operation receives wait states, these cycles must be added to to the cycle count in the tables. The behavior of the core with a frequency factor of two between the buses is shown in tables 22 and 23 (best case, delay may be larger depending on on which slave clock cycle an access is made to the core).

Table 22. Example of single read with FFACT = 2, Master freq. > Slave freq, without SPLIT support

Slave side clock cycle	**Core slave side activity**	**Master side clock cycle**	**Core master side activity**
0	Discovers access and transitions from idle state	0	Discovers slave side transition. Master interface output signals are assigned.
1	Slave side waits for master side, wait states are inserted on the AMBA bus.		
2		1	Bus access is granted, perform address phase.
3			
4		2	Register read data and transition to data ready state.
5			
6	Discovers that read data is ready, assign HREADY output register and data output register.	3	Idle
7	HREADY is driven on AMBA bus. Core has returned to idle state		

Table 23. Example of single read with FFACT = 2, Master freq. > Slave freq, without SPLIT support

Slave side clock cycle	**Core slave side activity**	**Master side clock cycle**	**Core master side activity**
0	Discovers access and transitions from idle state	0	Idle
		1	
1	Slave side waits for master side, wait states are inserted on the AMBA bus.	2	Discovers slave side transition. Master interface output signals are assigned.
		3	Bus access is granted, perform address phase.
2	Discovers that read data is ready, assign HREADY output register and data output register.	4	Register read data and transition to data ready state.
		5	Idle
3	HREADY is driven on AMBA bus. Core has returned to idle state	6	
		7	

Table 24 below lists the delays incurred for single operations that traverse the bridge while the bridge is in its idle state. The second column shows the number of cycles it takes the master side to perform the requested access, this column assumes that the master slave gets access to the bus immediately and that each access is completed with zero wait states. The table only includes the delay incurred by traversing the core. For instance, when the access initiating master reads the core's prefetch buffer, each additional read will consume one clock cycle. However, this delay would also have been present if the master accessed any other slave.

Write accesses are accepted with zero wait states if the bridge is idle, this means that performing a write to the idle core does not incur any extra latency. However, the core must complete the write operation on the master side before it can handle a new access on the slave side. If the core has not transitioned into its idle state, pending the completion of an earlier access, the delay suffered by an access be longer than what is shown in the tables in this section. Accesses may also suffer increased delays during collisions when the core has been instantiated to form a bi-directional bridge. Locked accesses that abort on-going read operations will also mean additional delays.

If the core has been implemented to use AMBA SPLIT responses there will be an additional delay where, typically, one cycle is required for the arbiter to react to the assertion of HSPLIT and one clock cycle for the repetition of the address phase.

Note that if the core has support for read and/or write combining, the number of cycles required for the master will change depending on the access size and length of the incoming burst access. For instance, in a system where the bus in the core's master side is wider than the bus on the slave side, write combining will allow the core to accept writes with zero wait states and then combine several accesses into one or several larger access. Depending on memory controller implementation this could reduce the time required to move data to external memory, and will reduce the load on the master side bus.

Table 24. Access latencies

Access	**Master acc. cycles**	**Slave cycles**	**Delay incurred by performing access over core**
Single read	3	1	1 * clk_{slv} + 3 * clk_{mst}
Burst read with prefetch	2 + (burst length)^{x}	2	2 * clk_{slv} + (2 + burst length)* clk_{mst}
Single write^{xx}	(2)	0	0
Burst write^{xx}	(2 + (burst length))	0	0

^{x} A prefetch operation ends at the address boundary defined by the prefetch buffer's size

^{xx} The core implements posted writes, the number of cycles taken by the master side can only affect the next access.

2.3 Registers

The core does not implement any registers.

2.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x020. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

2.5 Implementation

2.5.1 Technology mapping

The uni-directional AHB to AHB bridge has two technology mapping generics *memtech* and fcfsmtech. *memtech* selects which memory technology that will be used to implement the FIFO memories. *fcfsmtech* selects the memory technology to be used to implement the First-come, first-served buffer, if FCFS is enaled.

2.5.2 RAM usage

The uni-directional AHB to AHB bridge instantiates one or several *syncram_2p* blocks from the technology mapping library (TECHMAP). If prefetching is enabled max(*mstmaccsz, slvaccsz*)/32 *syncram_2p* block(s) with organization (max*(rburst,iburst*)-max(*mstmaccsz, slvaccsz*)/32) x 32 is used to implement read FIFO (max*(rburst,iburst*) is the size of the read FIFO in 32-bit words). max(*mstmaccsz, slvaccsz*)/32 *syncram_2p* block(s) with organization *(wburst* - max(*mstmaccsz, slvaccsz*)/32) x 32, is always used to implement the write FIFO (where *wburst* is the size of the write FIFO in 32-bit words).

If the core has support for first-come, first-served ordering then one *fcfs* x 4 *syncram_2p* block will be instantiated, using the technology specified by the VHDL generic *fcfsmtech*.

2.6 Configuration options

Table 25 shows the configuration options of the core (VHDL generics).

Table 25. Configuration options (VHDL generics)

Generic	**Function**	**Allowed range**	**Default**
memtech	Memory technology		
hsindex	Slave I/F AHB index	0 to NAHBMAX-1	0
hmindex	Master I/F AHB index	0 to NAHBMAX-1	0
dir	0 - clock frequency on the master bus is lower than or equal to the frequency on the slave bus 1 - clock frequency on the master bus is higher than or equal to the frequency on the slave bus (for VHDL generic *ffact* = 1 the value of dir does not matter)	0 - 1	0
ffact	Frequency scaling factor between AHB clocks on master and slave buses.	1 - 15	2
slv	Slave bridge. Used in bi-directional bridge configuration where *slv* is set to 0 for master bridge and 1 for slave bridge. When a deadlock condition is detected slave bridge (*slv*=1) will give RETRY response to current access, effectively resolving the deadlock situation. This generic must only be set to 1 for a bridge where the frequency of the bus connecting the master interface is higher or equal to the frequency of the AHB bus connecting to the bridge's slave interface. Otherwise a race condition during access collisions may cause the bridge to deadlock.	0 - 1	0
pfen	Prefetch enable. Enables read FIFO.	0 - 1	0
irqsync	Interrupt forwarding. Forward interrupts from slave interface to master interface and vice versa. 0 - no interrupt forwarding, 1 - forward interrupts 1 - 15, 2 - forward interrupts 0 - 31. Since interrupts are forwarded in both directions, interrupt forwarding should be enabled for one bridge only in a bi-directional AHB/AHB bridge.	0 - 2	0
wburst	Length of write bursts in 32-bit words. Determines write FIFO size and write burst address boundary. If the wburst generic is set to 2 the bridge will not perform write bursts over a 2x4=8 byte boundary. This generic must be set so that the buffer can contain two of the maximum sized accesses that the bridge can handle.	2 - 32	8
iburst	Instruction fetch burst length. This value is only used if the generic *ibrsten* is set to 1. Determines the length of prefetching instruction read bursts on the master side. The maximum of (iburst,rburst) determines the size of the core's read buffer FIFO.	4 - 8	8

Table 25. Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
rburst	Incremental read burst length. Determines the maximum length of incremental read burst of unspecified length (INCR) on the master interface. The maximum of *rburst* and *iburst* determine the read burst boundary. As an example, if the maximum value of these generics is 8 the bridge will not perform read bursts over a 8x4=32 byte boundary. This generic must be set so that the buffer can contain two of the maximum sized accesses that the bridge can handle. For systems where AHB masters perform fixed length burst (INCRx , WRAPx) *rburst* should not be less than the length of the longest fixed length burst.	4 - 32	8
bar0	Address area 0 decoded by the bridge's slave interface. Appears as memory address register (BAR0) on the slave interface. The generic has the same bit layout as bank address registers with bits [19:18] suppressed (use functions ahb2ahb_membar and ahb2ahb_iobar in gaisler.misc package to generate this generic).	0 - 1073741823	0
bar1	Address area 1 (BAR1)	0 - 1073741823	0
bar2	Address area 2 (BAR2)	0 - 1073741823	0
bar3	Address area 3 (BAR2)	0 - 1073741823	0
sbus	The number of the AHB bus to which the slave interface is connected. The value appears in bits [1:0] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	0
mbus	The number of the AHB bus to which the master interface is connected. The value appears in bits [3:2] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	0
ioarea	Address of the I/O area containing the configuration area for AHB bus connected to the bridge's master interface. This address appears in the bridge's slave interface user-defined register 1. In order for a master on the slave interface's bus to access the configuration area on the bus connected to the bridge's master interface, the I/O area must be mapped on one of the bridge's BARs. If this generic is set to 0, some tools, such as Aeroflex Gaisler's GRMON debug monitor, will not perform Plug'n'Play scanning over the bridge.	0 - 16#FFF#	0
ibrsten	Instruction fetch burst enable. If set, the bridge will perform bursts of *iburst* length for opcode access (HPROT[0] = '0'), otherwise bursts of *rburst* length will be used for both data and opcode accesses.	0 - 1	0

Table 25. Configuration options (VHDL generics)

Generic	**Function**	**Allowed range**	**Default**
lckdac	Locked access error detection and correction. Locked accesses may lead to deadlock if a locked access is made while an ongoing read access has received a SPLIT response. The value of *lckdac* determines how the core handles this scenario: 0: Core will deadlock 1: Core will issue an AMBA ERROR response to the locked access 2: Core will allow both accesses to complete. If the core is used to create a bidirectional bridge, a deadlock condition may arise when locked accesses are made simultaneously in both directions. With *lckdac* set to 0 the core will deadlock. With *lckdac* set to a non-zero value the slave bridge will issue an ERROR response to the incoming locked access.	0 - 2	0
slvmaccsz	The maximum size of accesses that will be made to the bridge’s slave interface. This value must equal *mstmaccsz* unless *rdcomb* /= 0 and *wrcomb* /= 0.	32 - 256	32
mstmaccsz	The maximum size of accesses that will be performed by the bridge’s master interface. This value must equal *mstmaccsz* unless *rdcomb* /= 0 and *wrcomb* /= 0.	32 - 256	32
rdcomb	Read combining. If this generic is set to a non-zero value the core will use the master interface’s maximum AHB access size when prefetching data and allow data to be read out using any other access size supported by the slave interface. If slvmaccsz > 32 and mstmaccsz > 32 and an incoming single access, or access to a non-prefetchable area, is larger than the size supported by the master interface the bridge will perform a series of small accesses in order to fetch all the data. If this generic is set to 2 the core will use a burst of small fetches. If this generic is set to 1 the bridge will not use a burst unless the incoming access was a burst. Read combining is only supported for single accesses and incremental bursts of unspecified length.	0 - 2	0
wrcomb	Write combining. If this generic is set to a non-zero value the core may assemble several small write accesses (that are part of a burst) into one or more larger accesses or assemble one or more accesses into several smaller accesses. The settings are as follows: 0: No write combining 1: Combine if burst can be preserved 2: Combine if burst can be preserved and allow single accesses to be converted to bursts (only applicable if slvmaccsz > 32) Only supported for single accesses and incremental bursts of unspecified length	0 - 2	0

Table 25. Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
combmask	Read/write combining mask. This generic determines which ranges that the core can perform read/write combining to (only available when rdcomb respectively wrcomb are non-zero). The value given for combmask is treated as a 16-bit vector with LSB bit (right-most) indicating address 0x0 - 0x10000000. Making an access to an address in an area marked as ‘0’ in combmask is equivalent to making an access over a bridge with rdcomb = 0 and wrcomb = 0. However, combmask is not taken into account when the core performs a prefetch operation (see pfen generic). When a prefetch operation is initiated, the core will always use the maximum supported access size (when rdcomb /= 0).	0 - 16#FFFF#	16#FFFF#
allbrst	Support all burst types 2: Support all types of burst and always prefetch for wrapping and fixed length bursts. 1: Support all types of bursts 0: Only support incremental bursts of unspecified length See section 2.2.7 for more information. When allbrst is enabled, the core’s read buffer (size set via rburst/iburst generics) must have at least 16 slots.	0 - 2	0
ifctrlen	Interface control enable. When this generic is set to 1 the input signals *ifctrl.mstifen* and *ifctrl.slvifen* can be used to force the AMBA slave respectively master interface into an idle state. This functionality is intended to be used when the clock of one interface has been gated-off and any stimuli on one side of the bridge should not be propagated to the interface on the other side of the bridge. When this generic is set to 0, the ifctrl.* input signals are unused.	0 - 1	0
fcfs	First-come, first-served operation. When this generic is set to a non-zero value, the core will keep track of the order of incoming accesses and handle the requests in the same order. If this generic is set to zero the bridge will not preserve the order and leave this up to bus arbitration. If FCFS is enabled the value of this generic must be higher or equal to the number of masters that may perform accesses over the bridge.	0 - NAHBMST	0
fcfsmtech	Memory technology to use for FCFS buffer. When VHDL generic *fcfs* is set to a non-zero value, the core will instantiate a 4 bit x *fcfs* buffer to keep track of the incoming master indexes. This generic decides the memory technology to use for the buffer.	0 - NTECH	0 (inferred)
scantest	Enable scan support	0 - 1	0
split	Use AMBA SPLIT responses. When this generic is set to 1 the core will issue AMBA SPLIT responses. When this generic is set to 0 the core will insert waitstates instead and may also issue AMBA RETRY responses. If this generic is set to 0, the *fcfs* generic must also be set to 0, otherwise a simulation failure will be asserted.	0 - 1	1

2.7 Signal descriptions

Table 26 shows the interface signals of the core (VHDL ports).

Table 26. Signal descriptions (VHDL ports)

Signal name	**Field**	**Type**	**Function**	**Active**
RST		Input	Reset	Low
HCLKM		Input	AHB master bus clock	-
HCLKS		Input	AHB slave bus clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
AHBSO2	*	Input	AHB slave input vector signals (on master i/f side). Used to decode cachability and prefetchability Plug&Play information on bus connected to the bridge's master interface.	-
LCKI	slck blck mlck	Input	Used in systems with multiple AHB/AHB bridges (e.g. bi-directional AHB/AHB bridge) to detect deadlock conditions. Tie to "000" in systems with only uni-directional AHB/AHB bus.	High
LCKO	slck blck mlck	Output	Indicates possible deadlock condition	High
IFCTRL	mstifen	Input	Enable master interface. This input signal is unused if the VHDL generic *ifctrlen* is 0. If VHDL generic *ifctrlen* is 1 this signal must be set to '1' in order to enable the core's AMBA master interface, otherwise the master interface will always be idle and will not respond to stimuli on the core's AMBA slave interface. This signal is intended to be used to keep the core's master interface in a good state when the core's slave interface clock has been gated off. Care should be taken to ensure that the bridge is idle when the master interface is disabled.	High
	slvifen	Input	Enable slave interface. This input signal is unused if the VHDL generic *ifctrlen* is 0. If VHDL generic *ifctrlen* is 1 this signal must be set to '1' in order to enable the core's AMBA slave interface, otherwise the interface will always be ready and the bridge will not propagate stimuli on the core's AMBA slave interface to the core's AMBA master interface. This signal is intended to be used to keep the slave interface in a good state when the core's master interface clock has been gated off. Care should be taken to ensure that the bridge is idle when the slave interface is disabled.	High

* see GRLIB IP Library User's Manual

2.8 Library dependencies

Table 27 shows the libraries used when instantiating the core (VHDL libraries).

Table 27. Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

2.9 Instantiation

GRLIB contains two example designs with AHB2AHB and LEON processors: *designs/leon3-ahb2ahb* (only available in commercial distributions) and *designs/leon4-ahb2ahb* (only in distributions that include LEON4 processor). The LEON/GRLIB Configuration and Development Guide contains more information on how to use the bridge to create multi-bus systems.

3 AHBBRIDGE - Bi-directional AHB/AHB bridge

3.1 Overview

A pair of uni-directional bridges (AHB2AHB) can be instantiated to form a bi-directional bridge. The bi-directional AHB/AHB bridge (AHBBRIDGE) instantiates two uni-directional bridges that are configured to suit the bus architecture shown in figure 2. The bus architecture consists of two AHB buses: a high-speed AHB bus hosting LEON3 CPU(s) and an external memory controller and a low-speed AHB bus hosting communication IP-cores.

Note: For other architectures, a more general bi-directional bridge that is more suitable can be created by instantiating two uni-directional AHB to AHB bridges (see AHB2AHB core). AHBBRIDGE is not suitable for LEON4 systems and for other systems with wide AHB buses.

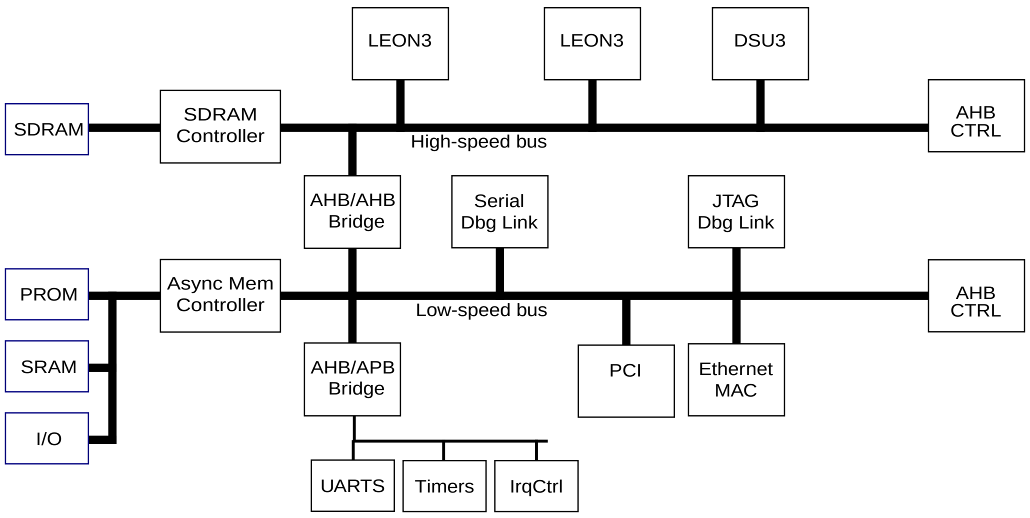


Figure 2. LEON3 system with a bi-directional AHB/AHB bridge

3.2 Operation

3.2.1 General

The AHB/AHB bridge is connected to each AHB bus through a pair consisting of an AHB master and an AHB slave interface. The address space occupied by the AHB/AHB bridge on each bus is determined by Bank Address Registers which are configured through VHDL generics. The bridge is capable of handling single and burst transfers in both directions. Internal FIFOs are used for data buffering. The bridge implements the AMBA SPLIT response to improve AHB bus utilization. For more information on AHB transfers please refer to the documentation for the uni-directional AHB/AHB bridge (AHB2AHB).

The requirements on the two bus clocks are that they are synchronous. The two uni-directional bridges forming the bi-directional AHB/AHB bridge are configured asymmetrically. Configuration of the bridge connecting high-speed bus with the low-speed bus (down bus) is optimized for the bus traffic generated by the LEON3 CPU since the CPU is the only master on the high-speed bus (except for the bridge itself). Read transfers generated by the CPU are single read transfers generated by single load instructions (LD), read bursts of length two generated by double load instructions (LDD) or incremental read bursts of maximal length equal to cache line size (4 or 8 words) generated during instruction cache line fill. The size of the read FIFO for the down bridge is therefore configurable to 4 or 8 entries which is the maximal read burst length. If a read burst is an instruction fetch (indicated on AHB HPROT signal) to a prefetchable area the bridge will prefetch data to the end of a instruction cache line. If a read burst to a prefetchable area is a data access, two words will be prefetched (this transfer is generated by the LDD instruction). The write FIFO has two entries capable of buffering the longest write burst (generated by the STD instruction). The down bridge also performs interrupt forwarding, interrupt lines 1-15 on both buses are monitored and an interrupt on one bus is forwarded to the other one.

Since the low-speed bus does not host a LEON3 CPU, all AHB transfers forwarded by the uni-directional bridge connecting the low-speed bus and the high-speed bus (up bridge) are data transfers. Therefore the bridge does not make a distinction between instruction and data transfers. The size of the read and write FIFOs for this bridge is configurable and should be set by the user to suite burst transfers generated by the cores on the low-speed bus.

Note that the bridge has been optimized for a LEON3 system with a specific set of masters and a specific bus topology. Therefore the core may not be suitable for a design containing later versions of the LEON processor or other masters. In general it is not recommended instantiate the AHBBRIDGE core and instead instantiate two uni-directional AHB to AHB bridges (AHB2AHB cores) with configurations tailored for a specific design.

3.2.2 Deadlock conditions

A deadlock situation can occur if the bridge is simultaneously accessed from both buses. The bridge contains deadlock detection logic which will resolve a deadlock condition by giving a RETRY response on the low-speed bus.

There are several deadlock conditions that can occur with locked accesses. If the VHDL generic *lckdac* is 0, the bridge will deadlock if two simultaneous accesses from both buses are locked, or if a locked access is made while the bridge has issued a SPLIT response to a read access and the splitted access has not completed. If *lckdac* is greater than 0, the bridge will resolve the deadlock condition from two simultaneous locked accesses by giving an ERROR response on the low-speed bus. If *lckdac* is 1 and a locked access is made while the bridge has issued a SPLIT response to a read access, the bridge will respond with ERROR to the incoming locked access. If *lckdac* is 2 the bridge will allow both the locked access and the splitted read access to complete. Note that with *lckdac* set to 2 and two incoming locked accesses, the access on the low-speed bus will still receive an ERROR response.

3.2.3 Read and write combining

The bridge can be configured to support read and write combining so that prefetch operations and write bursts are always performed with the maximum access size possible on the master interface. Please see the documentation for the uni-directional AHB/AHB bridge (AHB2AHB) for a description of read and write combining and note that the same VHDL generics are used to specify both the maximum master and maximum slave access size on the bi-directional AHB/AHB bridge.

3.3 Registers

The core does not implement any registers.

3.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x020. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

3.5 Configuration options

Table 28 shows the configuration options of the core (VHDL generics).

Table 28. Configuration options

Generic	**Function**	**Allowed range**	**Default**
memtech	Memory technology	-	0
ffact	Frequency ratio	1 -	2
hsb_hsindex	AHB slave index on the high-speed bus	0 to NAHBMAX-1	0
hsb_hmindex	AHB master index on the high-speed bus	0 to NAHBMAX-1	0
hsb_iclsize	Cache line size (in number of 32-bit words) for CPUs on the high-speed bus. Determines the number of the words that are prefetched by the bridge when CPU performs instruction bursts.	4, 8	8
hsb_bank0	Address area 0 mapped on the high-speed bus and decoded by the bridge's slave interface on the low-speed bus. Appears as memory address register (BAR0) on the bridge's low-speed bus slave interface. The generic has the same bit layout as bank address registers with bits [19:18] suppressed (use functions ahb2ahb_membar and ahb2ahb_iobar in gaisler.misc package to generate this generic).	0 - 1073741823	0
hsb_bank1	Address area 1 mapped on the high-speed bus	0 - 1073741823	0
hsb_bank2	Address area 2 mapped on the high-speed bus	0 - 1073741823	0
hsb_bank3	Address area 3 mapped on the high-speed bus	0 - 1073741823	0
hsb_ioarea	Address of high-speed bus I/O area that contains the high-speed bus configuration area. Will appear in the bridge's user-defined register 1 on the low-speed bus. Note that to allow low-speed bus masters to read the high-speed bus configuration area, the area must be mapped on one of the *hsb_bank* generics.	0 - 16#FFF#	0
lsb_hsindex	AHB slave index on the low-speed bus	0 to NAHBMAX-1	0
lsb_hmindex	AHB master index on the low-speed bus	0 to NAHBMAX-1	0
lsb_rburst	Size of the prefetch buffer for read transfers initiated on the low-speed-bus and crossing the bridge.	16, 32	16
lsb_wburst	Size of the write buffer for write transfers initiated on the low-speed bus and crossing the bridge.	16, 32	16
lsb_bank0	Address area 0 mapped on the low-speed bus and decoded by the bridge's slave interface on the high-speed bus. Appears as memory address register (BAR0) on the bridge's high-speed bus slave interface. The generic has the same bit layout as bank address registers with bits [19:18] suppressed (use functions ahb2ahb_membar and ahb2ahb_iobar in gaisler.misc package to generate this generic).	0 - 1073741823	0
lsb_bank1	Address area 1 mapped on the low-speed bus	0 - 1073741823	0
lsb_bank2	Address area 2 mapped on the low-speed bus	0 - 1073741823	0
lsb_bank3	Address area 3 mapped on the low-speed bus	0 - 1073741823	0
lsb_ioarea	Address of low-speed bus I/O area that contains the low-speed bus configuration area. Will appear in the bridge's user-defined register 1 on the high-speed bus. Note that to allow high-speed bus masters to read the low-speed bus configuration area, the area must be mapped on one of the *lsb_bank* generics.	0 - 16#FFF#	0

Table 28. Configuration options

Generic	Function	Allowed range	Default
lckdac	Locked access error detection and correction. This generic is mapped to the generic with the same name on the two AHB2AHB cores instantiated by AHBBRIDGE. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	0 - 2	0
maccsz	This generic is propagated to the slvmaccsz and mst-maccsz VHDL generics on the two AHB2AHB cores instantiated by AHBBRIDGE. The generic determines the maximum AHB access size supported by the bridge. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	32 - 256	32
rdcomb	Read combining, this generic is mapped to the generic with the same name on the two AHB2AHB cores instantiated by AHBBRIDGE. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	0 - 2	0
wrcomb	Write combining, this generic is mapped to the generic with the same name on the two AHB2AHB cores instantiated by AHBBRIDGE. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	0 - 2	0
combmask	Read/Write combining mask, this generic is mapped to the generic with the same name on the two AHB2AHB cores instantiated by AHBBRIDGE. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	0 - 16#FFFF#	16#FFFF#
allbrst	Support all burst types, this generic is mapped to the generic with the same name on the two AHB2AHB cores instantiated by AHBBRIDGE. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	0 - 2	0
fcfs	First-come, first-served operation, this generic is mapped to the generic with the same name on the two AHB2AHB cores instantiated by AHBBRIDGE. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	0 - NAHBMST	0
scantest	Enable scan support	0 - 1	0

3.6 Signal descriptions

Table 29 shows the interface signals of the core (VHDL ports).

Table 29. Signal descriptions

Signal name	Type	Function	Active
RST	Input	Reset	Low
HSB_HCLK	Input	High-speed AHB clock	-
LSB_HCLK	Input	Low-speed AHB clock	-
HSB_AHBSI	Input	High-speed bus AHB slave input signals	-
HSB_AHBSO	Output	High-speed bus AHB slave output signals	-
HSB_AHBSOV	Input	High-speed bus AHB slave input signals	-
HSB_AHBMI	Input	High-speed bus AHB master input signals	-
HSB_AHBMO	Output	High-speed bus AHB master output signals	-
LSB_AHBSI	Input	Low-speed bus AHB slave input signals	-
LSB_AHBSO	Output	Low-speed bus AHB slave output signals	-
LSB_AHBSOV	Input	Low-speed bus AHB slave input signals	-
LSB_AHBMI	Input	Low-speed bus AHB master input signals	-
LSB_AHBMO	Output	Low-speed bus AHB master output signals	-

3.7 Library dependencies

Table 30 shows the libraries used when instantiating the core (VHDL libraries).

Table 30. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

4 AHBCTRL - AMBA AHB controller with plug&play support

4.1 Overview

The AMBA AHB controller is a combined AHB arbiter, bus multiplexer and slave decoder according to the AMBA 2.0 standard.

The controller supports up to 16 AHB masters, and 16 AHB slaves. The maximum number of masters and slaves are defined in the GRLIB.AMBA package, in the VHDL constants NAHBSLV and NAHBMST. It can also be set with the *nahbm* and *nahbs* VHDL generics.

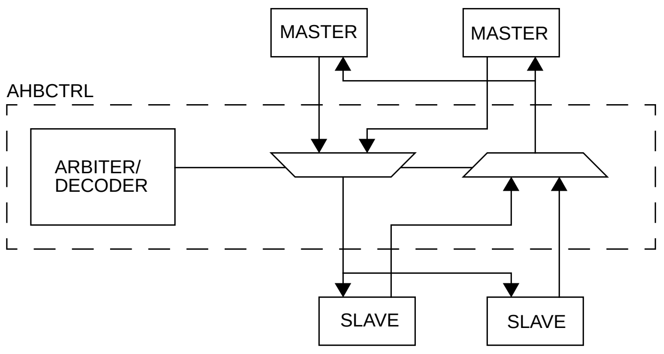


Figure 3. AHB controller block diagram

4.2 Operation

4.2.1 Arbitration

The AHB controller supports two arbitration algorithms: fixed-priority and round-robin. The selection is done by the VHDL generic *rrobin*. In fixed-priority mode (*rrobin* = 0), the bus request priority is equal to the master's bus index, with index 0 being the lowest priority. If no master requests the bus, the master with bus index 0 (set by the VHDL generic *defmast)* will be granted.

In round-robin mode, priority is rotated one step after each AHB transfer. If no master requests the bus, the last owner will be granted (bus parking). The VHDL generic *mprio* can be used to specify one or more masters that should be prioritized when the core is configured for round-robin mode.

Note that there are AHB slaves that implement split-like functionality by giving AHB retry responses until the access has finished and the original master tries again. All masters on the bus accessing such slaves must be round-robin arbitrated without prioritization to avoid deadlock situations. For GRLIB this applies to the GRPCI and GRPCI2 cores.

During incremental bursts, the AHB master should keep the bus request asserted until the last access as recommended in the AMBA 2.0 specification, or it might loose bus ownership. For fixed-length burst, the AHB master will be granted the bus during the full burst, and can release the bus request immediately after the first access has started. For this to work however, the VHDL generic *fixbrst* should be set to 1.

4.2.2 Decoding

Decoding (generation of HSEL) of AHB slaves is done using the plug&play method explained in the GRLIB User's Manual. A slave can occupy any binary aligned address space with a size of 1 - 4096 Mbyte. A specific I/O area is also decoded, where slaves can occupy 256 byte - 1 Mbyte. The default address of the I/O area is 0xFFF00000, but can be changed with the *ioaddr* and *iomask* VHDL generics. Access to unused addresses will cause an AHB error response.

The I/O area can be placed within a memory area occupied by a slave. The slave will not be selected when the I/O area is accessed.

4.2.3 Plug&play information

GRLIB devices contain a number of plug&play information words which are included in the AHB records they drive on the bus (see the GRLIB user's manual for more information). These records are combined into an array which is connected to the AHB controller unit.

The plug&play information is mapped on a read-only address area, defined by the *cfgaddr* and *cfgmask* VHDL generics, in combination with the *ioaddr* and *iomask* VHDL generics. By default, the area is mapped on address 0xFFFFF000 - 0xFFFFFFFF. The master information is placed on the first 2 kbyte of the block (0xFFFFF000 - 0xFFFFF800), while the slave information is placed on the second 2 kbyte block. Each unit occupies 32 bytes, which means that the area has place for 64 masters and 64 slaves. The address of the plug&play information for a certain unit is defined by its bus index. The address for masters is thus 0xFFFFF000 + n*32, and 0xFFFFF800 + n*32 for slaves.

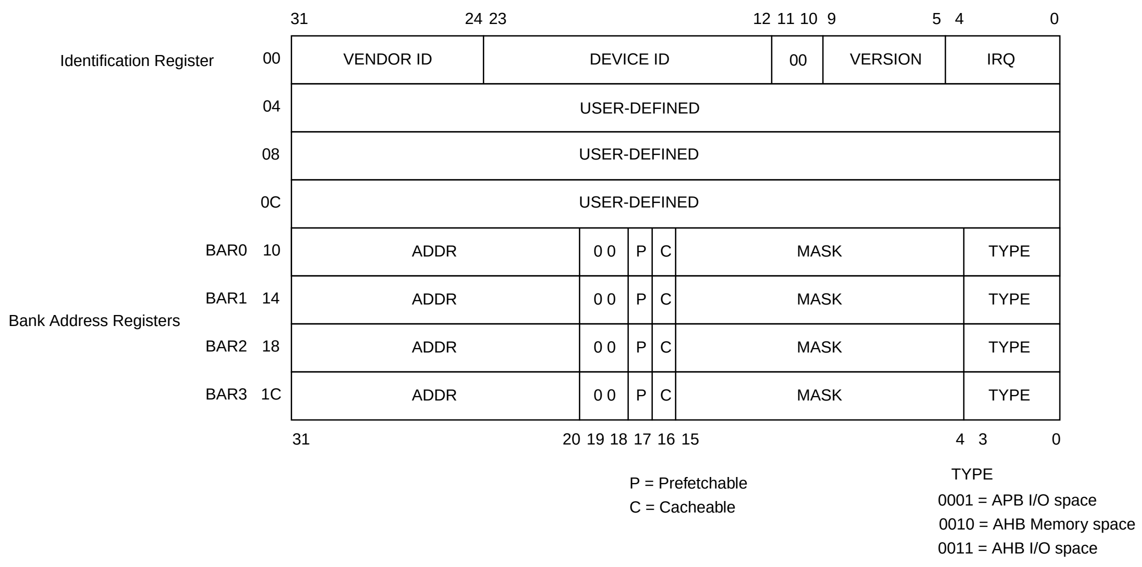


Figure 4. AHB plug&play information record

4.3 AHB split support

AHB SPLIT functionality is supported if the *split* VHDL generic is set to 1. In this case, all slaves must drive the AHB SPLIT signal.

It is important to implement the split functionality in slaves carefully since locked splits can otherwise easily lead to deadlocks. A locked access to a slave which is currently processing (it has returned a split response but not yet split complete) an access which it returned split for to another master must be handled first. This means that the slave must either be able to return an OKAY response to the locked access immediately or it has to split it but return split complete to the master performing the locked transfer before it has finished the first access which received split.

4.4 Locked accesses

The GRLIB AHB controller treats HLOCK as coupled to a specific access. If a previous access by a master received a SPLIT/RETRY response then the arbiter will disregard the current value of HLOCK. This is done as opposed to always treating HLOCK as being valid for the next access which can result in a previously non-locked access being treated as locked when it is retried. Consider the following sequence:

T0: MSTx write 0
T1: MSTx write 1, HLOCK asserted as next access performed by master will be locked
T2: MSTx locked read

If (the non-locked) write 0 access at T0 receives a RETRY or SPLIT response (given at time T1), then the next access to be performed may be a retry of write 0. In this case the arbiter will disregard the HLOCK setting and the retried access will not have HMASTLOCK set.

4.5 AHB bus monitor

An AHB bus monitor is integrated into the core. It is enabled with the *enbusmon* generic. It has the same functionality as the AHB and arbiter parts in the AMBA monitor core (AMBAMON). For more information on which rules are checked se the AMBAMON documentation.

4.6 Registers

The core does not implement any registers.

4.7 Configuration options

Table 31 shows the configuration options of the core (VHDL generics).

Table 31. Configuration options

Generic	**Function**	**Allowed range**	**Default**
ioaddr	The MSB address of the I/O area. Sets the 12 most significant bits in the 32-bit AHB address (i.e. 31 downto 20)	0 - 16#FFF#	16#FFF#
iomask	The I/O area address mask. Sets the size of the I/O area and the start address together with ioaddr.	0 - 16#FFF#	16#FFF#
cfgaddr	The MSB address of the configuration area. Sets 12 bits in the 32-bit AHB address (i.e. 19 downto 8).	0 - 16#FFF#	16#FF0#
cfgmask	The address mask of the configuration area. Sets the size of the configuration area and the start address together with cfgaddr. If set to 0, the configuration will be disabled.	0 - 16#FFF#	16#FF0#
rrobin	Selects between round-robin (1) or fixed-priority (0) bus arbitration algorithm.	0 - 1	0
split	Enable support for AHB SPLIT response	0 - 1	0
defmast	Default AHB master	0 - NAHBMST-1	0
ioen	AHB I/O area enable. Set to 0 to disable the I/O area	0 - 1	1
disirq	Set to 1 to disable interrupt routing	0 - 1	0
nahbm	Number of AHB masters	1 - NAHBMST	NAHBMST
nahbs	Number of AHB slaves	1 - NAHBSLV	NAHBSLV
timeout	Perform bus timeout checks (NOT IMPLEMENTED).	0 - 1	0
fixbrst	Enable support for fixed-length bursts	0 - 1	0
debug	Print configuration (0=none, 1=short, 2=all cores)	0 - 2	2
fpnpen	Enables full decoding of the PnP configuration records. When disabled the user-defined registers in the PnP configuration records are not mapped in the configuration area.	0 - 1	0
icheck	Check bus index	0 - 1	1
devid	Assign unique device identifier readable from plug and play area.	N/A	0
enbusmon	Enable AHB bus monitor	0 - 1	0

Table 31. Configuration options

Generic	Function	Allowed range	Default
assertwarn	Enable assertions for AMBA recommendations. Violations are asserted with severity warning.	0 - 1	0
asserterr	Enable assertions for AMBA requirements. Violations are asserted with severity error.	0 - 1	0
hmstdisable	Disable AHB master rule check. To disable a master rule check a value is assigned so that the binary representation contains a one at the position corresponding to the rule number, e.g 0x80 disables rule 7.	N/A	0
hslvdisable	Disable AHB slave tests. Values are assigned as for hmstdisable.	N/A	0
arbdisable	Disable Arbiter tests. Values are assigned as for hmstdisable.	N/A	0
mprio	Master(s) with highest priority. This value is converted to a vector where each position corresponds to a master. To prioritize masters x and y set this generic to $2^x + 2^y$.	N/A	0
mcheck	Check if there are any intersections between core memory areas. If two areas intersect an assert with level failure will be triggered (in simulation). mcheck = 1 does not report intersects between AHB IO areas and AHB memory areas (as IO areas are allowed to override memory areas). mcheck = 2 triggers on all overlaps.	0 - 2	1
ccheck	Perform sanity checks on PnP configuration records (in simulation).	0 - 1	1
acdm	AMBA compliant data multiplexing (for HSIZE > word). If this generic is set to 1, and the AMBA bus data width in the system exceeds 32-bits, the core will ensure AMBA compliant data multiplexing for access sizes (HSIZE) over 32-bits. GRLIB cores have an optimization where they drive the same data on all lanes. Read data is always taken from the lowest lanes. If an AMBA compliant core from another vendor is introduced in the design, that core may not always place valid data on the low part of the bus. By setting this generic to 1, the AHBCTRL core will replicate the data, allowing the non-GRLIB cores to be instantiated without modification.	0 - 1	0
index	AHB index for trace print-out, currently unused	N/A	0
ahbtrace	AHB trace print-out to simulator console in simulation.	0 - 1	0
hwdebug	Enable hardware debug registers. If this generic is set to 1 the configuration area will include to diagnostic registers at offsets 0xFF4 and 0xFF8. Offset 0xFF4 will show a 32-bit register where bit n shows the current status of AHB master n's HBUSREQ signal. Offset 0xFF8 will show a 32-bit register where bit n shows the current SPLIT status of AHB master n. The bit will be set when AHB master n receives a SPLIT reply and will be re-set to '0' when HSPLIT for AHB master n has been asserted. This functionality is not intended to be used in production systems but can provide valuable information while debugging systems with cores that have problems with AMBA SPLIT replies.	0 - 1	0

4.8 Signal descriptions

Table 32 shows the interface signals of the core (VHDL ports).

Table 32. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
MSTI	*	Output	AMBA AHB master interface record array	-
MSTO	*	Input	AMBA AHB master interface record array	-
SLVI	*	Output	AMBA AHB slave interface record array	-
SLVO	*	Input	AMBA AHB slave interface record array	-

* see GRLIB IP Library User's Manual

4.9 Library dependencies

Table 33 shows libraries used when instantiating the core (VHDL libraries).

Table 33. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions

4.10 Component declaration

```
library grlib;
use grlib.amba.all;

component ahbctrl
  generic (
    defmast : integer := 0;-- default master
    split   : integer := 0;-- split support
    rrobin  : integer := 0;-- round-robin arbitration
    timeout : integer range 0 to 255 := 0;  -- HREADY timeout
    ioaddr  : ahb_addr_type := 16#fff#;  -- I/O area MSB address
    iomask  : ahb_addr_type := 16#fff#;   -- I/O area address mask
    cfgaddr : ahb_addr_type := 16#ff0#;  -- config area MSB address
    cfgmask : ahb_addr_type := 16#ff0#; -- config area address maskk
    nahbm   : integer range 1 to NAHBMST := NAHBMST; -- number of masters
    nahbs   : integer range 1 to NAHBSLV := NAHBSLV; -- number of slaves
    ioen    : integer range 0 to 15 := 1;   -- enable I/O area
    disirq  : integer range 0 to 1 := 0; -- disable interrupt routing
    fixbrst : integer range 0 to 1 := 0; -- support fix-length bursts
    debug : integer range 0 to 2 := 2; -- print configuration to consolee
    fpnpen : integer range 0 to 1 := 0;     -- full PnP configuration decoding
    icheck  : integer range 0 to 1 := 1
    devid       : integer := 0;     -- unique device ID
    enbusmon    : integer range 0 to 1 := 0; --enable bus monitor
    assertwarn  : integer range 0 to 1 := 0; --enable assertions for warnings
    asserterr   : integer range 0 to 1 := 0; --enable assertions for errors
    hmstdisable : integer := 0; --disable master checks
    hslvdisable : integer := 0; --disable slave checks
    arbdisable  : integer := 0; --disable arbiter checks
    mprio       : integer := 0; --master with highest priority
    enebterm    : integer range 0 to 1 := 0  --enable early burst termination
);
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    msti    : out ahb_mst_in_type;
    msto    : in  ahb_mst_out_vector;
    slvi    : out ahb_slv_in_type;
    slvo    : in  ahb_slv_out_vector;
    testen  : in  std_ulogic := ’0’;
    testrst : in  std_ulogic := ’1’;
    scanen  : in  std_ulogic := ’0’;
    testoen : in  std_ulogic := ’1’
  );
  end component;
```

4.11 Instantiation

This example shows the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;

.
.

  -- AMBA signals
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

begin

-- ARBITER

ahb0 : ahbctrl -- AHB arbiter/multiplexer
  generic map (defmast => CFG_DEFMST, split => CFG_SPLIT,
rrobin => CFG_RROBIN, ioaddr => CFG_AHBIO, nahbm => 8, nahbs => 8)
  port map (rstn, clkm, ahbmi, ahbmo, ahbsi, ahbso);


-- AHB slave

sr0 : srctrl generic map (hindex => 3)
port map (rstn, clkm, ahbsi, ahbso(3), memi, memo, sdo3);

-- AHB master

e1 : eth_oc
      generic map (mstndx => 2, slvndx => 5, ioaddr => CFG_ETHIO, irq => 12, memtech =>
memtech)
      port map (rstn, clkm, ahbsi, ahbso(5), ahbmi  => ahbmi,
ahbmo => ahbmo(2), ethi1, etho1);
...
end;
```

4.12 Debug print-out

If the debug generic is set to 2, the plug&play information of all attached AHB units are printed to the console during the start of simulation. Reporting starts by scanning the master interface array from 0 to NAHBMST - 1 (defined in the grlib.amba package). It checks each entry in the array for a valid vendor-id (all nonzero ids are considered valid) and if one is found, it also retrieves the device-id. The descriptions for these ids are obtained from the GRLIB.DEVICES package, and are then printed on standard out together with the master number. If the index check is enabled (done with a VHDL generic), the report module also checks if the hindex number returned in the record matches the array number of the record currently checked (the array index). If they do not match, the simulation is aborted and an error message is printed.

This procedure is repeated for slave interfaces found in the slave interface array. It is scanned from 0 to NAHBSLV - 1 and the same information is printed and the same checks are done as for the master interfaces. In addition, the address range and memory type is checked and printed. The address information includes type, address, mask, cacheable and pre-fetchable fields. From this information, the report module calculates the start address of the device and the size of the range. The information finally printed is type, start address, size, cacheability and pre-fetchability. The address ranges currently defined are AHB memory, AHB I/O and APB I/O. APB I/O ranges are ignored by this module.

```
# vsim -c -quiet leon3mp
VSIM 1> run
# LEON3 MP Demonstration design
# GRLIB Version 1.0.7
# Target technology: inferred,  memory library: inferred
# ahbctrl: AHB arbiter/multiplexer rev 1
# ahbctrl: Common I/O area disabled
# ahbctrl: Configuration area at 0xfffff000, 4 kbyte
# ahbctrl: mst0: Aeroflex Gaisler        Leon3 SPARC V8 Processor
# ahbctrl: mst1: Aeroflex Gaisler        AHB Debug UART
# ahbctrl: slv0: European Space Agency   Leon2 Memory Controller
# ahbctrl:       memory at 0x00000000, size 512 Mbyte, cacheable, prefetch
# ahbctrl:       memory at 0x20000000, size 512 Mbyte
# ahbctrl:       memory at 0x40000000, size 1024 Mbyte, cacheable, prefetch
# ahbctrl: slv1: Aeroflex Gaisler        AHB/APB Bridge
# ahbctrl:       memory at 0x80000000, size 1 Mbyte
# apbctrl: APB Bridge at 0x80000000 rev 1
# apbctrl: slv0: European Space Agency   Leon2 Memory Controller
# apbctrl:       I/O ports at 0x80000000, size 256 byte
# apbctrl: slv1: Aeroflex Gaisler        Generic UART
# apbctrl:       I/O ports at 0x80000100, size 256 byte
# apbctrl: slv2: Aeroflex Gaisler        Multi-processor Interrupt Ctrl.
# apbctrl:       I/O ports at 0x80000200, size 256 byte
# apbctrl: slv3: Aeroflex Gaisler        Modular Timer Unit
# apbctrl:       I/O ports at 0x80000300, size 256 byte
# apbctrl: slv7: Aeroflex Gaisler        AHB Debug UART
# apbctrl:       I/O ports at 0x80000700, size 256 byte
# apbctrl: slv11: Aeroflex Gaisler        General Purpose I/O port
# apbctrl:       I/O ports at 0x80000b00, size 256 byte
# grgpio11: 8-bit GPIO Unit rev 0
# gptimer3: GR Timer Unit rev 0, 8-bit scaler, 2 32-bit timers, irq 8
# irqmp: Multi-processor Interrupt Controller rev 3, #cpu 1
# apbuart1: Generic UART rev 1, fifo 4, irq 2
# ahbuart7: AHB Debug UART rev 0
# leon3_0: LEON3 SPARC V8 processor rev 0
# leon3_0: icache 1*8 kbyte, dcache 1*8 kbyte
VSIM 2>
```

5 AHBJTAG - JTAG Debug Link with AHB Master Interface

5.1 Overview

The JTAG debug interface provides access to on-chip AMBA AHB bus through JTAG. The JTAG debug interface implements a simple protocol which translates JTAG instructions to AHB transfers. Through this link, a read or write transfer can be generated to any address on the AHB bus.

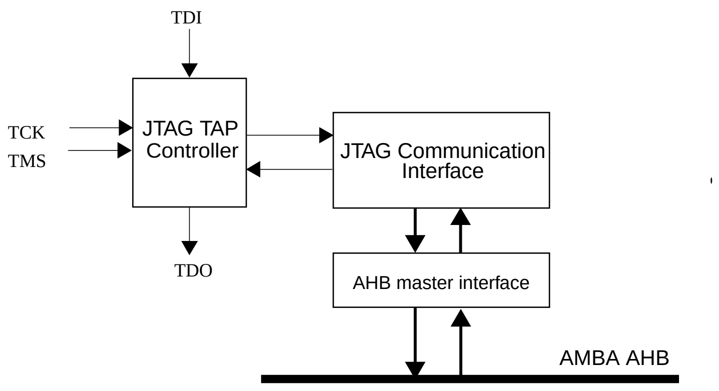


Figure 5. JTAG Debug link block diagram

5.2 Operation

5.2.1 Transmission protocol

The JTAG Debug link decodes two JTAG instructions and implements two JTAG data registers: the command/address register and data register. A read access is initiated by shifting in a command consisting of read/write bit, AHB access size and AHB address into the command/address register. The AHB read access is performed and data is ready to be shifted out of the data register. Write access is performed by shifting in command, AHB size and AHB address into the command/data register followed by shifting in write data into the data register. Sequential transfers can be performed by shifting in command and address for the transfer start address and shifting in SEQ bit in data register for following accesses. The SEQ bit will increment the AHB address for the subsequent access. Sequential transfers should not cross a 1 kB boundary. Sequential transfers are always word based.

Table 34. JTAG debug link Command/Address register

34	33 32	31 ... 0
W	SIZE	AHB ADDRESS

34	Write (W) - ‘0’ - read transfer, ‘1’ - write transfer
33 32	AHB transfer size - “00” - byte, “01” - half-word, “10” - word, “11”- reserved
31 30	AHB address

Table 35. JTAG debug link Data register

32	31 ... 0
SEQ	AHB DATA

32	Sequential transfer (SEQ) - If ‘1’ is shifted in this bit position when read data is shifted out or write data shifted in, the subsequent transfer will be to next word address. When read out from the device, this bit is ‘1’ if the AHB access has completed and ‘0’ otherwise.
31 30	AHB Data - AHB write/read data. For byte and half-word transfers data is aligned according to big-endian order where data with address offset 0 data is placed in MSB bits.

As of version 1 of the JTAG debug link the core will signal AHB access completion by setting bit 32 of the data register. In previous versions the debug host could not determine if an AHB accesses had finished when the read data was shifted out of the JTAG debug link data register. As of version 1 a debug host can look at bit 32 of the received data to determine if the access was successful. If bit 32 is ‘1’ the access completed and the data is valid. If bit 32 is ‘0’, the AHB access was not finished when the host started to read data. In this case the host can repeat the read of the data register until bit 32 is set to ‘1’, signaling that the data is valid and that the AMBA AHB access has completed.

It should be noted that while bit 32 returns ‘0’, new data will not be shifted into the data register. The debug host should therefore inspect bit 32 when shifting in data for a sequential AHB access to see if the previous command has completed. If bit 32 is ‘0’, the read data is not valid and the command just shifted in has been dropped by the core.

Inspection of bit 32 should not be done for JTAG Debug links with version number 0.

5.3 Implementation

5.3.1 Clocking

Except for the TAP state machine and instruction register, the JTAG debug link operates in the AMBA clock domain. To detect when to shift the address/data register, the JTAG clock and TDI are resynchronized to the AMBA domain. The JTAG clock must be less than 1/3 of the AHB clock frequency for the debug link commands to work when nsync=2, and less than 1/2 of the AHB frequency when nsync=1.

5.4 Registers

The core does not implement any registers mapped in the AMBA AHB or APB address space.

5.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x01C. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

5.6 Configuration options

Table 36 shows the configuration options of the core (VHDL generics).

Table 36. Configuration options

Generic	**Function**	**Allowed range**	**Default**
tech	Target technology	0 - NTECH	0
hindex	AHB master index	0 - NAHBMST-1	0
nsync	Number of synchronization registers between clock regions	1 - 2	1
idcode	JTAG IDCODE instruction code (generic tech only)	0 - 255	9
manf	Manufacturer id. Appears as bits 11-1 in TAP controllers device identification register. Used only for generic technology. Default is Aeroflex Gaisler manufacturer id.	0 - 2047	804
part	Part number (generic tech only). Bits 27-12 in device id. reg.	0 - 65535	0
ver	Version number (generic tech only). Bits 31-28 in device id. reg.	0 - 15	0
ainst	Code of the JTAG instruction used to access JTAG Debug link command/address register. For Actel TAPs (tech VHDL generic is set to an Actel technology) this generic should be set to 16, for all other technologies the default value (2) can be used.	0 - 255	2
dinst	Code of the JTAG instruction used to access JTAG Debug link data register For Actel TAPs (tech VHDL generic is set to an Actel technology) this generic should be set to 17, for all other technologies the default value (3) can be used.	0 - 255	3
scantest	Enable scan test support	0 - 1	0
oepol	Output enable polarity for TDOEN	0 - 1	1
tcknen	Support externally inverted TCK (generic tech only)	0 - 1	0

5.7 Signal descriptions

Table 37 shows the interface signals of the core (VHDL ports).

Table 37. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	System clock (AHB clock domain)	-
TCK	N/A	Input	JTAG clock*	-
TMS	N/A	Input	JTAG TMS signal*	High
TDI	N/A	Input	JTAG TDI signal*	High
TDO	N/A	Output	JTAG TDO signal*	High
AHBI	***	Input	AHB Master interface input	-
AHBO	***	Output	AHB Master interface output	-
TAPO_TCK	N/A	Output	TAP Controller User interface TCK signal**	High
TAPO_TDI	N/A	Output	TAP Controller User interface TDI signal**	High
TAPO_INST[7:0]	N/A	Output	TAP Controller User interface INSTsignal**	High
TAPO_RST	N/A	Output	TAP Controller User interface RST signal**	High
TAPO_CAPT	N/A	Output	TAP Controller User interface CAPT signal**	High
TAPO_SHFT	N/A	Output	TAP Controller User interface SHFT signal**	High
TAPO_UPD	N/A	Output	TAP Controller User interface UPD signal**	High
TAPI_TDO	N/A	Input	TAP Controller User interface TDO signal**	High
TRST	N/A	Input	JTAG TRST signal	Low
TDOEN	N/A	Output	Output-enable for TDO	See oepol
TCKN	N/A	Input	Inverted JTAG clock* (if tcknen is set)	-
TAPO_TCKN	N/A	Output	TAP Controller User interface TCKN signal**	High
TAPO_NINST	N/A	Output	TAP Controller User interface NINSTsignal**	High
TAPO_IUPD	N/A	Output	TAP Controller User interface IUPD signal**	High

*) If the target technology is Xilinx or Altera the cores JTAG signals TCK, TCKN, TMS, TDI and TDO are not used. Instead the dedicated FPGA JTAG pins are used. These pins are implicitly made visible to the core through TAP controller instantiation.

**) User interface signals from the JTAG TAP controller. These signals are used to interface additional user defined JTAG data registers such as boundary-scan register. For more information on the JTAG TAP controller user interface see JTAG TAP Controller IP-core documentation. If not used tie TAPI_TDO to ground and leave TAPO_* outputs unconnected.

***) see GRLIB IP Library User's Manual

5.8 Signal definitions and reset values

The signals and their reset values are described in table 38.

Table 38. Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
dsutck	Input	JTAG clock	-	-
dsutms	Input	JTAG TMS	High	-
dsutdi	Input	JTAG TDI	High	-
dsutdo	Output	JTAG TDO	High	undefined

5.9 Timing

The timing waveforms and timing parameters are shown in figure 6 and are defined in table 39.

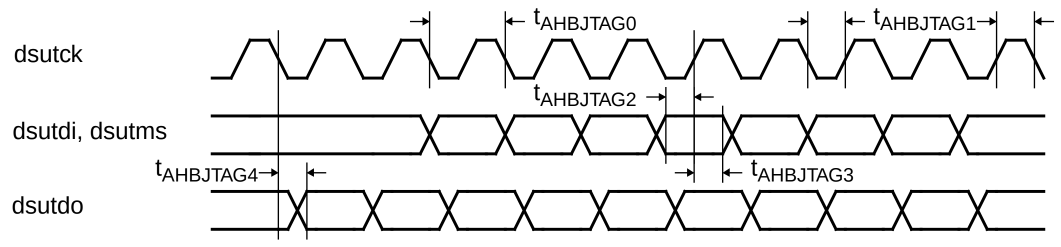


Figure 6. Timing waveforms

Table 39. Timing parameters

Name	**Parameter**	**Reference edge**	**Min**	**Max**	**Unit**
$t_{AHBJTAG0}$	clock period	-	100	-	ns
$t_{AHBJTAG1}$	clock low/high period	-	40	-	ns
$t_{AHBJTAG2}$	data input to clock setup	rising *dsutck* edge	15	-	ns
$t_{AHBJTAG3}$	data input from clock hold	rising *dsutck* edge	0	-	ns
$t_{AHBJTAG4}$	clock to data output delay	falling *dsutck* edge	-	25	ns

5.10 Library dependencies

Table 40 shows libraries used when instantiating the core (VHDL libraries).

Table 40. Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	JTAG	Signals, component	Signals and component declaration

5.11 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.jtag.all;

entity ahbjtag_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- JTAG signals
    tck  : in std_ulogic;
    tms  : in std_ulogic;
    tdi  : in std_ulogic;
    tdo  : out std_ulogic
);
end;

architecture rtl of ahbjtag_ex is
```

```
  -- AMBA signals
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
signal gnd : std_ulogic;

constant clkperiod : integer := 100;

begin

gnd <= ‘0’;

  -- AMBA Components are instantiated here
  ...

-- AHB JTAG
  ahbjtag0 : ahbjtag generic map(tech => 0, hindex => 1)
 port map(rstn, clkm, tck, tckn, tms, tdi, tdo, ahbmi, ahbmo(1),
               open, open, open, open, open, open, open, gnd);


jtagproc : process
  begin
wait;
    jtagcom(tdo, tck, tms, tdi, 100, 20, 16#40000000#, true);
    wait;
   end process;

end;
```

5.12 Simulation

DSU communication over the JTAG debug link can be simulated using *jtagcom* procedure. The *jtagcom* procedure sends JTAG commands to the AHBJTAG on JTAG signals TCK, TMS, TDI and TDO. The commands read out and report the device identification code, optionally put the CPU(s) in debug mode, perform three write operations to the memory and read out the data from the memory. The JTAG test works if the generic JTAG tap controller is used and will not work with built-in TAP macros (such as Altera and Xilinx JTAG macros) since these macros don’t have visible JTAG pins. The jtagcom procedure is part of *jtagtst* package in *gaisler* library and has following declaration:

```
procedure jtagcom(signal tdo           : in std_ulogic;
                    signal tck, tms, tdi : out std_ulogic;
                    cp, start, addr      : in integer;
                          -- cp - TCK clock period in ns
                          -- start - time in us when JTAG test is started
                          -- addr - read/write operation destination address
                    haltcpu          : in boolean);
```

6 AHBRAM - Single-port RAM with AHB interface

6.1 Overview

AHBRAM implements on-chip RAM with an AHB slave interface. Memory size is configurable in binary steps through a VHDL generic. Minimum size is 1KiB and maximum size is dependent on target technology and physical resources. Read accesses have zero or one waitstate (configured at implementation time), write access have one waitstate. The RAM supports byte- and half-word accesses, as well as all types of AHB burst accesses.

Internally, the AHBRAM instantiates a SYNCRAM block with byte writes. Depending on the target technology map, this will translate into memory with byte enables or to multiple 8-bit wide SYNCRAM blocks.

The size of the RAM implemented within AHBRAM can be read via the core's AMBA plug&play version field. The version field will display log2(number of bytes), for a 1 KiB SYNCRAM the version field will have the value 10, where 2^{10} = 1024 bytes = 1 KiB.

6.2 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x00E. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

6.3 Configuration options

Table 41 shows the configuration options of the core (VHDL generics).

Table 41. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave bus index	0 - NAHBSLV-1	0
haddr	The MSB address of the AHB area. Sets the 12 most significant bits in the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
hmask	The AHB area address mask. Sets the size of the AHB area and the start address together with *haddr.*	0 - 16#FFF#	16#FF0#
tech	Technology to implement on-chip RAM	0 - NTECH	0
kbytes	RAM size in KiB. The size of the RAM implemented will be the minumum size that will hold the size specified by *kbytes*. A value of 1 here will instantiate a 1 KiB SYNCRAM, a value of 3 will instantiate a 4 KiB SYNCRAM. The actual RAM usage on the target technology then depends on the available RAM resources and the technology map.	target-dependent	1
pipe	Add registers on data outputs. If set to 0 the AMBA data outputs will be connected directly to the core's internal RAM. If set to 1 the core will include registers on the data outputs. Settings this generic to 1 makes read accesses have one waitstate, otherwise the core will respond to read accesses with zero waitstates.	0 - 1	0
maccsz	Maximum access size supported. This generic restricts the maximum AMBA access size supported by the core and selects the width of the SYNCRAMBW RAM used internally. The default value is assigned from AHBDW, which sets the maximum bus width for the GRLIB design.	32, 64, 128, 256	AHBDW

6.4 Signal descriptions

Table 42 shows the interface signals of the core (VHDL ports).

Table 42. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AMB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User’s Manual

6.5 Library dependencies

Table 43 shows libraries used when instantiating the core (VHDL libraries).

Table 43. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	MISC	Component	Component declaration

6.6 Component declaration

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

component ahbram
generic ( hindex : integer := 0; haddr : integer := 0; hmask : integer := 16#fff#;
tech : integer := 0; kbytes : integer := 1);
port (
rst : in std_ulogic;
clk : in std_ulogic;
ahbsi : in ahb_slv_in_type;
ahbso : out ahb_slv_out_type
);
end component;
```

6.7 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

.
.

ahbram0 : ahbram generic map (hindex => 7, haddr => CFG_AHBRADDR,
tech => CFG_MEMTECH, kbytes => 8)
    port map ( rstn, clkm, ahbsi, ahbso(7));
```

7 AHBDPRAM - Dual-port RAM with AHB interface

7.1 Overview

AHBDPRAM implements a 32-bit wide on-chip RAM with one AHB slave interface port and one back-end port for a user application. The AHBDPRAM is therefore useful as a buffer memory between the AHB bus and a custom IP core with a RAM interface

The memory size is configurable in binary steps through the *abits* VHDL generic. The minimum size is 1kB while maximum size is dependent on target technology and physical resources. Read accesses are zero-waitstate, write access have one waitstate. The RAM optionally supports byte- and half-word accesses, as well as all types of AHB burst accesses. Internally, the AHBRAM instantiates one 32-bit or four 8-bit wide SYNCRAM_DP blocks. The target technology must have support for dual-port RAM cells.

The back-end port consists of separate clock, address, datain, dataout, enable and write signals. All these signals are sampled on the rising edge of the back-end clock (CLKDP), implementing a synchronous RAM interface. Read-write collisions between the AHB port and the back-end port are not handled and must be prevented by the user. If byte write is enabled, the WRITE(0:3) signal controls the writing of each byte lane in big-endian fashion. WRITE(0) controls the writing of DATAIN(31:24) and so on. If byte write is disabled, WRITE(0) controls writing to the complete 32-bit word.

7.2 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x00F. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

7.3 Configuration options

Table 44 shows the configuration options of the core (VHDL generics).

Table 44. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave bus index	0 - NAHBSLV-1	0
haddr	The MSB address of the AHB area. Sets the 12 most significant bits in the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
hmask	The AHB area address mask. Sets the size of the AHB area and the start address together with *haddr.*	0 - 16#FFF#	16#FF0#
tech	Technology to implement on-chip RAM	0 - NTECH	2
abits	Address bits. The RAM size in Kbytes is equal to 2**(abits +2)	8 - 19	8
bytewrite	If set to 1, enabled support for byte and half-word writes	0 - 1	0

7.4 Signal descriptions

Table 45 shows the interface signals of the core (VHDL ports).

Table 45. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB Reset	Low
CLK	N/A	Input	AHB Clock	-
AHBSI	*	Input	AMB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
CLKDP		Input	Clock for back-end port	-
ADDRESS(abits-1:0)		Input	Address for back-end port	-
DATAIN(31 : 0)		Input	Write data for back-end port	-
DATAOUT(31 : 0)		Output	Read data from back-end port	-
ENABLE		Input	Chip select for back-end port	High
WRITE(0 : 3)		Input	Write-enable byte select for back-end port	High

* see GRLIB IP Library User's Manual

7.5 Library dependencies

Table 46 shows libraries used when instantiating the core (VHDL libraries).

Table 46. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	MISC	Component	Component declaration

7.6 Component declaration

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

component ahbdpram
  generic (
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#fff#;
    tech    : integer := 2;
    abits   : integer range 8 to 19 := 8;
    bytewrite : integer range 0 to 1 := 0
  );
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;
    clkdp   : in std_ulogic;
    address : in std_logic_vector((abits -1) downto 0);
    datain  : in std_logic_vector(31 downto 0);
    dataout : out std_logic_vector(31 downto 0);
    enable  : in std_ulogic;-- active high chip select
    write  : in std_logic_vector(0 to 3)-- active high byte write enable
  );
  end component;
```

8 AHBROM - Single-port ROM with AHB interface

8.1 Overview

The AHBROM core implements a 32-bit wide on-chip ROM with an AHB slave interface. Read accesses take zero waitstates, or one waitstate if the pipeline option is enabled. The ROM supports byte- and half-word accesses, as well as all types of AHB burst accesses.

8.2 PROM generation

The AHBPROM is automatically generated by the make utility in GRLIB. The input format is a sparc-elf binary file, produced by the BCC cross-compiler (sparc-elf-gcc). To create a PROM, first compile a suitable binary and the run the make utility:

```
bash$ sparc-elf-gcc prom.S -o prom.exe
bash$ make ahbrom.vhd

Creating ahbrom.vhd : file size 272 bytes, address bits 9
```

The default binary file for creating a PROM is prom.exe. To use a different file, run make with the FILE parameter set to the input file:

```
bash$ make ahbrom.vhd FILE=myfile.exe
```

The created PROM is realized in synthesizable VHDL code, using a CASE statement. For FPGA targets, most synthesis tools will map the CASE statement on a block RAM/ROM if available. For ASIC implementations, the ROM will be synthesized as gates. It is then recommended to use the *pipe* option to improve the timing.

8.3 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x01B. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

8.4 Configuration options

Table 47 shows the configuration options of the core (VHDL generics).

Table 47. Configuration options

Generic	**Function**	**Allowed range**	**Default**
hindex	AHB slave bus index	0 - NAHBSLV-1	0
haddr	The MSB address of the AHB area. Sets the 12 most significant bits in the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
hmask	The AHB area address mask. Sets the size of the AHB area and the start address together with *haddr.*	0 - 16#FFF#	16#FF0#
tech	Not used		
pipe	Add a pipeline stage on read data	0	0
kbytes	Not used		

8.5 Signal descriptions

Table 48 shows the interface signals of the core (VHDL ports).

Table 48. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AMB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User's Manual

8.6 Library dependencies

Table 49 shows libraries used when instantiating the core (VHDL libraries).

Table 49. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions

8.7 Component declaration

```
component ahbrom
generic ( hindex : integer := 0; haddr : integer := 0; hmask : integer := 16#fff#;
pipe : integer := 0; tech : integer := 0);
port (
rst : in std_ulogic;
clk : in std_ulogic;
ahbsi : in ahb_slv_in_type;
ahbso : out ahb_slv_out_type
);
end component;
```

8.8 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
.
.

brom : entity work.ahbrom
      generic map (hindex => 8, haddr => CFG_AHBRODDR, pipe => CFG_AHBROPIP)
      port map ( rstn, clkm, ahbsi, ahbso(8));
```

9 AHBSTAT - AHB Status Registers

9.1 Overview

The status registers store information about AMBA AHB accesses triggering an error response. There is a status register and a failing address register capturing the control and address signal values of a failing AMBA bus transaction, or the occurence of a correctable error being signaled from a fault tolerant core.

The status register and the failing address register are accessed from the AMBA APB bus.

9.2 Operation

9.2.1 Errors

The registers monitor AMBA AHB bus transactions and store the current HADDR, HWRITE, HMASTER and HSIZE internally. The monitoring are always active after startup and reset until an error response (HRESP = “01”) is detected. When the error is detected, the status and address register contents are frozen and the New Error (NE) bit is set to one. At the same time an interrupt is generated, as described hereunder.

Note that many of the fault tolerant units containing EDAC signal an un-correctable error as an AMBA error response, so that it can be detected by the processor as described above.

9.2.2 Correctable errors

Not only error responses on the AHB bus can be detected. Many of the fault tolerant units containing EDAC have a correctable error signal which is asserted each time a correctable error is detected. When such an error is detected, the effect will be the same as for an AHB error response. The only difference is that the Correctable Error (CE) bit in the status register is set to one when a correctable error is detected.

When the CE bit is set the interrupt routine can acquire the address containing the correctable error from the failing address register and correct it. When it is finished it resets the CE bit and the monitoring becomes active again. Interrupt handling is described in detail hereunder.

The correctable error signals from the fault tolerant units should be connected to the *stati.cerror* input signal vector of the AHB status register core, which is or-ed internally and if the resulting signal is asserted, it will have the same effect as an AHB error response.

9.2.3 Interrupts

The interrupt is generated on the line selected by the *pirq* VHDL generic.

The interrupt is connected to the interrupt controller to inform the processor of the error condition. The normal procedure is that an interrupt routine handles the error with the aid of the information in the status registers. When it is finished it resets the NE bit and the monitoring becomes active again. Interrupts are generated for both AMBA error responses and correctable errors as described above.

9.3 Registers

The core is programmed through registers mapped into APB address space.

Table 50. AHB Status registers

APB address offset	**Registers**
0x0	AHB Status register
0x4	AHB Failing address register

Table 51. AHB Status register

31 … 10	9	8	7	6 … 3	2 … 0
RESERVED	CE	NE	HWRITE	HMASTER	HSIZE

31: 10	RESERVED
9	CE: Correctable Error. Set if the detected error was caused by a correctable error and zero otherwise.
8	NE: New Error. Deasserted at start-up and after reset. Asserted when an error is detected. Reset by writing a zero to it.
7	The HWRITE signal of the AHB transaction that caused the error.
6: 3	The HMASTER signal of the AHB transaction that caused the error.
2: 0	The HSIZE signal of the AHB transaction that caused the error

Table 52. AHB Failing address register

31 … 0
AHB FAILING ADDRESS

31: 0	The HADDR signal of the AHB transaction that caused the error.

9.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x052. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

9.5 Configuration options

Table 53 shows the configuration options of the core (VHDL generics).

Table 53. Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAHBSLV-1	0
paddr	APB address	0 - 16#FFF#	0
pmask	APB address mask	0 - 16#FFF#	16#FFF#
pirq	Interrupt line driven by the core	0 - 16#FFF#	0
nftslv	Number of FT slaves connected to the cerror vector	1 - NAHBSLV-1	3

9.6 Signal descriptions

Table 54 shows the interface signals of the core (VHDL ports).

Table 54. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB slave input signals	-
AHBSI	*	Input	AHB slave output signals	-
STATI	CERROR	Input	Correctable Error Signals	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-

* see GRLIB IP Library User’s Manual

9.7 Library dependencies

Table 55 shows libraries used when instantiating the core (VHDL libraries).

Table 55. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MISC	Component	Component declaration

9.8 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the status register. There are three Fault Tolerant units with EDAC connected to the status register *cerror* vector. The connection of the different memory controllers to external memory is not shown.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;
  --other signals
  ....
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo, sdo2: sdctrl_out_type;

  signal sdi : sdctrl_in_type;

-- correctable error vector
  signal stati : ahbstat_in_type;
  signal aramo : ahbram_out_type;

begin

  -- AMBA Components are defined here ...

-- AHB Status Register
  astat0 : ahbstat generic map(pindex => 13, paddr => 13, pirq => 11,
    nftslv => 3)
    port map(rstn, clkm, ahbmi, ahbsi, stati, apbi, apbo(13));
    stati.cerror(3 to NAHBSLV-1) <= (others => ‘0’);

--FT AHB RAM
  a0 : ftahbram generic map(hindex => 1, haddr => 1, tech => inferred,
    kbytes => 64, pindex => 4, paddr => 4, edacen => 1, autoscrub => 0,
    errcnt => 1, cntbits => 4)
    port map(rst, clk, ahbsi, ahbso, apbi, apbo(4), aramo);
    stati.cerror(0) <= aramo.ce;
-- SDRAM controller
  sdc : ftsdctrl generic map (hindex => 3, haddr => 16#600#, hmask => 16#F00#,
    ioaddr => 1, fast => 0, pwron => 1, invclk => 0, edacen => 1, errcnt => 1,
    cntbits => 4)
    port map (rstn, clk, ahbsi, ahbso(3), sdi, sdo);
    stati.cerror(1) <= sdo.ce;

-- Memory controller
  mctrl0 : ftsrctrl generic map (rmw => 1, pindex => 10, paddr => 10,
    edacen => 1, errcnt => 1, cntbits => 4)
    port map (rstn, clk, ahbsi, ahbso(0), apbi, apbo(10), memi, memo, sdo2);
    stati.cerror(2) <= memo.ce;
end;
```

10 AHBTRACE - AHB Trace buffer

10.1 Overview

The trace buffer consists of a circular buffer that stores AMBA AHB data transfers. The address, data and various control signals of the AHB bus are stored and can be read out for later analysis.

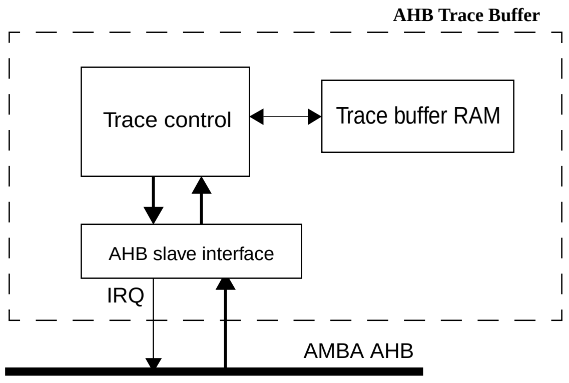


Figure 7. Block diagram

The trace buffer is 128 bits wide, the information stored is indicated in the table below:

Table 56. AHB Trace buffer data allocation

Bits	**Name**	**Definition**
127:96	Time tag	The value of the time tag counter
95	AHB breakpoint hit	Set to ‘1’ if a DSU AHB breakpoint hit occurred.
94:80	Hirq	AHB HIRQ[15:1]
79	Hwrite	AHB HWRITE
78:77	Htrans	AHB HTRANS
76:74	Hsize	AHB HSIZE
73:71	Hburst	AHB HBURST
70:67	Hmaster	AHB HMASTER
66	Hmastlock	AHB HMASTLOCK
65:64	Hresp	AHB HRESP
63:32	Load/Store data	AHB HRDATA or HWDATA
31:0	Load/Store address	AHB HADDR

In addition to the AHB signals, a 32-bit counter is also stored in the trace as time tag.

10.2 Operation

The trace buffer is enabled by setting the enable bit (EN) in the trace control register. Each AMBA AHB transfer is then stored in the buffer in a circular manner. The address to which the next transfer is written is held in the trace buffer index register, and is automatically incremented after each transfer. Tracing is stopped when the EN bit is reset, or when a AHB breakpoint is hit. An interrupt is generated when a breakpoint is hit.

Note: the LEON3 and LEON4 Debug Support Units (DSU3/DSU4) also includes an AHB trace buffer. The standalone trace buffer is intended to be used in system without a processor or when the DSU3 is not present.

The size of the trace buffer is configured by means of the *kbytes* VHDL generic, defining the size of the complete buffer in kbytes.

The size of the trace buffer is *kbytes* kbyte, with the resulting line depth of *kbytes*/16 kbyte.

10.3 Registers

10.3.1 Register address map

The trace buffer occupies 128 KiB of address space in the AHB I/O area. The following register address are decoded:

Table 57. Trace buffer address space

Address	**Register**
0x000000	Trace buffer control register
0x000004	Trace buffer index register
0x000008	Time tag counter
0x00000C	Trace buffer master/slave filter register
0x000010	AHB break address 1
0x000014	AHB mask 1
0x000018	AHB break address 2
0x00001C	AHB mask 2
0x010000 - 0x020000	Trace buffer
..0	Trace bits 127 - 96
...4	Trace bits 95 - 64
...8	Trace bits 63 - 32
...C	Trace bits 31 - 0

10.3.2 Trace buffer control register

The trace buffer is controlled by the trace buffer control register:

Table 58. Trace buffer control register

31 16	15	14 12	11 6	5	4	3	2	1	0
DCNT	BA	BSEL	RESERVED	RF	AF	FR	FW	DM	EN

31: 16	Trace buffer delay counter (DCNT) - Note that the number of bits actually implemented depends on the size of the trace buffer.
15	Bus select Available (BA) - If this field is set to ‘1’, the core has several buses connected. The bus to trace is selected via the BSEL field. If this field is ‘0’, the core is only capable of tracing one AHB bus.
14: 12	Bus select (BSEL) - If the BA field is ‘1’ this field selects the bus to trace. If the BA field is ‘0’, this field is not writable.
11: 6	RESERVED
5	Retry filter (RF) - If this bit is set to ‘1’, AHB retry responses will not be included in the trace buffer. This bit can only be set of the core has been implemented with support for filtering
4	Address Filter (AF) - If this bit is set to ‘1’, only the address range defined by AHB trace buffer breakpoint 2’s address and mask will be included in the trace buffer. This bit can only be set of the core has been implemented with support for filtering
3	Filter Reads (FR) - If this bit is set to ‘1’, read accesses will not be included in the trace buffer. This bit can only be set of the core has been implemented with support for filtering.
2	Filter Writes (FW) - If this bit is set to ‘1’, write accesses will not be included in the trace buffer. This bit can only be set of the core has been implemented with support for filtering.

Table 58. Trace buffer control register

1	Delay counter mode (DM) - Indicates that the trace buffer is in delay counter mode.
0	Trace enable (EN) - Enables the trace buffer

10.3.3 Trace buffer index register

The trace buffer index register indicates the address of the next 128-bit line to be written.

Table 59. Trace buffer index register

31 … 4	3 … 0
INDEX	0x0

31: 4	Trace buffer index counter (INDEX). Note that the number of bits actually implemented depends on the size of the trace buffer
3: 0	Read as 0x0

10.3.4 Trace buffer time tag register

The time tag register contains a 32-bit counter that increments each clock when the trace buffer is enabled. The value of the counter is stored in the trace to provide a time tag.

Table 60. Trace buffer time tag counter

31 … 0
TIME TAG VALUE

10.3.5 Trace buffer master/slave filter register

The master/slave filter register allows filtering out specified master and slaves from the trace. This register can only be assigned if the trace buffer has been implemented with support for filtering.

Table 61. Trace buffer master/slave filter register

31 … 16	15 … 0
SMASK[15:0]	MMASK[15:0]

31: 16	Slave Mask (SMASK) - If SMASK[n] is set to ‘1’, the trace buffer will not save accesses performed to slave n.
15: 0	Master Mask (MMASK) - If MMASK[n] is set to ‘1’, the trace buffer will not save accesses performed by master n.

10.3.6 Trace buffer breakpoint registers

The DSU contains two breakpoint registers for matching AHB addresses. A breakpoint hit is used to freeze the trace buffer by clearing the enable bit. Freezing can be delayed by programming the DCNT field in the trace buffer control register to a non-zero value. In this case, the DCNT value will be decremented for each additional trace until it reaches zero and after two additional entries, the trace buffer is frozen. A mask register is associated with each breakpoint, allowing breaking on a block of addresses. Only address bits with the corresponding mask bit set to ‘1’ are compared during breakpoint detection. To break on AHB load or store accesses, the LD and/or ST bits should be set.

Table 62. Trace buffer AHB breakpoint address register

31 … 2	1 0
BADDR[31:2]	0b00

31: 2	Breakpoint address (BADDR) - Bits 31:2 of breakpoint address
1: 0	Reserved, read as 0

Table 63. Trace buffer AHB breakpoint mask register

31 ... 2	1	0
BMASK[31:2]	LD	ST

31: 2	Breakpoint mask (BMASK) - Bits 31:2 of breakpoint mask
1	Load (LD) - Break on data load address
0	Store (ST) - Break on data store address

10.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x017. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

10.5 Configuration options

Table 64 shows the configuration options of the core (VHDL generics).

Table 64. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave bus index	0 - NAHBSLV-1	0
ioaddr	The MSB address of the I/O area. Sets the 12 most significant bits in the 20-bit I/O address.	0 - 16#FFF#	16#000#
iomask	The I/O area address mask. Sets the size of the I/O area and the start address together with ioaddr.	0 - 16#FFF#	16#E00#
irq	Interrupt number	0 - NAHBIRQ-1	0
tech	Technology to implement on-chip RAM	0 - NTECH	0
kbytes	Trace buffer size in kbytes	1 - 64	1
ahbfilt	If this generic is set to 1 the core will be implemented with support for AHB trace buffer filters.	0 - 1	0
ntrace	Number of buses to trace. This generic is only available if the entity ahbtrace_mmb is instantiated.	1 - 8	1

10.6 Signal descriptions

Table 65 shows the interface signals of the core (VHDL ports).

Table 65. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB master input signals	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User’s Manual

10.7 Library dependencies

Table 66 shows libraries used when instantiating the core (VHDL libraries).

Table 66. Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	MISC	Component	Component declaration

10.8 Component declaration

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

component ahbtrace is
  generic (
    hindex  : integer := 0;
    ioaddr  : integer := 16#000#;
    iomask  : integer := 16#E00#;
    tech    : integer := 0;
    irq     : integer := 0;
    kbytes  : integer := 1);
  port (
    rst    : in  std_ulogic;
    clk    : in  std_ulogic;
    ahbmi  : in  ahb_mst_in_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : out ahb_slv_out_type);
end component;

-- Tracebuffer that can trace separate bus:
component ahbtrace_mb is
  generic (
    hindex  : integer := 0;
    ioaddr    : integer := 16#000#;
    iomask    : integer := 16#E00#;
    tech    : integer := DEFMEMTECH;
    irq     : integer := 0;
    kbytes  : integer := 1);
  port (
    rst    : in  std_ulogic; clk    : in  std_ulogic;
    ahbsi  : in  ahb_slv_in_type;       -- Register interface
    ahbso  : out ahb_slv_out_type;
    tahbmi : in  ahb_mst_in_type; tahbsi : in  ahb_slv_in_type -- Trace
  );
  end component;

-- Tracebuffer that can trace several separate buses:
component ahbtrace_mmb is
  generic (
    hindex  : integer := 0;
    ioaddr    : integer := 16#000#;
    iomask    : integer := 16#E00#;
    tech    : integer := DEFMEMTECH;
    irq     : integer := 0;
    kbytes  : integer := 1;
    ntrace  : integer range 1 to 8 := 1);
  port (
    rst     : in  std_ulogic; clk    : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;       -- Register interface
    ahbso   : out ahb_slv_out_type;
    tahbmiv : in  ahb_mst_in_vector_type(0 to ntrace-1);
    tahbsiv : in  ahb_slv_in_vector_type(0 to ntrace-1) -- Trace
  );
  end component;
```

11 AHBUART- AMBA AHB Serial Debug Interface

11.1 Overview

The interface consists of a UART connected to the AMBA AHB bus as a master. A simple communication protocol is supported to transmit access parameters and data. Through the communication link, a read or write transfer can be generated to any address on the AMBA AHB bus.

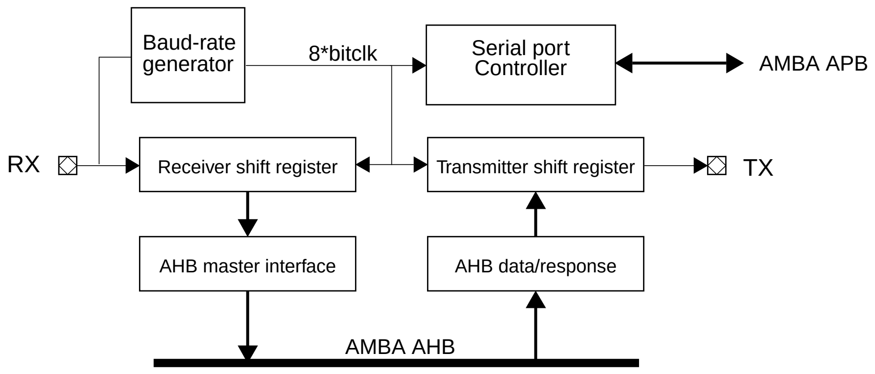


Figure 8. **Block diagram**

11.2 Operation

11.2.1 Transmission protocol

The interface supports a simple protocol where commands consist of a control byte, followed by a 32-bit address, followed by optional write data. Write access does not return any response, while a read access only returns the read data. Data is sent on 8-bit basis as shown below.

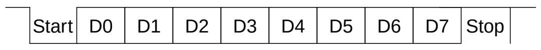


Figure 9. **Data frame**

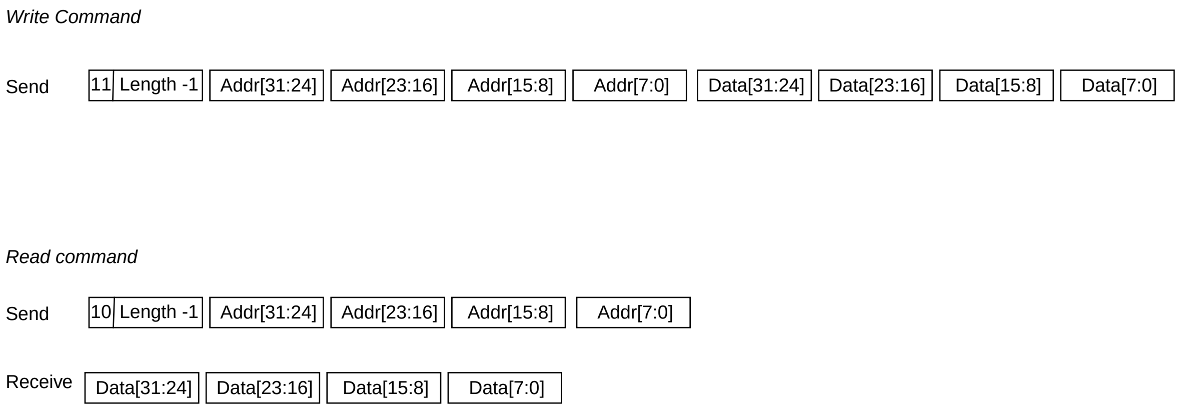


Figure 10. **Commands**

Block transfers can be performed be setting the length field to n-1, where n denotes the number of transferred words. For write accesses, the control byte and address is sent once, followed by the number of data words to be written. The address is automatically incremented after each data word. For read accesses, the control byte and address is sent once and the corresponding number of data words is returned.

11.2.2 Baud rate generation

The UART contains a 18-bit down-counting scaler to generate the desired baud-rate. The scaler is clocked by the system clock and generates a UART tick each time it underflows. The scaler is reloaded with the value of the UART scaler reload register after each underflow. The resulting UART tick frequency should be 8 times the desired baud-rate.

If not programmed by software, the baud rate will be automatically discovered. This is done by searching for the shortest period between two falling edges of the received data (corresponding to two bit periods). When three identical two-bit periods has been found, the corresponding scaler reload value is latched into the reload register, and the BL bit is set in the UART control register. If the BL bit is reset by software, the baud rate discovery process is restarted. The baud-rate discovery is also restarted when a ‘break’ or framing error is detected by the receiver, allowing to change to baudrate from the external transmitter. For proper baudrate detection, the value 0x55 should be transmitted to the receiver after reset or after sending break.

The best scaler value for manually programming the baudrate can be calculated as follows:

```
scaler = (((system_clk*10)/(baudrate*8))-5)/10
```

11.3 Registers

The core is programmed through registers mapped into APB address space.

Table 67. AHB UART registers

APB address offset	**Register**
0x4	AHB UART status register
0x8	AHB UART control register
0xC	AHB UART scaler register

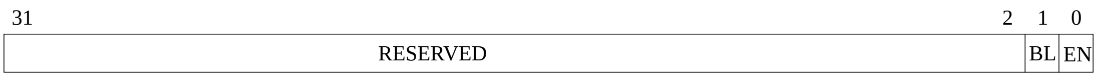


Figure 11. **AHB UART control register**

0: Receiver enable (EN) - if set, enables both the transmitter and receiver. Reset value: ‘0’.
1: Baud rate locked (BL) - is automatically set when the baud rate is locked. Reset value: ‘0’.

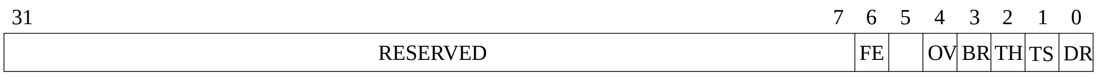


Figure 12. **AHB UART status register**

0: Data ready (DR) - indicates that new data has been received by the AMBA AHB master interface. Read only. Reset value: ‘0’.
1: Transmitter shift register empty (TS) - indicates that the transmitter shift register is empty. Read only. Reset value: ‘1’
2: Transmitter hold register empty (TH) - indicates that the transmitter hold register is empty. Read only. Reset value: ‘1’
3: Break (BR) - indicates that a BREAKE has been received. Reset value: ‘0’
4: Overflow (OV) - indicates that one or more character have been lost due to receiver overflow. Reset value: ‘0’
6: Frame error (FE) - indicates that a framing error was detected. Reset value: ‘0’

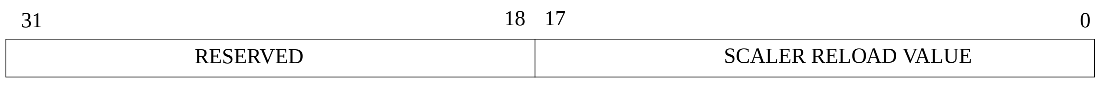


Figure 13. **AHB UART scaler reload register**

17:0 Baudrate scaler reload value = (((system_clk*10)/(baudrate*8))-5)/10. Reset value: “3FFFF“.

11.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x007. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

11.5 Configuration options

Table 68 shows the configuration options of the core (VHDL generics).

Table 68. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#

11.6 Signal descriptions

Table 69 shows the interface signals of the core (VHDL ports)..

Table 69. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
UARTI	RXD	Input	UART receiver data	High
	CTSN	Input	UART clear-to-send	High
	EXTCLK	Input	Use as alternative UART clock	-
UARTO	RTSN	Output	UART request-to-send	High
	TXD	Output	UART transmit data	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-

* see GRLIB IP Library User’s Manual

11.7 Library dependencies

Table 70 shows libraries used when instantiating the core (VHDL libraries).

Table 70. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	UART	Signals, component	Signals and component declaration

11.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.uart.all;

entity ahbuart_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- UART signals
    ahbrxd   : in  std_ulogic;
    ahbtxd   : out std_ulogic
    );
end;

architecture rtl of ahbuart_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- UART signals
  signal ahbuarti : uart_in_type;
  signal ahbuarto : uart_out_type;

begin

  -- AMBA Components are instantiated here
  ...


  -- AHB UART
  ahbuart0 : ahbuart
  generic map (hindex => 5, pindex => 7, paddr => 7)
  port map (rstn, clk, ahbuarti, ahbuarto, apbi, apbo(7), ahbmi, ahbmo(5));

  -- AHB UART input data
  ahbuarti.rxd <= ahbrxd;

  -- connect AHB UART output to entity output signal
  ahbtxd <= ahbuarto.txd;

end;
```

12 AMBAMON - AMBA Bus Monitor

12.1 Overview

The AMBA bus monitor checks the AHB and APB buses for violations against a set of rules. When an error is detected a signal is asserted and error message is (optionally) printed.

12.2 Rules

This section lists all rules checked by the AMBA monitor. The rules are divided into four different tables depending on which type of device they apply to.

Some requirements of the AMBA specification are not adopted by the GRLIB implementation (on a system level). These requirements are listed in the table below.

Table 71. Requirements not checked in GRLIB

Rule Number	Description	References
1	A slave which issues RETRY must only be accessed by one master at a time.	AMBA Spec. Rev 2.0 3-38.

Table 72. AHB master rules.

Rule Number	Description	References
1	Busy can only occur in the middle of bursts. That is only after a NON-SEQ, SEQ or BUSY.	AMBA Spec. Rev 2.0 3-9. http://www.arm.com/support/faqip/492.html
2	Busy can only occur in the middle of bursts. It can be the last access of a burst but only for INCR bursts.	AMBA Spec. Rev 2.0 3-9. http://www.arm.com/support/faqip/492.html
3	The address and control signals must reflect the next transfer in the burst during busy cycles.	AMBA Spec. Rev 2.0 3-9.
4	The first transfer of a single access or a burst must be NONSEQ (this is ensured together with rule 1).	AMBA Spec. Rev 2.0 3-9.
5	HSIZE must never be larger than the bus width.	AMBA Spec. Rev 2.0 3-43.
6	HADDR must be aligned to the transfer size.	AMBA Spec. Rev 2.0 3-12, 3-25. http://www.arm.com/support/faqip/582.html
7	Address and controls signals can only change when hready is low if the previous HTRANS value was IDLE, BUSY or if an ERROR, SPLIT or RETRY response is given.	http://www.arm.com/support/faqip/487.html http://www.arm.com/support/faqip/579.html
8	Address and control signals cannot change between consecutive BUSY cycles.	AMBA Spec. Rev 2.0 3-9.
9	Address must be related to the previous access according to HBURST and HSIZE and control signals must be identical for SEQUENTIAL accesses.	AMBA Spec. Rev 2.0 3-9.
10	Master must cancel the following transfer when receiving an RETRY response.	AMBA Spec. Rev 2.0 3-22.
11	Master must cancel the following transfer when receiving an SPLIT response.	AMBA Spec. Rev 2.0 3-22.

Table 72. AHB master rules.

Rule Number	Description	References
12	Master must reattempt the transfer which received a RETRY response.	AMBA Spec. Rev 2.0 3-21. http://www.arm.com/support/faqip/603.html.
13	Master must reattempt the transfer which received a SPLIT response.	AMBA Spec. Rev 2.0 3-21. http://www.arm.com/support/faqip/603.html.
14	Master can optionally cancel the following transfer when receiving an ERROR response. Only a warning is given if assertions are enabled if it does not cancel the following transfer.	AMBA Spec. Rev 2.0 3-23.
15	Master must hold HWDATA stable for the whole data phase when wait states are inserted. Only the appropriate byte lanes need to be driven for subword transfers.	AMBA Spec. Rev 2.0 3-7. AMBA Spec. Rev 2.0 3-25.
16	Bursts must not cross a 1 kB address boundary.	AMBA Spec. Rev 2.0 3-11.
17	HMASTLOCK indicates that the current transfer is part of a locked sequence. It must have the same timing as address/control.	AMBA Spec. Rev 2.0 3-28.
18	HLOCK must be asserted at least one clock cycle before the address phase to which it refers.	AMBA Spec. Rev 2.0 3-28.
19	HLOCK must be asserted for the duration of a burst and can only be deasserted so that HMASTLOCK is deasserted after the final address phase.	http://www.arm.com/support/faqip/597.html
20	HLOCK must be deasserted in the last address phase of a burst.	http://www.arm.com/support/faqip/588.html
21	HTRANS must be driven to IDLE during reset.	http://www.arm.com/support/faqip/495.html
22	HTRANS can only change from IDLE to NONSEQ or stay IDLE when HREADY is deasserted.	http://www.arm.com/support/faqip/579.html

Table 73. AHB slave rules.

Rule Number	Description	References
1	AHB slave must respond with a zero wait state OKAY response to BUSY cycles in the same way as for IDLE.	AMBA Spec. Rev 2.0 3-9.
2	AHB slave must respond with a zero wait state OKAY response to IDLE.	AMBA Spec. Rev 2.0 3-9.
3	HRESP should be set to ERROR, SPLIT or RETRY only one cycle before HREADY is driven high.	AMBA Spec. Rev 2.0 3-22.
4	Two-cycle ERROR response must be given.	AMBA Spec. Rev 2.0 3-22.
5	Two-cycle SPLIT response must be given.	AMBA Spec. Rev 2.0 3-22.
6	Two-cycle RETRY response must be given.	AMBA Spec. Rev 2.0 3-22.
7	SPLIT complete signalled to master which did not have pending access.	AMBA Spec. Rev 2.0 3-36.
8	Split complete must not be signalled during same cycle as SPLIT.	http://www.arm.com/support/faqip/616.html
9	It is recommended that slaves drive HREADY high and HRESP to OKAY when not selected. A warning will be given if this is not fol-lowed.	http://www.arm.com/support/faqip/476.html

Table 73. AHB slave rules.

Rule Number	**Description**	**References**
10	It is recommended that slaves do not insert more than 16 wait states. If this is violated a warning will be given if assertions are enabled.	AMBA Spec. Rev 2.0 3-20.
11	Slaves should not assert the HSPLIT (Split complete) signal for more than one cycle for each SPLIT response. If a slave asserts HSPLIT for more than one cycle it will not cause the system to malfunction. It can however be a indication that a core does not perform as expected. Therefore assertion of HSPLIT during more than one cycle for a SPLIT response is reported as a warning.	No reference

Table 74. APB slave rules.

Rule Number	**Description**	**References**
1	The bus must move to the SETUP state or remain in the IDLE state when in the IDLE state.	AMBA Spec. Rev 2.0 5-4.
2	The bus must move from SETUP to ENABLE in one cycle.	AMBA Spec. Rev 2.0 5-4.
3	The bus must move from ENABLE to SETUP or IDLE in one cycle.	AMBA Spec. Rev 2.0 5-5.
4	The bus must never be in another state than IDLE, SETUP, ENABLE.	AMBA Spec. Rev 2.0 5-4.
5	PADDR must be stable during transition from SETUP to ENABLE.	AMBA Spec. Rev 2.0 5-5.
6	PWRITE must be stable during transition from SETUP to ENABLE.	AMBA Spec. Rev 2.0 5-5.
7	PWDATA must be stable during transition from SETUP to ENABLE.	AMBA Spec. Rev 2.0 5-5.
8	Only one PSEL must be enabled at a time.	AMBA Spec. Rev 2.0 5-4.
9	PSEL must be stable during transition from SETUP to ENABLE.	AMBA Spec. Rev 2.0 5-5.

Table 75. Arbiter rules

Rule Number	**Description**	**References**
1	HreadyIn to slaves and master must be driven by the currently selected device.	http://www.arm.com/support/faqip/482.html
2	A master which received a SPLIT response must not be granted the bus until the slave has set the corresponding HSPLIT line.	AMBA Spec. Rev 2.0 3-35.
3	The dummy master must be selected when a SPLIT response is received for a locked transfer.	http://www.arm.com/support/faqip/14307.html

12.3 Configuration options

Table 76 shows the configuration options of the core (VHDL generics).

***Table 76.* Configuration options**

Generic	Function	Allowed range	Default
asserterr	Enable assertions for AMBA requirements. Violations are asserted with severity error.	0 - 1	1
assertwarn	Enable assertions for AMBA recommendations. Violations are asserted with severity warning.	0 - 1	1
hmstdisable	Disable AHB master rule check. To disable a master rule check a value is assigned so that the binary representation contains a one at the position corresponding to the rule number, e.g 0x80 disables rule 7.	-	0
hslvdisable	Disable AHB slave tests. Values are assigned as for hmstdisable.	-	0
pslvdisable	Disable APB slave tests. Values are assigned as for hmstdisable.	-	0
arbdisable	Disable Arbiter tests. Values are assigned as for hmstdisable.	-	0
nahbm	Number of AHB masters in the system.	0 - NAHBMST	NAHBMST
nahbs	Number of AHB slaves in the system.	0 - NAHBSLV	NAHBSLV
napb	Number of APB slaves in the system.	0 - NAPBSLV	NAPBSLV
ebterm	Relax rule checks to allow use in systems with early burst termination. This generic should be set to 0 for systems that use GRLIB's AHBCTRL core.	0 - 1	0

12.4 Signal descriptions

Table 77 shows the interface signals of the core (VHDL ports).

Table 77. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
AHBMI	*	Input	AHB master interface input record	-
AHBMO	*	Input	AHB master interface output record array	-
AHBSI	*	Input	AHB slave interface input record	-
AHBSO	*	Input	AHB slave interface output record array	-
APBI	*	Input	APB slave interface input record	
APBO	*	Input	APB slave interface output record array	
ERR	N/A	Output	Error signal (error detected)	High

* see GRLIB IP Library User's Manual

12.5 Library dependencies

Table 78 shows libraries used when instantiating the core (VHDL libraries).

Table 78. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	SIM	Component	Component declaration

12.6 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.sim.all;

entity ambamon_ex is
  port (
    clk : in std_ulogic;
    rst : in std_ulogic
end;

architecture rtl of ambamon_ex is
-- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);



  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);


begin
  -- AMBA Components are instantiated here
   ...
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.sim.all;

entity ambamon_ex is
  port (
    clk : in  std_ulogic;
    rst : in  std_ulogic;
    err : out std_ulogic
end;

architecture rtl of ambamon_ex is
  -- AHB signals
  signal ahbmi  : ahb_mst_in_type;
  signal ahbmo  : ahb_mst_out_vector := (others => apb_none);

  -- AHB signals
  signal ahbsi  : ahb_slv_in_type;
  signal ahbso  : ahb_slv_out_vector := (others => apb_none);

  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);


begin

  mon0 : ambamon
  generic map(
    assert_err => 1,
    assert_war => 0,
    nahbm      => 2,
    nahbs      => 2,
    napb       => 1
  )
  port map(
    rst        => rst,
    clk        => clk,
    ahbmi      => ahbmi,
    ahbmo      => ahbmo,
    ahbsi      => ahbsi,
    ahbso      => ahbso,
    apbi       => apbi,
    apbo       => apbo,
    err        => err);

end;
```

13 APBCTRL - AMBA AHB/APB bridge with plug&play support

13.1 Overview

The AMBA AHB/APB bridge is a APB bus master according the AMBA 2.0 standard.

The controller supports up to 16 slaves. The actual maximum number of slaves is defined in the GRLIB.AMBA package, in the VHDL constant NAPBSLV. The number of slaves can also be set using the *nslaves* VHDL generic.

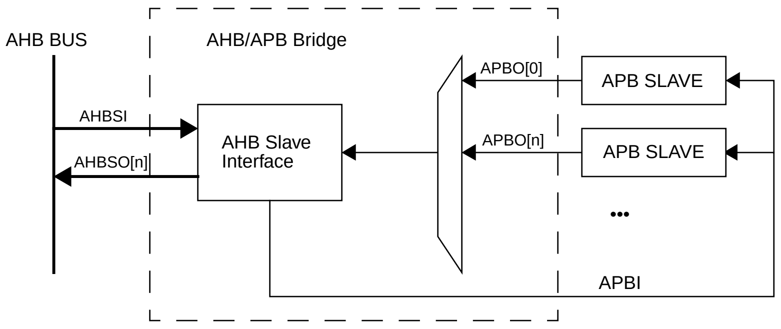


Figure 14. AHB/APB bridge block diagram

13.2 Operation

13.2.1 Decoding

Decoding (generation of PSEL) of APB slaves is done using the plug&play method explained in the GRLIB IP Library User’s Manual. A slave can occupy any binary aligned address space with a size of 256 bytes - 1 Mbyte. Writes to unassigned areas will be ignored, while reads from unassigned areas will return an arbitrary value. AHB error response will never be generated.

13.2.2 Plug&play information

GRLIB APB slaves contain two plug&play information words which are included in the APB records they drive on the bus (see the GRLIB IP Library User’s Manual for more information). These records are combined into an array which is connected to the APB bridge.

The plug&play information is mapped on a read-only address area at the top 4 kbytes of the bridge address space. Each plug&play block occupies 8 bytes. The address of the plug&play information for a certain unit is defined by its bus index. If the bridge is mapped on AHB address 0x80000000, the address for the plug&play records is thus 0x800FF000 + n*8.

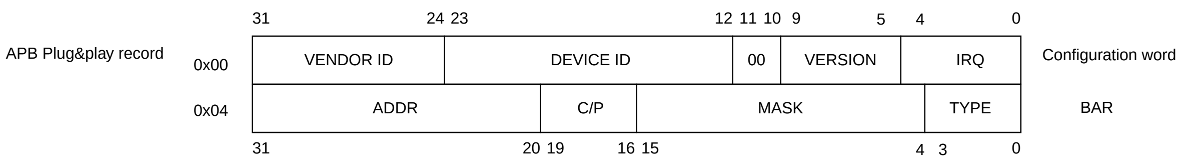


Figure 15. APB plug&play information

13.3 APB bus monitor

An APB bus monitor is integrated into the core. It is enabled with the enbusmon generic. It has the same functionality as the APB parts in the AMBA monitor core (AMBAMON). For more information on which rules are checked se the AMBAMON documentation.

13.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x006. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

13.5 Configuration options

Table 79 shows the configuration options of the core (VHDL generics).

Table 79. Configuration options

Generic	**Function**	**Allowed range**	**Default**
hindex	AHB slave index	0 - NAHBSLV-1	0
haddr	The MSB address of the AHB area. Sets the 12 most significant bits in the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
hmask	The AHB area address mask. Sets the size of the AHB area and the start address together with haddr.	0 - 16#FFF#	16#FFF#
nslaves	The maximum number of slaves	1 - NAPBSLV	NAPBSLV
debug	Print debug information during simulation	0 - 2	2
icheck	Enable bus index checking (PINDEX)	0 - 1	1
enbusmon	Enable APB bus monitor	0 - 1	0
asserterr	Enable assertions for AMBA requirements. Violations are asserted with severity error.	0 - 1	0
assertwarn	Enable assertions for AMBA recommendations. Violations are asserted with severity warning.	0 - 1	0
pslvdisable	Disable APB slave rule check. To disable a slave rule check a value is assigned so that the binary representation contains a one at the position corresponding to the rule number, e.g 0x80 disables rule 7.	N/A	0
mcheck	Check if there are any intersections between APB slave memory areas. If two areas intersect an assert with level failure will be triggered (in simulation).	0 - 1	1
ccheck	Perform sanity checks on PnP configuration records (in simulation).	0 - 1	1

13.6 Signal descriptions

Table 80 shows the interface signals of the core (VHDL ports).

Table 80. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
AHBI	*	Input	AHB slave input	-
AHBO	*	Output	AHB slave output	-
APBI	*	Output	APB slave inputs	-
APBO	*	Input	APB slave outputs	-

* see GRLIB IP Library User's Manual

13.7 Library dependencies

Table 81 shows libraries used when instantiating the core (VHDL libraries).

Table 81. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions

13.8 Component declaration

```
library grlib;
use grlib.amba.all;

component apbctrl
  generic (
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#fff#;
    nslaves : integer range 1 to NAPBSLV := NAPBSLV;
    debug   : integer range 0 to 2 := 2;   -- print config to console
    icheck  : integer range 0 to 1 := 1
  );
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbi    : in  ahb_slv_in_type;
    ahbo    : out ahb_slv_out_type;
    apbi    : out apb_slv_in_type;
    apbo    : in  apb_slv_out_vector
  );
  end component;
```

13.9 Instantiation

This example shows how an APB bridge can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use work.debug.all;

.
.
```

```
  -- AMBA signals

  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);

signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);


begin


-- APB bridge

apb0 : apbctrl-- AHB/APB bridge
  generic map (hindex => 1, haddr => CFG_APBADDR)
  port map (rstn, clk, ahbsi, ahbso(1), apbi, apbo );


-- APB slaves

uart1 : apbuart
    generic map (pindex => 1, paddr => 1,  pirq => 2)
    port map (rstn, clk, apbi, apbo(1), u1i, u1o);

irqctrl0 : irqmp
    generic map (pindex => 2, paddr => 2)
    port map (rstn, clk, apbi, apbo(2), irqo, irqi);


...
end;
```

13.10 Debug print-out

The APB bridge can print-out the plug-play information from the attached during simulation. This is enabled by setting the debug VHDL generic to 2. Reporting starts by scanning the array from 0 to NAPBSLV - 1 (defined in the grlib.amba package). It checks each entry in the array for a valid vendor-id (all nonzero ids are considered valid) and if one is found, it also retrieves the device-id. The description for these ids are obtained from the GRLIB.DEVICES package, and is printed on standard out together with the slave number. If the index check is enabled (done with a VHDL generic), the report module also checks if the pindex number returned in the record matches the array number of the record currently checked (the array index). If they do not match, the simulation is aborted and an error message is printed.

The address range and memory type is also checked and printed. The address information includes type, address and mask. The address ranges currently defined are AHB memory, AHB I/O and APB I/O. All APB devices are in the APB I/O range so the type does not have to be checked. From this information, the report module calculates the start address of the device and the size of the range. The information finally printed is start address and size.

14 APBPS2 - PS/2 host controller with APB interface

14.1 Introduction

The PS/2 interface is a bidirectional synchronous serial bus primarily used for keyboard and mouse communications. The APBPS2 core implements the PS2 protocol with a APB back-end. Figure 16 shows a model of APBPS2 and the electrical interface.

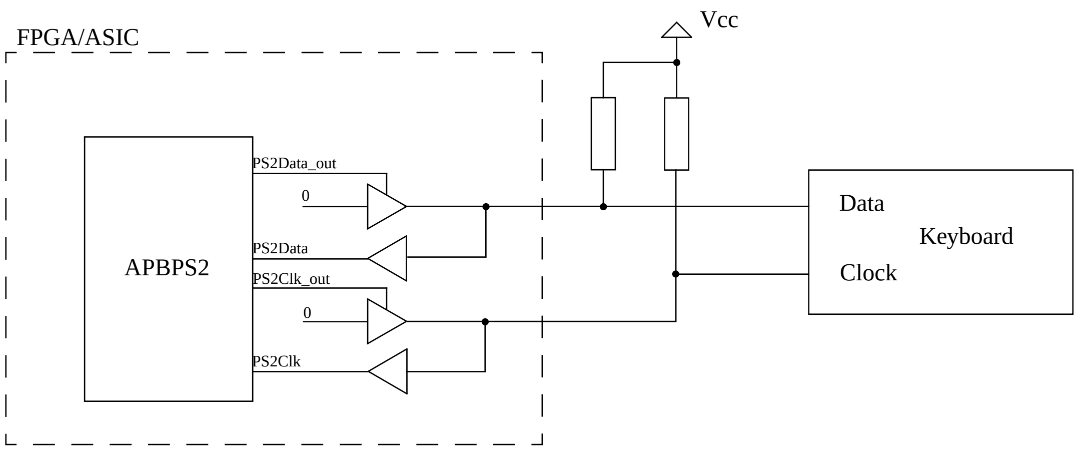


Figure 16. APBPS2 electrical interface

PS/2 data is sent in 11 bits frames. The first bit is a start bit followed by eight data bits, one odd parity bit and finally one stop bit. Figure 17 shows a typical PS/2 data frame.

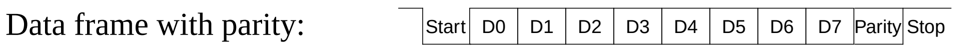


Figure 17. PS/2 data frame

14.2 Receiver operation

The receiver of APBPS2 receives the data from the keyboard or mouse, and converts it to 8-bit data frames to be read out via the APB bus. It is enabled through the receiver enable (RE) bit in the PS/2 control register. If a parity error or framing error occurs, the data frame will be discarded. Correctly received data will be transferred to a 16 byte FIFO. The data ready (DR) bit in the PS/2 status register will be set, and retained as long as the FIFO contains at least one data frame. When the FIFO is full, the receiver buffer full (RF) bit in the status register is set. The keyboard will be inhibited and buffer data until the FIFO gets read again. Interrupt is sent when a correct stop bit is received then it’s up to the software to handle any resend operations if the parity bit is wrong. Figure 18 shows a flow chart for the operations of the receiver state machine.

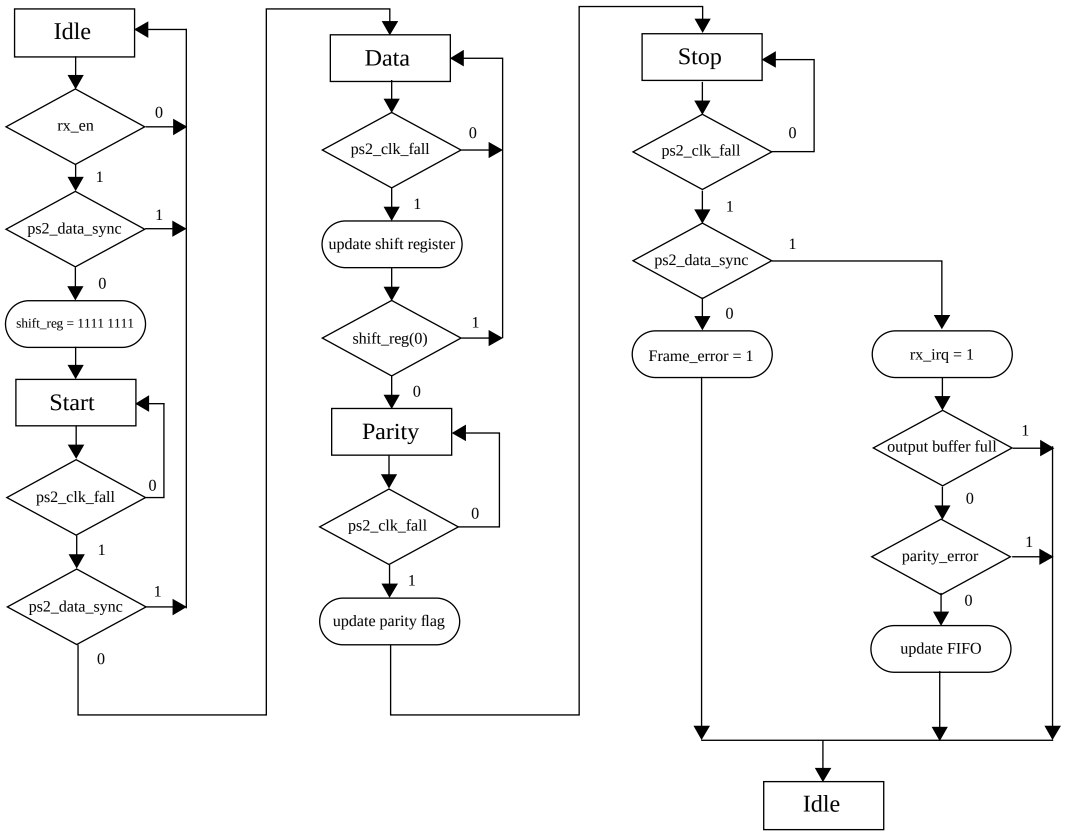


Figure 18. Flow chart for the receiver state machine

14.3 Transmitter operations

The transmitter part of APBPS2 is enabled for through the transmitter enable (TE) bit in the PS/2 control register. The PS/2 interface has a 16 byte transmission FIFO that stores commands sent by the CPU. Commands are used to set the LEDs on the keyboard, and the typematic rate and delay. Typematic rate is the repeat rate of a key that is held down, while the delay controls for how long a key has to be held down before it begins automatically repeating. Typematic repeat rates, delays and possible other commands are listed in table 89.

If the TE bit is set and the transmission FIFO is not empty a transmission of the command will start. The host will pull the clock line low for at least 100 us and then transmit a start bit, the eight bit command, an odd parity bit, a stop bit and wait for an acknowledgement bit by the device. When this happens an interrupt is generated. Figure 19 shows the flow chart for the transmission state machine.

14.4 Clock generation

A PS/2 interface should generate a clock of 10.0 - 16.7 kHz. To transmit data, a PS/2 host must inhibit communication by pulling the clock low for at least 100 microseconds. To do this, APBPS2 divides the APB clock with either a fixed or programmable division factor. The divider consist of a 17-bit down-counter and can divide the APB clock with a factor of 1 - 131071. The division rate, and the reset value of the timer reload register, is set to the *fKHz* generic divided by 10 in order to generate the 100 microsecond clock low time. If the VHDL generic *fixed* is 0, the division rate can be programmed through the timer reload register and should be programmed with the system frequency in kHz divided by ten. The reset value of the reload register is always set to the *fKHz* value divided by ten. However, the register will not be readable via the APB interface unless the *fixed* VHDL generic has been set to 0.

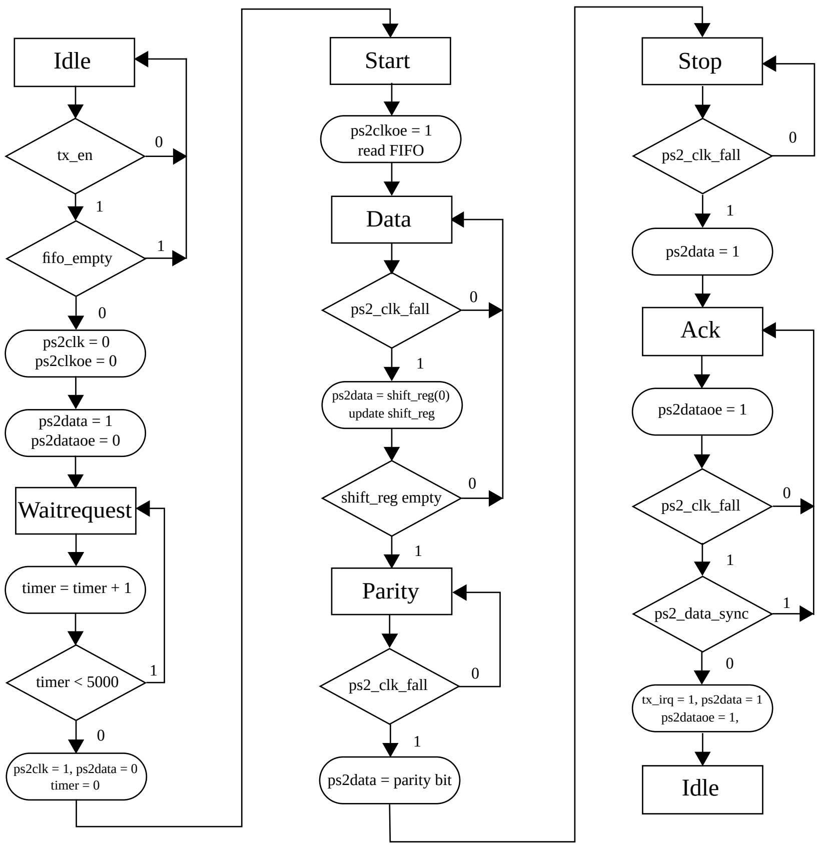


Figure 19. Flow chart for the transmitter state machine

14.5 Registers

The core is controlled through registers mapped into APB address space.

Table 82. APB PS/2 registers

APB address offset	Register
0x00	PS/2 Data register
0x04	PS/2 Status register
0x08	PS/2 Control register
0x0C	PS/2 Timer reload register

14.5.1 PS/2 Data Register

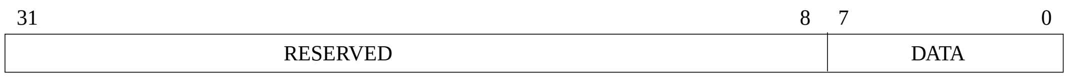


Figure 20. PS/2 data register

[7:0]: Receiver holding FIFO (read access) and Transmitter holding FIFO (write access). If the receiver FIFO is not empty, read accesses retrieve the next byte from the FIFO. Bytes written to this field are stored in the transmitter holding FIFO if it is not full.

14.5.2 PS/2 Status Register

31 27	26 22		5	4	3	2	1	0
RCNT	TCNT	RESERVED	TF	RF	KI	FE	PE	DR

Figure 21. PS/2 status register

0: Data ready (DR) - indicates that new data is available in the receiver holding register (read only).
1: Parity error (PE) - indicates that a parity error was detected.
2: Framing error (FE) - indicates that a framing error was detected.
3: Keyboard inhibit (KI) - indicates that the keyboard is inhibited.
4: Receiver buffer full (RF) - indicates that the output buffer (FIFO) is full (read only).
5: Transmitter buffer full (TF) - indicates that the input buffer (FIFO) is full (read only).
[26:22]: Transmit FIFO count (TCNT) - shows the number of data frames in the transmit FIFO (read only).
[31:27]: Receiver FIFO count (RCNT) - shows the number of data frames in the receiver FIFO (read only).

14.5.3 PS/2 Control Register

31	3	2	1	0
RESERVED	TI	RI	TE	RE

Figure 22. PS/2 control register

0: Receiver enable (RE) - if set, enables the receiver.
1: Transmitter enable (TE) - if set, enables the transmitter.
2: Keyboard interrupt enable (RI) - if set, interrupts are generated when a frame is received
3: Host interrupt enable (TI) - if set, interrupts are generated when a frame is transmitted

14.5.4 PS/2 Timer Reload Register

31 17	16 0
RESERVED	TIMER RELOAD REG

Figure 23. PS/2 timer register

[16:0]: PS/2 timer reload register

14.6 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x060. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

14.7 Configuration options

Table 83 shows the configuration options of the core (VHDL generics).

Table 83. Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Index of the interrupt line.	0 - NAHBIRQ-1	0
fKHz	Frequency of APB clock in KHz. This value divided by 10 is the reset value of the timer reload register.	1 - 1310710	50000
fixed	Used fixed clock divider to generate PS/2 clock.	0 - 1	0
oepol	Output enable polarity	0 - 1	0

14.8 Signal descriptions

Table 84 shows the interface signals of the core (VHDL ports).

Table 84. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
PS2I	PS2_CLK_I	Input	PS/2 clock input	-
	PS2_DATA_I	Input	PS/2 data input	-
PS2O	PS2_CLK_O	Output	PS/2 clock output	-
	PS2_CLK_OE	Output	PS/2 clock output enable	Low
	PS2_DATA_O	Output	PS/2 data output	-
	PS2_DATA_OE	Output	PS/2 data output enable	Low

* see GRLIB IP Library User's Manual

14.9 Library dependencies

Table 85 shows libraries used when instantiating the core (VHDL libraries).

Table 85. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	APB signal definitions
GAISLER	MISC	Signals, component	PS/2 signal and component declaration

14.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
```

```
use grlib.amba.all;
use grlib.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity apbps2_ex is
 port (
    rstn : in std_ulogic;
    clk : in std_ulogic;

    -- PS/2 signals
    ps2clk : inout std_ulogic;
    ps2data : inout std_ulogic
    );
end;

architecture rtl of apbuart_ex is

  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- PS/2 signals
  signal kbdi : ps2_in_type;
  signal kbdo : ps2_out_type;

begin

ps20 : apbps2 generic map(pindex => 5, paddr => 5, pirq => 4)
      port map(rstn, clkm, apbi, apbo(5), kbdi, kbdo);

kbdclk_pad : iopad generic map (tech => padtech)
      port map (ps2clk,kbdo.ps2_clk_o, kbdo.ps2_clk_oe, kbdi.ps2_clk_i);

kbdata_pad : iopad generic map (tech => padtech)
        port map (ps2data, kbdo.ps2_data_o, kbdo.ps2_data_oe, kbdi.ps2_data_i);

end;
```

14.11 Keboard scan codes

Table 86. Scan code set 2, 104-key keyboard

KEY	**MAKE**	**BREAK**	- -	**KEY**	**MAKE**	**BREAK**	- -	**KEY**	**MAKE**	**BREAK**
A	1C	F0,1C		9	46	F0,46		[	54	FO,54
B	32	F0,32			`0E	F0,0E		INSERT	E0,70	E0,F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0,F0,6C
D	23	F0,23		=	55	FO,55		PG UP	E0,7D	E0,F0,7D
E	24	F0,24		\	5D	F0,5D		DELETE	E0,71	E0,F0,71
F	2B	F0,2B		BKSP	66	F0,66		END	E0,69	E0,F0,69
G	34	F0,34		SPACE	29	F0,29		PG DN	E0,7A	E0,F0,7A
H	33	F0,33		TAB	0D	F0,0D		U ARROW	E0,75	E0,F0,75
I	43	F0,43		CAPS	58	F0,58		L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B		L SHFT	12	FO,12		D ARROW	E0,72	E0,F0,72
K	42	F0,42		L CTRL	14	FO,14		R ARROW	E0,74	E0,F0,74
L	4B	F0,4B		L GUI	E0,1F	E0,F0,1F		NUM	77	F0,77
M	3A	F0,3A		L ALT	11	F0,11		KP /	E0,4A	E0,F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7C	F0,7C
O	44	F0,44		R CTRL	E0,14	E0,F0,14		KP -	7B	F0,7B
P	4D	F0,4D		R GUI	E0,27	E0,F0,27		KP +	79	F0,79
Q	15	F0,15		R ALT	E0,11	E0,F0,11		KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D		APPS	E0,2F	E0,F0,2F		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	76	F0,76		KP 1	69	F0,69
U	3C	F0,3C		F1	5	F0,05		KP 2	72	F0,72
V	2A	F0,2A		F2	6	F0,06		KP 3	7A	F0,7A
W	1D	F0,1D		F3	4	F0,04		KP 4	6B	F0,6B
X	22	F0,22		F4	0C	F0,0C		KP 5	73	F0,73
Y	35	F0,35		F5	3	F0,03		KP 6	74	F0,74
Z	1A	F0,1A		F6	0B	F0,0B		KP 7	6C	F0,6C
0	45	F0,45		F7	83	F0,83		KP 8	75	F0,75
1	16	F0,16		F8	0A	F0,0A		KP 9	7D	F0,7D
2	1E	F0,1E		F9	1	F0,01		]	5B	F0,5B
3	26	F0,26		F10	9	F0,09		;	4C	F0,4C
4	25	F0,25		F11	78	F0,78			52	F0,52
5	2E	F0,2E		F12	7	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-				

Table 87. Windows multimedia scan codes

KEY	MAKE	BREAK
Next Track	E0, 4D	E0, F0, 4D
Previous Track	E0, 15	E0, F0, 15
Stop	E0, 3B	E0, F0, 3B
Play/Pause	E0, 34	E0, F0, 34
Mute	E0, 23	E0, F0, 23
Volume Up	E0, 32	E0, F0, 32
Volume Down	E0, 21	E0, F0, 21
Media Select	E0, 50	E0, F0, 50
E-Mail	E0, 48	E0, F0, 48
Calculator	E0, 2B	E0, F0, 2B
My Computer	E0, 40	E0, F0, 40
WWW Search	E0, 10	E0, F0, 10
WWW Home	E0, 3A	E0, F0, 3A
WWW Back	E0, 38	E0, F0, 38
WWW Forward	E0, 30	E0, F0, 30
WWW Stop	E0, 28	E0, F0, 28
WWW Refresh	E0, 20	E0, F0, 20
WWW Favorites	E0, 18	E0, F0, 18

Table 88. ACPI scan codes (Advanced Configuration and Power Interface)

KEY	MAKE	BREAK
Power	E0, 37	E0, F0, 37
Sleep	E0, 3F	E0, F0, 3F
Wake	E0, 5E	E0, F0, 5E

14.12 Keyboard commands

Table 89. Transmit commands:

Command	Description
0xED	Set status LED's - keyboard will reply with ACK (0xFA). The host follows this command with an argument byte*
0xEE	Echo command - expects an echo response
0xF0	Set scan code set - keyboard will reply with ACK (0xFA) and wait for another byte. 0x01-0x03 which determines the scan code set to use. 0x00 returns the current set.
0xF2	Read ID - the keyboard responds by sending a two byte device ID of 0xAB 0x83
0xF3	Set typematic repeat rate - keyboard will reply with ACK (0xFA) and wait for another byte which determines the typematic rate.
0xF4	Keyboard enable - clears the keyboards output buffer, enables keyboard scanning and returns an acknowledgement.
0xF5	Keyboard disable - resets the keyboard, disables keyboard scanning and returns an acknowledge-ment.
0xF6	Set default - load default typematic rate/delay (10.9cps/500ms) and scan code set 2
0xFE	Resend - upon receipt of the resend command the keyboard will retransmit the last byte
0xFF	Reset - resets the keyboard

* bit 0 controls the scroll lock, bit 1 the num lock, bit 2 the caps lock, bit 3-7 are ignored

Table 90. Receive commands:

Command	Description
0xFA	Acknowledge
0xAA	Power on self test passed (BAT completed)
0xEE	Echo respond
0xFE	Resend - upon receipt of the resend command the host should retransmit the last byte
0x00	Error or buffer overflow
0xFF	Error of buffer overflow

Table 91. The typematic rate/delay argument byte

MSB							LSB
0	DELAY	DELAY	RATE	RATE	RATE	RATE	RATE

Table 92. Typematic repeat rates

Bits 0-4	Rate (cps)		Bits 0-4	Rate (cps)		Bits 0-4	Rate (cps)		Bits 0-4	Rate (cps)
00h	30		08h	15		10h	7.5		18h	3.7
01h	26.7		09h	13.3		11h	6.7		19h	3.3
02h	24		0Ah	12		12h	6		1Ah	3
03h	21.8		0Bh	10.9		13h	5.5		1Bh	2.7
04h	20.7		0Ch	10		14h	5		1Ch	2.5
05h	18.5		0Dh	9.2		15h	4.6		1Dh	2.3
06h	17.1		0Eh	8.6		16h	4.3		1Eh	2.1
07h	16		0Fh	8		17h	4		1Fh	2

Table 93. Typematic delays

Bits 5-6	Delay (seconds)
00b	0.25
01b	0.5
10b	0.75
11b	1

15 APBUART - AMBA APB UART Serial Interface

15.1 Overview

The interface is provided for serial communications. The UART supports data frames with 8 data bits, one optional parity bit and one stop bit. To generate the bit-rate, each UART has a programmable 12-bit clock divider. Two FIFOs are used for data transfer between the APB bus and UART, when *fifosize* VHDL generic > 1. Two holding registers are used data transfer between the APB bus and UART, when *fifosize* VHDL generic = 1. Hardware flow-control is supported through the RTSN/CTSN hand-shake signals, when *flow* VHDL generic is set. Parity is supported, when *parity* VHDL generic is set.

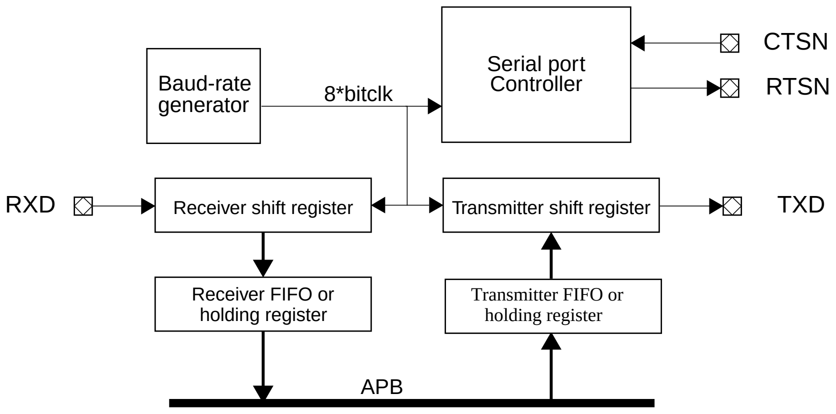

Serial port Controller
CTSN
RTSN
Baud-rate generator
8*bitclk
RXD
Receiver shift register
Transmitter shift register
TXD
Receiver FIFO or holding register
Transmitter FIFO or holding register
APB

Figure 24. Block diagram

15.2 Operation

15.2.1 Transmitter operation

The transmitter is enabled through the TE bit in the UART control register. Data that is to be transferred is stored in the FIFO/holding register by writing to the data register. This FIFO is configurable to different sizes via the *fifosize* VHDL generic. When the size is 1, only a single holding register is used but in the following discussion both will be referred to as FIFOs. When ready to transmit, data is transferred from the transmitter FIFO/holding register to the transmitter shift register and converted to a serial stream on the transmitter serial output pin (TXD). It automatically sends a start bit followed by eight data bits, an optional parity bit, and one stop bit (figure 25). The least significant bit of the data is sent first.

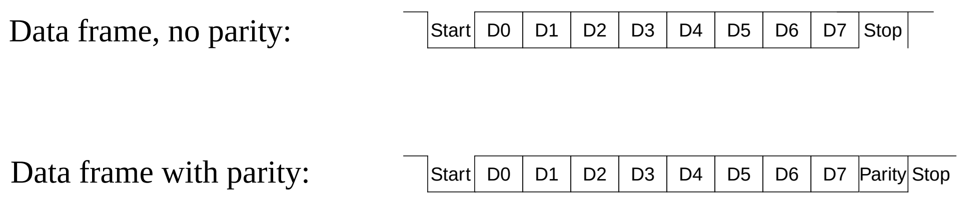


Figure 25. UART data frames

Following the transmission of the stop bit, if a new character is not available in the transmitter FIFO, the transmitter serial data output remains high and the transmitter shift register empty bit (TS) will be set in the UART status register. Transmission resumes and the TS is cleared when a new character is loaded into the transmitter FIFO. When the FIFO is empty the TE bit is set in the status register. If the transmitter is disabled, it will immediately stop any active transmissions including the character currently being shifted out from the transmitter shift register. The transmitter holding register may not be loaded when the transmitter is disabled or when the FIFO (or holding register) is full. If this is done, data might be overwritten and one or more frames are lost.

The discussion above applies to any FIFO configurations including the special case with a holding register (VHDL generic *fifosize* = 1). If FIFOs are used (VHDL generic *fifosize* > 1) some additional status and control bits are available. The TF status bit (not to be confused with the TF control bit) is set if the transmitter FIFO is currently full and the TH bit is set as long as the FIFO is *less* than half-full (less than half of entries in the FIFO contain data). The TF control bit enables FIFO interrupts when set. The status register also contains a counter (TCNT) showing the current number of data entries in the FIFO.

When flow control is enabled, the CTSN input must be low in order for the character to be transmitted. If it is deasserted in the middle of a transmission, the character in the shift register is transmitted and the transmitter serial output then remains inactive until CTSN is asserted again. If the CTSN is connected to a receivers RTSN, overrun can effectively be prevented.

15.2.2 Receiver operation

The receiver is enabled for data reception through the receiver enable (RE) bit in the UART control register. The receiver looks for a high to low transition of a start bit on the receiver serial data input pin. If a transition is detected, the state of the serial input is sampled a half bit clocks later. If the serial input is sampled high the start bit is invalid and the search for a valid start bit continues. If the serial input is still low, a valid start bit is assumed and the receiver continues to sample the serial input at one bit time intervals (at the theoretical centre of the bit) until the proper number of data bits and the parity bit have been assembled and one stop bit has been detected. The serial input is shifted through an 8-bit shift register where all bits have to have the same value before the new value is taken into account, effectively forming a low-pass filter with a cut-off frequency of 1/8 system clock.

The receiver also has a configurable FIFO which is identical to the one in the transmitter. As mentioned in the transmitter part, both the holding register and FIFO will be referred to as FIFO.

During reception, the least significant bit is received first. The data is then transferred to the receiver FIFO and the data ready (DR) bit is set in the UART status register as soon as the FIFO contains at least one data frame. The parity, framing and overrun error bits are set at the received byte boundary, at the same time as the receiver ready bit is set. The data frame is not stored in the FIFO if an error is detected. Also, the new error status bits are or:ed with the old values before they are stored into the status register. Thus, they are not cleared until written to with zeros from the AMBA APB bus. If both the receiver FIFO and shift registers are full when a new start bit is detected, then the character held in the receiver shift register will be lost and the overrun bit will be set in the UART status register. A break received (BR) is indicated when a BREAK has been received, which is a framing error with all data received being zero.

If flow control is enabled, then the RTSN will be negated (high) when a valid start bit is detected and the receiver FIFO is full. When the holding register is read, the RTSN will automatically be reasserted again.

When the VHDL generic *fifosize* > 1, which means that holding registers are not considered here, some additional status and control bits are available. The RF status bit (not to be confused with the RF control bit) is set when the receiver FIFO is full. The RH status bit is set when the receiver FIFO is half-full (at least half of the entries in the FIFO contain data frames). The RF control bit enables receiver FIFO interrupts when set. A RCNT field is also available showing the current number of data frames in the FIFO.

15.3 Baud-rate generation

Each UART contains a 12-bit down-counting scaler to generate the desired baud-rate, the number of scaler bits can be increased with VHDL generic *sbits*. The scaler is clocked by the system clock and generates a UART tick each time it underflows. It is reloaded with the value of the UART scaler reload register after each underflow. The resulting UART tick frequency should be 8 times the desired baud-rate. One appropriate formula to calculate the scaler value for a desired baud rate, using integer division where the remainder is discarded, is:

*scaler value = (system_clock_frequency) / (baud_rate * 8 + 7).*

To calculate the exact required scaler value use:

*scaler value = (system_clock_frequency) / (baud_rate * 8) - 1*

If the EC bit is set, the ticks will be generated with the same frequency as the external clock input instead of at the scaler underflow rate. In this case, the frequency of external clock must be less than half the frequency of the system clock.

15.4 Loop back mode

If the LB bit in the UART control register is set, the UART will be in loop back mode. In this mode, the transmitter output is internally connected to the receiver input and the RTSN is connected to the CTSN. It is then possible to perform loop back tests to verify operation of receiver, transmitter and associated software routines. In this mode, the outputs remain in the inactive state, in order to avoid sending out data.

15.5 FIFO debug mode

FIFO debug mode is entered by setting the debug mode bit in the control register. In this mode it is possible to read the transmitter FIFO and write the receiver FIFO through the FIFO debug register. The transmitter output is held inactive when in debug mode. A write to the receiver FIFO generates an interrupt if receiver interrupts are enabled.

15.6 Interrupt generation

Interrupts are generated differently when a holding register is used (VHDL generic *fifosize* = 1) and when FIFOs are used (VHDL generic *fifosize* > 1). When holding registers are used, the UART will generate an interrupt under the following conditions: when the transmitter is enabled, the transmitter interrupt is enabled and the transmitter holding register moves from full to empty; when the receiver is enabled, the receiver interrupt is enabled and the receiver holding register moves from empty to full; when the receiver is enabled, the receiver interrupt is enabled and a character with either parity, framing or overrun error is received.

For FIFOs, two different kinds of interrupts are available: normal interrupts and FIFO interrupts. For the transmitter, normal interrupts are generated when transmitter interrupts are enabled (TI), the transmitter is enabled and the transmitter FIFO goes from containing data to being empty. FIFO interrupts are generated when the FIFO interrupts are enabled (TF), transmissions are enabled (TE) and the UART is less than half-full (that is, whenever the TH status bit is set). This is a level interrupt and the interrupt signal is continuously driven high as long as the condition prevails. The receiver interrupts work in the same way. Normal interrupts are generated in the same manner as for the holding register. FIFO interrupts are generated when receiver FIFO interrupts are enabled, the receiver is enabled and the FIFO is half-full. The interrupt signal is continuously driven high as long as the receiver FIFO is half-full (at least half of the entries contain data frames).

To reduce interrupt occurrence a delayed receiver interrupt is available. It is enabled using the delayed interrupt enable (DI) bit. When enabled a timer is started each time a character is received and an interrupt is only generated if another character has not been received within 4 character + 4 bit times. If receiver FIFO interrupts are enabled a pending character interrupt will be cleared when the FIFO interrupt is active since the character causing the pending irq state is already in the FIFO and is noticed by the driver through the FIFO interrupt.

There is also a separate interrupt for break characters. When enabled an interrupt will always be generated immediately when a break character is received even when delayed receiver interrupts are enabled. When break interrupts are disabled no interrupt will be generated for break characters when delayed interrupts are enabled.

When delayed interrupts are disabled the behavior is the same for the break interrupt bit except that an interrupt will be generated for break characters if receiver interrupt enable is set even if break interrupt is disabled.

An interrupt can also be enabled for the transmitter shift register. When enabled the core will generate an interrupt each time the shift register goes from a non-empty to an empty state.

15.7 Registers

The core is controlled through registers mapped into APB address space.

Table 94. UART registers

APB address offset	**Register**
0x0	UART Data register
0x4	UART Status register
0x8	UART Control register
0xC	UART Scaler register
0x10	UART FIFO debug register

15.7.1 UART Data Register

Table 95. UART data register

31 … 8	7 … 0
RESERVED	DATA

7: 0	Receiver holding register or FIFO (read access)
7: 0	Transmitter holding register or FIFO (write access)

15.7.2 UART Status Register

Table 96. UART status register

31 … 26	25 … 20	19 … 11	10	9	8	7	6	5	4	3	2	1	0
RCNT	TCNT	RESERVED	RF	TF	RH	TH	FE	PE	OV	BR	TE	TS	DR

31: 26	Receiver FIFO count (RCNT) - shows the number of data frames in the receiver FIFO. Reset: 0
25: 20	Transmitter FIFO count (TCNT) - shows the number of data frames in the transmitter FIFO. Reset: 0
10	Receiver FIFO full (RF) - indicates that the Receiver FIFO is full. Reset: 0
9	Transmitter FIFO full (TF) - indicates that the Transmitter FIFO is full. Reset: 0
8	Receiver FIFO half-full (RH) -indicates that at least half of the FIFO is holding data. Reset: 0
7	Transmitter FIFO half-full (TH) - indicates that the FIFO is less than half-full. Reset: 0
6	Framing error (FE) - indicates that a framing error was detected. Reset: 0
5	Parity error (PE) - indicates that a parity error was detected. Reset: 0
4	Overrun (OV) - indicates that one or more character have been lost due to overrun. Reset: 0
3	Break received (BR) - indicates that a BREAK has been received. Reset: 0
2	Transmitter FIFO empty (TE) - indicates that the transmitter FIFO is empty. Reset: 1
1	Transmitter shift register empty (TS) - indicates that the transmitter shift register is empty. Reset: 1
0	Data ready (DR) - indicates that new data is available in the receiver holding register. Reset: 0

15.7.3 UART Control Register

Table 97. UART control register

31	30 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FA	RESERVED	SI	DI	BI	DB	RF	TF	EC	LB	FL	PE	PS	TI	RI	TE	RE

31	FIFOs available (FA) - Set to 1 when receiver and transmitter FIFOs are available. When 0, only holding register are available. Read only.
30: 15	RESERVED
14	Transmitter shift register empty interrupt enable (SI) - When set, an interrupt will be generated when the transmitter shift register becomes empty. See section 15.6 for more details.
13	Delayed interrupt enable (DI) - When set, delayed receiver interrupts will be enabled and an interrupt will only be generated for received characters after a delay of 4 character times + 4 bits if no new character has been received during that interval. This is only applicable if receiver interrupt enable is set. See section 15.6 for more details. Not Reset.
12	Break interrupt enable (BI) - When set, an interrupt will be generated each time a break character is received. See section 16.6 for more details. Not Reset.
11	FIFO debug mode enable (DB) - when set, it is possible to read and write the FIFO debug register. Not Reset.
10	Receiver FIFO interrupt enable (RF) - when set, Receiver FIFO level interrupts are enabled. Not Reset.
9	Transmitter FIFO interrupt enable (TF) - when set, Transmitter FIFO level interrupts are enabled. Not Reset.
8	External Clock (EC) - if set, the UART scaler will be clocked by UARTI.EXTCLK. Reset: 0
7	Loop back (LB) - if set, loop back mode will be enabled. Not Reset.
6	Flow control (FL) - if set, enables flow control using CTS/RTS (when implemented). Reset: 0
5	Parity enable (PE) - if set, enables parity generation and checking (when implemented). Not Reset.
4	Parity select (PS) - selects parity polarity (0 = even parity, 1 = odd parity) (when implemented). Not Reset.
3	Transmitter interrupt enable (TI) - if set, interrupts are generated when characters are transmitted (see section 15.6 for details). Not Reset.
2	Receiver interrupt enable (RI) - if set, interrupts are generated when characters are received (see section 15.6 for details). Not Reset.
1	Transmitter enable (TE) - if set, enables the transmitter. Reset: 0
0	Receiver enable (RE) - if set, enables the receiver. Reset: 0

15.7.4 UART Scaler Register

Table 98. UART scaler reload register

31 sbits	sbits-1 0
RESERVED	SCALER RELOAD VALUE

sbits-1:0	Scaler reload value

15.7.5 UART FIFO Debug Register

Table 99. UART FIFO debug register

31 8	7 0
RESERVED	DATA

7: 0	Transmitter holding register or FIFO (read access)
7: 0	Receiver holding register or FIFO (write access)

15.8 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x00C. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

15.9 Configuration options

Table 100 shows the configuration options of the core (VHDL generics).

*Table 100.*Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
console	Prints output from the UART on console during VHDL simulation and speeds up simulation by always returning ‘1’ for Data Ready bit of UART Status register. Does not affect synthesis.	0 - 1	0
pirq	Index of the interrupt line.	0 - NAHBIRQ-1	0
parity	Enables parity	0 - 1	1
flow	Enables flow control	0 - 1	1
fifosize	Selects the size of the Receiver and Transmitter FIFOs	1, 2, 4, 8, 16, 32	1
abits	Selects the number of APB address bits used to decode the register addresses	3 - 8	8
sbits	Selects the number of bits in the scaler	12-32	12

15.10 Signal descriptions

Table 101 shows the interface signals of the core (VHDL ports).

*Table 101.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
UARTI	RXD	Input	UART receiver data	-
	CTSN	Input	UART clear-to-send	Low
	EXTCLK	Input	Use as alternative UART clock	-
UARTO	RTSN	Output	UART request-to-send	Low
	TXD	Output	UART transmit data	-
	SCALER	Output	UART scaler value	-
	TXEN	Output	Output enable for transmitter	High
	FLOW	Output	Unused	-
	RXEN	Output	Receiver enable	High

* see GRLIB IP Library User’s Manual

15.11 Library dependencies

Table 102 shows libraries that should be used when instantiating the core.

*Table 102.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	APB signal definitions
GAISLER	UART	Signals, component	Signal and component declaration

15.12 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.uart.all;

entity apbuart_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- UART signals
    rxd   : in  std_ulogic;
    txd   : out std_ulogic
    );
end;

architecture rtl of apbuart_ex is

  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- UART signals
  signal uarti : uart_in_type;
  signal uarto : uart_out_type;

begin

  -- AMBA Components are instantiated here
   ...


  -- APB UART
  uart0 : apbuart
  generic map (pindex => 1, paddr => 1,  pirq => 2,
console => 1, fifosize => 1)
  port map (rstn, clk, apbi, apbo(1), uarti, uarto);

  -- UART input data
  uarti.rxd <= rxd;

  -- APB UART inputs not used in this configuration
  uarti.ctsn <= ’0’; uarti.extclk <= ’0’;

  -- connect APB UART output to entity output signal
  txd <= uarto.txd;

end;
```

16 APBVGA - VGA controller with APB interface

16.1 Introduction

The APBVGA core is a text-only video controller with a resolution of 640x480 pixels, creating a display of 80x37 characters. The controller consists of a video signal generator, a 4 Kbyte text buffer, and a ROM for character pixel information. The video controller is controlled through an APB interface.

A block diagram for the data path is shown in figure 26.

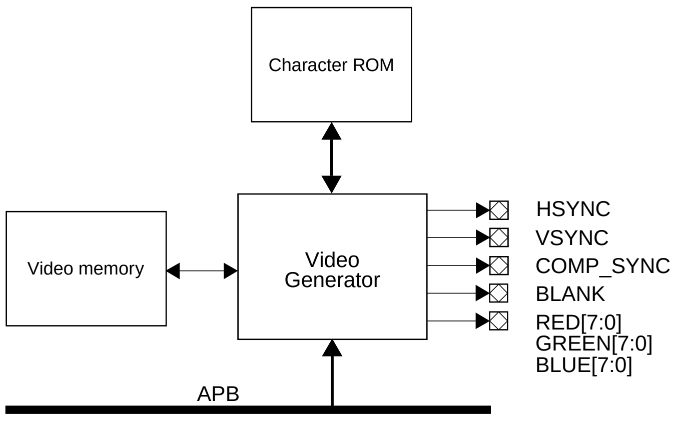


Figure 26. APBVGA block diagram

16.2 Operation

The video timing of APBVGA is fixed to generate a 640x480 display with 60 Hz refresh rate. The text font is encoded using 8x13 pixels. The display is created by scanning a segment of 2960 characters of the 4 Kbyte text buffer, rasterizing the characters using the character ROM, and sending the pixel data to an external video DAC using three 8-bit color channels. The required pixel clock is 25.175 MHz, which should be provided on the VGACLK input.

Writing to the video memory is made through the VGA data register. Bits [7:0] contains the character to be written, while bits [19:8] defines the text buffer address. Foreground and background colours are set through the background and foreground registers. These 24 bits corresponds to the three pixel colors, RED, GREEN and BLUE. The eight most significant bits defines the red intensity, the next eight bits defines the green intensity and the eight least significant bits defines the blue intensity. Maximum intensity for a color is received when all eight bits are set and minimum intensity when none of the bits are set. Changing the foreground color results in that all characters change their color, it is not possible to just change the color of one character. In addition to the color channels, the video controller generates HSYNC, VSYNC, CSYNC and BLANK. Togetherm the signals are suitable to drive an external video DAC such as ADV7125 or similar.

APBVGA implements hardware scrolling to minimize processor overhead. The controller monitors maintains a reference pointer containing the buffer address of the first character on the top-most line. When the text buffer is written with an address larger than the reference pointer + 2960, the pointer is incremented with 80. The 4 Kbyte text buffer is sufficient to buffer 51 lines of 80 characters. To simplify hardware design, the last 16 bytes (4080 - 4095) should not be written. When address 4079 has been written, the software driver should wrap to address 0. Sofware scrolling can be implemented by only using the first 2960 address in the text buffer, thereby never activating the hardware scolling mechanism.

16.3 Registers

The APB VGA is controlled through three registers mapped into APB address space.

*Table 103.*APB VGA registers

APB address offset	**Register**
0x0	VGA Data register (write-only, reads will return 0x00000000).
0x4	VGA Background color (write-only, reads will return 0x00000000).
0x8	VGA Foreground color (write-only, reads will return 0x00000000).

16.3.1 VGA Data Register

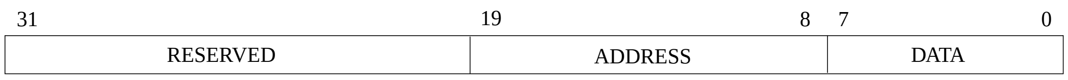


Figure 27. VGA data register

[19:8]: Video memory address (write access)
[7:0]: Video memory data (write access)

16.3.2 VGA Background Color

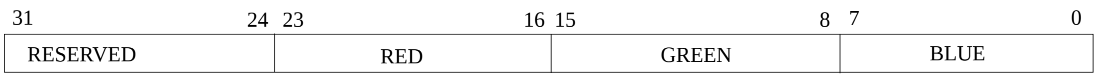


Figure 28. VGA Background color

[23:16]: Video background color red.
[15:8]: Video background color green.
[7:0]: Video background color blue.

16.3.3 VGA Foreground Color

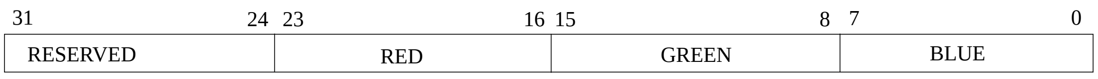


Figure 29. VGA Foreground color

[23:16]: Video foreground color red.
[15:8]: Video foreground color green.
[7:0]: Video foreground color blue.

16.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x061. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

16.5 Configuration options

Table 104 shows the configuration options of the core (VHDL generics).

*Table 104.*Configuration options

Generic	Function	Allowed range	Default
memtech	Technology to implement on-chip RAM	0 - NTECH	2
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#

16.6 Signal descriptions

Table 105 shows the interface signals of the core (VHDL ports).

*Table 105.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
VGACLK	N/A	Input	VGA Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
VGAO	HSYNC	Output	Horizontal synchronization	High
	VSYNC		Vertical synchronization	High
	COMP_SYNC		Composite synchronization	Low
	BLANK		Blanking	Low
	VIDEO_OUT_R[7:0]		Video out, color red	-
	VIDEO_OUT_G[7:0]		Video out, color green	-
	VIDEO_OUT_B[7:0]		Video out, color blue	-
	BITDEPTH[1:0]		Constant High	-

* see GRLIB IP Library User's Manual

16.7 Library dependencies

Table 106 shows libraries used when instantiating the core (VHDL libraries).

*Table 106.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	APB signal definitions
GAISLER	MISC	Signals, component	VGA signal and component declaration

16.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
```

```
library gaisler;
use gaisler.misc.all;

.
.


architecture rtl of apbuart_ex is

signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
signal vgao  : apbvga_out_type;

begin
  -- AMBA Components are instantiated here
   ...

  -- APB VGA
  vga0 : apbvga
  generic map (memtech => 2, pindex => 6, paddr => 6)
  port map (rstn, clk, vgaclk, apbi, apbo(6), vgao);
end;
```

17 B1553BC - AMBA plug&play interface for Actel Core1553BBC

17.1 Overview

The interface provides a complete Mil-Std-1553B Bus Controller (BC). The interface connects to the MIL-STD-1553B bus through external transceivers and transformers. The interface is based on the Actel Core1553BBC core.

The interface provides a complete, MIL-STD-1553B Bus Controller (BC). The interface reads message descriptor blocks from the memory and generates messages that are transmitted on and transmitted on the 1553B bus. Data received is written to the memory.

The interface consists of five main blocks: the 1553B encoder, the 1553B decoder, a protocol controller block, a CPU interface, and a backend interface.

A single 1553B encoder takes each word to be transmitted and serializes it using Manchester encoding. The encoder includes independent logic to prevent the BC from transmitting for greater than the allowed period and to provide loopback fail logic. The loopback logic monitors the received data and verifies that the interface has correctly received every word that is transmitted. The encoder output is gated with the bus enable signals to select which buses the encoder should be transmitting. Since the BC knows which bus is in use at any time, only a single decoder is required.

The decoder takes the serial Manchester received data from the bus and extracts the received data words The decoder contains a digital phased lock loop (PLL) that generates a recovery clock used to sample the incoming serial data. The data is then deserialized and the 16-bit word decoded. The decoder detects whether a command, status or data word has been received and checks that no Manchester encoding or parity errors occurred in the word.

The protocol controller block handles all the message sequencing and error recovery. This is a complex state machine that reads the 1553B message frames from memory and transmits them on the 1553B bus. The AMBA interface allows a system processor to access the control registers. It also allows the processor to directly access the memory connected to the backend interface, this simplifies the system design.

The B1553BC core provides an AMBA interface with GRLIB plug&play for the Actel Core1553BBC core (MIL-STD-1553B Bus Controller). B1553BC implements two AMBA interfaces: one AHB master interface for the memory interface, and one APB slave interface for the CPU interface and control registers.

The Actel Core1553BBC core, entity named BC1553B, is configured to use the shared memory interface, and only internal register access is allowed through the APB slave interface. Data is read and stored via DMA using the AHB master interface.

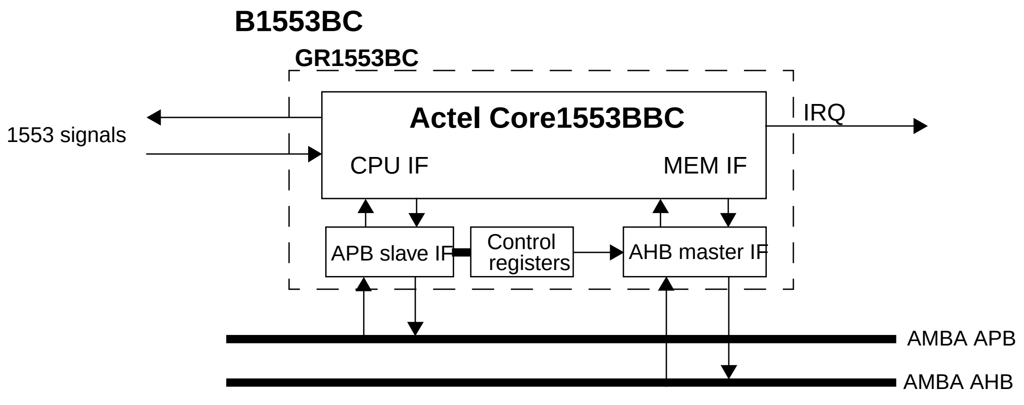


Figure 30. Block diagram

17.2 AHB interface

The Core1553BBC operates on a 65536 x 16 bit memory buffer, and therefore a 128 kilobyte aligned memory area should be allocated. The memory is accessed via the AMBA AHB bus. The Core1553BBC uses only 16 address bits, and the top 15 address bits of the 32-bit AHB address can be programmed in the AHB page address register. The 16-bit address provided by the Core1553BBC is left-shifted one bit, and forms the AHB address together with the AHB page address register. Note that all pointers given to the Core1553BBC core need to be right-shifted one bit because of this. All AHB accesses are done as half word single transfers.

The endianness of the interface depends on the endian VHDL generic.

The AMBA AHB protection control signal HPROT is driven permanently with “0011”, i.e a not cacheable, not bufferable, privileged data access. The AMBA AHB lock signal HLOCK is driven with ‘0’.

17.3 Operation

To transmit data on the 1553 bus, an instruction list and 1553 messages should be set up in the memory by the processor. After the bus interface has been activated, it will start to process the instruction list and read/write data words from/to the specified memory locations. Interrupts are generated when interrupt instructions are executed, on errors or when the interface has completed the list.

17.4 Synthesis

The B1553BC core is a wrapper providing GRLIB compatible signals and plug&play information around the GR1553B core, which in turn provides an AMBA interface around the Microsemi/Actel Core1553BBC core.

17.5 Registers

The core is programmed through registers mapped into APB address space. The internal registers of Core1553BBC are mapped on the eight lowest APB addresses. These addresses are 32-bit word aligned although only the lowest 16 bits are used. Refer to the *Actel Core1553BBC MIL-STD-1553B Bus Controller* data sheet for detailed information.

*Table 107.*B1553BC registers

APB address offset	Register
0x00	Control/Status
0x04	Setup
0x08	List pointer
0x0C	Message pointer
0x10	Clock value
0x14	Asynchronous list pointer
0x18	Stack pointer
0x1C	Interrupt register
0x20	GR1553 status/control
0x24	AHB page address register

Table 108. GR1553 status register (read)

31 - 3	2	1	0
RESERVED	extflag	memfail	busy

Table 108. GR1553 status register (read)

31: 3	RESERVED
2	External flag bit. Drives the extflag input of the Core1553BBC. Resets to zero.
1	Memory failure. Shows the value of the memfail output from Core1553BBC.
0	Busy. Shows the value of the busy output from Core1553BBC.

Table 109. GR1553 status register (write)

31 1	0
RESERVED	extflag

31: 2	RESERVED
0	External flag bit. Drives the extflag input of the Core1553BBC. Resets to zero.

Table 110. GR1553 status register (write)

31 17	16 0
ahbaddr	RESERVED

31: 17	Holds the 15 top most bits of the AHB address of the allocated memory area
16: 0	RESERVED

17.6 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x070. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

17.7 Configuration options

Table 111 shows the configuration options of the core (VHDL generics).

*Table 111.*Configuration options

Generic	**Function**	**Allowed range**	**Default**
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Interrupt number	0 - NAHBIRQ -1	0

17.8 Configuration options for underlying GR1553BC core

Table 111 shows the configuration options of the core (VHDL generics).

*Table 112.*Configuration options

Generic	**Function**	**Allowed range**	**Default**
endian	Data endianness of the AHB bus (Big = 0, Little = 1)	0 - 1	0

17.9 Signal descriptions

Table 113 shows the interface signals of the core (VHDL ports).

*Table 113.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
B1553I	-	Input	1553 bus input signals	-
	busainp		Positive data input from the A receiver	High
	busainn		Negative data input from the A receiver	Low
	busbinp		Positive data to the B receiver	High
	busbinn		Negative data to the B receiver	Low
B1553O	-	Output	1553 bus output signals	-
	busainen		Enable for the A receiver	High
	busaoutin		Inhibit for the A transmitter	High
	busaoutp		Positive data to the A transmitter	High
	busaoutn		Negative data to the A transmitter	Low
	busbinen		Enable for the B receiver	High
	busboutin		Inhibit for the B transmitter	High
	busboutp		Positive output to the B transmitter	High
	busboutn		Negative output to the B transmitter	Low
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-

* see GRLIB IP Library User’s Manual

17.10 Signal descriptions for underlying GR1553BC core

Table 113 shows the interface signals of the core (VHDL ports).

*Table 114.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
1553 bus input signals				
BUSAINP	N/A	Input	Positive data input from the A receiver	High
BUSAINN		Input	Negative data input from the A receiver	Low
BUSBINP	N/A	Input	Positive data to the B receiver	High
BUSBINN		Input	Negative data to the B receiver	Low
1553 bus output signals				
BUSAINEN	N/A	Output	Enable for the A receiver	High
BUSAOUTIN		Output	Inhibit for the A transmitter	High
BUSAOUTP	N/A	Output	Positive data to the A transmitter	High
BUSAOUTN		Output	Negative data to the A transmitter	Low
BUSBINEN	N/A	Output	Enable for the B receiver	High
BUSBOUTIN		Output	Inhibit for the B transmitter	High
BUSBOUTP	N/A	Output	Positive output to the B transmitter	High
BUSBOUTN		Output	Negative output to the B transmitter	Low
Interrupt				
INTOUT	N/A	Output	Interrupt	High
AHB signals				
HGRANT	N/A	Input	Bus grant	High
HREADY	N/A	Input	Transfer done	High
HRESP	N/A	Input	Response type	-
HRDATA	N/A	Input	Read data bus	-
HBUSREQ	N/A	Output	Bus request	High
HLOCK	N/A	Output	Lock request	High
HTRANS	N/A	Output	Transfer type	-
HADDR	N/A	Output	Address bus (byte addresses)	-
HWRITE	N/A	Output	Write	High
HSIZE	N/A	Output	Transfer size	-
HBURST	N/A	Output	Burst type	-
HPROT	N/A	Output	Protection control	-
HWDATA	N/A	Output	Write data bus	-
APB signals				
PSEL	N/A	Input	Slave select	High
PENABLE	N/A	Input	Strobe	High
PADDR	N/A	Input	Address bus (byte addresses)	-
PWRITE	N/A	Input	Write	High
PWDATA	N/A	Input	Write data bus	-
PRDATA	N/A	Output	Read data bus	-

17.11 Library dependencies

Table 115 shows libraries used when instantiating the core (VHDL libraries).

*Table 115.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	Signal definitions
GAISLER	B1553	Signals, component	Signal and component declaration

The B1553BC depends on GRLIB, GAISLER, GR1553 and Core1553BBC.

17.12 Library dependencies for underlying GR1553BC core

Table 115 shows libraries used when instantiating the core (VHDL libraries).

*Table 116.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
IEEE	Std_Logic_1164	All	Type declarations

The GR1553BC depends on GR1553 and Core1553BBC.

17.13 Component declaration

The core has the following component declaration.

```
component b1553bc is
    generic (
      hindex  : integer := 0;
      pindex  : integer := 0;
      paddr   : integer := 0;
      pmask   : integer := 16#fff#;
      pirq    : integer := 0
      );
    port (
      rstn    : in  std_ulogic;
      clk     : in  std_ulogic;
      b1553i  : in  b1553_in_type;
      b1553o  : out b1553_out_type;
      apbi    : in  apb_slv_in_type;
      apbo    : out apb_slv_out_type;
      ahbi    : in  ahb_mst_in_type;
      ahbo    : out ahb_mst_out_type
      );
  end component;
```

17.14 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.b1553.all;
...
signal bin : b1553_in_type;
signal bout : b1553_out_type;
...
bc1553_0 : b1553bc
    generic map (hindex => 2, pindex => 12, paddr => 12, pirq => 2)
    port map (rstn, clkm, bin, bout, apbi, apbo(12), ahbmi, ahbmo(2));
```

18 B1553BRM - AMBA plug&play interface for Actel Core1553BRM

18.1 Overview

The interface provides a complete Mil-Std-1553B Bus Controller (BC), Remote Terminal (RT) or Monitor Terminal (MT). The interface connects to the MIL-STD-1553B bus through external transceivers and transformers. The interface is based on the Actel Core1553BRM core.

The interface consists of six main blocks: 1553 encoder, 1553B decoders, a protocol controller block, AMBA bus interface, command word legality interface, and a backend interface.

The interface can be configured to provide all three functions BC, RT and MT or any combination of the three. All variations use all six blocks except for the command legalization interface, which is only required on RT functions that implement RT legalization function externally.

A single 1553 encoder takes each word to be transmitted and serializes it using Manchester encoding. The encoder also includes independent logic to prevent the interface from transmitting for greater than the allowed period as well as loopback fail logic. The loopback logic monitors the received data and verifies that the interface has correctly received every word that it transmits. The output of the encoder is gated with the bus enable signals to select which buses the interface should be transmitting on. Two decoders take the serial Manchester received data from each bus and extract the received data words.

The decoder contains a digital phased lock loop (PLL) that generates a recovery clock used to sample the incoming serial data. The data is then de-serialized and the 16-bit word decoded. The decoder detects whether a command, status, or data word has been received, and checks that no Manchester encoding or parity errors occurred in the word.

The protocol controller block handles all the message sequencing and error recovery for all three operating modes, Bus Controller, Remote Terminal, and Bus Monitor. This is complex state machine that processes messages based on the message tables setup in memory, or reacts to incoming command words. The protocol controller implementation varies depending on which functions are implemented. The AMBA interface allows a system processor to access the control registers. It also allows the processor to directly access the memory connected to the backend interface, this simplifies the system design.

The interface comprises 33 16-bit registers. Of the 33 registers, 17 are used for control function and 16 for RT command legalization.

The B1553BRM core provides an AMBA interface for the Actel Core1553BRM core (MIL-STD-1553B Bus Controller/Remote Terminal/Bus Monitor). The B1553BRM core implements two AMBA interfaces: one AHB master interface for the memory interface, and one APB slave interface for the CPU interface and control registers.

The Actel Core1553BRM core, entity named BRM, is configured to use the shared memory interface, and only internal register access is allowed through the APB slave interface. Data is read and stored via DMA using the AHB master interface.

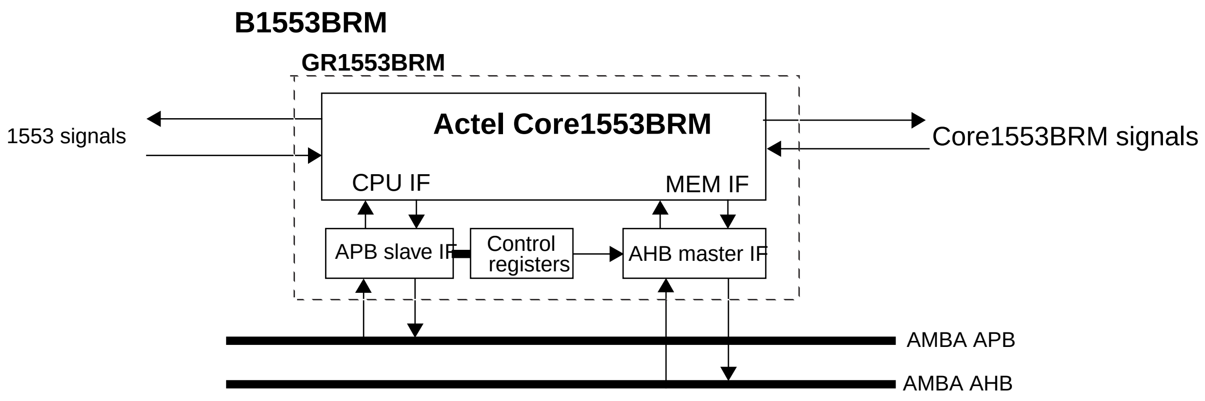


Figure 31. Block diagram

18.2 AHB interface

The amount of memory that the Mil-Std-1553B interface can address is 128 (2***abit* VHDL generic, i.e. *abit* => 128) kbytes. The base address of this memory area must be aligned to a boundary of its own size and written into the AHB page address register.

The 16 bit address provided by the Core1553BRM core is shifted left one bit, and forms the AHB address together with the AHB page address register. Note that all pointers given to the Core1553BRM core needs to be right shifted one bit because of this.

The amount of memory needed for the Core1553BRM core is operation and implementation specific. Any configuration between 1 to 128 kilobytes is possible although a typical system needs at least 4 kbyte of memory. The allocated memory area needs to be aligned to a boundary of its own size and the number of bits needed to address this area must be specificed with the *abits* VHDL generic.

The address bus of the Core1553BRM is 16 bits wide but the amount of bits actually used depends on the setup of the data structures. The AHB page address register should be programmed with the 32-*abits* top bits of the 32-bit AHB address, *abit* being a VHDL generic. The address provided by the Core1553BRM core is shifted left one bit, and forms the AHB address together with the AHB page address register. Note that all pointers given to the Core1553BRM core needs to be right shifted one bit because of this.

When the Core1553BRM core has been granted access to the bus it expects to be able to do a series of uninterrupted accesses. To handle this requirement the AHB master locks the bus during these transfers. In the worst case, the Core1553BRM can do up to 7 writes in one such access and each write takes 2 plus the number of waitstate cycles with 4 idle cycles between each write strobe. This means care has to be taken if using two simultaneous active Core1553BRM cores on the same AHB bus.All AHB accesses are done as half word single transfers.

The endianness of the interface depends on the endian VHDL generic.

The AMBA AHB protection control signal HPROT is driven permanently with "0011" i.e a not cacheable, not bufferable, privileged data access. During all AHB accesses the AMBA AHB lock signal HLOCK is driven with `1' and `0' otherwise.

18.3 Operation

The mode of operation can be selected with the mselin VHDL generic or later changed by writing to the “operation and status” register of the Core1553BRM core. For information about how the core functions during the different modes of operation see the *Actel Core1553BRM MIL-STD-1553 BC, RT, and MT* data sheet.

18.4 Synthesis

The B1553BRM core is a wrapper providing GRLIB compatible signals and plug&play information around the GR1553BRM core, which in turn provides an AMBA interface around the Microsemi/ Actel Core1553BRM core.

18.5 Registers

The core is programmed through registers mapped into APB address space. The internal registers of Core1553BRM are mapped on the 33 lowest APB addresses. These addresses are 32-bit word aligned although only the lowest 16 bits are used. Refer to the *Actel Core1553BRM MIL-STD-1553 BC, RT, and MT* data sheet for detailed information.

*Table 117.*B1553BRM registers

APB address offset	**Register**
0x00 - 0x84	Core1553BRM registers
0x100	B1553BRM status/control
0x104	B1553BRM interrupt settings
0x108	AHB page address register

B1553BRM status/control register

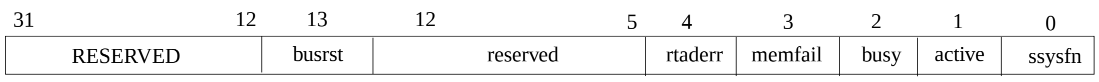


Figure 32. B1553BRM status/control register

13	Bus reset. If set a bus reset mode code has been received. Generates an irq when set.
12:5	Reserved
4:	Address error. Shows the value of the rtaderr output from Core1553BRM.
3:	Memory failure. Shows the value of the memfail output from Core1553BRM.
2:	Busy. Shows the value of the busy output from Core1553BRM.
1:	Active. Show the value of the active output from Core1553BRM.
0:	Ssyfn. Connects directly to the ssyfn input of the Core1553BRM core. Resets to 1.

B1553BRM interrupt register

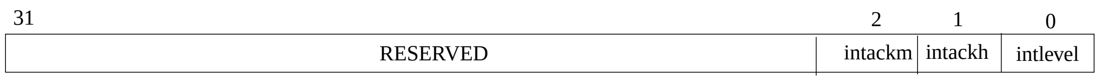


Figure 33. B1553RM interrupt register

2:	Message interrupt acknowledge. Controls the intackm input signal of the Core1553BRM core.
1:	Hardware interrupt acknowledge. Controls the intackh input signal of the Core1553BRM core.
0:	Interrupt level. Controls the intlevel input signal of the Core1553BRM core.

AHB page address register

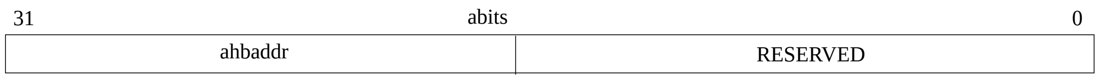


Figure 34. AHB page address register

[31:17]: Holds the top most bits of the AHB address of the allocated memory area.

18.6 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x072. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

18.7 Configuration options

Table 118 shows the configuration options of the core (VHDL generics).

*Table 118.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0-NAHBMST-1	0
pindex	APB slave index	0-NAPBSLV-1	0
paddr	ADDR field of the APB BAR	0-16#FFF#	0
pmask	MASK field of the APB BAR	0-16#FF0#	16#FF0#
pirq	Index of the interrupt line	0-NAHBIRQ-1	0
endian	Data endianness of the AHB bus (Big = 0, Little = 1)	0 -1	0
ahbaddr	Reset value for address register	16#00000#-16#FFFFF#	16#00000#
abits	Number of bits needed to address the memory area	12-17	17
rtaddr	RT address	0 - 31	0
rtaddrp	RT address parity bit. Set to achieve odd parity.	0 - 1	1
lockn	Lock rtaddrin, rtaddrp, mselin and abstdin	0-1	0
mselin	Mode select	0-3	0
abstdin	Bus standard A/B	0-1	0
bcenable	Enable bus controller	0-1	1
rtenable	Enable remote terminal	0-1	1
mtenable	Enable bus monitor	0-1	1
legregs	Enable legalization registers	0-1	1
enhanced	Enable enhanced register	0-1	1
initfreq	Initial operation frequency	12,16,20,24	20
betiming	Backend timing	0-1	1

The VHDL generics hindex, pindex, paddr, pmask and pirq belong to the B1553BRM entity.

The VHDL generics endian, ahbaddr and abits belong to the GR1553BRM entity.

The VHDL generics rtaddr, rtaddrp, lockn, mselin and abstdin belong to the Core1553BRM core, and drive the corresponding signal or generic.

The VHDL generics bcenable, rtenable, mtenable, legregs, enhanced, initfreq and betiming belong to the RTL version of the Core1553BRM core, and are otherwise ignored.

18.8 Signal descriptions

Table 119 shows the interface signals of the core (VHDL ports).

*Table 119.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
RSTOUTN	N/A	Output	Reset from BRM core	Low
CLK	N/A	Input	System clock (AHB)	-
TCLK	N/A	Input	External time base	-
B1553I	-	Input	1553 bus input signals	-
	busainp		Positive data input from the A receiver	High
	busainn		Negative data input from the A receiver	Low
	busbinp		Positive data to the B receiver	High
	busbinn		Negative data to the B receiver	Low
B1553O	-	Output	1553 bus output signals	-
	busainen		Enable for the A receiver	High
	busaoutin		Inhibit for the A transmitter	High
	busaoutp		Positive data to the A transmitter	High
	busaoutn		Negative data to the A transmitter	Low
	busbinen		Enable for the B receiver	High
	busboutin		Inhibit for the B transmitter	High
	busboutp		Positive output to the B transmitter	High
	busboutn		Negative output to the B transmitter	Low
BRMI	-	Input	BRM input signals	-
	cmdok		Command word validation alright	High
BRMO	-	Output	BRM output signals	-
	msgstart		Message process started	High
	cmdsync		Start of command word on bus	High
	syncnow		Synchronize received	High
	busreset		Reset command received	High
	opmode		Operating mode	-
	cmdval		Active command	-
	cmdokout		Command word validated	High
	cmdstb		Active command value changed	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-

* see GRLIB IP Library User’s Manual

18.9 Signal descriptions of the underlying GR1553BRM core

Table 119 shows the interface signals of the core (VHDL ports).

*Table 120.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RSTN	N/A	Input	Reset	Low
RSTOUTN	N/A	Output	Reset	Low
CLK	N/A	Input	Clock	-
TCLK	N/A	Input	Transmit clock	-
1553 bus input signals				
BUSAINP	N/A	Input	Positive data input from the A receiver	High
BUSAINN	N/A	Input	Negative data input from the A receiver	Low
BUSBINP	N/A	Input	Positive data to the B receiver	High
BUSBINN	N/A	Input	Negative data to the B receiver	Low
1553 bus output signals				
BUSAINEN	N/A	Output	Enable for the A receiver	High
BUSAOUTIN	N/A	Output	Inhibit for the A transmitter	High
BUSAOUTP	N/A	Output	Positive data to the A transmitter	High
BUSAOUTN	N/A	Output	Negative data to the A transmitter	Low
BUSBINEN	N/A	Output	Enable for the B receiver	High
BUSBOUTIN	N/A	Output	Inhibit for the B transmitter	High
BUSBOUTP	N/A	Output	Positive output to the B transmitter	High
BUSBOUTN	N/A	Output	Negative output to the B transmitter	Low
BRM input signals				
CMDOK	N/A	Input	Command word validation alright	High
BRM output signals				
MSGSTART	N/A	Output	Message process started	High
CMDSYNC	N/A	Output	Start of command word on bus	High
SYNCNOW	N/A	Output	Synchronize received	High
BUSRESET	N/A	Output	Reset command received	High
OPMODE	N/A	Output	Operating mode	-
CMDVAL	N/A	Output	Active command	-
CMDOKOUT	N/A	Output	Command word validated	High
CMDSTB	N/A	Output	Active command value changed	High
Interrupts				
INTOUTH	N/A	Output	Hardware interrupt request	High
INTOUTM	N/A	Output	Message interrupt request	High
AHB signals				
HGRANT	N/A	Input	Bus grant	High
HREADY	N/A	Input	Transfer done	High
HRESP	N/A	Input	Response type	-
HRDATA	N/A	Input	Read data bus	-
HBUSREQ	N/A	Output	Bus request	High
HLOCK	N/A	Output	Lock request	High
HTRANS	N/A	Output	Transfer type	-
HADDR	N/A	Output	Address bus (byte addresses)	-

*Table 120.*Signal descriptions

Signal name	Field	Type	Function	Active
HWRITE	N/A	Output	Write	High
HSIZE	N/A	Output	Transfer size	-
HBURST	N/A	Output	Burst type	-
HPROT	N/A	Output	Protection control	-
HWDATA	N/A	Output	Write data bus	-
APB signals				
PSEL	N/A	Input	Slave select	High
PENABLE	N/A	Input	Strobe	High
PADDR	N/A	Input	Address bus (byte addresses)	-
PWRITE	N/A	Input	Write	High
PWDATA	N/A	Input	Write data bus	-
PRDATA	N/A	Output	Read data bus	-

18.10 Library dependencies

Table 121 shows libraries used when instantiating the core (VHDL libraries).

*Table 121.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	Signal definitions
GAISLER	B1553	Signals, component	Signal and component declaration

The B1553BRM depends on VHDL libraries GRLIB, GAISLER, GR1553 and Core1553BRM.

18.11 Library dependencies of the underlying GR1553BRM core

Table 121 shows libraries used when instantiating the core (VHDL libraries).

*Table 122.*Library dependencies

Library	Package	Imported unit(s)	Description
IEEE	Std_Logic_1164	All	Type declarations

The GR1553BRM depends on VHDL libraries GR1553 and Core1553BRM.

18.12 Component declaration

The core has the following component declaration.

```
component b1553brm is
     generic (
       hindex       : integer := 0;
       pindex       : integer := 0;
       paddr        : integer := 0;
       pmask        : integer := 16#ff0#;
       pirq         : integer := 0;
       ahbaddr      : integer range 0 to 16#FFFFF# := 0;
       abits        : integer range 12 to 17 := 16;
       rtaddr       : integer range 0 to 31 := 0;
       rtaddrp      : integer range 0 to 1  := 1;
       lockn        : integer range 0 to 1  := 1;
       mselin       : integer range 0 to 3  := 1;
       abstdin      : integer range 0 to 1  := 0;

       bcenable     : integer range 0 to 1  := 1;
```

```
      rtenable     : integer range 0 to 1  := 1;
      mtenable     : integer range 0 to 1  := 1;
      legregs      : integer range 0 to 4  := 1;
      enhanced     : integer range 0 to 1  := 1;
      initfreq     : integer range 12 to 24:= 20;
      betiming     : integer range 0 to 1  := 1
      );
    port (
      rstn      : in    std_ulogic;
      rstoutn   : out   std_ulogic;
      clk       : in    std_ulogic;
      tclk      : in    std_ulogic;
      brmi      : in    brm1553_in_type;
      brmo      : out   brm1553_out_type;
      b1553i    : in    b1553_in_type;
      b1553o    : out   b1553_out_type;
      apbi      : in    apb_slv_in_type;
      apbo      : out   apb_slv_out_type;
      ahbi      : in    ahb_mst_in_type;
      ahbo      : out   ahb_mst_out_type
      );
  end component;
```

18.13 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.b1553.all;

...
signal bin : b1553_in_type;
signal bout : b1553_out_type;
signal brmi : brm1553_in_type;
signal brmo : brm1553_out_type;
...

bc1553_0 : b1553brm
    generic map (hindex => 2, pindex => 12, paddr => 16#10#, pirq => 2,
abits => 17, mselin => 0)
    port map (rstn, open, clkm, gnd(0), brmi, brmo, bin, bout, apbi, apbo(12), ahbmi,
ahbmo(2));
```

19 B1553RT - AMBA plug&play interface for Actel Core1553BRT

19.1 Overview

The interface provides a complete Mil-Std-1553B Remote Terminal (RT). The interface connects to the MIL-STD-1553B bus through external transceivers and transformers. The interface is based on the Actel Core1553BRT core.

The interface provides a complete, dual-redundant MIL-STD-1553B remote terminal (RT) apart from the transceivers required to interface to the bus. At a high level, the interface simply provides a set of memory mapped sub-addresses that 'receive data written to' or 'transmit data read from.' The interface requires 2,048 words of memory, which can be shared with a local processor. The interface supports all 1553B mode codes and allows the user to designate as illegal any mode code or any particular sub-address for both transmit and receive operations. The command legalization can be done internally or via an external command legalization interface.

The interface consists of six main blocks: 1553B encoders, 1553B decoders, backend interface, command decoder, RT controller blocks and a command legalization block.

A single 1553B encoder is used for the interface. This takes each word to be transmitted and serializes it, after which the signal is Manchester encoded. The encoder also includes both logic to prevent the RT from transmitting for greater than the allowed period and loopback fail logic. The loopback logic monitors the received data and verifies that the interface has correctly received every word that it transmits. The output of the encoder is gated with the bus enable signals to select which buses the RT should use to transmit.

The interface includes two 1553B decoders. The decoder takes the serial Manchester data received from the bus and extracts the received data words. The decoder contains a digital phased lock loop (PLL) that generates a recovery clock used to sample the incoming serial data. The data is then deserialized and the 16-bit word decoded. The decoder detects whether a command or data word is received, and also performs Manchester encoding and parity error checking.

The command decoder and RT controller blocks decode the incoming command words, verifying the legality. Then the protocol state machine responds to the command, transmitting or receiving data or processing a mode code.

The B1553RT core provides an AMBA interface with GRLIB plug&play for the Actel Core1553BRT (MIL-STD-1553B Remote Terminal). B1553RT implements two AMBA interfaces: one AHB master interface for the memory interface, and one APB slave interface for the control registers.

The Actel Core1553BRT core, entity named RT1553B, is configured to use the shared memory interface. Data is read and stored via DMA using the AHB master interface.

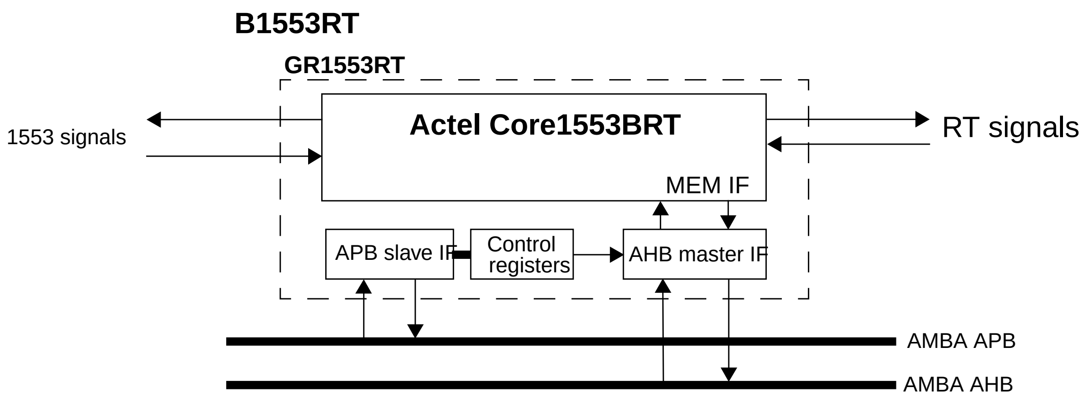


Figure 35. B1553RT block diagram

19.2 Synthesis

The B1553RT core is a wrapper providing GRLIB compatible signals and plug&play information around the GR1553B core, which in turn provides an AMBA interface around the Microsemi/Actel Core1553BRT core.

19.3 Operation

19.3.1 Memory map

The Core1553BRT core operates on a 2048*16 bit memory buffer, and therefore a 4 kilobyte memory area should be allocated. The memory is accessed via the AMBA AHB bus. The Core1553BRT uses only 11 address bits, and the top 20 address bits of the 32-bit AHB address can be programmed in the AHB page address register. The 11-bit address provided by the Core1553BRT core is left-shifted one bit, and forms the AHB address together with the AHB page address register. All AHB accesses are done as half word single transfers.

The used memory area has the following address map. Note that all 1553 data is 16 bit wide and will occupy two bytes. Every sub-address needs memory to hold up to 32 16 bit words.

*Table 123.*Memory map for 1553 data

Address	Content
0x000-0x03F	RX transfer status/command words
0x040-0x07F	Receive sub-address 1 ...
0x780-0x7BF	Receive sub-address 30
0x7C0-0x7FF	TX transfer status/command words
0x800-0x83F	Not used, except 0x800-0x801 for vector word if extmdata=1
0x840-0x87F	Transfer sub-address 1 ...
0xF80-0xFBF	Transfer sub-address 30
0xFC0-0xFFF	Not used, except 0xFC0-0xFC1 for vector word if extmdata=1

19.3.2 Data transfers

At the start of a bus transfer the core writes the 1553B command word (if the wrtcmd bit is set in the control register) to the address subaddress*2 for receive commands and 0x7C0 + subaddress*2 for transmit commands. After a bus transfer has completed a transfer status word is written to the same location as the command word (if wrttsw bit is set in the control register). The command word of the last transfer can always be read out through the interrupt vector and command value register.

The transfer status word written to memory has the following layout:

*Table 124.*Transfer Status Word layout

Bit	Name	Description
15	USED	Always set to 1 at the end of bus transfer
14	OKAY	Set to 1 if no errors were detected
13	BUSN	Set to 0 if transfer was on bus A, to 1 if bus B
12	BROADCAST	Transfer was a broadcast command
11	LPBKERRB	The loopback logic detected error on bus B
10	LPBKERRA	The loopback logic detected error on bus A
9	ILLCMD	Illegal command
8	MEMIFERR	DMA access error did not omplete in time
7	MANERR	Manchester coding error detected
6	PARERR	Parity error detected
5	WCNTERR	Wrong number of words was received
4:0	COUNT	For sub address 1-30: number of words received/transmitted, 0 means 32
		For sub address 0 and 31: received/transmitted mode code

The AMBA AHB protection control signal HPROT is driven permanently with “0011”, i.e a not cacheable, not bufferable, privileged data access. The AMBA AHB lock signal HLOCK is driven with ‘0’.

19.3.3 Mode commands

All mode codes defined by 1553B are legal except dynamic bus control (0), selected transmitter shutdown (20) and override selected transmitter shutdown (21).

Like data transfers, mode commands will write a command word before the transfer if the wrtcmd bit is set and a transmit status word after if the wrttsw bit is set. The transmit vector word and sync with data word will also store/fetch the data word from memory if the extmdata control bit is set. The locations are tabulated below. The default mapping for sync with data word with subaddress 0 places the data word and TSW at the same address 0, therefore a re-mapping has been implemented if wrttsw is set.

*Table 125.*Mode command memory map

Mode codes	Subaddress	CMD / TSW Location	Data word location (if extmdata=1)
1 synchronize 2 transmit status 3 initiate self-test 4 transmitter shutdown 5 override tx shutdown 6 inhibit TF 7 override inhibit TF 8 reset	0	0x7C0	(no data)
	31	0x7FE	(no data)
16 tx vector word	0	0x7C0	0x800
	31	0x7FE	0xFC0
17 synchronize with data	0	0x000	0x000 if wrttsw=0 0x7C0 if wrttsw=1
	31	0x03E	0x7C0
18 tx last command 19 tx BIT word	0	0x7C0	(internal)
	31	0x7FE	(internal)

The transfer BIT word mode code transfers a word as specified in the table below:

*Table 126.*Built In Test word

Bit	Name	Description
15	BUSINUSE	Set to 0 if transfer was on bus A, to 1 if bus B
14	LPBKERRB	The loopback logic detected error on bus B. Cleared by CLRERR.
13	LPBKERRA	The loopback logic detected error on bus A. Cleared by CLRERR.
12	SHUTDOWNB	Indicates that bus B has been shutdown
11	SHUTDOWNA	Indicates that bus A has been shutdown
10	TFLAGINH	Terminal flag inhibit setting
9	WCNTERR	Word count error has occured. Cleared by CLRERR.
8	MANERR	Manchester coding error detected. Cleared by CLRERR.
7	PARERR	Parity error detected. Cleared by CLRERR.
6	RTRTTO	RT to RT transfer timeout. Cleared by CLRERR.
5	MEMFAIL	DMA transfer not completed in time. Cleared by CLRERR.
4:0	VERSION	Core1553RT version

19.4 Registers

The core is programmed through registers mapped into APB address space.

*Table 127.*B1553RT registers

APB Address offset	Register
0x00	Status
0x04	Control
0x08	Vector word
0x0C	Interrupt vector and command value
0x10	AHB page address register
0x14	Interrupt pending/mask register

Status register (read only)

31 28	27 4	3	2	1	0
revision	RESERVED	extleg	rtaderr	memfail	busy

Figure 36. Status register

31:28 Core revision, read-only field.
0001: Added brdis,disabl,extleg fields. Remapped sync data word if extmdata and wrttsw are set.
0000: First revision. Reset mode command resets all control registers.
27:4 Reserved
3: Reads ‘1’ if configured with external legalization interface, ‘0’ if all supported accesses are legal
2: RT address error. Incorrect RT address parity bit.
1: Memory failure. DMA transfer did not complete in time. Cleared using CLRERR bit in control register.
0: Busy. Indicates that the RT is busy with a transfer.

Control register

31 23	22	21	20	19	18	17	16	15	14	13	12 8	7 6	5	4	3	2	1	0
RESERVED	brdis	disabl	reset	sa30loop	bcasten	intenbbr	extmdata	wrtcmd	wrttsw	rtaddrp	rtaddr	clkspd	clrerr	intack	tf	ssf	busy	sreq

Figure 37. Control register

31:23 Reserved
22: If ‘1’ the disable bit (bit 21) will be set to ‘1’ automatically when a reset mode command is recieved.
21: Set to ‘1’ to disable 1553 transceiver (both receiver and transmitter). Reset to ‘1’.
20: Writing ‘1’ will reset the Core1553RT and forces the B1553RT DMA to idle state. Self clearing.
19: Set to ‘1’ to enable internal loopback of subaddress 30. Transmits from sa 30 reads from the receive buffer for sa 30.
18: Set to ‘1’ to enable broadcasts messages. If ‘0’ address 31 is treated as normal RT address.
17: ‘1’ enables interrupts for bad messages. If ‘0’ only good messages generates interrupts.
16: If ‘1’ mode code data is written to / read from memory. If ‘0’ the vword register is used for transmit vector word mode code and the data for synchronize with data is discarded.
15: If ‘1’ the command word is written to memory at the start of a bus transfer.
14: If ‘1’ the transfer status word is written to memory at the end of a bus transfer.
13: RT address parity bit. Odd parity over rtaddr and rtaddrp must be achieved.
12:8 RT address.
7:6 Clock speed. Should be set to indicate the clock frequency of the core. 0 - 12, 1 - 16, 2 - 20, 3 - 24 MHz
5: Set to ‘1’ and then to ‘0’ to clear internal errors.
4: Clear the interrupt. Should be set to ‘1’ to give a interrupt pulse on each message.
3: Controls the terminal flag bit in the 1553B status word. This can be masked by the "inhibit terminal flag bit" mode code.
2: Controls the subsystem flag bit in the 1553B status word.
1: Controls the busy bit in the 1553B status word.
0: Controls the service request bit in the 1553B status word.

Vector word register

31 16	15 0
RESERVED	vword

Figure 38. Vector word register

[15:0] Used for transmit vector word mode code if extmdata bit is ‘0’ in control register.

Interrupt vector and command value register

31	18 7	6 0
RESERVED	cmdval	intvect

Figure 39. Interrupt vector register

[18:7] For each message the CMDVAL output of Core1553BRT is latched into this register.
18 - broadcast
17 - 1 for transmit, 0 for receive
16:12 - subaddress
11:7 - word count / mode code

[6:0] Shows the value of the interrupt vector output of the Core1553BRT.

AHB page address register

31	12 0
ahbaddr	RESERVED

Figure 40. Address register

[31:12]: Holds the 20 top most bits of the AHB address of the allocated memory area. Resets to the value specified with the ahbaddr VHDL generic.

Interrupt pending/mask register

31	18	17	16		2	1	0
RESERVED	MASK2	MASK1	MASK0	RESERVED	AHBERR	MEMFAIL	RT

Figure 41. Address register

[31:19]:	Reserved.
18:	MASK2 - Interrupt mask for AHBERR interrupt. Interrupt enabled if 1.
17:	MASK1 - Interrupt mask for MEMFAIL interrupt. Interrupt enabled if 1.
18:	MASK0 - Interrupt mask for RT interrupt. Interrupt enabled if 1.
[15:3]:	Reserved.
2 :	AHBERR - 1 if an AHB error has occured
1:	MEMFAIL - 1 if an Core1553RT DMA has not occured in time.
0:	RT - 1 if the Core1553RT has received/transmitted a message.

19.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x071. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

19.6 Configuration options

Table 128 shows the configuration options of the core (VHDL generics).

*Table 128.*Configuration options

Generic	Function	Allowed range	Default
endian	Endianness of the AHB bus (Big = 0)	0 - 1	0
ahbaddr	Reset value for address register	16#00000#-16#FFFFF#	16#00000#
clkspd	Clock speed	0 - 3	1
rtaddr	RT address	0 - 31	0
rtaddrp	RT address parity bit. Set to achieve odd parity.	0 - 1	1
wrtcmd	Write command word to memory	0 - 1	1
wrttsw	Write status word to memory	0 - 1	1
extmdata	Read/write mode code data from/to memory	0 - 1	0
intenbbr	Generate interrupts for bad messages	0 - 1	0
bcasten	Broadcast enable	0 - 1	1
sa30loop	Use sub-address 30 as loopback	0 - 1	0

All VHDL generics except endian are reset values for the corresponding bits in the wrapper control register.

19.7 Signal descriptions

Table 129 shows the interface signals of the core (VHDL ports).

*Table 129.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
B1553I	-	Input	1553 bus input signals	-
	busainp		Positive data input from the A receiver	High
	busainn		Negative data input from the A receiver	Low
	busbinp		Positive data to the B receiver	High
	busbinn		Negative data to the B receiver	Low
B1553O	-	Output	1553 bus output signals	-
	busainen		Enable for the A receiver	High
	busaoutin		Inhibit for the A transmitter	High
	busaoutp		Positive data to the A transmitter	High
	busaoutn		Negative data to the A transmitter	Low
	busbinen		Enable for the B receiver	High
	busboutin		Inhibit for the B transmitter	High
	busboutp		Positive output to the B transmitter	High
	busboutn		Negative output to the B transmitter	Low
RTI	-	Input	RT input signals	-
	cmdok		Command word validation alright	High
	useextok		Enable external command word validation	High
RTO	-	Output	RT output signals	-
	msgstart		Message process started	High
	cmdsync		Start of command word on bus	High
	syncnow		Synchronize received	High
	busreset		Reset command received	High
	cmdval		Active command	-
	cmdokout		Command word validated	High
	cmdstb		Active command value changed	High
	addrlat		Address latch enable	High
	intlat		Interrupt latch enable	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-

* see GRLIB IP Library User’s Manual

19.8 Signal descriptions for underlying GR1553RT core

Table 129 shows the interface signals of the core (VHDL ports).

*Table 130.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
1553 bus input signals				
BUSAINP	N/A	Input	Positive data input from the A receiver	High
BUSAINN	N/A	Input	Negative data input from the A receiver	Low
BUSBINP	N/A	Input	Positive data to the B receiver	High
BUSBINN	N/A	Input	Negative data to the B receiver	Low
1553 bus output signals				
BUSAINEN	N/A	Output	Enable for the A receiver	High
BUSAOUTIN	N/A	Output	Inhibit for the A transmitter	High
BUSAOUTP	N/A	Output	Positive data to the A transmitter	High
BUSAOUTN	N/A	Output	Negative data to the A transmitter	Low
BUSBINEN	N/A	Output	Enable for the B receiver	High
BUSBOUTIN	N/A	Output	Inhibit for the B transmitter	High
BUSBOUTP	N/A	Output	Positive output to the B transmitter	High
BUSBOUTN	N/A	Output	Negative output to the B transmitter	Low
RT input signals				
CMDOK	N/A	Input	Command word validation alright	High
USEEXTOK	N/A	Input	Enable external command word validation	High
RT output signals				
MSGSTART	N/A	Output	Message process started	High
CMDSYNC	N/A	Output	Start of command word on bus	High

*Table 130.*Signal descriptions

Signal name	Field	Type	Function	Active
SYNCNOW	N/A	Output	Synchronize received	High
BUSRESET	N/A	Output	Reset command received	High
CMDVAL	N/A	Output	Active command	-
CMDOKOUT	N/A	Output	Command word validated	High
CMDSTB	N/A	Output	Active command value changed	High
ADDRLAT	N/A	Output	Address latch enable	High
INTLAT	N/A	Output	Interrupt latch enable	High
Interrupt				
INTOUT	N/A	Output	Interrupt	High
AHB signals				
HGRANT	N/A	Input	Bus grant	High
HREADY	N/A	Input	Transfer done	High
HRESP	N/A	Input	Response type	-
HRDATA	N/A	Input	Read data bus	-
HBUSREQ	N/A	Output	Bus request	High
HLOCK	N/A	Output	Lock request	High
HTRANS	N/A	Output	Transfer type	-
HADDR	N/A	Output	Address bus (byte addresses)	-
HWRITE	N/A	Output	Write	High
HSIZE	N/A	Output	Transfer size	-
HBURST	N/A	Output	Burst type	-
HPROT	N/A	Output	Protection control	-
HWDATA	N/A	Output	Write data bus	-
APB signals				
PSEL	N/A	Input	Slave select	High
PENABLE	N/A	Input	Strobe	High
PADDR	N/A	Input	Address bus (byte addresses)	-
PWRITE	N/A	Input	Write	High
PWDATA	N/A	Input	Write data bus	-
PRDATA	N/A	Output	Read data bus	-

19.9 Library dependencies

Table 131 shows libraries that should be used when instantiating the core (VHDL libraries).

*Table 131.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	Signal definitions
GAISLER	B1553	Signals, component	Signal and component declaration

The B1553RT depends on GRLIB, GAISLER, GR1553 and Core1553BRT.

19.10 Library dependencies for underlying GR1553RT core

Table 131 shows libraries that should be used when instantiating the core (VHDL libraries).

*Table 132.*Library dependencies

Library	Package	Imported unit(s)	Description
IEEE	Std_Logic_1164	All	Type declarations

The GR1553RT depends on GR1553 and Core1553BRT.

19.11 Component declaration

The core has the following component declaration.

```
component b1553rt is
    generic (
        hindex    : integer := 0;
        pindex    : integer := 0;
        paddr     : integer := 0;
        pmask     : integer := 16#fff#;
        pirq      : integer := 0;
        ahbaddr   : integer range 0 to 16#FFFFF# := 0;
        clkspd    : integer range 0 to 3 := 1;
        rtaddr    : integer range 0 to 31 := 0;
        rtaddrp   : integer range 0 to 1 := 1;
        wrtcmd    : integer range 0 to 1 := 1;
        wrttsw    : integer range 0 to 1 := 1;
        extmdata  : integer range 0 to 1 := 0;
        intenbbr  : integer range 0 to 1 := 0;
        bcasten   : integer range 0 to 1 := 1;
        sa30loop  : integer range 0 to 1 := 0);
    port (
      rstn      : in    std_ulogic;
      clk       : in    std_ulogic;
      b1553i    : in    b1553_in_type;
      b1553o    : out   b1553_out_type;
      rti       : in    rt1553_in_type;
      rto       : out   rt1553_out_type;
      apbi      : in    apb_slv_in_type;
      apbo      : out   apb_slv_out_type;
      ahbi      : in    ahb_mst_in_type;
      ahbo      : out   ahb_mst_out_type);
  end component;
```

19.12 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.b1553.all;
...
signal bin : b1553_in_type;
signal bout : b1553_out_type;
signal rti : rt1553_in_type;
signal rto : rt1553_out_type;
...
rt : b1553rt
    generic map (hindex => 3, pindex => 13, paddr => 13, pmask => 16#fff#,
pirq => 3, rtaddr => 1, rtaddrp => 0, sa30loop => 1)
    port map (rstn, clkm, bin, bout, rti, rto, apbi, apbo(13), ahbmi, ahbmo(3));

  rti.useextok <= ’0’;
```

20 CAN_OC - GRLIB wrapper for OpenCores CAN Interface core

20.1 Overview

CAN_OC is GRLIB wrapper for the CAN core from Opencores. It provides a bridge between AMBA AHB and the CAN Core registers. The AHB slave interface is mapped in the AHB I/O space using the GRLIB plug&play functionality. The CAN core interrupt is routed to the AHB interrupt bus, and the interrupt number is selected through the *irq* generic. The FIFO RAM in the CAN core is implemented using the GRLIB parametrizable SYNCRAM_2P memories, assuring portability to all supported technologies.

This CAN interface implements the CAN 20.A and 2.0B protocols. It is based on the Philips SJA1000 and has a compatible register map with a few exceptions.

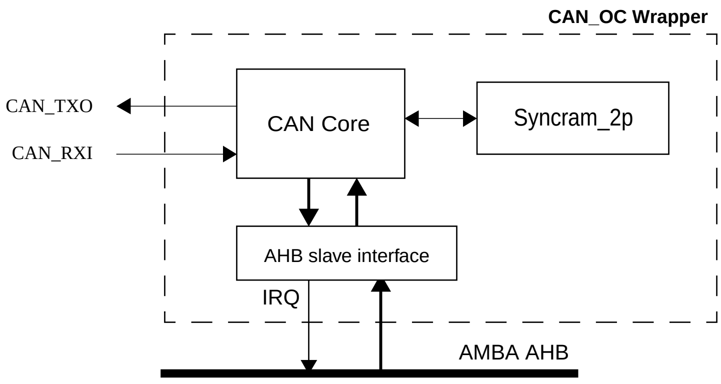


Figure 42. Block diagram

20.2 Opencores CAN controller overview

This CAN controller is based on the Philips SJA1000 and has a compatible register map with a few exceptions. It also supports both BasicCAN (PCA82C200 like) and PeliCAN mode. In PeliCAN mode the extended features of CAN 2.0B is supported. The mode of operation is chosen through the Clock Divider register.

This document will list the registers and their functionality. The Philips SJA1000 data sheet can be used as a reference if something needs clarification. See also the Design considerations chapter for differences between this core and the SJA1000.

The register map and functionality is different between the two modes of operation. First the Basic-CAN mode will be described followed by PeliCAN. Common registers (clock divisor and bus timing) are described in a separate chapter. The register map also differs depending on whether the core is in operating mode or in reset mode. When reset the core starts in reset mode awaiting configuration. Operating mode is entered by clearing the reset request bit in the command register. To re-enter reset mode set this bit high again.

20.3 AHB interface

All registers are one byte wide and the addresses specified in this document are byte addresses. Byte reads and writes should be used when interfacing with this core. The read byte is duplicated on all byte lanes of the AHB bus. The wrapper is big endian so the core expects the MSB at the lowest address.

The bit numbering in this document uses bit 7 as MSB and bit 0 as LSB.

20.4 BasicCAN mode

20.4.1 BasicCAN register map

*Table 133.*BasicCAN address allocation

Address	Operating mode		Reset mode	
	Read	**Write**	**Read**	**Write**
0	Control	Control	Control	Control
1	(0xFF)	Command	(0xFF)	Command
2	Status	-	Status	-
3	Interrupt	-	Interrupt	-
4	(0xFF)	-	Acceptance code	Acceptance code
5	(0xFF)	-	Acceptance mask	Acceptance mask
6	(0xFF)	-	Bus timing 0	Bus timing 0
7	(0xFF)	-	Bus timing 1	Bus timing 1
8	(0x00)	-	(0x00)	-
9	(0x00)	-	(0x00)	-
10	TX id1	TX id1	(0xFF)	-
11	TX id2, rtr, dlc	TX id2, rtr, dlc	(0xFF)	-
12	TX data byte 1	TX data byte 1	(0xFF)	-
13	TX data byte 2	TX data byte 2	(0xFF)	-
14	TX data byte 3	TX data byte 3	(0xFF)	-
15	TX data byte 4	TX data byte 4	(0xFF)	-
16	TX data byte 5	TX data byte 5	(0xFF)	-
17	TX data byte 6	TX data byte 6	(0xFF)	-
18	TX data byte 7	TX data byte 7	(0xFF)	-
19	TX data byte 8	TX data byte 8	(0xFF)	-
20	RX id1	-	RX id1	-
21	RX id2, rtr, dlc	-	RX id2, rtr, dlc	-
22	RX data byte 1	-	RX data byte 1	-
23	RX data byte 2	-	RX data byte 2	-
24	RX data byte 3	-	RX data byte 3	-
25	RX data byte 4	-	RX data byte 4	-
26	RX data byte 5	-	RX data byte 5	-
27	RX data byte 6	-	RX data byte 6	-
28	RX data byte 7	-	RX data byte 7	-
29	RX data byte 8	-	RX data byte 8	-
30	(0x00)	-	(0x00)	-
31	Clock divider	Clock divider	Clock divider	Clock divider

20.4.2 Control register

The control register contains interrupt enable bits as well as the reset request bit.

*Table 134.*Bit interpretation of control register (CR) (address 0)

Bit	**Name**	**Description**
CR.7	-	reserved
CR.6	-	reserved
CR.5	-	reserved
CR.4	Overrun Interrupt Enable	1 - enabled, 0 - disabled
CR.3	Error Interrupt Enable	1 - enabled, 0 - disabled
CR.2	Transmit Interrupt Enable	1 - enabled, 0 - disabled
CR.1	Receive Interrupt Enable	1 - enabled, 0 - disabled
CR.0	Reset request	Writing 1 to this bit aborts any ongoing transfer and enters reset mode. Writing 0 returns to operating mode.

20.4.3 Command register

Writing a one to the corresponding bit in this register initiates an action supported by the core.

*Table 135.*Bit interpretation of command register (CMR) (address 1)

Bit	**Name**	**Description**
CMR.7	-	reserved
CMR.6	-	reserved
CMR.5	-	reserved
CMR.4	-	not used (go to sleep in SJA1000 core)
CMR.3	Clear data overrun	Clear the data overrun status bit
CMR.2	Release receive buffer	Free the current receive buffer for new reception
CMR.1	Abort transmission	Aborts a not yet started transmission.
CMR.0	Transmission request	Starts the transfer of the message in the TX buffer

A transmission is started by writing 1 to CMR.0. It can only be aborted by writing 1 to CMR.1 and only if the transfer has not yet started. If the transmission has started it will not be aborted when setting CMR.1 but it will not be retransmitted if an error occurs.

Giving the Release receive buffer command should be done after reading the contents of the receive buffer in order to release this memory. If there is another message waiting in the FIFO a new receive interrupt will be generated (if enabled) and the receive buffer status bit will be set again.

To clear the Data overrun status bit CMR.3 must be written with 1.

20.4.4 Status register

The status register is read only and reflects the current status of the core.

*Table 136.*Bit interpretation of status register (SR) (address 2)

Bit	Name	Description
SR.7	Bus status	1 when the core is in bus-off and not involved in bus activities
SR.6	Error status	At least one of the error counters have reached or exceeded the CPU warning limit (96).
SR.5	Transmit status	1 when transmitting a message
SR.4	Receive status	1 when receiving a message
SR.3	Transmission complete	1 indicates the last message was successfully transferred.
SR.2	Transmit buffer status	1 means CPU can write into the transmit buffer
SR.1	Data overrun status	1 if a message was lost because no space in fifo.
SR.0	Receive buffer status	1 if messages available in the receive fifo.

Receive buffer status is cleared when the Release receive buffer command is given and set high if there are more messages available in the fifo.

The data overrun status signals that a message which was accepted could not be placed in the fifo because not enough space left. NOTE: This bit differs from the SJA1000 behavior and is set first when the fifo has been read out.

When the transmit buffer status is high the transmit buffer is available to be written into by the CPU. During an on-going transmission the buffer is locked and this bit is 0.

The transmission complete bit is set to 0 when a transmission request has been issued and will not be set to 1 again until a message has successfully been transmitted.

20.4.5 Interrupt register

The interrupt register signals to CPU what caused the interrupt. The interrupt bits are only set if the corresponding interrupt enable bit is set in the control register.

*Table 137.*Bit interpretation of interrupt register (IR) (address 3)

Bit	Name	Description
IR.7	-	reserved
IR.6	-	reserved
IR.5	-	reserved
IR.4	-	not used (wake-up interrupt of SJA1000)
IR.3	Data overrun interrupt	Set when SR.1 goes from 0 to 1.
IR.2	Error interrupt	Set when the error status or bus status are changed.
IR.1	Transmit interrupt	Set when the transmit buffer is released (status bit 0->1)
IR.0	Receive interrupt	This bit is set while there are more messages in the fifo.

This register is reset on read with the exception of IR.0. Note that this differs from the SJA1000 behavior where all bits are reset on read in BasicCAN mode. This core resets the receive interrupt bit when the release receive buffer command is given (like in PeliCAN mode).

Also note that bit IR.5 through IR.7 reads as 1 but IR.4 is 0.

20.4.6 Transmit buffer

The table below shows the layout of the transmit buffer. In BasicCAN only standard frame messages can be transmitted and received (EFF messages on the bus are ignored).

*Table 138.*Transmit buffer layout

Addr	**Name**	**Bits**							
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
10	ID byte 1	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
11	ID byte 2	ID.2	ID.1	ID.0	RTR	DLC.3	DLC.2	DLC.1	DLC.0
12	TX data 1	TX byte 1							
13	TX data 2	TX byte 2							
14	TX data 3	TX byte 3							
15	TX data 4	TX byte 4							
16	TX data 5	TX byte 5							
17	TX data 6	TX byte 6							
18	TX data 7	TX byte 7							
19	TX data 8	TX byte 8							

If the RTR bit is set no data bytes will be sent but DLC is still part of the frame and must be specified according to the requested frame. Note that it is possible to specify a DLC larger than 8 bytes but should not be done for compatibility reasons. If DLC > 8 still only 8 bytes can be sent.

20.4.7 Receive buffer

The receive buffer on address 20 through 29 is the visible part of the 64 byte RX FIFO. Its layout is identical to that of the transmit buffer.

20.4.8 Acceptance filter

Messages can be filtered based on their identifiers using the acceptance code and acceptance mask registers. The top 8 bits of the 11 bit identifier are compared with the acceptance code register only comparing the bits set to zero in the acceptance mask register. If a match is detected the message is stored to the fifo.

20.5 PeliCAN mode

20.5.1 PeliCAN register map

*Table 139.*PeliCAN address allocation

#	Operating mode				Reset mode	
	Read		**Write**		**Read**	**Write**
0	Mode		Mode		Mode	Mode
1	(0x00)		Command		(0x00)	Command
2	Status		-		Status	-
3	Interrupt		-		Interrupt	-
4	Interrupt enable		Interrupt enable		Interrupt enable	Interrupt enable
5	reserved (0x00)		-		reserved (0x00)	-
6	Bus timing 0		-		Bus timing 0	Bus timing 0
7	Bus timing 1		-		Bus timing 1	Bus timing 1
8	(0x00)		-		(0x00)	-
9	(0x00)		-		(0x00)	-
10	reserved (0x00)		-		reserved (0x00)	-
11	Arbitration lost capture		-		Arbitration lost capture	-
12	Error code capture		-		Error code capture	-
13	Error warning limit		-		Error warning limit	Error warning limit
14	RX error counter		-		RX error counter	RX error counter
15	TX error counter		-		TX error counter	TX error counter
16	RX FI SFF	RX FI EFF	TX FI SFF	TX FI EFF	Acceptance code 0	Acceptance code 0
17	RX ID 1	RX ID 1	TX ID 1	TX ID 1	Acceptance code 1	Acceptance code 1
18	RX ID 2	RX ID 2	TX ID 2	TX ID 2	Acceptance code 2	Acceptance code 2
19	RX data 1	RX ID 3	TX data 1	TX ID 3	Acceptance code 3	Acceptance code 3
20	RX data 2	RX ID 4	TX data 2	TX ID 4	Acceptance mask 0	Acceptance mask 0
21	RX data 3	RX data 1	TX data 3	TX data 1	Acceptance mask 1	Acceptance mask 1
22	RX data 4	RX data 2	TX data 4	TX data 2	Acceptance mask 2	Acceptance mask 2
23	RX data 5	RX data 3	TX data 5	TX data 3	Acceptance mask 3	Acceptance mask 3
24	RX data 6	RX data 4	TX data 6	TX data 4	reserved (0x00)	-
25	RX data 7	RX data 5	TX data 7	TX data 5	reserved (0x00)	-
26	RX data 8	RX data 6	TX data 8	TX data 6	reserved (0x00)	-
27	FIFO	RX data 7	-	TX data 7	reserved (0x00)	-
28	FIFO	RX data 8	-	TX data 8	reserved (0x00)	-
29	RX message counter		-		RX msg counter	-
30	(0x00)		-		(0x00)	-
31	Clock divider		Clock divider		Clock divider	Clock divider

The transmit and receive buffers have different layout depending on if standard frame format (SFF) or extended frame format (EFF) is to be transmitted/received. See the specific section below.

20.5.2 Mode register

*Table 140.*Bit interpretation of mode register (MOD) (address 0)

Bit	Name	Description
MOD.7	-	reserved
MOD.6	-	reserved
MOD.5	-	reserved
MOD.4	-	not used (sleep mode in SJA1000)
MOD.3	Acceptance filter mode	1 - single filter mode, 0 - dual filter mode
MOD.2	Self test mode	If set the controller is in self test mode
MOD.1	Listen only mode	If set the controller is in listen only mode
MOD.0	Reset mode	Writing 1 to this bit aborts any ongoing transfer and enters reset mode. Writing 0 returns to operating mode

Writing to MOD.1-3 can only be done when reset mode has been entered previously.

In Listen only mode the core will not send any acknowledgements. Note that unlike the SJA1000 the Opencores core does not become error passive and active error frames are still sent!

When in Self test mode the core can complete a successful transmission without getting an acknowledgement if given the Self reception request command. Note that the core must still be connected to a real bus, it does not do an internal loopback.

20.5.3 Command register

Writing a one to the corresponding bit in this register initiates an action supported by the core.

*Table 141.*Bit interpretation of command register (CMR) (address 1)

Bit	Name	Description
CMR.7	-	reserved
CMR.6	-	reserved
CMR.5	-	reserved
CMR.4	Self reception request	Transmits and simultaneously receives a message
CMR.3	Clear data overrun	Clears the data overrun status bit
CMR.2	Release receive buffer	Free the current receive buffer for new reception
CMR.1	Abort transmission	Aborts a not yet started transmission.
CMR.0	Transmission request	Starts the transfer of the message in the TX buffer

A transmission is started by writing 1 to CMR.0. It can only be aborted by writing 1 to CMR.1 and only if the transfer has not yet started. Setting CMR.0 and CMR.1 simultaneously will result in a so called single shot transfer, i.e. the core will not try to retransmit the message if not successful the first time.

Giving the Release receive buffer command should be done after reading the contents of the receive buffer in order to release this memory. If there is another message waiting in the FIFO a new receive interrupt will be generated (if enabled) and the receive buffer status bit will be set again.

The Self reception request bit together with the self test mode makes it possible to do a self test of the core without any other cores on the bus. A message will simultaneously be transmitted and received and both receive and transmit interrupt will be generated.

20.5.4 Status register

The status register is read only and reflects the current status of the core.

*Table 142.*Bit interpretation of command register (SR) (address 2)

Bit	Name	Description
SR.7	Bus status	1 when the core is in bus-off and not involved in bus activities
SR.6	Error status	At least one of the error counters have reached or exceeded the error warning limit.
SR.5	Transmit status	1 when transmitting a message
SR.4	Receive status	1 when receiving a message
SR.3	Transmission complete	1 indicates the last message was successfully transferred.
SR.2	Transmit buffer status	1 means CPU can write into the transmit buffer
SR.1	Data overrun status	1 if a message was lost because no space in fifo.
SR.0	Receive buffer status	1 if messages available in the receive fifo.

Receive buffer status is cleared when there are no more messages in the fifo. The data overrun status signals that a message which was accepted could not be placed in the fifo because not enough space left. NOTE: This bit differs from the SJA1000 behavior and is set first when the fifo has been read out.

When the transmit buffer status is high the transmit buffer is available to be written into by the CPU. During an on-going transmission the buffer is locked and this bit is 0.

The transmission complete bit is set to 0 when a transmission request or self reception request has been issued and will not be set to 1 again until a message has successfully been transmitted.

20.5.5 Interrupt register

The interrupt register signals to CPU what caused the interrupt. The interrupt bits are only set if the corresponding interrupt enable bit is set in the interrupt enable register.

*Table 143.*Bit interpretation of interrupt register (IR) (address 3)

Bit	Name	Description
IR.7	Bus error interrupt	Set if an error on the bus has been detected
IR.6	Arbitration lost interrupt	Set when the core has lost arbitration
IR.5	Error passive interrupt	Set when the core goes between error active and error passive
IR.4	-	not used (wake-up interrupt of SJA1000)
IR.3	Data overrun interrupt	Set when data overrun status bit is set
IR.2	Error warning interrupt	Set on every change of the error status or bus status
IR.1	Transmit interrupt	Set when the transmit buffer is released
IR.0	Receive interrupt	Set while the fifo is not empty.

This register is reset on read with the exception of IR.0 which is reset when the fifo has been emptied.

20.5.6 Interrupt enable register

In the interrupt enable register the separate interrupt sources can be enabled/disabled. If enabled the corresponding bit in the interrupt register can be set and an interrupt generated.

*Table 144.*Bit interpretation of interrupt enable register (IER) (address 4)

Bit	Name	Description
IR.7	Bus error interrupt	1 - enabled, 0 - disabled
IR.6	Arbitration lost interrupt	1 - enabled, 0 - disabled
IR.5	Error passive interrupt	1 - enabled, 0 - disabled
IR.4	-	not used (wake-up interrupt of SJA1000)
IR.3	Data overrun interrupt	1 - enabled, 0 - disabled
IR.2	Error warning interrupt	1 - enabled, 0 - disabled.
IR.1	Transmit interrupt	1 - enabled, 0 - disabled
IR.0	Receive interrupt	1 - enabled, 0 - disabled

20.5.7 Arbitration lost capture register

*Table 145.*Bit interpretation of arbitration lost capture register (ALC) (address 11)

Bit	Name	Description
ALC.7-5	-	reserved
ALC.4-0	Bit number	Bit where arbitration is lost

When the core loses arbitration the bit position of the bit stream processor is captured into arbitration lost capture register. The register will not change content again until read out.

20.5.8 Error code capture register

*Table 146.*Bit interpretation of error code capture register (ECC) (address 12)

Bit	Name	Description
ECC.7-6	Error code	Error code number
ECC.5	Direction	1 - Reception, 0 - transmission error
ECC.4-0	Segment	Where in the frame the error occurred

When a bus error occurs the error code capture register is set according to what kind of error occurred, if it was while transmitting or receiving and where in the frame it happened. As with the ALC register the ECC register will not change value until it has been read out. The table below shows how to interpret bit 7-6 of ECC.

*Table 147.*Error code interpretation

ECC.7-6	Description
0	Bit error
1	Form error
2	Stuff error
3	Other

Bit 4 downto 0 of the ECC register is interpreted as below

*Table 148.*Bit interpretation of ECC.4-0

ECC.4-0	Description
0x03	Start of frame
0x02	ID.28 - ID.21
0x06	ID.20 - ID.18
0x04	Bit SRTR
0x05	Bit IDE
0x07	ID.17 - ID.13
0x0F	ID.12 - ID.5
0x0E	ID.4 - ID.0
0x0C	Bit RTR
0x0D	Reserved bit 1
0x09	Reserved bit 0
0x0B	Data length code
0x0A	Data field
0x08	CRC sequence
0x18	CRC delimiter
0x19	Acknowledge slot
0x1B	Acknowledge delimiter
0x1A	End of frame
0x12	Intermission
0x11	Active error flag
0x16	Passive error flag
0x13	Tolerate dominant bits
0x17	Error delimiter
0x1C	Overload flag

20.5.9 Error warning limit register

This registers allows for setting the CPU error warning limit. It defaults to 96. Note that this register is only writable in reset mode.

20.5.10 RX error counter register (address 14)

This register shows the value of the rx error counter. It is writable in reset mode. A bus-off event resets this counter to 0.

20.5.11 TX error counter register (address 15)

This register shows the value of the tx error counter. It is writable in reset mode. If a bus-off event occurs this register is initialized as to count down the protocol defined 128 occurrences of the bus-free signal and the status of the bus-off recovery can be read out from this register. The CPU can force a bus-off by writing 255 to this register. Note that unlike the SJA1000 this core will signal bus-off immediately and not first when entering operating mode. The bus-off recovery sequence starts when entering operating mode after writing 255 to this register in reset mode.

20.5.12 Transmit buffer

The transmit buffer is write-only and mapped on address 16 to 28. Reading of this area is mapped to the receive buffer described in the next section. The layout of the transmit buffer depends on whether a standard frame (SFF) or an extended frame (EFF) is to be sent as seen below.

Table 149.

#	Write (SFF)	Write(EFF)
16	TX frame information	TX frame information
17	TX ID 1	TX ID 1
18	TX ID 2	TX ID 2
19	TX data 1	TX ID 3
20	TX data 2	TX ID 4
21	TX data 3	TX data 1
22	TX data 4	TX data 2
23	TX data 5	TX data 3
24	TX data 6	TX data 4
25	TX data 7	TX data 5
26	TX data 8	TX data 6
27	-	TX data 7
28	-	TX data 8

TX frame information (this field has the same layout for both SFF and EFF frames)

*Table 150.*TX frame information address 16

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FF	RTR	-	-	DLC.3	DLC.2	DLC.1	DLC.0

Bit 7 - FF selects the frame format, i.e. whether this is to be interpreted as an extended or standard frame. 1 = EFF, 0 = SFF.
Bit 6 - RTR should be set to 1 for an remote transmission request frame.
Bit 5:4 - are don’t care.
Bit 3:0 - DLC specifies the Data Length Code and should be a value between 0 and 8. If a value greater than 8 is used 8 bytes will be transmitted.

TX identifier 1 (this field is the same for both SFF and EFF frames)

*Table 151.*TX identifier 1 address 17

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Bit 7:0 - The top eight bits of the identifier.

TX identifier 2, SFF frame

*Table 152.*TX identifier 2 address 18

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.20	ID.19	ID.18	-	-	-	-	-

Bit 7:5 - Bottom three bits of an SFF identifier.
Bit 4:0 - Don’t care.

TX identifier 2, EFF frame

*Table 153.*TX identifier 2 address 18

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Bit 7:0 - Bit 20 downto 13 of 29 bit EFF identifier.

TX identifier 3, EFF frame

*Table 154.*TX identifier 3 address 19

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Bit 7:0 - Bit 12 downto 5 of 29 bit EFF identifier.

TX identifier 4, EFF frame

*Table 155.*TX identifier 4 address 20

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
ID.4	ID.3	ID.2	ID.1	ID.0	-	-	-

Bit 7:3 - Bit 4 downto 0 of 29 bit EFF identifier
Bit 2:0 - Don’t care

Data field

For SFF frames the data field is located at address 19 to 26 and for EFF frames at 21 to 28. The data is transmitted starting from the MSB at the lowest address.

20.5.13 Receive buffer

Table 156.

#	Read (SFF)	Read (EFF)
16	RX frame information	RX frame information
17	RX ID 1	RX ID 1
18	RX ID 2	RX ID 2
19	RX data 1	RX ID 3
20	RX data 2	RX ID 4
21	RX data 3	RX data 1
22	RX data 4	RX data 2
23	RX data 5	RX data 3
24	RX data 6	RX data 4
25	RX data 7	RX data 5
26	RX data 8	RX data 6
27	RX FI of next message in fifo	RX data 7
28	RX ID1 of next message in fifo	RX data 8

RX frame information (this field has the same layout for both SFF and EFF frames)

*Table 157.*RX frame information address 16

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FF	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0

Bit 7 - Frame format of received message. 1 = EFF, 0 = SFF.
Bit 6 - 1 if RTR frame.
Bit 5:4 - Always 0.
Bit 3:0 - DLC specifies the Data Length Code.

RX identifier 1(this field is the same for both SFF and EFF frames)

*Table 158.*RX identifier 1 address 17

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Bit 7:0 - The top eight bits of the identifier.

RX identifier 2, SFF frame

*Table 159.*RX identifier 2 address 18

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.20	ID.19	ID.18	RTR	0	0	0	0

Bit 7:5 - Bottom three bits of an SFF identifier.
Bit 4 - 1 if RTR frame.
Bit 3:0 - Always 0.

RX identifier 2, EFF frame

*Table 160.*RX identifier 2 address 18

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Bit 7:0 -	Bit 20 downto 13 of 29 bit EFF identifier.

RX identifier 3, EFF frame

*Table 161.*RX identifier 3 address 19

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Bit 7:0 -	Bit 12 downto 5 of 29 bit EFF identifier.

RX identifier 4, EFF frame

*Table 162.*RX identifier 4 address 20

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.4	ID.3	ID.2	ID.1	ID.0	RTR	0	0

Bit 7:3 -	Bit 4 downto 0 of 29 bit EFF identifier
Bit 2-	1 if RTR frame
Bit 1:0 -	Don’t care

Data field

For received SFF frames the data field is located at address 19 to 26 and for EFF frames at 21 to 28.

20.5.14 Acceptance filter

The acceptance filter can be used to filter out messages not meeting certain demands. If a message is filtered out it will not be put into the receive fifo and the CPU will not have to deal with it.

There are two different filtering modes, single and dual filter. Which one is used is controlled by bit 3 in the mode register. In single filter mode only one 4 byte filter is used. In dual filter two smaller filters are used and if either of these signals a match the message is accepted. Each filter consists of two parts the acceptance code and the acceptance mask. The code registers are used for specifying the pattern to match and the mask registers specify don’t care bits. In total eight registers are used for the acceptance filter as shown in the table below. Note that they are only read/writable in reset mode.

*Table 163.*Acceptance filter registers

Address	Description
16	Acceptance code 0 (ACR0)
17	Acceptance code 1 (ACR1)
18	Acceptance code 2 (ACR2)
19	Acceptance code 3 (ACR3)
20	Acceptance mask 0 (AMR0)
21	Acceptance mask 1 (AMR1)
22	Acceptance mask 2 (AMR2)
23	Acceptance mask 3 (AMR3)

Single filter mode, standard frame

When receiving a standard frame in single filter mode the registers ACR0-3 are compared against the incoming message in the following way:

ACR0.7-0 & ACR1.7-5 are compared to ID.28-18
ACR1.4 is compared to the RTR bit.
ACR1.3-0 are unused.
ACR2 & ACR3 are compared to data byte 1 & 2.

The corresponding bits in the AMR registers selects if the results of the comparison doesn't matter. A set bit in the mask register means don't care.

Single filter mode, extended frame

When receiving an extended frame in single filter mode the registers ACR0-3 are compared against the incoming message in the following way:

ACR0.7-0 & ACR1.7-0 are compared to ID.28-13
ACR2.7-0 & ACR3.7-3 are compared to ID.12-0
ACR3.2 are compared to the RTR bit
ACR3.1-0 are unused.

The corresponding bits in the AMR registers selects if the results of the comparison doesn't matter. A set bit in the mask register means don't care.

Dual filter mode, standard frame

When receiving a standard frame in dual filter mode the registers ACR0-3 are compared against the incoming message in the following way:

Filter 1

ACR0.7-0 & ACR1.7-5 are compared to ID.28-18
ACR1.4 is compared to the RTR bit.
ACR1.3-0 are compared against upper nibble of data byte 1
ACR3.3-0 are compared against lower nibble of data byte 1

Filter 2

ACR2.7-0 & ACR3.7-5 are compared to ID.28-18
ACR3.4 is compared to the RTR bit.

The corresponding bits in the AMR registers selects if the results of the comparison doesn't matter. A set bit in the mask register means don't care.

Dual filter mode, extended frame

When receiving a standard frame in dual filter mode the registers ACR0-3 are compared against the incoming message in the following way:

Filter 1

ACR0.7-0 & ACR1.7-0 are compared to ID.28-13

Filter 2

ACR2.7-0 & ACR3.7-0 are compared to ID.28-13

The corresponding bits in the AMR registers selects if the results of the comparison doesn’t matter. A set bit in the mask register means don’t care.

20.5.15 RX message counter

The RX message counter register at address 29 holds the number of messages currently stored in the receive fifo. The top three bits are always 0.

20.6 Common registers

There are three common registers with the same addresses and the same functionality in both BasiCAN and PeliCAN mode. These are the clock divider register and bus timing register 0 and 1.

20.6.1 Clock divider register

The only real function of this register in the GRLIB version of the Opencores CAN is to choose between PeliCAN and BasiCAN. The clkout output of the Opencore CAN core is not connected and it is its frequency that can be controlled with this register.

*Table 164.*Bit interpretation of clock divider register (CDR) (address 31)

Bit	Name	Description
CDR.7	CAN mode	1 - PeliCAN, 0 - BasiCAN
CDR.6	-	unused (cbp bit of SJA1000)
CDR.5	-	unused (rxinten bit of SJA1000)
CDR.4	-	reserved
CDR.3	Clock off	Disable the clkout output
CDR.2-0	Clock divisor	Frequency selector

20.6.2 Bus timing 0

*Table 165.*Bit interpretation of bus timing 0 register (BTR0) (address 6)

Bit	Name	Description
BTR0.7-6	SJW	Synchronization jump width
BTR0.5-0	BRP	Baud rate prescaler

The CAN core system clock is calculated as:

$t_{scl} = 2*t_{clk}*(BRP+1)$

where t_{clk} is the system clock.

The sync jump width defines how many clock cycles (t_{scl}) a bit period may be adjusted with by one re-synchronization.

20.6.3 Bus timing 1

*Table 166.*Bit interpretation of bus timing 1 register (BTR1) (address 7)

Bit	Name	Description
BTR1.7	SAM	1 - The bus is sampled three times, 0 - single sample point
BTR1.6-4	TSEG2	Time segment 2
BTR1.3-0	TSEG1	Time segment 1

The CAN bus bit period is determined by the CAN system clock and time segment 1 and 2 as shown in the equations below:

$$t_{tseg1} = t_{scl} * (TSEG1+1)$$
$$t_{tseg2} = t_{scl} * (TSEG2+1)$$
$$t_{bit} = t_{tseg1} + t_{tseg2} + t_{scl}$$

The additional t_{scl} term comes from the initial sync segment. Sampling is done between TSEG1 and TSEG2 in the bit period.

20.7 Design considerations

This section lists known differences between this CAN controller and SJA1000 on which is it based:

- All bits related to sleep mode are unavailable
- Output control and test registers do not exist (reads 0x00)
- Clock divisor register bit 6 (CBP) and 5 (RXINTEN) are not implemented
- Overrun irq and status not set until fifo is read out

BasicCAN specific differences:

- The receive irq bit is not reset on read, works like in PeliCAN mode
- Bit CR.6 always reads 0 and is not a flip flop with no effect as in SJA1000

PeliCAN specific differences:

- Writing 256 to tx error counter gives immediate bus-off when still in reset mode
- Read Buffer Start Address register does not exist
- Addresses above 31 are not implemented (i.e. the internal RAM/FIFO access)
- The core transmits active error frames in Listen only mode

20.8 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x019. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

20.9 Configuration options

Table 167 shows the configuration options of the core (VHDL generics).

*Table 167.*Configuration options

Generic	Function	Allowed range	Default
slvndx	AHB slave bus index	0 - NAHBSLV-1	0
ioaddr	The AHB I/O area base address. Compared with bit 19-8 of the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
iomask	The I/O area address mask. Sets the size of the I/O area and the start address together with ioaddr.	0 - 16#FFF#	16#FF0#
irq	Interrupt number	0 - NAHBIRQ-1	0
memtech	Technology to implement on-chip RAM	0	0 - NTECH

20.10 Signal descriptions

Table 168 shows the interface signals of the core (VHDL ports).

*Table 168.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK		Input	AHB clock	
RESETN		Input	Reset	Low
AHBSI	*	Input	AMBA AHB slave inputs	-
AHBSO	*	Input	AMBA AHB slave outputs	
CAN_RXI		Input	CAN receiver input	High
CAN_TXO		Output	CAN transmitter output	High

*1) see AMBA specification

20.11 Library dependencies

Table 169 shows libraries that should be used when instantiating the core.

*Table 169.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	CAN	Component	Component declaration

20.12 Component declaration

```
library grlib;
use grlib.amba.all;
use gaisler.can.all;

component can_oc
   generic (
    slvndx    : integer := 0;
    ioaddr    : integer := 16#000#;
    iomask    : integer := 16#FF0#;
    irq       : integer := 0;
    memtech   : integer := 0);
   port (
      resetn  : in  std_logic;
      clk     : in  std_logic;
```

```
        ahbsi   : in  ahb_slv_in_type;
        ahbso   : out ahb_slv_out_type;
        can_rxi : in  std_logic;
        can_txo : out std_logic
  );
     end component;
```

21 CLKGEN - Clock generation

21.1 Overview

The CLKGEN clock generator implements internal clock generation and buffering.

21.2 Technology specific clock generators

21.2.1 Overview

The core is a wrapper that instantiates technology specific primitives depending on the value of the *tech* VHDL generic. Each supported technology has its own subsection below. Table 170 lists the subsection applicable for each technology setting. The table is arranged after the technology's numerical value in GRLIB. The subsections are ordered in alphabetical order after technology vendor.

*Table 170.*Overview of technology specific clock generator sections

Technology	Numerical value	Comment	Section
inferred	0	Default when no technology specific generator is available.	21.2.2
virtex	1		21.2.12
virtex2	2		21.2.13
memvirage	3	No technology specific clock generator available.	21.2.2
axcel	4		21.2.3
proasic	5		21.2.3
atc18s	6	No technology specific clock generator available.	21.2.2
altera	7		21.2.7
umc	8	No technology specific clock generator available.	21.2.2
rhumc	9		21.2.10
apa3	10		21.2.5
spartan3	11		21.2.11
ihp25	12	No technology specific clock generator available.	21.2.2
rhlib18t	13		21.2.9
virtex4	14		21.2.13
lattice	15	No technology specific clock generator available.	21.2.2
ut25	16	No technology specific clock generator available.	21.2.2
spartan3e	17		21.2.11
peregrine	18	No technology specific clock generator available.	21.2.2
memartisan	19	No technology specific clock generator available.	21.2.2
virtex5	20		21.2.14
custom1	21	No technology specific clock generator available.	21.2.2
ihp25rh	22	No technology specific clock generator available.	21.2.2
stratix1	23		21.2.7
stratix2	24		21.2.7
eclipse	25	No technology specific clock generator available.	21.2.2
stratix3	26		21.2.8
cyclone3	27		21.2.6
memvirage90	28	No technology specific clock generator available.	21.2.2
tsmc90	29	No technology specific clock generator available.	21.2.2
easic90	30		21.2.15
atc18rha	31	No technology specific clock generator available.	21.2.2

*Table 170.*Overview of technology specific clock generator sections

Technology	Numerical value	Comment	Section
smic013	32	No technology specific clock generator available.	21.2.2
tm65gpl	33	No technology specific clock generator available.	21.2.2
axdsp	34		21.2.3
spartan6	35		21.2.11
virtex6	36		21.2.14
actfus	37		21.2.17
stratix4	38		21.2.18
st65lp	39	No technology specific clock generator available.	21.2.2
st65gp	40	No technology specific clock generator available.	21.2.2
easic45	41		21.2.16

21.2.2 Generic technology

This implementation is used when the clock generator does not support instantiation of technology specific primitives or when the inferred technology has been selected.

This implementation connects the input clock, CLKIN or PCICLKIN depending on the *pcien* and *pcisysclk* VHDL generic, to the SDCLK, CLK1XU, and CLK outputs. The CLKN output is driven by the inverted input clock. The PCICLK output is directly driven by PCICLKIN. Both clock lock signals are always driven to ‘1’ and the CLK2X output is always driven to ‘0’.

In simulation, CLK, CLKN and CLK1XU transitions are skewed 1 ns relative to the SDRAM clock output.

21.2.3 ProASIC

Generics used in this technology:	pcisysclk
Instantiated technology primitives:	None
Signals not driven in this technology:	clk4x, clk1xu, clk2xu, clkb, clkc

This technology selection does not instantiate any technology specific primitives. The core’s clock output, CLK, is driven by the CLKIN or PCICLKIN input depending on the value of VHDL generics *pcien* and *pcisysclk*.

The PCICLK is always directly connected to PCICLKIN. Outputs SDCLK, CLKN and CLK2X, are driven to ground. Both clock lock signals, CGO.CLKLOCK and CGO.PCILOCK, are always driven high.

21.2.4 Actel Axcelerator

Generics used in this technology:	pcisysclk, clk_mul, clk_div, pcien, freq
Instantiated technology primitives:	PLL
Signals not driven in this technology:	clk4x, clk1xu, clk2xu, clkb, clkc

This technology selection has two modes. The first one is used if VHDL generics *clk_mul* and *clk_div* are equal and does not instantiate any technology specific primitives. The core’s clock output, CLK, is driven by the CLKIN or PCICLKIN input depending on the value of VHDL generics *pcien* and *pcisysclk*.

The second mode is used if VHDL generics *clk_mul* and *clk_div* are different and instantiates a PLL. The core’s clock output CLK is either driven by the pciclkin input or the main output from the PLL depending on the values of VHDL generics *pcien* and *pcisysclk*. When the PLL drives the CLK output the resulting frequency is the frequency of CLKIN multiplied by the VHDL generic *clk_mul* and divided by the VHDL generic *clk_div*. Clock buffers are not instantiated within the clock generator and has to be done externally.

For both modes the following applies:

The PCICLK is always directly connected to PCICLKIN. Outputs SDCLK, CLKN and CLK2X, are driven to ground. Both clock lock signals, CGO.CLKLOCK and CGO.PCILOCK, are always driven high.

21.2.5 Actel ProASIC3

Generics used in this technology:	clk_mul, clk_div, clk_odiv, pcisysclk, pcien, freq, clkb_odiv, clkc_odiv
Instantiated technology primitives:	PLLINT, PLL
Signals not driven in this technology:	clkn, sdclk, clk2x, clk4x, clk1xu, clk2xu

This technology instantiates a PLL and a PLLINT to generate the main clock. The instantiation of a PLLINT macro allows the PLL reference clock to be driven from an I/O that is routed through the regular FPGA routing fabric. Figure 43 shows the instantiated primitives, the PLL EXTFB input is not shown and the EXTFB port on the instantiated component is always tied to ground. The figure shows which of the core’s output ports that are driven by the PLL. The PCICLOCK will directly connected to PCICLKIN if VHDL generic *pcien* is non-zero, while CGO.PCILOCK is always driven high. The VHDL generics *pcien* and *pcisysclk* are used to select the reference clock. The values driven on the PLL inputs are listed in tables 171 and 172.

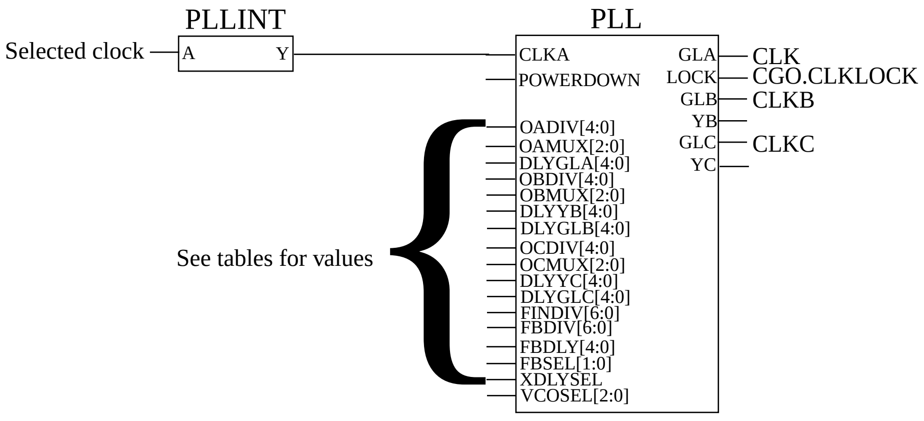


Figure 43. Actel ProASIC3 clock generation

*Table 171.*Constant input signals on Actel ProASIC3 PLL

Signal name	Value	Comment
OADIV[4:0]	VHDL generic *clk_odiv* - 1	Output divider
OAMUX[2:0]	0b100	Post-PLL MUXA
DLYGLA[4:0]	0	Delay on Global A
OBDIV[4:0]	VHDL generic *clkb_odiv* - 1 when *clkb_odiv* > 0, otherwise 0	Output divider
OBMUX[2:0]	0 when VHDL generic *clkb_odiv* = 0, otherwise 0b100	Post-PLL MUXB
DLYYB[4:0]	0	Delay on YB
DLYGLB[4:0]	0	Delay on Global B
OCDIV[4:0]	VHDL generic *clkc_odiv* - 1 when *clkc_odiv* > 0, otherwise 0	Output divider
OCMUX[2:0]	0 when VHDL generic *clkc_odiv* = 0, otherwise 0b100	Post-PLL MUXC
DLYYC[4:0]	0	Delay on YC
DLYGLC[4:0]	0	Delay on Global C
FINDIV[6:0]	VHDL generic *clk_div* - 1	Input divider
FBDIV[6:0]	VHDL generic *clk_mul* - 1	Feedback divider
FBDLY[4:0]	0	Feedback delay
FBSEL[1:0]	0b01	2-bit PLL feedback MUX
XDLYSEL	0	1-bit PLL feedback MUX
VCOSEL[2:0]	*See table 172 below*	VCO gear control. Selects one of four frequency ranges.

The PLL primitive has one parameter, VCOFREQUENCY, which is calculated with:

$$VCOFREQUENCY = \frac{freq \cdot clkmul}{clkdiv} / 1000$$

The calculations are performed with integer precision. This value is also used to determine the value driven on PLL input VCOSEL[2:0]. Table 172 lists the signal value depending on the value of VCOFREQUENCY.

*Table 172.*VCOSEL[2:0] on Actel ProASIC3 PLL

Value of VCOFREQUENCY	Value driven on VCOSEL[2:0]
< 44	0b000
< 88	0b010
< 175	0b100
>= 175	0b110

21.2.6 Altera Cyclone III

Generics used in this technology:	clk_mul, clk_div, sdramen, pcien, pcisysclk, freq, clk2xen
Instantiated technology primitives:	ALTPLL
Signals not driven in this technology:	clk4x, clk1xu, clk2xu, clkb, clkc

This technology instantiates an ALTPLL primitive to generate the required clocks, see figure 44. The ALTPLL attributes are listed in table 173. As can be seen in this table the attributes OPERATION_MODE and COMPENSATE_CLOCK depend on the VHDL generic *sdramen*.

*Table 173.*Altera Cyclone III ALTPLL attributes

Attribute name*	**Value with *sdramen* = 1**	**Value with *sdramen* = 0**
INTENDED_DEVICE_FAMILY	“Cyclone III”	“Cyclone III”
OPERATION_MODE	“ZERO_DELAY_BUFFER”	“NORMAL”
COMPENSATE_CLOCK	“CLK1”	“clock0”
INCLK0_INPUT_FREQUENCY	1000000000 / (VHDL generic *freq*)	1000000000 / (VHDL generic *freq*)
WIDTH_CLOCK	5	5
CLK0_MULTIPLY_BY	VHDL generic *clk_mul*	VHDL generic *clk_mul*
CLK0_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*
CLK1_MULTIPLY_BY	VHDL generic *clk_mul*	VHDL generic *clk_mul*
CLK1_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*
CLK2_MULTIPLY_BY	VHDL generic *clk_mul * 2*	VHDL generic *clk_mul * 2*
CLK2_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*

*Any attributes not listed are assumed to have their default value

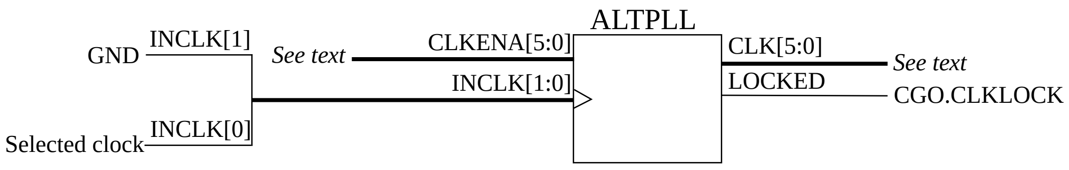


Figure 44. Altera Cyclone III ALTPLL

The value driven on the ALTPLL clock enable signal is dependent on the VHDL generics *clk2xen* and *sdramen*, table 174 lists the effect of these generics.

*Table 174.*Effect of VHDL generics *clk2xen* and *sdramen* on ALTPLL clock enable input

Value of sdramen	**Value of clk2xen**	**Value of CLKENA[5:0]**
0	0	0b000001
0	1	0b000101
1	0	0b000011
1	1	0b000111

Table 175 lists the connections of the core's input and outputs to the ALTPLL ports.

*Table 175.*Connections between core ports and ALTPLL ports

Core signal	**Core direction**	**ALTPLL signal**
CLKIN/PCICLKIN*	Input	INCLK[0]
CLK	Output	CLK[0]
CLKN	Output	$\overline{\text{CLK[0]}}$ (CLK[0] through an inverter)
CLK2X	Output	CLK[2]
SDCLK	Output	CLK[1]
CGO.CLKLOCK	Output	LOCKED

* Depending on VHDL generics PCIEN and PCISYSCLK, as described below.

The clocks can be generated using either the CLKIN input or the PCICLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. If *pcien* is 0 or *pcisysclk* is 0 the input clock to the ALT-PLL will be CLKIN. If *pcien* is non-zero and *pcisysclk* is 1 the input to the ALTPLL will be PCI-CLKIN.

The PCICLK output will connected to the PCICLKIN input if VHDL generic *pcien* is non-zero. Otherwise the PCICLK output will be driven to ground. The CGO.PCILOCK signal is always driven high.

21.2.7 Altera Stratix 1/2

Generics used in this technology:	clk_mul, clk_div, sdramen, pcien, pcisysclk, freq, clk2xen
Instantiated technology primitives:	ALTPLL
Signals not driven in this technology:	clk4x, clk1xu, clk2xu, clkb, clkc

This technology instantiates an ALTPLL primitive to generate the required clocks, see figure 45. The ALTPLL attributes are listed in table 176. As can be seen in this table the OPERATION_MODE attribute depends on the VHDL generic *sdramen*.

*Table 176.*Altera Stratix 1/2 ALTPLL attributes

Attribute name*	**Value with *sdramen* = 1**	**Value with *sdramen* = 0**
OPERATION_MODE	"ZERO_DELAY_BUFFER"	"NORMAL"
INCLK0_INPUT_FREQUENCY	1000000000 / (VHDL generic *freq*)	1000000000 / (VHDL generic *freq*)
WIDTH_CLOCK	6	6
CLK0_MULTIPLY_BY	VHDL generic *clk_mul*	VHDL generic *clk_mul*
CLK0_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*
CLK1_MULTIPLY_BY	VHDL generic *clk_mul* * 2	VHDL generic *clk_mul* * 2
CLK1_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*
EXTCLK0_MULTIPLY_BY	VHDL generic *clk_mul*	VHDL generic *clk_mul*
EXTCLK0_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*

*Any attributes not listed are assumed to have their default value

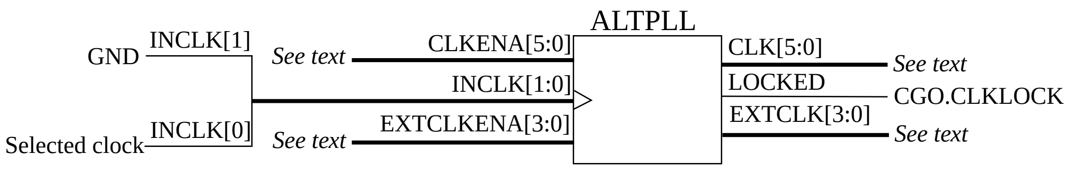


Figure 45. Altera Stratix 1/2 ALTPLL

The values driven on the ALTPLL clock enable signals are dependent on the VHDL generic *clk2xen*, table 177 lists the effect of *clk2xen*.

*Table 177.*Effect of VHDL generic *clk2xen* on ALTPLL clock enable inputs

Signal	**Value with *clk2xen* = 0**	**Value with *clk2xen* /= 0**
CLKENA[5:0]	0b000001	0b000011
EXTCLKENA[3:0]	0b0001	0b0011

Table 178 lists the connections of the core’s input and outputs to the ALTPLL ports.

*Table 178.*Connections between core ports and ALTPLL ports

Core signal	**Core direction**	**ALTPLL signal**
CLKIN/PCICLKIN*	Input	INCLK[0]
CLK	Output	CLK[0]
CLKN	Output	$\overline{\text{CLK[0]}}$ (CLK[0] through an inverter)
CLK2X	Output	CLK[1]
SDCLK	Output	EXTCLK[0]
CGO.CLKLOCK	Output	LOCKED

* Depending on VHDL generics PCIEN and PCISYSCLK, as described below.

The clocks can be generated using either the CLKIN input or the PCICLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. If *pcien* is 0 or *pcisysclk* is 0 the input clock to the ALTPLL will be CLKIN. If *pcien* is non-zero and *pcisysclk* is 1 the input to the ALTPLL will be PCICLKIN.

The PCICLK output will connected to the PCICLKIN input if VHDL generic *pcien* is non-zero. Otherwise the PCICLK output will be driven to ground. The CGO.PCILOCK signal is always driven high.

21.2.8 Altera Stratix 3

This technology is not fully supported at this time.

21.2.9 RHLIB18t

Generics used in this technology:	clk_mul, clk_div
Instantiated technology primitives:	lfdll_top
Signals not driven in this technology:	-

Please contact Aeroflex Gaisler for information concerning the use of this clock generator.

21.2.10 RHUMC

Generics used in this technology:	None
Instantiated technology primitives:	pll_ip
Signals not driven in this technology:	-

Please contact Aeroflex Gaisler for information concerning the use of this clock generator.

21.2.11 Xilinx Spartan 3/3e/6

Generics used in this technology:	clk_mul, clk_div, sdramen, noclkfb, pcien, pcidll, pcisysclk, freq, clk2xen, clksel
Instantiated technology primitives:	BUFG, BUFMUX, DCM, BUFGDLL
Signals not driven in this technology:	clk4x, clkb, clkc

The main clock is generated with a DCM which is instantiated with the attributes listed in table 179. The input clock source connected to the CLKIN input is either the core's CLKIN input or the PCI-CLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. The main DCM's connections is shown in figure 46.

*Table 179.*Spartan 3/e DCM attributes

Attribute name*	**Value**
CLKDV_DIVIDE	2.0
CLKFX_DIVIDE	Determined by core's VHDL generic *clk_div*
CLKFX_MULTIPLY	Determined by core's VHDL generic *clk_mul*
CLKIN_DIVIDE_BY_2	false
CLKIN_PERIOD	10.0
CLKOUT_PHASE_SHIFT	"NONE"
CLK_FEEDBACK	"2X"
DESKEW_ADJUST	"SYSTEM_SYNCHRONOUS"
DFS_FREQUENCY_MODE	"LOW"
DLL_FREQUENCY_MODE	"LOW"
DSS_MODE	"NONE"
DUTY_CYCLE_CORRECTION	true
FACTORY_JF	X"C080"
PHASE_SHIFT	0
STARTUP_WAIT	false

*Any attributes not listed are assumed to have their default value

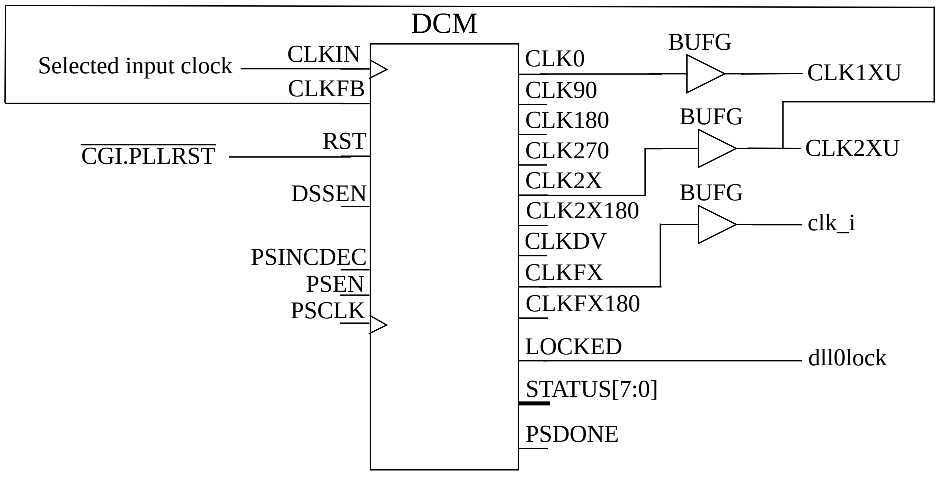


Figure 46. Spartan 3/e generation of main clock

If the VHDL generic *clk2xen* is non-zero the DCM shown in figure 47 is instantiated. The attributes of this DCM are the same as in table 179, except that the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes are both set to 2 and the CLK_FEEDBACK attribute is set to "1X". The dll0lock signal is connected to the LOCKED output of the main clock DCM. When this signal is low all the bits in the shift register connected to the CLK2X DCM's RST input are set to '1'. When the dll0lock signal is asserted it will take four main clock cycles until the RST input is deasserted. Depending on the value of the *clksel* VHDL generic the core's CLK2X output is either driven by a BUFG or a BUFGMUX. Figure 48 shows the two alternatives and how the CGI.CLKSEL(0) input is used to selected between the CLK0 and CLK2X output of the CLK2X DCM.

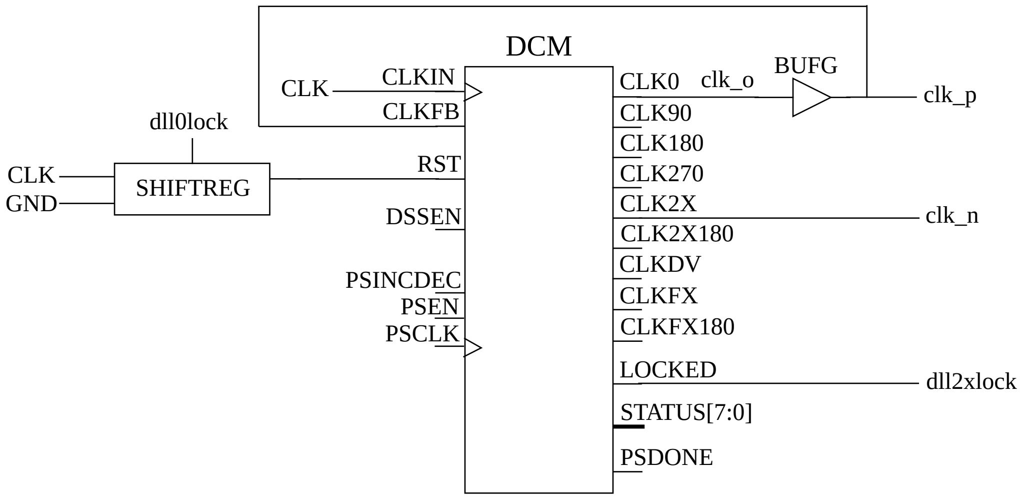


Figure 47. Spartan 3/e generation of CLK2X clock when VHDL generic *clk2xen* is non-zero

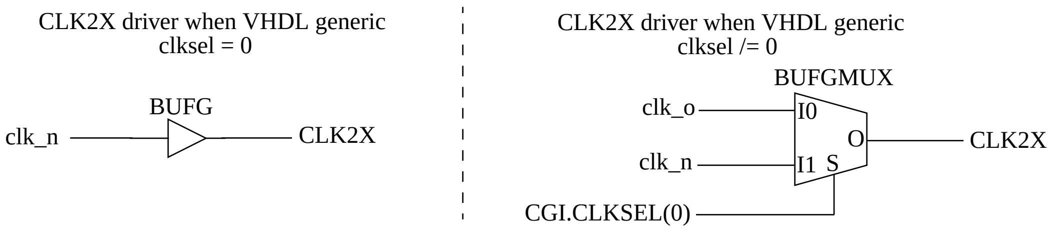

CLK2X driver when VHDL generic clksel = 0
BUFG
clk_n
CLK2X
CLK2X driver when VHDL generic clksel /= 0
BUFGMUX
clk_o
I0
clk_n
I1
S
O
CLK2X
CGI.CLKSEL(0)

Figure 48. Spartan 3/e selection of CLK2X clock when VHDL generic *clk2xen* is non-zero

The value of the *clk2xen* VHDL generic also decides which output that drives the core's CLK output. If the VHDL generic is non-zero the CLK output is driven by the clk_p signal originating from the CLK2X DCM. Otherwise the CLK output is connected to the clk_i signal originating from the main clock DCM. The core's CLKN output is driven by the selected signal through an inverter. Figure 49 illustrates the connections.

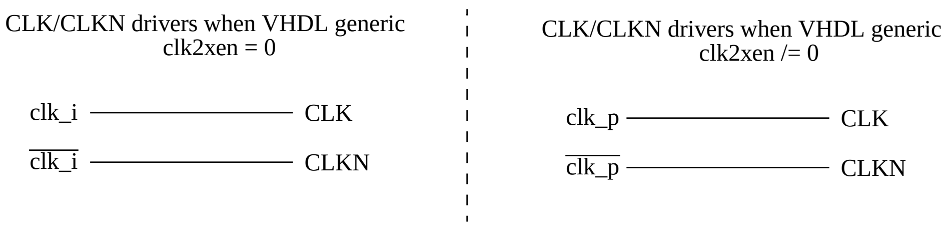

CLK/CLKN drivers when VHDL generic clk2xen = 0
clk_i
CLK
$\overline{\text{clk\_i}}$
CLKN
CLK/CLKN drivers when VHDL generic clk2xen /= 0
clk_p
CLK
$\overline{\text{clk\_p}}$
CLKN

Figure 49. Spartan 3/e clock generator outputs CLK and CLKN

If the VHDL generic *clk2xen* is zero the dll0lock signal from the main clock DCM is either connected to the SDRAM DCM, described below, or if the SDRAM DCM is non-existent, to the core's CGO.CLKLOCK output. This setting also leads to the core's CLK2X output being driven by the main clock DCM's CLK2X output via a BUFG, please see figure 50.

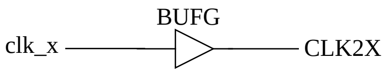


Figure 50. Spartan 3/e generation of CLK2X clock when VHDL generic *clk2xen* is zero

If the SDRAM clock is enabled, via the *sdramen* VHDL generic, and the clock generator is configured to use clock feedback the DCM shown in figure 51 is instantiated. This DCM has the same attributes as the CLK2X DCM. The input to the SDRAM DCM input clock is determined via the *clk2xen* VHDL generic. If the VHDL generic is set to 0 the input is the main CLK, if the generic is set to 1 the input is the clk_p out of the CLK2X DCM shown in figure 48. If the *clk2xen* VHDL generic is set to 2 the clock input to the SDRAM DCM depends on the *clksel* VHDL generic. The input in this last case is the CLK2X output shown in figure 50.

If the CLK2X DCM has been instantiated the SDRAM DCM RST input depends on the LOCKED output of the CLK2X DCM. If the CLK2X DCM has not been instantiated the SDRAM DCM RST input depends on the LOCKED output from the main clock DCM. The applicable LOCKED signal is utilized to keep the SDRAM DCM in reset until its input clock has been stabilized. This is done with a shift register with the same method used for the CLK2X DCM RST.

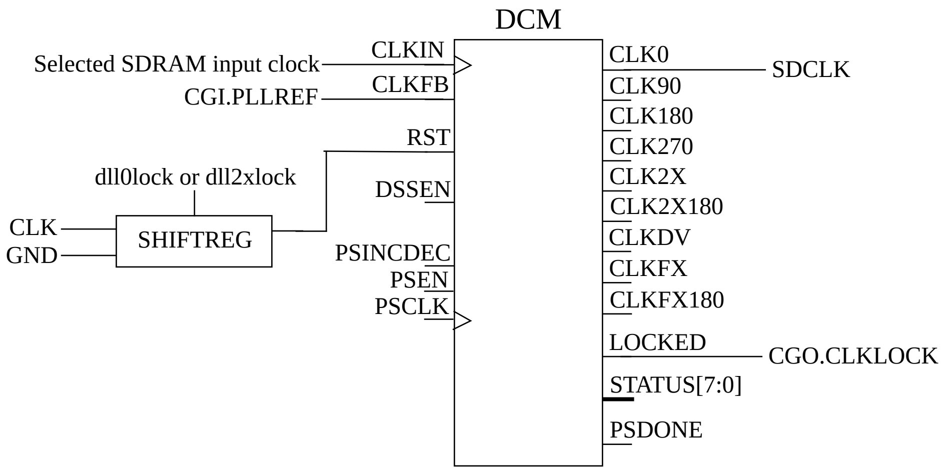


Figure 51. Spartan 3/e generation of SDRAM clock

If the SDRAM clock is disabled (*sdramen* VHDL generic set to 0) or the core has been configured not to use clock feedback (*noclockfb* VHDL generic set to 1) the driver of the core's SDCLK output is determined by the value of the *clk2xen* VHDL generic. If the *clk2xen* VHDL generic is set to 2, the SDRAM clock output is the same as the CLK2X output shown in figure 48, in other words it also depends on the *clksel* VHDL generic. If the *clk2xen* VHDL generic has any other value the SDCLK output is the same as the core's CLK output.

When the *sdramen* VHDL generic is set to 0 the core's CGO.CLKLOCK output is connected to the CLK2X DCM's LOCKED output, if the DCM exists, otherwise the CGO.CLKLOCK output is connected to the main clock DCM's LOCKED output.

If PCI clock generation is enabled via the *pcien* VHDL generic the core instantiates either a BUFG or a BUFGDLL as depicted in figure 52 below. Note that the PCI clock must be enabled if the main clock is to be driven by the PCICLKIN input. If the PCI clock is disabled the PCICLK output is driven to zero. The CGO.PCILOCK output is always driven high in all configurations.

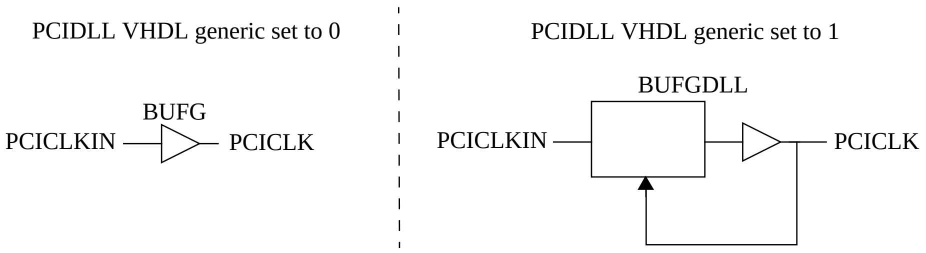


Figure 52. Spartan 3/e PCI clock generation

21.2.12 Xilinx Virtex

Generics used in this technology:	clk_mul, clk_div, sdramen, noclkfb, pcien, pcidll, pcisysclk
Instantiated technology primitives:	BUFG, BUFGDLL, CLKDLL
Signals not driven in this technology:	clk4x, clk1xu, clk2xu, clkb, clkc

The main clock is generated with the help of a CLKDLL. Figure 53 below shows how the CLKDLL primitive is connected. The input clock source is either the core’s CLKIN input or the PCICLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. The figure shows three potential drivers of the BUFG driving the output clock CLK, the driver is selected via the VHDL generics *clk_mul* and *clk_div*. If *clk_mul/clk_div* is equal to 2 the CLK2X output is selected, if *clk_div/clk_mul* equals 2 the CLKDV output is selected, otherwise the CLK0 output drives the BUFG. The inverted main clock output, CLKN, is the BUFG output connected via an inverter.

The figure shows a dashed line connecting the CLKDLL’s LOCKED output to the core output CGO.CLKLOCK. The driver of the CGO.CLKLOCK output depends on the instantiation of a CLKDLL for the SDRAM clock. See description of the SDRAM clock below.

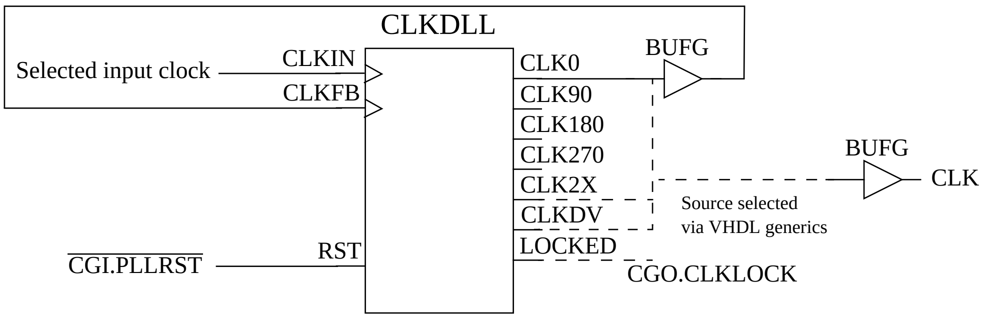


Figure 53. Virtex generation of main clock

If the SDRAM clock is enabled, via the *sdramen* VHDL generic, and the clock generator is configured to use clock feedback, VHDL generic *noclkfb* set to 0, a CLKDLL is instantiated as depicted in figure 54. Note how the CLKDLL’s RST input is connected via a shift register clocked by the main clock. The shift register is loaded with all ‘1’ when the LOCKED signal of the main clock CLKDLL is low. When the LOCKED signal from the main clock CLKDLL is asserted the SDRAM CLKDLL’s RST input will be deasserted after four main clock cycles.

For all other configurations the SDRAM clock is driven by the main clock and the CGO.CLKLOCK signal is driven by the main clock CLKDLL’s LOCKED output. The SDRAM CLKDLL must be present if the core’s CLK2X output shall be driven.

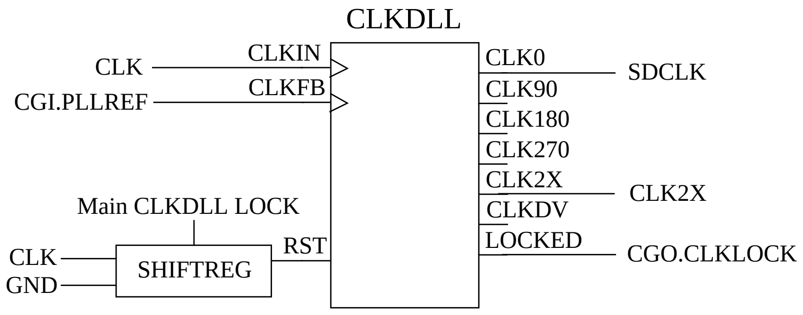


Figure 54. Virtex generation of SDRAM clock with feedback clock enabled

If PCI clock generation is enabled via the *pcien* VHDL generic the core instantiates either a BUFG or a BUFGDLL as depicted in figure 55 below. Note that the PCI clock must be enabled if the main clock is to be driven by the PCICLKIN input. If the PCI clock is disabled the PCICLK output is driven to zero. The CGO.PCILOCK output is always driven high in all configurations.

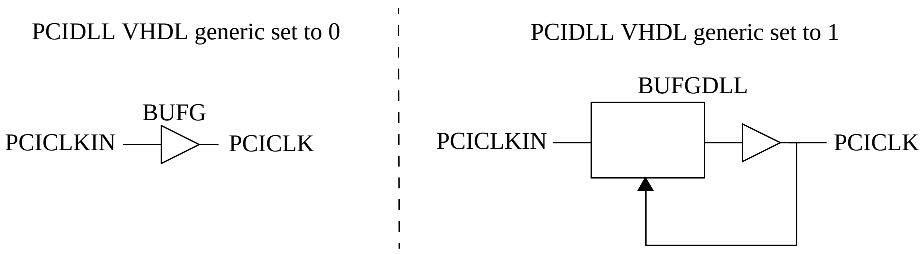


Figure 55. Virtex PCI clock generation

21.2.13 Xilinx Virtex 2/4

Generics used in this technology:	clk_mul, clk_div, sdramen, noclkfb, pcien, pcidll, pcisysclk, freq, clk2xen, clksel
Instantiated technology primitives:	BUFG, BUFMUX, DCM, BUFGDLL
Signals not driven in this technology:	clk4x, clkb, clkc

The main clock is generated with a DCM which is instantiated with the attributes listed in table 180. The input clock source connected to the CLKIN input is either the core’s CLKIN input or the PCI-CLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. The main DCM’s connections is shown in figure 56.

*Table 180.*Virtex 2/4 DCM attributes

Attribute name*	**Value**
CLKDV_DIVIDE	2.0
CLKFX_DIVIDE	Determined by core’s VHDL generic *clk_div*
CLKFX_MULTIPLY	Determined by core’s VHDL generic *clk_mul*
CLKIN_DIVIDE_BY_2	false
CLKIN_PERIOD	10.0
CLKOUT_PHASE_SHIFT	“NONE”
CLK_FEEDBACK	“1X”
DESKEW_ADJUST	“SYSTEM_SYNCHRONOUS”
DFS_FREQUENCY_MODE	“LOW”
DLL_FREQUENCY_MODE	“LOW”
DSS_MODE	“NONE”
DUTY_CYCLE_CORRECTION	true
FACTORY_JF	X”C080”
PHASE_SHIFT	0
STARTUP_WAIT	false

*Any attributes not listed are assumed to have their default value

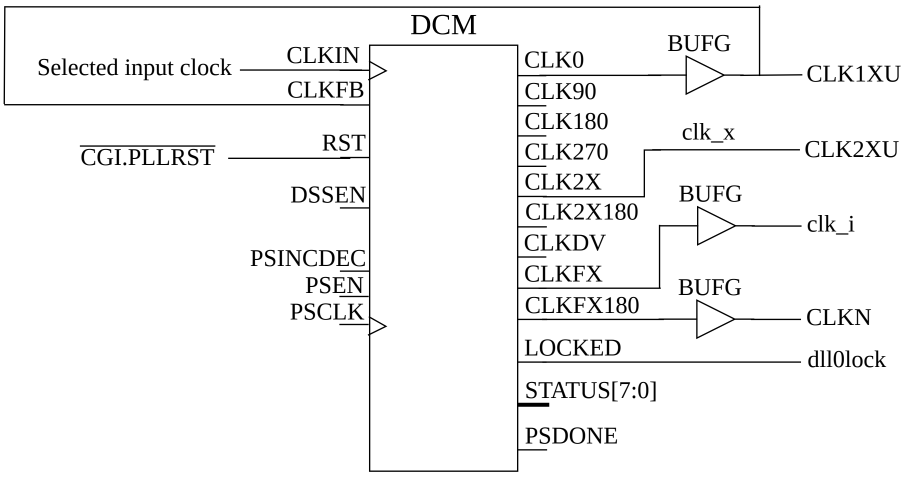


Figure 56. Virtex 2/4 generation of main clock

If the VHDL generic *clk2xen* is non-zero the DCM shown in figure 57 is instantiated. The attributes of this DCM are the same as in table 180, except that the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes are both set to 2. The dll0lock signal is connected to the LOCKED output of the main clock DCM. When this signal is low all the bits in the shift register connected to the CLK2X DCM’s RST input are set to ‘1’. When the dll0lock signal is asserted it will take four main clock cycles until the RST input is deasserted. Depending on the value of the *clksel* VHDL generic the core’s CLK2X output is either driven by a BUFG or a BUFGMUX. Figure 58 shows the two alternatives and how the CGI.CLKSEL(0) input is used to selected between the CLK0 and CLK2X output of the CLK2X DCM.

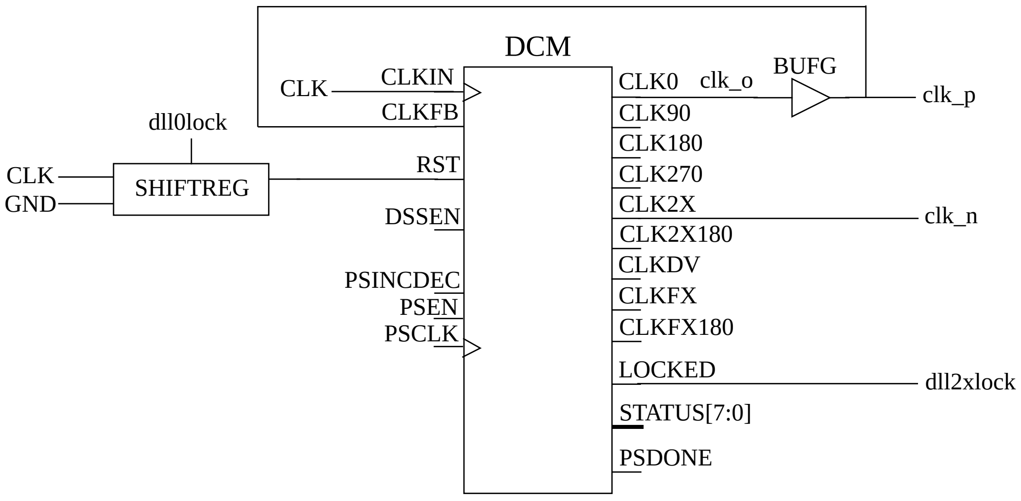


Figure 57. Virtex 2/4 generation of CLK2X clock when VHDL generic *clk2xen* is non-zero

The value of the *clk2xen* VHDL generic also decides which output that drives the core’s CLK output. If the VHDL generic is non-zero the CLK output is driven by the clk_p signal originating from the CLK2X DCM. Otherwise the CLK output is connected to the clk_i signal originating from the main clock DCM. Note that the CLKN output always originates from the main clock DCM, as shown in figure 56.

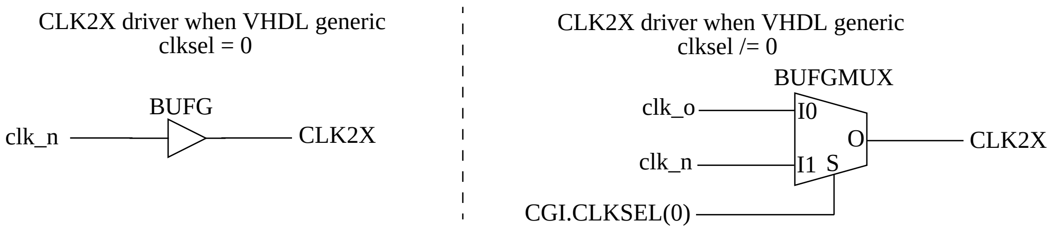

CLK2X driver when VHDL generic clksel = 0
BUFG
clk_n
CLK2X
CLK2X driver when VHDL generic clksel /= 0
BUFGMUX
clk_o
I0
O
CLK2X
clk_n
I1 S
CGI.CLKSEL(0)

Figure 58. Virtex 2/4 selection of CLK2X clock when VHDL generic *clk2xen* is non-zero

If the VHDL generic *clk2xen* is zero the dll0lock signal from the main clock DCM is either connected to the SDRAM DCM, described below, or if the SDRAM DCM is non-existent, to the core’s CGO.CLKLOCK output. This setting also leads to the core’s CLK2X output being driven by the main clock DCM’s CLK2X output via a BUFG, please see figure 59.

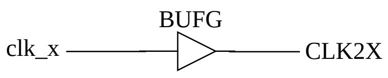


Figure 59. Virtex 2/4 generation of CLK2X clock when VHDL generic *clk2xen* is zero

If the SDRAM clock is enabled, via the *sdramen* VHDL generic, and the clock generator is configured to use clock feedback the DCM shown in figure 60. The input to the SDRAM DCM input clock is determined via the *clk2xen* VHDL generic. If the VHDL generic is set to 0 the input is the main CLK, if the generic is set to 1 the input is the clk_p out of the CLK2X DCM shown in figure 57. If the *clk2xen* VHDL generic is set to 2 the clock input to the SDRAM DCM depends on the *clksel* VHDL generic. The input in this last case is the CLK2X output shown in figure 58.

If the CLK2X DCM has been instantiated the SDRAM DCM RST input depends on the LOCKED output of the CLK2X DCM. If the CLK2X DCM has not been instantiated the SDRAM DCM RST input depends on the LOCKED output from the main clock DCM. The applicable LOCKED signal is utilized to keep the SDRAM DCM in reset until its input clock has been stabilized. This is done with a shift register with the same method used for the CLK2X DCM RST.

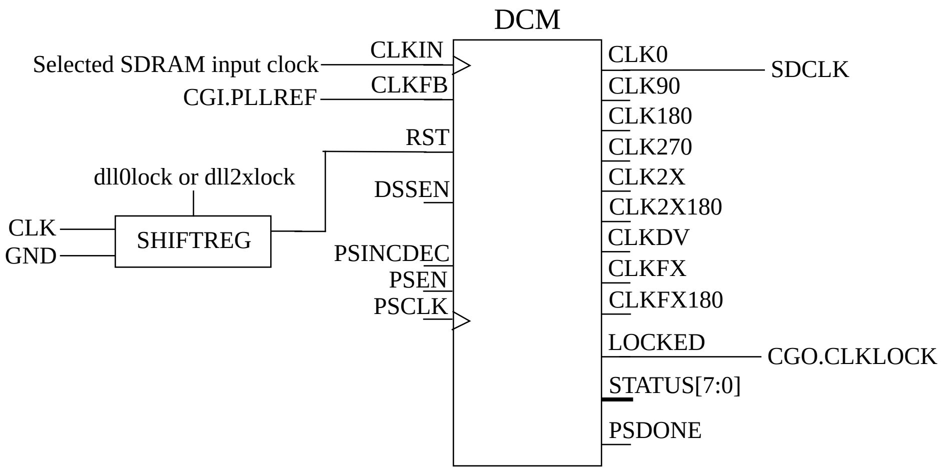


Figure 60. Virtex 2/4 generation of SDRAM clock

If the SDRAM clock is disabled (*sdramen* VHDL generic set to 0) or the core has been configured not to use clock feedback (*noclockfb* VHDL generic set to 1) the driver of the core's SDCLK output is determined by the value of the *clk2xen* VHDL generic. If the clk2xen VHDL generic is set to 2, the SDRAM clock output is the same as the CLK2X output shown in figure 58, in other words it also depends on the *clksel* VHDL generic. If the *clk2xen* VHDL generic has any other value the SDCLK output is the same as the core's CLK output.

When the *sdramen* VHDL generic is set to 0 the core's CGO.CLKLOCK output is connected to the CLK2X DCM's LOCKED output, if the DCM exists, otherwise the CGO.CLKLOCK output is connected to the main clock DCM's LOCKED output.

If PCI clock generation is enabled via the *pcien* VHDL generic the core instantiates either a BUFG or a BUFGDLL as depicted in figure 61 below. Note that the PCI clock must be enabled if the main clock is to be driven by the PCICLKIN input. If the PCI clock is disabled the PCICLK output is driven to zero. The CGO.PCILOCK output is always driven high in all configurations.

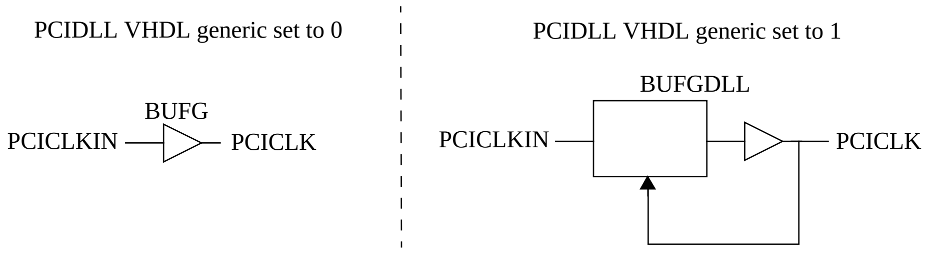


Figure 61. Virtex 2/4 PCI clock generation

21.2.14 Xilinx Virtex 5/6

Generics used in this technology:	clk_mul, clk_div, sdramen, noclkfb, pcien, pcidll, pcisysclk, freq, clk2xen, clksel
Instantiated technology primitives:	BUFG, BUFMUX, DCM, BUFGDLL
Signals not driven in this technology:	clk4x, clkb, clkc

The main clock is generated with a DCM which is instantiated with the attributes listed in table 181. The input clock source connected to the CLKIN input is either the core's CLKIN input or the PCI-

CLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. The main DCM's connections is shown in figure 62.

*Table 181.*Virtex 5 DCM attributes

Attribute name*	**Value**
CLKDV_DIVIDE	2.0
CLKFX_DIVIDE	Determined by core's VHDL generic *clk_div*
CLKFX_MULTIPLY	Determined by core's VHDL generic *clk_mul*
CLKIN_DIVIDE_BY_2	false
CLKIN_PERIOD	10.0
CLKOUT_PHASE_SHIFT	"NONE"
CLK_FEEDBACK	"1X"
DESKEW_ADJUST	"SYSTEM_SYNCHRONOUS"
DFS_FREQUENCY_MODE	"LOW"
DLL_FREQUENCY_MODE	"LOW"
DSS_MODE	"NONE"
DUTY_CYCLE_CORRECTION	true
FACTORY_JF	X"C080"
PHASE_SHIFT	0
STARTUP_WAIT	false

*Any attributes not listed are assumed to have their default value

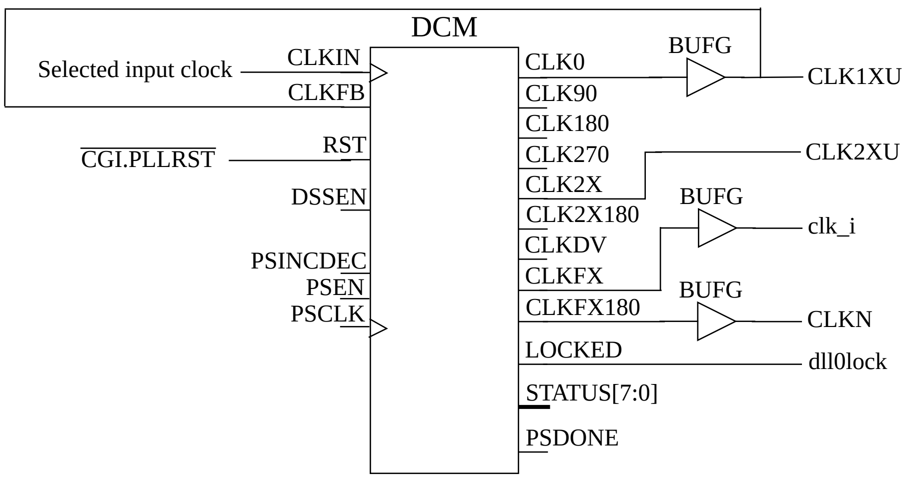


Figure 62. Virtex 5 generation of main clock

If the VHDL generic *clk2xen* is non-zero the DCM shown in figure 63 is instantiated. The attributes of this DCM are the same as in table 181, except that the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes are both set to 2. The dll0lock signal is connected to the LOCKED output of the main clock DCM. When this signal is low all the bits in the shift register connected to the CLK2X DCM's RST input are set to '1'. When the dll0lock signal is asserted it will take four main clock cycles until the RST input is deasserted. Depending on the value of the *clksel* VHDL generic the core's CLK2X output is either driven by a BUFG or a BUFGMUX. Figure 64 shows the two alternatives and how the CGI.CLKSEL(0) input is used to selected between the CLK0 and CLK2X output of the CLK2X DCM.

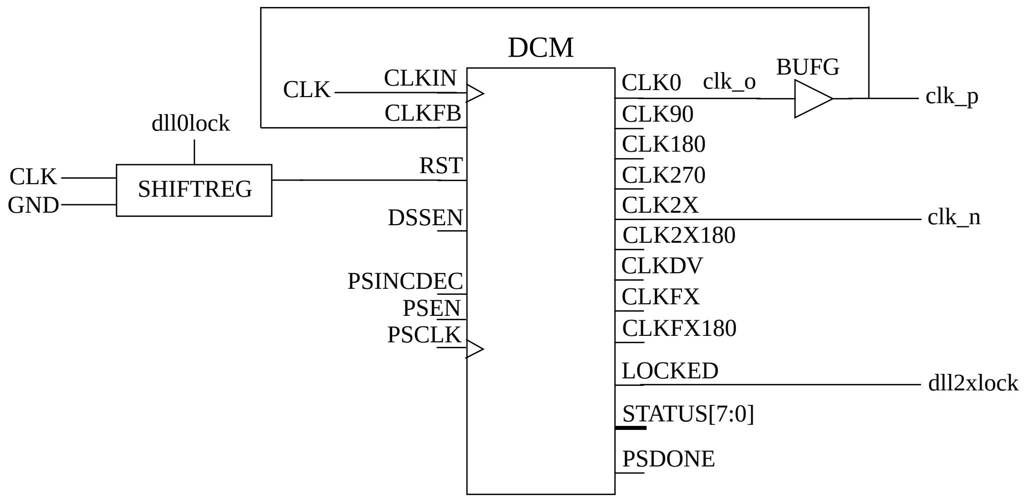


Figure 63. Virtex 5 generation of CLK2X clock when VHDL generic *clk2xen* is non-zero

The value of the *clk2xen* VHDL generic also decides which output that drives the core's CLK output. If the VHDL generic is non-zero the CLK output is driven by the clk_p signal originating from the CLK2X DCM. Otherwise the CLK output is connected to the clk_i signal originating from the main clock DCM. Note that the CLKN output always originates from the main clock DCM, as shown in figure 62.

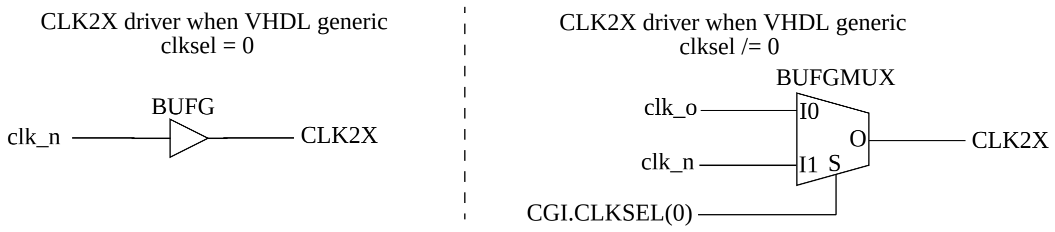

CLK2X driver when VHDL generic clksel = 0
BUFG
clk_n
CLK2X
CLK2X driver when VHDL generic clksel /= 0
BUFGMUX
clk_o
I0
O
clk_n
I1 S
CLK2X
CGI.CLKSEL(0)

Figure 64. Virtex 5 selection of CLK2X clock when VHDL generic *clk2xen* is non-zero

If the VHDL generic *clk2xen* is zero the dll0lock signal from the main clock DCM is either connected to the SDRAM DCM, described below, or if the SDRAM DCM is non-existent, to the core's CGO.CLKLOCK output. This setting also leads to the core's CLK2X output being driven directly by the main clock DCM's CLK2X output.

If the SDRAM clock is enabled, via the *sdramen* VHDL generic, and the clock generator is configured to use clock feedback the DCM shown in figure 65. This DCM has the same attributes as the main clock DCM described in table 181, with the exceptions that CLKFX_MULTIPLY and CLKFX_DIVIDE are both set to 2 and DESKEW_ADJUST is set to "SOURCE_SYNCHRONOUS".

The input to the SDRAM DCM input clock is determined via the *clk2xen* VHDL generic. If the VHDL generic is set to 0 the input is the main CLK, if the generic is set to 1 the input is the clk_p out of the CLK2X DCM shown in figure 57. If the *clk2xen* VHDL generic is set to 2 the clock input to the SDRAM DCM depends on the *clksel* VHDL generic. The input in this last case is the CLK2X output shown in figure 64.

If the CLK2X DCM has been instantiated the SDRAM DCM RST input depends on the LOCKED output of the CLK2X DCM. If the CLK2X DCM has not been instantiated the SDRAM DCM RST input depends on the LOCKED output from the main clock DCM. The applicable LOCKED signal is utilized to keep the SDRAM DCM in reset until its input clock has been stabilized. This is done with a shift register with the same method used for the CLK2X DCM RST.

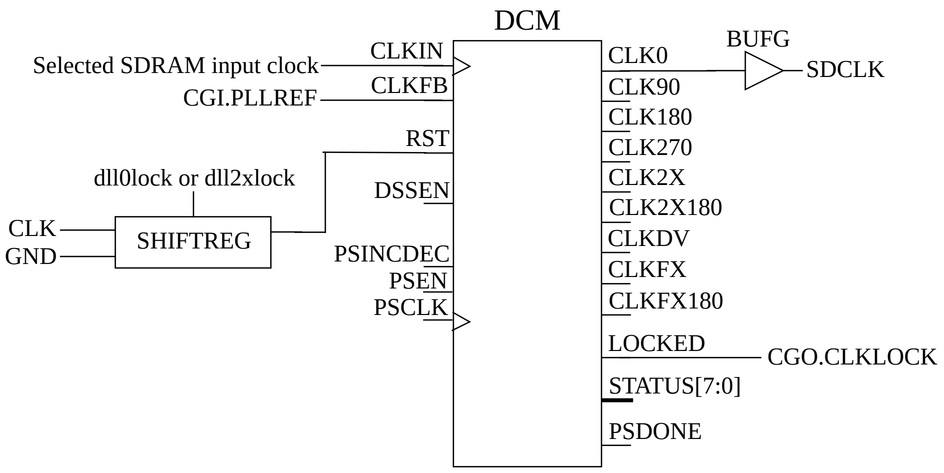


Figure 65. Virtex 5 generation of SDRAM clock

If the SDRAM clock is disabled (*sdramen* VHDL generic set to 0) or the core has been configured not to use clock feedback (*noclockfb* VHDL generic set to 1) the driver of the core’s SDCLK output is determined by the value of the *clk2xen* VHDL generic. If the clk2xen VHDL generic is set to 2, the SDRAM clock output is the same as the CLK2X output shown in figure 64, in other words it also depends on the *clksel* VHDL generic. If the *clk2xen* VHDL generic has any other value the SDCLK output is the same as the core’s CLK output.

When the *sdramen* VHDL generic is set to 0 the core’s CGO.CLKLOCK output is connected to the CLK2X DCM’s LOCKED output, if the DCM exists, otherwise the CGO.CLKLOCK output is connected to the main clock DCM’s LOCKED output.

If PCI clock generation is enabled via the *pcien* VHDL generic the core instantiates either a BUFG or a BUFGDLL as depicted in figure 66 below. Note that the PCI clock must be enabled if the main clock is to be driven by the PCICLKIN input. If the PCI clock is disabled the PCICLK output is driven to zero. The CGO.PCILOCK output is always driven high in all configurations.

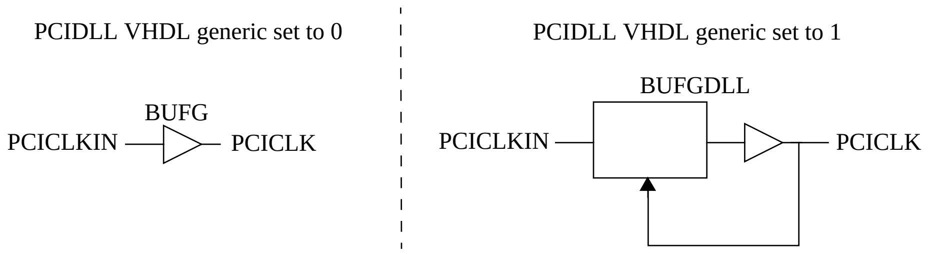


Figure 66. Virtex 5 PCI clock generation

21.2.15 eASIC90 (Nextreme)

Generics used in this technology:	clk_mul, clk_div, freq, pcisysclk, pcien
Instantiated technology primitives:	eclkgen
Signals not driven in this technology:	sdclk, pciclk, clk1xu, clk2xu, clkb, clkc

Please contact Aeroflex Gaisler for information concerning the use of this clock generator.

21.2.16 eASIC45 (Nextreme2)

Generics used in this technology:	clk_mul, clk_div, freq, pcisysclk, pcien, sdramen, clk2xen
Instantiated technology primitives:	eclkgen
Signals not driven in this technology:	clk1xu, clk2xu, clkb, clkc

An example instantiating eASIC's clock generator wrapper that generates clk, clkn and clk2x is provided. Note that the example does not instantiate buffers on the clock outputs. Please contact Aeroflex Gaisler for information concerning the use of this clock generator.

21.2.17 Actel Fusion

Generics used in this technology:	clk_mul, clk_div, clk_odiv, pcisysclk, pcien, freq, clkb_odiv, clkc_odiv
Instantiated technology primitives:	PLLINT, PLL
Signals not driven in this technology:	clkn, sdclk, clk2x, clk4x, clk1xu, clk2xu

This technology instantiates a PLL and a PLLINT to generate the main clock. The instantiation of a PLLINT macro allows the PLL reference clock to be driven from an I/O that is routed through the regular FPGA routing fabric. Figure 67 shows the instantiated primitives, the PLL EXTFB input is not shown and the EXTFB port on the instantiated component is always tied to ground. The OADIVRST port on the PLL is driven by CGI.PLLRST. The figure shows which of the core's output ports that are driven by the PLL. The PCICLOCK will directly connected to PCICLKIN if VHDL generic *pcien* is non-zero, while CGO.PCILOCK is always driven high. The VHDL generics *pcien* and *pcisysclk* are used to select the reference clock. The values driven on the PLL inputs are listed in tables 182 and 183.

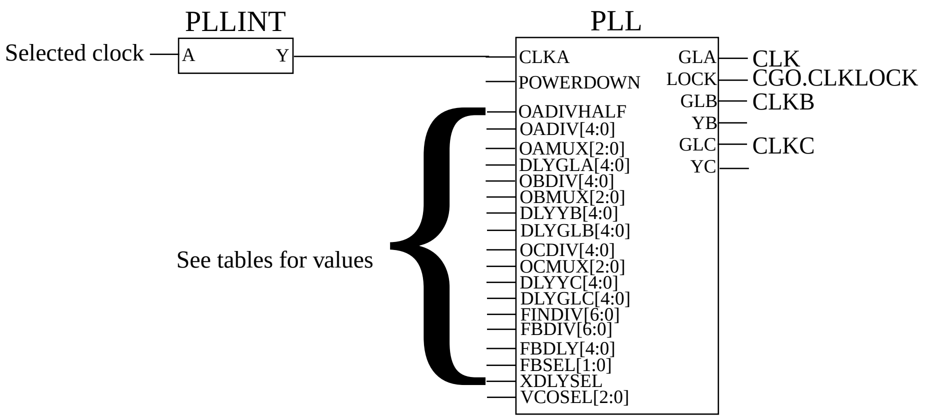


Figure 67. Actel Fusion clock generation

*Table 182.*Constant input signals on Actel Fusion PLL

Signal name	Value	Comment
OADIVHALF	0	Division by half
OADIV[4:0]	VHDL generic *clk_odiv* - 1	Output divider
OAMUX[2:0]	0b100	Post-PLL MUXA
DLYGLA[4:0]	0	Delay on Global A
OBDIV[4:0]	VHDL generic *clkb_odiv* - 1 when *clkb_odiv* > 0, otherwise 0	Output divider
OBMUX[2:0]	0 when VHDL generic *clkb_odiv* = 0, otherwise 0b100	Post-PLL MUXB

*Table 182.*Constant input signals on Actel Fusion PLL

Signal name	Value	Comment
DLYYB[4:0]	0	Delay on YB
DLYGLB[4:0]	0	Delay on Global B
OCDIV[4:0]	VHDL generic *clkc_odiv* - 1 when *clkc_odiv* > 0, otherwise 0	Output divider
OCMUX[2:0]	0 when VHDL generic *clkc_odiv* = 0, otherwise 0b100	Post-PLL MUXC
DLYYC[4:0]	0	Delay on YC
DLYGLC[4:0]	0	Delay on Global C
FINDIV[6:0]	VHDL generic *clk_div* - 1	Input divider
FBDIV[6:0]	VHDL generic *clk_mul* - 1	Feedback divider
FBDLY[4:0]	0	Feedback delay
FBSEL[1:0]	0b01	2-bit PLL feedback MUX
XDLYSEL	0	1-bit PLL feedback MUX
VCOSEL[2:0]	*See table 172 below*	VCO gear control. Selects one of four frequency ranges.

The PLL primitive has one parameter, VCOFREQUENCY, which is calculated with:

$$VCOFREQUENCY = \frac{freq \cdot clkmul}{clkdiv} / 1000$$

The calculations are performed with integer precision. This value is also used to determine the value driven on PLL input VCOSEL[2:0]. Table 172 lists the signal value depending on the value of VCOF-REQUENCY.

*Table 183.*VCOSEL[2:0] on Actel Fusion PLL

Value of VCOFREQUENCY	Value driven on VCOSEL[2:0]
< 44	0b000
< 88	0b010
< 175	0b100
>= 175	0b110

21.2.18 Altera Stratix 4

This technology is not fully supported at this time.

21.3 Configuration options

Table 184 shows the configuration options of the core (VHDL generics).

*Table 184.*Configuration options

Generic name	Function	Allowed range	Default
tech	Target technology	0 - NTECH	inferred
clk_mul	Clock multiplier, used in clock scaling. Not all techbologies support clock scaling.		1
clk_div	Clock divisor, used in clock scaling. Not all technologies support clock scaling.		1
sdramen	When this generic is set to 1 the core will generate a clock on the SDCLK. Not supported by all technologies. See technology specific description.		0
noclkfb	When this generic is set to 0 the core will use the CGI.PLLREF input as feedback clock for some technologies. See technology specific description.		1
pcien	When this generic is set to 1 the PCI clock is activated. Otherwise the PCICLKIN input is typically unused. See technology specific descriptions.		0
pcidll	When this generic is set to 1, a DLL will be instantiated for the PCI input clock for some technologies. See the technology specific descriptions.		0
pcisysclk	When this generic is set to 1 the clock generator will use the pciclkin input as the main clock reference. This also requires generic pcien to be set to 1.		0
freq	Clock frequency in kHz		25000
clk2xen	Enables 2x clock output. Not available in all technolgies and may have additional options. See technology specific description.		0
clksel	Enable clock select. Not available in all technologies.		0
clk_odiv	ProASIC3/Fusion output divider for GLA. Only used in ProASIC3/Fusion technology.	1 - 32	1
clkb_odiv	ProASIC3/Fusion output divider for GLB. Only used in ProASIC3/Fusion technology. Set this value to 0 to disable generation of GLB.	0 - 32	0
clkc_odiv	ProASIC3/Fusion output divider for GLC. Only used in ProASIC3/Fusion technology. Set this value to 0 to disable generation of GLC.	0 - 32	0

21.4 Signal descriptions

Table 185 shows the interface signals of the core (VHDL ports).

*Table 185.*Signal descriptions

Signal name	Field	Type	Function	Active
CLKIN	N/A	Input	Reference clock input	-
PCICLKIN	N/A	Input	PCI clock input	
CLK	N/A	Output	Main clock	-
CLKN	N/A	Output	Inverted main clock	-
CLK2X	N/A	Output	2x clock	-
SDCLK	N/A	Output	SDRAM clock	-
PCICLK	N/A	Output	PCI clock	-
CGI	PLLREF	Input	Optional reference for PLL	-
	PLLRST	Input	Optional reset for PLL	
	PLLCTRL	Input	Optional control for PLL	
	CLKSEL	Input	Optional clock select	
CGO	CLKLOCK	Output	Lock signal for main clock	
	PCILOCK	Output	Lock signal for PCI clock	
CLK4X	N/A	Output	4x clock	
CLK1XU	N/A	Output	Unscaled 1x clock	
CLK2XU	N/A	Output	Unscaled 2x clock	
CLKB	N/A	Output	GLB output from ProASIC3/Fusion PLL	-
CLKC	N/A	Output	GLC output from ProASIC3/Fusion PLL	

21.5 Library dependencies

Table 186 shows the libraries used when instantiating the core (VHDL libraries).

*Table 186.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Component, signals	Core signal definitions
TECHMAP	ALLCLKGEN	Component	Technology specific CLKGEN components

21.6 Instantiation

This example shows how the core can be instantiated together with the GRLIB reset generator.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;
library gaisler;
use gaisler.misc.all;

entity clkgen_ex is
  port (
    resetn  : in  std_ulogic;
    clk  : in  std_ulogic; -- 50 MHz main clock
    pllref   : in  std_ulogic
  );
end;

architecture example of clkgen_ex is
```

```
signal lclk, clkm, rstn, rstraw, sdclkl, clk50: std_ulogic;
signal cgi   : clkgen_in_type;
signal cgo   : clkgen_out_type;

begin
  cgi.pllctrl <= "00"; cgi.pllrst <= rstraw;

  pllref_pad : clkpad generic map (tech => padtech) port map (pllref, cgi.pllref);

  clk_pad : clkpad generic map (tech => padtech) port map (clk, lclk);

  clkgen0 : clkgen  -- clock generator
    generic map (clktech, CFG_CLKMUL, CFG_CLKDIV, CFG_MCTRL_SDEN,
                 CFG_CLK_NOFB, 0, 0, 0, BOARD_FREQ)
    port map (lclk, lclk, clkm, open, open, sdclkl, open, cgi, cgo, open, clk50);

  sdclk_pad : outpad generic map (tech => padtech, slew => 1, strength => 24)
    port map (sdclk, sdclkl);

  resetn_pad : inpad generic map (tech => padtech) port map (resetn, rst);

  rst0 : rstgen  -- reset generator
    port map (rst, clkm, cgo.clklock, rstn, rstraw);

end;
```

22 DDRSPA - 16-, 32- and 64-bit DDR266 Controller

22.1 Overview

DDRSPA is a DDR266 SDRAM controller with AMBA AHB back-end. The controller can interface two 16-, 32- or 64-bit DDR266 memory banks to a 32-bit AHB bus. The controller acts as a slave on the AHB bus where it occupies a configurable amount of address space for DDR SDRAM access. The DDR controller is programmed by writing to a configuration register mapped located in AHB I/O address space. Internally, DDRSPA consists of a ABH/DDR controller and a technology specific DDR PHY. Currently supported technologies for the PHY includes Xilinx Virtex2/Virtex4 and Altera Stratix-II. The modular design of DDRSPA allows to add support for other target technologies in a simple manner.

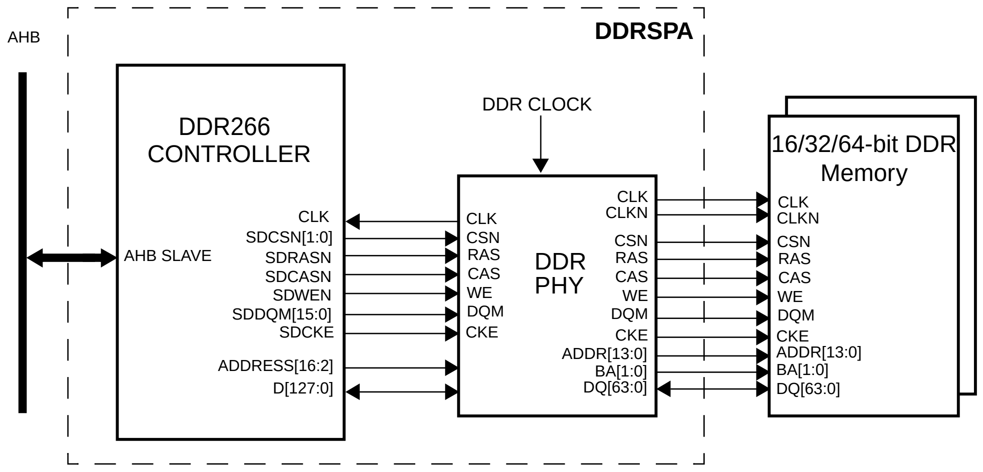


Figure 68. DDRSPA Memory controller conected to AMBA bus and DDR SDRAM

22.2 Operation

22.2.1 General

Double data-rate SDRAM (DDR RAM) access is supported to two banks of 16-, 32- or 64-bit DDR266 compatible memory devices. The controller supports 64M, 128M, 256M, 512M and 1G devices with 9- 12 column-address bits, up to 14 row-address bits, and 4 internal banks. The size of each of each chip select can be programmed in binary steps between 8 Mbyte and 1024 Mbyte. The DDR data width is set by the *ddrbits* VHDL generic, and will affect the width of DM, DQS and DQ signals. The DDR data width does not change the behavior of the AHB interface, except for data latency. When the VHDL generic *mobile* is set to a value not equal to 0, the controller supports mobile DDR SDRAM (LPDDR).

22.2.2 Read cycles

An AHB read access to the controller will cause a corresponding access to the external DDR RAM. The read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command. CAS latency of 2 (CL=2) or 3 (CL=3) can be used. Byte, half-word (16-bit) and word (32-bit) AHB accesses are supported. Incremental AHB burst access are supported for 32-bit words only. The read cycle(s) are always terminated with a PRE-CHARGE command, no banks are left open between two accesses. DDR read cycles are always performed in (aligned) 8-word bursts, which are stored in a FIFO. After an initial latency, the data is then read out on the AHB bus with zero waitstates.

22.2.3 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. An AHB write burst will store up to 8 words in a FIFO, before writing the data to the DDR memory. As in the read case, only word bursts are supported

22.2.4 Initialization

If the *pwron* VHDL generic is 1, then the DDR controller will automatically perform the DDR initialization sequence as described in the JEDEC DDR266 standard: PRE-CHARGE, LOAD-EXTMODE-REG, LOAD-MODE-REG, PRE-CHARGE, 2xREFRESH and LOAD-MODE-REG; or as described in the JEDEC LPDDR standard when mobile DDR is enabled: PRE-CHARGE, 2xREFRESH, LOAD-MODE-REG and LOAD-EXTMODE-REG. The VHDL generics *col* and *Mbyte* can be used to also set the correct address decoding after reset. In this case, no further software initialization is needed. The DDR initialization can be performed at a later stage by setting bit 15 in the DDR control register.

22.2.5 Configurable DDR SDRAM timing parameters

To provide optimum access cycles for different DDR devices (and at different frequencies), three timing parameters can be programmed through the memory configuration register (SDCFG): TRCD, TRP and TRFCD. The value of these field affects the SDRAM timing as described in table 187.

*Table 187.*DDR SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
Precharge to activate (t_{RP})	TRP + 2
Auto-refresh command period (t_{RFC})	TRFC + 3
Activate to read/write (t_{RCD})	TRCD + 2
Activate to Activate (t_{RC})	TRCD + 8
Activate to Precharge (t_{RAS})	TRCD + 6

If the TCD, TRP and TRFC are programmed such that the DDR200/266 specifications are fulfilled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 100 and 133 MHz operation and the resulting SDRAM timing (in ns):

*Table 188.*DDR SDRAM example programming

DDR SDRAM settings	t_{RCD}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
100 MHz: CL=2, TRP=0, TRFC=4, TRCD=0	20	80	20	70	60
133 MHz: CL=2, TRP=1, TRFC=6, TRCD=1	22.5	75	22.5	67.5	52.5

When the DDRSPA controller uses CAS latency (CL) of two cycles a DDR SDRAM speed grade of -75Z or better is needed to meet 133 MHz timing.

When mobile DDR support is enabled, two additional timing parameters can be programmed though the Power-Saving configuration register.

*Table 189.*Mobile DDR SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
Exit Power-down mode to first valid command (t_{XP})	TXP + 1
Exit Self Refresh mode to first valid command (t_{XSR})	TXSR + 1
CKE minimum pulse width (t_{CKE})	TCKE + 1

22.2.6 Extended timing fields

The DDRSPA controller can be configured with extended timing fields to provide support for DDR333 and DDR400. These fields can be detected by checking the XTF bit in the SDCFG register.

When the extended timing fields are enabled, extra upper bits are added to increase the range of the TRP, TRFC, TXSR and TXP fields. A new TWR field allow increasing the write recovery time. A new TRAS field to directly control the Active to Precharge period has been added.

*Table 190.*DDR SDRAM extended timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
Activate to Activate (t_{RC})	TRAS+TRCD + 2
Activate to Precharge (t_{RAS})	TRAS + 6
Write recovery time (t_{WR})	TWR+2

*Table 191.*DDR SDRAM extended timing example programming

DDR SDRAM settings	t_{RCD}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}	t_{WR}
166 MHz: CL=2, TRP=1, TRFC=9, TRCD=1, TRAS=1, TWR=1	18	60	18	72	42	18
200 MHz: CL=3, TRP=1, TRFC=11, TRCD=1, TRAS=2, TWR=1	15	55	15	70	40	15

22.2.7 Refresh

The DDRSPA controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the SDCFG register. Depending on SDRAM type, the required period is typically 7.8 us (corresponding to 780 at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by bit 31 in SDCTRL register.

22.2.8 Self Refresh

The self refresh mode can be used to retain data in the SDRAM even when the rest of the system is powered down. When in the self refresh mode, the SDRAM retains data without external clocking and refresh are handled internally. The memory array that is refreshed during the self refresh operation is defined in the extended mode register. These settings can be changed by setting the PASR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the PASR bits are changed. The supported “Partial Array Self Refresh” modes are: Full, Half, Quarter, Eighth, and Sixteenth array. “Partial Array Self Refresh” is only supported when mobile DDR functionality is enabled. To enable the self refresh mode, set the PMODE bits in the Power-Saving configuration register to “010” (Self Refresh). The controller will enter self refresh mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. When exiting this mode and mobile DDR is disabled, the controller introduce a delay of 200 clock cycles and a AUTO REFRESH command before any other memory access is allowed. When mobile DDR is enabled the delay before the AUTO REFRESH command is defined by tXSR in the Power-Saving configuration register. The minimum duration of this mode is defined by tRFC. This mode is only available when the VHDL generic *mobile* is >= 1.

22.2.9 Clock Stop

In the clock stop mode, the external clock to the SDRAM is stop at a low level (DDR_CLK is low and DDR_CLKB is high). This reduce the power consumption of the SDRAM while retaining the data. To enable the clock stop mode, set the PMODE bits in the Power-Saving configuration register to “100” (Clock Stop). The controller will enter clock stop mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. The REFRESH command will still be issued by the controller in this mode. This mode is only available when the VHDL generic *mobile* is >= 1 and mobile DDR functionality is enabled.

22.2.10 Power-Down

When entering the power-down mode all input and output buffers, including DDR_CLK and DDR_CLKB and excluding DDR_CKE, are deactivated. This is a more efficient power saving mode then clock stop mode, with a grater reduction of the SDRAM's power consumption. All data in the SDRAM is retained during this operation. To enable the power-down mode, set the PMODE bits in the Power-Saving configuration register to "001" (Power-Down). The controller will enter power-down mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits is cleared. The REFRESH command will still be issued by the controller in this mode. When exiting this mode a delay of one or two (when tXP in the Power-Saving configuration register is '1') clock cycles are added before issue any command to the memory. This mode is only available when the VHDL generic *mobile* is >= 1.

22.2.11 Deep Power-Down

The deep power-down operating mode is used to achieve maximum power reduction by eliminating the power of the memory array. Data will not be retained after the device enters deep power-down mode. To enable the deep power-down mode, set the PMODE bits in the Power-Saving configuration register to "101" (Deep Power-Down). To exit the deep power-down mode the PMODE bits in the Power-Saving configuration register must be cleared followed by the mobile SDRAM initialization sequence. The mobile SDRAM initialization sequence can be performed by setting bit 15 in the DDR control register. This mode is only available when the VHDL generic *mobile* is >= 1 and mobile DDR functionality is enabled.

22.2.12 Status Read Register

The status read register (SRR) is used to read the manufacturer ID, revision ID, refresh multiplier, width type, and density of the SDRAM. To Read the SSR a LOAD MODE REGISTER command with BA0 = 1 and BA1 = 0 must be issued followed by a READ command with the address set to 0. This command sequence is executed then the Status Read Register is read. This register is only available when the VHDL generic *mobile* is >= 1 and mobile DDR functionality is enabled. Only DDR_CSB[0] is enabled during this operation.

22.2.13 Temperature-Compensated Self Refresh

The settings for the temperature-compensation of the Self Refresh rate can be controlled by setting the TCSR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the TCSR bits are changed. Note that some vendors implements a Internal Temperature-Compensated Self Refresh feature, which makes the memory to ignore the TCSR bits. This functionality is only available when the VHDL generic *mobile* >= 1 and mobile DDR functionality is enabled.

22.2.14 Drive Strength

The drive strength of the output buffers can be controlled by setting the DS bits in the Power-Saving configuration register. The extended mode register is automatically updated when the DS bits are changed. The available options are: full, three-quarter, one-half, and one-quarter drive strengths. This functionality is only available when the VHDL generic *mobile* is >= 1 and mobile DDR functionality is enabled.

22.2.15 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in SDCFG: PRE-CHARGE, LOAD-EXTMODE-REG, LOAD-MODE-REG and REFRESH. If the LEMR command is issued, the PLL Reset bit as programmed in SDCFG will be used, when mobile DDR support is enabled the DS, TCSR and PASR as programmed in Power-Saving configuration register will be used. If the LMR command is issued, the CAS latency as programmed in the Power-Saving configuration register will be used and remaining fields are fixed: 8 word sequential burst. The command field will be cleared after a command has been executed.

22.2.16 Clocking

The DDR controller is designed to operate with two clock domains, one for the DDR memory clock and one for the AHB clock. The two clock domains do not have to be the same or be phase-aligned. The DDR input clock (CLK_DDR) can be multiplied and divided by the DDR PHY to form the final DDR clock frequency. The final DDR clock is driven on one output (CLKDDRO), which should always be connected to the CLKDDRI input. If the AHB clock and DDR clock area generated from the same clock source, a timing-ignore constraint should be placed between the CLK_AHB and CLKDDRI to avoid optimization of false-paths during synthesis and place&route.

The Xilinx version of the PHY generates the internal DDR read clock using an external clock feed-back. The feed-back should have the same delay as DDR signals to and from the DDR memories. The feed-back should be driven by DDR_CLK_FB_OUT, and returned on DDR_CLK_FB. Most Xilinx FPGA boards with DDR provides clock feed-backs of this sort. The supported frequencies for the Xilinx PHY depends on the clock-to-output delay of the DDR output registers, and the internal delay from the DDR input registers to the read data FIFO. Virtex2 and Virtex4 can typically run at 120 MHz, while Spartan3e can run at 100 MHz.

The read data clock in the Xilinx version of the PHY is generated using a DCM to offset internal delay of the DDR clock feed back. If the automatic DCM phase adjustment does not work due to unsuitable pin selection, extra delay can be added through the RSKEW VHDL generic. The VHDL generic can be between -255 and 255, and is passed directly to the PHASE_SHIFT generic of the DCM.

The Altera version of the PHY use the DQS signals and an internal PLL to generate the DDR read clock. No external clock feed-back is needed and the DDR_CLK_FB_OUT/DDR_CLK_FB signals are not used. The supported frequencies for the Altera PHY are 100, 110, 120 and 130 MHz. For Altera CycloneIII, the read data clock is generated by the PLL. The phase shift of the read data clock is set be the VHDL generic RSKEW in ps (e.g. a value of 2500 equals 90’ phase for a 100MHz system).

22.2.17 Pads

The DDRSPA core has technology-specific pads inside the core. The external DDR signals should therefore be connected directly the top-level ports, without any logic in between.

22.2.18 Registers

The DDRSPA core implements two control registers. The registers are mapped into AHB I/O address space defined by the AHB BAR1 of the core.

*Table 192.*DDR controller registers

Address offset - AHB I/O - BAR1	**Register**
0x00	SDRAM control register
0x04	SDRAM configuration register (read-only)
0x08	SDRAM Power-Saving configuration register
0x0C	Reserved
0x10	Status Read Register (Only available when mobile DDR support is enabled)
0x14	PHY configuration register 0 (Only available when VHDL generic confapi = 1, TCI RTL_PHY)
0x18	PHY configuration register 1 (Only available when VHDL generic confapi = 1, TCI TRL_PHY)

Table 193. SDRAM control register (SDCTRL)

31	30	29 27	26	25 23	22 21	20 18	17	16	15	14 0
Refresh	tRP	tRFC	tRCD	SDRAM bank size	SDRAM col. size	SDRAM command	PR	IN	CE	SDRAM refresh load value

31	SDRAM refresh. If set, the SDRAM refresh will be enabled. This register bit is read only when Power-Saving mode is other then none.
30	SDRAM tRP timing. tRP will be equal to 2 or 3 system clocks (0/1). When mobile DDR support is enabled, this bit also represent the MSB in the tRFC timing.
29: 27	SDRAM tRFC timing. tRFC will be equal to 3 + field-value system clocks. When mobile DDR support is enabled, this field is extended with the bit 30.
26	SDRAM tRCD delay. Sets tRCD to 2 + field value clocks.
25: 23	SDRAM banks size. Defines the decoded memory size for each SDRAM chip select: “000”= 8 Mbyte, “001”= 16 Mbyte, “010”= 32 Mbyte .... “111”= 1024 Mbyte.
22: 21	SDRAM column size. “00”=512, “01”=1024, “10”=2048, “11”=4096
20: 18	SDRAM command. Writing a non-zero value will generate an SDRAM command: “010”=PRECHARGE, “100”=AUTO-REFRESH, “110”=LOAD-COMMAND-REGISTER, “111”=LOAD-EXTENDED-COMMAND-REGISTER. The field is reset after command has been executed.
17	PLL Reset. This bit is used to set the PLL RESET bit during LOAD-CONFIG-REG commands.
16	Initialize (IN). Set to ‘1’ to perform power-on DDR RAM initialisation. Is automatically cleared when initialisation is completed. This register bit is read only when Power-Saving mode is other then none.
15	Clock enable (CE). This value is driven on the CKE inputs of the DDR RAM. Should be set to ‘1’ for correct operation. This register bit is read only when Power-Saving mode is other then none.
14: 0	The period between each AUTO-REFRESH command - Calculated as follows: tREFRESH = ((reload value) + 1) / DDRCLOCK

Table 194. SDRAM configuration register (SDCFG)

31 21	20	19 16	15	14 12	11 0
Reserved	XTF	CONFAPI	MD	Data width	DDR Clock frequency

31: 21	Reserved
20	Extended timing fields for DDR400 available
19: 16	Register API configuration. 0 = Standard register API. 1 = TCI TSMC90 PHY register API.

Table 194. SDRAM configuration register (SDCFG)

15	Mobile DDR support enabled. ‘1’ = Enabled, ‘0’ = Disabled (read-only)
14: 12	DDR data width: “001” = 16 bits, “010” = 32 bits, “011” = 64 bits (read-only)
11: 0	Frequency of the (external) DDR clock (read-only)

Table 195. SDRAM Power-Saving configuration register

31	30	29 28	27 26	25	24	23 20	19	18 16	15 12	11	10	9 8	7 5	4 3	2 0
ME	CL	TRAS	xXS*	xXP	tC	tXSR	tXP	PMODE	Reserved	TWR	xTRP	xTRFC	DS	TCSR	PASR

31	Mobile DDR functionality enabled. ‘1’ = Enabled (support for Mobile DDR SDRAM), ‘0’ = disabled (support for standard DDR SDRAM)
30	CAS latency; ‘0’ => CL = 2, ‘1’ => CL = 3
29: 28	SDRAM extended tRAS timing, tRAS will be equal to field-value + 6 system clocks. (Reserved when extended timing fields are disabled)
27: 26	SDRAM extended tXSR field, extend tXSR with field-value * 16 clocks (Reserved when extended timing fields are disabled)
25	SDRAM extended tXP field, extend tXP with 2*field-value clocks (Reserved when extended timing fields are disabled)
24	SDRAM tCKE timing, tCKE will be equal to 1 or 2 clocks (0/1). (Read only when Mobile DDR support is disabled).
23: 20	SDRAM tXSR timing. tXSR will be equal to field-value system clocks. (Read only when Mobile DDR support is disabled).
19	SDRAM tXP timing. tXP will be equal to 2 or 3 system clocks (0/1). (Read only when Mobile DDR support is disabled).
18: 16	Power-Saving mode (Read only when Mobile DDR support is disabled). “000”: none “001”: Power-Down (PD) “010”: Self-Refresh (SR) “100”: Clock-Stop (CKS) “101”: Deep Power-Down (DPD)
15: 12	Reserved
11	SDRAM extended tWR timing, tWR will be equal to field-value + 2 clocks (Reserved when extended timing fields are disabled)
10	SDRAM extended tRP timing, extend tRP with field-value * 2 clocks
9: 8	SDRAM extended tRFC timing, extend tRFC with field-value * 8 clocks
7: 5	Selectable output drive strength (Read only when Mobile DDR support is disabled). “000”: Full “001”: One-half “010”: One-quarter “011”: Three-quarter
4: 3	Reserved for Temperature-Compensated Self Refresh (Read only when Mobile DDR support is disabled). “00”: 70ªC “01”: 45ªC “10”: 15ªC “11”: 85ªC
2: 0	Partial Array Self Refresh (Read only when Mobile DDR support is disabled). “000”: Full array (Banks 0, 1, 2 and 3) “001”: Half array (Banks 0 and 1) “010”: Quarter array (Bank 0) “101”: One-eighth array (Bank 0 with row MSB = 0) “110”: One-sixteenth array (Bank 0 with row MSB = 00)

Table 196. Status Read Register

31 16	15 0
SRR_16	SRR

Table 196. Status Read Register

31: 16	Status Read Register when 16-bit DDR memory is used (read only)
15: 0	Status Read Register when 32/64-bit DDR memory is used (read only)

Table 197. PHY configuration register 0 (TCI RTL_PHY only)

31	30	29	28	27 22	21 16	15 8	7 0
R1	R0	P1	P0	TSTCTRL1	TSTCTRL0	MDAJ_DLL1	MDAJ_DLL0

31	Reset DLL 1 (active high)
30	Reset DLL 1 (active high)
29	Power Down DLL 1 (active high)
28	Power Down DLL 1 (active high)
27: 22	Test control DLL 1 tstclkin(1) is connected to SIGI_1 on DDL 1 when bit 26:25 is NOT equal to “00“. tstclkin(0) is connected to SIGI_0 on DDL 1 when bit 23:22 is NOT equal to “00“.
21: 16	Test control DLL 0
15: 8	Master delay adjustment input DLL 1
7: 0	Master delay adjustment input DLL 0

Table 198. PHY configuration register 1 (TCI RTL_PHY only)

31 24	23 16	15 8	7 0
ADJ_RSYNC	ADJ_90	ADJ_DQS1	ADJ_DQS0

31: 24	Slave delay adjustment input for resync clock (Slave 1 DLL 1)
23: 16	Slave delay adjustment input for 90’ clock (Slave 0 DLL 1)
15: 8	Slave delay adjustment input for DQS 1 (Slave 1 DLL 0)
7: 0	Slave delay adjustment input for DQS 0 (Slave 0 DLL 0)

22.3 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x025. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

22.4 Configuration options

Table 199 shows the configuration options of the core (VHDL generics).

*Table 199.*Configuration options

Generic	Function	Allowed range	Default
fabtech	PHY technology selection	virtex2, virtex4, spartan3e, altera	virtex2
memtech	Technology selection for DDR FIFOs	infered, virtex2, virtex4, spartan3e, altera	infered
hindex	AHB slave index	0 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where DDR control register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space	0 - 16#FFF#	16#FFF#
ddrbits	Data bus width of external DDR memory	16, 32, 64	16
MHz	DDR clock input frequency in MHz.	10 - 200	100
clkmul, clkdiv	The DDR input clock is multiplied with the clkmul generic and divided with clkdiv to create the final DDR clock	2 - 32	2
rstdel	Clock reset delay in micro-seconds.	1 - 1023	200
col	Default number of column address bits	9 - 12	9
Mbyte	Default memory chip select bank size in Mbyte	8 - 1024	16
pwron	Enable SDRAM at power-on initialization	0 - 1	0
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
ahbfreq	Frequency in MHz of the AHB clock domain	1 - 1023	50
rskew	Additional read data clock skew Read data clock phase for Altera CycloneIII	-255 - 255. 0 - 9999	0
mobile	Enable Mobile DDR support 0: Mobile DDR support disabled 1: Mobile DDR support enabled but not default 2: Mobile DDR support enabled by default 3: Mobile DDR support only (no regular DDR support)	0 - 3	0
confapi	Set the PHY configuration register API: 0 = standard register API (conf0 and conf1 disabled). 1 = TCI RTL_PHY register API.		
conf0	Reset value for PHY register 0, conf[31:0]	0 - 16#FFFFFFFF#	0
conf1	Reset value for PHY register1, conf[63:32]	0 - 16#FFFFFFFF#	0
regoutput	Enables registers on signal going from controller to PHY	0 - 1	0
ddr400	Enables extended timing fields for DDR400 support	0 - 1	1
scantest	Enable scan test support	0 - 1	0
phyiconf	PHY implementation configuration. This generic sets technology specific implementation options for the DDR PHY. Meaning of values depend on the setting of VHDL generic *fabtech*. For fabtech:s virtex4, virtex5, virtex6: phyiconf selects type of pads used for DDR clock pairs. 0 instantiates a differiental pad and 1 instantiates two outpads.	0 - 16#FFFFFFFF#	0

22.5 Implementation

22.5.1 Technology mapping

The core has two technology mapping VHDL generics: *memtech* and *padtech*. The VHDL generic *memtech* controls the technology used for memory cell implementation. The VHDL generic *padtech* controls the technology used in the PHY implementation. See the GRLIB Users's Manual for available settings.

22.5.2 RAM usage

The FIFOs in the core are implemented with the *syncram_2p* (with separate clock for each port) component found in the technology mapping library (TECHMAP). The number of RAMs used for the FIFO implementation depends on the DDR data width, set by the *ddrbits* VHDL generic.

*Table 200.*RAM usage

RAM dimension (depth x width)	**Number of RAMs (DDR data width 64)**	**Number of RAMs (DDR data width 32)**	**Number of RAMs (DDR data width 16)**
4 x 128	1		
4 x 32	4		
5 x 64		1	
5 x 32		2	
6 x 32			2

22.6 Signal descriptions

Table 201 shows the interface signals of the core (VHDL ports).

*Table 201.*Signal descriptions

Signal name	**Type**	**Function**	**Active**
RST_DDR	Input	Reset input for DDR clock domain	Low
RST_AHB	Input	Reset input for AHB clock domain	Low
CLK_DDR	Input	DDR input Clock	-
CLK_AHB	Input	AHB clock	-
LOCK	Output	DDR clock generator locked	High
CLKDDRO		Internal DDR clock output after clock multiplication	
CLKDDRI		Clock input for the internal DDR clock domain. Must be connected to CLKDDRO.	
AHBSI	Input	AHB slave input signals	-
AHBSO	Output	AHB slave output signals	-
DDR_CLK[2:0]	Output	DDR memory clocks (positive)	High
DDR_CLKB[2:0]	Output	DDR memory clocks (negative)	Low
DDR_CLK_FB_OUT	Output	Same a DDR_CLK, but used to drive an external clock feedback.	-
DDR_CLK_FB	Input	Clock input for the DDR clock feed-back	-
DDR_CKE[1:0]	Output	DDR memory clock enable	High
DDR_CSB[1:0]	Output	DDR memory chip select	Low
DDR_WEB	Output	DDR memory write enable	Low
DDR_RASB	Output	DDR memory row address strobe	Low
DDR_CASB	Output	DDR memory column address strobe	Low
DDR_DM[DDRBITS/8-1:0]	Output	DDR memory data mask	Low
DDR_DQS[DDRBITS/8-1:0]	Bidir	DDR memory data strobe	Low
DDR_AD[13:0]	Output	DDR memory address bus	Low
DDR_BA[1:0]	Output	DDR memory bank address	Low
DDR_DQ[DDRBITS-1:0]	BiDir	DDR memory data bus	-

1) see GRLIB IP Library User's Manual 2) Polarity selected with the oepol generic

22.7 Library dependencies

Table 202 shows libraries used when instantiating the core (VHDL libraries).

*Table 202.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

22.8 Component declaration

```
component ddrspa
  generic (
    fabtech : integer := 0;
    memtech : integer := 0;
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#f00#;
    ioaddr  : integer := 16#000#;
    iomask  : integer := 16#fff#;
    MHz     : integer := 100;
    clkmul  : integer := 2;
    clkdiv  : integer := 2;
    col     : integer := 9;
    Mbyte   : integer := 16;
    rstdel  : integer := 200;
    pwron   : integer := 0;
    oepol   : integer := 0;
    ddrbits : integer := 16;
    ahbfreq : integer := 50
  );
  port (
    rst_ddr : in  std_ulogic;
    rst_ahb : in  std_ulogic;
    clk_ddr : in  std_ulogic;
    clk_ahb : in  std_ulogic;
    lock    : out std_ulogic;-- DCM locked
    clkddro : out std_ulogic;-- DCM locked
    clkddri : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;
    ddr_clk : out std_logic_vector(2 downto 0);
    ddr_clkb: out std_logic_vector(2 downto 0);
    ddr_clk_fb_out  : out std_logic;
    ddr_clk_fb  : in std_logic;
    ddr_cke  : out std_logic_vector(1 downto 0);
    ddr_csb  : out std_logic_vector(1 downto 0);
    ddr_web  : out std_ulogic;                       -- ddr write enable
    ddr_rasb  : out std_ulogic;                       -- ddr ras
    ddr_casb  : out std_ulogic;                       -- ddr cas
    ddr_dm   : out std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dm
    ddr_dqs  : inout std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dqs
    ddr_ad      : out std_logic_vector (13 downto 0);   -- ddr address
    ddr_ba      : out std_logic_vector (1 downto 0);    -- ddr bank address
    ddr_dq    : inout  std_logic_vector (ddrbits-1 downto 0) -- ddr data

  );
  end component;
```

22.9 Instantiation

This examples shows how the core can be instantiated.

The DDR SDRAM controller decodes SDRAM area at 0x40000000 - 0x7FFFFFFF. The SDRAM registers are mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;

entity ddr_Interface is
  port ( ddr_clk  : out std_logic_vector(2 downto 0);
    ddr_clkb  : out std_logic_vector(2 downto 0);
    ddr_clk_fb  : in std_logic;
    ddr_clk_fb_out  : out std_logic;
    ddr_cke  : out std_logic_vector(1 downto 0);
    ddr_csb  : out std_logic_vector(1 downto 0);
    ddr_web  : out std_ulogic;                       -- ddr write enable
    ddr_rasb  : out std_ulogic;                       -- ddr ras
    ddr_casb  : out std_ulogic;                       -- ddr cas
    ddr_dm   : out std_logic_vector (7 downto 0);    -- ddr dm
    ddr_dqs  : inout std_logic_vector (7 downto 0);    -- ddr dqs
    ddr_ad      : out std_logic_vector (13 downto 0);   -- ddr address
    ddr_ba      : out std_logic_vector (1 downto 0);    -- ddr bank address
    ddr_dq  : inout std_logic_vector (63 downto 0); -- ddr data

);
end;

architecture rtl of mctrl_ex is

  -- AMBA bus
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal clkml, lock : std_ulogic;


begin

-- DDR controller


ddrc : ddrspa generic map ( fabtech => virtex4, ddrbits => 64, memtech => memtech,
hindex => 4, haddr => 16#400#, hmask => 16#F00#, ioaddr => 1,
pwron => 1, MHz => 100, col => 9, Mbyte => 32, ahbfreq => 50, ddrbits => 64)
port map (
rstneg, rstn, lclk, clkm, lock, clkml, clkml,  ahbsi, ahbso(4),
ddr_clk, ddr_clkb, ddr_clk_fb_out, ddr_clk_fb,
ddr_cke, ddr_csb, ddr_web, ddr_rasb, ddr_casb,
ddr_dm, ddr_dqs, ddr_adl, ddr_ba, ddr_dq);
```

23 DDR2SPA - 16-, 32- and 64-bit Single-Port Asynchronous DDR2 Controller

23.1 Overview

DDR2SPA is a DDR2 SDRAM controller with AMBA AHB back-end. The controller can interface 16-, 32- or 64-bit wide DDR2 memory with one or two chip selects. The controller acts as a slave on the AHB bus where it occupies a configurable amount of address space for DDR2 SDRAM access. The DDR2 controller is programmed by writing to configuration registers mapped located in AHB I/O address space.

Internally, DDR2SPA consists of a ABH/DDR2 controller and a technology specific DDR2 PHY. Currently supported technologies for the PHY is Xilinx Virtex4 and Virtex5 and Altera StratixIII. The modular design of DDR2SPA allows to add support for other target technologies in a simple manner. The DDR2SPA is used in the following GRLIB template designs: *leon3-xilinx-ml5xx, leon3-altera-ep3sl150*.

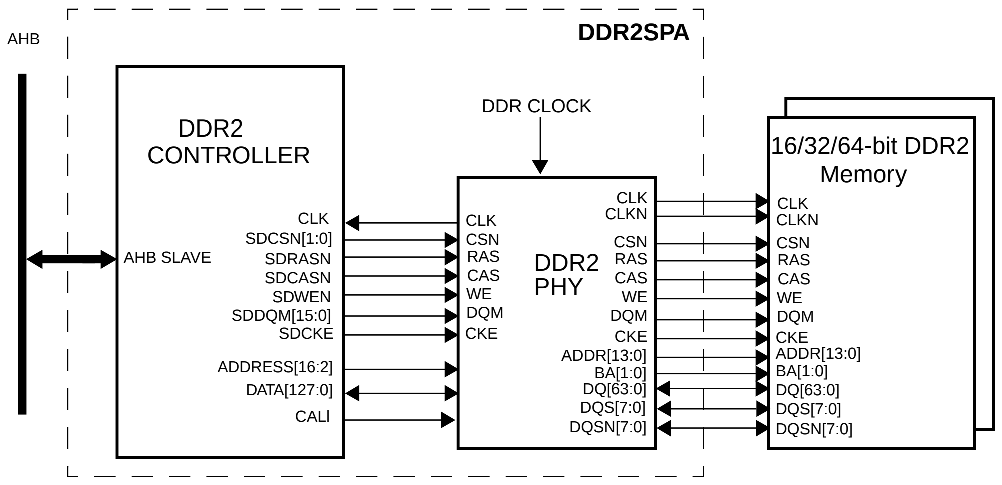


Figure 69. DDR2SPA Memory controller connected to AMBA bus and DDR2 SDRAM

23.2 Operation

23.2.1 General

Single DDR2 SDRAM chips are typically 4,8 or 16 data bits wide. By putting multiple identical chips side by side, wider SDRAM memory banks can be built. Since the command signals are common for all chips, the memories behave as one single wide memory chip.

This memory controller supports one or two (identical) such 16/32/64-bit wide DDR2 SDRAM memory banks. The size of the memory can be programmed in binary steps between 8 Mbyte and 1024 Mbyte, or between 32 Mbyte and 4096 Mbyte. The DDR data width is set by the DDRBITS generic, and will affect the width of DM, DQS and DQ signals. The DDR data width does not change the behavior of the AHB interface, except for data latency.

23.2.2 Data transfers

An AHB read or write access to the controller will cause a corresponding access cycle to the external DDR2 RAM. The cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a sequence of READ or WRITE commands (the count depending on memory width and burst length setting). After the sequence, a PRECHARGE command is performed to deactivate the SDRAM bank.

All access types are supported, but only incremental bursts of 32 bit width and incremental bursts of maximum width (if wider than 32) are handled efficiently. All other bursts are handled as single-accesses. For maximum throughput, incremental bursts of full AHB width with both alignment and length corresponding to the burstlen generic should be performed.

The maximum supported access size can be limited by using the ahbbits generic, which is set to the full AHB bus size by default. Accesses larger than this size are not supported.

The memory controller's FIFO has room for two write bursts which improves throughput, since the second write can be written into the FIFO while the first write is being written to the DDR memory.

In systems with high DDR clock frequencies, the controller may have to insert wait states for the minimum activate-to-precharge time (t_{RAS}) to expire before performing the precharge command. If a new AHB access to the same memory row is performed during this time, the controller will perform the access in the same access cycle.

23.2.3 Initialization

If the **`pwron`** VHDL generic is 1, then the DDR2 controller will automatically on start-up perform the DDR2 initialization sequence as described in the JEDEC DDR2 standard. The VHDL generics **col** and **Mbyte** can be used to also set the correct address decoding after reset. In this case, no further software initialization is needed except for enabling the auto-refresh function. If power-on initialization is not enabled, the DDR2 initialization can be started at a later stage by setting bit 16 in the DDR2 control register DDR2CFG1.

23.2.4 Big memory support

The total memory size for each chip select is set through the 3-bit wide SDRAM banks size field, which can be set in binary steps between 8 Mbyte and 1024 Mbyte. To support setting even larger memory sizes of 2048 and 4096 Mbyte, a fourth bit has been added to this configuration field.

Only 8 different sizes are supported by the controller, either the lower range of 8 MB - 1 GB, or the higher range of 32 MB - 4 GB. Which range is determined by the bigmem generic, and can be read by software through the DDR2CFG2 register.

23.2.5 Configurable DDR2 SDRAM timing parameters

To provide optimum access cycles for different DDR2 devices (and at different frequencies), six timing parameters can be programmed through the memory configuration registers: TRCD, TCL, TRTP, TWR, TRP and TRFC. For faster memories (DDR2-533 and higher), the TRAS setting also needs to be configured to satisfy timing. The value of these fields affects the DDR2RAM timing as described in table 203. Note that if the CAS latency setting is changed after initialization, this change needs also to be programmed into the memory chips by executing the Load Mode Register command.

*Table 203.*DDR2 SDRAM programmable minimum timing parameters

DDR2 SDRAM timing parameter	Minimum timing (clocks)
CAS latency, CL	TCL + 3
Activate to read/write command (t_{RCD})	TRCD + 2
Read to precharge (t_{RTP})	TRTP + 2
Write recovery time (t_{WR})	TWR-2
Precharge to activate (t_{RP})	TRP + 2
Activate to precharge (t_{RAS})	TRAS + 1
Auto-refresh command period (t_{RFC})	TRFC + 3

If TRCD, TCL, TRTP, TWR, TRP, TRFC and TRAS are programmed such that the DDR2 specifications are full filled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 130, 200 and 400 MHz operation and the resulting DDR2 SDRAM timing (in ns):

*Table 204.*DDR2 SDRAM example programming

DDR2 SDRAM settings	**CL**	t_{RCD}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
130 MHz: TCL=0,TRCD=0,TRTP=0,TRP=0,TRAS=0,TRFC=7	3	15	76	15	76	61
200 MHz: TCL=0,TRCD=1,TRTP=0,TRP=1,TRAS=1,TRFC=13	3	15	60	15	80	45
400 MHz: TCL=2,TRCD=4,TRTP=1,TRP=4,TRAS=10,TRFC=29	5	15	60	15	80	45

23.2.6 Refresh

The DDR2SPA controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the DDR2CFG1 register. Depending on SDRAM type, the required period is typically 7.8 us (corresponding to 780 at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by bit 31 in DDR2CFG1 register.

23.2.7 DDR2 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in SDCFG1: PRE-CHARGE, LOAD-EXTMODE-REG, LOAD-MODE-REG and REFRESH. If the LMR command is issued, the PLL Reset bit as programmed in DDR2CFG1, CAS Latency setting as programmed in DDR2CFG4 and the WR setting from DDR2CFG3 will be used, remaining fields are fixed: 4 word sequential burst. If the LEMR command is issued, the OCD bits will be used as programmed in the DDR2CFG1 register, and all other bits are set to zero. The command field will be cleared after a command has been executed.

23.2.8 Registered SDRAM

Registered memory modules (RDIMM:s) have one cycle extra latency on the control signals due to the external register. They can be supported with this core by setting the REG bit in the DDR2CFG4 register.

This should not be confused with Fully-Buffered DDR2 memory, which uses a different protocol and is not supported by this controller.

23.2.9 Clocking

The DDR2 controller operates in two separate clock domains, one domain synchronous to the DDR2 memory and one domain synchronous to the AHB bus. The two clock domains do not have to be the same or be phase-aligned.

The clock for the DDR2 memory domain is generated from the controller's ddr_clk input via a technology-specific PLL component. The multiplication and division factor can be selected via the clkmul/clkdiv configuration options. The final DDR2 clock is driven on one output (CLKDDRO), which should always be connected to the CLKDDRI input.

The ddr_rst input asynchronously resets the PHY layer and the built-in PLL. The ahb_rst input should be reset simultaneously and then kept in reset until the PLL has locked (indicated by the lock output).

If the AHB and DDR2 clocks are based on the same source clock and are kept phase-aligned by the PLL, the clock domain transition is synchronous to the least common multiple of the two clock frequencies. In this case, the nosync configuration option can be used to remove the synchronization and handshaking between the two clock domains, which saves a few cycles of memory access latency. If nosync is not set in this case, a timing-ignore constraint should be placed between the CLK_AHB and CLKDDRI to avoid optimization of false-paths during synthesis and place&route.

The supported DDR2 frequencies depends on the clock-to-output delay of the DDR output registers, and the internal delay from the DDR input registers to the read data FIFO. Virtex5 can typically run at 200 MHz.

When reading data, the data bus (DQ) signals should ideally be sampled 1/4 cycle after each data strobe (DQS) edge. How this is achieved is technology-specific as described in the following sections.

23.2.10 Read data clock calibration on Xilinx Virtex

On Xilinx Virtex4/5 the data signal inputs are delayed via the I/O pad IDELAY feature to get the required 1/4 cycle shift. The delay of each byte lane is tuned independently between 0-63 tap delays, each tap giving 78 ps delay, and the initial value on startup is set via the generics ddelayb[7:0].

The delays can be tuned at runtime by using the DDR2CFG3 control register. There are two bits in the control register for each byte. One bit determines if the delay should be increased or decreased and the other bit is set to perform the update. Setting bit 31 in the DDR2CFG3 register resets the delays to the initial value.

To increase the calibration range, the controller can add additional read latency cycles. The number of additional read latency cycles is set by the RD bits in the DDR2CFG3 register.

23.2.11 Read data clock calibration on Altera Stratix

On Altera StratixIII, the technology’s delay chain feature is used to delay bytes of input data in a similar fashion as the Virtex case above. The delay of each byte lane is tuned between 0-15 tap delays, each tap giving 50 ps delay, and the initial value on startup is 0.

The delays are tuned at runtime using the DDR2CFG3 register, and extra read cycles can be added using DDR2CFG3, the same way as described for Virtex.

The data sampling clock can also be skewed on Stratix to increase the calibration range. This is done writing the PLL_SKEW bits in the DDR2CFG3 register.

23.2.12 Read data clock calibration on Xilinx Spartan-3

On Spartan3, a clock loop is utilized for sampling of incoming data. The DDR_CLK_FB_OUT port should therefore be connected to a signal path of equal length as the DDR_CLK + DDR_DQS signal path. The other end of the signal path is to be connected to the DDR_CLK_FB port. The fed back clock can then be skewed for alignment with incoming data using the rskew generic. The rskew generic can be set between +/-255 resulting in a linear +/-360 degree change of the clock skew. Bits 29 and 30 in the DDR2CFG3 register can be used for altering the skew at runtime.

23.2.13 Pads

The DDR2SPA core has technology-specific pads inside the core. The external DDR2 signals should therefore be connected directly the top-level ports, without any logic in between.

23.3 Fault-tolerant operation (preliminary)

23.3.1 Overview

The memory controller can be configured to support bit-error tolerant operation by setting the ft generic (not supported in all versions of GRLIB). In this mode, the DDR data bus is widened and the extra bits are used to store 16 or 32 checkbits corresponding to each 64 bit data word. The variant to be used can be configured at run-time depending on the connected DDR2 data width and the desired level of fault tolerance.

When writing, the controller generates the check bits and stores them along with the data. When reading, the controller will transparently correct any correctable bit errors and provide the corrected data on the AHB bus. However, the corrected bits are not written back to the memory so external scrubbing is necessary to avoid uncorrectable errors accumulating over time.

An extra corrected error output signal is asserted when a correctable read error occurs, at the same cycle as the corrected data is delivered. This can be connected to an interrupt input or to a memory scrubber. In case of uncorrectable error, this is signaled by giving an AHB error response to the master.

23.3.2 Memory setup

In order to support error-correction, the DDR2 data bus needs to be expanded. The different possible physical configurations are tabulated below. For software, there is no noticeable difference between these configurations.

If the hardware is built for the wider code, it is still possible to leave the upper half of the checkbit data bus unconnected and use it for code B.

*Table 205.*Configurations of FT DDR2 memory banks

Data bits (DDRBITS)	**Checkbits (FTBITS)**	**Interleaving modes supported**
64	32	A and B
64	16	B only
32	16	A and B
32	8	B only
16	8	A only

23.3.3 Error-correction properties

The memory controller uses an interleaved error correcting code which works on nibble (4-bit) units of data. The codec can be used in two interleaving modes, mode A and mode B.

In mode A, the basic code has 16 data bits, 8 check bits and can correct one nibble error. This code is interleaved by 4 using the pattern in table 206 to create a code with 64 data bits and 32 check bits.

This code can tolerate one nibble error in each of the A,B,C,D groups shown below. This means that we can correct 100% of single errors in two adjacent nibbles, or in any 8/16-bit wide data bus lane, that would correspond to a physical DDR2 chip. The code can also correct 18/23=78% of all possible random two-nibble errors.

This interleaving pattern was designed to also provide good protection in case of reduced (32/16-bit) DDR bus width with the same data-checkbit relation, so software will see the exact same checkbits on diagnostic reads.

In mode B, the basic code has 32 data bits, 8 check bits and can correct one nibble error. This code is then interleaved by a factor of two to create a code with 64 data bits and 16 check bits.

Note that when configured for a 16-bit wide DDR data bus, code A must be used to get protection from multi-column errors since each data bus nibbles holds four code word nibbles.

Table 206. Mode Ax4 interleaving pattern (64-bit data width)

63:60	59:56	55:52	51:48	47:44	43:40	39:36	35:32	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
C	D	A	B	A	B	C	D	B	A	D	C	D	C	B	A
								127:120	119:112	111:104	103:96	95:88	87:80	79:72	71:64
								C_{cb}	D_{cb}	A_{cb}	B_{cb}	C_{cb}	D_{cb}	A_{cb}	B_{cb}

Table 207. Mode Bx2 interleaving pattern (64-bit data width)

63:60	59:56	55:52	51:48	47:44	43:40	39:36	35:32	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
A	B	A	B	A	B	A	B	B	A	B	A	B	A	B	A
												95:88	87:80	79:72	71:64
												A_{cb}	B_{cb}	A_{cb}	B_{cb}

23.3.4 Data transfers

The read case behaves the same way as the non-FT counterpart, except a few cycles extra are needed for error detection and correction. There is no extra time penalty in the case data is corrected compared to the error-free case.

Only writes of 64 bit width or higher will translate directly into write cycles to the DDR memory. Other types of write accesses will generate a read-modify-write cycle in order to correctly update the check-bits. In the special case where an uncorrectable error is detected while performing the RMW cycle, the write is aborted and the incorrect checkbits are left unchanged so they will be detected upon the next read.

Only bursts of maximum AHB width is supported, other bursts will be treated as single accesses.

The write FIFO only has room for one write (single or burst).

23.3.5 DDR2 behavior

The behavior over the DDR2 interface is largely unchanged, the same timing parameters and setup applies as for the non-FT case. The checkbit data and data-mask signals follow the same timing as the corresponding signals for regular data.

23.3.6 Configuration

Whether the memory controller is the FT or the non-FT version can be detected by looking at the FTV bit in the DDR2CFG2 register.

Checkbits are always written out to memory when writing even if EDACEN is disabled. Which type of code, A or B, that is used for both read and write is controlled by the CODE field in the DDR2FTCFG register.

Code checking on read is disabled on reset and is enabled by setting the EDACEN bit in the DDR2FTCFG register. Before enabling this, the code to be used should be set in the CODE field and the memory contents should be (re-)initialized.

23.3.7 Diagnostic checkbit access

The checkbits and data can be accessed directly for testing and fault injection. This is done by writing the address of into the DDR2FTDA register. The check-bits and data can then be read and written via the DDR2FTDC and DDR2FTDD register. Note that for checkbits the DDR2FTDA address is 64-bit aligned, while for data it is 32-bit aligned.

After the diagnostic data register has been read, the FT control register bits 31:19 can be read out to see if there were any correctable or uncorrectable errors detected, and where the correctable errors were located. For the 64 databit wide version, there is one bit per byte lane describing whether a correctable error occurred.

23.3.8 Code boundary

The code boundary feature allows you to gradually switch the memory from one interleaving mode to the other and regenerate the checkbits without stopping normal operation. This can be used when recovering from memory faults, as explained further below.

If the boundary address enable (BAEN) control bit is set, the core will look at the address of each access, and use the interleaving mode selected in the CODE field for memory accesses above or equal to the boundary address, and the opposite code for memory accesses below to the boundary address.

If the boundary address update (BAUPD) control bit is also set, the core will shift the boundary upwards whenever the the address directly above the boundary is written to. Since the written data is now below the boundary, it will be written using the opposite code. The write can be done with any size supported by the controller.

23.3.9 Data muxing

When code B is used instead of code A, the upper half of the checkbits are unused. The controller supports switching in this part of the data bus to replace another faulty part of the bus. To do this, one sets the DATAMUX field to a value between 1-4 to replace a quarter of the data bus, or to 5 to replace the active checkbit half.

23.3.10 Memory fault recovery

The above features are designed to, when combined and integrated correctly, make the system cabable to deal with a permanent fault in an external memory chip.

A basic sequence of events is as follows:

1. The system is running correctly with EDAC enabled and the larger code A is used.
2. A memory chip gets a fault and delivers incorrect data. The DDR2 controller keeps delivering error-free data but reports a correctable error on every read access.
3. A logging device (such as the memory scrubber core) registers the high frequency of correctable errors and signals an interrupt.
4. The CPU performs a probe using the DDR2 FT diagnostic registers to confirm that the error is permanent and on which physical lane the error is.
5. After determining that a permanent fault has occurred, the CPU reconfigures the FTDDR2 controller as follows (all configuration register fields changed with a single register write):

   The data muxing control field is set so the top checkbit half replaces the failed part of the data bus.

   The code boundary register is set to the lowest memory address.

   The boundary address enable and boundary address update enable bits are set.

   The mask correctable error bit is set
6. The memory data and checkbits are now regenerated using locked read-write cycles to use the smaller code and replace the broken data with the upper half of the checkbit bus. This can be done in hardware using an IP core, such as the AHB memory scrubber, or by some other means depending on system design.
7. After the whole memory has been regenerated, the CPU disables the code boundary, changes the code selection field to code B, and unsets the mask correctable error bit.

After this sequence, the system is now again fully operational, but running with the smaller code and replacement chip and can again recover from any single-nibble error. Note that during this sequence, it is possible for the system to operate and other masters can both read and write to memory while the regeneration is ongoing.

23.4 Registers

The DDR2SPA core implements between 5 and 12 control registers, depending on the FT generic and target technology. The registers are mapped into AHB I/O address space defined by the AHB BAR1 of the core. Only 32-bit single-accesses are supported to the registers.

Older revisions of the core only have registers DDRCFG1-4, which are aliased on the following addresses. For that reason, check the REG5 bit in DDR2CFG2 before using these bits for backward compatibility.

For backward compatibility, some of the bits in DDR2CFG5 are mirrored in other registers. Writing to these bits will affect the contents of DDR2CFG5 and vice versa.

*Table 208.*DDR2 controller registers

Address offset - AHB I/O - BAR1	**Register**
0x00	DDR2 SDRAM control register (DDR2CFG1)
0x04	DDR2 SDRAM configuration register (DDR2CFG2)
0x08	DDR2 SDRAM control register (DDR2CFG3)
0x0C	DDR2 SDRAM control register (DDR2CFG4)
0x10*	DDR2 SDRAM control register (DDR2CFG5)
0x14*	Reserved
0x18	DDR2 Technology specific register (DDR2TSR1)
0x1C*	DDR2 Technology specific register (DDR2TSR2)
0x20	DDR2 FT Configuration Register (FT only) (DDR2FTCFG)
0x24	DDR2 FT Diagnostic Address register (FT only) (DDR2FTDA)
0x28	DDR2 FT Diagnostic Checkbit register (FT only) (DDR2FTDC)
0x2C	DDR2 FT Diagnostic Data register (FT only) (DDR2FTDD)
0x30	DDR2 FT Code Boundary Register (FT only) (DDR2FTBND)

* Older DDR2SPA versions contain aliases of DDR2CFG1-4 at these addresses. Therefore, check bit 15 of DDR2CFG2 before using these registers.

Table 209. DDR2 SRAM control register 1 (DDR2CFG1)

31	30	29 28	27	26	25 23	22 21	20 18	17	16	15	14 0
Refresh	OCD	EMR	bank size 3	(TRCD)	SDRAM bank size2:0	SDRAM col. size	SDRAM command	PR	IN	CE	SDRAM refresh load value

31	SDRAM refresh. If set, the SDRAM refresh will be enabled.
30	OCD operation
29: 28	Selects Extended mode register (1,2,3)
27	SDRAM banks size bit 3. By enabling this bit the memory size can be set to “1000” = 2048 Mbyte and “1001” = 4096 Mbyte. See the section on big-memory support.
26	Lowest bit of TRCD field in DDR2CFG4, for backward compatibility
25: 23	SDRAM banks size. Defines the decoded memory size for each SDRAM chip select: “000”= 8 Mbyte, “001”= 16 Mbyte, “010”= 32 Mbyte.... “111”= 1024 Mbyte.
22: 21	SDRAM column size. “00”=512, “01”=1024, “10”=2048, “11”=4096
20: 18	SDRAM command. Writing a non-zero value will generate an SDRAM command: “010”=PRE-CHARGE, “100”=AUTO-REFRESH, “110”=LOAD-COMMAND-REGISTER, “111”=LOAD-EXTENDED-COMMAND-REGISTER. The field is reset after command has been executed.
17	PLL Reset. This bit is used to set the PLL RESET bit during LOAD-CONFIG-REG commands.
16	Initialize (IN). Set to ‘1’ to perform power-on DDR RAM initialisation. Is automatically cleared when initialisation is completed.
15	Clock enable (CE). This value is driven on the CKE inputs of the DDR RAM. Should be set to ‘1’ for correct operation.
14: 0	The period between each AUTO-REFRESH command - Calculated as follows: tREFRESH = ((reload value) + 1) / DDRCLOCK

Table 210. DDR2 SDRAM configuration register 2 (DDR2CFG2) (read-only)

31 26	25 18	17	16	15	14 12	11 0
RESERVED	PHY Tech	BIG	FTV	REG5	Data width	DDR Clock frequency

31: 26	Reserved
25: 18	PHY technology identifier (read-only), value 0 is for generic/unknown
17	Big memory support, if ‘1’ then memory can be set between 32 Mbyte and 4 Gbyte, if ‘0’ then memory size can be set between 8 Mbyte and 1 Gbyte (read-only).
16	Reads ‘1’ if the controller is fault-tolerant version and EDAC registers exist (read-only)
15	Reads ‘1’ if DDR2CFG5 register exists (read-only)
14: 12	SDRAM data width: “001” = 16 bits, “010” = 32 bits, “011” = 64 bits (read-only)
11: 0	Frequency of the (external) DDR clock (read-only)

Table 211. DDR2 SDRAM configuration register 3 (DDR2CFG3)

31	30 29	28	27 23	22 18	17 16	15 8	7 0
	PLL	(TRP)	tWR	(TRFC)	RD	inc/dec delay	Update delay

31	Reset byte delay
30: 29	PLL_SKEW Bit 29: Update clock phase Bit 30: 1 = Inc / 0 = Dec clock phase
28	Lowest bit of DDR2CFG4 TRP field for backward compatibility
27: 23	SDRAM write recovery time. tWR will be equal to field value - 2DDR clock cycles
22: 18	Lower 5 bits of DDR2CFG4 TRFC field for backward compatibility.
17: 16	Number of added read delay cycles, default = 1
15: 8	Set to ‘1’ to increment byte delay, set to ‘0’ to decrement delay
7: 0	Set to ‘1’ to update byte delay

Table 212. DDR2 SDRAM configuration register 4 (DDR2CFG4)

31 28	27 24	23 22	21	20 14	13	12 11	10 9	8	7 0
inc/dec CB delay	Update CB delay	RDH	REG	RESERVED	TRTP	RES	TCL	B8	DQS gating offset

31: 28	Set to ‘1’ to increment checkbits byte delay, set to ‘0’ to decrement delay
27: 24	Set to ‘1’ to update checkbits byte delay
23: 22	Read delay high bits, setting this field to N adds 4 x N read delay cycles
21	Registered memory (1 cycle extra latency on control signals)
20: 14	Reserved
13	SDRAM read-to-precharge timing, tRTP will be equal to field value + 2 DDR-clock cycles.
12: 11	Reserved
10: 9	SDRAM CAS latency timing. CL will be equal to field value + 3 DDR-clock cycles. Note: You must reprogram the memory’s MR register after changing this value
8	Enables address generation for DDR2 chips with eight banks 1=addressess generation for eight banks 0=address generation for four banks
7: 0	Number of half clock cycles for which the DQS input signal will be active after a read command is given. After this time the DQS signal will be gated off to prevent latching of faulty data. Only valid if the dqsgating generic is enabled.

Table 213. DDR2 SDRAM configuration register 5 (DDR2CFG5)

31	30 28	27 26	25 18	17 16	15	14 11	10 8	7 5	4 0
R	TRP	RES	TRFC	ODT	DS	RESERVED	TRCD	RESERVED	TRAS

31	Reserved
30: 28	SDRAM tRP timing. tRP will be equal to 2 + field value DDR-clock cycles
27: 26	Reserved
25: 18	SDRAM tRFC timing. tRFC will be equal to 3 + field-value DDR-clock cycles.
17: 16	SDRAM-side on-die termination setting (0=disabled, 1-3=75/150/50 ohm) Note: You must reprogram the EMR1 register after changing this value.
15	SDRAM-side output drive strength control (0=full strength, 1=half strength) Note: You must reprogram the EMR1 register after changing this value
14: 11	Reserved
10: 8	SDRAM RAS-to-CAS delay (TRCD). tRCD will be equal to field value + 2 DDR-clock cycles
7: 5	Reserved
4: 0	SDRAM RAS to precharge timing. TRAS will be equal to 2+ field value DDR-clock cycles

Table 214. DDR2 FT configuration register (DDR2FTCFG)

31 20	19	18	16	15					8	7 5	4	3	2	1	0
Diag data read error location	DDERR	RESERVED								DATAMUX	CEM	BAUPD	BAEN	CODE	EDEN

31: 20	Bit field describing location of corrected errors for last diagnostic data read (read-only) One bit per byte lane in 64+32-bit configuration
19	Set high if last diagnostic data read contained an uncorrectable error (read-only)
18: 16	Data width, read-only field. 001=16+8, 010=32+16, 011=64+32 bits
15: 8	Reserved
7: 5	Data mux control, setting this nonzero switchess in the upper checkbit half with another data lane. For 64-bit interface 000 = no switching 001 = Data bits 15:0, 010 = Data bits 31:16, 011: Data bits 47:32, 100: Data bits 63:48, 101 = Checkbits 79:64, 110,111 = Undefined
4	If set high, the correctable error signal is masked out.
3	Enable automatic boundary shifting on write
2	Enable the code boundary
1	Code selection, 0=Code A (64+32/32+16/16+8), 1=Code B (64+16/32+8)
0	EDAC Enable

Table 215. DDR2 FT Diagnostic Address (DDR2FTDA)

31 2	1 0
MEMORY ADDRESS	RESERVED

31: 3	Address to memory location for checkbit read/write, 64/32-bit aligned for checkbits/data
1: 0	Reserved (address bits always 0 due to alignment)

Table 216. DDR2 FT Diagnostic Checkbits (DDR2FTDC)

31 24	23 16	15 8	7 0
CHECKBITS D	CHECKBITS C	CHECKBITS B	CHECKBITS A

31: 24	Checkbits for part D of 64-bit data word (undefined for code B)
23: 16	Checkbits for part C of 64-bit data word (undefined for code B)
15: 8	Checkbits for part B of 64-bit data word
7: 0	Checkbits for part A of 64-it data word.

Table 217. DDR2 FT Diagnostic Data (DDR2FTDD)

31 0
DATA BITS

31: 0	Uncorrected data bits for 32-bit address set in DDR2FTDA

Table 218. DDR2 FT Boundary Address Registre (DDR2FTBND)

31 3	2 0
CHECKBIT CODE BOUNDARY ADDRESS	0

31: 3	Code boundary address, 64-bit aligned
2: 0	Zero due to alignment

23.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x02E. The revision decribed in this document is revision 1. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

23.6 Configuration options

Table 219 shows the configuration options of the core (VHDL generics).

*Table 219.*Configuration options

Generic	Function	Allowed range	Default
fabtech	PHY technology selection	virtex4, virtex5, stratix3	virtex4
memtech	Technology selection for DDR FIFOs	inferred, virtex2, virtex4, spartan3e, altera	inferred
hindex	AHB slave index	0 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where DDR control register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space	0 - 16#FFF#	16#FFF#
ddrbits	Data bus width of external DDR memory	16, 32, 64	16
MHz	DDR clock input frequency in MHz.	10 - 200	100
clkmul, clkdiv	The DDR input clock is multiplied with the clkmul generic and divided with clkdiv to create the final DDR clock	2 - 32	2
rstdel	Clock reset delay in micro-seconds.	1 - 1023	200
col	Default number of column address bits	9 - 12	9
Mbyte	Default memory chip select bank size in Mbyte	8 - 1024	16
pwron	Enable SDRAM at power-on initialization	0 - 1	0
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
ahbfreq	Frequency in MHz of the AHB clock domain	1 - 1023	50
readdly	Additional read latency cycles (used to increase calibration range)	0-3	1
TRFC	Reset value for the tRFC timing parameter in ns.	75-155	130
ddelayb0*	Input data delay for bit[7:0]	0-63	0
ddelayb1*	Input data delay for bit[15:8]	0-63	0
ddelayb2*	Input data delay for bit[23:16]	0-63	0
ddelayb3*	Input data delay for bit[31:24]	0-63	0
ddelayb4*	Input data delay for bit[39:32]	0-63	0
ddelayb5*	Input data delay for bit[47:40]	0-63	0
ddelayb6*	Input data delay for bit[55:48]	0-63	0
ddelayb7*	Input data delay for bit[63:56]	0-63	0
cbdelayb0*	Input data delay for checkbit[7:0]	0-63	0
cbdelayb1*	Input data delay for checkbit[15:8]	0-63	0
cbdelayb2*	Input data delay for checkbit[23:16]	0-63	0
cbdelayb3*	Input data delay for checkbit[31:24]	0-63	0

*Table 219.*Configuration options

Generic	Function	Allowed range	Default
numidelctrl*	Number of IDELAYCTRL the core will instantiate	-	4
norefclk*	Set to 1 if no 200 MHz reference clock is connected to clkref200 input.	0-1	0
odten	Enable odt: 0 = Disabled, 1 = 75Ohm, 2 =150Ohm, 3 = 50Ohm	0-3	0
rskew**	Set the phase relationship between the DDR controller clock and the input data sampling clock. Sets the phase in ps.	0 - 9999	0
octen**	Enable on chip termination: 1 = enabled, 0 = disabled	0 - 1	0
dqsgating***	Enable gating of DQS signals when doing reads. 1 = enable, 0 = disable	0 - 1	0
nosync	Disable insertion of synchronization registers between AHB clock domain and DDR clock domain. This can be done if the AHB clock's rising edges always are in phase with a rising edge on the DDR clock. If this generic is set to 1 the clkmul and clkdiv generics should be equal. Otherwise the DDR controller may scale the incoming clock and loose the clocks' edge alignment in the process.	0 - 1	0
eightbanks	Enables address generation for DDR2 chips with eight banks. The DDR_BA is extended to 3 bits if set to 1.	0 - 1	0
dqsse	Single-ended DQS. The value of this generic is written to bit 10 in the memory's Extended Mode register. If this bit is 1 DQS is used in a single-ended mode. Currently this bit should only, and must be, set to 1 when the Stratix2 DDR2 PHY is used. This is the only PHY that supports single ended DQS without modification.	0 - 1	0
burstlen	DDR access burst length in 32-bit words	8,16,32,..,256	8
ahbbits	AHB bus width	32,64,128,256	AHBDW
ft	Enable fault-tolerant version	0 - 1	0
ftbits	Extra DDR data bits used for checkbits	0,8,16,32	0
bigmem	Big memory support, changes the range of supported total memory bank sizes from 8MB-1GB to 32MB-4GB	0 - 1	0
raspipe	Enables an extra pipeline stage in the address decoding to improve timing at the cost of one DDR-cycle latency	0 - 1	0
* only available in Virtex4/5 implementation. ** only available in Altera and Spartan3 implementations. *** only available on Nextreme/eASIC implementations			

23.7 Implementation

23.7.1 Technology mapping

The core has two technology mapping VHDL generics: *memtech* and *padtech*. The VHDL generic *memtech* controls the technology used for memory cell implementation. The VHDL generic *padtech* controls the technology used in the PHY implementation. See the GRLIB Users's Manual for available settings.

23.7.2 RAM usage

The FIFOs in the core are implemented with the *syncram_2p* (with separate clock for each port) component found in the technology mapping library (TECHMAP). The number of RAMs used for the FIFO implementation depends om the DDR data width, set by the *ddrbits* VHDL generic, and the AHB bus width in the system.

The RAM block usage is tabulated below for the default burst length of 8 words. If the burst length is doubled, the depths for all the RAMs double as well but the count and width remain the same.

*Table 220.*Block-RAM usage for default burst length

DDR width	AHB width	Write FIFO block-RAM usage			Read-FIFO block-RAM usage			Total RAM count
		Count	Depth	Width	Count	Depth	Width	
16	32	1	16	32	1	8	32	2
16	64	2	8	32	2	4	32	4
16	128	4	4	32	4	2	32	8
16	256	8	2	32	8	1	32	16
32	32	2	8	32	1	4	64	3
32	64	2	8	32	1	4	64	3
32	128	4	4	32	2	2	64	6
32	256	8	2	32	4	1	64	12
64	32	4	4	32	1	2	128	5
64	64	4	4	32	1	2	128	5
64	128	4	4	32	1	2	128	5
64	256	8	2	32	2	1	128	10

23.7.3 Xilinx Virtex-specific issues

The Xilinx tools require one IDELAYCTRL macro to be instantiated in every region where the IDELAY feature is used. Since the DDR2 PHY uses the IDELAY on every data (DQ) pin, this affects the DDR2 core. For this purpose, the core has a numidelctrl generic, controlling how many IDELAYCTRL's get instantiated in the PHY.

The tools allow for two ways to do this instantiation:

- Instantiate the same number of IDELAYCTRL as the number of clock regions containing DQ pins and place the instances manually using UCF LOC constraints.
- Instantiate just one IDELAYCTRL, which the ISE tools will then replicate over all regions.

The second solution is the simplest, since you just need to set the numidelctrl to 1 and no extra constraints are needed. However, this approach will not work if IDELAY is used anywhere else in the FPGA design.

For more information on IDELAYCTRL, see Xilinx Virtex4/5 User's Guide.

23.7.4 Design tools

To run the design in Altera Quartus 7.2 you have to uncomment the lines in the .qsf file that assigns the MEMORY_INTERFACE_DATA_PIN_GROUP for the DDR2 interface. These group assignments result in error when Altera Quartus 8.0 is used.

23.8 Signal descriptions

Table 221 shows the interface signals of the core (VHDL ports).

*Table 221.*Signal descriptions

Signal name	**Type**	**Function**	**Active**
RST_DDR	Input	Reset input for the DDR PHY	Low
RST_AHB	Input	Reset input for AHB clock domain	Low
CLK_DDR	Input	DDR input Clock	-
CLK_AHB	Input	AHB clock	-
CLKREF200	Input	200 MHz reference clock	-
LOCK	Output	DDR clock generator locked	High
CLKDDRO		Internal DDR clock output after clock multiplication	
CLKDDRI		Clock input for the internal DDR clock domain. Must be connected to CLKDDRO.	
AHBSI	Input	AHB slave input signals	-
AHBSO	Output	AHB slave output signals	-
DDR_CLK[2:0]	Output	DDR memory clocks (positive)	High
DDR_CLKB[2:0]	Output	DDR memory clocks (negative)	Low
DDR_CLK_FB_OUT	Output	DDR data synchronization clock, connect this to a signal path with equal length of the DDR_CLK trace + DDR_DQS trace	-
DDR_CLK_FB	Input	DDR data synchronization clock, connect this to the other end of the signal path connected to DDR_CLK_FB_OUT	-
DDR_CKE[1:0]	Output	DDR memory clock enable	High
DDR_CSB[1:0]	Output	DDR memory chip select	Low
DDR_WEB	Output	DDR memory write enable	Low
DDR_RASB	Output	DDR memory row address strobe	Low
DDR_CASB	Output	DDR memory column address strobe	Low
DDR_DM[(DDRBITS+FTBITS)/8-1:0]	Output	DDR memory data mask	Low
DDR_DQS[(DDRBITS+FTBITS)/8-1:0]	Bidir	DDR memory data strobe	Low
DDR_DQSN[(DDRBITS+FTBITS)/8-1:0]	Bidir	DDR memory data strobe (inverted)	High
DDR_AD[13:0]	Output	DDR memory address bus	Low
DDR_BA[2 or 1:0] 3)	Output	DDR memory bank address	Low
DDR_DQ[DDRBITS+FTBITS-1:0]	BiDir	DDR memory data bus	-
DDR_ODT[1:0]	Output	DDR memory odt	Low

1) see GRLIB IP Library User’s Manual
2) Polarity selected with the oepol generic
3) DDR_BA[2:0] if the eightbanks generic is set to 1 else DDR_BA[1:0]
4) Only used on Virtex4/5
5) Only used on Spartan3

23.9 Library dependencies

Table 222 shows libraries used when instantiating the core (VHDL libraries).

*Table 222.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

23.10 Component declaration

```
component ddr2spa
  generic (
    fabtech : integer := 0;
    memtech : integer := 0;
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#f00#;
    ioaddr  : integer := 16#000#;
    iomask  : integer := 16#fff#;
    MHz     : integer := 100;
    clkmul  : integer := 2;
    clkdiv  : integer := 2;
    col     : integer := 9;
    Mbyte   : integer := 16;
    rstdel  : integer := 200;
    pwron   : integer := 0;
    oepol   : integer := 0;
    ddrbits : integer := 16;
    ahbfreq : integer := 50;
    readdly : integer := 1;
    ddelayb0: integer := 0;
    ddelayb1: integer := 0;
    ddelayb2: integer := 0;
    ddelayb3: integer := 0;
    ddelayb4: integer := 0;
    ddelayb5: integer := 0;
    ddelayb6: integer := 0;
    ddelayb7: integer := 0
  );
  port (
    rst_ddr    : in  std_ulogic;
    rst_ahb    : in  std_ulogic;
    clk_ddr    : in  std_ulogic;
    clk_ahb    : in  std_ulogic;
    clkref200 : in std_ulogic;
    lock       : out std_ulogic;-- DCM locked
    clkddro    : out std_ulogic;-- DCM locked
    clkddri    : in  std_ulogic;
    ahbsi      : in  ahb_slv_in_type;
    ahbso      : out ahb_slv_out_type;
    ddr_clk    : out std_logic_vector(2 downto 0);
    ddr_clkb   : out std_logic_vector(2 downto 0);
    ddr_cke    : out std_logic_vector(1 downto 0);
    ddr_csb    : out std_logic_vector(1 downto 0);
    ddr_web    : out std_ulogic;                       -- ddr write enable
    ddr_rasb   : out std_ulogic;                       -- ddr ras
    ddr_casb   : out std_ulogic;                       -- ddr cas
    ddr_dm     : out std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dm
    ddr_dqs    : inout std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dqs
    ddr_dqsn   : inout std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dqs
    ddr_ad     : out std_logic_vector (13 downto 0);   -- ddr address
    ddr_ba     : out std_logic_vector (1 downto 0);    -- ddr bank address
    ddr_dq     : inout  std_logic_vector (ddrbits-1 downto 0); -- ddr data
    ddr_odt    : out std_logic_vector(1 downto 0) -- odt
  );
  end component;
```

23.11 Instantiation

This example shows how the core can be instantiated.

The DDR SDRAM controller decodes SDRAM area at 0x40000000 - 0x7FFFFFFF. The DDR2 SDRAM registers are mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;

entity ddr_Interface is
  port (
    ddr_clk  : out std_logic_vector(2 downto 0);
    ddr_clkb : out std_logic_vector(2 downto 0);
    ddr_cke  : out std_logic_vector(1 downto 0);
    ddr_csb  : out std_logic_vector(1 downto 0);
    ddr_web  : out std_ulogic;                       -- ddr write enable
    ddr_rasb : out std_ulogic;                       -- ddr ras
    ddr_casb : out std_ulogic;                       -- ddr cas
    ddr_dm   : out std_logic_vector (7 downto 0);    -- ddr dm
    ddr_dqs  : inout std_logic_vector (7 downto 0);    -- ddr dqs
    ddr_dqsn : inout std_logic_vector (7 downto 0);    -- ddr dqsn
    ddr_ad   : out std_logic_vector (13 downto 0);   -- ddr address
    ddr_ba   : out std_logic_vector (1 downto 0);    -- ddr bank address
    ddr_dq   : inout std_logic_vector (63 downto 0); -- ddr data
    ddr_odt  : out std_logic_vector (1 downto 0) -- ddr odt
);
end;

architecture rtl of mctrl_ex is

  -- AMBA bus
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal clkml, lock, clk_200,
signal clk_200 : std_ulogic; -- 200 MHz reference clock
signal ddrclkin, ahbclk : std_ulogic; -- DDR input clock and AMBA sys clock
signal rstn  : std_ulogic; -- Synchronous reset signal
signal reset : std_ulogic; -- Asynchronous reset signal


begin

-- DDR controller


ddrc : ddr2spa generic map ( fabtech => virtex4, ddrbits => 64, memtech => memtech,
hindex => 4, haddr => 16#400#, hmask => 16#F00#, ioaddr => 1,
pwron => 1, MHz => 100, col => 9, Mbyte => 32, ahbfreq => 50, ddrbits => 64,
readdly => 1, ddelayb0 => 0, ddelayb1 => 0, ddelayb2 => 0, ddelayb3 => 0,
ddelayb4 => 0, ddelayb5 => 0, ddelayb6 => 0, ddelayb7 => 0)
port map (
reset, rstn, ddrclkin, ahbclk, clk_200, lock, clkml, clkml,  ahbsi, ahbso(4),
ddr_clk, ddr_clkb,
ddr_cke, ddr_csb, ddr_web, ddr_rasb, ddr_casb,
ddr_dm, ddr_dqs, ddr_adl, ddr_ba, ddr_dq, ddr_odt);
```

24 DIV32 - Signed/unsigned 64/32 divider module

24.1 Overview

The divider module performs signed/unsigned 64-bit by 32-bit division. It implements the radix-2 non-restoring iterative division algorithm. The division operation takes 36 clock cycles. The divider leaves no remainder. The result is rounded towards zero. Negative result, zero result and overflow (according to the overflow detection method B of SPARC V8 Architecture manual) are detected.

24.2 Operation

The division is started when '1' is samples on DIVI.START on positive clock edge. Operands are latched externally and provided on inputs DIVI.Y, DIVI.OP1 and DIVI.OP2 during the whole operation. The result appears on the outputs during the clock cycle following the clock cycle after the DIVO.READY was asserted. Asserting the HOLD input at any time will freeze the operation, until HOLDN is de-asserted.

24.3 Signal descriptions

Table 223 shows the interface signals of the core (VHDL ports).

*Table 223.*Signal declarations

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
HOLDN	N/A	Input	Hold	Low
DIVI	Y[32:0]	Input	Dividend - MSB part Y[32] - Sign bit Y[31:0] - Dividend MSB part in 2's complement format	High
	OP1[32:0]		Dividend - LSB part OP1[32] - Sign bit OP1[31:0] - Dividend LSB part in 2's complement format	High
	FLUSH		Flush current operation	High
	SIGNED		Signed division	High
	START		Start division	High
DIVO	READY	Output	The result is available one clock after the ready signal is asserted.	High
	NREADY		The result is available three clock cycles, assuming hold=HIGH, after the nready signal is asserted.	High
	ICC[3:0]		Condition codes ICC[3] - Negative result ICC[2] - Zero result ICC[1] - Overflow ICC[0] - Not used. Always '0'.	High
	RESULT[31:0]		Result	High

24.4 Library dependencies

Table 224 shows libraries used when instantiating the core (VHDL libraries).

*Table 224.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	ARITH	Signals, component	Divider module signals, component declaration

24.5 Component declaration

The core has the following component declaration.

```
component div32
port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    holdn   : in  std_ulogic;
    divi    : in  div32_in_type;
    divo    : out div32_out_type
);
end component;
```

24.6 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use gaisler.arith.all;

.
.
.

signal divi  : div32_in_type;
signal divo  : div32_out_type;

begin

div0 : div32 port map (rst, clk, holdn, divi, divo);

end;
```

25 DSU3 - LEON3 Hardware Debug Support Unit

25.1 Overview

To simplify debugging on target hardware, the LEON3 processor implements a debug mode during which the pipeline is idle and the processor is controlled through a special debug interface. The LEON3 Debug Support Unit (DSU) is used to control the processor during debug mode. The DSU acts as an AHB slave and can be accessed by any AHB master. An external debug host can therefore access the DSU through several different interfaces. Such an interface can be a serial UART (RS232), JTAG, PCI, USB or ethernet. The DSU supports multi-processor systems and can handle up to 16 processors.

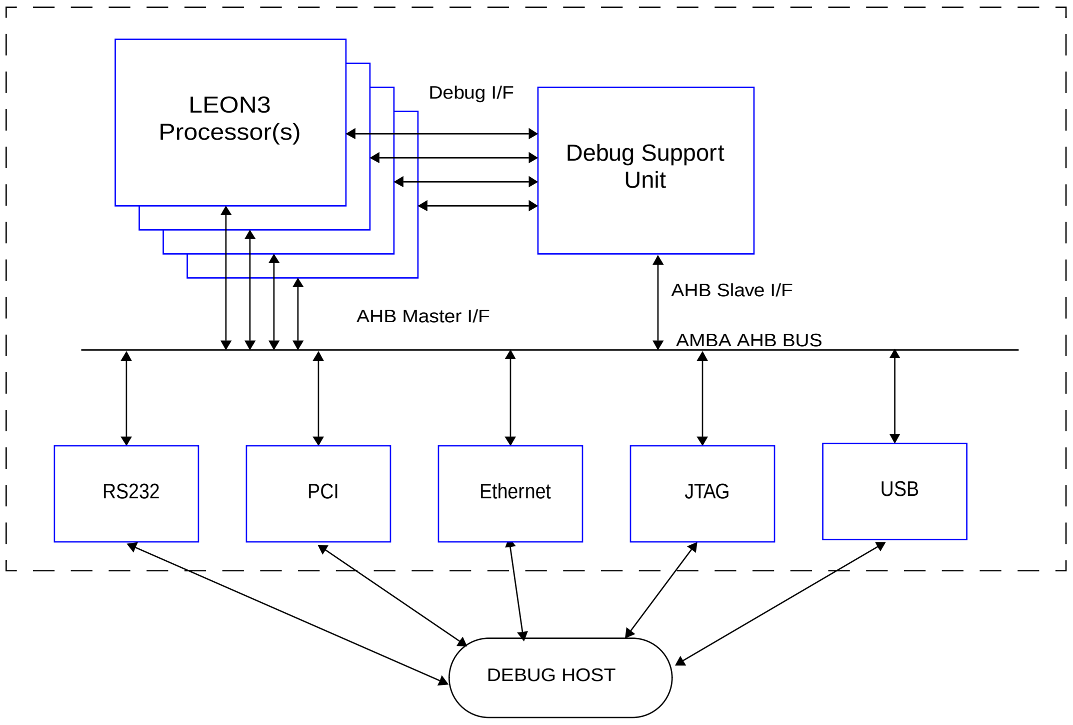


Figure 70. LEON3/DSU Connection

25.2 Operation

Through the DSU AHB slave interface, any AHB master can access the processor registers and the contents of the instruction trace buffer. The DSU control registers can be accessed at any time, while the processor registers, caches and trace buffer can only be accessed when the processor has entered debug mode. In debug mode, the processor pipeline is held and the processor state can be accessed by the DSU. Entering the debug mode can occur on the following events:

- executing a breakpoint instruction (ta 1)
- integer unit hardware breakpoint/watchpoint hit (trap 0xb)
- rising edge of the external break signal (DSUBRE)
- setting the break-now (BN) bit in the DSU control register
- a trap that would cause the processor to enter error mode
- occurrence of any, or a selection of traps as defined in the DSU control register
- after a single-step operation
- one of the processors in a multiprocessor system has entered the debug mode
- DSU AHB breakpoint or watchpoint hit

The debug mode can only be entered when the debug support unit is enabled through an external signal (DSUEN). For DSU break (DSUBRE), and the break-now BN bit, to have effect the Break-on-IU-watchpoint (BW) bit must be set in the DSU control register. This bit is set when DSUBRE is active after reset and should also be set by debug monitor software (like Aeroflex Gaisler's GRMON) when initializing the DSU. When the debug mode is entered, the following actions are taken:

- PC and nPC are saved in temporary registers (accessible by the debug unit)
- an output signal (DSUACT) is asserted to indicate the debug state
- the timer unit is (optionally) stopped to freeze the LEON timers and watchdog

The instruction that caused the processor to enter debug mode is not executed, and the processor state is kept unmodified. Execution is resumed by clearing the BN bit in the DSU control register or by de-asserting DSUEN. The timer unit will be re-enabled and execution will continue from the saved PC and nPC. Debug mode can also be entered after the processor has entered error mode, for instance when an application has terminated and halted the processor. The error mode can be reset and the processor restarted at any address.

When a processor is in the debug mode, an access to ASI diagnostic area is forwarded to the IU which performs access with ASI equal to value in the DSU ASI register and address consisting of 20 LSB bits of the original address.

25.3 AHB Trace Buffer

The AHB trace buffer consists of a circular buffer that stores AHB data transfers. The address, data and various control signals of the AHB bus are stored and can be read out for later analysis. The trace buffer is 128 bits wide, the information stored is indicated in the table below:

*Table 225.*AHB Trace buffer data allocation

Bits	Name	Definition
127	AHB breakpoint hit	Set to '1' if a DSU AHB breakpoint hit occurred.
126	-	Not used
125:96	Time tag	DSU time tag counter
95	-	Not used
94:80	Hirq	AHB HIRQ[15:1]
79	Hwrite	AHB HWRITE
78:77	Htrans	AHB HTRANS
76:74	Hsize	AHB HSIZE
73:71	Hburst	AHB HBURST
70:67	Hmaster	AHB HMASTER
66	Hmastlock	AHB HMASTLOCK
65:64	Hresp	AHB HRESP
63:32	Load/Store data	AHB HRDATA or HWDATA
31:0	Load/Store address	AHB HADDR

In addition to the AHB signals, the DSU time tag counter is also stored in the trace.

The trace buffer is enabled by setting the enable bit (EN) in the trace control register. Each AHB transfer is then stored in the buffer in a circular manner. The address to which the next transfer is written is held in the trace buffer index register, and is automatically incremented after each transfer. Tracing is stopped when the EN bit is reset, or when a AHB breakpoint is hit. Tracing is temporarily suspended when the processor enters debug mode. Note that neither the trace buffer memory nor the breakpoint registers (see below) can be read/written by software when the trace buffer is enabled.

25.4 Instruction trace buffer

The instruction trace buffer consists of a circular buffer that stores executed instructions. The instruction trace buffer is located in the processor, and read out via the DSU. The trace buffer is 128 bits wide, the information stored is indicated in the table below:

*Table 226.*Instruction trace buffer data allocation

Bits	**Name**	**Definition**
127	-	Unused
126	Multi-cycle instruction	Set to ‘1’ on the second and third instance of a multi-cycle instruction (LDD, ST or FPOP)
125:96	Time tag	The value of the DSU time tag counter
95:64	Load/Store parameters	Instruction result, Store address or Store data
63:34	Program counter	Program counter (2 lsb bits removed since they are always zero)
33	Instruction trap	Set to ‘1’ if traced instruction trapped
32	Processor error mode	Set to ‘1’ if the traced instruction caused processor error mode
31:0	Opcode	Instruction opcode

During tracing, one instruction is stored per line in the trace buffer with the exception of multi-cycle instructions. Multi-cycle instructions are entered two or three times in the trace buffer. For store instructions, bits [63:32] correspond to the store address on the first entry and to the stored data on the second entry (and third in case of STD). Bit 126 is set on the second and third entry to indicate this. A double load (LDD) is entered twice in the trace buffer, with bits [63:32] containing the loaded data. Multiply and divide instructions are entered twice, but only the last entry contains the result. Bit 126 is set for the second entry. For FPU operation producing a double-precision result, the first entry puts the MSB 32 bits of the results in bit [63:32] while the second entry puts the LSB 32 bits in this field.

When the processor enters debug mode, tracing is suspended. The trace buffer and the trace buffer control register can be read and written while the processor is in the debug mode. During the instruction tracing (processor in normal mode) the trace buffer and the trace buffer control register can not be accessed.

25.5 DSU memory map

The DSU memory map can be seen in table 227 below. In a multiprocessor systems, the register map is duplicated and address bits 27 - 24 are used to index the processor.

*Table 227.*DSU memory map

Address offset	**Register**
0x000000	DSU control register
0x000008	Time tag counter
0x000020	Break and Single Step register
0x000024	Debug Mode Mask register
0x000040	AHB trace buffer control register
0x000044	AHB trace buffer index register
0x000050	AHB breakpoint address 1
0x000054	AHB mask register 1
0x000058	AHB breakpoint address 2
0x00005c	AHB mask register 2
0x100000 - 0x10FFFF	Instruction trace buffer (..0: Trace bits 127 - 96, ..4: Trace bits 95 - 64, ..8: Trace bits 63 - 32, ..C : Trace bits 31 - 0)
0x110000	Instruction Trace buffer control register
0x200000 - 0x210000	AHB trace buffer (..0: Trace bits 127 - 96, ..4: Trace bits 95 - 64, ..8: Trace bits 63 - 32, ..C : Trace bits 31 - 0)
0x300000 - 0x3007FC	IU register file, port1 (%asr16.dpsel = 0) IU register file, port 2 (%asr16.dpsel = 1)
0x300800 - 0x300FFC	IU register file check bits (LEON3FT only)
0x301000 - 0x30107C	FPU register file
0x301800 - 0x30187C	FPU register file check bits (LEON3FT only)
0x400000 - 0x4FFFFC	IU special purpose registers
0x400000	Y register
0x400004	PSR register
0x400008	WIM register
0x40000C	TBR register
0x400010	PC register
0x400014	NPC register
0x400018	FSR register
0x40001C	CPSR register
0x400020	DSU trap register
0x400024	DSU ASI register
0x400040 - 0x40007C	ASR16 - ASR31 (when implemented)

0x700000 - 0x7FFFFC	ASI diagnostic access (ASI = value in DSU ASI register, address = address[19:0])
	ASI = 0x9 : Local instruction RAM, ASI = 0xB : Local data RAM
	ASI = 0xC : Instruction cache tags, ASI = 0xD : Instruction cache data
	ASI = 0xE : Data cache tags, ASI = 0xF : Data cache data
	ASI = 0x1E : Separate snoop tags

The addresses of the IU registers depends on how many register windows has been implemented:

- %o*n* : 0x300000 + (((psr.cwp * 64) + 32 + *n**4) mod (NWINDOWS*64))
- %l*n* : 0x300000 + (((psr.cwp * 64) + 64 + *n**4) mod (NWINDOWS*64))
- %i*n* : 0x300000 + (((psr.cwp * 64) + 96 + *n**4) mod (NWINDOWS*64))
- %g*n* : 0x300000 + (NWINDOWS*64) + n*4
- %f*n* : 0x301000 + *n**4

25.6 DSU registers

25.6.1 DSU control register

The DSU is controlled by the DSU control register:

.

31	11	10	9	8	7	6	5	4	3	2	1	0
	PW	HL	PE	EB	EE	DM	BZ	BX	BS	BW	BE	TE

Figure 71. DSU control register

[0]: Trace enable (TE). Enables instruction tracing. If set the instructions will be stored in the trace buffer. Remains set when then processor enters debug or error mode.

[1]: Break on error (BE) - if set, will force the processor to debug mode when the processor would have entered error condition (trap in trap).

[2]: Break on IU watchpoint (BW)- if set, debug mode will be forced on a IU watchpoint (trap 0xb).

[3]: Break on S/W breakpoint (BS) - if set, debug mode will be forced when an breakpoint instruction (ta 1) is executed.

[4]: Break on trap (BX) - if set, will force the processor into debug mode when any trap occurs.

[5]: Break on error traps (BZ) - if set, will force the processor into debug mode on all *except* the following traps: priviledged_instruction, fpu_disabled, window_overflow, window_underflow, asynchronous_interrupt, ticc_trap.

[6]: Debug mode (DM). Indicates when the processor has entered debug mode (read-only).

[7]: EE - value of the external DSUEN signal (read-only)

[8]: EB - value of the external DSUBRE signal (read-only)

[9]: Processor error mode (PE) - returns ‘1’ on read when processor is in error mode, else ‘0’. If written with ‘1’, it will clear the error and halt mode.

[10]: Processor halt (HL). Returns ‘1’ on read when processor is halted. If the processor is in debug mode, setting this bit will put the processor in halt mode.

[11]: Power down (PW). Returns ‘1’ when processor in in power-down mode.

25.6.2 DSU Break and Single Step register

This register is used to break or single step the processor(s). This register controls all processors in a multi-processor system, and is only accessible in the DSU memory map of processor 0.

31		18	17	16	15		2	1	0
SS15	. . .	SS2	SS1	SS0	BN15	. . .	BN2	BN1	BN0

Figure 72. DSU Break and Single Step register

[15:0] : Break now (BNx) -Force processor x into debug mode if the Break on watchpoint (BW) bit in the processors DSU control register is set. If cleared, the processor x will resume execution.

[31:16] : Single step (SSx) - if set, the processor x will execute one instruction and return to debug mode. The bit remains set after the processor goes into the debug mode.

25.6.3 DSU Debug Mode Mask Register

When one of the processors in a multiprocessor LEON3 system enters the debug mode the value of the DSU Debug Mode Mask register determines if the other processors are forced in the debug mode. This register controls all processors in a multi-processor system, and is only accessible in the DSU memory map of processor 0.

31		18	17	16	15		2	1	0
DM15	. . .	DM2	DM1	DM0	ED15	. . .	ED2	ED1	ED0

Figure 73. DSU Debug Mode Mask register

[15:0] : Enter debug mode (EDx) - Force processor x into debug mode if any of processors in a multiprocessor system enters the debug mode. If 0, the processor x will not enter the debug mode.

[31:16]: Debug mode mask. If set, the corresponding processor will not be able to force running processors into debug mode even if it enters debug mode.

25.6.4 DSU trap register

The DSU trap register is a read-only register that indicates which SPARC trap type that caused the processor to enter debug mode. When debug mode is force by setting the BN bit in the DSU control register, the trap type will be 0xb (hardware watchpoint trap).

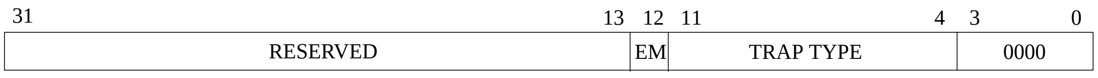


Figure 74. DSU trap register

[11:4]: 8-bit SPARC trap type

[12]: Error mode (EM). Set if the trap would have cause the processor to enter error mode.

25.6.5 Trace buffer time tag counter

The trace buffer time tag counter is incremented each clock as long as the processor is running. The counter is stopped when the processor enters debug mode, and restarted when execution is resumed.

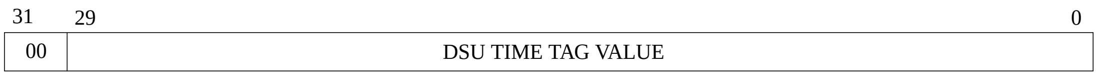


Figure 75. Trace buffer time tag counter

The value is used as time tag in the instruction and AHB trace buffer.

The width of the timer (up to 30 bits) is configurable through the DSU generic port.

25.6.6 DSU ASI register

The DSU can perform diagnostic accesses to different ASI areas. The value in the ASI diagnostic access register is used as ASI while the address is supplied from the DSU.

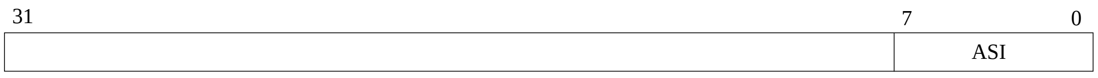


Figure 76. ASI diagnostic access register

[7:0]: ASI to be used on diagnostic ASI access

25.6.7 AHB Trace buffer control register

The AHB trace buffer is controlled by the AHB trace buffer control register:

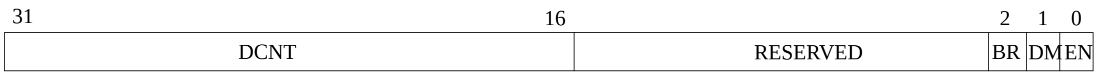


Figure 77. AHB trace buffer control register

[0]: Trace enable (EN). Enables the trace buffer.

[1]: Delay counter mode (DM). Indicates that the trace buffer is in delay counter mode.

[2]: Break (BR). If set, the processor will be put in debug mode when AHB trace buffer stops due to AHB breakpoint hit.

[31:16] Trace buffer delay counter (DCNT). Note that the number of bits actually implemented depends on the size of the trace buffer.

25.6.8 AHB trace buffer index register

The AHB trace buffer index register contains the address of the next trace line to be written.

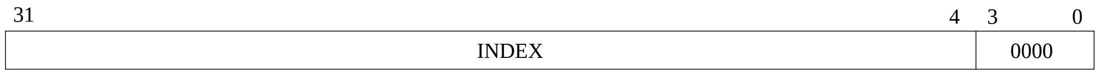


Figure 78. AHB trace buffer index register

31:4 Trace buffer index counter (INDEX). Note that the number of bits actually implemented depends on the size of the trace buffer.

25.6.9 AHB trace buffer breakpoint registers

The DSU contains two breakpoint registers for matching AHB addresses. A breakpoint hit is used to freeze the trace buffer by automatically clearing the enable bit. Freezing can be delayed by programming the DCNT field in the trace buffer control register to a non-zero value. In this case, the DCNT value will be decremented for each additional trace until it reaches zero, after which the trace buffer is frozen. A mask register is associated with each breakpoint, allowing breaking on a block of addresses. Only address bits with the corresponding mask bit set to ‘1’ are compared during breakpoint detection. To break on AHB load or store accesses, the LD and/or ST bits should be set.

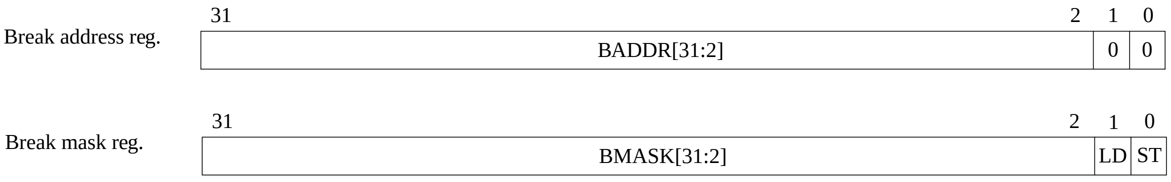


Figure 79. Trace buffer breakpoint registers

[31:2]:	Breakpoint address (bits 31:2)
[31:2]:	Breakpoint mask (see text)
[1]:	LD - break on data load address
[0]:	ST - beak on data store address

25.6.10 Instruction trace control register

The instruction trace control register contains a pointer that indicates the next line of the instruction trace buffer to be written.

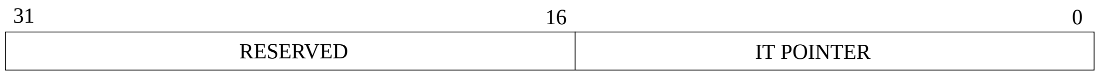


Figure 80. Instruction trace control register

[15:0] Instruction trace pointer. Note that the number of bits actually implemented depends on the size of the trace buffer.

25.7 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x017. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

25.8 Technology mapping

DSU3 has one technology mapping generic, *tech*. This generic controls the implementation of which technology that will be used to implement the trace buffer memories. The AHB trace buffer will use two identical SYNCRAM64 blocks to implement the buffer memory (SYNCRAM64 may then result in two 32-bit wide memories on the target technology). The depth will depend on the KBYTES generic, which indicates the total size of trace buffer in Kbytes. If KBYTES = 1 (1 Kbyte), then two RAM blocks of 64x64 will be used. If KBYTES = 2, then the RAM blocks will be 128x64 and so on.

25.9 Configuration options

Table 228 shows the configuration options of the core (VHDL generics).

*Table 228.*Configuration options

Generic	**Function**	**Allowed range**	**Default**
hindex	AHB slave index	0 - AHBSLVMAX-1	0
haddr	AHB slave address (AHB[31:20])	0 - 16#FFF#	16#900#
hmask	AHB slave address mask	0 - 16#FFF#	16#F00#
ncpu	Number of attached processors	1 - 16	1
tbits	Number of bits in the time tag counter	2 - 30	30
tech	Memory technology for trace buffer RAM	0 - TECHMAX-1	0 (inferred)
kbytes	Size of trace buffer memory in Kbytes. A value of 0 will disable the trace buffer function.	0 - 64	0 (disabled)

25.10 Signal descriptions

Table 229 shows the interface signals of the core (VHDL ports).

*Table 229.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB master input signals	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
DBGI	-	Input	Debug signals from LEON3	-
DBGO	-	Output	Debug signals to LEON3	-
DSUI	ENABLE	Input	DSU enable	High
	BREAK	Input	DSU break	High
DSUO	ACTIVE	Output	Debug mode	High
	PWD[n-1 : 0]	Output	Clock gating enable for processor [n]	High

* see GRLIB IP Library User’s Manual

25.11 Library dependencies

Table 230 shows libraries used when instantiating the core (VHDL libraries).

*Table 230.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	LEON3	Component, signals	Component declaration, signals declaration

25.12 Component declaration

The core has the following component declaration.

```
component dsu3
  generic (
    hindex : integer := 0;
    haddr : integer := 16#900#;
    hmask : integer := 16#f00#;
    ncpu    : integer := 1;
    tbits   : integer := 30;
    tech    : integer := 0;
    irq     : integer := 0;
    kbytes  : integer := 0
  );
  port (
    rst    : in  std_ulogic;
    clk    : in  std_ulogic;
    ahbmi  : in  ahb_mst_in_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : out ahb_slv_out_type;
    dbgi   : in l3_debug_out_vector(0 to NCPU-1);
    dbgo   : out l3_debug_in_vector(0 to NCPU-1);
    dsui   : in dsu_in_type;
    dsuo   : out dsu_out_type
  );
  end component;
```

25.13 Instantiation

This example shows how the core can be instantiated.

The DSU is always instantiated with at least one LEON3 processor. It is suitable to use a generate loop for the instantiation of the processors and DSU and showed below.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon3.all;

constant NCPU : integer := 1; -- select number of processors

signal leon3i : l3_in_vector(0 to NCPU-1);
signal leon3o : l3_out_vector(0 to NCPU-1);
signal irqi   : irq_in_vector(0 to NCPU-1);
signal irqo   : irq_out_vector(0 to NCPU-1);

signal dbgi : l3_debug_in_vector(0 to NCPU-1);
signal dbgo : l3_debug_out_vector(0 to NCPU-1);

signal dsui   : dsu_in_type;
signal dsuo   : dsu_out_type;

.
begin

cpu : for i in 0 to NCPU-1 generate
    u0 : leon3s-- LEON3 processor
    generic map (ahbndx => i, fabtech => FABTECH, memtech => MEMTECH)
    port map (clkm, rstn, ahbmi, ahbmo(i), ahbsi, ahbsi, ahbso,
 irqi(i), irqo(i), dbgi(i), dbgo(i));
    irqi(i) <= leon3o(i).irq; leon3i(i).irq <= irqo(i);
end generate;

dsu0 : dsu3-- LEON3 Debug Support Unit
    generic map (ahbndx => 2, ncpu => NCPU, tech => memtech, kbytes => 2)
    port map (rstn, clkm, ahbmi, ahbsi, ahbso(2), dbgo, dbgi, dsui, dsuo);
dsui.enable <= dsuen; dsui.break <= dsubre; dsuact <= dsuo.active;
```

26 DSU4 - LEON4 Hardware Debug Support Unit

26.1 Overview

To simplify debugging on target hardware, the LEON4 processor implements a debug mode during which the pipeline is idle and the processor is controlled through a special debug interface. The LEON4 Debug Support Unit (DSU) is used to control the processor during debug mode. The DSU acts as an AHB slave and can be accessed by any AHB master. An external debug host can therefore access the DSU through several different interfaces. Such an interface can be a serial UART (RS232), JTAG, PCI, USB or ethernet. The DSU supports multi-processor systems and can handle up to 16 processors.

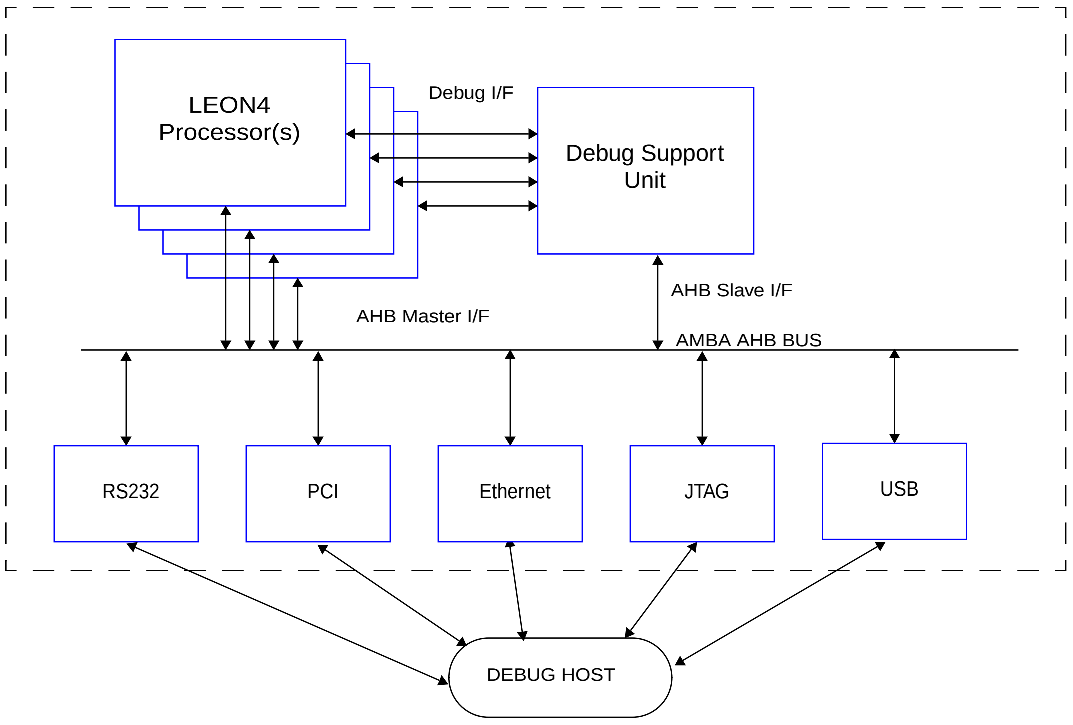


Figure 81. LEON4/DSU Connection

26.2 Operation

Through the DSU AHB slave interface, any AHB master can access the processor registers and the contents of the instruction trace buffer. The DSU control registers can be accessed at any time, while the processor registers, caches and trace buffer can only be accessed when the processor has entered debug mode. In debug mode, the processor pipeline is held and the processor state can be accessed by the DSU. Entering the debug mode can occur on the following events:

- executing a breakpoint instruction (ta 1)
- integer unit hardware breakpoint/watchpoint hit (trap 0xb)
- rising edge of the external break signal (DSUBRE)
- setting the break-now (BN) bit in the DSU control register
- a trap that would cause the processor to enter error mode
- occurrence of any, or a selection of traps as defined in the DSU control register
- after a single-step operation
- one of the processors in a multiprocessor system has entered the debug mode
- DSU AHB breakpoint or watchpoint hit

The debug mode can only be entered when the debug support unit is enabled through an external signal (DSUEN). For DSU break, and the break-now BN bit, to have effect the Break-on-IU-watchpoint (BW) bit must be set in the DSU control register. This bit is set when DSUBRE is active after reset and should also be set by debug monitor software (like Aerofle Gaisler's GRMON) when initializing the DSU. When the debug mode is entered, the following actions are taken:

- PC and nPC are saved in temporary registers (accessible by the debug unit)
- an output signal (DSUACT) is asserted to indicate the debug state
- the timer unit is (optionally) stopped to freeze the LEON timers and watchdog

The instruction that caused the processor to enter debug mode is not executed, and the processor state is kept unmodified. Execution is resumed by clearing the BN bit in the DSU control register or by de-asserting DSUEN. The timer unit will be re-enabled and execution will continue from the saved PC and nPC. Debug mode can also be entered after the processor has entered error mode, for instance when an application has terminated and halted the processor. The error mode can be reset and the processor restarted at any address.

When a processor is in the debug mode, an access to ASI diagnostic area is forwarded to the IU which performs access with ASI equal to value in the DSU ASI register and address consisting of 20 LSB bits of the original address.

26.3 AHB Trace Buffer

The AHB trace buffer consists of a circular buffer that stores AHB data transfers. The address, data and various control signals of the AHB bus are stored and can be read out for later analysis. The trace buffer is 128, 160 or 224 bits wide, depending on the AHB bus width. The information stored is indicated in the table below:

*Table 231.*AHB Trace buffer data allocation

Bits	Name	Definition
223:160	Load/Store data	AHB HRDATA/HWDATA(127:64)
159:129	Load/Store data	AHB HRDATA/HWDATA(63:32)
127	AHB breakpoint hit	Set to '1' if a DSU AHB breakpoint hit occurred.
126	-	Not used
125:96	Time tag	DSU time tag counter
95	-	Not used
94:80	Hirq	AHB HIRQ[15:1]
79	Hwrite	AHB HWRITE
78:77	Htrans	AHB HTRANS
76:74	Hsize	AHB HSIZE
73:71	Hburst	AHB HBURST
70:67	Hmaster	AHB HMASTER
66	Hmastlock	AHB HMASTLOCK
65:64	Hresp	AHB HRESP
63:32	Load/Store data	AHB HRDATA/HWDATA(31:0)
31:0	Load/Store address	AHB HADDR

In addition to the AHB signals, the DSU time tag counter is also stored in the trace.

The trace buffer is enabled by setting the enable bit (EN) in the trace control register. Each AHB transfer is then stored in the buffer in a circular manner. The address to which the next transfer is written is held in the trace buffer index register, and is automatically incremented after each transfer. Tracing is stopped when the EN bit is reset, or when a AHB breakpoint is hit. Tracing is temporarily suspended when the processor enters debug mode, unless the trace force bit (TF) in the trace control register is set. If the trace force bit is set, the trace buffer is activated as long as the enable bit is set. The force bit is reset if an AHB breakpoint is hit and can also be cleared by software. Note that neither the trace buffer memory nor the breakpoint registers (see below) can be read/written by software when the trace buffer is enabled.

The DSU has an internal time tag counter and this counter is frozen when the processor enters debug mode. When AHB tracing is performed in debug mode (using the trace force bit) it may be desirable to also enable the time tag counter. This can be done using the timer enable bit (TE). Note that the time tag is also used for the instruction trace buffer and the timer enable bit should only be set when using the DSU as an AHB trace buffer only, and not when performing profiling or software debugging. The timer enable bit is reset on the same events as the trace force bit.

26.3.1 AHB trace buffer filters

The DSU can be implemented with filters that can be applied to the AHB trace buffer, breakpoints and watchpoints. If implemented, these filters are controlled via the AHB trace buffer filter control and AHB trace buffer filter mask registers. The fields in these registers allows masking access characteristics such as master, slave, read, write and address range so that accesses that correspond to the specified mask are not written into the trace buffer. Address range masking is done using the second AHB breakpoint register set. The values of the LD and ST fields of this register has no effect on filtering.

26.3.2 AHB statistics

The DSU can be implemented to generate statistics from the traced AHB bus. When statistics collection is enabled the DSU will assert outputs that are suitable to connect to a LEON4 statistics unit (L4STAT). The statistical outputs can be filtered by the AHB trace buffer filters, this is controlled by the Performance counter Filter bit (PF) in the AHB trace buffer filter control register. The DSU can collect data for the events listed in table 232 below.

*Table 232.*AHB events

Event	Description	Note
idle	HTRANS=IDLE	Active when HTRANS IDLE is driven on the AHB slave inputs and slave has asserted HREADY.
busy	HTRANS=BUSY	Active when HTRANS BUSY is driven on the AHB slave inputs and slave has asserted HREADY.
nseq	HTRANS=NONSEQ	Active when HTRANS NONSEQ is driven on the AHB slave inputs and slave has asserted HREADY.
seq	HTRANS=SEQ	Active when HTRANS SEQUENTIAL is driven on the AHB slave inputs and slave has asserted HREADY.
read	Read access	Active when HTRANS is SEQUENTIAL or NON-SEQUENTIAL, slave has asserted HREADY and the HWRITE input is low.
write	Write access	Active when HTRANS is SEQUENTIAL or NON-SEQUENTIAL, slave has asserted HREADY and the HWRITE input is high.
hsize[5:0]	Transfer size	Active when HTRANS is SEQUENTIAL or NON-SEQUENTIAL, slave has asserted HREADY and HSIZE is BYTE (hsize[0]), HWORD (HSIZE[1]), WORD (hsize[2]), DWORD (hsize[3]), 4WORD hsize[4], or 8WORD (hsize[5]).
ws	Wait state	Active when HREADY input to AHB slaves is low and AMBA response is OKAY.
retry	RETRY response	Active when master receives RETRY response
split	SPLIT response	Active when master receives SPLIT response

*Table 232.*AHB events

Event	Description	Note
spdel	SPLIT delay	Active during the time a master waits to be granted access to the bus after reception of a SPLIT response. The core will only keep track of one master at a time. This means that when a SPLIT response is detected, the core will save the master index. This event will then be active until the same master is re-allowed into bus arbitration and is granted access to the bus. This also means that the delay measured will include the time for re-arbitration, delays from other ongoing transfers and delays resulting from other masters being granted access to the bus before the SPLIT:ed master is granted again after receiving SPLIT complete. If another master receives a SPLIT response while this event is active, the SPLIT delay for the second master will not be measured.
locked	Locked access	Active while the HMASTLOCK signal is asserted on the AHB slave inputs.

26.4 Instruction trace buffer

The instruction trace buffer consists of a circular buffer that stores executed instructions. The instruction trace buffer is located in the processor, and read out via the DSU. The trace buffer is 128 bits wide, the information stored is indicated in the table below:

*Table 233.*Instruction trace buffer data allocation

Bits	Name	Definition
126	Multi-cycle instruction	Set to ‘1’ on the second instance of a multi-cycle instruction
125:96	Time tag	The value of the DSU time tag counter
95:64	Result or Store address/data	Instruction result, Store address or Store data
63:34	Program counter	Program counter (2 lsb bits removed since they are always zero)
33	Instruction trap	Set to ‘1’ if traced instruction trapped
32	Processor error mode	Set to ‘1’ if the traced instruction caused processor error mode
31:0	Opcode	Instruction opcode

During tracing, one instruction is stored per line in the trace buffer with the exception of atomic load/ store instructions, which are entered twice (one for the load and one for the store operation). Bits [63:32] in the buffer correspond to the store address and the loaded data for load instructions. Bit 126 is set for the second entry.

When the processor enters debug mode, tracing is suspended. The trace buffer and the trace buffer control register can be read and written while the processor is in the debug mode. During the instruction tracing (processor in normal mode) the trace buffer and the trace buffer control register can not be accessed. The traced instructions can optionally be filtered on instruction types. Which instructions are traced is defined in the instruction trace register [31:28], as defined in the table below:

*Table 234.*Trace filter operation

Trace filter	Instructions traced
0x0	All instructions
0x1	SPARC Format 2 instructions
0x2	Control-flow changes. All Call, branch and trap instructions including branch targets
0x4	SPARC Format 1 instructions (CALL)
0x8	SPARC Format 3 instructions except LOAD or STORE
0xC	SPARC Format 3 LOAD or STORE instructions

26.5 DSU memory map

The DSU memory map can be seen in table 235 below. In a multiprocessor systems, the register map is duplicated and address bits 27 - 24 are used to index the processor.

*Table 235.*DSU memory map

Address offset	**Register**
0x000000	DSU control register
0x000008	Time tag counter
0x000020	Break and Single Step register
0x000024	Debug Mode Mask register
0x000040	AHB trace buffer control register
0x000044	AHB trace buffer index register
0x000048	AHB trace buffer filter control register
0x00004c	AHB trace buffer filter mask register
0x000050	AHB breakpoint address 1
0x000054	AHB mask register 1
0x000058	AHB breakpoint address 2
0x00005c	AHB mask register 2
0x000070	Instruction count register
0x000080	AHB watchpoint control register
0x000090 - 0x00009C	AHB watchpoint 1 data registers
0x0000A0 - 0x0000AC	AHB watchpoint 1 mask registers
0x0000B0 - 0x0000BC	AHB watchpoint 2 data registers
0x0000C0 - 0x0000CC	AHB watchpoint 2 mask registers
0x100000 - 0x10FFFF	Instruction trace buffer (..0: Trace bits 127 - 96, ..4: Trace bits 95 - 64, ..8: Trace bits 63 - 32, ..C : Trace bits 31 - 0)
0x110000	Instruction Trace buffer control register
0x200000 - 0x210000	AHB trace buffer (..0: Trace bits 127 - 96, ..4: Trace bits 95 - 64, ..8: Trace bits 63 - 32, ..C : Trace bits 31 - 0)
0x300000 - 0x3007FC	IU register file. The addresses of the IU registers depends on how many register windows has been implemented: %o*n*: 0x300000 + (((psr.cwp * 64) + 32 + *n**4) mod (NWINDOWS*64)) %l*n*: 0x300000 + (((psr.cwp * 64) + 64 + *n**4) mod (NWINDOWS*64)) %i*n*: 0x300000 + (((psr.cwp * 64) + 96 + *n**4) mod (NWINDOWS*64)) %g*n*: 0x300000 + (NWINDOWS*64) + n*4 %f*n*: 0x301000 + *n**4
0x300800 - 0x300FFC	IU register file check bits (LEON4FT only)
0x301000 - 0x30107C	FPU register file
0x400000	Y register
0x400004	PSR register
0x400008	WIM register
0x40000C	TBR register
0x400010	PC register
0x400014	NPC register
0x400018	FSR register
0x40001C	CPSR register
0x400020	DSU trap register
0x400024	DSU ASI register

*Table 235.*DSU memory map

Address offset	**Register**
0x400040 - 0x40007C	ASR16 - ASR31 (when implemented)

0x700000 - 0x7FFFFC	ASI diagnostic access (ASI = value in DSU ASI register, address = address[19:0])
	ASI = 0x9 : Local instruction RAM
	ASI = 0xB : Local data RAM
	ASI = 0xC : Instruction cache tags
	ASI = 0xD : Instruction cache data
	ASI = 0xE : Data cache tags
	ASI = 0xF : Data cache data
	ASI = 0x1E : Separate snoop tags

26.6 DSU registers

26.6.1 DSU control register

The DSU is controlled by the DSU control register:

Table 236. DSU control register

31	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED		PW	HL	PE	EB	EE	DM	BZ	BX	BS	BW	BE	TE

31: 12	Reserved
11	Power down (PW) - Returns ‘1’ when processor is in power-down mode.
10	Processor halt (HL) - Returns ‘1’ on read when processor is halted. If the processor is in debug mode, setting this bit will put the processor in halt mode.
9	Processor error mode (PE) - returns ‘1’ on read when processor is in error mode, else ‘0’. If written with ‘1’, it will clear the error and halt mode.
8	External Break (EB) - Value of the external DSUBRE signal (read-only)
7	External Enable (EE) - Value of the external DSUEN signal (read-only)
6	Debug mode (DM) - Indicates when the processor has entered debug mode (read-only).
5	Break on error traps (BZ) - if set, will force the processor into debug mode on all *except* the following traps: priviledged_instruction, fpu_disabled, window_overflow, window_underflow, asynchronous_interrupt, ticc_trap.
4	Break on trap (BX) - if set, will force the processor into debug mode when any trap occurs.
3	Break on S/W breakpoint (BS) - if set, debug mode will be forced when an breakpoint instruction (ta 1) is executed.
2	Break on IU watchpoint (BW) - if set, debug mode will be forced on a IU watchpoint (trap 0xb).
1	Break on error (BE) - if set, will force the processor to debug mode when the processor would have entered error condition (trap in trap).
0	Trace enable (TE) - Enables instruction tracing. If set the instructions will be stored in the trace buffer. Remains set when then processor enters debug or error mode

26.6.2 DSU Break and Single Step register

This register is used to break or single step the processor(s). This register controls all processors in a multi-processor system, and is only accessible in the DSU memory map of processor 0.

Table 237. DSU Break and Single Step register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SS[15:0]																BN[15:0]															

31: 16	Single step (SSx) - if set, the processor x will execute one instruction and return to debug mode. The bit remains set after the processor goes into the debug mode.
15: 0	Break now (BNx) -Force processor x into debug mode if the Break on watchpoint (BW) bit in the processors DSU control register is set. If cleared, the processor x will resume execution.

26.6.3 DSU Debug Mode Mask Register

When one of the processors in a multiprocessor LEON4 system enters the debug mode the value of the DSU Debug Mode Mask register determines if the other processors are forced in the debug mode. This register controls all processors in a multi-processor system, and is only accessible in the DSU memory map of processor 0.

Table 238. DSU Debug Mode Mask register

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DM[15:0]	ED[15:0]

31: 16	Debug mode mask (DMx) - If set, the corresponding processor will not be able to force running processors into debug mode even if it enters debug mode.
15: 0	Enter debug mode (EDx) - Force processor x into debug mode if any of processors in a multiprocessor system enters the debug mode. If 0, the processor x will not enter the debug mode.

26.6.4 DSU trap register

The DSU trap register is a read-only register that indicates which SPARC trap type that caused the processor to enter debug mode. When debug mode is force by setting the BN bit in the DSU control register, the trap type will be 0xb (hardware watchpoint trap).

Table 239. DSU Trap register

31 13	12	11 4	3 0
RESERVED	EM	TRAPTYPE	0000

31: 13	RESERVED
12	Error mode (EM) - Set if the trap would have cause the processor to enter error mode.
11: 4	Trap type (TRAPTYPE) - 8-bit SPARC trap type
3: 0	Read as 0x0

26.6.5 Trace buffer time tag counter

The trace buffer time tag counter is incremented each clock as long as the processor is running. The counter is stopped when the processor enters debug mode (unless the timer enable bit in the AHB trace buffer control register is set), and restarted when execution is resumed.

Table 240. Trace buffer time tag counter

31 30	29 0
0b00	TIMETAG

31: 30	Read as 0b00
29: 0	DSU Time Tag Value (TIMETAG)

The value is used as time tag in the instruction and AHB trace buffer.

The width of the timer (up to 30 bits) is configurable through the DSU generic port.

26.6.6 DSU ASI register

The DSU can perform diagnostic accesses to different ASI areas. The value in the ASI diagnostic access register is used as ASI while the address is supplied from the DSU.

Table 241. ASI diagnostic access register

31 8	7 0
RESERVED	ASI

Table 241. ASI diagnostic access register

31: 8	RESERVED
7: 0	ASI (ASI) - ASI to be used on diagnostic ASI access

26.6.7 AHB Trace buffer control register

The AHB trace buffer is controlled by the AHB trace buffer control register:

Table 242. AHB trace buffer control register

31 16	15 8	7	6	5	4 3	2	1	0
DCNT	RESERVED	SF	TE	TF	BW	BR	DM	EN

31: 16	Trace buffer delay counter (DCNT) - Note that the number of bits actually implemented depends on the size of the trace buffer.
15: 8	RESERVED
7	Sample Force (SF) - If this bit is written to ‘1’ it will have the same effect on the AHB trace buffer as if HREADY was asserted on the bus at the same time as a sequential or non-sequential transfer is made. This means that setting this bit to ‘1’ will cause the values in the trace buffer’s sample registers to be written into the trace buffer, and new values will be sampled into the registers. This bit will automatically be cleared after one clock cycle. Writing to the trace buffer still requires that the trace buffer is enabled (EN bit set to ‘1’) and that the CPU is not in debug mode or that tracing is forced (TF bit set to ‘1’). This functionality is primarily of interest when the trace buffer is tracing a separate bus and the traced bus appears to have frozen.
6	Timer enable (TE) - Activates time tag counter also in debug mode.
5	Trace force (TF) - Activates trace buffer also in debug mode. Note that the trace buffer must be disabled when reading out trace buffer data via the core’s register interface.
4: 3	Bus width (BW) - This value corresponds to log2(Supported bus width / 32)
2	Break (BR) - If set, the processor will be put in debug mode when AHB trace buffer stops due to AHB breakpoint hit.
1	Delay counter mode (DM) - Indicates that the trace buffer is in delay counter mode.
0	Trace enable (EN) - Enables the trace buffer.

26.6.8 AHB trace buffer index register

The AHB trace buffer index register contains the address of the next trace line to be written.

Table 243. AHB trace buffer index register

31 4	3 2 1 0
INDEX	0x0

31: 4	Trace buffer index counter (INDEX) - Note that the number of bits actually implemented depends on the size of the trace buffer.
3: 0	Read as 0x0

26.6.9 AHB trace buffer filter control register

The trace buffer filter control register is only available if the core has been implemented with support for AHB trace buffer filtering.

Table 244. AHB trace buffer filter control register

31 14	13 12	11 10	9 8	7 4	3	2	1	0
RESERVED	WPF	R	BPF	RESERVED	PF	AF	FR	FW

31: 14	RESERVED

Table 244. AHB trace buffer filter control register

13: 12	AHB watchpoint filtering (WPF) - Bit 13 of this field applies to AHB watchpoint 2 and bit 12 applies to AHB watchpoint 1. If the WPF bit for a watchpoint is set to '1' then the watchpoint will not trigger unless the access also passes through the filter. This functionality can be used to, for instance, set a AHB watchpoint that only triggers if a specified master performs an access to a specified slave.
11: 10	RESERVED
9: 8	AHB breakpoint filtering (BPF) - Bit 9 of this field applies to AHB breakpoint 2 and bit 8 applies to AHB breakpoint 1. If the BPF bit for a breakpoint is set to '1' then the breakpoint will not trigger unless the access also passes through the filter. This functionality can be used to, for instance, set a AHB breakpoint that only triggers if a specified master performs an access to a specified slave. Note that if a AHB breakpoint is coupled with an AHB watchpoint then the setting of the corresponding bit in this field has no effect.
7: 4	RESERVED
3	Performance counter Filter (PF) - If this bit is set to '1', the cores performance counter (statistical) outputs will be filtered using the same filter settings as used for the trace buffer. If a filter inhibits a write to the trace buffer, setting this bit to '1' will cause the same filter setting to inhibit the pulse on the statistical output.
2	Address Filter (AF) - If this bit is set to '1', only the address range defined by AHB trace buffer breakpoint 2's address and mask will be included in the trace buffer.
1	Filter Reads (FR) - If this bit is set to '1', read accesses will not be included in the trace buffer.
0	Filter Writes (FW) - If this bit is set to '1', write accesses will not be included in the trace buffer.

26.6.10 AHB trace buffer filter mask register

The trace buffer filter mask register is only available if the core has been implemented with support for AHB trace buffer filtering.

Table 245. AHB trace buffer filter mask register

31 16	15 0
SMASK[15:0]	MMASK[15:0]

31: 16	Slave Mask (SMASK) - If SMASK[n] is set to '1', the trace buffer will not save accesses performed to slave n.
15: 0	Master Mask (MMASK) - If MMASK[n] is set to '1', the trace buffer will not save accesses performed by master n.

26.6.11 AHB trace buffer breakpoint registers

The DSU contains two breakpoint registers for matching AHB addresses. A breakpoint hit is used to freeze the trace buffer by automatically clearing the enable bit. Freezing can be delayed by programming the DCNT field in the trace buffer control register to a non-zero value. In this case, the DCNT value will be decremented for each additional trace until it reaches zero, after which the trace buffer is frozen. A mask register is associated with each breakpoint, allowing breaking on a block of addresses. Only address bits with the corresponding mask bit set to '1' are compared during breakpoint detection. To break on AHB load or store accesses, the LD and/or ST bits should be set.

Table 246. AHB trace buffer break address register

31 2	1 0
BADDR[31:2]	0b00

31: 2	Break point address (BADDR) - Bits 31:2 of breakpoint address
1: 0	Read as 0b00

Table 247. AHB trace buffer break mask register

31 2	1	0
BMASK[31:2]	LD	ST

Table 247. AHB trace buffer break mask register

31: 2	Breakpoint mask (BMASK) - (see text)
1	Load (LD) - Break on data load address
0	Store (ST) - Break on data store address

26.6.12 Instruction trace control register

The instruction trace control register contains a pointer that indicates the next line of the instruction trace buffer to be written.

Table 248. Instruction trace control register

31	16	15 0
ITRACE CFG	RESERVED	ITPOINTER

31: 28	Trace filter configuration
27: 16	RESERVED
15: 0	Instruction trace pointer (ITPOINTER) - Note that the number of bits actually implemented depends on the size of the trace buffer

26.6.13 Instruction count register

The DSU contains an instruction count register to allow profiling of application, or generation of debug mode after a certain clocks or instructions. The instruction count register consists of a 29-bit down-counter, which is decremented on either each clock (IC=0) or on each executed instruction (IC=1). In profiling mode (PE=1), the counter will set to all ones after an underflow without generating a processor break. In this mode, the counter can be periodically polled and statistics can be formed on CPI (clocks per instructions). In non-profiling mode (PE=0), the processor will be put in debug mode when the counter underflows. This allows a debug tool such as GRMON to execute a defined number of instructions, or for a defined number of clocks.

Table 249. Instruction count register

31	30	29	28 0
CE	IC	PE	ICOUNT[28:0]

31	Counter Enable (CE) - Counter enable
30	Instruction Count (IC) - Instruction (1) or clock (0) counting
29	Profiling Enable (PE) - Profiling enable
28: 0	Instruction count (ICOUNT) - Instruction count

26.6.14 AHB watchpoint control register

The DSU has two AHB watchpoints that can be used to freeze the AHB tracebuffer, or put the processor in debug mode, when a specified data pattern occurs on the AMBA bus. These watchpoints can also be coupled with the two AHB breakpoints so that a watchpoint will not trigger unless the AHB breakpoint is triggered. This also means that when a watchpoint is coupled with an AHB breakpoint, the breakpoint will not cause an AHB tracebuffer freeze, or put the processor(s), in debug mode unless also the watchpoint is triggered.

Table 250. AHB watchpoint control register

31 7	6	5	4	3	2	1	0
RESERVED	IN	CP	EN	R	IN	CP	EN

31: 7	RESERVED

Table 250. AHB watchpoint control register

6	Invert (IN) - Invert AHB watchpoint 2. If this bit is set the watchpoint will trigger if data on the AHB bus does NOT match the specified data pattern (typically only usable if the watchpoint has been coupled with an address by setting the CP field).
5	Couple (CP) - Couple AHB watchpoint 2 with AHB breakpoint 1
4	Enable (EN) - Enable AHB watchpoint 2
3	RESERVED
2	Invert (IN) - Invert AHB watchpoint 1. If this bit is set the watchpoint will trigger if data on the AHB bus does NOT match the specified data pattern (typically only usable if the watchpoint has been coupled with an address by setting the CP field).
1	Couple (CP) - Couple AHB watchpoint 1 with AHB breakpoint 1
0	Enable (EN) - Enable AHB watchpoint 1

26.6.15 AHB watchpoint data and mask registers

The AHB watchpoint data and mask registers specify the data pattern for an AHB watchpoint. A watchpoint hit is used to freeze the trace buffer by automatically clearing the enable bit. A watchpoint hit can also be used to force the processor(s) to debug mode.

A mask register is associated with each data register. Only data bits with the corresponding mask bit set to ‘1’ are compared during watchpoint detection.

Table 251. AHB watchpoint data register

31 0
DATA[127-n*32 : 96-n*32]

31: 0	AHB watchpoint data (DATA) - Specifies the data pattern of one word for an AHB watchpoint. The lower part of the register address specifies with part of the bus that the register value will be compared against: Offset 0x0 specifies the data value for AHB bus bits 127:96, 0x4 for bits 95:64, 0x8 for 63:32 and offset 0xC for bits 31:0.

Table 252. AHB watchpoint mask register

31 0
MASK[127-n*32 : 96-n*32]

31: 0	AHB watchpoint mask (MASK) - Specifies the mask to select bits for comparison out of one word for an AHB watchpoint. The lower part of the register address specifies with part of the bus that the register value will be compared against: Offset 0x0 specifies the data value for AHB bus bits 127:96, 0x4 for bits 95:64, 0x8 for 63:32 and offset 0xC for bits 31:0.

In a system with 64-bit bus width only half of the data and mask registers must be written. For AHB watchpoint 1, a data value with 64-bits would be written to the AHB watchpoint data registers at offsets 0x98 and 0x9C. The corresponding mask bits would be set in mask registers at offsets 0xA8 and 0xAC.

In most GRLIB systems with wide AMBA buses, the data for an access size that is less than the full bus width will be replicated over the full bus. For instance, a 32-bit write access from a LEON processor on a 64-bit bus will place the same data on bus bits 64:32 and 31:0.

26.7 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x017. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

26.8 Technology mapping

DSU4 has one technology mapping generic, *tech*. This generic controls the implementation of which technology that will be used to implement the trace buffer memories. The AHB trace buffer will use two identical SYNCRAM64 blocks to implement the buffer memory (SYNCRAM64 may then result in two 32-bit wide memories on the target technology, depending on the technology map), with one additional 32-bit wide SYNCRAM if the system's AMBA data bus width is 64-bits, and also one additional 64-bit wide SYNCRAM if the system's AMBA data bus width exceeds 64 bits.

The depth of the RAMs depends on the KBYTES generic, which indicates the total size of trace buffer in Kbytes. If KBYTES = 1 (1 Kbyte), then the depth will be 64. If KBYTES = 2, then the RAM depth will be 128 and so on.

26.9 Configuration options

Table 253 shows the configuration options of the core (VHDL generics).

*Table 253.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	0 - AHBSLVMAX-1	0
haddr	AHB slave address (AHB[31:20])	0 - 16#FFF#	16#900#
hmask	AHB slave address mask	0 - 16#FFF#	16#F00#
ncpu	Number of attached processors	1 - 16	1
tbits	Number of bits in the time tag counter	2 - 30	30
tech	Memory technology for trace buffer RAM	0 - TECHMAX-1	0 (inferred)
kbytes	Size of trace buffer memory in KiB. A value of 0 will disable the trace buffer function.	0 - 64	0 (disabled)
bwidth	Traced AHB bus width	32, 64, 128	64
ahbpf	AHB performance counters and filtering. If *ahbpf* is non-zero the core will support AHB trace buffer filtering. If *ahbpf* is larger than 1 then the core's statistical outputs will be enabled.	0 - 2	0
ahbwp	AHB watchpoint enable. If *ahbwp* is non-zero (default) then the core will support AHB watchpoints (also referred to as AHB data breakpoints). Pipeline registers will be added when *ahbwp* is set to 2 (default value), one register for each bit on the AMBA data bus. This setting is recommended in order to improve timing but has a cost in area. The pipeline registers will also lead to the AHB watchpoint being triggered one cycle later. It is recommended to leave this functionality enabled. However, the added logic can create critical timing paths from the AMBA data vectors and so AHB watchpoints can be completely disabled by setting this generic to 0.	0 - 2	2

26.10 Signal descriptions

Table 254 shows the interface signals of the core (VHDL ports).

*Table 254.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB master input signals	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
DBGI	-	Input	Debug signals from LEON4	-
DBGO	-	Output	Debug signals to LEON4	-
DSUI	ENABLE	Input	DSU enable	High
	BREAK	Input	DSU break	High
DSUO	ACTIVE	Output	Debug mode	High
	PWD[n-1 : 0]	Output	Clock gating enable for processor [n]	High
	ASTAT (record)	Output	AHB statistic/performance counter events	-

* see GRLIB IP Library User’s Manual

26.11 Library dependencies

Table 255 shows libraries used when instantiating the core (VHDL libraries).

*Table 255.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	LEON4	Component, signals	Component declaration, signals declaration

26.12 Component declaration

The core has the following component declaration.

```
component dsu4
  generic (
    hindex : integer := 0;
    haddr : integer := 16#900#;
    hmask : integer := 16#f00#;
    ncpu    : integer := 1;
    tbits   : integer := 30;
    tech    : integer := 0;
    irq     : integer := 0;
    kbytes  : integer := 0
  );
  port (
    rst    : in  std_ulogic;
    clk    : in  std_ulogic;
    ahbmi  : in  ahb_mst_in_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : out ahb_slv_out_type;
    dbgi   : in l4_debug_out_vector(0 to NCPU-1);
    dbgo   : out l4_debug_in_vector(0 to NCPU-1);
    dsui   : in dsu4_in_type;
    dsuo   : out dsu4_out_type
  );
  end component;
```

26.13 Instantiation

This example shows how the core can be instantiated.

The DSU is always instantiated with at least one LEON4 processor. It is suitable to use a generate loop for the instantiation of the processors and DSU and showed below.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon4.all;

constant NCPU : integer := 1; -- select number of processors

signal leon4i : l4_in_vector(0 to NCPU-1);
signal leon4o : l4_out_vector(0 to NCPU-1);
signal irqi   : irq_in_vector(0 to NCPU-1);
signal irqo   : irq_out_vector(0 to NCPU-1);

signal dbgi : l4_debug_in_vector(0 to NCPU-1);
signal dbgo : l4_debug_out_vector(0 to NCPU-1);

signal dsui   : dsu4_in_type;
signal dsuo   : dsu4_out_type;

.
begin

cpu : for i in 0 to NCPU-1 generate
    u0 : leon4s                   -- LEON4 processor
    generic map (ahbndx => i, fabtech => FABTECH, memtech => MEMTECH)
    port map (clkm, rstn, ahbmi, ahbmo(i), ahbsi, ahbsi, ahbso,
 irqi(i), irqo(i), dbgi(i), dbgo(i));
    irqi(i) <= leon4o(i).irq; leon4i(i).irq <= irqo(i);
end generate;

dsu0 : dsu4           -- LEON4 Debug Support Unit
    generic map (ahbndx => 2, ncpu => NCPU, tech => memtech, kbytes => 2)
    port map (rstn, clkm, ahbmi, ahbsi, ahbso(2), dbgo, dbgi, dsui, dsuo);
dsui.enable <= dsuen; dsui.break <= dsubre; dsuact <= dsuo.active;
```

27 FTAHBRAM - On-chip SRAM with EDAC and AHB interface

27.1 Overview

The FTAHBRAM core is a version of the AHBRAM core with added Error Detection And Correction (EDAC). The on-chip memory is accessed via an AMBA AHB slave interface. The memory implements 2 kbytes of data (configured via the *kbytes* VHDL generics). Registers are accessed via an AMB APB interface.

The on-chip memory implements volatile memory that is protected by means of Error Detection And Correction (EDAC). One error can be corrected and two errors can be detected, which is performed by using a (32, 7) BCH code. Some of the optional features available are single error counter, diagnostic reads and writes and autoscrubbing (automatic correction of single errors during reads). Configuration is performed via a configuration register.

Figure 82 shows a block diagram of the internals of the memory.

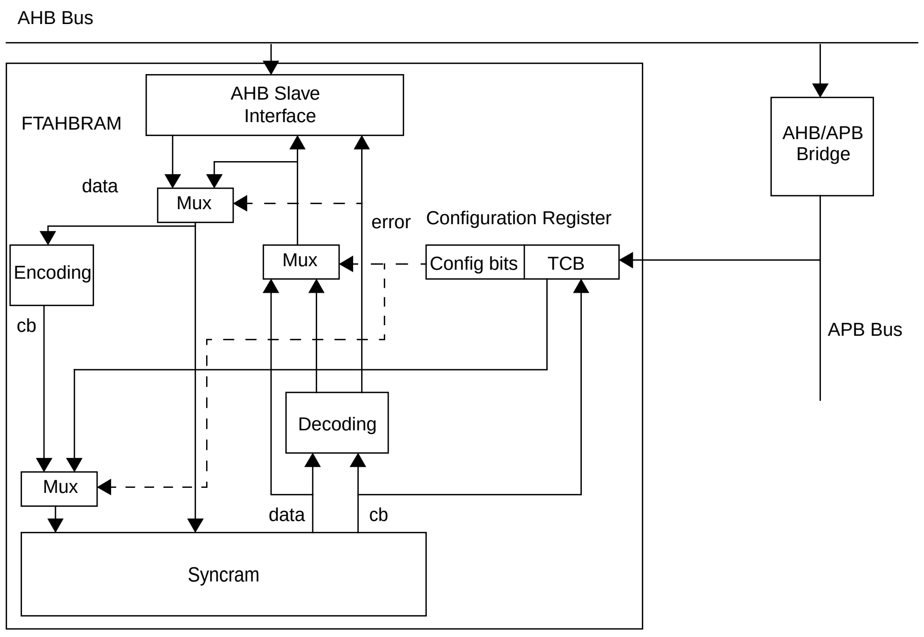


Figure 82. Block diagram

27.2 Operation

The on-chip fault tolerant memory is accessed through an AMBA AHB slave interface.

The memory address range is configurable with VHDL generics. As for the standard AHB RAM, the memory technology and size is configurable through the tech and kbytes VHDL generics. The minimum size is 1 kb and the maximum is technology dependent but the values can only be increased in binary steps.

Run-time configuration is done by writing to a configuration register accessed through an AMBA APB interface.

The address of the interface and the available options are configured with VHDL generics. The EDAC functionality can be completely removed by setting the edacen VHDL generic to zero during synthesis. The APB interface is also removed since it is redundant without EDAC.

The following can be configured during run-time: EDAC can be enabled and disabled. When it is disabled, reads and writes will behave as the standard memory. Read and write diagnostics can be controlled through separate bits. The single error counter can be reset.

If EDAC is disabled (EN bit in configuration register set to 0) write data is passed directly to the memory area and read data will appear on the AHB bus immediately after it arrives from memory. If EDAC is enabled write data is passed to an encoder which outputs a 7-bit checksum. The checksum is stored together with the data in memory and the whole operation is performed without any added waitstates. This applies to word stores (32-bit). If a byte or halfword store is performed, the whole word to which the byte or halfword belongs must first be read from memory (read - modify - write). A new checksum is calculated when the new data is placed in the word and both data and checksum are stored in memory. This is done with 1 - 2 additional waitstates compared to the non EDAC case.

Reads with EDAC disabled are performed with 0 or 1 waitstates while there could also be 2 waitstates when EDAC is enabled. There is no difference between word and subword reads. Table 256 shows a summary of the number of waitstates for the different operations with and without EDAC.

*Table 256.*Summary of the number of waitstates for the different operations for the memory.

Operation	**Waitstates with EDAC Disabled**	**Waitstates with EDAC Enabled**
Read	0 - 1	0 - 2
Word write	0	0
Subword write	0	1 - 2

If the ahbpipe VHDL generic is set to 1, pipeline registers are enabled for the AHB input signals. If the pipeline registers are enabled, one extra waitstate should be added to the read and subword write cases in Table 256.

When EDAC is used, the data is decoded the first cycle after it arrives from the memory and appears on the bus the next cycle if no uncorrectable error is detected. The decoding is done by comparing the stored checksum with a new one which is calculated from the stored data. This decoding is also done during the read phase for a subword write. A so-called syndrome is generated from the comparison between the checksum and it determines the number of errors that occured. One error is automatically corrected and this situation is not visible on the bus. Two or more detected errors cannot be corrected so the operation is aborted and the required two cycle error response is given on the AHB bus (see the AMBA manual for more details). If no errors are detected data is passed through the decoder unaltered.

As mentioned earlier the memory provides read and write diagnostics when EDAC is enabled. When write diagnostics are enabled, the calculated checksum is not stored in memory during the write phase. Instead, the TCB field from the configuration register is used. In the same manner, if read diagnostics are enabled, the stored checksum from memory is stored in the TCB field during a read (and also during a subword write). This way, the EDAC functionality can be tested during run-time. Note that checkbits are stored in TCB during reads and subword writes even if a multiple error is detected.

An additional feature is the single error counter which can be enabled with the *errcnten* VHDL generic. A single error counter (SEC) field is present in the configuration register, and is incremented each time a single databit error is encountered (reads or subword writes). The number of bits of this counter is 8, set with the *cntbits* VHDL generic. It is accessed through the configuration register. Each counter bit can be reset to zero by writing a one to it. The counter saturates at the value 2^{8} - 1 ($2^{cntbits}$ - 1). Each time a single error is detected the aramo.ce signal will be driven high for one cycle. This signal should be connected to an AHB status register which stores information and generates interrupts (see the AHB Status register documentation for more information).

Autoscrubbing is an error handling feature which is enabled with the *autoscrub* VHDL generic and cannot be controlled through the configuration register. If enabled, every single error encountered during a read results in the word being written back with the error corrected and new checkbits generated. It is not visible externally except for that it can generate an extra waitstate. This happens if the read is followed by an odd numbered read in a burst sequence of reads or if a subword write follows. These situations are very rare during normal operation so the total timing impact is negligible. The aramo.ce signal is normally used to generate interrupts which starts an interrupt routine that corrects errors.

Since this is not necessary when autoscrubbing is enabled, aramo.ce should not be connected to an AHB status register or the interrupt should be disabled in the interrupt controller.

27.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 257.*FTAHBRAM registers

APB Address offset	Register
0x0	Configuration Register

Table 258. Configuration Register

31 13+8	12+8 13	12 10	9	8	7	6 0
	SEC	MEMSIZE	WB	RB	EN	TCB

12+8: 13	Single error counter (SEC): Incremented each time a single error is corrected (includes errors on checkbits). Each bit can be set to zero by writing a one to it. This feature is only available if the errcnten VHDL generic is set.
12: 10	Log2 of the current memory size
9	Write Bypass (WB): When set, the TCB field is stored as check bits when a write is performed to the memory.
8	Read Bypass (RB) : When set during a read or subword write, the check bits loaded from memory are stored in the TCB field.
7	EDAC Enable (EN): When set, the EDAC is used otherwise it is bypassed during read and write operations.
6: 0	Test Check Bits (TCB) : Used as checkbits when the WB bit is set during writes and loaded with the check bits during a read operation when the RB bit is set.

Any unused most significant bits are reserved. Always read as ‘000...0’.

All fields except TCB are initialised at reset. The EDAC is initally disabled (EN = 0), which also applies to diagnostics fiels (RB and WB are zero).

When available, the single error counter (SEC) field is cleared to zero.

27.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x050. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

27.5 Configuration options

Table 259 shows the configuration options of the core (VHDL generics).

*Table 259.*Configuration options

Generic	Function	Allowed range	Default
hindex	Selects which AHB select signal (HSEL) will be used to access the memory.	0 to NAHBMAX-1	0
haddr	ADDR field of the AHB BAR	0 to 16#FFF#	0
hmask	MASK field of the AHB BAR	0 to 16#FFF#	16#FFF#
tech	Memory technology	0 to NTECH	0
kbytes	SRAM size in kbytes	1 to targetdep.	1
pindex	Selects which APB select signal (PSEL) will be used to access the memory configuration registers	0 to NAPBMAX-1	0
paddr	The 12-bit MSB APB address	0 to 16#FFF#	0
pmask	The APB address mask	0 to 16#FFF#	16#FFF#
edacen	Enable (1)/Disable (0) on-chip EDAC	0 to 1	0
autoscrub	Automatically store back corrected data with new check-bits during a read when a single error is detected. Is ignored when edacen is deasserted.	0 to 1	0
errcnten	Enables a single error counter	0 to 1	0
cntbits	number of bits in the single error counter	1 to 8	1
ahbpipe	Enable pipeline register on AHB input signals	0 to 1	0

27.6 Signal descriptions

Table 260 shows the interface signals of the core (VHDL ports).

*Table 260.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
ARAMO	CE	Output	Single error detected	High

* see GRLIB IP Library User’s Manual

27.7 Library dependencies

Tabel 261 shows libraries used when instantiating the core (VHDL libraries).

*Table 261.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Signals and component declaration

27.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
library gaisler;

use grlib.amba.all;
use gaisler.misc.all;

entity ftram_ex is
  port(
    rst : std_ulogic;
    clk : std_ulogic;

    .... --others signals
  );
end;

architecture rtl of ftram_ex is

--AMBA signals
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_type;
signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector;



--other needed signals here
signal stati   : ahbstat_in_type;
signal aramo   : ahbram_out_type;

begin

--other component instantiations here
...

-- AHB Status Register
  astat0 : ahbstat generic map(pindex => 13, paddr => 13, pirq => 11,
    nftslv => 3)
    port map(rstn, clkm, ahbmi, ahbso, stati, apbi, apbo(13));
    stati.cerror(1 to NAHBSLV-1) <= (others => ‘0’);

--FT AHB RAM
a0 : ftahbram generic map(hindex => 1, haddr => 1, tech => inferred,
  kbytes => 64, pindex => 4, paddr => 4, edacen => 1, autoscrub => 0,
  errcnt => 1, cntbits => 4)
  port map(rst, clk, ahbsi, ahbso(1), apbi, apbo(4), aramo);
  stati.cerror(0) <= aramo.ce;

end architecture;
```

28 FTMCTRL - 8/16/32-bit Memory Controller with EDAC

28.1 Overview

The FTMCTRL combined 8/16/32-bit memory controller provides a bridge between external memory and the AHB bus. The memory controller can handle four types of devices: PROM, asynchronous static ram (SRAM), synchronous dynamic ram (SDRAM) and memory mapped I/O devices (IO). The PROM, SRAM and SDRAM areas can be EDAC-protected using a (39,7) BCH code. The BCH code provides single-error correction and double-error detection for each 32-bit memory word.

The SDRAM area can optionally also be protected using Reed-Solomon coding. In this case a 16-bit checksum is used for each 32-bit word, and any two adjacent 4-bit (nibble) errors can be corrected.

The EDAC capability is determined through a VHDL generic.

The memory controller is configured through three configuration registers accessible via an APB bus interface. The PROM, IO, and SRAM external data bus can be configured in 8-, 16-, or 32-bit mode, depending on application requirements. The controller decodes three address spaces on the AHB bus (PROM, IO, and SRAM/SDRAM). The addresses are determined through VHDL generics. The IO area is marked as non-cacheable in the core’s AMBA plug’n’play information record.

External chip-selects are provided for up to four PROM banks, one IO bank, five SRAM banks and two SDRAM banks. Figure 83 below shows how the connection to the different device types is made.

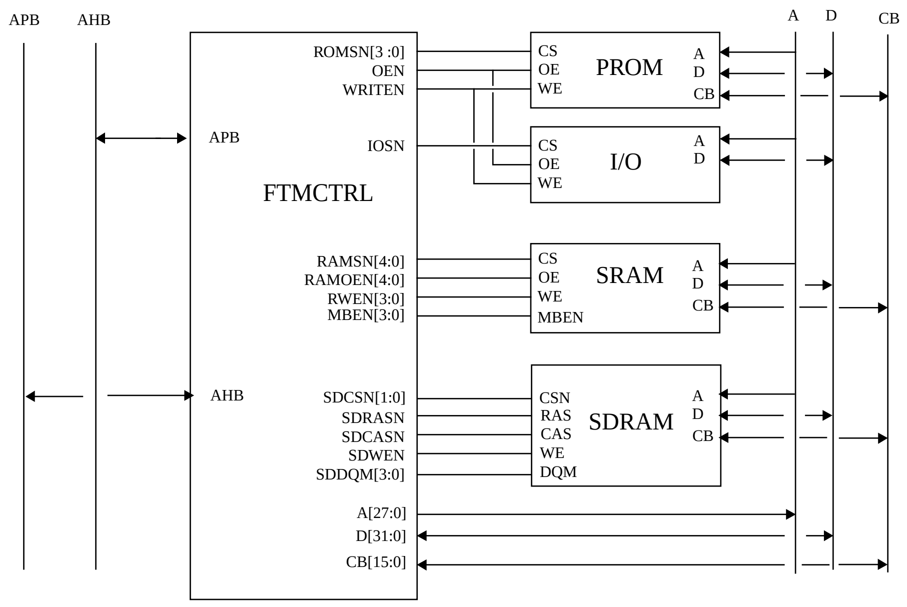


Figure 83. FTMCTRL connected to different types of memory devices

28.2 PROM access

Up to four PROM chip-select signals are provided for the PROM area, ROMSN[3:0]. There are two modes: one with two chip-select signals and one with four. The size of the banks can be set in binary steps from 16KiB to 256MiB. If the AHB memory area assigned to the memory controller for PROM accesses is larger than the combined size of the memory banks then the PROM memory area will wrap, starting with the first chip-select being asserted again when accessing addresses higher than the last decoded bank.

A read access to PROM consists of two data cycles and between 0 and 30 waitstates (in the default configuration, see *wsshift* VHDL generic documentation for details). The read data (and optional

EDAC check-bits) are latched on the rising edge of the clock on the last data cycle. On non-consecutive accesses, a idle cycle is placed between the read cycles to prevent bus contention due to slow turn-off time of PROM devices. Figure 84 shows the basic read cycle waveform (zero waitstate) for non-consecutive PROM reads. Note that the address is undefined in the idle cycle. Figure 85 shows the timing for consecutive cycles (zero waitstate). Waitstates are added by extending the data2 phase. This is shown in figure 86 and applies to both consecutive and non-consecutive cycles. Only an even number of waitstates can be assigned to the PROM area.

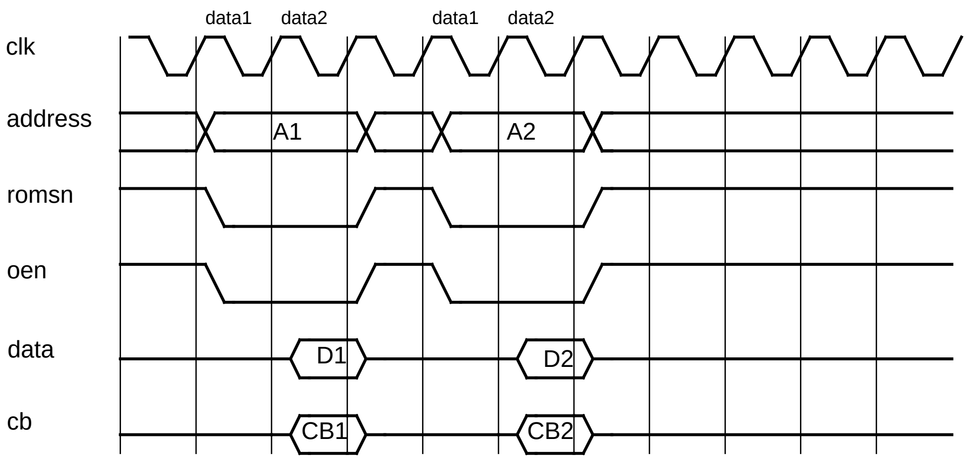


Figure 84. Prom non-consecutive read cyclecs.

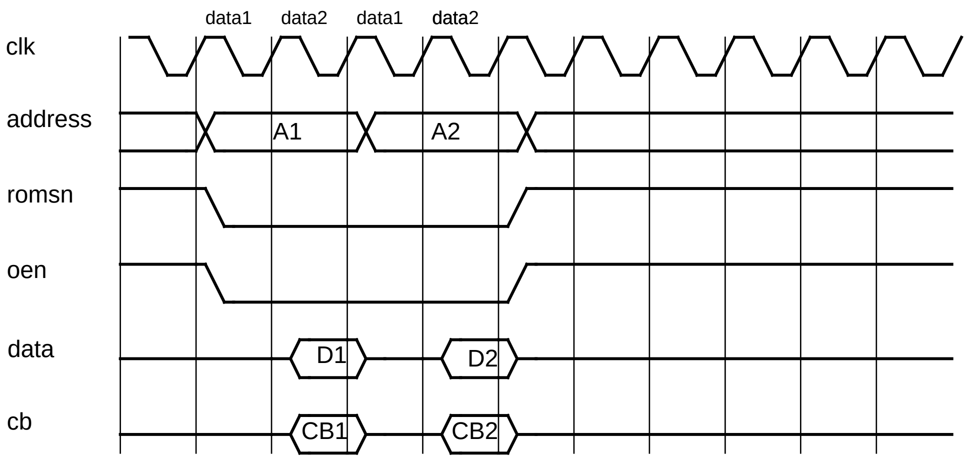


Figure 85. Prom consecutive read cyclecs.

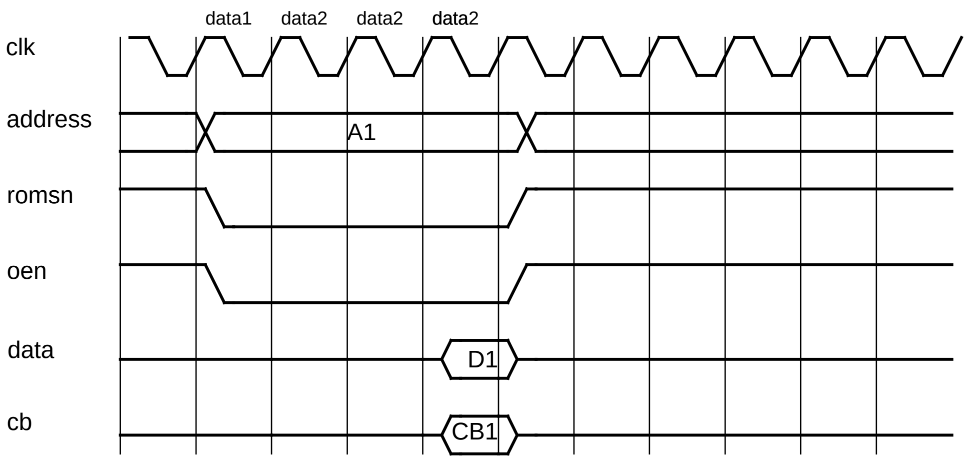


Figure 86. Prom read access with two waitstates.

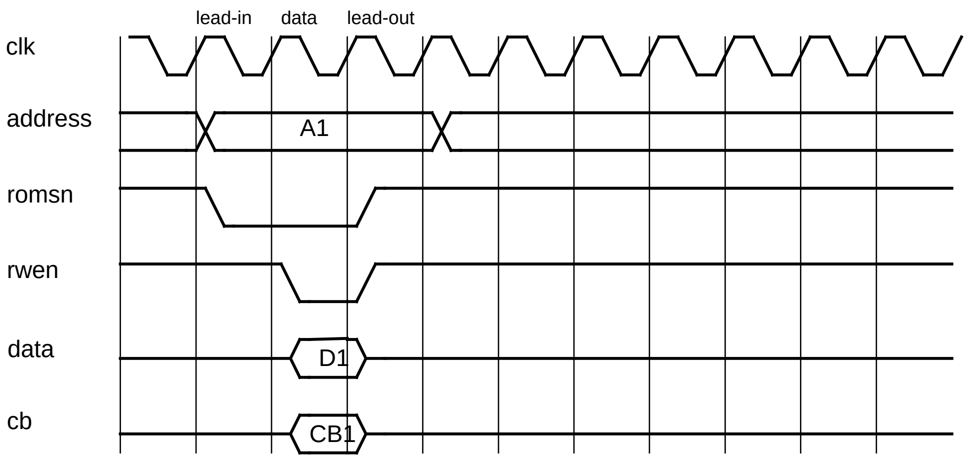


Figure 87. Prom write cycle (0-waitstates)

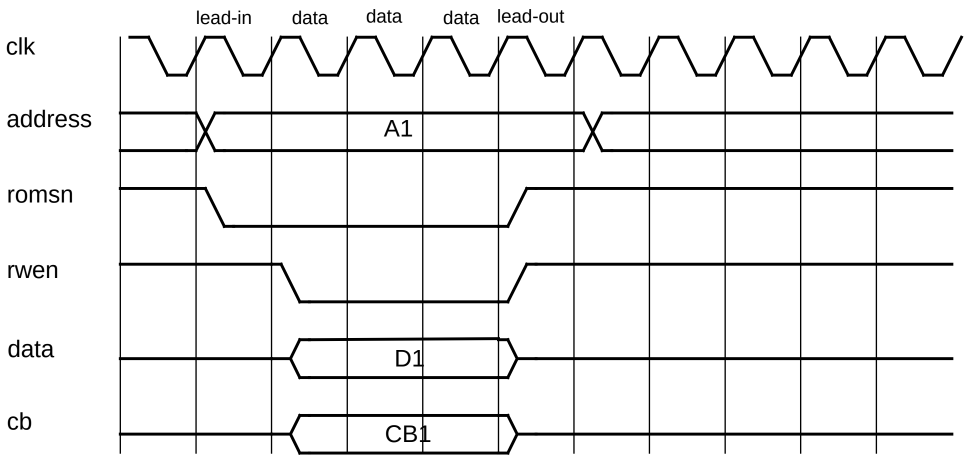


Figure 88. Prom write cycle (2-waitstates)

28.3 Memory mapped IO

Accesses to IO have similar timing as PROM accesses. The IO select (IOSN) and output enable (OEN) signals are delayed one clock to provide stable address before IOSN is asserted. All accesses are performed as non-consecutive accesses as shown in figure 89. The data2 phase is extended when waitstates are added.

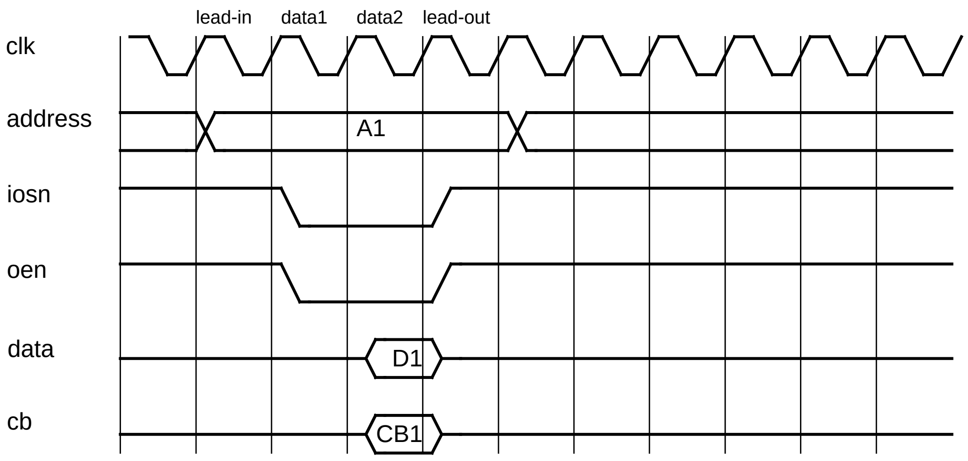


Figure 89. I/O read cycle (0-waitstates)

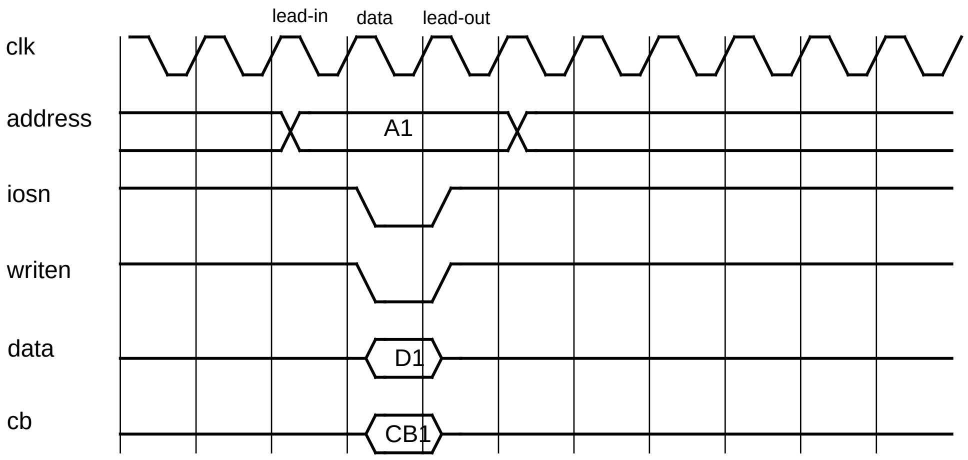


Figure 90. I/O write cycle (0-waitstates)

28.4 SRAM access

The SRAM area is divided on up to five RAM banks. The size of banks 1-4 (RAMSN[3:0]) is programmed in the RAM bank-size field (MCFG2[12:9]) and can be set in binary steps from 8KiB to 256MiB. The fifth bank (RAMSN[4]) decodes the upper 512MiB (controlled by means of the *sdrasel* VHDL generic) and cannot be used simultaneously with SDRAM memory. A read access to SRAM consists of two data cycles and between zero and three waitstates (in the default configuration, see *wsshift* VHDL generic documentation for details). The read data (and optional EDAC check-bits) are latched on the rising edge of the clock on the last data cycle. Accesses to RAMSN[4] can further be stretched by de-asserting BRDYN until the data is available. On non-consecutive accesses, a idle cycle is added after a read cycle to prevent bus contention due to slow turn-off time of memories. Figure 91 shows the basic read cycle waveform (zero waitstate). Waitstates are added in the same way as for PROM in figure 86.

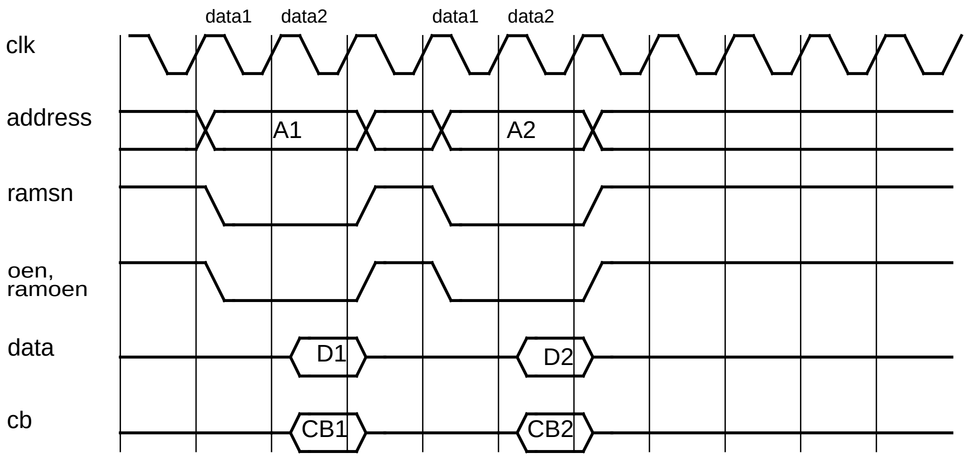


Figure 91. Sram non-consecutive read cyclecs.

For read accesses to RAMSN[4:0], a separate output enable signal (RAMOEN[n]) is provided for each RAM bank and only asserted when that bank is selected. A write access is similar to the read access but takes a minimum of three cycles. Waitstates are added in the same way as for PROM.

Each byte lane has an individual write strobe to allow efficient byte and half-word writes. If the memory uses a common write strobe for the full 16- or 32-bit data, the read-modify-write bit MCFG2 should be set to enable read-modify-write cycles for sub-word writes.

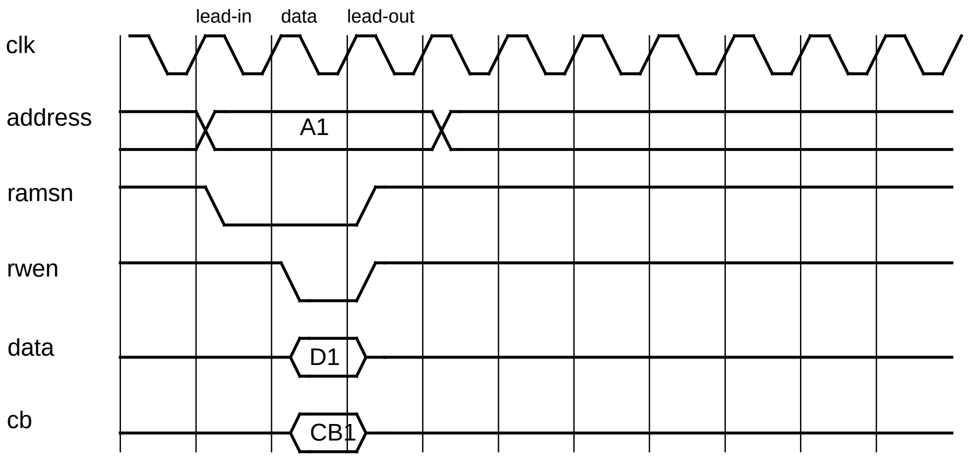


Figure 92. Sram write cycle (0-waitstates)

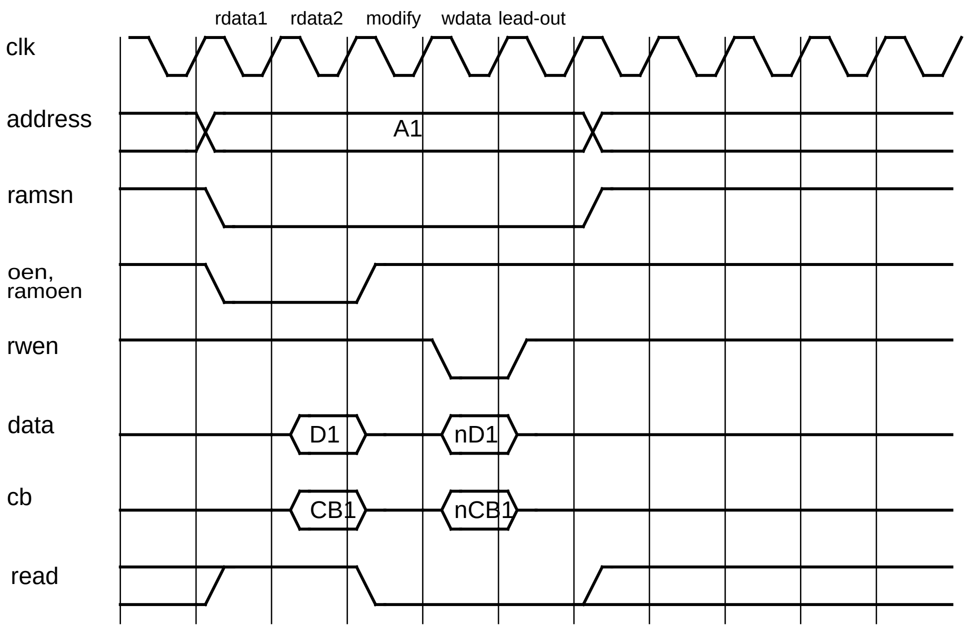


Figure 93. Sram read-modify-write cycle (0-waitstates)

28.5 8-bit and 16-bit PROM and SRAM access

To support applications with low memory and performance requirements efficiently, the SRAM and PROM areas can be individually configured for 8- or 16-bit operation by programming the ROM and RAM width fields in the memory configuration registers. Since reads to memory are always done on 32-bit word basis, read access to 8-bit memory will be transformed in a burst of four read cycles while access to 16-bit memory will generate a burst of two 16-bit reads. During writes, only the necessary bytes will be written. Figure 94 shows an interface example with 8-bit PROM and 8-bit SRAM. Figure 95 shows an example of a 16-bit memory interface.

All possible combinations of width, EDAC, and RMW are not supported. The supported combinations are given in table 262, and the behavior of setting an unsupported combination is undefined. It is not allowed to set the ROM or RAM width fields to 8-bit or 16-bit width if the core does not implement support for these widths.

*Table 262.*FTMCTRL supported SRAM and PROM configurations

PROM/SRAM bus width	**RWEN resolution (SRAM)**	**EDAC**	**RMW bit (SRAM)**	**Core configuration**
8	Bus width	None	0	8-bit support
8	Bus width	BCH	1	8-bit support, EDAC
16	Byte	None	0	16-bit support
16	Bus width	None	1	16-bit support
32	Byte	None	0	
32	Bus width	None	1	
32+7	Bus width	BCH	1	EDAC support

8-bit width support is set with *ram8* VHDL generic and 16-bit width support is set with *ram16* VHDL genericis.

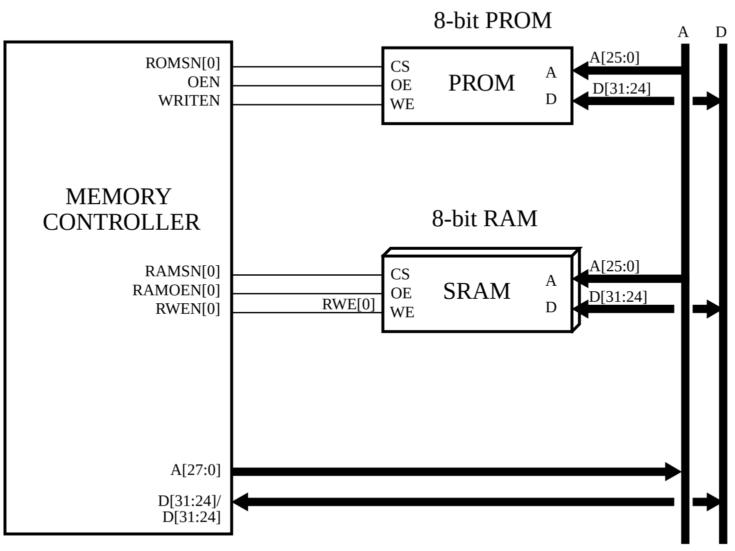


Figure 94. 8-bit memory interface example

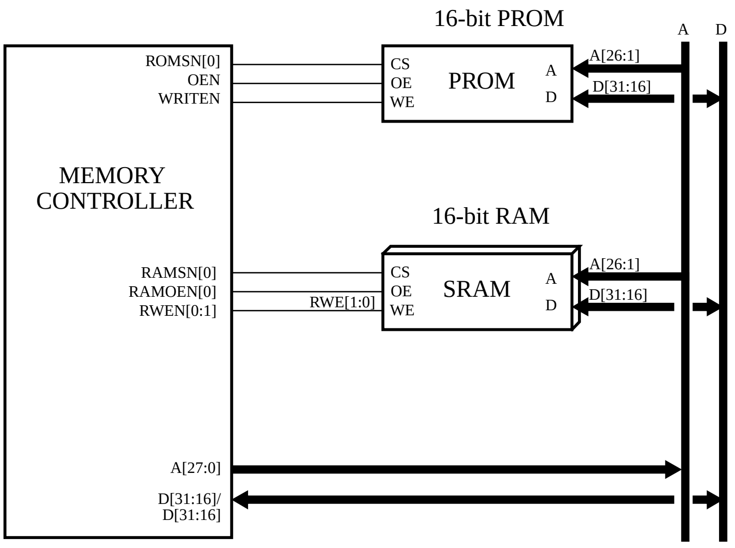


Figure 95. 16-bit memory interface example

In 8-bit mode, the PROM/SRAM devices should be connected to the MSB byte of the data bus (D[31:24]). The LSB address bus should be used for addressing (A[25:0]). In 16-bit mode, D[31:16] should be used as data bus, and A[26:1] as address bus.

28.6 8- and 16-bit I/O access

Similar to the PROM/SRAM areas, the IO area can also be configured to 8- or 16-bits mode. However, the I/O device will NOT be accessed by multiple 8/16 bits accesses as the memory areas, but only with one single access just as in 32-bit mode. To access an IO device on an 8-bit bus, only byte accesses should be used (LDUB/STB instructions for the CPU). To accesses an IO device on a 16-bit bus, only halfword accesses should be used (LDUH/STH instructions for the CPU).

To access the I/O-area in 8- or 16-bit mode, *ram8* VHDL generic or *ram16* VHDL generic must be set respectively.

28.7 Burst cycles

To improve the bandwidth of the memory bus, accesses to consecutive addresses can be performed in burst mode. Burst transfers will be generated when the memory controller is accessed using an AHB burst request. These includes instruction cache-line fills, double loads and double stores. The timing of a burst cycle is identical to the programmed basic cycle with the exception that during read cycles, the idle cycle will only occurs after the last transfer. Burst cycles will not be generated to the IO area.

Only word (HSIZE = “010”) bursts of incremental type (HBURST=INCR, INCR4, INCR8 or INCR16) are supported.

28.8 SDRAM access

28.8.1 General

Synchronous dynamic RAM (SDRAM) access is supported to two banks of PC100/PC133 compatible devices. This is implemented by a special version of the SDCTRL SDRAM controller core from Aeroflex Gaisler, which is optionally instantiated as a sub-block. The SDRAM controller supports 64M, 256M and 512M devices with 8 - 12 column-address bits, and up to 13 row-address bits. The size of the two banks can be programmed in binary steps between 4MiB and 512MiB. The operation of the SDRAM controller is controlled through MCFG2 and MCFG3 (see below). Both 32- and 64-bit data bus width is supported, allowing the interface of 64-bit DIMM modules. The memory controller can be configured to use either a shared or separate bus connecting the controller and SDRAM devices.

28.8.2 Address mapping

The two SDRAM chip-select signals are decoded. SDRAM area is mapped into the upper half of the RAM area defined by BAR2 register, and cannot be used simultaneously with fifth SRAM bank (RAMSN[4]). When the SDRAM enable bit is set in MCFG2, the controller is enabled and mapped into upper half of the RAM area as long as the SRAM disable bit is not set. If the SRAM disable bit is set, all access to SRAM is disabled and the SDRAM banks are mapped into the lower half of the RAM area.

28.8.3 Initialisation

When the SDRAM controller is enabled, it automatically performs the SDRAM initialisation sequence of PRECHARGE, 8x AUTO-REFRESH and LOAD-MODE-REG on both banks simultaneously. The controller programs the SDRAM to use single location access on write. The controller programs the SDRAM to use line burst of length 8 when *pageburst* VHDL generic is 0. The controller programs the SDRAM to use page burst when *pageburst* VHDL generic is 1. The controller programs the SDRAM to use page burst or line burst of length 8, selectable via the MCFG2 register, when *pageburst* VHDL generic is 2.

28.8.4 Configurable SDRAM timing parameters

To provide optimum access cycles for different SDRAM devices (and at different frequencies), three SDRAM parameters can be programmed through memory configuration register 2 (MCFG2): TCAS, TRP and TRFCD. The value of these field affects the SDRAM timing as described in table 263.

*Table 263.*SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
CAS latency, RAS/CAS delay (t_{CAS}, t_{RCD})	TCAS + 2
Precharge to activate (t_{RP})	TRP + 2
Auto-refresh command period (t_{RFC})	TRFC + 3
Activate to precharge (t_{RAS})	TRFC + 1
Activate to Activate (t_{RC})	TRP + TRFC + 4

If the TCAS, TRP and TRFC are programmed such that the PC100/133 specifications are fulfilled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 100 and 133 MHz operation and the resulting SDRAM timing (in ns):

*Table 264.*SDRAM example programming

SDRAM settings	t_{CAS}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
100 MHz, CL=2; TRP=0, TCAS=0, TRFC=4	20	80	20	70	50
100 MHz, CL=3; TRP=0, TCAS=1, TRFC=4	30	80	20	70	50
133 MHz, CL=2; TRP=1, TCAS=0, TRFC=6	15	82	22	67	52
133 MHz, CL=3; TRP=1, TCAS=1, TRFC=6	22	82	22	67	52

28.8.5 Refresh

The SDRAM controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the MCFG3 register. Depending on SDRAM type, the required period is typically 7.8 or 15.6 µs (corresponding to 780 or 1560 clocks at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by setting bit 31 in MCFG2.

28.8.6 SDRAM commands

The controller can issue three SDRAM commands by writing to the SDRAM command field in MCFG2: PRE-CHARGE, AUTO-REFRESH and LOAD-MODE-REG (LMR). If the LMR command is issued, the CAS delay as programmed in MCFG2 will be used. Line burst of length 8 will be set for read when *pageburst* VHDL generic is 0. Page burst will be set for read when *pageburst* VHDL generic is 1. Page burst or line burst of length 8, selectable via the MCFG2 register will be set, when *pageburst* VHDL generic is 2. Remaining fields are fixed: single location write, sequential burst. The command field will be cleared after a command has been executed. When changing the value of the CAS delay, a LOAD-MODE-REGISTER command should be generated at the same time. NOTE: when issuing SDRAM commands, the SDRAM refresh must be disabled.

28.8.7 Read cycles

A read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command after the programmed CAS delay. A read burst is performed if a burst access has been requested on the AHB bus. The read cycle is terminated with a PRE-CHARGE command, no banks are left open between two accesses.

28.8.8 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. A write burst on the AHB bus will generate a burst of write commands without idle cycles in-between.

After the WRITE command has completed, if there is an immediately following read or write access (not RMW) to the same 1KiB page on the AHB bus, this access is performed during the same access cycle without closing and re-opening the row.

28.8.9 Read-modify-write cycles

If EDAC is enabled and a byte or half-word write is performed, the controller will perform a read-modify-write cycle to update the checkbits correctly. This is done by performing an ACTIVATE command, followed by READ, WRITE and PRE-CHARGE. The write command interrupts the read burst and the data mask signals will be raised two cycles before this happens as required by the SDRAM standard.

28.8.10 Address bus

The memory controller can be configured to either share the address and data buses with the SRAM, or to use separate address and data buses. When the buses are shared, the address bus of the SDRAMs should be connected to A[14:2], the bank address to A[16:15]. The MSB part of A[14:2] can be left unconnected if not used. When separate buses are used, the SDRAM address bus should be connected to SA[12:0] and the bank address to SA[14:13].

28.8.11 Data bus

SDRAM can be connected to the memory controller through the common or separate data bus. If the separate bus is used the width is configurable to 32 or 64 bits. 64-bit data bus allows the 64-bit SDRAM devices to be connected using the full data capacity of the devices. 64-bit SDRAM devices can be connected to 32-bit data bus if 64-bit data bus is not available but in this case only half the full data capacity will be used. There is a drive signal vector and separate data vector available for SDRAM. The drive vector has one drive signal for each data bit. These signals can be used to remove timing problems with the output delay when a separate SDRAM bus is used.

28.8.12 Clocking

The SDRAM controller is designed for an external SDRAM clock that is in phase or slightly earlier than the internal AHB clock. This provides the maximum margin for setup and hold on the external signals, and allows highest possible frequency. For Xilinx and Altera device, the GRLIB Clock Generator (CLKGEN) can be configured to produce a properly synchronized SDRAM clock. For other FPGA targets, the custom clock synchronization must be designed. For ASIC targets, the SDRAM clock can be derived from the AHB clock with proper delay adjustments during place&route.

28.8.13 Initialisation

Each time the SDRAM is enabled (bit 14 in MCFG2), an SDRAM initialisation sequence will be sent to both SDRAM banks. The sequence consists of one PRECHARGE, eight AUTO-REFRESH and one LOAD-COMMAND-REGISTER command.

28.9 Memory EDAC

28.9.1 BCH EDAC

The FTMCTRL is provided with an BCH EDAC that can correct one error and detect two errors in a 32-bit word. For each word, a 7-bit checksum is generated according to the equations below. A correctable error will be handled transparently by the memory controller, but adding one waitstate to the access. If an un-correctable error (double-error) is detected, the current AHB cycle will end with an error response. The EDAC can be used during access to PROM, SRAM and SDRAM areas by setting the corresponding EDAC enable bits in the MCFG3 register. The equations below show how the EDAC checkbits are generated:

```
CB0 = D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14 ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31
CB1 = D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12 ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28
C̅B̅2̅ = D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15 ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31
C̅B̅3̅ = D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13 ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29
CB4 = D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15 ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31
CB5 = D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
CB6 = D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
```

If the SRAM is configured in 8-bit mode, the EDAC checkbit bus (CB[7:0]) is not used but it is still possible to use EDAC protection. Data is always accessed as words (4 bytes at a time) and the corresponding checkbits are located at the address acquired by inverting the word address (bits 2 to 27) and using it as a byte address. The same chip-select is kept active. A word written as four bytes to addresses 0, 1, 2, 3 will have its checkbits at address 0xFFFFFFF, addresses 4, 5, 6, 7 at 0xFFFFFFE and so on. All the bits up to the maximum bank size will be inverted while the same chip-select is always asserted. This way all the bank sizes can be supported and no memory will be unused (except for a maximum of 4 byte in the gap between the data and checkbit area). A read access will automatically read the four data bytes individually from the nominal addresses and the EDAC checkbit byte from the top part of the bank. A write cycle is performed the same way. Byte or half-word write accesses will result in an automatic read-modify-write access where 4 data bytes and the checkbit byte are firstly read, and then 4 data bytes and the newly calculated checkbit byte are writen back to the memory. This 8-bit mode applies to SRAM while SDRAM always uses 32-bit accesses. The size of the memory bank is determined from the settings in MCFG2. The EDAC cannot be used on memory areas configured in 16-bit mode.

If the ROM is configured in 8-bit mode, EDAC protection is provided in a similar way as for the SRAM memory described above. The difference is that write accesses are not being handled automatically. Instead, write accesses must only be performed as individual byte accesses by the software, writing one byte at a time, and the corresponding checkbit byte must be calculated and be written to the correct location by the software.

NOTE: when the EDAC is enabled in 8-bit bus mode, only the first bank select (RAMSN[0], PROMSN[0]) can be used.

The operation of the EDAC can be tested trough the MCFG3 register. If the WB (write bypass) bit is set, the value in the TCB field will replace the normal checkbits during memory write cycles. If the RB (read bypass) is set, the memory checkbits of the loaded data will be stored in the TCB field during memory read cycles. NOTE: when the EDAC is enabled, the RMW bit in memory configuration register 2 must be set.

Data access timing with EDAC enabled is identical to access without EDAC, if the *edac* VHDL generic is set to 1. To improve timing of the HREADY output, a pipeline stage can be inserted in the EDAC error detection by setting the *edac* VHDL generic to 2. One clock extra latency will then occur on single word reads, or on the first data word in a burst.

EDAC is not supported for 64-bit wide SDRAM data buses.

28.9.2 Reed-Solomon EDAC

The Reed-Solomon EDAC provides block error correction, and is capable of correcting up to two 4-bit nibble errors in a 32-bit data word or 16-bit checksum. The Reed-Solomon EDAC can be enabled for the SDRAM area only, and uses a 16-bit checksum. Operation and timing is identical to the BCH EDAC with the pipeline option enabled. The Reed-Solomon EDAC is enabled by setting the RSE and

RE bits in MCFG3, and the RMW bit in MCFG2. The Reed-Solomon EDAC is not supported for 64-bit wide SDRAM buses.

The Reed-Solomon data symbols are 4-bit wide, represented as GF(2^4). The basic Reed-Solomon code is a shortened RS(15, 13, 2) code, represented as RS(6, 4, 2). It has the capability to detect and correct a single symbol error anywhere in the codeword. The EDAC implements an interleaved RS(6, 4, 2) code where the overall data is represented as 32 bits and the overall checksum is represented as 16 bits. The codewords are interleaved nibble-wise. The interleaved code can correct two 4-bit errors when each error is located in a nibble and not in the same original RS(6, 4, 2) codeword.

The Reed-Solomon RS(15, 13, 2) code has the following definition:

- there are 4 bits per symbol;
- there are 15 symbols per codeword;
- the code is systematic;
- the code can correct one symbol error per codeword;
- the field polynomial is

$$f(x) = x^4 + x + 1$$

- the code generator polynomial is

$$g(x) = \prod_{i=0}^{1} (x + \alpha^i) = \sum_{j=0}^{2} g_j \cdot x^j$$

  for which the highest power of *x* is stored first;
- a codeword is defined as 15 symbols:

  $c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}$

  where c_0 to c_{12} represent information symbols and c_{13} to c_{14} represent check symbols.

The shortened and interleaved RS(6, 4, 2) code has the following definition:

- the codeword length is shortened to 4 information symbols and 2 check symbols and as follows:

  $c_0 = c_1 = c_2 = c_3 = c_4 = c_5 = c_6 = c_7 = c_8 = 0$

  where the above information symbols are suppressed or virtually filled with zeros;
- two codewords are interleaved (i.e. interleaved depth *I*=2) with the following mapping to the 32-bit data and 16-bit checksum, were $c_{i,j}$ is a symbol with codeword index *i* and symbol index *j*:

  $c_{0,9}$ = sd[31:28]

  $c_{1,9}$ = sd[27:24]

  $c_{0,10}$ = sd[23:20]

  $c_{1,10}$ = sd[19:16]

  $c_{0,11}$ = sd[15:12]

  $c_{1,11}$ = sd[11:8]

  $c_{0,12}$ = sd[7:4]

  $c_{1,12}$ = sd[3:0]

  $c_{0,13}$ = scb[15:12]

  $c_{1,13}$ = scb[11:8]

$c_{0,14}$ = scb[7:4]

$c_{1,14}$ = scb[3:0]

where SD[ ] is interchanable with DATA[] and SCB[ ] is interchangable with CB[ ]

Note that the FTMCTRL must have the *edac* VHDL generic set to 3 to enable the RS EDAC functionality. The Reed-Solomon EDAC is not supported for 64-bit wide SDRAM buses.

28.9.3 EDAC Error reporting

As mentioned above an un-correctable error results in an AHB error response which can be monitored on the bus. Correctable errors however are handled transparently and are not visible on the AHB bus. A sideband signal is provided which is asserted during one clock cycle for each access for which a correctable error is detected. This can be used for providing an external scrubbing mechanism and/or statistics. The correctable error signal is most commonly connected to the AHB status register which monitors both this signal and error responses on the bus. Please see the AHB status register section for more information.

28.10 Bus Ready signalling

The BRDYN signal can be used to stretch all types of access cycles to the PROM, I/O area and the SRAM area decoded by RAMSN[4]. This covers read and write accesses in general, and additionally read-modify-write accesses to the SRAM area. The accesses will always have at least the pre-programmed number of waitstates as defined in memory configuration registers 1 & 2, but will be further stretched until BRDYN is asserted. BRDYN should be asserted in the cycle preceding the last one. If bit 29 in MCFG1 is set, BRDYN can be asserted asynchronously with the system clock. In this case, the read data must be kept stable until the de-assertion of OEN/RAMOEN and BRDYN must be asserted for at least 1.5 clock cycle. The use of BRDYN can be enabled separately for the PROM, I/O and RAMSN[4] areas. It is recommended that BRDYN is asserted until the corresponding chip select signal is de-asserted, to ensure that the access has been properly completed and avoiding the system to stall.

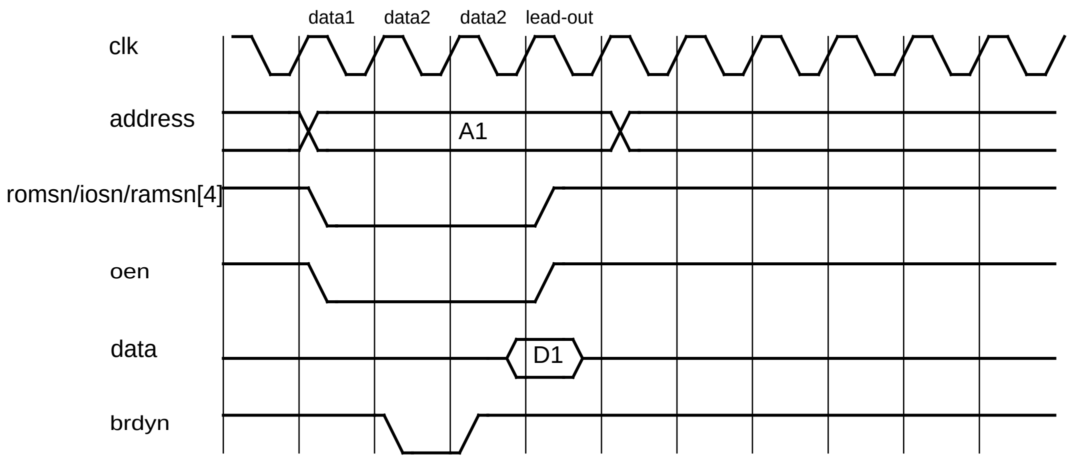


Figure 96. READ cycle with one extra data2 cycle added with BRDYN (synchronous sampling). Lead-out cycle is only applicable for I/O accesses.

Figure 97 shows the use of BRDYN with asynchronous sampling. BRDYN is kept asserted for more than 1.5 clock-cycle. Two synchronization registers are used so it will take at least one additional cycle from when BRDYN is first asserted until it is visible internally. In figure 97 one cycle is added to the data2 phase.

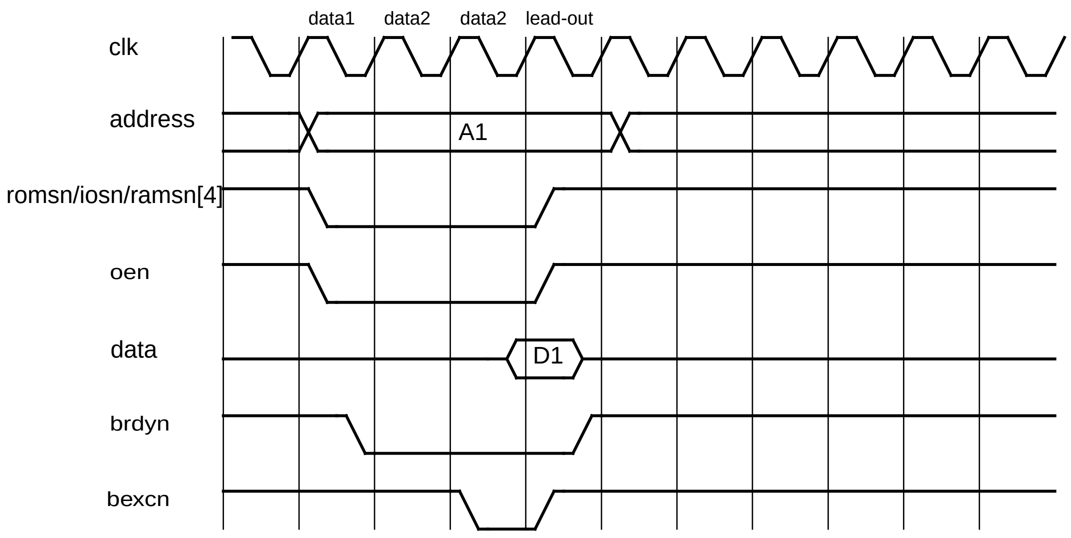


Figure 97. BRDYN (asynchronous) sampling and BEXCN timing. Lead-out cycle is only applicable for I/O-accesses.

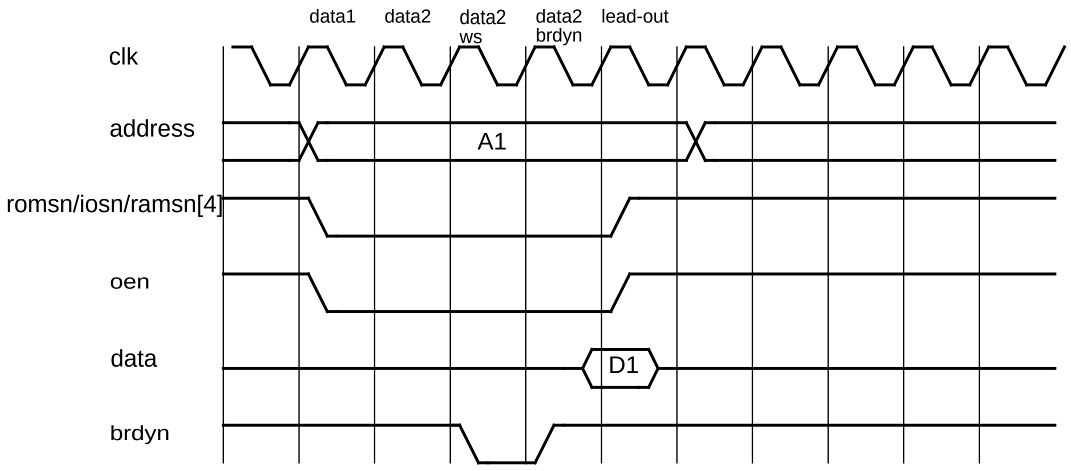


Figure 98. Read cycle with one waitstate (configured) and one BRDYN generated waitstate (synchronous sampling).

If burst accesses and BRDYN signaling are to be used together, special care needs to be taken to make sure BRDYN is raised between the separate accesses of the burst. The controller does not raise the select and OEN signal (in the read case) between accesses during the burst so if BRDYN is kept asserted until the select signal is raised, all remaining accesses in the burst will finish with the configured fixed number of wait states.

28.11 Access errors

An access error can be signalled by asserting the BEXCN signal for read and write accesses. For reads it is sampled together with the read data. For writes it is sampled on the last rising edge before chip select is de-asserted, which is controlled by means of waitstates or bus ready signalling. If the usage of BEXCN is enabled in memory configuration register 1, an error response will be generated on the internal AHB bus. BEXCN can be enabled or disabled through memory configuration register 1, and is active for all areas (PROM, IO and RAM). BEXCN is only sampled in the last access for 8- and 16-bit mode for RAM and PROM. That is, when four bytes are written for a word access to 8-bit wide memory BEXCN is only sampled in the last access with the same timing as a single access in 32-bit mode.

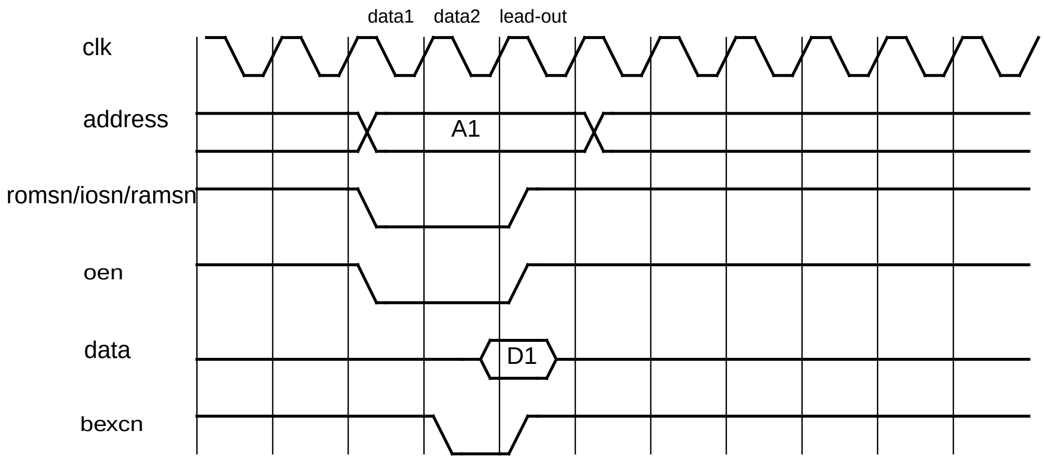


Figure 99. Read cycle with BEXCN.

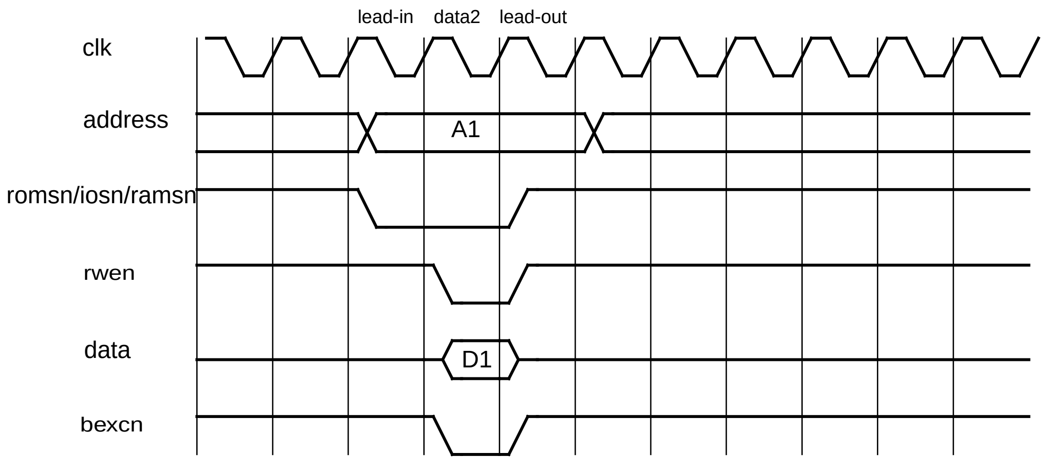


Figure 100. Write cycle with BEXCN. Chip-select (iosn) is not asserted in lead-in cycle for io-accesses.

28.12 Attaching an external DRAM controller

To attach an external DRAM controller, RAMSN[4] should be used since it allows the cycle time to vary through the use of BRDYN. In this way, delays can be inserted as required for opening of banks and refresh.

28.13 Output enable timing

A drive signal vector for the data I/O-pads is provided which has one drive signal for each data bit. It can be used if the synthesis tool does not generate separate registers automatically for the current technology. This can remove timing problems with output delay. An additional vector is used for the separate SDRAM bus.

28.14 Read strobe

The READ signal indicates the direction of the current PROM,SRAM,IO or SDRAM transfer, and it can be used to drive external bi-directional buffers on the data bus. It always is valid at least one cycle before and after the bus is driven, at other times it is held either constant high or low.

28.15 Registers

The core is programmed through registers mapped into APB address space.

*Table 265.*FTMCTRL memory controller registers

APB Address offset	**Register**
0x0	Memory configuration register 1 (MCFG1)
0x4	Memory configuration register 2 (MCFG2)
0x8	Memory configuration register 3 (MCFG3)
0xC	Memory configuration register 4 (MCFG4)

28.15.1 Memory configuration register 1 (MCFG1)

Memory configuration register 1 is used to program the timing of rom and IO accesses.

Table 266. Memory configuration register 1

31	30	29	28	27	26	25	24	23	20	19	18	17
	PBRDY	ABRDY	IOBUSW		IBRDY	BEXCN		IO WAITSTATES		IOEN		ROMBANKSZ

	14	13	12	11	10	9	8	7	4	3	0
		RESERVED		PWEN		PROM WIDTH		PROM WRITE WS		PROM READ WS	

31	RESERVED
30	PROM area bus ready enable (PBRDY) - Enables bus ready (BRDYN) signalling for the PROM area. Reset to ‘0’.
29	Asynchronous bus ready (ABRDY) - Enables asynchronous bus ready.
28 : 27	I/O bus width (IOBUSW) - Sets the data width of the I/O area (“00”=8, “01”=16, “10” =32).
26	I/O bus ready enable (IBRDY) - Enables bus ready (BRDYN) signalling for the I/O area. Reset to ‘0’.
25	Bus error enable (BEXCN) - Enables bus error signalling for all areas. Reset to ‘0’.
24	RESERVED
23 : 20	I/O waitstates (IO WAITSTATES) - Sets the number of waitstates during I/O accesses (“0000”=0, “0001”=1, “0010”=2,..., “1111”=15). The values above describe the default configuration The core can be configred at implementation to extend the number of waitstates. The number of wait states inserted will be (IO WAIT-STATES)*2^{wsshift}, where *wsshift* can be read from the first user-defined register in the core’s plug&play area (default is wsshift = 0).
19	I/O enable (IOEN) - Enables accesses to the memory bus I/O area.
18	RESERVED
17: 14	PROM bank size (ROMBANKSZ) - Returns current PROM bank size when read. “0000” is a special case and corresponds to a bank size of 256MiB. All other values give the bank size in binary steps: “0001”=16KiB, “0010”=32KiB, “0011”=64KiB,... , “1111”=256MiB (i.e. 8KiB * 2**ROMBANKSZ). For value “0000” or “1111” only two chip selects are available. For other values, two chip select signals are available for fixed bank sizes. For other values, four chip select signals are available for programmable bank sizes. Programmable bank sizes can be changed by writing to this register field. The written values correspond to the bank sizes and number of chip-selects as above. Reset to “0000” when programmable. Programmable ROMBANKSZ is only available when romasel VHDL generic is 0. For other values this is a read-only register field containing the fixed bank size value.
13:12	RESERVED
11	PROM write enable (PWEN) - Enables write cycles to the PROM area.
10	RESERVED
9 : 8	PROM width (PROM WIDTH) - Sets the data width of the PROM area (“00”=8, “01”=16, “10”=32).

Table 266. Memory configuration register 1

7 : 4	PROM write waitstates (PROM WRITE WS) - Sets the number of wait states for PROM write cycles (“0000”=0, “0001”=2, “0010”=4,..., “1111”=30). The values above describe the default configuration The core can be configred at implementation to extend the number of waitstates. The number of wait states inserted will be (PROM WRITE WS)*2*$2^{wsshift}$, where *wsshift* can be read from the first user-defined register in the core’s plug&play area (default is wsshift = 0).
3 : 0	PROM read waitstates (PROM READ WS) - Sets the number of wait states for PROM read cycles (“0000”=0, “0001”=2, “0010”=4,...,”1111”=30). Reset to “1111”. The values above describe the default configuration The core can be configred at implementation to extend the number of waitstates. The number of wait states inserted will be (PROM READ WS)*2*$2^{wsshift}$, where *wsshift* can be read from the first user-defined register in the core’s plug&play area (default is wsshift = 0).

During reset, the prom width (bits [9:8]) are set with value on BWIDTH inputs. The prom waitstates fields are set to 15 (maximum). External bus error and bus ready are disabled. All other fields are undefined.

28.15.2 Memory configuration register 2 (MCFG2)

Memory configuration register 2 is used to control the timing of the SRAM and SDRAM.

Table 267. Memory configuration register 2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SDRF	TRP	SDRAM TRFC			TCAS	SDRAM BANKSZ			SDRAM COLSZ		SDRAM CMD		D64	SDPB	
15	14	13	12			9	8	7	6	5	4	3	2	1	0
	SE	SI	RAM BANK SIZE					RBRDY	RMW	RAM WIDTH		RAM WRITE WS		RAM READ WS	

31	SDRAM refresh (SDRF) - Enables SDRAM refresh.
30	SRAM TRP parameter (TRP) - t_{RP} will be equal to 2 or 3 system clocks (0/1).
29 : 27	SDRAM TRFC parameter (SDRAM TRFC) - t_{RFC} will be equal to 3+field-value system clocks.
26	SDRAM TCAS parameter (TCAS) - Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-COMMAND-REGISTER command must be issued at the same time. Also sets RAS/CAS delay (t_{RCD}).
25 : 23	SDRAM bank size (SDRAM BANKSZ) - Sets the bank size for SDRAM chip selects (“000”=4MiB, “001”=8MiB, “010”=16MiB,...,. “111”=512MiB).
22 : 21	SDRAM column size (SDRAM COLSZ) - “00”=256, “01”=512, “10”=1024, “11”=4096 when bit 25:23=”111” 2048 otherwise.
20 : 19	SDRAM command (SDRAM CMD) - Writing a non-zero value will generate a SDRAM command. “01”=PRECHARGE, “10”=AUTO-REFRESH, “11”=LOAD-COMMAND-REGISTER. The field is reset after the command has been executed.
18	64-bit SDRAM data bus (D64) - Reads ‘1’ if the memory controller is configured for 64-bit SDRAM data bus width, ‘0’ otherwise. Read-only.
17	SDRAM Page Burst (SDPB) - SDRAM programmed for page bursts on read when set, else programmed for line burst lengths of 8 on read. Programmable when pageburst VHDL generic is 2, else read-only.
16 : 15	RESERVED
14	SDRAM enable (SE) - Enables the SDRAM controller and disables fifth SRAM bank (RAMSN[4]).
13	SRAM disable (SI) - Disables accesses to SRAM bank if bit 14 (SE) is set to ‘1’.
12 : 9	RAM bank size (RAM BANK SIZE) - Sets the size of each RAM bank (“0000”=8KiB, “0001”=16KiB, “0010”=32KiB, “0011”= 64KiB,.., “1111”=256MiB)(i.e. 8KiB * 2**RAM BANK SIZE).
8	RESERVED
7	RAM bus ready enable (RBRDY) - Enables bus ready signalling for the RAM area.
6	Read-modify-write enable (RMW) - Enables read-modify-write cycles for sub-word writes to 16- bit 32-bit areas with common write strobe (no byte write strobe).

Table 267. Memory configuration register 2

5 : 4	RAM width (RAM WIDTH) - Sets the data width of the RAM area (“00”=8, “01”=16, “1X”=32).
3 : 2	RAM write waitstates (RAM WRITE WS) - Sets the number of wait states for RAM write cycles (“00”=0, “01”=1, “10”=2, “11”=3). The values above describe the default configuration The core can be configred at implementation to extend the number of waitstates. The number of wait states inserted will be (RAM WRITE WS)*$2^{wsshift}$, where *wsshift* can be read from the first user-defined register in the core’s plug&play area (default is wsshift = 0).
1 : 0	RAM read waitstates (RAM READ WS) - Sets the number of wait states for RAM read cycles (“00”=0, “01”=1, “10”=2, “11”=3). The values above describe the default configuration The core can be configred at implementation to extend the number of waitstates. The number of wait states inserted will be (RAM READ WS)*$2^{wsshift}$, where *wsshift* can be read from the first user-defined register in the core’s plug&play area (default is wsshift = 0).

28.15.3 Memory configuration register 3 (MCFG3)

MCFG3 contains the reload value for the SDRAM refresh counter and to control and monitor the memory EDAC.

Table 268. Memory configuration register 3

31	28	27	26
RESERVED	RSE	ME	SDRAM REFRESH COUNTER

12	11	10	9	8	7	0
	WB	RB	RE	PE	TCB	

31 : 29	RESERVED
28	Reed-Solomon EDAC enable (RSE) - if set, will enable Reed-Solomon protection of SDRAM area when implemented
27	Memory EDAC (ME) - Indicates if memory EDAC is present. (read-only)
26 : 12	SDRAM refresh counter reload value (SDRAM REFRESH COUNTER)
11	EDAC diagnostic write bypass (WB) - Enables EDAC write bypass.
10	EDAC diagnostic read bypass (RB) - Enables EDAC read bypass.
9	RAM EDAC enable (RE) - Enable EDAC checking of the RAM area (including SDRAM).
8	PROM EDAC enable (PE) - Enable EDAC checking of the PROM area. Ar reset, this bit is initialized with the value of MEMI.EDAC.
7 : 0	Test checkbits (TCB) - This field replaces the normal checkbits during write cycles when WB is set. It is also loaded with the memory checkbits during read cycles when RB is set.

The period between each AUTO-REFRESH command is calculated as follows:

$t_{REFRESH}$ = ((reload value) + 1) / SYSCLK

28.15.4 Memory configuration register 4 (MCFG4)

MCFG4 is only present if the Reed-Solomon EDAC has been enabled with the *edac* VHDL generic.

MCFG4 provides means to insert Reed-Solomon EDAC errors into memory for diagnostic purposes.

Table 269. Memory configuration register 4

31	16
RESERVED	WB

15	0
TCB[15:0]	

31 : 17	RESERVED

Table 269. Memory configuration register 4

16	EDAC diagnostic write bypass (WB) - Enables EDAC write bypass. Identical to WB in MCFG3.
15 : 0	Test checkbits (TCB) - This field replaces the normal checkbits during write cycles when WB is set. It is also loaded with the memory checkbits during read cycles when RB is set. Note that TCB[7:0] are identical to TCB[7:0] in MCFG3

28.16 Vendor and device identifiers

The core has vendor identifier 0x01 (GAISLER) and device identifier 0x054. For description of vendor and device identifiers, see GRLIB IP Library User's Manual.

28.17 Configuration options

Table 270 shows the configuration options of the core (VHDL generics).

*Table 270.*Configuration options

Generic	**Function**	**Allowed range**	**Default**
hindex	AHB slave index	1 - NAHBSLV-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0x1FFFFFFF. Also see documentation of romasel VHDL generic below.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.. Also see documentation of romasel VHDL generic below.	0 - 16#FFF#	16#E00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space. Default I/O area is 0x20000000 - 0x2FFFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#E00#
ramaddr	ADDR field of the AHB BAR2 defining RAM address space. Default RAM area is 0x40000000-0x7FFFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR2 defining RAM address space.	0 -16#FFF#	16#C00#
paddr	ADDR field of the APB BAR configuration registers address space.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR configuration registers address space.	0 - 16#FFF#	16#FFF#
wprot	RAM write protection.	0 - 1	0
invclk	unused	N/A	0
fast	Enable fast SDRAM address decoding.	0 - 1	0

*Table 270.*Configuration options

Generic	Function	Allowed range	Default
romasel	Sets the PROM bank size. *romasel 0*: selects a programmable mode where the ROM-BANKSZ field in the MCFG1 register sets the bank size. When romasel is 0 and the bank size is configured (MCFG1 register, ROMBANKSZ field, via the core's register interface) to 0b000 or 0b1111 then address bit 28 is used to decode the banks. This means that the core must be mapped at a 512 MiB address boundary (0x0, 0x20000000, 0x40000000, .. see romaddr and rommask VHDL generics) for address decoding to work correctly. *romasel 1 - 14:* Values 1 - 14 sets the size in binary steps (1 = 16KiB, 2 = 32KiB, 3=64KiB, ..., 14=128MiB). Four chip-selects are available for these values. 15 sets the bank size to 256MiB with two chip-selects. *romasel 1- 16:* Values 16 - 28 sets the bank size in binary steps (16 = 64 KiB, 17 = 128KiB, ... 28 = 256MiB). Two chip-selects are available for this range. The selected bank size is readable from the rombanksz field in the MCFG1 register for the non-programmable modes. The PROM area will wrap back to the first bank after the end of the last decoded bank. As an example, if romasel is set to 14 the following banks will be decoded: bank 0: 0x00000000 - 0x07FFFFFFF bank 1: 0x08000000 - 0x0FFFFFFF bank 2: 0x10000000 - 0x17FFFFFFF bank 3: 0x18000000 - 0x1FFFFFFF ...bank 0 starting again at 0x20000000 (the same pattern applies for other values less than 14, addresses will wrap after the last decoded bank). If romasel is 15 then the address decoding will result in the following: bank 0: 0x00000000 - 0x0FFFFFFFF bank 1: 0x10000000 - 0x1FFFFFFF .. bank 0 starting again at offset 0x20000000 When instantiating the core care must be taken to see how many chip-selects that will be used as a result of the setting of romasel. This affects the base address at which the core can be placed (setting of romaddr and rommask VHDL generics). As an example, placing the PROM area at a 256 MiB address boundry, like the base address 0x10000000 and using romasel = 0, 14, 15 or 28 will NOT result in ROM chip-select 0 getting asserted for an access to the PROM base address as the address decoding requires that the core has been placed on a 512 MiB address boundary.	0 - 28	28
sdrasel	log2(RAM address space size) - 1. E.g if size of the RAM address space is 0x40000000 sdrasel is log2(2^30)-1= 29.	0 - 31	29
srbanks	Number of SRAM banks.	0 - 5	4
ram8	Enable 8-bit PROM, SRAM and I/O access.	0 - 1	0
ram16	Enable 16-bit PROM, SRAM and I/O access.	0 - 1	0
sden	Enable SDRAM controller.	0 - 1	0
sepbus	SDRAM is located on separate bus.	0 - 1	1
sdbits	32 or 64 -bit SDRAM data bus.	32, 64	32

*Table 270.*Configuration options

Generic	Function	Allowed range	Default
oepol	Select polarity of drive signals for data pads. 0 = active low, 1 = active high.	0 - 1	0
edac	Enable EDAC. 0 = No EDAC; 1 = BCH EDAC; 2 = BCH EDAC with pipelining; 3 = BCH + RS EDAC	0 - 3	0
sdlsb	Select least significant bit of the address bus that is connected to SDRAM.	-	2
syncrst	Choose between synchronous and asynchronous reset for chip-select, oen and drive signals.	0 - 1	0
pageburst	Line burst read of length 8 when 0, page burst read when 1, programmable read burst type when 2.	0-2	0
scantest	Enable scan test support	0 - 1	0
netlist	Use technology specific netlist instead of RTL code	0 - 1	0
tech	Technology to use for netlists	0 - NTECH	0
rahold	Add additional lead-out cycles for holding the address bus after PROM writes. This is used when a PROM device needs extra hold time on the address bus during write cycles.	0 - 16	0
wsshift	Wait state counter shift. This value defines the number of steps to shift the wait state counter. The number of waitstates that the core can generate is limited by $2^{wsshift}$. See the wait state fields in the core's APB register descriptions to see the effect of this generic. The value of this generic can be read out in the first user-defined register of the core's plug&play area. This means that if wsshift is non-zero then the AHB controller must have full plug&play decoding enabled.	-	0

28.18 Scan support

Scan support is enabled by setting the SCANTEST generic to 1. When enabled, the asynchronous reset of any flip-flop will be connected to AHBI.testrst during when AHBI.testen = ‘1’.

28.19 Signal descriptions

Table 271 shows the interface signals of the core (VHDL ports).

*Table 271.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
MEMI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Bus ready strobe	Low
	BEXCN	Input	Bus exception	Low
	CB[15:0]	Input	EDAC checkbits	High
	WRN[3:0]	Input	SRAM write enable feedback signal	Low
	BWIDTH[1:0]	Input	Sets the reset value of the PROM data bus width field in the MCFG1 register	High
	EDAC	Input	The reset value for the PROM EDAC enable bit	High
	SD[31:0]	Input	SDRAM separate data bus	High
	SCB[15:0]	Input	SDRAM separate checkbit bus	High

*Table 271.*Signal descriptions

Signal name	Field	Type	Function	Active
MEMO	ADDRESS[31:0]	Output	Memory address	High
	CB[15:0]	Output	EDAC Checkbit	
	DATA[31:0]	Output	Memory data	-
	SDDATA[63:0]	Output	Sdram memory data	-
	RAMSN[4:0]	Output	SRAM chip-select	Low
	RAMOEN[4:0]	Output	SRAM output enable	Low
	IOSN	Output	Local I/O select	Low
	ROMSN[3:0]	Output	PROM chip-select	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0]. Any WRN[ ] signal can be used for CB[ ].	Low
	MBEN[3:0]	Output	Read/write byte enable: MBEN[0] corresponds to DATA[31:24], MBEN[1] corresponds to DATA[23:16], MBEN[2] corresponds to DATA[15:8], MBEN[3] corresponds to DATA[7:0]. Any MBEN[ ] signal can be used for CB[ ].	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus. Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0]. Any BDRIVE[ ] signal can be used for CB[ ].	Low/High
	VBDRIVE[31:0]	Output	Vectored I/O-pad drive signals.	Low/High
	SVBDRIVE[63:0]	Output	Vectored I/O-pad drive signals for separate sdram bus.	Low/High
	READ	Output	Read strobe	High
	SA[14:0]	Output	SDRAM separate address bus	High
	CE	Output	Single error detected	High
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
WPROT	WPROTHIT	Input	Unused	-

*Table 271.*Signal descriptions

Signal name	Field	Type	Function	Active
SDO	SDCASN	Output	SDRAM column address strobe	Low
	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDDQM[7:0]	Output	SDRAM data mask: SDDQM[7] corresponds to SD[63:56], SDDQM[6] corresponds to SD[55:48], SDDQM[5] corresponds to SD[47:40], SDDQM[4] corresponds to SD[39:32], SDDQM[3] corresponds to SD[31:24], SDDQM[2] corresponds to SD[23:16], SDDQM[1] corresponds to SD[15:8], SDDQM[0] corresponds to SD[7:0]. Any SDDQM[ ] signal can be used for CB[ ].	Low
	SDRASN	Output	SDRAM row address strobe	Low
	SDWEN	Output	SDRAM write enable	Low

* see GRLIB IP Library User's Manual

28.20 Library dependencies

Table 272 shows libraries used when instantiating the core (VHDL libraries).

*Table 272.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals	Memory bus signals definitions
		Components	FTMCTRL component

28.21 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined on the example designs port map and connected to the memory controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

Memory controller decodes default memory areas: PROM area is 0x0 - 0x1FFFFFFF, I/O-area is 0x20000000-0x3FFFFFFF and RAM area is 0x40000000 - 0x7FFFFFFF. SDRAM controller is enabled. SDRAM clock is synchronized with system clock by clock generator.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;

    -- memory bus
    address  : out   std_logic_vector(27 downto 0); -- memory bus
    data     : inout std_logic_vector(31 downto 0);
    ramsn    : out   std_logic_vector(4 downto 0);
    ramoen   : out   std_logic_vector(4 downto 0);
    rwen     : inout std_logic_vector(3 downto 0);
    romsn    : out   std_logic_vector(3 downto 0);
    iosn     : out   std_logic;
    oen      : out   std_logic;
    read     : out   std_logic;
    writen   : inout std_logic;
    brdyn    : in    std_logic;
    bexcn    : in    std_logic;
-- sdram i/f
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo : sdram_out_type;

  signal wprot : wprot_out_type;  -- dummy signal, not used
  signal clkm, rstn : std_ulogic; -- system clock and reset

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;

  signal gnd : std_ulogic;

begin

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                               tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  -- Memory controller
  ftmctrl0 : ftmctrl generic map (srbanks => 1, sden => 1, edac => 1)
    port map (rstn, clkm, memi, memo, ahbsi, ahbso(0), apbi, apbo(0), wprot, sdo);

  -- memory controller inputs not used in this configuration
  memi.brdyn <= ’1’; memi.bexcn <= ’1’; memi.wrn <= "1111";
  memi.sd <= sd;
  -- prom width at reset
  memi.bwidth <= "10";

  -- I/O pads driving data memory bus data signals
  datapads : for i in 0 to 3 generate
      data_pad : iopadv generic map (width => 8)
      port map (pad => memi.data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(i),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  -- connect memory controller outputs to entity output signals
  address <= memo.address; ramsn <= memo.ramsn; romsn <= memo.romsn;
  oen <= memo.oen; rwen <= memo.wrn; ramoen <= "1111" & memo.ramoen(0);
  sa <= memo.sa;
  writen <= memo.writen; read <= memo.read; iosn <= memo.iosn;
  sdcke <= sdo.sdcke; sdwen <= sdo.sdwen; sdcsn <= sdo.sdcsn;
  sdrasn <= sdo.rasn; sdcasn <= sdo.casn; sddqm <= sdo.dqm;
end;
```

29 FTSDCTRL - 32/64-bit PC133 SDRAM Controller with EDAC

29.1 Overview

The fault tolerant SDRAM memory interface handles PC133 SDRAM compatible memory devices attached to a 32- or 64-bit wide data bus. The interface acts as a slave on the AHB bus where it occupies configurable amount of address space for SDRAM access. An optional Error Detection And Correction Unit (EDAC) logic (only for the 32 - bit bus) corrects one bit error and detects two bit errors.

The SDRAM controller function is programmed by means of register(s) mapped into AHB I/O address space. Chip-select decoding is done for two SDRAM banks.

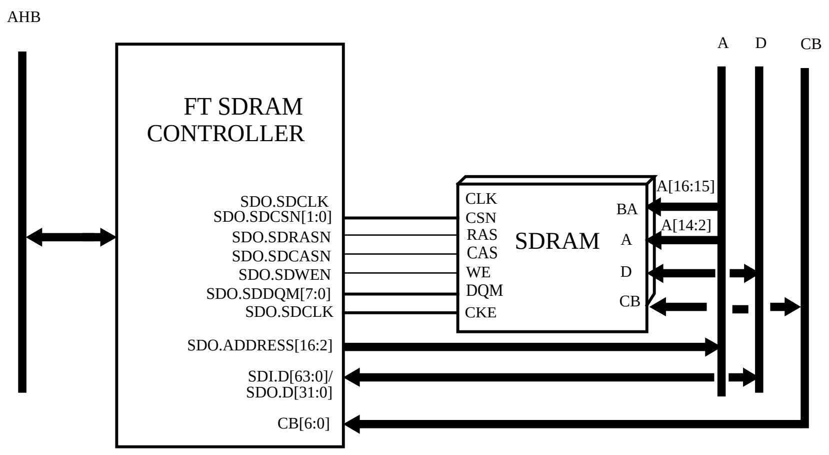


Figure 101. FT SDRAM memory controller connected to AMBA bus and SDRAM

29.2 Operation

29.2.1 General

Synchronous Dynamic RAM (SDRAM) access is supported to two banks of PC100/PC133 compatible devices. The controller supports 64, 256 and 512 Mbyte devices with 8 - 12 column-address bits, up to 13 row-address bits, and 4 banks. The size of each of the two banks can be programmed in binary steps between 4 Mbyte and 512 Mbyte. The operation of the SDRAM controller is controlled through the configuration register SDCFG. A second register, ECFG, is available for configuring the EDAC functions. SDRAM banks data bus width is configurable between 32 and 64 bits.

29.2.2 Initialisation

When the SDRAM controller is enabled, it automatically performs the SDRAM initialisation sequence of PRECHARGE, 8x AUTO-REFRESH and LOAD-MODE-REG on both banks simultaneously. The controller programs the SDRAM to use page burst on read and single location access on write.

29.2.3 Configurable SDRAM timing parameters

To provide optimum access cycles for different SDRAM devices (and at different frequencies), some SDRAM parameters can be programmed through SDRAM configuration register (SDCFG) The programmable SDRAM parameters can be seen in table below:

*Table 273.*SDRAM programmable timing parameters

Function	**Parameter**	**range**	**unit**
CAS latency, RAS/CAS delay	t_{CAS}, t_{RCD}	2 - 3	clocks
Precharge to activate	t_{RP}	2 - 3	clocks
Auto-refresh command period	t_{RFC}	3 - 11	clocks
Auto-refresh interval		10 - 32768	clocks

Remaining SDRAM timing parameters are according the PC100/PC133 specification.

29.2.4 Refresh

The SDRAM controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the SDCFG register. Depending on SDRAM type, the required period is typically 7.8 or 15.6 µs (corresponding to 780 or 1560 clocks at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by setting bit 31 in SDCFG register.

29.2.5 SDRAM commands

The controller can issue three SDRAM commands by writing to the SDRAM command field in SDCFG: PRE-CHARGE, AUTO-REFRESH and LOAD-MODE-REG (LMR). If the LMR command is issued, the CAS delay as programmed in SDCFG will be used, remaining fields are fixed: page read burst, single location write, sequential burst. The command field will be cleared after a command has been executed. Note that when changing the value of the CAS delay, a LOAD-MODE-REGISTER command should be generated at the same time.

29.2.6 Read cycles

A read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command after the programmed CAS delay. A read burst is performed if a burst access has been requested on the AHB bus. The read cycle is terminated with a PRE-CHARGE command, no banks are left open between two accesses. Note that only word bursts are supported by the SDRAM controller. The AHB bus supports bursts of different sizes such as bytes and halfwords but they cannot be used.

29.2.7 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. A write burst on the AHB bus will generate a burst of write commands without idle cycles in-between. As in the read case, only word bursts are supported.

29.2.8 Address bus connection

The SDRAM address bus should be connected to SA[12:0], the bank address to SA[14:13], and the data bus to SD[31:0] or SD[63:0] if 64-bit data bus is used.

29.2.9 Data bus

Data bus width is configurable to 32 or 64 bits. 64-bit data bus allows the 64-bit SDRAM devices to be connected using the full data capacity of the devices. 64-bit SDRAM devices can be connected to 32-bit data bus if 64-bit data bus is not available but in this case only half the full data capacity will be used.

29.2.10 Clocking

The SDRAM clock typically requires special synchronisation at layout level. For Virtex targets, GR Clock Generator can be configured to produce a properly synchronised SDRAM clock. For other FPGA targets, the GR Clock Generator can produce an inverted clock.

29.2.11 EDAC

The controller optionally contains Error Detection And Correction (EDAC) logic, using a BCH(32, 7) code. It is capable of correcting one bit error and detecting two bit errors. The EDAC logic does not add any additional waitstates during normal operation. Detected errors will cause additional waitstates for correction (single errors) or error reporting (multiple errors). Single errors are automatically corrected and generally not visible externally unless explicitly checked.

This checking is done by monitoring the ce signal and single error counter. This counter holds the number of detected single errors. The ce signal is asserted one clock cycle when a single error is detected and should be connected to the AHB status register. This module stores the AHB status of the instruction causing the single error and generates interrupts (see the AHB status register documentation for more information).

The EDAC functionality can be enabled/disabled during run-time from the ECFG register (and the logic can also be completely removed during synthesis with VHDL generics. The ECFG register also contains control bits and checkbit fields for diagnostic reads. These diagnostic functions are used for testing the EDAC functions on-chip and allows one to store arbitrary checkbits with each written word. Checkbits read from memory can also be controlled.

64-bit bus support is not provided when EDAC is enabled. Thus, the sd64 and edacen VHDL generics should never be set to one at the same time.

The equations below show how the EDAC checkbits are generated:

`CB0 = D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14 ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31`

`CB1 = D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12 ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28`

$\overline{\text{CB2}}$ `= D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15 ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31`

$\overline{\text{CB3}}$ `= D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13 ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29`

`CB4 = D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15 ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31`

`CB5 = D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31`

`CB6 = D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31`

29.3 Registers

The memory controller is programmed through register(s) mapped into the AHB I/O space defined by the controllers AHB BAR1.

If EDAC is enabled through the use of the edacen VHDL generic, an EDAC configuration register will be available.

*Table 274.*FT SDRAM controller registers

AHB address offset	**Register**
0x0	SDRAM Configuration register
0x4	EDAC Configuration register

29.3.1 SDRAM configuration register (SDCFG)

SDRAM configuration register is used to control the timing of the SDRAM.

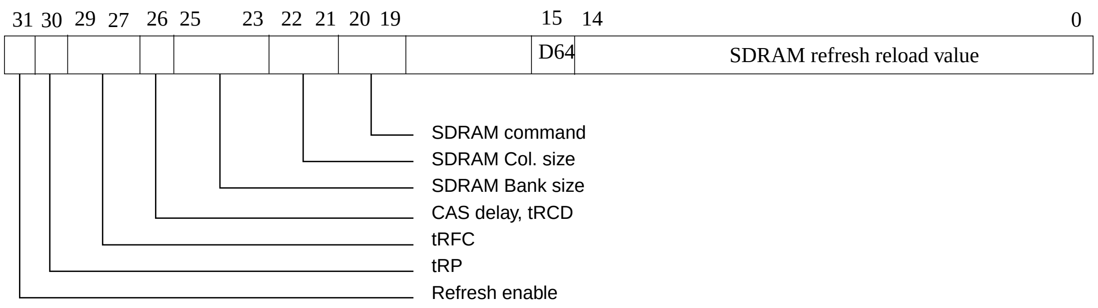


Figure 102. SDRAM configuration register

[14:0]: The period between each AUTO-REFRESH command - Calculated as follows:$t_{REFRESH}$ = ((reload value) + 1) / SYSCLK

[15]: 64-bit data bus (D64) - Reads ‘1’ if memory controller is configured for 64-bit data bus, otherwise ‘0’. Read-only.

[20:19]: SDRAM command. Writing a non-zero value will generate an SDRAM command: “01”=PRECHARGE, “10”=AUTO-REFRESH, “11”=LOAD-COMMAND-REGISTER. The field is reset after command has been executed.

[22:21]: SDRAM column size. “00”=256, “01”=512, “10”=1024, “11”=4096 when bit[25:23]= “111”, 2048 otherwise.

[25:23]: SDRAM banks size. Defines the banks size for SDRAM chip selects: “000”=4 Mbyte, “001”=8 Mbyte, “010”=16 Mbyte .... “111”=512 Mbyte.

[26]: SDRAM CAS delay. Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-COMMAND-REGISTER command must be issued at the same time. Also sets RAS/CAS delay (tRCD).

[29:27]: SDRAM t_{RFC} timing. t_{RFC} will be equal to 3 + field-value system clocks.

[30]: SDRAM t_{RP} timing. t_{RP} will be equal to 2 or 3 system clocks (0/1).

[31]: SDRAM refresh. If set, the SDRAM refresh will be enabled.

29.3.2 EDAC Configuration register (ECFG)

The EDAC configuration register controls the EDAC functions of the SDRAM controller during run time.

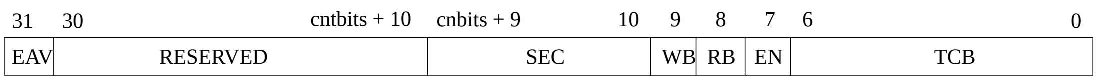


Figure 103. EDAC configuration register

[6:0] TCB : Test checkbits. These bits are written as checkbits into memory during a write operation when the WB bit in the ECFG register is set. Checkbits read from memory during a read operation are written to this field when the RB bit is set.

[7] EN : EDAC enable. Run time enable/disable of the EDAC functions. If EDAC is disabled no error detection will be done during reads and subword writes. Checkbits will still be written to memory during write operations.

[8] RB : Read bypass. Store the checkbits read from memory during a read operation into the TCB field.

[9] WB : Write bypass. Write the TCB field as checkbits into memory for all write operations.

[cntbits + 9:10] SEC : Single error counter. This field is available when the errcnt VHDL generic is set to one during synthesis. It increments each time a single error is detected. It saturates when the maximum value is reached. The maximum value is the largest number representable in the number of bits used, which in turn is determined by the cntbits VHDL generic. Each bit in the counter can be reset by writing a one to it.

[30:cntbits + 10] Reserved.

[31] EAV : EDAC available. This bit is always one if the SDRAM controller contains EDAC.

29.4 Vendor and device identifiers

The module has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x055. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

29.5 Configuration options

Table 275 shows the configuration options of the core (VHDL generics).

*Table 275.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where SDCFG register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#FFF#
wprot	Write protection.	0 - 1	0
invclk	Inverted clock is used for the SDRAM.	0 - 1	0
fast	Enable fast SDRAM address decoding.	0 - 1	0
pwron	Enable SDRAM at power-on.	0 - 1	0
sdbits	32 or 64 -bit data bus width.	32, 64	32
edacen	EDAC enable. If set to one, EDAC logic will be included in the synthesized design. An EDAC configuration register will also be available.	0 - 1	0
errcnt	Include an single error counter which is accessible from the EDAC configuration register.	0 - 1	0
cntbits	Number of bits used in the single error counter	1 - 8	1

29.6 Signal descriptions

Table 276 shows the interface signals of the core (VHDL ports).

*Table 276.*Signals declarations

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SDI	WPROT	Input	Not used	-
	DATA[63:0]	Input	Data	-
	CB[7:0]	Input	Checkbits	-
SDO	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDWEN	Output	SDRAM write enable	Low
	RASN	Output	SDRAM row address strobe	Low
	CASN	Output	SDRAM column address strobe	Low
	DQM[7:0]	Output	SDRAM data mask: DQM[7] corresponds to DATA[63:56], DQM[6] corresponds to DATA[55:48], DQM[5] corresponds to DATA[47:40], DQM[4] corresponds to DATA[39:32], DQM[3] corresponds to DATA[31:24], DQM[2] corresponds to DATA[23:16], DQM[1] corresponds to DATA[15:8], DQM[0] corresponds to DATA[7:0]. Any DQM[ ] signal can be used for CB[ ].	Low
	BDRIVE	Output	Drive SDRAM data bus	Low
	ADDRESS[16:2]	Output	SDRAM address	-
	DATA[31:0]	Output	SDRAM data	-
	CB[7:0]	Output	Checkbits	-
CE	N/A	Output	Correctable Error	High

* see GRLIB IP Library User’s Manual

29.7 Library dependencies

Table 5 shows libraries used when instantiating the core (VHDL libraries).

*Table 277.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

29.8 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the FT SDRAM controller. The external SDRAM bus is defined in the example designs port map and connected to the SDRAM controller. System clock and reset are generated by GR Clock Generator and Reset Generator. It is also shown how the correctable error (CE) signal is connected to the ahb status register. It is not mandatory to connect this signal. In this example, 3 units can be connected to the status register.

The SDRAM controller decodes SDRAM area: 0x60000000 - 0x6FFFFFFF. SDRAM Configuration and EDAC configuration registers are mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;
    ... -- other signals

-- sdram memory bus
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0); -- optional sdram data
    cb       : inout std_logic_vector(7 downto 0) --EDAC checkbits
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect SDRAM controller and SDRAM memory bus
  signal sdi   : sdctrl_in_type;
  signal sdo   : sdctrl_out_type;

  signal clkm, rstn : std_ulogic; -- system clock and reset
signal ce : std_logic_vector(0 to 2); --correctable error signal vector

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;

  signal gnd : std_ulogic;

begin

  -- AMBA Components are defined here ...
  ...

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                                tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);

  -- AHB Status Register
  astat0 : ahbstat generic map(pindex => 13, paddr => 13, pirq => 11,
    nftslv => 3)
    port map(rstn, clkm, ahbmi, ahbsi, ce, apbi, apbo(13));

  -- SDRAM controller
  sdc : ftsdctrl generic map (hindex => 3, haddr => 16#600#, hmask => 16#F00#,
    ioaddr => 1, fast => 0, pwron => 1, invclk => 0, edacen => 1, errcnt => 1,
    cntbits => 4)
    port map (rstn, clkm, ahbsi, ahbso(3), sdi, sdo, ce(0));

  -- input signals
  sdi.data(31 downto 0) <= sd(31 downto 0);

  -- connect SDRAM controller outputs to entity output signals
  sa <= sdo.address; sdcke <= sdo.sdcke; sdwen <= sdo.sdwen;
  sdcsn <= sdo.sdcsn; sdrasn <= sdo.rasn; sdcasn <= sdo.casn;
  sddqm <= sdo.dqm;

  -- I/O pads driving data bus signals
  sd_pad : iopadv generic map (width => 32)
      port map (sd(31 downto 0), sdo.data, sdo.bdrive, sdi.data(31 downto 0));

  -- I/O pads driving checkbit signals
  cb_pad : iopadv generic map (width => 8)
      port map (cb, sdo.cb, sdo.bdrive, sdi.cb);


end;
```

30 FTSDCTRL64 - 64-bit PC133 SDRAM Controller with EDAC

30.1 Overview

The SDRAM controller handles PC133 SDRAM compatible memory devices attached to a 64 bit wide data bus. The controller acts as a slave on the AHB bus where it occupies a configurable amount of address space for SDRAM access. Error correction is optionally implemented using BCH or Reed-Solomon codes. The SDRAM controller function is programmed by writing to configuration registers mapped into AHB I/O address space. Chip-select decoding is provided for two SDRAM banks.

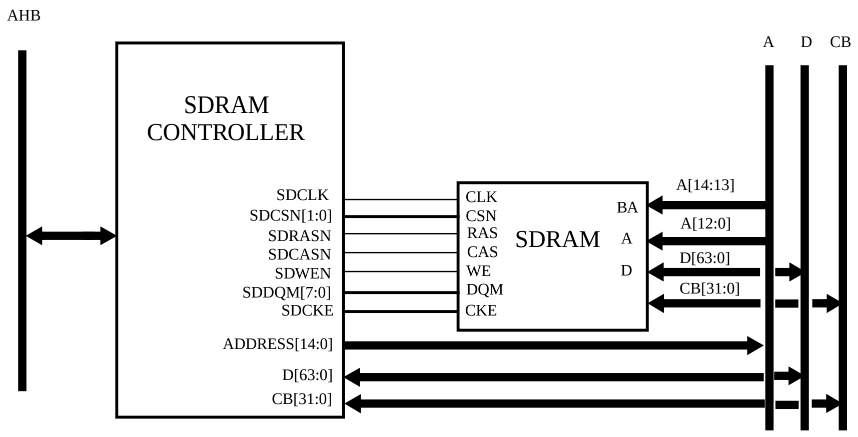


Figure 104. SDRAM Memory controller connected to AMBA bus and SDRAM

30.2 Operation

30.2.1 General

Synchronous dynamic RAM (SDRAM) access is supported to two memory banks of PC100/PC133 compatible devices. The controller supports 64M, 256M and 512M devices with 8 - 12 column-address bits, up to 13 row-address bits, and 4 internal banks. The size of each of the two memory banks can be programmed in binary steps between 4 Mbyte and 512 Mbyte. The operation of the SDRAM controller is controlled through four configuration registers (see section 30.3). The FTSDCTRL64 controller also supports mobile SDRAM if required.

30.2.2 Initialization

When the SDRAM controller is enabled, it automatically performs the SDRAM initialization sequence of PRECHARGE, 8x AUTO-REFRESH and LOAD-MODE-REG on both banks simultaneously. When mobile SDRAM functionality is enabled, the initialization sequence is appended with a LOAD-EXTMODE-REG command. The controller programs the SDRAM to use page burst on read accesses and single location access on write accesses. If the *pwron* VHDL generic is 1, the initialization sequence is also sent automatically when reset is released. Note that some SDRAM devices require a stable clock of 100 us before any commands might be sent. When using on-chip PLL, this might not always be the case and the *pwron* VHDL generic should be set to 0 in such cases.

30.2.3 Configurable SDRAM timing parameters

To provide optimum access cycles for different SDRAM devices (and at different frequencies), three SDRAM parameters can be programmed through memory configuration register 2 (MCFG2): TCAS, TRP and TRFCD. The value of these fields affect the SDRAM timing as described in table 278.

*Table 278.*SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
CAS latency, RAS/CAS delay (t_{CAS}, t_{RCD})	TCAS + 2
Precharge to activate (t_{RP})	TRP + 2
Auto-refresh command period (t_{RFC})	TRFC + 3
Activate to precharge (t_{RAS})	TRFC + 1
Activate to Activate (t_{RC})	TRP + TRFC + 4

If the TCAS, TRP and TRFC are programmed such that the PC100/133 specifications are fulfilled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 100 and 133 MHz operation and the resulting SDRAM timing (in ns):

*Table 279.*SDRAM example programming

SDRAM settings	t_{CAS}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
100 MHz, CL=2; TRP=0, TCAS=0, TRFC=4	20	80	20	70	50
100 MHz, CL=3; TRP=0, TCAS=1, TRFC=4	30	80	20	70	50
133 MHz, CL=2; TRP=1, TCAS=0, TRFC=6	15	82	22	67	52
133 MHz, CL=3; TRP=1, TCAS=1, TRFC=6	22	82	22	67	52

When mobile SDRAM support is enabled, one additional timing parameter (TXSR) can be programmed though the Power-Saving configuration register.

*Table 280.*Mobile SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
Exit Self Refresh mode to first valid command (t_{XSR})	tXSR

30.2.4 Refresh

The SDRAM controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the SDCFG register. Depending on SDRAM type, the required period is typically 7.8 or 15.6 µs (corresponding to 780 or 1560 clocks at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by setting bit 31 in SDCFG register.

30.2.5 Self Refresh

The self refresh mode can be used to retain data in the SDRAM even when the rest of the system is powered down. When in the self refresh mode, the SDRAM retains data without external clocking and refresh are handled internally. The memory array that is refreshed during the self refresh operation is defined in the extended mode register. These settings can be changed by setting the PASR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the PASR bits are changed. The supported “Partial Array Self Refresh” modes are: Full, Half, Quarter, Eighth, and Sixteenth array. “Partial Array Self Refresh” is only supported when mobile SDRAM functionality is enabled. To enable the self refresh mode, set the PMODE bits in the Power-Saving configuration register to “010” (Self Refresh). The controller will enter self refresh mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. When exiting this mode the controller introduce a delay defined by tXSR in the Power-Saving configuration register and a AUTO REFRESH command before any other memory access is allowed. The minimum duration of this mode is defined by tRAS. This mode is only available when the VHDL generic *mobile* is >= 1.

30.2.6 Power-Down

When entering the power-down mode all input and output buffers, excluding SDCKE, are deactivated. All data in the SDRAM is retained during this operation. To enable the power-down mode, set the PMODE bits in the Power-Saving configuration register to “001” (Power-Down). The controller will enter power-down mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits is cleared. The REFRESH command will still be issued by the controller in this mode. When exiting this mode a delay of one clock cycles are added before issue any command to the memory. This mode is only available when the VHDL generic *mobile* is >= 1.

30.2.7 Deep Power-Down

The deep power-down operating mode is used to achieve maximum power reduction by eliminating the power of the memory array. Data will not be retained after the device enters deep power-down mode. To enable the deep power-down mode, set the PMODE bits in the Power-Saving configuration register to “101” (Deep Power-Down). To exit the deep power-down mode the PMODE bits in the Power-Saving configuration register must be cleared. The controller will respond with an AMBA ERROR response to an AMBA access, that will result in a memory access, during Deep Power-Down mode. This mode is only available when the VHDL generic *mobile* is >= 1 and mobile SDRAM functionality is enabled.

30.2.8 Temperature-Compensated Self Refresh

The settings for the temperature-compensation of the Self Refresh rate can be controlled by setting the TCSR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the TCSR bits are changed. Note that some vendors implements a Internal Temperature-Compensated Self Refresh feature, which makes the memory ignore the TCSR bits. This functionality is only available when the VHDL generic *mobile* is >= 1 and mobile SDRAM functionality is enabled.

30.2.9 Drive Strength

The drive strength of the output buffers can be controlled by setting the DS bits in the Power-Saving configuration register. The extended mode register is automatically updated when the DS bits are changed. The available options are: full, three-quarter, one-half, and one-quarter drive strengths. This functionality is only available when the VHDL generic *mobile* is >= 1 and mobile SDRAM functionality is enabled.

30.2.10 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in the SDRAM Configuration register: PRE-CHARGE, AUTO-REFRESH, LOAD-MODE-REG (LMR) and LOAD-EXTMODE-REG (EMR). If the LMR command is issued, the CAS delay as programmed in SDCFG will be used, remaining fields are fixed: page read burst, single location write, sequential burst. If the EMR command is issued, the DS, TCSR and PASR as programmed in Power-Saving configuration register will be used. The command field will be cleared after a command has been executed. Note that when changing the value of the CAS delay, a LOAD-MODE-REGISTER command should be generated at the same time.

30.2.11 Read cycles

A read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command with data read after the programmed CAS delay. A read burst is performed if a burst access has been requested on the AHB bus. The read cycle is terminated with a PRE-CHARGE command, no banks are left open between two accesses. Note that only 64-bit AHB bursts are supported by the SDRAM controller. The AHB bus supports bursts of different sizes such as bytes and half-words but they cannot be used.

30.2.12 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. A write burst on the AHB bus will generate a burst of write commands without idle cycles in-between. As in the read case, only 64-bit bursts are supported.

30.2.13 Address bus connection

The SDRAM address bus should be connected to SA[12:0], the bank address to SA[14:13].

30.2.14 Data bus

The external SDRAM data bus should be connected to SD[63:0]. The polarity of the output enable signal to the data pads can be selected with the oepol generic. Sometimes it is difficult to fulfil the output delay requirements of the output enable signal. In this case, the vbdrive signal can be used instead of bdrive. Each bit in this vector is driven by a separate register.

30.2.15 Clocking

The SDRAM controller is designed for an external SDRAM clock that is in phase or slightly earlier than the internal AHB clock. This provides the maximum margin for setup and hold on the external signals, and allows highest possible frequency. For Xilinx and Altera devices, the GRLIB Clock Generator (CLKGEN) can be configured to produce a properly synchronized SDRAM clock. For other FPGA targets, the custom clock synchronization must be designed. For ASIC targets, the SDRAM clock can be derived from the AHB clock with proper delay adjustments during place&route.

30.2.16 EDAC

The controller optionally contains Error Detection And Correction (EDAC) logic, using a BCH(64, 8) or a Reed-Solomon (64, 32) code. The BCH code It is capable of correcting one bit error and detecting two bit errors, while the RS code can correct four nibble errors. Correctable errors are automatically corrected and generally not visible externally unless explicitly checked. This checking is done by monitoring the ce signal and single error counter. This counter holds the number of detected single errors. The ce signal is asserted one clock cycle when a single error is detected and should be connected to the AHB status register. This module stores the AHB status of the instruction causing the single error and generates interrupts (see the AHB status register documentation for more information).

The EDAC functionality can be enabled/disabled during run-time from the EDAC configuration register (and the logic can also be completely removed during synthesis with VHDL generics). The EDAC checkbits register also contains checkbit fields for diagnostic reads and writes. These diagnostic functions are used for testing the EDAC functions on-chip and allows one to store arbitrary checkbits with each written word. Checkbits read from memory can also be controlled.

30.3 Registers

The memory controller is programmed through register(s) mapped into the AHB I/O space defined by the controllers AHB BAR1.

*Table 281.*SDRAM controller registers

AHB address offset	Register
0x0	SDRAM Configuration register
0x4	SDRAM Power-Saving configuration register
0x8	EDAC Configuration register
0xC	EDAC checkbits registers

Table 282. SDRAM configuration register

31	30	29 27	26	25 23	22 21	20 18	17	16	15	14 0
Refresh	tRP	tRFC	tCD	SDRAM bank size	SDRAM col. size	SDRAM command	Page-Burst	MS	D64	SDRAM refresh load value

31	SDRAM refresh. If set, the SDRAM refresh will be enabled.
30	SDRAM tRP timing. tRP will be equal to 2 or 3 system clocks (0/1). When mobile SDRAM support is enabled, this bit also represent the MSB in the tRFC timing.
29: 27	SDRAM tRFC timing. tRFC will be equal to 3 + field-value system clocks. When mobile SDRAM support is enabled, this field is extended with the bit 30.
26	SDRAM CAS delay. Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-COMMAND-REGISTER command must be issued at the same time. Also sets RAS/CAS delay (tRCD).
25: 23	SDRAM banks size. Defines the decoded memory size for each SDRAM chip select: “000”= 4 Mbyte, “001”= 8 Mbyte, “010”= 16 Mbyte .... “111”= 512 Mbyte.
22: 21	SDRAM column size. “00”=256, “01”=512, “10”=1024, “11”=4096 when bit[25:23]= “111”, 2048 otherwise.
20: 18	SDRAM command. Writing a non-zero value will generate an SDRAM command: “010”=PRE-CHARGE, “100”=AUTO-REFRESH, “110”=LOAD-COMMAND-REGISTER, “111”=LOAD-EXTENDED-COMMAND-REGISTER. The field is reset after command has been executed.
17	1 = pageburst is used for read operations, 0 = line burst of length 8 is used for read operations. (Only available when VHDL generic pageburst i set to 2)
16	Mobile SDR support enabled. ‘1’ = Enabled, ‘0’ = Disabled (read-only)
15	64-bit data bus (D64) - Reads ‘1’ to indicate 64-bit data bus. Read-only.
14: 0	The period between each AUTO-REFRESH command - Calculated as follows: tREFRESH = ((reload value) + 1) / SYSCLK

Table 283. SDRAM Power-Saving configuration register

31	30	29 24	23 20	19	18 16	15 7	6 5	4 3	2 0
ME	CE	Reserved	tXSR	res	PMODE	Reserved	DS	TCSR	PASR

31	Mobile SDRAM functionality enabled. ‘1’ = Enabled (support for Mobile SDRAM), ‘0’ = disabled (support for standard SDRAM)
30	Clock enable (CE). This value is driven on the CKE inputs of the SDRAM. Should be set to ‘1’ for correct operation. This register bit is read only when Power-Saving mode is other then none.
29: 24	Reserved
23: 20	SDRAM tXSR timing. tXSR will be equal to field-value system clocks. (Read only when Mobile SDR support is disabled).
19	Reserved

Table 283. SDRAM Power-Saving configuration register

18: 16	Power-Saving mode (Read only when Mobile SDR support is disabled). “000”: none “001”: Power-Down (PD) “010”: Self-Refresh (SR) “101”: Deep Power-Down (DPD)
15: 7	Reserved
6: 5	Selectable output drive strength (Read only when Mobile SDR support is disabled). “00”: Full “01”: One-half “10”: One-quarter “11”: Three-quarter
4: 3	Reserved for Temperature-Compensated Self Refresh (Read only when Mobile SDR support is disabled). “00”: 70ªC “01”: 45ªC “10”: 15ªC “11”: 85ªC
2: 0	Partial Array Self Refresh (Read only when Mobile SDR support is disabled). “000”: Full array (Banks 0, 1, 2 and 3) “001”: Half array (Banks 0 and 1) “010”: Quarter array (Bank 0) “101”: One-eighth array (Bank 0 with row MSB = 0) “110”: One-sixteenth array (Bank 0 with row MSB = 00)

Table 284. EDAC Configuration register

31 30 29	25 24	23 22 21	20	19	18	17	16	15 7 6 5 4 3 2 0
Reserved	SEC	Reserved	ED	RS	WB	RB	EN	Reserved

25: 24	Single error counter. This field is increments each time a single error is detected. It saturates when the maximum value is reached (3).
20	Disable EDAC checking. EDAC errors will be ignored if set to 1
19	Reed-Solomon enable. Set to 1 to enable RS coding instead of BCH.
18	Write bypass. Write the EDAC checkbits register as checkbits into memory for all write operations.
17	Read bypass. Store the checkbits read from memory during a read operation into the EDAC checkbits register.
16	EDAC enable. Set to 1 to enable EDAC error detection and correction.

Table 285. EDAC checkbits register

31 0
EDAC Test Checkbits

31 0	Checkbits for diagnostic read/write

30.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x058. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

30.5 Configuration options

Table 286 shows the configuration options of the core (VHDL generics).

*Table 286.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where SDCFG register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#FFF#
wprot	Write protection.	0 - 1	0
invclk	Inverted clock is used for the SDRAM.	0 - 1	0
pwron	Enable SDRAM at power-on initialization	0 - 1	0
sdbits	32 or 64-bit data bus width.	32, 64	32
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
pageburst	Enable SDRAM page burst operation. 0: Controller uses line burst of length 8 for read operations. 1: Controller uses pageburst for read operations. 2: Controller uses pageburst/line burst depending on PageBurst bit in SDRAM configuration register.	0 - 2	0
mobile	Enable Mobile SDRAM support 0: Mobile SDRAM support disabled 1: Mobile SDRAM support enabled but not default 2: Mobile SDRAM support enabled by default 3: Mobile SDRAM support only (no regular SDR support)	0 - 3	0
edac	Enable EDAC 0: No EDAC 1: BCH EDAC 2: RS EDAC 3: BCH and RS EDAC	0 - 3	0

30.6 Signal descriptions

Table 287 shows the interface signals of the core (VHDL ports).

*Table 287.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
AHBSI	1)	Input	AHB slave input signals	-
AHBSO	1)	Output	AHB slave output signals	-
SDI	WPROT	Input	Not used	-
	DATA[63:0]	Input	Data	High
SDO	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDWEN	Output	SDRAM write enable	Low
	RASN	Output	SDRAM row address strobe	Low
	CASN	Output	SDRAM column address strobe	Low
	DQM[7:0]	Output	SDRAM data mask: DQM[7] corresponds to DATA[63:56], DQM[6] corresponds to DATA[55:48], DQM[5] corresponds to DATA[47:40], DQM[4] corresponds to DATA[39:32], DQM[3] corresponds to DATA[31:24], DQM[2] corresponds to DATA[23:16], DQM[1] corresponds to DATA[15:8], DQM[0] corresponds to DATA[7:0].	Low
	BDRIVE	Output	Drive SDRAM data bus	Low/High^2
	VBDRIVE[63:0]	Output	Identical to BDRIVE but has one signal for each data bit. Every index is driven by its own register. This can be used to reduce the output delay.	Low/High^2
	VCBDRIVE[31:0]	Output	Identical to BDRIVE but has one signal for each check bit. Every index is driven by its own register. This can be used to reduce the output delay.	Low/High^2
	ADDRESS[14:0]	Output	SDRAM address	-
	DATA[63:0]	Output	SDRAM data	-
	CB[31:0]	Outputs	EDAC checkbits. BCH uses [7:0] only.	

1) see GRLIB IP Library User’s Manual

2) Polarity selected with the oepol generic

30.7 Library dependencies

Table 288 shows libraries used when instantiating the core (VHDL libraries).

*Table 288.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

30.8 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the SDRAM controller. The external SDRAM bus is defined on the example designs port map and connected to the SDRAM controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

SDRAM controller decodes SDRAM area:0x60000000 - 0x6FFFFFFF. SDRAM Configuration register is mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;
sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

signal sdi   : sdctrl_in_type;
  signal sdo   : sdctrl_out_type;

  signal clkm, rstn : std_ulogic;
signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;
  signal gnd : std_ulogic;

begin

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                              tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);
  -- SDRAM controller
  sdc : ftsdctrl64 generic map (hindex => 3, haddr => 16#600#, hmask => 16#F00#,
    ioaddr => 1, pwron => 0, invclk => 0)
    port map (rstn, clkm, ahbsi, ahbso(3), sdi, sdo);

  -- connect SDRAM controller outputs to entity output signals
  sa <= sdo.address; sdcke <= sdo.sdcke; sdwen <= sdo.sdwen;
  sdcsn <= sdo.sdcsn; sdrasn <= sdo.rasn; sdcasn <= sdo.casn;
  sddqm <= sdo.dqm;

--Data pad instantiation with scalar bdrive
sd_pad : iopadv generic map (width => 32)
port map (sd(63 downto 0), sdo.data(63 downto 0), sdo.bdrive, sdi.data(63 downto 0));
end;

--Alternative data pad instantiation with vectored bdrive
sd_pad : iopadvv generic map (width => 32)
port map (sd(63 downto 0), sdo.data(63 downto 0), sdo.vbdrive(63 downto 0), sdi.data(63
downto 0));
end;
```

31 FTSRCTRL - Fault Tolerant 32-bit PROM/SRAM/IO Controller

31.1 Overview

The fault tolerant 32-bit PROM/SRAM memory interface uses a common 32-bit memory bus to interface PROM, SRAM and I/O devices. Support for 8-bit PROM banks can also be separately enabled. In addition it also provides an Error Detection And Correction Unit (EDAC), correcting one and detecting two errors. Configuration of the memory controller functions is performed through the APB bus interface.

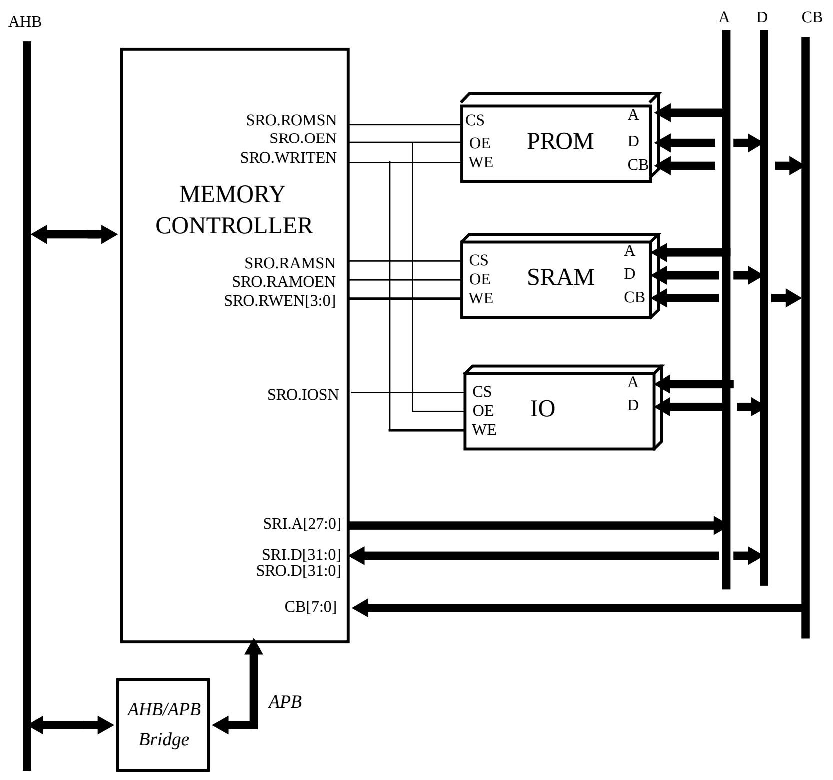


Figure 105. 32-bit FT PROM/SRAM/IO controller

31.2 Operation

The controller is configured through VHDL generics to decode three address ranges: PROM, SRAM and I/O area. By default the PROM area is mapped into address range 0x0 - 0x00FFFFFF, the SRAM area is mapped into address range 0x40000000 - 0x40FFFFFF, and the I/O area is mapped to 0x20000000 - 0x20FFFFFF.

One chip select is decoded for the I/O area, while SRAM and PROM can have up to 8 chip select signals. The controller generates both a common write-enable signal (WRITEN) as well as four byte-write enable signals (WREN). If the SRAM uses a common write enable signal the controller can be configured to perform read-modify-write cycles for byte and half-word write accesses. Number of waitstates is separately configurable for the three address ranges.

The EDAC function is optional, and can be enabled with the *edacen* VHDL generic. The configuration of the EDAC is done through a configuration register accessed from the APB bus. During nominal operation, the EDAC checksum is generated and checked automatically. Single errors are corrected without generating any indication of this condition in the bus response. If a multiple error is detected, a two cycle error response is given on the AHB bus.

Single errors can be monitored in two ways:

- by monitoring the CE signal which is asserted for one cycle each time a single error is detected.
- by checking the single error counter which is accessed from the MCFG3 configuration register.

The CE signal can be connected to the AHB status register which stores information of the AHB instruction causing the error and also generates interrupts. See the AHB status register documentation for more information. When EDAC is enabled, one extra latency cycle is generated during reads and subword writes.

The EDAC function can be enabled for SRAM and PROM area accesses, but not for I/O area accesses. For the SRAM area, the EDAC functionality is only supported for accessing 32-bit wide SRAM banks. For the PROM area, the EDAC functionality is supported for accessing 32-bit wide PROM banks, as well as for read accesses to 8-bit wide PROM banks.

The equations below show how the EDAC checkbits are generated:

```
CB0 = D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14 ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31
CB1 = D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12 ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28
C̅B̅2̅ = D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15 ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31
C̅B̅3̅ = D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13 ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29
CB4 = D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15 ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31
CB5 = D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
CB6 = D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
```

31.2.1 8-bit PROM access

The FTSRCTRL controller can be configured to access an 8-bit wide PROM. The data bus of the external PROM should be connected to the upper byte of the 32-bit data bus, i.e. D[31:24]. The 8-bit mode is enabled with the prom8en VHDL generic. When enabled, read accesses to the PROM area will be done in four-byte bursts for all 32-, 16- and 8-bit AMBA AHB accesses. The whole 32-bit word is then output on the AHB data bus, allowing the master to chose the bytes needed (big-endian).

Writes should be done one byte at a time. For correct word aligned 32-bit word write accesses, the byte should always be driven on bits 31 to 24 on the AHB data bus. For non-aligned 32-bit word write accesses, the byte should be driven on the bits of the AHB data bus that correspond to the byte address (big-endian). For correct half-word aligned 16-bit half-word write accesses, the byte should always be driven on bits 31 to 24, or 15 to 8, on the AHB data bus. For non-aligned 16-bit half-word write accesses, the byte should be driven on the bits of the AHB data bus that correspond to the byte address (big-endian). For 8-bit word write accesses the byte should always be driven on the AHB data bus bits that corresponds to the byte address (big-endian). To summarize, all legal AMBA AHB write accesses are supported according to the AMBA standard, additional illegal accesses are supported as described above, and it is always the addressed byte that is output.

It is possible to dynamically switch between 8- and 32-bit PROM mode by writing to the RBW field of the MCFG1 register. The BWIDTH[1:0] input signal determines the reset value of this RBW register field. When RBW is “00” then 8-bit mode is selected. If RBW is “10” then 32-bit mode is selected. Other RBW values are reserved for future use. SRAM access is not affected by the 8-bit PROM mode.

It is also possible to use the EDAC in the 8-bit PROM mode, configured by the edacen VHDL generic, and enabled via the MCFG3 register. Read accesses to the 8-bit PROM area will be done in five-byte bursts for all 32-, 16- and 8-bit AMBA AHB accesses. After a potential correction, the whole 32-bit word is output on the AHB data bus, allowing the master to chose the bytes needed (big-endian). EDAC support is not provided for write accesses, they are instead performed in the same way as without the EDAC enabled. The checksum byte must be written by the user into the correct byte address location.

The fifth byte corresponds to the EDAC checksum and is located in the upper part of the effective memory area, as explained in detail in the definition of the MCFG1 memory configuration register. The EDAC checksums are located in the upper quarter of what is defined as available EDAC area by means of the EBSZ field and the ROMBSZ field or rombanksz VHDL generic. When set to 0, the size of the available EDAC area is defined as the PROM bank size. When set to 1, as twice the PROM bank size. When set to 2, as four times the PROM bank size. And when set to 3, as eight times the PROM bank size. For any other value than 0, the use of multiple PROM banks is required.

Example, if ROMBSZ=10 and EBSZ=1, the EDAC area is 8KiB*2^ROMBSZ*2^EBSZ= 16MiB=0x01000000. The checksum byte for the first word located at address 0x00000000 to 0x00000003 is located at 0x00C00000. The checksum byte for the second word located at address 0x00000004 to 0x00000007 is located at 0x00C00001, and so on. Since EBSZ=1, two PROM banks are required for implementing the EDAC area, each bank with size 8MiB=0x00800000.

31.2.2 Access errors

The active low Bus Exception signal (BEXCN) can be used to signal access errors. It is enabled by setting the BEXCEN bit in MCFG1 and is active for all types of accesses to all areas (PROM, SRAM and I/O). The BEXCN signal is sampled on the same cycle as read data is sampled. For writes it is sampled on the last rising edge before writen/rwen is de-asserted (writen and rwen are clocked on the falling edge). When a bus exception is detected an error response will be generated for the access.

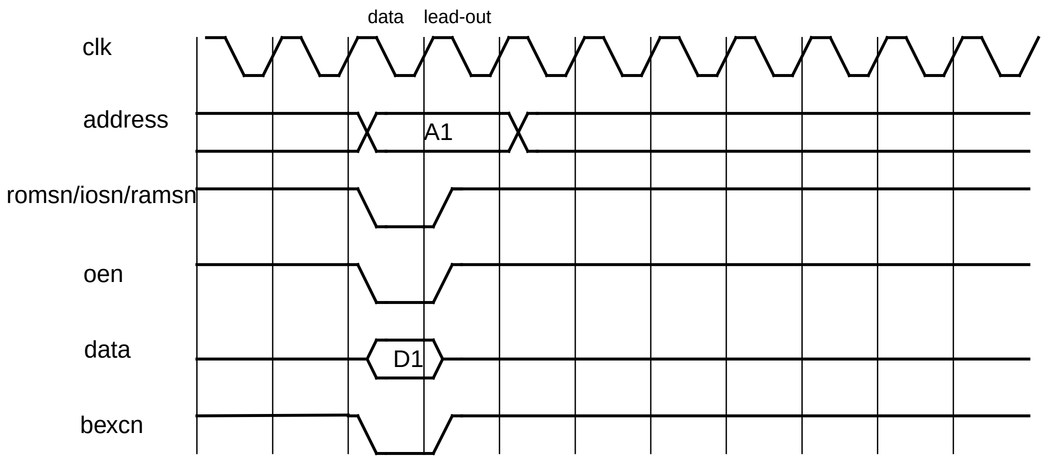


Figure 106. Read cycle with BEXCN.

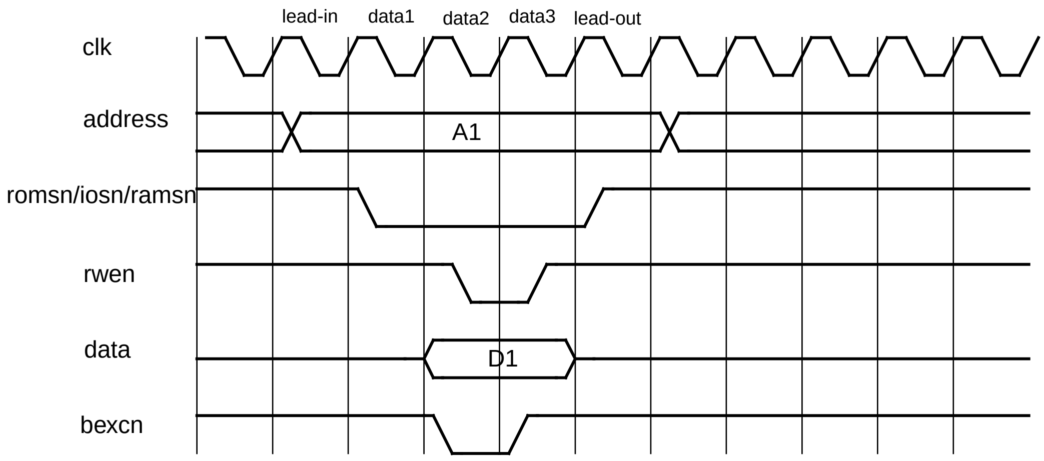


Figure 107. Write cycle with BEXCN.

31.2.3 Using bus ready signalling

The Bus Ready (BRDYN) signal can be used to add waitstates to I/O-area accesses, covering the complete memory area and both read and write accesses. It is enabled by setting the Bus Ready Enable

(BRDYEN) bit in the MCFG1 register. An access will have at least the amount of waitstates set with the VHDL generic or through the register, but will be further stretched until BRDYN is asserted. Additional waitstates can thus be inserted after the pre-set number of waitstates by de-asserting the BRDYN signal. BRDYN should be asserted in the cycle preceding the last one. It is recommended that BRDYN remains asserted until the IOSN signal is de-asserted, to ensure that the access has been properly completed and avoiding the system to stall. Read accesses will have the same timing as when EDAC is enabled while write accesses will have the timing as for single accesses even if bursts are performed.

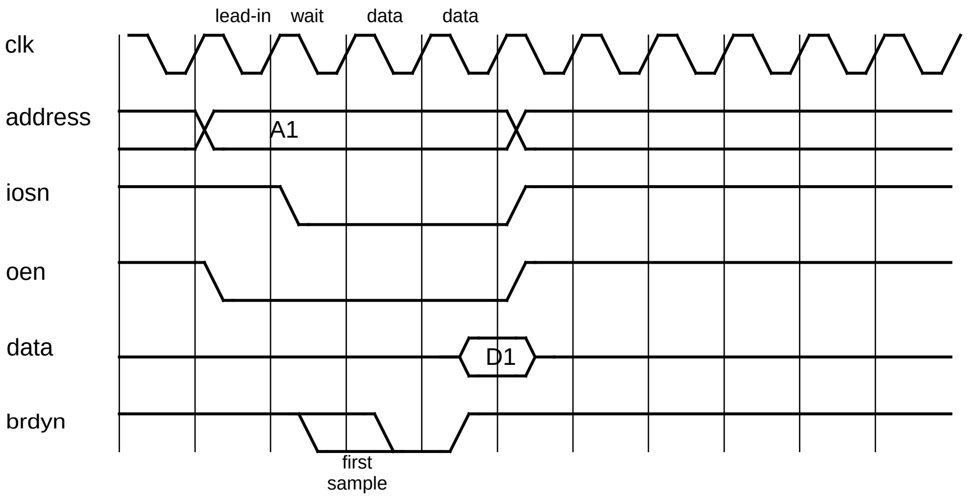


Figure 108. I/O READ cycle, programmed with 1 wait state, and with an extra data cycle added with BRDYN.

31.3 PROM/SRAM/IO waveforms

The internal and external waveforms of the interface are presented in the figures hereafter.

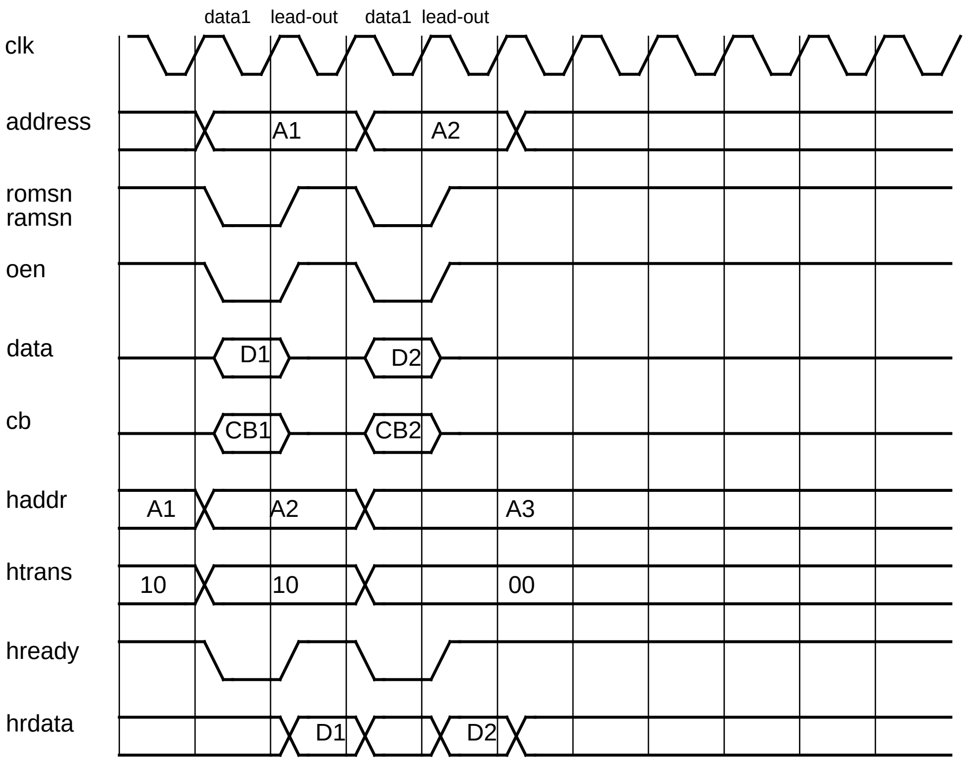


Figure 109. PROM/SRAM non-consecutive read cyclecs.

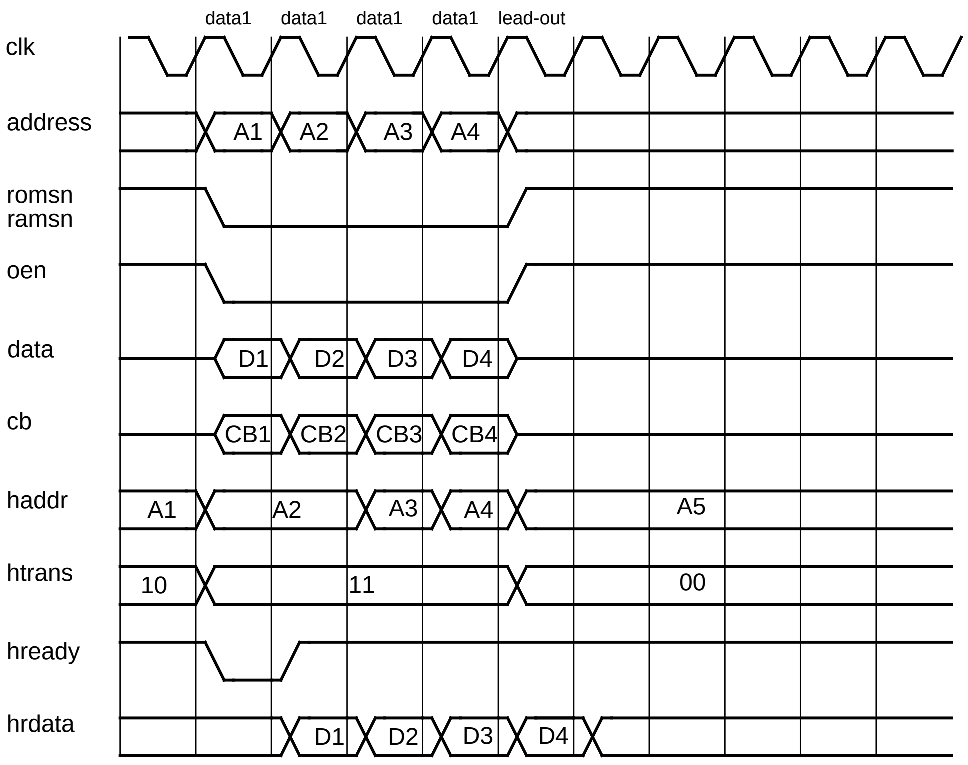


Figure 110. 32-bit PROM/SRAM sequential read access with 0 wait-states and EDAC disabled.

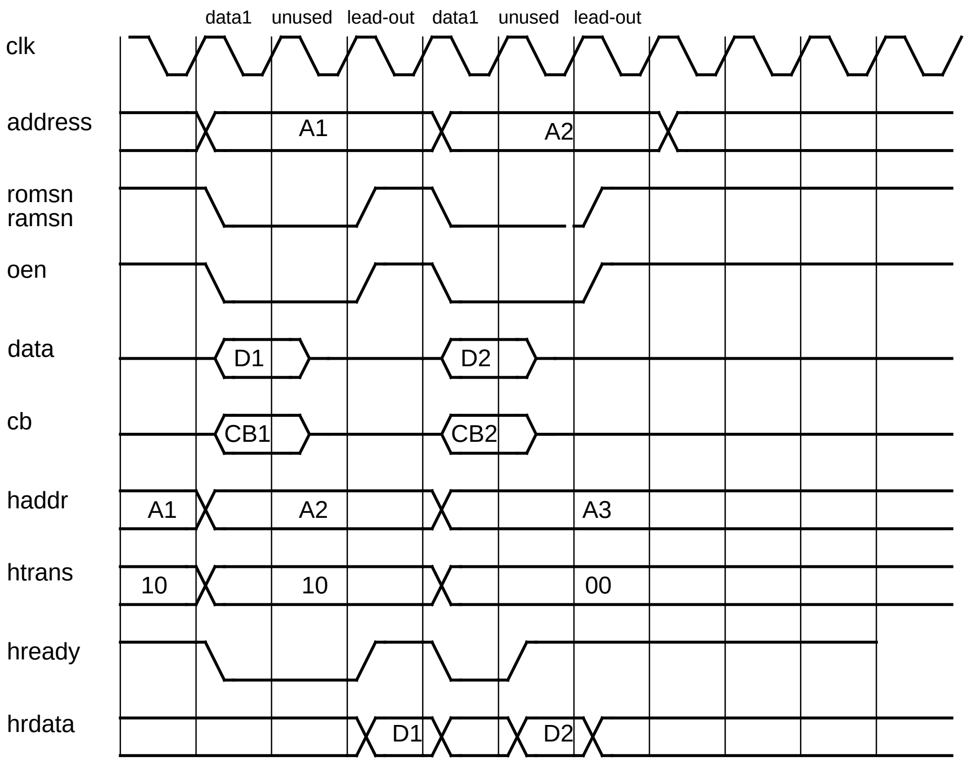


Figure 111. 32-bit PROM/SRAM non-sequential read access with 0 wait-states and EDAC enabled.

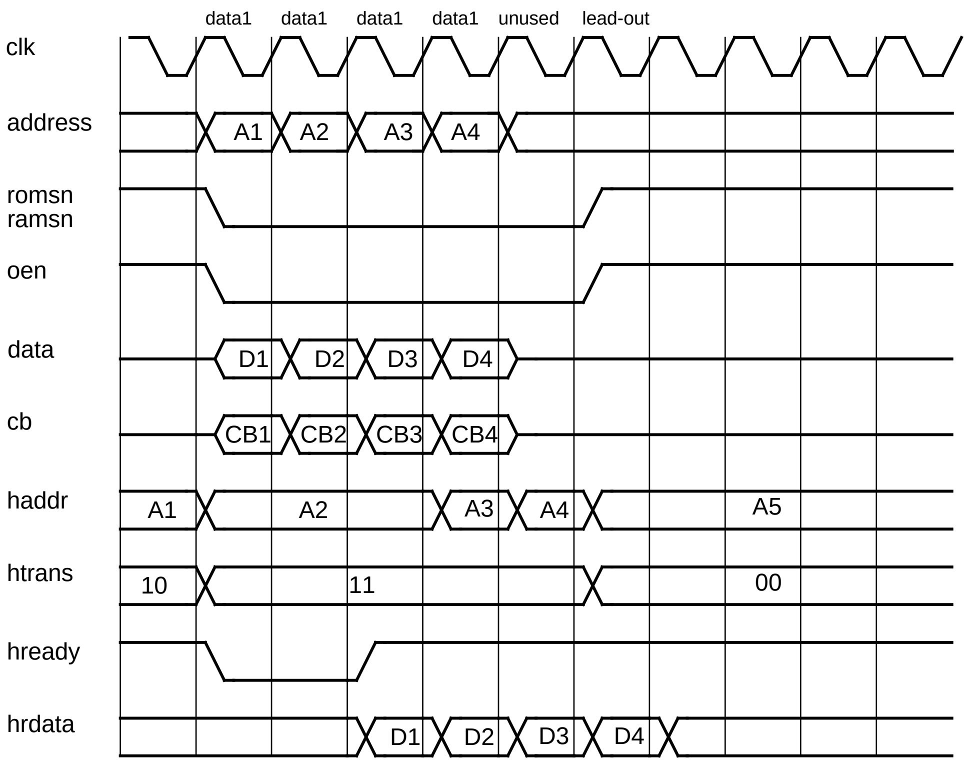


Figure 112. 32-bit PROM/SRAM sequential read access with 0 wait-states and EDAC enabled..

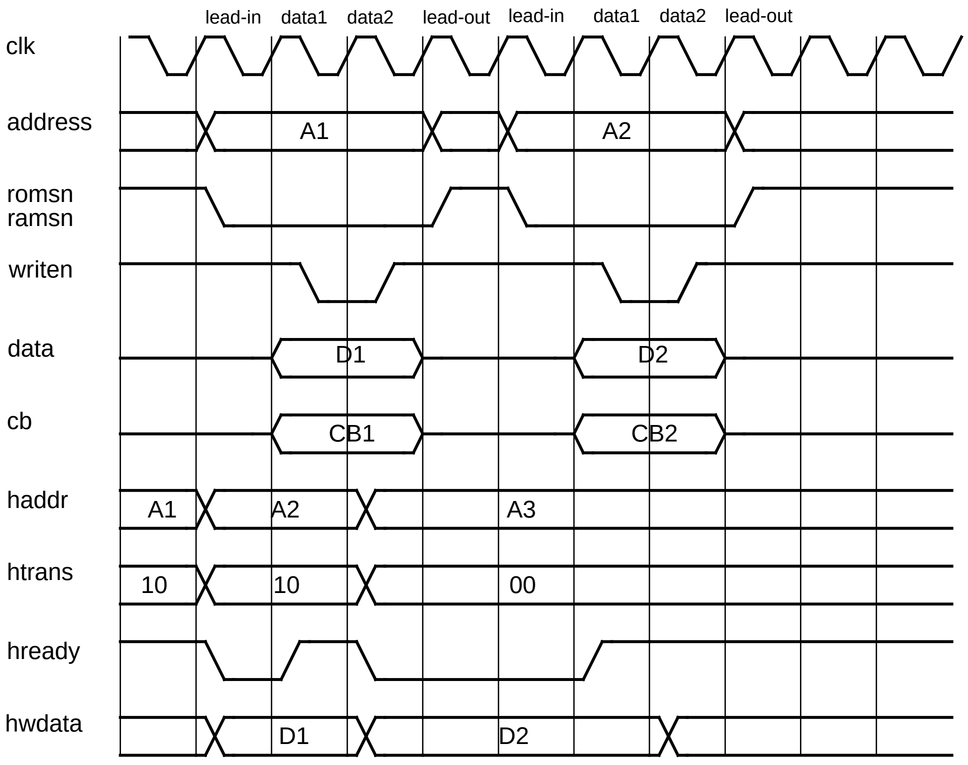


Figure 113. 32-bit PROM/SRAM non-sequential write access with 0 wait-states and EDAC disabled.

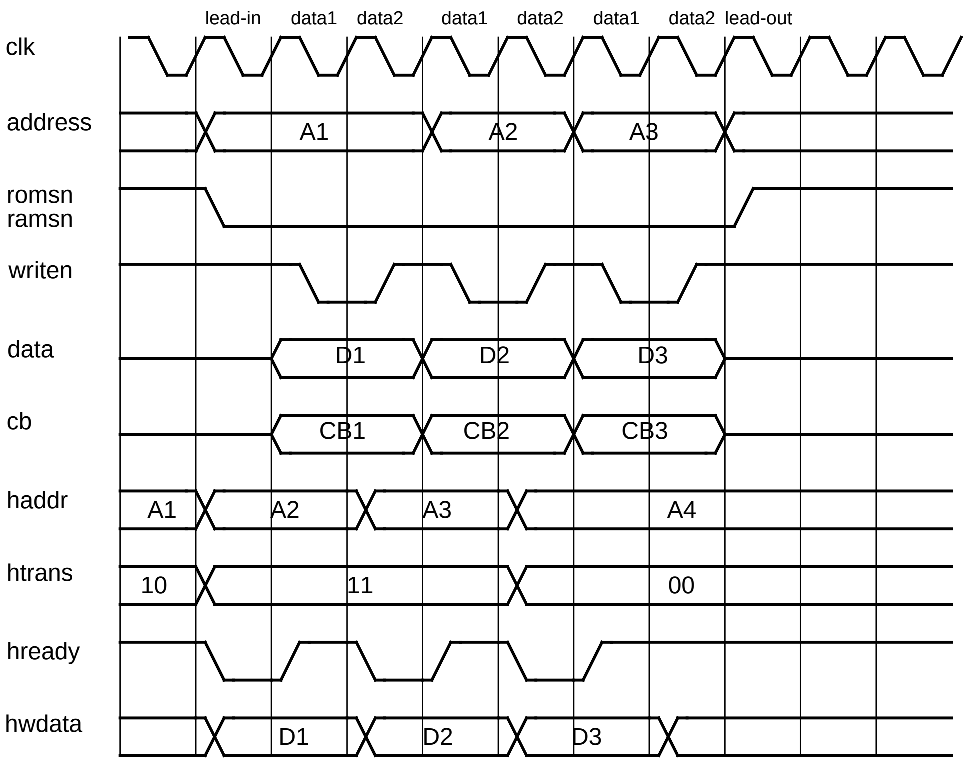


Figure 114. 32-bit PROM/SRAM sequential write access with 0 wait-states and EDAC disabled.

If waitstates are configured through the VHDL generics or registers, one extra data cycle will be inserted for each waitstate in both read and write cycles. The timing for write accesses is not affected when EDAC is enabled while one extra latency cycle is introduced for single access reads and at the beginning of read bursts.

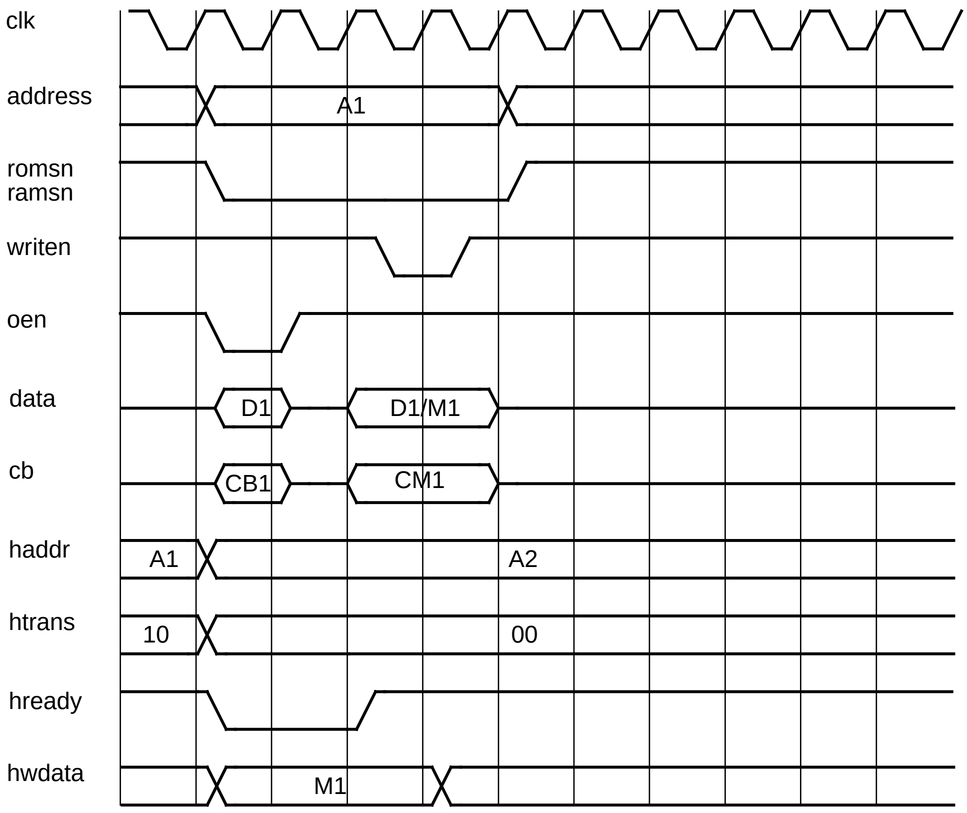


Figure 115. 32-bit PROM/SRAM rmw access with 0 wait-states and EDAC disabled.

Read-Modify-Write (RMW) accesses will have an additional waitstate inserted to accommodate decoding when EDAC is enabled.

I/O accesses are similar to PROM and SRAM accesses but a lead-in and lead-out cycle is always present.

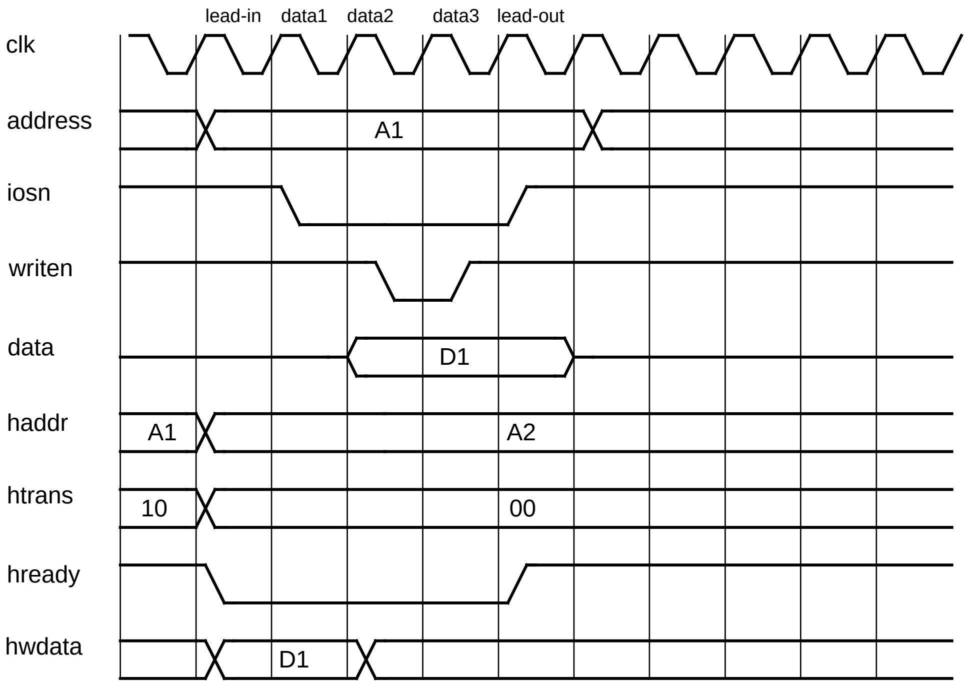


Figure 116. I/O write access with 0 wait-states.

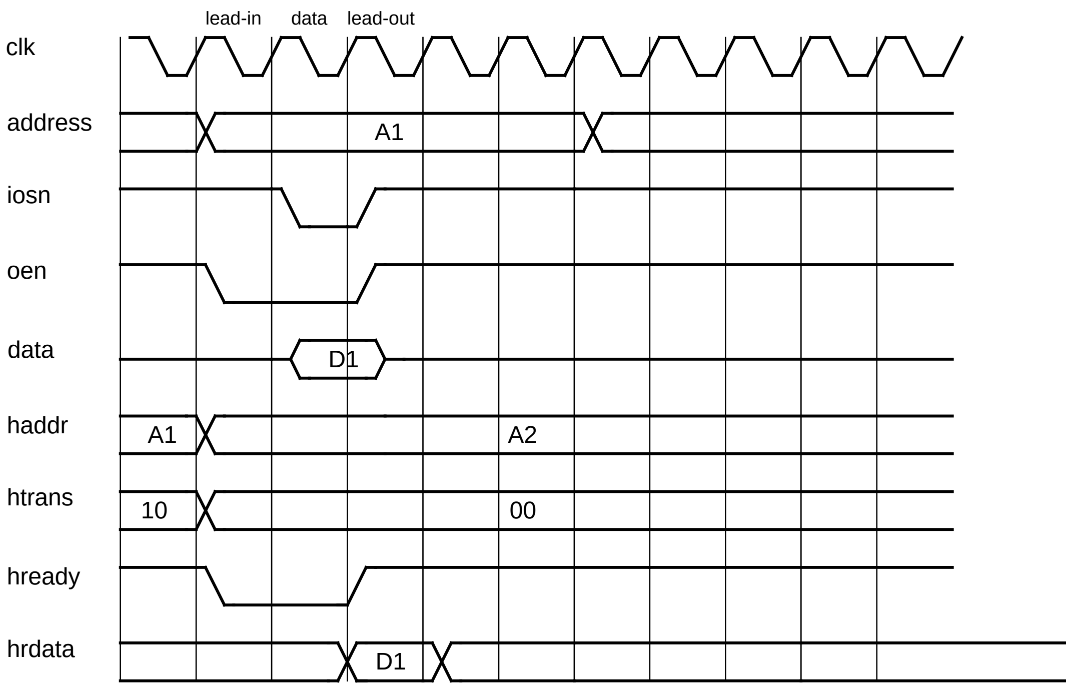


Figure 117. I/O read access with 0 wait-states

31.4 Registers

The core is programmed through registers mapped into APB address space.

*Table 289.*FT PROM/SRAM/IO controller registers

APB Address offset	Register
0x0	Memory configuration register 1
0x4	Memory configuration register 2
0x8	Memory configuration register 3

Table 290. Memory configuration register 1.

31 27	26	25	24	23 20	19 18	17 14	13 12	11	10	9 8	7 4	3 0
RESERVED	BR	BE		IOWS		ROMBSZ	EBSZ	RW		RBW	RESERVED	ROMWS

31: 27	RESERVED
26	Bus ready enable (BR) - Enables the bus ready signal (BRDYN) for I/O-area.
25	Bus exception enable (BE) - Enables the bus exception signal (BEXCEN) for PROM, SRAM and I/O areas
24	RESERVED
23: 20	I/O wait states (IOWS) - Sets the number of waitstates for accesses to the I/O-area. Only available if the wsreg VHDL generic is set to one.
19: 18	RESERVED
17: 14	ROM bank size (ROMBSZ) - Sets the PROM bank size. Only available if the rombanksz VHDL generic is set to zero. Otherwise, the rombanksz VHDL generic sets the bank size and the value can be read from this field. 0 = 8KiB, 1 = 16KiB, 2 = 32KiB, 3 = 64KiB, ..., 15=256 MiB (i.e. 8 KiB * 2**ROMBSZ).

Table 290. Memory configuration register 1.

13: 12	EDAC bank size (EBSZ) - Sets the EDAC bank size for 8-bit PROM support. Only available if the rombanksz VHDL generic is zero, and edacen and prom8en VHDL generics are one. Otherwise, the value is fixed to 0. The resulting EDAC bank size is 2^EBSZ * 2^ROMBSZ * 8KiB. Note that only the three lower quarters of the bank can be used for user data. The EDAC checksums are placed in the upper quarter of the bank.
11	ROM write enable (RW) - Enables writes to the PROM memory area. When disabled, writes to the PROM area will generate an ERROR response on the AHB bus.
10	RESERVED
9: 8	ROM data bus width (RBW) - Sets the PROM data bus width. “00” = 8-bit, “10” = 32-bit, others reserved.
7: 4	RESERVED
3: 0	ROM waitstates (ROMWS) - Sets the number of waitstates for accesses to the PROM area. Reset to all-ones. Only available if the wsreg generic is set to one.

Table 291. Memory configuration register 2.

31 … 13	12 … 9	8 7	6	5 4 3 2	1 0
RESERVED	RAMBSZ		RW	RESERVED	RAMW

31: 13	RESERVED
12: 9	RAM bank size (RAMBSZ) - Sets the RAM bank size. Only available if the banksz VHDL generic is set to zero. Otherwise, the banksz VHDL generic sets the bank size and the value can be read from this field. 0 = 8KiB, 1 = 16KiB, 2 = 32KiB, 3 = 64KiB, ..., 15=256 MiB (i.e. 8 KiB * 2**RAMBSZ)
8: 7	RESERVED
6	Read-modify-write enable (RW) - Enables read-modify-write cycles for write accesses. Only available if the rmw VHDL generic is set to one.
5: 2	RESERVED
1: 0	RAM waitstates (RAMW) - Sets the number of waitstates for accesses to the RAM area. Only available if the wsreg VHDL generic is set to one.

Table 292. Memory configuration register 3.

31 … 20	19 … 12	11	10	9	8	7 6 5 4 3 2 1 0
RESERVED	SEC	WB	RB	SE	PE	TCB

31: 20	RESERVED
19: 12	Single error counter.(SEC) - This field increments each time a single error is detected until the maximum value that can be stored in the field is reached. Each bit can be reset by writing a one to it.
11	Write bypass (WB) - Enables EDAC write bypass. When enabled the TCB field will be used as checkbits in all write operations.
10	Read bypass (RB) - Enables EDAC read bypass. When enabled checkbits read from memory in all read operations will be stored in the TCB field.
9	SRAM EDAC enable (SE) - Enables EDAC for the SRAM area.
8	PROM EDAC enable (PE) - Enables EDAC for the PROM area. Reset value is taken from the input signal sri.edac.
7: 0	Test checkbits (TCB) - Used as checkbits in write operations when WB is activated and checkbits from read operations are stored here when RB is activated.

All the fields in MCFG3 register are available if the edacen VHDL generic is set to one except SEC field which also requires that the errcnt VHDL generic is set to one. The exact breakpoint between the SEC and RESERVED field depends on the cntbits generic. The breakpoint is 11+cntbits. The values shown in the table is for maximum cntbits value 8.

31.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x051. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

31.6 Configuration options

Table 289 shows the configuration options of the core (VHDL generics).

Table 293. Controller configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index.	1 - NAHBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0xFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#FF0#
ramaddr	ADDR field of the AHB BAR1 defining RAM address space. Default RAM area is 0x40000000-0x40FFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR1 defining RAM address space.	0 -16#FFF#	16#FF0#
ioaddr	ADDR field of the AHB BAR2 defining IO address space. Default RAM area is 0x20000000-0x20FFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR2 defining IO address space.	0 - 16#FFF#	16#FF0#
ramws	Number of waitstates during access to SRAM area.	0 - 15	0
romws	Number of waitstates during access to PROM area.	0 - 15	2
iows	Number of waitstates during access to IO area.	0 - 15	2
rmw	Enable read-modify-write cycles.	0 - 1	0
srbanks	Set the number of RAM banks.	1 - 8	1
banksz	Set the size of bank 1 - 4. 1 = 16KiB, 2 = 32KiB, 3 = 64KiB, ... , 15 = 256 MiB (i.e. 8 KiB * 2**banksz). If set to zero, the bank size is set with the rambsz field in the MCFG2 register.	0 - 15	15
rombanks	Sets the number of PROM banks available.	1 - 8	1
rombanksz	Sets the size of one PROM bank. 1 = 16KiB, 2 = 32KiB, 3 = 64KiB, ... , 15 = 256 MiB (i.e. 8 KiB * 2**rombanksz). If set to zero, the bank size is set with the rombsz field in the MCFG1 register.	0 - 15	15
rombankszdef	Sets the reset value of the rombsz register field in MCFG1 if available.	0 - 15	15
pindex	APB slave index.	1 - NAPBSLV-1	0
paddr	APB address.	1 - 16#FFF#	0
pmask	APB address mask.	1 - 16#FFF#	16#FFF#
edacen	EDAC enable. If set to one, EDAC logic is synthesized.	0 - 1	0
errcnt	If one, a single error counter is added.	0 - 1	0
cntbits	Number of bits in the single error counter.	1 - 8	1
wsreg	Enable programmable waitstate generation.	0 - 1	0
prom8en	Enable 8-bit PROM mode.	0 - 1	0
oepol	Select polarity of output enable signals. 0 = active low, 1 = active high.	0 - 1	0

31.7 Signal descriptions

Table 294 shows the interface signals of the core (VHDL ports).

*Table 294.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low

*Table 294.*Signal descriptions

Signal name	Field	Type	Function	Active
SRI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Bus ready strobe	Low
	BEXCN	Input	Bus exception	Low
	WRN[3:0]	Input	Not used	-
	BWIDTH[1:0]	Input	Sets the reset value of the PROM data bus width field in the MCFG1 register	-
	SD[31:0]	Input	Not used	-
	CB[7:0]	Input	Checkbits	-
	PROMDATA[31:0]	Input	Not used	-
	EDAC	Input	The reset value for the PROM EDAC enable bit	High

*Table 294.*Signal descriptions

Signal name	Field	Type	Function	Active
SRO	ADDRESS[31:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data	High
	RAMSN[7:0]	Output	SRAM chip-select	Low
	RAMOEN[7:0]	Output	SRAM output enable	Low
	IOSN	Output	IO area chip select	Low
	ROMSN[7:0]	Output	PROM chip-select	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0]. Any WRN[ ] signal can be used for CB[ ].	Low
	MBEN[3:0]	Output	Byte enable: MBEN[0] corresponds to DATA[31:24], MBEN[1] corresponds to DATA[23:16], MBEN[2] corresponds to DATA[15:8], MBEN[3] corresponds to DATA[7:0]. Any MBEN[ ] signal can be used for CB[ ].	
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus.Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0]. Any BDRIVE[ ] signal can be used for CB[ ].	Low
	READ	Output	Read strobe	High
	RAMN	Output	Common SRAM Chip Select. Always asserted when one of the 8 RAMSN signals is asserted.	Low
	ROMN	Output	Common PROM Chip Select. Always asserted when one of the 8 ROMSN signals is asserted.	Low
	SA[14:0]	Output	Not used	-
	CB[7:0]	Output	Checkbits	-
	PSEL	Output	Not used	-
	CE	Output	Single error detected.	High
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SDO	SDCASN	Output	Not used. All signals are drive to inactive state.	Low

* see GRLIB IP Library User’s Manual

31.8 Library dependencies

Table 295 shows libraries used when instantiating the core (VHDL libraries).

*Table 295.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

31.9 Component declaration

The core has the following component declaration.

```
component ftsrctrl is
  generic (
    hindex       : integer := 0;
    romaddr      : integer := 0;
    rommask      : integer := 16#ff0#;
    ramaddr      : integer := 16#400#;
    rammask      : integer := 16#ff0#;
    ioaddr       : integer := 16#200#;
    iomask       : integer := 16#ff0#;
    ramws        : integer := 0;
    romws        : integer := 2;
    iows         : integer := 2;
    rmw          : integer := 0;
    srbanks      : integer range 1 to 8  := 1;
    banksz       : integer range 0 to 15 := 15;
    rombanks     : integer range 1 to 8  := 1;
    rombanksz    : integer range 0 to 15 := 15;
    rombankszdef : integer range 0 to 15 := 15;
    pindex       : integer := 0;
    paddr        : integer := 0;
    pmask        : integer := 16#fff#;
    edacen       : integer range 0 to 1 := 1;
    errcnt       : integer range 0 to 1 := 0;
    cntbits      : integer range 1 to 8 := 1;
    wsreg        : integer := 0;
    oepol        : integer := 0;
    prom8en      : integer := 0
  );
  port (
    rst          : in  std_ulogic;
    clk          : in  std_ulogic;
    ahbsi        : in  ahb_slv_in_type;
    ahbso        : out ahb_slv_out_type;
    apbi         : in  apb_slv_in_type;
    apbo         : out apb_slv_out_type;
    sri          : in  memory_in_type;
    sro          : out memory_out_type;
    sdo          : out sdctrl_out_type
  );
end component;
```

31.10 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined in the example design's port map and connected to the memory controller. System clock and reset are generated by GR Clock Generator and Reset Generator. The CE signal of the memory controller is also connected to the AHB status register.

Memory controller decodes default memory areas: PROM area is 0x0 - 0xFFFFFF and RAM area is 0x40000000 - 0x40FFFFF.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;

    -- memory bus
    address  : out   std_logic_vector(27 downto 0); -- memory bus
    data     : inout std_logic_vector(31 downto 0);
    ramsn    : out   std_logic_vector(4 downto 0);
    ramoen   : out   std_logic_vector(4 downto 0);
    rwen     : inout std_logic_vector(3 downto 0);
    romsn    : out   std_logic_vector(1 downto 0);
    iosn     : out   std_logic;
    oen      : out   std_logic;
    read     : out   std_logic;
    writen   : inout std_logic;
    brdyn    : in    std_logic;
    bexcn    : in    std_logic;
-- sdram i/f
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0); -- optional sdram data
    cb       : inout std_logic_vector(7 downto 0); --checkbits
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo : sdctrl_out_type;

  signal wprot : wprot_out_type;  -- dummy signal, not used
  signal clkm, rstn : std_ulogic; -- system clock and reset

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;
  signal gnd : std_ulogic;

  signal stati : ahbstat_in_type; --correctable error vector

begin

  -- AMBA Components are defined here ...


  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                              tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);

  -- AHB Status Register
  astat0 : ahbstat generic map(pindex => 13, paddr => 13, pirq => 11,
    nftslv => 1)
    port map(rstn, clkm, ahbmi, ahbsi, stati, apbi, apbo(13));

  stati.cerror(0) <= memo.ce;

  -- Memory controller
  mctrl0 : ftsrctrl generic map (rmw => 1, pindex => 10, paddr => 10,
    edacen => 1, errcnt => 1, cntbits => 4)
    port map (rstn, clkm, ahbsi, ahbso(0), apbi, apbo(10), memi, memo,
  sdo);


  -- I/O pads driving data memory bus data signals
  datapads : for i in 0 to 3 generate
      data_pad : iopadv generic map (width => 8)
      port map (pad => data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(i),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  --I/O pads driving checkbit signals
  cb_pad : iopadv generic map (width => 8)
      port map (pad => cb,
                o => memi.cb,
                en => memo.bdrive(0),
                i => memo.cb;

  -- connect memory controller outputs to entity output signals
  address <= memo.address; ramsn <= memo.ramsn; romsn <= memo.romsn;
  oen <= memo.oen; rwen <= memo.wrn; ramoen <= memo.ramoen;
  writen <= memo.writen; read <= memo.read; iosn <= memo.iosn;
  sdcke <= sdo.sdcke; sdwen <= sdo.sdwen; sdcsn <= sdo.sdcsn;
  sdrasn <= sdo.rasn; sdcasn <= sdo.casn; sddqm <= sdo.dqm;

end;
```

32 FTSRCTRL8 - 8-bit SRAM/16-bit IO Memory Controller with EDAC

32.1 Overview

The fault tolerant 8-bit SRAM/16-bit I/O memory interface uses a common 16-bit data bus to interface 8-bit SRAM and 16-bit I/O devices. It provides an Error Detection And Correction unit (EDAC), correcting up to two errors and detecting up to four errors in a data byte. The EDAC eight checkbits are stored in parallel with the 8-bit data in SRAM memory. Configuration of the memory controller functions is performed through the APB bus interface.

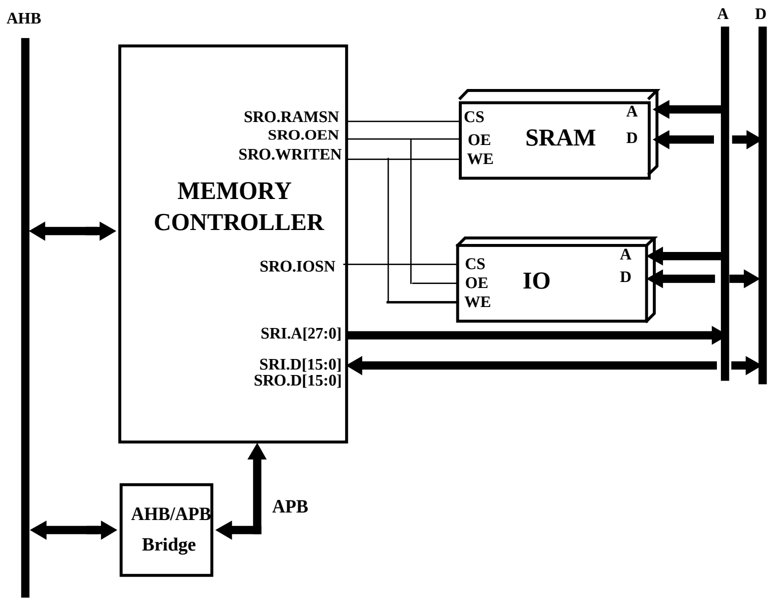


Figure 118. Block diagram

32.2 Operation

The controller is configured through VHDL generics to decode two address ranges: SRAM and I/O area. By default the SRAM area is mapped into address range 0x40000000 - 0x40FFFFFF, and the I/O area is mapped to 0x20000000 - 0x20FFFFFF.

One chip select is decoded for the I/O area, while SRAM can have up to 8 chip select signals. The controller generates a common write-enable signal (WRITEN) for both SRAM and I/O. The number of waitstates may be separately configured for the two address ranges.

The EDAC function is optional, and can be enabled with the edacen VHDL generic. The configuration of the EDAC is done through a configuration register accessed from the APB bus. During nominal operation, the EDAC checksum is generated and checked automatically. The 8-bit input to the EDAC function is split into two 4-bit nibbles. A modified hamming(8,4,4) coding featuring a single error correction and double error detection is applied to each 4-bit nibble. This makes the EDAC capable of correcting up to two errors and detecting up to four errors per 8-bit data. Single errors (correctable errors) are corrected without generating any indication of this condition in the bus response. If a multiple error (uncorrectable errors) is detected, a two cycle error response is given on the AHB bus.

Single errors may be monitored in two ways:

- by monitoring the CE signal which is asserted for one cycle each time a correctable error is detected.
- by checking the single error counter which is accessed from the MCFG3 configuration register.

The CE signal can be connected to the AHB status register which stores information of the AHB instruction causing the error and also generates interrupts. See the AHB status register documentation for more information.

The EDAC function can only be enabled for SRAM area accesses. If a 16-bit or 32-bit bus access is performed, the memory controller calculates the EDAC checksum for each byte read from the memory but the indication of single error is only signaled when the access is done. (I.e. if more than one byte in a 32-bit access has a single error, only one error is indicated for the hole 32-bit access.)

The equations below show how the EDAC checkbits are generated:

```
CB7 = Data[15] ^ Data[14] ^ Data[13]     // i.e. Data[7]
C̅B̅6̅ = Data[15] ^ Data[14] ^ Data[12]     // i.e. Data[6]
C̅B̅5̅ = Data[15] ^ Data[13] ^ Data[12]     // i.e. Data[5]
CB4 = Data[14] ^ Data[13] ^ Data[12]     // i.e. Data[4]
CB3 = Data[11] ^ Data[10] ^ Data[ 9]     // i.e. Data[3]
C̅B̅2̅ = Data[11] ^ Data[10] ^ Data[ 8]     // i.e. Data[2]
C̅B̅1̅ = Data[11] ^ Data[ 9] ^ Data[ 8]     // i.e. Data[1]
CB0 = Data[10] ^ Data[ 9] ^ Data[ 8]     // i.e. Data[0]
```

32.2.1 Memory access

The memory controller supports 32/16/8-bit single accesses and 32-bit burst accesses to the SRAM. A 32-bit or a 16-bit access is performed as multiple 8-bit accesses on the 16-bit memory bus, where data is transferred on data lines 8 to 15 (Data[15:8]). The eight checkbits generated/used by the EDAC are transferred on the eight first data lines (Data[7:0]). For 32-bit and 16-bit accesses, the bytes read from the memory are arranged according to the big-endian order (i.e. for a 32-bit read access, the bytes read from memory address A, A+1, A+2, and A+3 correspond to the bit[31:24], bit[23:16], bit[15:8], and bit[7:0] in the 32-bit word transferred to the AMBA bus. The table 303 shows the expected latency from the memory controller.

*Table 296.*FTSCTRL8 access latency

Accesses	Single data	First data (burst)	Middle data (burst)	Last data (burst)
32-bit write	10	8	8	10
32-bit read	6	6	4	4
16-bit write	4 (+1)	-	-	-
16-bit read	4	-	-	-
8-bit write	4	-	-	-
8-bit read	3	-	-	-

One extra cycle is added for 16-bit burst accesses when Bus Exception is enabled.

32.2.2 I/O access

The memory controller accepts 32/16/8-bit single accesses to the I/O area, but the access generated towards the I/O device is always 16-bit. The two least significant bits of the AMBA address (byte address) determine which half word that should be transferred to the I/O device. (i.e. If the byte address is 0 and it is a 32-bit access, bits 16 to 31 on the AHB bus is transferred on the 16-bit memory bus. If the byte address is 2 and it is a 16-bit access, bit 0 to 15 on the AHB bus is transferred on the 16-bit memory bus.) If the access is an 8-bit access, the data is transferred on data lines 8 to 15 (Data[15:8]) on the memory bus. In case of a write, data lines 0 to 7 is also written to the I/O device but these data lines do not transfer any valid data.

32.2.3 Using Bus Exception

The active low Bus Exception signal (BEXCN) can be used to signal access errors. It is enabled by setting the BEXCEN bit in MCFG1 and is only active for the I/O area. The BEXCN signal is sampled on the same cycle as data is written to memory or read data is sampled. When a bus exception is detected an error response will be generated for the access. One additional latency cycle is added to the AMBA access when the Bus Exception is enable.

32.2.4 Using Bus Ready

The Bus Ready (BRDYN) signal can be used to add waitstates to I/O-area accesses. It is enabled by setting the Bus Ready Enable (BRDYEN) bit in the MCFG1 register. An access will have at least the amount of waitstates set with the VHDL generic or through the register, but will be further stretched until BRDYN is asserted. Additional waitstates can thus be inserted after the pre-set number of wait-states by deasserting the BRDYN signal. BRDYN should be asserted in the cycle preceding the last one. It is recommended that BRDY remains asserted until the IOSN signal is de-asserted, to ensure that the access has been properly completed and avoiding the system to stall.

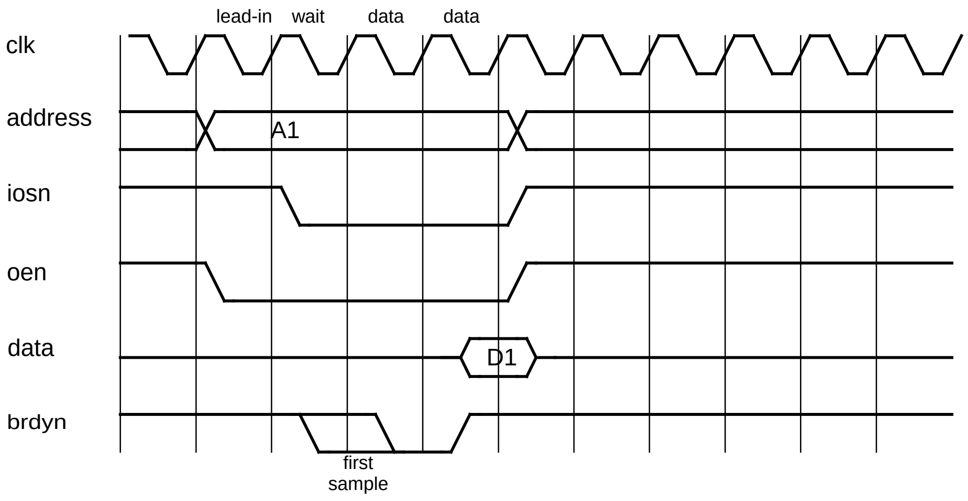


Figure 119. I/O READ cycle, programmed with 1 wait state, and with an extra data cycle added with BRDYN.

32.3 SRAM/IO waveforms

The internal and external waveforms of the interface are presented in the figures below.

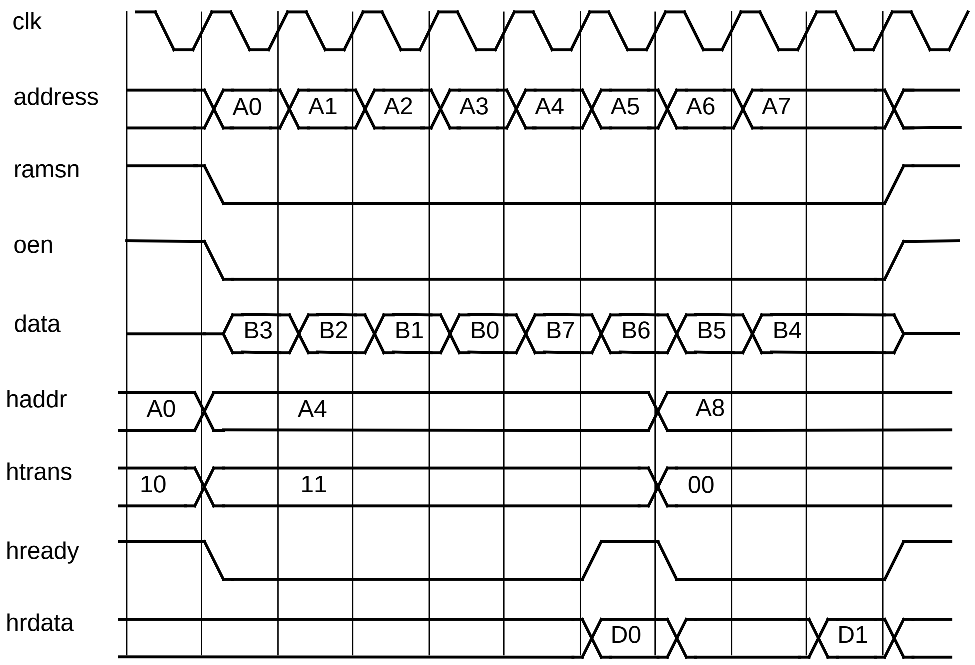


Figure 120. 32-bit SRAM sequential read accesses with 0 wait-states and EDAC enabled.

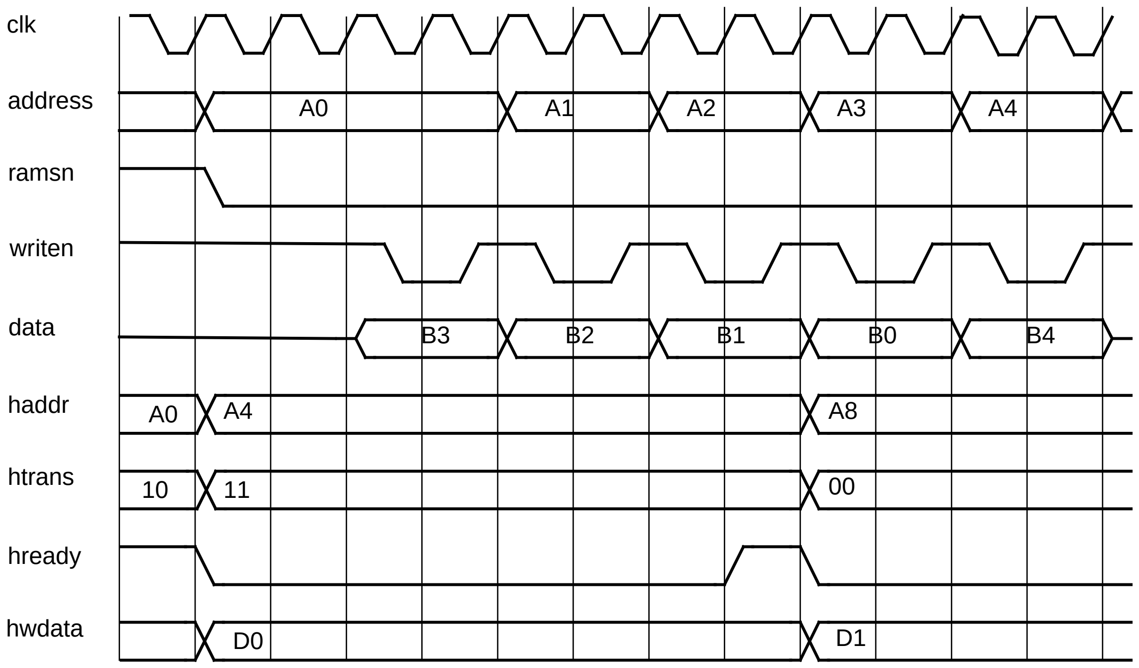


Figure 121. 32-bit SRAM sequential writeaccess with 0 wait-states and EDAC enabled.

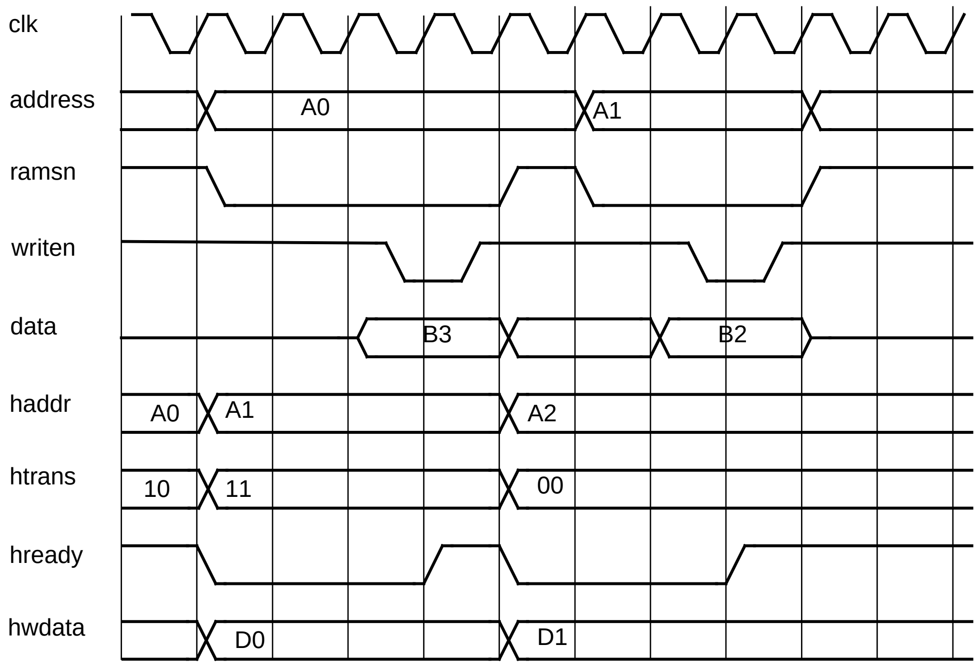


Figure 122. 8-bit SRAM non-sequential write access with 0 wait-states and EDAC enabled.

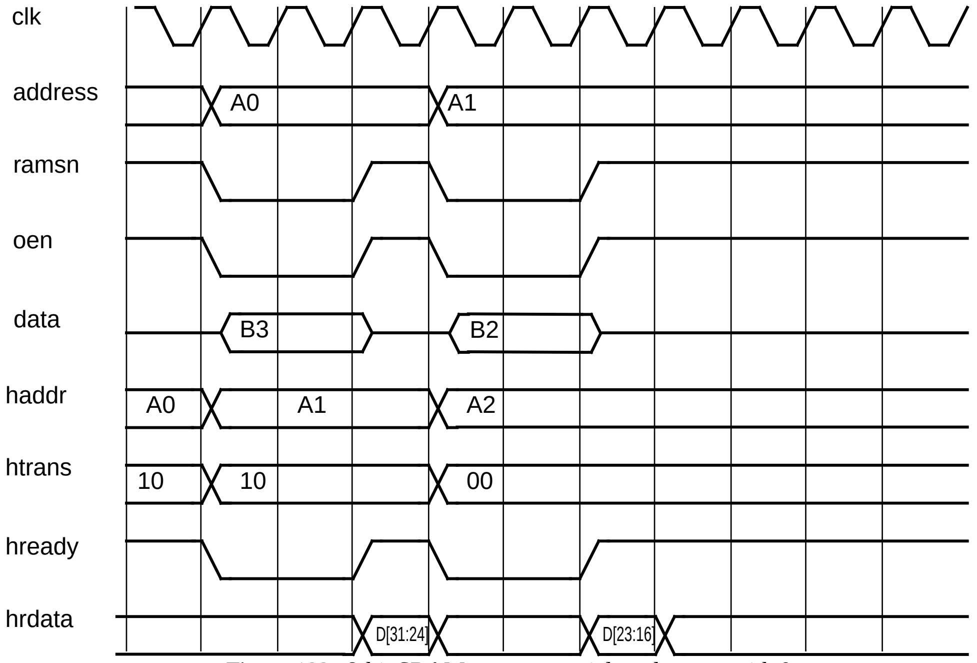


Figure 123. 8-bit SRAM non-sequential read access with 0 wait-states and EDAC enabled.

On a read access, data is sampled one clock cycle before HREADY is asserted.

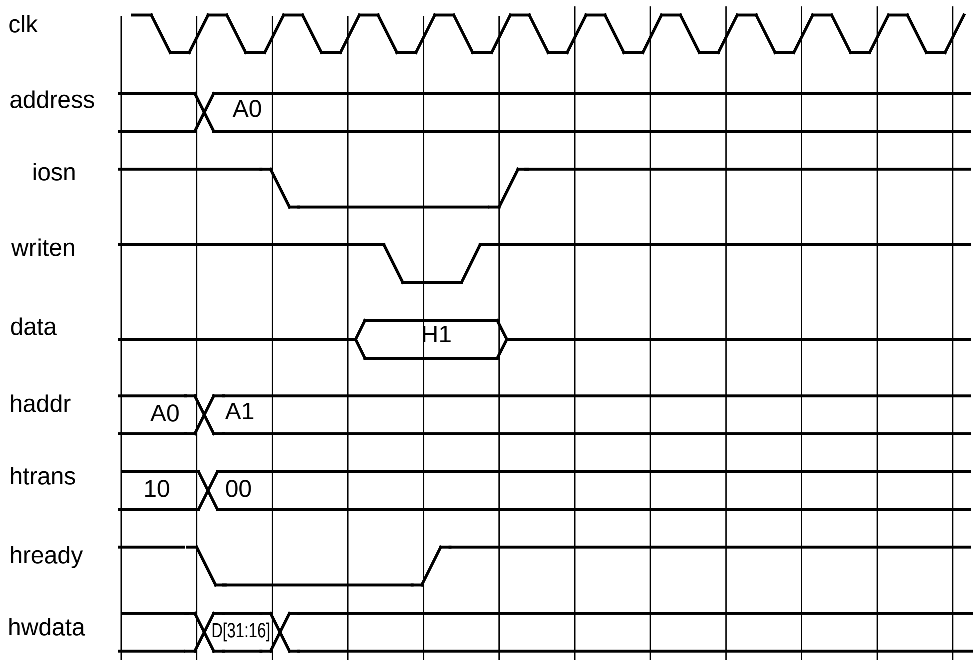


Figure 124. 16-bit I/O non-sequential write access with 0 wait-states.

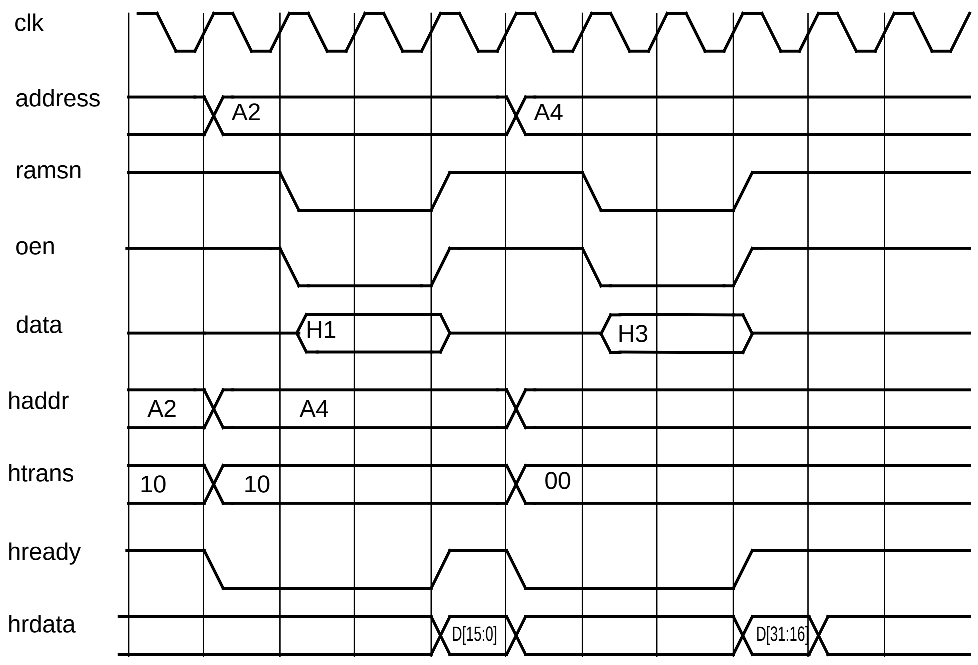


Figure 125. 16-bit I/O non-sequential read access with 0 wait-states.

I/O write accesses are extended with one extra latency cycle if the bus exception is enabled.

If waitstates are configured through the VHDL generics or registers, one extra data cycle will be inserted for each waitstate in both read and write cycles.

32.4 Registers

The core is programmed through registers mapped into APB address space.

*Table 297.*FT SRAM/IO controller registers

APB Address offset	Register
0x0	Memory configuration register 1
0x4	Memory configuration register 2
0x8	Memory configuration register 3

Table 298. MCFG1 register

31 27	26	25	24	23 20	19 0
RESERVED	BRDY	BEXC		IOWS	RESERVED

31 : 27	RESERVED
26	BRDYEN: Enables the BRDYN signal.
25	BEXCEN: Enables the BEXCN signal.
24	RESERVED
23 : 20	IOWS: Sets the number of waitstates for accesses to the IO area. Only available if the wsreg VHDL generic is set to one.
19 : 0	RESERVED

Table 299. MCFG2 register

31 13	12 9	8 2	1 0
RESERVED	RAMBSZ	RESERVED	RAMWS

31 : 12	RESERVED
12 : 9	RAMBSZ: Sets the SRAM bank size. Only available if the banksz VHDL generic is set to zero. Otherwise the banksz VHDL generic sets the bank size. 0 = 8 kB, 15 = 256 MB.
8 : 2	RESERVED
1 : 0	RAMWS: Sets the number of waitstates for accesses to the RAM area. Only available if the wsreg VHDL generic is set to one.

Table 300. MCFG3 register

31 cnt + 13	cnt + 12 12	11	10	9	8	7 0
RESERVED	SEC	WB	RB	SEN		TCB

31 : cnt+13	RESERVED
cnt+12 : 12	SEC. Single error counter. This field increments each time a single error is detected. It saturates at the maximum value that can be stored in this field. Each bit can be reset by writing a one to it. cnt = the number of counter bits.
11	WB: Write bypass. If set, the TCB field will be used as checkbits in all write operations.
10	RB: Read bypass. If set, checkbits read from memory in all read operations will be stored in the TCB field.
9	SEN: SRAM EDAC enable. If set, EDAC will be active for the SRAM area.

Table 300. MCFG3 register

8	RESERVED
7 : 0	TCB: Used as checkbits in write operations when WB is one and checkbits from read operations are stored here when RB is one.

All the fields in the MCFG3 register are available if the edacen VHDL generic is set to one except for the SEC field which also requires that the errcnt VHDL generic is set to one.

32.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x056. For description of vendor and device identifiers see the GRLIB IP Library User’s Manual.

32.6 Configuration options

Table 297 shows the configuration options of the core (VHDL generics).

Table 301. Controller configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index.	1 - NAHBSLV-1	0
ramaddr	ADDR field of the AHB BAR1 defining RAM address space. Default RAM area is 0x40000000-0x40FFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR1 defining RAM address space.	0 -16#FFF#	16#FF0#
ioaddr	ADDR field of the AHB BAR2 defining IO address space. Default RAM area is 0x20000000-0x20FFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR2 defining IO address space.	0 - 16#FFF#	16#FF0#
ramws	Number of waitstates during access to SRAM area.	0 - 15	0
iows	Number of waitstates during access to IO area.	0 - 15	2
srbanks	Set the number of RAM banks.	1 - 8	1
banksz	Set the size of bank 1 - 4. 1 = 16 kB, ... , 15 = 256 MB. If set to zero, the bank size is set with the rambsz field in the MCFG2 register.	0 - 15	15
pindex	APB slave index.	1 - NAPBSLV-1	0
paddr	APB address.	1 - 16#FFF#	0
pmask	APB address mask.	1 - 16#FFF#	16#FFF#
edacen	EDAC enable. If set to one, EDAC logic is synthesized.	0 - 1	0
errcnt	If one, a single error counter is added.	0 - 1	0
cntbits	Number of bits in the single error counter.	1 - 8	1
wsreg	Enable programmable waitstate generation.	0 - 1	0

32.7 Signal descriptions

Table 302 shows the interface signals of the core (VHDL ports).

*Table 302.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low

*Table 302.*Signal descriptions

Signal name	Field	Type	Function	Active
SRI	DATA[31:0]	Input	Memory data: [15:0] used for IO accesses [7:0] used for checkbits for SRAM accesses [15:8] use for data for SRAM accesses	High
	BRDYN	Input	Bus ready strobe	Low
	BEXCN	Input	Bus exception	Low
	WRN[3:0]	Input	Not used	-
	BWIDTH[1:0]	Input	Not used	-
	SD[31:0]	Input	Not used	-
	CB[7:0]	Input	Not used	-
	PROMDATA[31:0]	Input	Not used	-
	EDAC	Input	Not used	-
SRO	ADDRESS[31:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data: [15:0] used for IO accesses [7:0] used for checkbits for SRAM accesses [15:8] use for data for SRAM accesses	High
	RAMSN[7:0]	Output	SRAM chip-select	Low
	RAMOEN[7:0]	Output	SRAM output enable	Low
	IOSN	Output	IO area chip select	Low
	ROMSN[7:0]	Output	Not used	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[15:8], WRN[1] corresponds to DATA[7:0], WRN[3:2] Not used	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus. Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[15:8], BDRIVE[1] corresponds to DATA[7:0], BDRIVE[3:2] Not used	Low
	VBDRIVE[31:0]	Output	Vectored I/O-pad drive signal.	Low
	READ	Output	Read strobe	High
	RAMN	Output	Common SRAM Chip Select. Always asserted when one of the 8 RAMSN signals is asserted.	Low
	ROMN	Output	Not used	-
	SA[14:0]	Output	Not used	-
	CB[7:0]	Output	Not used	-
	PSEL	Output	Not used	-
	CE	Output	Single error detected.	High
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User’s Manual

32.8 Library dependencies

Table 303 shows libraries used when instantiating the core (VHDL libraries).

*Table 303.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

32.9 Component declaration

The core has the following component declaration.

```
component ftsrctrl8 is
  generic (
    hindex       : integer := 0;
    ramaddr      : integer := 16#400#;
    rammask      : integer := 16#ff0#;
    ioaddr       : integer := 16#200#;
    iomask       : integer := 16#ff0#;
    ramws        : integer := 0;
    iows         : integer := 2;
    srbanks      : integer range 1 to 8  := 1;
    banksz       : integer range 0 to 15 := 15;
    pindex       : integer := 0;
    paddr        : integer := 0;
    pmask        : integer := 16#fff#;
    edacen       : integer range 0 to 1 := 1;
    errcnt       : integer range 0 to 1 := 0;
    cntbits      : integer range 1 to 8 := 1;
    wsreg        : integer := 0;
    oepol        : integer := 0
  );
  port (
    rst          : in  std_ulogic;
    clk          : in  std_ulogic;
    ahbsi        : in  ahb_slv_in_type;
    ahbso        : out ahb_slv_out_type;
    apbi         : in  apb_slv_in_type;
    apbo         : out apb_slv_out_type;
    sri          : in  memory_in_type;
    sro          : out memory_out_type
  );
end component;
```

32.10 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined in the example design’s port map and connected to the memory controller. The system clock and reset are generated by GR Clock Generator and Reset Generator. The CE signal of the memory controller is also connected to the AHB status register.

The memory controller decodes default memory areas: I/O area is 0x20000000 - 0x20FFFFFF and RAM area is 0x40000000 - 0x40FFFFF.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
library grlib;
use grlib.amba.all;
library techmap;
use techmap.gencomp.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.misc.all;

entity ftsrctrl8_ex is
  port (
    resetn     : in  std_ulogic;
    clk        : in  std_ulogic;

    address    : out std_logic_vector(27 downto 0);
    data       : inout std_logic_vector(31 downto 0);
    ramsn      : out std_logic_vector (3 downto 0);
    ramoen     : out std_logic_vector (3 downto 0);
    rwen       : out std_logic_vector (3 downto 0);
    oen        : out std_ulogic;
    writen     : out std_ulogic;
    read       : out std_ulogic;
    iosn       : out std_ulogic;
    brdyn      : in  std_ulogic; -- Bus ready
    bexcn      : in  std_ulogic  -- Bus exception
   );
end;

architecture rtl of ftsrctrl8_ex is
signal memi  : memory_in_type;
signal memo  : memory_out_type;

signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi : ahb_mst_in_type;
signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

signal clkm, rstn, rstraw : std_ulogic;
signal cgi   : clkgen_in_type;
signal cgo   : clkgen_out_type;

signal stati : ahbstat_in_type;

begin

  -- clock and reset
  cgi.pllctrl <= "00"; cgi.pllrst <= rstraw; cgi.pllref  <= ’0’;
  clk_pad : clkpad port map (clk, clkm);
  rst0 : rstgen               -- reset generator
  port map (resetn, clkm, ’1’, rstn, rstraw);

  -- AHB controller
  ahb0 : ahbctrl              -- AHB arbiter/multiplexer
  generic map (rrobin => 1, ioaddr => 16#fff#, devid => 16#201#)
  port map (rstn, clkm, ahbmi, ahbmo, ahbsi, ahbso);

  -- Memory controller
  sr0 : ftsrctrl8 generic map (hindex => 0, pindex => 0, edacen => 1)
    port map (rstn, clkm, ahbsi, ahbso(0), apbi, apbo(0), memi, memo);

  brdyn_pad : inpad port map (brdyn, memi.brdyn);
  bexcn_pad : inpad port map (bexcn, memi.bexcn);

  addr_pad : outpadv generic map (width => 28 )
    port map (address, memo.address(27 downto 0));
  rams_pad : outpadv generic map (width => 4)
    port map (ramsn, memo.ramsn(3 downto 0));
  oen_pad  : outpad
    port map (oen, memo.oen);
  rwen_pad : outpadv generic map (width => 4)
```

```
      port map (rwen, memo.wrn);
    roen_pad : outpadv generic map (width => 4)
      port map (ramoen, memo.ramoen(3 downto 0));
    wri_pad  : outpad
      port map (writen, memo.writen);
    read_pad : outpad
      port map (read, memo.read);
    iosn_pad : outpad
      port map (iosn, memo.iosn);
    data_pad : iopadvv generic map (width => 8) -- SRAM and I/O Data
      port map (data(15 downto 8), memo.data(15 downto 8),
         memo.vbdrive(15 downto 8), memi.data(15 downto 8));
    cbdata_pad : iopadvv generic map (width => 8) -- SRAM checkbits and I/O Data
      port map (data(7 downto 0), memo.data(7 downto 0),
         memo.vbdrive(7 downto 0), memi.data(7 downto 0));

    -- APB bridge and AHB stat
    apb0 : apbctrl              -- AHB/APB bridge
    generic map (hindex => 1, haddr => 16#800#)
      port map (rstn, clkm, ahbsi, ahbso(1), apbi, apbo );

    stati.cerror(0) <= memo.ce;
    ahbstat0 : ahbstat generic map (pindex => 15, paddr => 15, pirq => 1)
      port map (rstn, clkm, ahbmi, ahbsi, stati, apbi, apbo(15));
  end;
```

33 GR1553B - MIL-STD-1553B / AS15531 Interface

33.1 Overview

This interface core connects the AMBA AHB/APB bus to a single- or dual redundant MIL-STD-1553B bus, and can act as either Bus Controller, Remote Terminal or Bus Monitor.

MIL-STD-1553B (and derived standard SAE AS15531) is a bus standard for transferring data between up to 32 devices over a shared (typically dual-redundant) differential wire. The bus is designed for predictable real-time behavior and fault-tolerance. The raw bus data rate is fixed at 1 Mbit/s, giving a maximum of around 770 kbit/s payload data rate.

One of the terminals on the bus is the Bus Controller (BC), which controls all traffic on the bus. The other terminals are Remote Terminals (RTs), which act on commands issued by the bus controller. Each RT is assigned a unique address between 0-30. In addition, the bus may have passive Bus Monitors (BM:s) connected.

There are 5 possible data transfer types on the MIL-STD-1553 bus:

- BC-to-RT transfer (“receive”)
- RT-to-BC transfer (“transmit”)
- RT-to-RT transfer
- Broadcast BC-to-RTs
- Broadcast RT-to-RTs

Each transfer can contain 1-32 data words of 16 bits each.

The bus controller can also send “mode codes” to the RTs to perform administrative tasks such as time synchronization, and reading out terminal status.

33.2 Electrical interface

The core is connected to the MIL-STD-1553B bus wire through single or dual transceivers, isolation transformers and transformer or stub couplers as shown in figure 126. If single-redundancy is used, the unused bus receive P/N signals should be tied both-high or both-low. The transmitter enables are typically inverted and therefore called transmitter inibit (txinh). See the standard and the respective component’s data sheets for more information on the electrical connection.

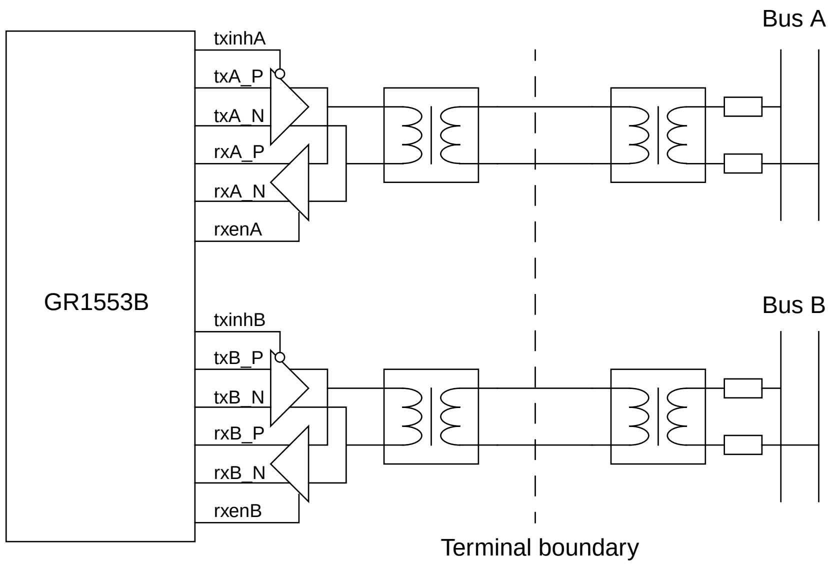


Figure 126. Interface between core and MIL-STD-1553B bus (dual-redundant, transformer coupled)

33.3 Operation

33.3.1 Operating modes

The core contains three separate control units for the Bus Controller, Remote Terminal and Bus Monitor handling, with a shared 1553 codec. All parts may not be present in the hardware, which parts are available can be checked from software by looking at the BCSUP/RTSUP/BMSUP register bits.

The operating mode of the core is controlled by starting and stopping of the BC/RT/BM units via register writes. At start-up, none of the parts are enabled, and the core is completely passive on both the 1553 and AMBA bus.

The BC and RT parts of the core can not be active on the 1553 bus at the same time. While the BC is running or suspended, only the BC (and possibly BM) has access to the 1553 bus, and the RT can only receive and respond to commands when both the BC schedules are completely stopped (not running or even suspended).

The Bus Monitor, however, is only listening on the codec receivers and can therefore operate regardless of the enabled/disabled state of the other two parts.

33.3.2 Register interface

The core is configured and controlled through control registers accessed over the APB bus. Each of the BC,RT,BM parts has a separate set of registers, plus there is a small set of shared registers.

Some of the control register fields for the BC and RT are protected using a ‘key’, a field in the same register that has to be written with a certain value for the write to take effect. The purpose of the keys are to give RT/BM designers a way to ensure that the software can not interfere with the bus traffic by enabling the BC or changing the RT address. If the software is built without knowledge of the key to a certain register, it is very unlikely that it will accidentally perform a write with the correct key to that control register.

33.3.3 Interrupting

The core has one interrupt output, which can be generated from several different source events. Which events should cause an interrupt can be controlled through the IRQ Enable Mask register.

33.3.4 MIL-STD-1553 Codec

The core’s internal codec receives and transmits data words on the 1553 bus, and generates and checks sync patterns and parity.

Loop-back checking logic checks that each transmitted word is also seen on the receive inputs. If the transmitted word is not echoed back, the transmitter stops and signals an error condition, which is then reported back to the user.

33.4 Bus Controller Operation

33.4.1 Overview

When operating as Bus Controller, the core acts as master on the MIL-STD-1553 bus, initiates and performs transfers.

This mode works based on a scheduled transfer list concept. The software sets up in memory a sequence of transfer descriptors and branches, data buffers for sent and received data, and an IRQ pointer ring buffer. When the schedule is started (through a BC action register write), the core processes the list, performs the transfers one after another and writes resulting status into the transfer list and incoming data into the corresponding buffers.

33.4.2 Timing control

In each transfer descriptor in the schedule is a “slot time” field. If the scheduled transfer finishes sooner than its slot time, the core will pause the remaining time before scheduling the next command. This allows the user to accurately control the message timing during a communication frame.

If the transfer uses more than its slot time, the overshooting time will be subtracted from the following command’s time slot. The following command may in turn borrow time from the following command and so on. The core can keep track of up to one second of borrowed time, and will not insert pauses again until the balance is positive, except for intermessage gaps and pauses that the standard requires.

If you wish to execute the schedule as fast as possible you can set all slot times in the schedule to zero. If you want to group a number of transfers you can move all the slot time to the last transfer.

The schedule can be stopped or suspended by writing into the BC action register. When suspended, the schedule’s time will still be accounted, so that the schedule timing will still be correct when the schedule is resumed. When stopped, on the other hand, the schedule’s timers will be reset.

When the extsync bit is set in the schedule’s next transfer descriptor, the core will wait for a positive edge on the external sync input before starting the command. The schedule timer and the time slot balance will then be reset and the command is started. If the sync pulse arrives before the transfer is reached, it is stored so the command will begin immediately. The trigger memory is cleared when stopping (but not when suspending) the schedule. Also, the trigger can be set/cleared by software through the BC action register.

33.4.3 Bus selection

Each transfer descriptor has a bus selection bit that allows you to control on which one of the two redundant buses (‘0’ for bus A, ‘1’ for bus B) the transfer will occur.

Another way to control the bus usage is through the per-RT bus swap register, which has one register bit for each RT address. The bus swap register is an optional feature, software can check the BCFEAT read-only register field to see if it is available.

Writing a ‘1’ to a bit in the per-RT Bus Swap register inverts the meaning of the bus selection bit for all transfers to the corresponding RT, so ‘0’ now means bus ‘B’ and ‘1’ means bus ‘A’. This allows you to switch all transfers to one or a set of RT:s over to the other bus with a single register write and without having to modify any descriptors.

The hardware determines which bus to use by taking the exclusive-or of the bus swap register bit and the bus selection bit. Normally it only makes sense to use one of these two methods for each RT, either the bus selection bit is always zero and the swap register is used, or the swap register bit is always zero and the bus selection bit is used.

If the bus swap register is used for bus selection, the store-bus descriptor bit can be enabled to automatically update the register depending on transfer outcome. If the transfer succeeded on bus A, the bus swap register bit is set to ‘0’, if it succeeds on bus B, the swap register bit is set to ‘1’. If the transfer fails, the bus swap register is set to the opposite value.

33.4.4 Secondary transfer list

The core can be set up with a secondary “asynchronous” transfer list with the same format as the ordinary schedule. This transfer list can be commanded to start at any time during the ordinary schedule. While the core is waiting for a scheduled command’s slot time to finish, it will check if the next asynchronous transfer’s slot time is lower than the remaining sleep time. In that case, the asynchronous command will be scheduled.

If the asynchronous command doesn’t finish in time, time will be borrowed from the next command in the ordinary schedule. In order to not disturb the ordinary schedule, the slot time for the asynchronous messages must therefore be set to pessimistic values.

The exclusive bit in the transfer descriptor can be set if one does not want an asynchronous command scheduled during the sleep time following the transfer.

Asynchronous messages will not be scheduled while the schedule is waiting for a sync pulse or the schedule is suspended and the current slot time has expired, since it is then not known when the next scheduled command will start.

33.4.5 Interrupt generation

Each command in the transfer schedule can be set to generate an interrupt after certain transfers have completed, with or without error. Invalid command descriptors always generate interrupts and stop the schedule. Before a transfer-triggered interrupt is generated, the address to the corresponding descriptor is written into the BC transfer-triggered IRQ ring buffer and the BC Transfer-triggered IRQ Ring Position Register is incremented.

A separate error interrupt signals DMA errors. If a DMA error occurs when reading/writing descriptors, the executing schedule will be suspended. DMA errors in data buffers will cause the corresponding transfer to fail with an error code (see table 307).

Whether any of these interrupt events actually cause an interrupt request on the AMBA bus is controlled by the IRQ Mask Register setting.

33.4.6 Transfer list format

The BC:s transfer list is an array of transfer descriptors mixed with branches as shown in table 304. Each entry has to be aligned to start on a 128-bit (16-byte) boundary. The two unused words in the branch case are free to be used by software to store arbitrary data.

*Table 304.*GR1553B transfer descriptor format

Offset	Value for transfer descriptor	DMA R/W	Value for branch	DMA R/W
0x00	Transfer descriptor word 0 (see table 305)	R	Condition word (see table 309)	R
0x04	Transfer descriptor word 1 (see table 306)	R	Jump address, 128-bit aligned	R
0x08	Data buffer pointer, 16-bit aligned. For write buffers, if bit 0 is set the received data is discarded and the pointer is ignored. This can be used for RT-to-RT transfers where the BC is not interested in the data transferred.	R	Unused	-
0x0C	Result word, written by core (see table 307)	W	Unused	-

The transfer descriptor words are structured as shown in tables 305-307 below.

Table 305. GR1553B BC transfer descriptor word 0 (offset 0x00)

31	30	29	28	27	26	25	24 23	22 20	19	18	17 16	15 0
0	WTRIG	EXCL	IRQE	IRQN	SUSE	SUSN	RETMD	NRET	STBUS	GAP	RESERVED	STIME

31	Must be 0 to identify as descriptor
30	Wait for external trigger (WTRIG)
29	Exclusive time slot (EXCL) - Do not schedule asynchronous messages
28	IRQ after transfer on Error (IRQE)
27	IRQ normally (IRQN) - Always interrupts after transfer
26	Suspend on Error (SUSE) - Suspends the schedule (or stops the async transfer list) on error
25	Suspend normally (SUSN) - Always suspends after transfer
24 : 23	Retry mode (RETMD). 00 - Retry on same bus only. 01 - Retry alternating on both buses 10: Retry first on same bus, then on alternating bus. 11 - Reserved, do not use
22 : 20	Number of retries (NRET) - Number of automatic retries per bus The total number of tries (including the first attempt) is NRET+1 for RETMD=00, 2 x (NRET+1) for RETMD=01/10
19	Store bus (STBUS) - If the transfer succeeds and this bit is set, store the bus on which the transfer succeeded (0 for bus A, 1 for bus B) into the per-RT bus swap register. If the transfer fails and this bit is set, store the opposite bus instead. (only if the per-RT bus mask is supported in the core) See section 33.4.3 for more information.
18	Extended intermessage gap (GAP) - If set, adds an additional amount of gap time, corresponding to the RTTO field, after the transfer
17 : 16	Reserved - Set to 0 for forward compatibility
15 : 0	Slot time (STIME) - Allocated time in 4 microsecond units, remaining time after transfer will insert delay

Table 306. GR1553B BC transfer descriptor word 1 (offset 0x04)

31	30	29 26	25 21	20 16	15 11	10	9 5	4 0
DUM	BUS	RTTO	RTAD2	RTSA2	RTAD1	TR	RTSA1	WCMC

31	Dummy transfer (DUM) - If set to ‘1’ no bus traffic is generated and transfer “succeeds” immediately For dummy transfers, the EXCL,IRQN,SUSN,STBUS,GAP,STIME settings are still in effect, other bits and the data buffer pointer are ignored.	
30	Bus selection (BUS) - Bus to use for transfer, 0 - Bus A, 1 - Bus B	
29:26	RT Timeout (RTTO) - Extra RT status word timeout above nominal in units of 4 us (0000 -14 us, 1111 -74 us). Note: This extra time is also used as extra intermessage gap time if the GAP bit is set.	
25:21	Second RT Address for RT-to-RT transfer (RTAD2)	See table 308 for details on how to setup RTAD1,RTSA1,RTAD2,RTSA2,WCMC,TR for different transfer types.
20:16	Second RT Subaddress for RT-to-RT transfer (RTSA2)	
15:11	RT Address (RTAD1)	
10	Transmit/receive (TR)	Note that bits 15:0 correspond to the (first) command word on the 1553 bus
9:5	RT Subaddress (RTSA1)	
4:0	Word count/Mode code (WCMC)	

Table 307. GR1553B transfer descriptor result word (offset 0x0C)

31	30 24	23 16	15 8	7 4	3	2 0
0	Reserved	RT2ST	RTST	RETCNT	RES	TFRST

31	Always written as 0
30:24	Reserved - Mask away on read for forward compatibility
23:16	RT 2 Status Bits (RT2ST) - Status bits from receiving RT in RT-to-RT transfer, otherwise 0 Same bit pattern as for RTST below
15:8	RT Status Bits (RTST) - Status bits from RT (transmitting RT in RT-to-RT transfer) 15 - Message error, 14 - Instrumentation bit or reserved bit set, 13 - Service request, 12 - Broadcast command received, 11 - Busy bit, 10 - Subsystem flag, 9 - Dynamic bus control acceptance, 8 - Terminal flag
7:4	Retry count (RETCNT) - Number of retries performed
3	Reserved - Mask away on read for forward compatibility
2:0	Transfer status (TFRST) - Outcome of last try 000 - Success (or dummy bit was set) 001 - RT did not respond (transmitting RT in RT-to-RT transfer) 010 - Receiving RT of RT-to-RT transfer did not respond 011 - A responding RT:s status word had message error, busy, instrumentation or reserved bit set (*) 100 - Protocol error (improperly timed data words, decoder error, wrong number of data words) 101 - The transfer descriptor was invalid 110 - Data buffer DMA timeout or error response 111 - Transfer aborted due to loop back check failure

* Error code 011 is issued only when the number of data words match the success case, otherwise code 100 is used. Error code 011 can be issued for a correctly executed “transmit last command” or “transmit last status word” mode code since these commands do not reset the status word.

*Table 308.*GR1553B BC Transfer configuration bits for different transfer types

Transfer type	RTAD1 (15:11)	RTSA1 (9:5)	RTAD2 (25:21)	RTSA2 (20:16)	WCMC (4:0)	TR (10)	Data buffer direction
Data, BC-to-RT	RT address (0-30)	RT subaddr (1-30)	Don’t care	0	Word count (0 for 32)	0	Read (2-64 bytes)
Data, RT-to-BC	RT address (0-30)	RT subaddr (1-30)	Don’t care	0	Word count (0 for 32)	1	Write (2-64 bytes)
Data, RT-to-RT	Recv-RT addr (0-30)	Recv-RT subad. (1-30)	Xmit-RT addr (0-30)	Xmit-RT subad. (1-30)	Word count (0 for 32)	0	Write (2-64 bytes)
Mode, no data	RT address (0-30)	0 or 31 (*)	Don’t care	Don’t care	Mode code (0-8)	1	Unused
Mode, RT-to-BC	RT address (0-30)	0 or 31 (*)	Don’t care	Don’t care	Mode code (16/18/19)	1	Write (2 bytes)
Mode, BC-to-RT	RT address (0-30)	0 or 31 (*)	Don’t care	Don’t care	Mode code (17/20/21)	0	Read (2 bytes)
Broadcast Data, BC-to-RTs	31	RTs subaddr (1-30)	Don’t care	0	Word count (0 for 32)	0	Read (2-64 bytes)
Broadcast Data, RT-to-RTs	31	Recv-RTs subad. (1-30)	Xmit-RT addr (0-30)	Xmit-RT subad. (1-30)	Word count (0 for 32)	0	Write (2-64 bytes)
Broadcast Mode, no data	31	0 or 31 (*)	Don’t care	Don’t care	Mode code (1, 3-8)	1	Unused
Broadcast Mode, BC-to-RT	31	0 or 31 (*)	Don’t care	Don’t care	Mode code (17/20/21)	0	Read (2 bytes)

(*) The standard allows using either of subaddress 0 or 31 for mode commands.

The branch condition word is formed as shown in table 309.

Table 309. GR1553B branch condition word (offset 0x00)

31	30 27	26	25	24	23 16	15 8	7 0
1	Reserved (0)	IRQC	ACT	MODE	RT2CC	RTCC	STCC

31	Must be 1 to identify as branch
30 : 27	Reserved - Set to 0
26	Interrupt if condition met (IRQC)
25	Action (ACT) - What to do if condition is met, 0 - Suspend schedule, 1 - Jump
24	Logic mode (MODE): 0 = Or mode (any bit set in RT2CC, RTCC is set in RT2ST,RTST, or result is in STCC mask) 1 - And mode (all bits set in RT2CC,RTCC are set in RT2ST,RTST and result is in STCC mask)
23:16	RT 2 Condition Code (RT2CC) - Mask with bits corresponding to RT2ST in result word of last transfer
15:8	RT Condition Code (RTCC) - Mask with bits corresponding to RTST in result word of last transfer
7:0	Status Condition Code (STCC) - Mask with bits corresponding to status value of last transfer

Note that you can get a constant true condition by setting MODE=0 and STCC=0xFF, and a constant false condition by setting STCC=0x00. 0x800000FF can thus be used as an end-of-list marker.

33.5 Remote Terminal Operation

33.5.1 Overview

When operating as Remote Terminal, the core acts as a slave on the MIL-STD-1553B bus. It listens for requests to its own RT address (or broadcast transfers), checks whether they are configured as legal and, if legal, performs the corresponding transfer or, if illegal, sets the message error flag in the status word. Legality is controlled by the subaddress control word for data transfers and by the mode code control register for mode codes.

To start the RT, set up the subaddress table and log ring buffer, and then write the address and RT enable bit is into the RT Config Register.

33.5.2 Data transfer handling

The Remote Terminal mode uses a three-level structure to handle data transfer DMA. The top level is a subaddress table, where each subaddress has a subaddress control word, and pointers to a transmit descriptor and a receive descriptor. Each descriptor in turn contains a descriptor control/status word, pointer to a data buffer, and a pointer to a next descriptor, forming a linked list or ring of descriptors. Data buffers can reside anywhere in memory with 16-bit alignment.

When the RT receives a data transfer request, it checks in the subaddress table that the request is legal. If it is legal, the transfer is then performed with DMA to or from the corresponding data buffer. After a data transfer, the descriptor's control/status word is updated with success or failure status and the subaddress table pointer is changed to point to the next descriptor.

If logging is enabled, a log entry will be written into a log ring buffer area. A transfer-triggered IRQ may also be enabled. To identify which transfer caused the interrupt, the RT Event Log IRQ Position points to the corresponding log entry. For that reason, logging must be enabled in order to enable interrupts.

If a request is legal but can not be fulfilled, either because there is no valid descriptor ready or because the data can not be accessed within the required response time, the core will signal a RT table access error interrupt and not respond to the request. Optionally, the terminal flag status bit can be automatically set on these error conditions.

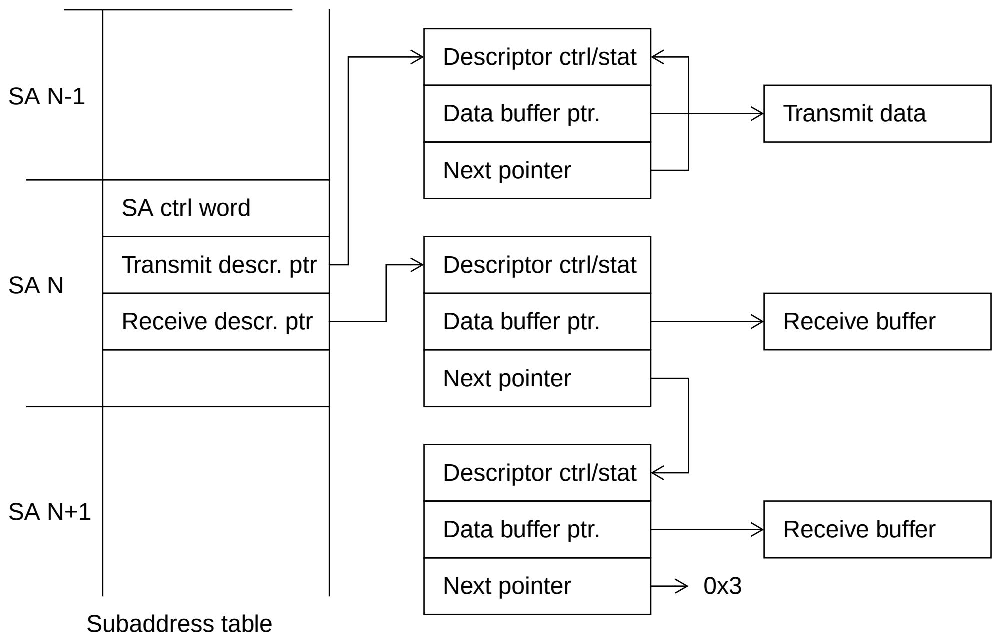


Figure 127. RT subaddress data structure example diagram

33.5.3 Mode Codes

Which of the MIL-STD-1553B mode codes that are legal and should be logged and interrupted are controlled by the RT Mode Code Control register. As for data transfers, to enable interrupts you must also enable logging. Inhibit mode codes are controlled by the same fields as their non-inhibit counterpart and mode codes that can be broadcast have two separate fields to control the broadcast and non-broadcast variants.

The different mode codes and the corresponding action taken by the RT are tabulated below. Some mode codes do not have a built-in action, so they will need to be implemented in software if desired. The relation between each mode code to the fields in the RT Mode Code control register is also shown.

*Table 310.*RT Mode Codes

Mode code		**Description**	**Built-in action, if mode code is enabled**	**Can log/ IRQ**	**Enabled after reset**	**Ctrl. reg bits**
0	00000	Dynamic bus control	If the DBCA bit is set in the RT Bus Status register, a Dynamic Bus Control Acceptance response is sent.	Yes	No	17:16
1	00001	Synchronize	The time field in the RT sync register is updated. The output rtsync is pulsed high one AMBA cycle.	Yes	Yes	3:0
2	00010	Transmit status word	Transmits the RT:s status word Enabled always, can not be logged or disabled.	No	Yes	-
3	00011	Initiate self test	No built-in action	Yes	No	21:18
4	00100	Transmitter shutdown	The RT will stop responding to commands on the other bus (not the bus on which this command was given).	Yes	Yes	11:8
5	00101	Override transmitter shutdown	Removes the effect of an earlier transmitter shutdown mode code received on the same bus	Yes	Yes	11:8
6	00110	Inhibit terminal flag	Masks the terminal flag of the sent RT status words	Yes	No	25:22
7	00111	Override inhibit terminal flag	Removes the effect of an earlier inhibit terminal flag mode code.	Yes	No	25:22
8	01000	Reset remote terminal	The fail-safe timers, transmitter shutdown and inhibit terminal flag inhibit status are reset. The Terminal Flag and Service Request bits in the RT Bus Status register are cleared. The extreset output is pulsed high one AMBA cycle.	Yes	No	29:26
16	10000	Transmit vector word	Responds with vector word from RT Status Words Register	Yes	No	13:12
17	10001	Synchronize with data word	The time and data fields in the RT sync register are updated. The rtsync output is pulsed high one AMBA cycle	Yes	Yes	7:4
18	10010	Transmit last command	Transmits the last command sent to the RT. Enabled always, can not be logged or disabled.	No	Yes	-
19	10011	Transmit BIT word	Responds with BIT word from RT Status Words Register	Yes	No	15:14
20	10100	Selected transmitter shutdown	No built-in action	No	No	-
21	10101	Override selected transmitter shutdown	No built-in action	No	No	-

33.5.4 Event Log

The event log is a ring of 32-bit entries, each entry having the format given in table 311. Note that for data transfers, bits 23-0 in the event log are identical to bits 23-0 in the descriptor status word.

Table 311. GR1553B RT Event Log entry format

31	30 29	28 24	23 10	9	8 3	2 0
IRQSR	TYPE	SAMC	TIMEL	BC	SZ	TRES

31	IRQ Source (IRQSRC) - Set to ‘1’ if this transfer caused an interrupt
30 : 29	Transfer type (TYPE) - 00 - Transmit data, 01 - Receive data, 10 - Mode code
28 : 24	Subaddress / Mode code (SAMC) - If TYPE=00/01 this is the transfer subaddress, If TYPE=10, this is the mode code
23 : 10	TIMEL - Low 14 bits of time tag counter.
9	Broadcast (BC) - Set to 1 if request was to the broadcast address
8 : 3	Transfer size (SZ) - Count in 16-bit words (0-32)
2 : 0	Transfer result (TRES)
	000 = Success
	001 = Superseded (canceled because a new command was given on the other bus)
	010 = DMA error or memory timeout occurred
	011 = Protocol error (improperly timed data words or decoder error)
	100 = The busy bit or message error bit was set in the transmitted status word and no data was sent
	101 = Transfer aborted due to loop back checker error

33.5.5 Subaddress table format

*Table 312.*GR1553B RT Subaddress table entry for subaddress number N, 0<N<31

Offset	**Value**	**DMA R/W**
0x10*N + 0x00	Subaddress N control word (table 313)	R
0x10*N + 0x04	Transmit descriptor pointer, 16-byte aligned (0x3 to indicate invalid pointer)	R/W
0x10*N + 0x08	Receive descriptor pointer, 16-byte aligned (0x3 to indicate invalid pointer)	R/W
0x10*N + 0x0C	Unused	-

Note: The table entries for mode code subaddresses 0 and 31 are never accessed by the core.

Table 313. GR1553B RT Subaddress table control word (offset 0x00)

31 19	18	17	16	15	14	13	12 8	7	6	5	4 0
0 (reserved)	WRAP	IGNDV	BCRXE	RXEN	RXLOG	RXIRQ	RXSZ	TXEN	TXLOG	TXIRQ	TXSZ

31 : 19	Reserved - set to 0 for forward compatibility
18	Auto-wraparound enable (WRAP) - Enables a test mode for this subaddress, where transmit transfers send back the last received data. This is done by copying the finished transfer’s descriptor pointer to the transmit descriptor pointer address after each successful transfer.
	Note: If WRAP=1, you should not set TXSZ > RXSZ as this might cause reading beyond buffer end
17	Ignore data valid bit (IGNDV) - If this is ‘1’ then receive transfers will proceed (and overwrite the buffer) if the receive descriptor has the data valid bit set, instead of not responding to the request.
	This can be used for descriptor rings where you don’t care if the oldest data is overwritten.
16	Broadcast receive enable (BCRXEN) - Allow broadcast receive transfers to this subaddress
15	Receive enable (RXEN) - Allow receive transfers to this subaddress
14	Log receive transfers (RXLOG) - Log all receive transfers in event log ring (only used if RXEN=1)
13	Interrupt on receive transfers (RXIRQ) - Each receive transfer will cause an interrupt (only if also RXEN,RXLOG=1)
12 : 8	Maximum legal receive size (RXSZ) to this subaddress - in16-bit words, 0 means 32
7	Transmit enable (TXEN) - Allow transmit transfers from this subaddress
6	Log transmit transfers (TXLOG) - Log all transmit transfers in event log ring (only if also TXEN=1)
5	Interrupt on transmit transfers (TXIRQ) - Each transmit transfer will cause an interrupt (only if TXEN,TXLOG=1)
4 : 0	Maximum legal transmit size (TXSZ) from this subaddress - in 16-bit words, 0 means 32

*Table 314.*GR1553B RT Descriptor format

Offset	**Value**	**DMA R/W**
0x00	Control and status word, see table 315	R/W
0x04	Data buffer pointer, 16-bit aligned	R
0x08	Pointer to next descriptor, 16-byte aligned or 0x0000003 to indicate end of list	R

Table 315. GR1553B RT Descriptor control/status word (offset 0x00)

31	30	29 - 26	25 - 10	9	8 - 3	2 - 0
DV	IRQEN	Reserved (0)	TIME	BC	SZ	TRES

31	Data valid (DV) - Should be set to 0 by software before and set to 1 by hardware after transfer. If DV=1 in the current receive descriptor before the receive transfer begins then a descriptor table error will be triggered. You can override this by setting the IGNDV bit in the subaddress table.
30	IRQ Enable override (IRQEN) - Log and IRQ after transfer regardless of SA control word settings Can be used for getting an interrupt when nearing the end of a descriptor list.
29 : 26	Reserved - Write 0 and mask out on read for forward compatibility
25 : 10	Transmission time tag (TTIME) - Set by the core to the value of the RT timer when the transfer finished.
9	Broadcast (BC) - Set by the core if the transfer was a broadcast transfer
8 : 3	Transfer size (SZ) - Count in 16-bit words (0-32)
2 : 0	Transfer result (TRES) 000 = Success 001 = Superseded (canceled because a new command was given on the other bus) 010 = DMA error or memory timeout occurred 011 = Protocol error (improperly timed data words or decoder error) 100 = The busy bit or message error bit was set in the transmitted status word and no data was sent 101 = Transfer aborted due to loop back checker error

33.6 Bus Monitor Operation

33.6.1 Overview

The Bus Monitor (BM) can be enabled by itself, or in parallel to the BC or RT. The BM acts as a passive logging device, writing received data with time stamps to a ring buffer.

33.6.2 Filtering

The Bus Monitor can also support filtering. This is an optional feature, software can check for this by testing whether the BM filter registers are writable.

Transfers can be filtered per RT address and per subaddress or mode code, and the filter conditions are logically AND:ed. If all bits of the three filter registers and bits 2-3 of the control register are set to ’1’, the BM core will log all words that are received on the bus.

In order to filter on subaddress/mode code, the BM has logic to track 1553 words belonging to the same message. All 10 message types are supported. If an unexpected word appears, the filter logic will restart. Data words not appearing to belong to any message can be logged by setting a bit in the control register.

The filter logic can be manually restarted by setting the BM enable bit low and then back to high. This feature is mainly to improve testability of the BM itself.

The filtering capability can be configured out of the BM to save area. If this is done, all words seen are logged and the filter control registers become read-only and always read out as all-ones. You can, however, still control whether Manchester/parity errors are logged.

33.6.3 No-response handling

In the MIL-STD-1553B protocol, a command word for a mode code using indicator 0 or a regular transfer to subaddress 8 has the same structure as a legal status word. Therefore ambiguity can arise when the subaddress or mode code filters are used, an RT is not responding on a subaddress, and the BC then commands the same RT again on subaddress 8 or mode code indicator 0 on the same bus. This can lead to the second command word being interpreted as a status word and filtered out.

The BM can use the instrumentation bit and reserved bits to disambiguate, which means that this case will never occur when subaddresses 1-7, 9-30 and mode code indicator 31 are used. Also, this case does not occur when the subaddress/mode code filters are unused and only the RT address filter is used.

33.6.4 Log entry format

Each log entry is two 32-bit words.

Table 316. GR1553B BM Log entry word 0 (offset 0x00)

31	30 24	23 0
1	Reserved	TIME

31	Always written as 1
30 : 24	Reserved - Mask out on read for forward compatibility
23 : 0	Time tag (TIME)

Table 317. GR1553B BM Log entry word 1 (offset 0x04)

31	30 20	19	18 17	16	15 0
0	Reserved	BUS	WST	WTP	WD

31	Always written as 0
30 : 20	Reserved - Mask out on read for forward compatibility

Table 317. GR1553B BM Log entry word 1 (offset 0x04)

19	Receive data bus (BUS) - 0:A, 1:B
18 : 17	Word status (WST) - 00=word OK, 01=Manchester error, 10=Parity error
16	Word type (WTP) - 0:Data, 1:Command/status
15 : 0	Word data (WD)

33.7 Clocking

The core operates in two clock domains, the AMBA clock domain and the 1553 codec clock domain, with synchronization and handshaking between the domains. The AMBA clock can be at any frequency but must be at a minimum of 10 MHz. A propagation delay of up to one codec clock cycle (50 ns) can be tolerated in each clock-domain crossing signal.

The core has two separate reset inputs for the two clock domains. They should be reset simultaneously, for instance by using two Reset generator cores connected to the same reset input but clocked by the respective clocks.

33.8 Registers

The core is programmed through registers mapped into APB address space. If the RT, BC or BM parts of the core have been configured out, the corresponding registers will become unimplemented and return zero when read. Reserved register fields should be written as zeroes and masked out on read.

*Table 318.*MIL-STD-1553B interface registers

APB address offset	Register	R/W	Reset value
0x00	IRQ Register	RW (write ‘1’ to clear)	0x00000000
0x04	IRQ Enable	RW	0x00000000
0x08...0x0F	(Reserved)		
0x10	Hardware config register	R (constant)	0x00000000*
0x14...0x3F	(Reserved)		
0x40...0x7F	BC Register area (see table 319)		
0x80...0xBF	RT Register area (see table 320)		
0xC0...0xFF	BM Register area (see table 321)		

(*) May differ depending on core configuration

*Table 319.*MIL-STD-1553B interface BC-specific registers

APB address offset	Register	R/W	Reset value
0x40	BC Status and Config register	RW	0xf0000000*
0x44	BC Action register	W	
0x48	BC Transfer list next pointer	RW	0x00000000
0x4C	BC Asynchronous list next pointer	RW	0x00000000
0x50	BC Timer register	R	0x00000000
0x54	BC Timer wake-up register	RW	0x00000000
0x58	BC Transfer-triggered IRQ ring position	RW	0x00000000
0x5C	BC Per-RT bus swap register	RW	0x00000000
0x60...0x67	(Reserved)		
0x68	BC Transfer list current slot pointer	R	0x00000000
0x6C	BC Asynchronous list current slot pointer	R	0x00000000
0x70...0x7F	(Reserved)		

(*) May differ depending on core configuration

*Table 320.*MIL-STD-1553B interface RT-specific registers

APB address offset	**Register**	**R/W**	**Reset value**
0x80	RT Status register	R	0x80000000*
0x84	RT Config register	RW	0x0000e03e***
0x88	RT Bus status bits register	RW	0x00000000
0x8C	RT Status words register	RW	0x00000000
0x90	RT Sync register	R	0x00000000
0x94	RT Subaddress table base address	RW	0x00000000
0x98	RT Mode code control register	RW	0x00000555
0x9C...0xA3	(Reserved)		
0xA4	RT Time tag control register	RW	0x00000000
0xA8	(Reserved)		
0xAC	RT Event log size mask	RW	0xfffffffc
0xB0	RT Event log position	RW	0x00000000
0xB4	RT Event log interrupt position	R	0x00000000
0xB8.. 0xBF	(Reserved)		

(*) May differ depending on core configuration

(***) Reset value is affected by the external RTADDR/RTPAR input signals

*Table 321.*MIL-STD-1553B interface BM-specific registers

APB address offset	**Register**	**R/W**	**Reset value**
0xC0	BM Status register	R	0x80000000*
0xC4	BM Control register	RW	0x00000000
0xC8	BM RT Address filter register	RW	0xffffffff
0xCC	BM RT Subaddress filter register	RW	0xffffffff
0xD0	BM RT Mode code filter register	RW	0xffffffff
0xD4	BM Log buffer start	RW	0x00000000
0xD8	BM Log buffer end	RW	0x00000007
0xDC	BM Log buffer position	RW	0x00000000
0xE0	BM Time tag control register	RW	0x00000000
0xE4...0xFF	(Reserved)		

(*) May differ depending on core configuration

Table 322. GR1553B IRQ Register

31 18	17	16	15 11	10	9	8	7 3	2	1	0
RESERVED	BMTOF	BMD	RESERVED	RTTE	RTD	RTEV	RESERVED	BCWK	BCD	BCEV

	Bits read ‘1’ if interrupt occurred, write back ‘1’ to acknowledge
17	BM Timer overflow (BMTOF)
16	BM DMA Error (BMD)
10	RT Table access error (RTTE)
9	RT DMA Error (RTD)
8	RT transfer-triggered event interrupt (RTEV)
2	BC Wake-up timer interrupt (BCWK)
1	BC DMA Error (BCD)
0	BC Transfer-triggered event interrupt (BCEV)

Table 323. GR1553B IRQ Enable Register

31 18	17	16	15 11	10	9	8	7 3	2	1	0
RESERVED	BMTOE	BMDE	RESERVED	RTTEE	RTDE	RTEVE	RESERVED	BCWKE	BCDE	BCEVE

17	BM Timer overflow interrupt enable (BMTOE)
16	BM DMA error interrupt enable (BMDE)
10	RT Table access error interrupt enable (RTTEE)
9	RT DMA error interrupt enable (RTDE)
8	RT Transfer-triggered event interrupt enable (RTEVE)
2	BC Wake up timer interrupt (BCWKE)
1	BC DMA Error Enable (BCDE)
0	BC Transfer-triggered event interrupt (BCEVE)

Table 324. GR1553B Hardware Configuration Register

31	30 12	11	10 9	8	7 0
MOD	RESERVED	XKEYS	ENDIAN	SCLK	CCFREQ

	Note: This register reads 0x0000 for the standard configuration of the core
31	Modified (MOD) - Reserved to indicate that the core has been modified / customized in an unspecified manner
11	Set if safety keys are enabled for the BM Control Register and for all RT Control Register fields.
10 : 9	AHB Endianness - 00=Big-endian, 01=Little-endian, 10/11=Reserved
8	Same clock (SCLK) - Reserved for future versions to indicate that the core has been modified to run with a single clock
7 : 0	Codec clock frequency (CCFREQ) - Reserved for future versions of the core to indicate that the core runs at a different codec clock frequency. Frequency value in MHz, a value of 0 means 20 MHz.

Table 325. GR1553B BC Status and Config Register

31	30 28	27 17	16	15 11	10	9 8	7 3	2 0
BCSUP	BCFEAT	RESERVED	BCCHK	ASADL	0	ASST	SCADL	SCST

31	BC Supported (BCSUP) - Reads ‘1’ if core supports BC mode
30 : 28	BC Features (BCFEAT) - Bit field describing supported optional features (‘1’=supported):
	30 BC Schedule timer supported
	29 BC Schedule time wake-up interrupt supported
	28 BC per-RT bus swap register and STBUS descriptor bit supported
16	Check broadcasts (BCCHK) - Writable bit, if set to ‘1’ enables waiting and checking for (unexpected) responses to all broadcasts.
15 : 11	Asynchronous list address low bits (ASADL) - Bit 8-4 of currently executing (if ASST=01) or next asynchronous command descriptor address
9 : 8	Asynchronous list state (ASST) - 00=Stopped, 01=Executing command, 10=Waiting for time slot
7 : 3	Schedule address low bits (SCADL) - Bit 8-4 of currently executing (if SCST=001) or next schedule descriptor address
2 : 0	Schedule state (SCST) - 000=Stopped, 001=Executing command, 010=Waiting for time slot, 011=Suspended, 100=Waiting for external trigger

Table 326. GR1553B BC Action Register

31 16	15 10	9	8	7 5	4	3	2	1	0
BCKEY	RESERVED	ASSTP	ASSRT	RESERVED	CLRT	SETT	SCSTP	SCSUS	SCSRT

31 : 16	Safety code (BCKEY) - Must be 0x1552 when writing, otherwise register write is ignored
9	Asynchronous list stop (ASSTP) - Write ‘1’ to stop asynchronous list (after current transfer, if executing)
8	Asynchronous list start (ASSRT) - Write ‘1’ to start asynchronous list
4	Clear external trigger (CLRT) - Write ‘1’ to clear trigger memory
3	Set external trigger (SETT) - Write ‘1’ to force the trigger memory to set
2	Schedule stop (SCSTP) - Write ‘1’ to stop schedule (after current transfer, if executing)
1	Schedule suspend (SCSUS) - Write ‘1’ to suspend schedule (after current transfer, if executing)
0	Schedule start (SCSRT) - Write ‘1’ to start schedule

Table 327. GR1553B BC Transfer list next pointer register

31 … 0
SCHEDULE TRANSFER LIST POINTER

31 : 0	Read: Currently executing (if SCST=001) or next transfer to be executed in regular schedule. Write: Change address. If running, this will cause a jump after the current transfer has finished.

Table 328. GR1553B BC Asynchronous list next pointer register

31 … 0
ASYNCHRONOUS LIST POINTER

31 :0	Read: Currently executing (if ASST=01) or next transfer to be executed in asynchronous schedule. Write: Change address. If running, this will cause a jump after the current transfer has finished.

Table 329. GR1553B BC Timer register

31 … 24	23 … 0
RESERVED	SCHEDULE TIME (SCTM)

23 : 0	Elapsed “transfer list” time in microseconds (read-only) Set to zero when schedule is stopped or on external sync.

Note: This register is an optional feature, see BC Status and Config Register, bit 30

Table 330. GR1553B BC Timer Wake-up register

31	30 … 24	23 … 0
WKEN	RESERVED	WAKE-UP TIME (WKTM)

31	Wake-up timer enable (WKEN) - If set, an interrupt will be triggered when WKTM=SCTM
23 : 0	Wake-up time (WKTM).

Note: This register is an optional feature, see BC Status and Config Register, bit 29

Table 331. GR1553B BC Transfer-triggered IRQ ring position register

31 … 0
BC IRQ SOURCE POINTER RING POSITION

31 : 0	The current write pointer into the transfer-tirggered IRQ descriptor pointer ring. Bits 1:0 are constant zero (4-byte aligned) The ring wraps at the 64-byte boundary, so bits 31:6 are only changed by user

Table 332. GR1553B BC per-RT Bus swap register

31 … 0
BC PER-RT BUS SWAP

31 : 0	The bus selection value will be logically exclusive-or:ed with the bit in this mask corresponding to the addressed RT (the receiving RT for RT-to-RT transfers). This register gets updated by the core if the STBUS descriptor bit is used. For more information on how to use this feature, see section 33.4.3.

Note: This register is an optional feature, see BC Status and Config Register, bit 28

Table 333. GR1553B BC Transfer list current slot pointer

31 … 0
BC TRANSFER SLOT POINTER

31 : 0	Points to the transfer descriptor corresponding to the current time slot (read-only, only valid while transfer list is running). Bits 3:0 are constant zero (128-bit/16-byte aligned)

Table 334. GR1553B BC Asynchronous list current slot pointer

31 ... 0
BC TRANSFER SLOT POINTER

31 : 0	Points to the transfer descriptor corresponding to the current asynchronous schedule time slot (read-only, only valid while asynchronous list is running). Bits 3:0 are constant zero (128-bit/16-byte aligned)

Table 335. GR1553B RT Status register (read-only)

31	30 ... 4	3	2	1	0
RTSUP	RESERVED	ACT	SHDA	SHDB	RUN

31	RT Supported (RTSUP) - Reads '1' if core supports RT mode
3	RT Active (ACT) - '1' if RT is currently processing a transfer
2	Bus A shutdown (SHDA) - Reads '1' if bus A has been shut down by the BC (using the transmitter shutdown mode command on bus B)
1	Bus B shutdown (SHDB) - Reads '1' if bus B has been shut down by the BC (using the transmitter shutdown mode command on bus A)
0	RT Running (RUN) - '1' if the RT is listening to commands.

Table 336. GR1553B RT Config register

31 ... 16	15	14	13	12 ... 7	6	5 ... 1	0
RTKEY	SYS	SYDS	BRS	RESERVED	RTEIS	RTADDR	RTEN

31 : 16	Safety code (RTKEY) - Must be written as 0x1553 when changing the RT address, otherwise the address field is unaffected by the write. When reading the register, this field reads 0x0000. If extra safety keys are enabled (see Hardware Config Register), the lower half of the key is used to also protect the other fields in this register.
15	Sync signal enable (SYS) - Set to '1' to pulse the rtsync output when a synchronize mode code (without data) has been received
14	Sync with data signal enable (SYDS) - Set to '1' to pulse the rtsync output when a synchronize with data word mode code has been received
13	Bus reset signal enable (BRS) - Set to '1' to pulse the busreset output when a reset remote terminal mode code has been received.
6	Reads '1' if current address was set through external inputs. After setting the address from software this field is set to '0'
5 : 1	RT Address (RTADDR) - This RT:s address (0-30)
0	RT Enable (RTEN) - Set to '1' to enable listening for requests

Table 337. GR1553B RT Bus status register

31 ... 9	8	7 ... 5	4	3	2	1	0
RESERVED	TFDE	RESERVED	SREQ	BUSY	SSF	DBCA	TFLG

8	Set Terminal flag automatically on DMA and descriptor table errors (TFDE)
4 : 0	These bits will be sent in the RT:s status responses over the 1553 bus.
4	Service request (SREQ)
3	Busy bit (BUSY) Note: If the busy bit is set, the RT will respond with only the status word and the transfer "fails"
2	Subsystem Flag (SSF)
1	Dynamic Bus Control Acceptance (DBCA) Note: This bit is only sent in response to the Dynamic Bus Control mode code
0	Terminal Flag (TFLG) The BC can mask this flag using the "inhibit terminal flag" mode command, if legal

Table 338. GR1553B RT Status words register

31 ... 16	15 ... 0
BIT WORD (BITW)	VECTOR WORD (VECW)

31 : 16	BIT Word - Transmitted in response to the "Transmit BIT Word" mode command, if legal
15 : 0	Vector word - Transmitted in response to the "Transmit vector word" mode command, if legal.

Table 339. GR1553B RT Sync register

31 16	15 0
SYNC TIME (SYTM)	SYNC DATA (SYD)

31 : 16	The value of the RT timer at the last sync or sync with data word mode command, if legal.
15 : 0	The data received with the last synchronize with data word mode command, if legal

Table 340. GR1553B RT Subaddress table base address register

31 9	8 0
SUBADDRESS TABLE BASE (SATB)	0

31 : 9	Base address, bits 31-9 for subaddress table
8 : 0	Always read ‘0’, writing has no effect

Table 341. GR1553B RT Mode code control register

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16
RESERVED	RRTB	RRT	ITFB	ITF	ISTB	IST	DBC

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
TBW	TVW	TSB	TS	SDB	SD	SB	S

	For each mode code: “00” - Illegal, “01” - Legal, “10” - Legal, log enabled, “11” - Legal, log and interrupt
29 : 28	Reset remote terminal broadcast (RRTB)
27 : 26	Reset remote terminal (RRT)
25 : 24	Inhibit & override inhibit terminal flag bit broadcast (ITFB)
23 : 22	Inhibit & override inhibit terminal flag (ITF)
21 : 20	Initiate self test broadcast (ISTB)
19 : 18	Initiate self test (IST)
17 : 16	Dynamic bus control (DBC)
15 : 14	Transmit BIT word (TBW)
13 : 12	Transmit vector word (TVW)
11 : 10	Transmitter shutdown & override transmitter shutdown broadcast (TSB)
9 : 8	Transmitter shutdown & override transmitter shutdown (TS)
7 : 6	Synchronize with data word broadcast (SDB)
5 : 4	Synchronize with data word (SD)
3 : 2	Synchronize broadcast (SB)
1 : 0	Synchronize (S)

Table 342. GR1553B RT Time tag control register

31 16	15 0
TIME RESOLUTION (TRES)	TIME TAG VALUE (TVAL)

31 : 16	Time tag resolution (TRES) - Time unit of RT:s time tag counter in microseconds, minus 1
15 : 0	Time tag value (TVAL) - Current value of running time tag counter

Table 343. GR1553B RT Event Log mask register

31 21	20 2	1 0
1	EVENT LOG SIZE MASK	0

31 : 0	Mask determining size and alignment of the RT event log ring buffer. All bits “above” the size should be set to ‘1’, all bits below should be set to ‘0’

Table 344. GR1553B RT Event Log position register

31 0
EVENT LOG WRITE POINTER

31 : 0	Address to first unused/oldest entry of event log buffer, 32-bit aligned

Table 345. GR1553B RT Event Log interrupt position register

31 … 0
EVENT LOG IRQ POINTER

31 : 0	Address to event log entry corresponding to interrupt, 32-bit aligned The register is set for the first interrupt and not set again until the interrupt has been acknowledged.

Table 346. GR1553B BM Status register

31	30	29 … 0
BMSUP	KEYEN	RESERVED

31	BM Supported (BMSUP) - Reads ‘1’ if BM support is in the core.
30	Key Enabled (KEYEN) - Reads ‘1’ if the BM validates the BMKEY field when the control register is written.

Table 347. GR1553B BM Control register

31 … 16	15 … 6	5	4	3	2	1	0
BMKEY	RESERVED	WRSTP	EXST	IMCL	UDWL	MANL	BMEN

31 : 16	Safety key - If extra safety keys are enabled (see KEYEN), this field must be 0x1543 for a write to be accepted. Is 0x0000 when read.
5	Wrap stop (WRSTP) - If set to ‘1’, BMEN will be set to ‘0’ and stop the BM when the BM log position wraps around from buffer end to buffer start
4	External sync start (EXST) - If set to ‘1’,BMEN will be set to ‘1’ and the BM is started when an external BC sync pulse is received
3	Invalid mode code log (IMCL) - Set to ‘1’ to log invalid or reserved mode codes.
2	Unexpected data word logging (UDWL) - Set to ‘1’ to log data words not seeming to be part of any command
1	Manchester/parity error logging (MANL) - Set to ‘1’ to log bit decoding errors
0	BM Enable (BMEN) - Must be set to ‘1’ to enable any BM logging

Table 348. GR1553B BM RT Address filter register

31 … 0
ADDRESS FILTER MASK

31	Enables logging of broadcast transfers
30 : 0	Each bit position set to ‘1’ enables logging of transfers with the corresponding RT address

Table 349. GR1553B BM RT Subaddress filter register

31 … 0
SUBADDRESS FILTER MASK

31	Enables logging of mode commands on subaddress 31
30 : 1	Each bit position set to ‘1’ enables logging of transfers with the corresponding RT subaddress
0	Enables logging of mode commands on subaddress 0

Table 350. GR1553B BM RT Mode code filter register

31–19	18	17	16
RESERVED	STSB	STS	TLC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSW	RRTB	RRT	ITFB	ITF	ISTB	IST	DBC	TBW	TVW	TSB	TS	SDB	SD	SB	S

	Each bit set to ‘1’ enables logging of a mode code:
18	Selected transmitter shutdown broadcast & override selected transmitter shutdown broadcast (STSB)
17	Selected transmitter shutdown & override selected transmitter shutdown (STS)
16	Transmit last command (TLC)
15	Transmit status word (TSW)
14	Reset remote terminal broadcast (RRTB)
13	Reset remote terminal (RRT)
12	Inhibit & override inhibit terminal flag bit broadcast (ITFB)
11	Inhibit & override inhibit terminal flag (ITF)
10	Initiate self test broadcast (ISTB)
9	Initiate self test (IST)
8	Dynamic bus control (DBC)
7	Transmit BIT word (TBW)
6	Transmit vector word (TVW)
5	Transmitter shutdown & override transmitter shutdown broadcast (TSB)
4	Transmitter shutdown & override transmitter shutdown (TS)
3	Synchronize with data word broadcast (SDB)
2	Synchronize with data word (SD)
1	Synchronize broadcast (SB)
0	Synchronize (S)

Table 351. GR1553B BM Log buffer start

31–0
BM LOG BUFFER START

31 : 0	Pointer to the lowest address of the BM log buffer (8-byte aligned) Due to alignment, bits 2:0 are always 0.

Table 352. GR1553B BM Log buffer end

31–22	21–0
	BM LOG BUFFER END

31 : 0	Pointer to the highest address of the BM log buffer Only bits 21:3 are settable, i.e. the buffer can not cross a 4 MB boundary Bits 31:22 read the same as the buffer start address.Due to alignment, bits 2:0 are always equal to 1

Table 353. GR1553B BM Log buffer position

31–22	21–0
	BM LOG BUFFER POSITION

31 : 0	Pointer to the next position that will be written to in the BM log buffer Only bits 21:3 are settable, i.e. the buffer can not cross a 4 MB boundary Bits 31:22 read the same as the buffer start address.Due to alignment, bits 2:0 are always equal to 0

Table 354. GR1553B BM Time tag control register

31–24	23–0
TIME TAG RESOLUTION	TIME TAG VALUE

31 : 24	Time tag resolution (TRES) - Time unit of BM:s time tag counter in microseconds, minus 1
23 : 0	Time tag value (TVAL) - Current value of running time tag counter

33.9 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x04D. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

33.10 Configuration options

Table 355 shows the configuration options of the core (VHDL generics).

*Table 355.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Index of the interrupt line.	0 - NAHBIRQ-1	0
bc_enable	Selects whether BC support is built into the core	0 - 1	1
rt_enable	Selects whether RT support is built into the core	0 - 1	1
bm_enable	Selects whether BM support is built into the core	0 - 1	1
bc_timer	Selects whether the BC timer and wake-up interrupt features are built into the core. 0=None, 1=Timer, 2=Timer and wake-up	0-2	1
bc_rtbusmask	Selects whether the BC per-RT bus swap register is built into the core.	0-1	1
extra_regkeys	Enables extra safety keys for the BM control register and for all fields in the RT control registers	0-1	0
syncrst	Selects reset configuration: 0: Asynchronous reset, all registers in core are reset 1: Synchronous, minimal set of registers are reset 2: Synchronous, most registers reset (increases area slightly to simplify netlist simulation)	0-2	1
ahbendian	Selects AHB bus endianness (for use in non-GRLIB systems), 0=Big endian, 1=Little endian	0 - 1	0
bm_filters	Enable BM filtering capability	0 - 1	1
codecfreq	Codec clock domain frequency in MHz	20 or 24	20
sameclk	AMBA clock and reset is same as codec (removes internal synchronization)	0 - 1	0

33.11 Signal descriptions

Tables 356-357 shows the interface signals of the core (VHDL ports).

*Table 356.*Signal descriptions on AMBA side

Signal name	**Field**	**Type**	**Function**	**Active**
CLK	N/A	Input	Clock, AMBA clock domain	-
RST	N/A	Input	Reset for registers in CLK clock domain	Low
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBSI	*	Input	APB slave input signals	-
APBSO	*	Output	APB slave output signals	-
AUXIN	EXTSYNC	Input	External sync input for Bus Controller Re-synchronized to AMBA clk internally. Edge-detection checks for the sampled pattern “01”, i.e. pulses should be at least one CLK cycle to always get detected.	Pos. edge
	RTADDR	Input	Reset value for RT address, if parity matches.	-
	RTPAR	Input	RT address odd parity	-
AUXOUT	RTSYNC	Output	Pulsed for one CLK cycle after receiving a synchronize mode command in RT mode	High
	BUSRESET	Output	Pulsed for one CLK cycle after receiving a reset remote terminal mode command in RT mode	High
	VALIDCMDA	Output	Pulsed for one CLK cycle after receiving a valid command word on bus A/B in RT mode	High
	VALIDCMDB	Output		High
	TIMEDOUTA	Output	Asserted when the terminal fail-safe timer has triggered on bus A/B.	High
	TIMEDOUTB	Output		High
	BADREG	Output	Pulsed for one CLK cycle when an invalid register access is performed, either: - an access to an undefined register, - read/write from a write-only/read-only register, - a read/write to a non-implemented part of the core - an incorrect BCKEY/BMKEY	High
	IRQVEC	Output	Auxiliary IRQ vector. Pulsed at the same time as the ordinary PIRQ line, but with a separate line for each interrupt: 7: BM Timer overflow, 6: BM DMA Error, 5: RT Table error, 4: RT DMA Error, 3: RT Event 2: BC Wake-up, 1: BC DMA Error, 0: BC Event	High

* see GRLIB IP Library User’s Manual

*Table 357.*Signal descriptions on 1553 side

Signal name	**Field**	**Type**	**Function**	**Active**
CODEC_CLK	N/A	Input	Codec clock	-
CODEC_RST	N/A	Input	Reset for registers in CODEC_CLK domain	Low
TXOUT	BUSA_TXP	Output	Bus A transmitter, positive output	High **
	BUSA_TXN	Output	Bus A transmitter, negative output	High **
	BUSA_TXEN	Output	Bus A transmitter enable	High
	BUSA_RXEN	Output	Bus A receiver enable	High
	BUSB_TXP	Output	Bus B transmitter, positive output	High **
	BUSB_TXN	Output	Bus B transmitter, negative output	High **
	BUSB_TXEN	Output	Bus B transmitter enable	High
	BUSB_RXEN	Output	Bus B receiver enable	High
	BUSA_TXIN	Output	Inverted version of BUSA_TXEN (for VHDL coding convenience)	High
	BUSB_TXIN	Output	Inverted version of BUSB_TXEN	High
TXOUT_FB	See TXOUT	Input	Feedback input to the terminal fail-safe timers. Should be tied directly to TXOUT, but are exposed to allow testing the fail-safe timer function. This input is re synchronized to CODEC_CLK so it can be asynchronous.	See TXOUT
RXIN	BUSA_RXP	Input	Bus A receiver, positive input	High **
	BUSA_RXN	Input	Bus A receiver, negative input	High **
	BUSB_RXP	Input	Bus B receiver, positive input	High **
	BUSB_RXN	Input	Bus B receiver, negative input	High **

** The core will put both P/N outputs low when not transmitting. For input, it accepts either both-low or both-high idle.

33.12 Library dependencies

Table 358 shows libraries used when instantiating the core (VHDL libraries).

*Table 358.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AHB/APB signal definitions
GAISLER	GR1553B_PKG	Signals, component	signal and component declaration

33.13 Instantiation

This example shows how the core can be instantiated in a GRLIB design.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, gaisler;
use grlib.amba.all;
use gaisler.gr1553b_pkg.all;
use gaisler.misc.rstgen;

entity gr1553b_ex is
  generic (
    padtech     : integer
    );
  port (
    rstn        : in std_ulogic;
    clk         : in std_ulogic;
    codec_clk   : in std_ulogic;

    -- MIL-STD-1553 signals
    txAen       : out std_ulogic;
    txAP        : out std_ulogic;
    txAN        : out std_ulogic;
    rxAen       : out std_ulogic;
    rxAP        : in std_ulogic;
    rxAN        : in std_ulogic;
    txAen       : out std_ulogic;
    txAP        : out std_ulogic;
    txAN        : out std_ulogic;
    rxAen       : out std_ulogic;
    rxAP        : in std_ulogic;
    rxAN        : in std_ulogic
    );
end;

architecture rtl of gr1553b_ex is

  -- System-wide synchronous reset
  signal rst       : std_logic;

  -- AMBA signals
  signal apbi      : apb_slv_in_type;
  signal apbo      : apb_slv_out_vector := (others => apb_none);
  signal ahbi      : ahb_mst_in_type;
  signal ahbo      : ahb_mst_out_vector := (others => apb_none);

  -- GR1553B signals
  signal codec_rst : std_ulogic;
  signal txout     : gr1553b_txout_type;
  signal rxin      : gr1553b_rxin_type;
  signal auxin     : gr1553b_auxin_type;
  signal auxout    : gr1553b_auxout_type;

begin

  rg0: rstgen port map (rstn, clk, ’1’, rst, open);

  -- AMBA Components are instantiated here
  ...

  -- Reset generation for 1553 codec
  rgc: rstgen port map (rstn, codec_clk, ’1’, codec_rst, open);

  -- GR1553B

  gr1553b0: gr1553b
  generic map (hindex => 4, pindex => 7, paddr => 7, pirq => 13, syncrst => 1,
               bc_enable => 1, rt_enable => 1, bm_enable => 1)
  port map (clk, rst, ahbi, ahbo(4), apbi, apbo(7), auxin, auxout,
            codec_clk, codec_rst, txout, txout, rxin);

  p: gr1553b_pads
  generic map (padtech => padtech, outen_pol => 0)
  port map (txout,rxin,
            rxAen,rxAP,rxAN,txAen,txAP,txAN,
            rxBen,rxBP,rxBN,txBen,txBP,txBN);

  auxin         <= gr1553b_auxin_zero;

end;
```

34 GPTIMER - General Purpose Timer Unit

34.1 Overview

The General Purpose Timer Unit provides a common prescaler and decrementing timer(s). The number of timers is configurable through the *ntimers* VHDL generic in the range 1 to 7. The prescaler width is configured through the *sbits* VHDL generic. Timer width is configured through the *tbits* VHDL generic. The timer unit acts a slave on AMBA APB bus. The unit is capable of asserting interrupts on timer underflow. The interrupt to use is configurable to be common for the whole unit or separate for each timer.

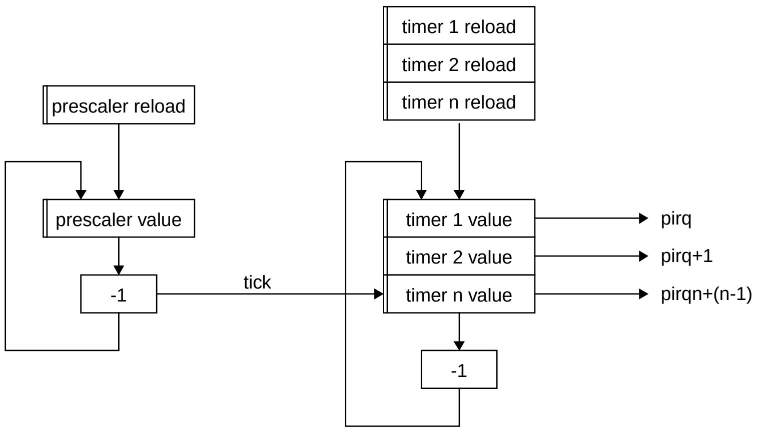


Figure 128. General Purpose Timer Unit block diagram

34.2 Operation

The prescaler is clocked by the system clock and decremented on each clock cycle. When the prescaler underflows, it is reloaded from the prescaler reload register and a timer tick is generated.

The operation of each timers is controlled through its control register. A timer is enabled by setting the enable bit in the control register. The timer value is then decremented on each prescaler tick. When a timer underflows, it will automatically be reloaded with the value of the corresponding timer reload register if the restart bit in the control register is set, otherwise it will stop at -1 and reset the enable bit.

The timer unit can be configured to generate common interrupt through a VHDL-generic. The shared interrupt will be signalled when any of the timers with interrupt enable bit underflows. The timer unit will signal an interrupt on appropriate line when a timer underflows (if the interrupt enable bit for the current timer is set), when configured to signal interrupt for each timer. The interrupt pending bit in the control register of the underflown timer will be set and remain set until cleared by writing ‘1’.

To minimize complexity, timers share the same decrementer. This means that the minimum allowed prescaler division factor is *ntimers*+1 (reload register = *ntimers*) where *ntimers* is the number of implemented timers. By setting the chain bit in the control register timer *n* can be chained with preceding timer *n*-1. Timer *n* will be decremented each time when timer *n*-1 underflows.

Each timer can be reloaded with the value in its reload register at any time by writing a ‘one’ to the load bit in the control register. The last timer acts as a watchdog, driving a watchdog output signal when expired, when the *wdog* VHDL generic is set to a time-out value larger than 0.

At reset, all timer are disabled except the watchdog timer (if enabled by the generics). The prescaler value and reload registers are set to all ones, while the watchdog timer is set to the *wdog* VHDL generic. All other registers are uninitialized

34.3 Registers

The core is programmed through registers mapped into APB address space. The number of implemented registers depend on the number of implemented timers.

*Table 359.*General Purpose Timer Unit registers

APB address offset	**Register**
0x00	Scaler value
0x04	Scaler reload value
0x08	Configuration register
0x0C	Unused
0x10	Timer 1 counter value register
0x14	Timer 1 reload value register
0x18	Timer 1 control register
0x1C	Unused
0x*n*0	Timer *n* counter value register
0x*n*4	Timer *n* reload value register
0x*n*8	Timer *n* control register

Table 360. Scaler value

31 16	16-1 0
“000..0”	SCALER VALUE

16-1: 0 Scaler value

Any unused most significant bits are reserved. Always reads as ‘000...0’.

Table 361. Scaler reload value

31 16	16-1 0
“000..0”	SCALER RELOAD VALUE

16-1: 0 Scaler reload value

Any unused most significant bits are reserved. Always read as ‘000...0’.

Table 362. Configuration Register

31 10	9	8	7 3	2 0
“000..0”	DF	SI	IRQ	TIMERS

31: 10	Reserved. Always reads as ‘000...0’.
9	Disable timer freeze (DF). If set the timer unit can not be freezed, otherwise signal GPTI.DHALT freezes the timer unit.
8	Separate interrupts (SI). Reads ‘1’ if the timer unit generates separate interrupts for each timer, otherwise ‘0’. Read-only.
7: 3	APB Interrupt: If configured to use common interrupt all timers will drive APB interrupt nr. IRQ, otherwise timer *n*will drive APB Interrupt IRQ+*n* (has to be less the MAXIRQ). Read-only.
2: 0	Number of implemented timers. Read-only.

Table 363. Timer counter value register

32-1 0
TIMER COUNTER VALUE

Table 363. Timer counter value register

32-1: 0	Timer Counter value. Decremented by 1 for each prescaler tick. Any unused most significant bits are reserved. Always reads as ‘000...0’.

Table 364. Timer reload value register

32-1 … 0
TIMER RELOAD VALUE

32-1: 0	Timer Reload value. This value is loaded into the timer counter value register when ‘1’ is written to load bit in the timers control register or when the RS bit is set in the control register and the timer underflows. Any unused most significant bits are reserved. Always reads as ‘000...0’.

Table 365. Timer control register

31 … 7	6	5	4	3	2	1	0
“000..0”	DH	CH	IP	IE	LD	RS	EN

31: 7	Reserved. Always reads as ‘000...0’.
6	Debug Halt (DH): Value of GPTI.DHALT signal which is used to freeze counters (e.g. when a system is in debug mode). Read-only.
5	Chain (CH): Chain with preceding timer. If set for timer *n*, timer *n* will be decremented each time when timer (*n*-1) underflows.
4	Interrupt Pending (IP): The core sets this bit to ‘1’ when an interrupt is signalled. This bit remains ‘1’ until cleared by writing ‘1’ to this bit, writes of ‘0’ have no effect.
3	Interrupt Enable (IE): If set the timer signals interrupt when it underflows.
2	Load (LD): Load value from the timer reload register to the timer counter value register.
1	Restart (RS): If set, the timer counter value register is reloaded with the value of the reload register when the timer underflows
0	Enable (EN): Enable the timer.

34.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x011. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

34.5 Configuration options

Table 366 shows the configuration options of the core (VHDL generics).

*Table 366.*Configuration options

Generic	Function	Allowed range	Default
pindex	Selects which APB select signal (PSEL) will be used to access the timer unit	0 to NAPBSLV-1	0
paddr	The 12-bit MSB APB address	0 to 4095	0
pmask	The APB address mask	0 to 4095	4095
nbits	Defines the number of bits in the timers	1 to 32	32
ntimers	Defines the number of timers in the unit	1 to 7	1
pirq	Defines which APB interrupt the timers will generate	0 to NAHBIRQ-1	0
sepirq	If set to 1, each timer will drive an individual interrupt line, starting with interrupt *pirq*. If set to 0, all timers will drive the same interrupt line (*pirq*).	0 to 1 (note: *ntimers* + p*irq* must be less than or equal to NAHBIRQ if *sepirq* is set to 1)	0
sbits	Defines the number of bits in the scaler	1 to 32	16
wdog	Watchdog reset value. When set to a non-zero value, the last timer will be enabled and pre-loaded with this value at reset. When the timer value reaches 0, the WDOG output is driven active.	0 to 2^{nbits} - 1	0
ewdogen	External watchdog enable. When set to a non-zero value, the enable bit of the watchdog timer will be set during core reset via the signal gpti.wdogen.Otherwise the enable bit will be set to ‘1’ during core reset.	0 - 1	0

34.6 Signal descriptions

Table 367 shows the interface signals of the core (VHDL ports).

*Table 367.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GPTI	DHALT	Input	Freeze timers	High
	EXTCLK	Input	Use as alternative clock	-
	WDOGEN	Input	Sets enable bit of the watchdog timer if VHDL generics wdog and ewdogen are set to non-zero values.	-
GPTO	TICK[0:7]	Output	Timer ticks. TICK[0] is high for one clock each time the scaler underflows. TICK[1-n] are high for one clock each time the corresponding timer underflows.	High
	WDOG	Output	Watchdog output. Equivalent to interrupt pending bit of last timer.	High
	WDOGN	Output	Watchdog output Equivalent to interrupt pending bit of last timer.	Low

* see GRLIB IP Library User’s Manual

34.7 Library dependencies

Table 368 shows libraries used when instantiating the core (VHDL libraries).

*Table 368.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Signals, component	Component declaration

34.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

entity gptimer_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ... -- other signals
    );
end;

architecture rtl of gptimer_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- GP Timer Unit input signals
  signal gpti : gptimer_in_type;

begin

  -- AMBA Components are instantiated here
  ...


  -- General Purpose Timer Unit
  timer0 : gptimer
  generic map (pindex => 3, paddr => 3, pirq => 8, sepirq => 1)
  port map (rstn, clk, apbi, apbo(3), gpti, open);

  gpti.dhalt <= ’0’; gpti.extclk <= ’0’; -- unused inputs

end;
```

35 GRTIMER - General Purpose Timer Unit

35.1 Overview

The General Purpose Timer Unit provides a common prescaler and decrementing timer(s). Number of timers is configurable through the *ntimers* VHDL generic in the range 1 to 7. Prescaler width is configured through the sbits VHDL generic. Timer width is configured through the *tbits* VHDL generic. The timer unit acts a slave on AMBA APB bus. The unit implements one 16 bit prescaler and 3 decrementing 32 bit timer(s). The unit is capable of asserting interrupt on timer(s) underflow. Interrupt is configurable to be common for the whole unit or separate for each timer.

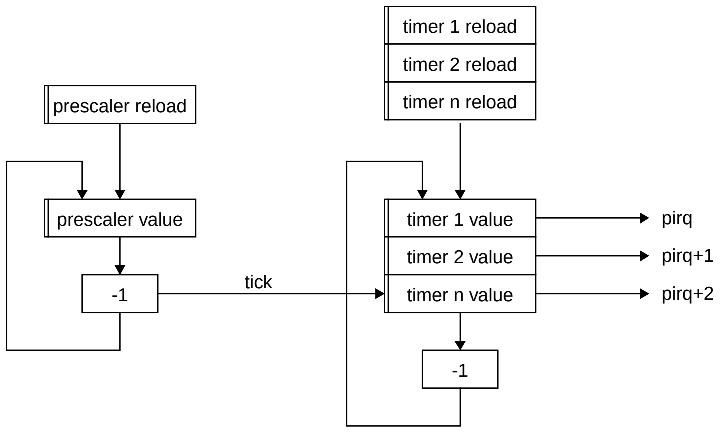


Figure 129. General Purpose Timer Unit block diagram

35.2 Operation

The prescaler is clocked by the system clock and decremented on each clock cycle. When the prescaler underflows, it is reloaded from the prescaler reload register and a timer tick is generated. Timers share the decrementer to save area.

The operation of each timers is controlled through its control register. A timer is enabled by setting the enable bit in the control register. The timer value is then decremented on each prescaler tick. When a timer underflows, it will automatically be reloaded with the value of the corresponding timer reload register if the restart bit in the control register is set, otherwise it will stop at -1 and reset the enable bit.

The timer unit can be configured to generate common interrupt through a VHDL generic. The shared interrupt will be signalled when any of the timers with interrupt enable bit underflows. If configured to signal interrupt for each timer the timer unit will signal an interrupt on appropriate line when a timer underflows (if the interrupt enable bit for the current timer is set). The interrupt pending bit in the control register of the underflown timer will be set and remain set until cleared by writing ‘1’.

To minimize complexity, timers share the same decrementer. This means that the minimum allowed prescaler division factor is *ntimers*+1 (reload register = *ntimers*) where *ntimers* is the number of implemented timers.

By setting the chain bit in the control register timer *n* can be chained with preceding timer *n*-1. Timer *n* will be decremented each time when timer *n*-1 underflows.

Each timer can be reloaded with the value in its reload register at any time by writing a ‘one’ to the load bit in the control register.

Each timers can to latch its value to dedicated registers on an event detected on the AMBA APB sideband interrupt bus signal. A dedicated mask register is provided to filter the interrupts. (In revision 1 of the core there was a possibility that the timers were in the middle of a decrement when the latching interrupt arrived, resulting in inconsistent timer values when used in chained configuration. This has been improved in revision 2.)

All timers can be forced to reload an event detected on the AMBA APB sideband interrupt bus signal.

The above mask register is provided to filter the interrupts.

35.3 Registers

The core is programmed through registers mapped into APB address space. The number of implemented registers depend on number of implemented timers.

*Table 369.*GRTIMER unit registers

APB address offset	**Register**
0x00	Scaler value
0x04	Scaler reload value
0x08	Configuration register
0x0C	Timer latch configuration register
0x10	Timer 1 counter value register
0x14	Timer 1 reload value register
0x18	Timer 1 control register
0x1C	Timer 1 latch register
0x*n*0	Timer *n* counter value register
0x*n*4	Timer *n* reload value register
0x*n*8	Timer *n* control register
0x*n*C	Timer *n* latch register

Figures 130 to 135 shows the layout of the timer unit registers.

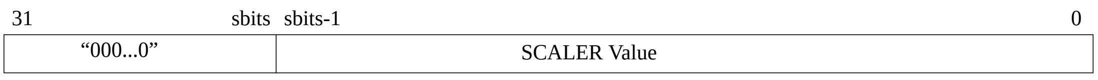


Figure 130. Scaler value

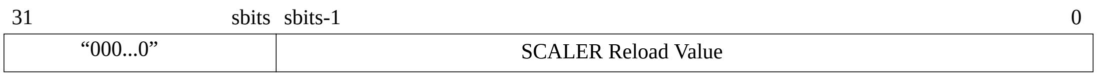


Figure 131. Scaler reload value

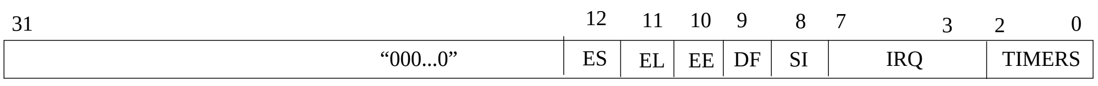


Figure 132. GRTIMER Configuration register

[31:13] - Reserved.

[12] Enable set (ES). If set, on the next matching interrupt, the timers will be loaded with the corresponding timer reload values. The bit is then automatically cleared, not to reload the timer values until set again. (Added to revision 2).

[11] Enable latching (EL). If set, on the next matching interrupt, the latches will be loaded with the corresponding timer values. The bit is then automatically cleared, not to load a timer value until set again.

[10] Enable external clock source (EE). If set the prescaler is clocked from the external clock source.
[9] Disable timer freeze (DF). If set the timer unit can not be freezed, otherwise signal GPTI.DHALT freezes the timer unit.
[8] Separate interrupts (SI). Reads ‘1’ if the timer unit generates separate interrupts for each timer, otherwise ‘0’. Read-only.
[7:3] APB Interrupt: If configured to use common interrupt all timers will drive APB interrupt nr. IRQ, otherwise timer *n*will drive APB Interrupt IRQ+*n* (has to be less the MAXIRQ). Read-only.
[2:0] Number of implemented timers. Read-only.

31 nbits	nbits-1 0
“000...0”	TIMER COUNTER VALUE

Figure 133. Timer counter value registers

[31:nbits] Reserved. Always reads as ‘000...0’
[nbits-1:0] Timer Counter value. Decremented by 1 for each prescaler tick.

31 nbits	nbits-1 0
“000...0”	TIMER RELOAD VALUE

Figure 134. Timer reload value registers

[31:nbits] Reserved. Always reads as ‘000...0’
[nbits-1:0] Timer Reload value. This value is loaded into the timer counter value register when ‘1’ is written to load bit in the timers control register.

31 7	6	5	4	3	2	1	0
“000...0”	DH	CH	IP	IE	LD	RS	EN

Figure 135. Timer control registers

[31:7] Reserved. Always reads as ‘000...0’
[6] Debug Halt (DH): Value of GPTI.DHALT signal which is used to freeze counters (e.g. when a system is in debug mode). Read-only.
[5] Chain (CH): Chain with preceding timer. If set for timer *n*, timer *n* will be decremented each time when timer (*n*-1) underflows.
[4] Interrupt Pending (IP): The core sets this bit to ‘1’ when an interrupt is signalled. This bit remains ‘1’ until cleared by writing ‘1’ to this bit, writes of ‘0’ have no effect.
[3] Interrupt Enable (IE): If set the timer signals interrupt when it underflows.
[2] Load (LD): Load value from the timer reload register to the timer counter value register.
[1] Restart (RS): If set the value from the timer reload register is loaded to the timer counter value register and decrementing the timer is restarted.
[0] Enable (EN): Enable the timer.

31 0
SELECT

Figure 136. Timer latch configuration register

[31:0] Specifies what bits of the AMBA APB interrupt bus shall cause the Timer Latch Register to latch the timer values.

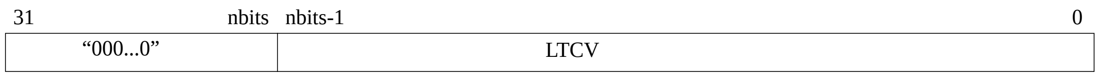


Figure 137. Timer latch register

[31:nbits] Reserved. Always reads as ‘000...0’
[nbits-1:0] Latched Timer Counter Value (LTCV). Value latch from corresponding timer.

35.4 Vendor and device identifiers

The module has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x038. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

35.5 Configuration options

Table 370 shows the configuration options of the core (VHDL generics).

*Table 370.*Configuration options

Generic	**Function**	**Allowed range**	**Default**
pindex	Selects which APB select signal (PSEL) will be used to access the timer unit	0 to NAPBMAX-1	0
paddr	The 12-bit MSB APB address	0 to 4095	0
pmask	The APB address mask	0 to 4095	4095
nbits	Defines the number of bits in the timers	1 to 32	32
ntimers	Defines the number of timers in the unit	1 to 7	1
pirq	Defines which APB interrupt the timers will generate	0 to NAHBIRQ-1	0
sepirq	If set to 1, each timer will drive an individual interrupt line, starting with interrupt *irq*. If set to 0, all timers will drive the same interrupt line (*irq*).	0 to NAHBIRQ-1 (Note: *ntimers* + *irq* must be less than or equal to NAHBIRQ)	0
sbits	Defines the number of bits in the scaler	1 to 32	16
wdog	Watchdog reset value. When set to a non-zero value, the last timer will be enabled and pre-loaded with this value at reset. When the timer value reaches 0, the WDOG output is driven active.	0 to 2^{nbits} - 1	0
glatch	Enable external timer latch (via interrupt)	0 to 1	0
gextclk	Enable external timer clock input	0 to 1	0
gset	Enable external timer reload (via interrupt)	0 to 1	0

35.6 Signal descriptions

Table 371 shows the interface signals of the core (VHDL ports).

*Table 371.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GPTI	DHALT	Input	Freeze timers	High
	EXTCLK	Input	Use as alternative clock	-
GPTO	TICK[1:7]	Output	Timer ticks	High
	WDOG	Output	Watchdog output. Equivalent to interrupt pending bit of last timer.	High
	WDOGN	Output	Watchdog output Equivalent to interrupt pending bit of last timer.	Low

* see GRLIB IP Library User’s Manual

35.7 Signal definitions and reset values

The signals and their reset values are described in table 372.

*Table 372.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
wdogn	Tri-state output	Watchdog output. Equivalent to interrupt pending bit of last timer.	Low	Tri-state
extclk	Input	External clock	-	-
tick[]	Output	Output tick	High	Logical 0

35.8 Timing

The timing waveforms and timing parameters are shown in figure 138 and are defined in table 373.

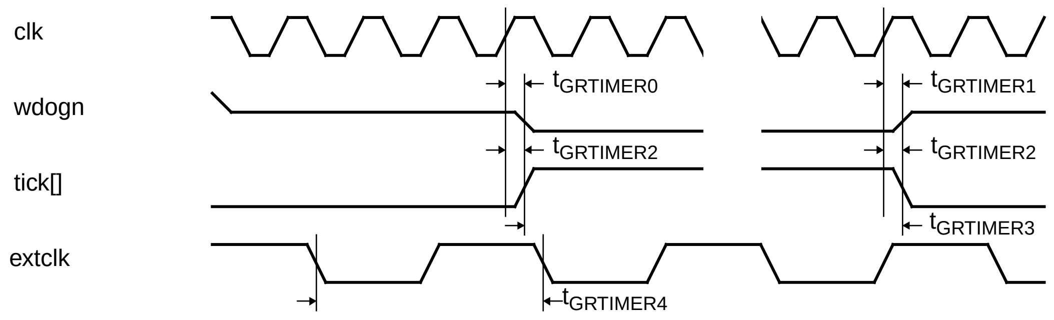


Figure 138. Timing waveforms

*Table 373.*Timing parameters

Name	**Parameter**	**Reference edge**	**Min**	**Max**	**Unit**
$t_{GRTIMER0}$	clock to output delay	rising *clk* edge	-	TBD	ns
$t_{GRTIMER1}$	clock to output tri-state	rising *clk* edge	-	TBD	ns
$t_{GRTIMER2}$	clock to tick output delay	rising *clk* edge	-	TBD	ns
$t_{GRTIMER3}$	tick output period	-	8	8	*clk* periods
$t_{GRTIMER4}$	external clock period	-	2		*clk* periods

35.9 Library dependencies

Table 374 shows the libraries used when instantiating the core (VHDL libraries)

*Table 374.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Signals, Component	Signal and component definitions

35.10 Instantiation

This example shows how the core can be instantiated.

```
TBD
```

36 GRACECTRL - AMBA System ACE Interface Controller

36.1 Overview

The core provides an AMBA AHB interface to the microprocessor interface of a Xilinx System ACE Compact Flash Solution. Accesses to the core’s memory space are directly translated to accesses on the System ACE microprocessor interface (MPU).

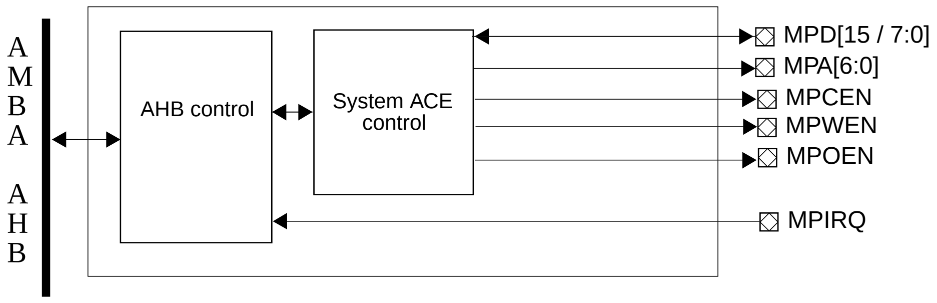


Figure 139. Block diagram

36.2 Operation

36.2.1 Operational model

The core has one AHB I/O area, accesses to this area are directly translated to accesses on the Xilinx System ACE’s Microprocessor Interface (MPU). When an access is made to the I/O area, the core first checks if there already is an ongoing access on the MPU. If an access is currently active, the core will respond with an AMBA SPLIT response. If the MPU bus is available, the core will start an access on the MPU bus and issue a SPLIT response to the AMBA master. If the core has been configured for a system that does not support SPLIT responses, it will insert wait states instead.

36.2.2 Bus widths

The AMBA access is directly translated to an MPU access where bits 6:0 of the AMBA address bus are connected to the MPU address bus. The core can be configured to connect to a 16-bit MPU interface or a 8-bit MPU interface. When the core is connected to a 8-bit MPU interface it can emulate 16-bit mode by translating 16-bit (half-word) AMBA accesses into two 8-bit MPU accesses. The mode to use is decided at implementation time via the VHDL generic *mode*.

The core does not perform any checks on the size of the AMBA access and software should only make half-word (16-bit), or byte (8-bit) depending on the setting of VHDL generic *mode*, accesses to the core’s memory area. Any other access size will be accepted by the core but the operation may not have the desired result. On AMBA writes the core uses address bit 1 (or address bits 1:0 for 8-bit mode) to select if it should propagate the high or the low part of the AMBA data bus to the MPU data bus. On read operations the core will propagate the read MPU data to all parts of the AMBA data bus.

It is recommended to set the *mode* VHDL generic to 2 for 8-bit MPU interfaces, and to 0 for 16-bit MPU interfaces. This way software can always assume that it communicates via a 16-bit MPU interface (accesses to the System ACE BUSMODEREG register are overriden by the core with suitable values when *mode* is set to 2).

36.2.3 Clocking and synchronization

The core has two clock inputs; the AMBA clock and the System ACE clock. The AMBA clock drives the AHB slave interface and the System ACE clock drives the System ACE interface state machine.

All signals crossing between the two clock domains are synchronized to prevent meta-stability. The system clock should have a higher frequency than the System ACE clock.

36.3 Registers

The core does implement any registers accessible via AMBA.

36.4 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x067. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

36.5 Implementation

36.5.1 Technology mapping

The core does not instantiate any technology specific primitives.

36.5.2 RAM usage

The core does not use any RAM components.

36.6 Configuration options

Table 375 shows the configuration options of the core (VHDL generics).

*Table 375.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB slave index	0 - (NAHBSLV-1)	0
hirq	Interrupt line	0 - (NAHBIRQ-1)	0
haddr	ADDR field of the AHB BAR0	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0	0 - 16#FFF#	16#FFF#
split	If this generic is set to 1 the core will issue AMBA SPLIT responses when it is busy performing an access to the System ACE. Otherwise the core will insert wait states until the operation completes. Note that SPLIT support on the AHBCTRL core MUST be enabled if this generic is set to 1.	0 - 1	0
swap	If this generic is set to 0 the core will connect the System ACE data(15:0) to AMBA data(15:0). If this generic is set to 1, the core will swap the System ACE data line and connect: System ACE data(15:8) <-> AMBA data(7:0) System ACE data(7 :0) <-> AMBA data(15:8). This generic only has effect for *mode* = 0.	0 - 1	0
oepol	Polarity of pad output enable signal	0 - 1	0
mode	Bus width mode 0: Core is connected to 16-bit MPU. Only half-word AMBA accesses should be made to the core. 1: Core is connected to 8-bit MPU. Only byte AMBA accesses should be made to the core. 2: Core is connected to 8-bit MPU but will emulate a 16-bit MPU interface. Only half-word AMBA accesses should be made to the core (recommended setting for 8-bit MPU interfaces).	0 - 2	0

36.7 Signal descriptions

Table 376 shows the interface signals of the core (VHDL ports).

*Table 376.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
CLKACE	N/A	Input	System ACE clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
ACEI	DI(15:0)	Input	Data line	-
	IRQ	Input	System ACE interrupt request	High
ACEO	ADDR(6:0)	Output	System ACE address	-
	DO(15:0)	Output	Data line	-
	CEN	Output	System ACE chip enable	Low
	WEN	Output	System ACE write enable	Low
	OEN	Output	System ACE output enable	Low
	DOEN	Output	Data line output enable	-

* see GRLIB IP Library User’s Manual

36.8 Library dependencies

Table 377 shows the libraries used when instantiating the core (VHDL libraries).

*Table 377.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GAISLER	MISC	Component, signals	Component and signal definitions
GRLIB	AMBA	Signals	AMBA signal definitions

36.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity gracectrl_ex is
  port (
    clk        : in  std_ulogic;
    clkace     : in  std_ulogic;
    rstn       : in  std_ulogic;
    sace_a     : out std_logic_vector(6 downto 0);
    sace_mpce  : out std_ulogic;
    sace_d     : inout std_logic_vector(15 downto 0);
    sace_oen   : out std_ulogic;
    sace_wen   : out std_ulogic;
    sace_mpirq : in  std_ulogic;
    );
end;
```

```
architecture rtl of gracectrl_ex is
  -- AMBA signals
  signal ahbsi  : ahb_slv_in_type;
  signal ahbso  : ahb_slv_out_vector := (others => ahbs_none);
  ...
  -- GRACECTRL signals
  signal acei : gracectrl_in_type;
  signal aceo : gracectrl_out_type;

begin

  -- AMBA Components are instantiated here
  ...

  -- GRACECTRL core is instantiated below
  grace0 : gracectrl generic map (hindex => 4, hirq => 4, haddr => 16#002#,
                                  hmask => 16#fff#, split => 1)
    port map (rstn, clk, ahbsi, ahbso(4), acei, acoo);
  sace_a_pads : outpadv generic map (width => 7, tech => padtech)
    port map (sace_a, aceo.addr);
  sace_mpce_pad : outpad generic map(tech => padtech)
    port map (sace_mpce, aceo.cen);
  sace_d_pads : iopadv generic map (tech => padtech, width => 16)
    port map (sace_d, aceo.do, aceo.doen, aceo.di);
  sace_oen_pad : outpad generic map (tech => padtech)
    port map (sace_oen, aceo.oen);
  sace_wen_pad : outpad generic map (tech => padtech)
    port map (sace_wen, aceo.wen);
  sace_mpirq_pad : inpad generic map (tech => padtech)
    port map (sace_mpirq, acei.irq);

end;
```

37 GRAES - Advanced Encryption Standard

37.1 Overview

The Advanced Encryption Standard (AES) is a symmetric encryption algorithm for high throughput application (like audio or video streams). The GRAES core implements the AES-128 algorithm, supporting the Electronic Codebook (ECB) method. The AES-128 algorithm is specified in the "Advanced Encryption Standard (AES)" document, Federal Information Processing Standards (FIPS) Publication 197. The document is established by the National Institute of Standards and Technology (NIST).

The core provides the following internal AMBA AHB slave interface, with sideband signals as per [GRLIB] including:

- interrupt bus
- configuration information
- diagnostic information

The core can be partition in the following hierarchical elements:

- Advanced Encryption Standard (AES) core
- AMBA AHB slave
- GRLIB plug&play wrapper

Note that the core can also be used without the GRLIB plug&play information.

37.2 Operation

The input and output for the AES algorithm each consist of sequences of 128 bits (digits with values of 0 or 1). These sequences will sometimes be referred to as blocks and the number of bits they contain will be referred to as their length. The cipher key for the AES-128 algorithm is a sequence of 128 bits (can also be 192 or 256 bits for other algorithms).

To transfer a 128 bit key or data block four write operations are necessary since the bus interface is 32 bit wide. After supplying a "key will be input" command to the control register, the key is input via four registers. After supplying a "data will be input" command to the control register, the input data is written via four registers. After the last input data register is written, the encryption or decryption is started. The progress can be observed via the debug register. When the operation is completed, an interrupt is generated. The output data is then read out via four registers. Note that the above sequence must be respected. It is not required to write a new key between each data input. There is no command needed for reading out the result.

The implementation requires around 89 clock cycles for a 128 bit data block in encryption direction and around 90 clock cycles for decryption direction. For decryption an initial key calculation is required. This takes around 10 additional clock cycles per every new key. Typically large amounts of data are decrypted (and also encrypted) with the same key. The key initialization for the decryption round does not influence the throughput.

37.3 Background

The Federal Information Processing Standards (FIPS) Publication Series of the National Institute of Standards and Technology (NIST) is the official series of publications relating to standards and guidelines adopted and promulgated under the provisions of the Information Technology Management Reform Act.

The Advanced Encryption Standard (AES) standard specifies the Rijndael algorithm, a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. Rijndael was designed to handle additional block sizes and key lengths, however they are not adopted in this standard.

37.4 AES-128 parameters

The GRAES core implements AES-128. An AES algorithm is defined by the following parameters according to FIPS-197:

- *Nk* number of 32-bit words comprising the cipher key
- *Nr* number of rounds

The AES-128 algorithm is specified as *Nk*=4 and *Nr*=10.

The GRAES core has been verified against the complete set of Known Answer Test vectors included in the AES Algorithm Validation Suite (AESAVS) from National Institute of Standards and Technology (NIST), Information Technology Laboratory, Computer Security Division.

37.5 Throughput

The data throughput for the GRAES core is around 128/90 bits per clock cycle, i.e. approximately 1.4 Mbits per MHz.

The underlaying AES core has been implemented in a dual crypto chip on 250 nm technology as depicted in the figure below. The throughput at 33 MHz operating frequency was 42 Mbit/s, the power consumption was 9,6 mW, and the size was 14,5 kgates.

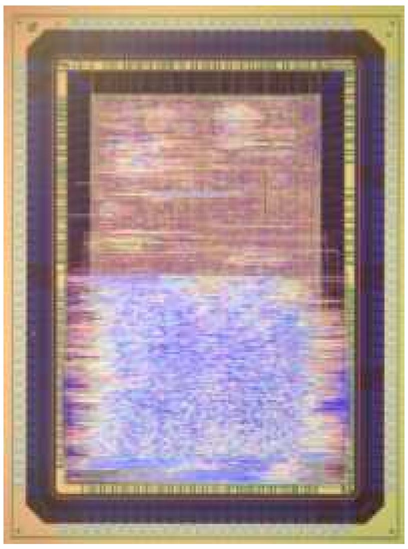


Figure 140. Dual Crypto Chip

37.6 Characteristics

The GRAES core has been synthesized for a Xilinx Virtex-2 XC2V6000-4 devices with the following results:

- LUTs: 5040 (7%)
- 256x1 ROMs (ROM256X1): 128

- Frequency:125 MHz

37.7 Registers

The core is programmed through registers mapped into AHB I/O address space.

*Table 378.*GRAES registers

AHB I/O address offset	**Register**
16#000#	Control Register
16#010#	Data Input 0 Register
16#014#	Data Input 1 Register
16#018#	Data Input 2 Register
16#01C#	Data Input 3 Register
16#020#	Data Output 0 Register
16#024#	Data Output 1 Register
16#028#	Data Output 2 Register
16#02C#	Data Output 3 Register
16#03C#	Debug Register

37.7.1 Control Register (W)

*Table 379.*Control Register

31	2	1	0
-		DE C	KE Y

31-2:	-	Unused
1:	DEC	0 = “encrypt”, 1 = “decrypt” (only relevant when KEY=1)
0:	KEY	0 = “data will be input”, 1 = “key will be input”

Note that the Data Input Registers cannot be written before a command is given to the Control Register. Note that the Data Input Registers must then be written in sequence, and all four registers must be written else the core ends up in an undefined state.

The KEY bit determines whether a key will be input (KEY=1), or data will be input (KEY=0). When a “key will be input” command is written, the DEC bit determines whether decryption (DEC=1) or encryption (DEC=0) should be applied to the subsequent data input.

Note that the register cannot be written after a command has been given, until the specific operation completes. A write access will be terminated with an AMBA AHB error response till the Data Input Register 3 has been written, and the with an AMBA AHB retry response till the operation completes. Any read access to this register results in an AMBA AHB error response.

37.7.2 Debug Register (R)

*Table 380.*Debug Register

31	0
FSM	

31-0:	FSM	Finite State Machine

Any write access to this register results in an AMBA AHB error response.

37.7.3 Data Input Registers (W)

*Table 381.*Data Input 0 Register

31 0
Data/Key(127 downto 96)

*Table 382.*Data Input 1 Register

31 0
Data/Key(95 downto 64)

*Table 383.*Data Input 2 Register

31 0
Data/Key(63 downto 32)

*Table 384.*Data Input 3 Register

31 0
Data/Key(31 downto 0)

Note that these registers can only be written with a key after a “key will be input” command has been written to the control register. Note that the registers must then be written in sequence, and all four registers must be written else the core ends up in an undefined state.

Note that these registers can only be written with data after a “data will be input” command has been written to the control register, else an AMBA AHB error response is given. Note that the registers must then be written in sequence and all four registers must be written else the core ends up in an undefined state. The encryption or decryption operation is started when the Data Input 3 Register is written to with data.

37.7.4 Data Output Registers (R)

*Table 385.*Data Output 0 Register

31 0
Data(127 downto 96)

*Table 386.*Data Output 1 Register

31 0
Data(95 downto 64)

*Table 387.*Data Output 2 Register

31 0
Data(63 downto 32)

*Table 388.*Data Output 3 Register

31 0
Data(31 downto 0)

Note that these registers can only be read after encryption or decryption has been completed. An AMBA AHB retry response is given to read accesses that occur while the encryption or decryption is in progress. If a read access is attempted before an encryption or decryption has even been initiated, then an AMBA AHB erro response is given. Write accesses to these registers result in an AMBA AHB error response.

37.8 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x073. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

37.9 Configuration options

Table 389 shows the configuration options of the core (VHDL generics).

*Table 389.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	0 - NAHBSLV-1	0
ioaddr	Addr field of the AHB I/O BAR	0 - 16#FFF#	0
iomask	Mask field of the AHB I/O BAR	0 - 16#FFF#	16#FFC#
hirq	Interrupt line used by the GRAES	0 - NAHBIRQ-1	0

37.10 Signal descriptions

Table 390 shows the interface signals of the core (VHDL ports).

*Table 390.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBI	*	Input	AHB slave input signals	-
AHBO	*	Output	AHB slave output signals	-
DEBUG[0:4]	N/A	Output	Debug information	-

* see GRLIB IP Library User's Manual

Note that the AES core can also be used without the GRLIB plug&play information. The AMBA AHB signals are then provided as IEEE Std_Logic_1164 compatible scalars and vectors.

37.11 Library dependencies

Table 391 shows libraries used when instantiating the core (VHDL libraries).

*Table 391.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	CRYPTO	Component	GRAES component declarations

37.12 Instantiation

This example shows how the core can be instantiated.

```
library  ieee;
use      ieee.std_logic_1164.all;
```

```
library  grlib;
use      grlib.amba.all;

library  gaisler;
use      gaisler.crypto.all;
...
...
   signal debug: std_logic_vector(0 to 4);
..
..
   GRAES0: graes
      generic map (
         hindex         => hindex,
         ioaddr         => ioaddr,
         iomask         => iomask,
         hirq           => hirq)
      port map (
         rstn           => rstn,
         clk            => clk,
         ahbi           => ahbsi,
         ahbo           => ahbso(hindex),
         debug          => debug);
```

38 GRAES_DMA - Advanced Encryption Standard with DMA

38.1 Overview

The Advanced Encryption Standard (AES) is a symmetric encryption algorithm for high throughput applications (like audio or video streams). The GRAES_DMA core implements the AES algorithm with 256-bit key length using CTR mode of operation. The AES algorithm is specified in the “Advanced Encryption Standard (AES)” document, Federal Information Processing Standards (FIPS) Publication 197. The document is established by the National Institute of Standards and Technology (NIST). DMA is used for efficiently transferring plaintext and ciphertext to the cryptographic core with minimum CPU involvement.

The core provides an AMBA AHB master interface, with sideband signals as per [GRLIB] including:

- interrupt bus
- configuration information
- diagnostic information

The core can be partition in the following hierarchical elements:

- Advanced Encryption Standard (AES) core
- AMBA AHB master

38.2 Operation

The input and output for the AES algorithm each consist of sequences of 128 bits (digits with values of 0 or 1). These sequences will sometimes be referred to as blocks and the number of bits they contain will be referred to as their length. The cipher key for the AES algorithm supported in this core is a sequence of 256 bits.

To encrypt a message a descriptor must be setup. It contains pointers to memory locations where the key, initialization vector and plaintext are located. The memory addresses for the key and initialization vector must be word aligned while the plaintext can start at any address. If the previous key and/or init vector are to be reused there are control bits in the descriptor which can be used to make the core skip the fetching of the respective pointers and also subsequently skip the fetching of the actual key and initvector. Currently the initvector and key always have to be loaded for the core to operate correctly.

When one or more descriptors have been enabled the core can be enabled and it will automatically start fetching the necessary values from memory, split the data into the required blocks, encrypt/decrypt and finally write back the result to memory. When each descriptor is finished the core will set the enable bit to 0. An interrupt can also optionally be generated. The result of the encryption or decryption can be either written back to the same memory address from where the plain or ciphertext was read or to a different location specified in an additional pointer. The layout of the descriptor is shown in the tables below.

Table 392. GRAES_DMA descriptor word 0 (address offset 0x0)

31 21	20 9	8	7	6	5	4	3	2	1	0
LEN	RESERVED	SP	KE	IV	DO	ED	MD		IE	EN

31: 21	Length (LEN) - Length in bytes of message to process
20: 9	RESERVED
8	Stop (SP) - When asserted descriptor processing is stopped when the current descriptor is finished i.e. the descriptor processing is stopped even if the next descriptor is enabled.

Table 392. GRAES_DMA descriptor word 0 (address offset 0x0)

7	Key (KE) - When set a new key will be fetched and used from the memory address set in the key address descriptor word. If not set the currently stored key is used and the key adddress word should not be included in the descriptor.
6	Initialization vector (IV) - When set a new initialization vectir will be fetched and used from the memory address set in the initialization vector address descriptor word. If not set the currently stored initialization vector is used and the initialization vector adddress word should not be included in the descriptor.
5	Dataout (DO) - When set the encrypted/decrypted output will be written to the memory address specified in the dataout descriptor word. Otherwise data is written to the same memory address from where the original plaintext/ciphertext was fetched and the dataout address word should not be included in the descriptor.
4	Encrypt-decrypt (ED) - If set to one encryption will be performed otherwise decryption
3	Mode (MD) - If set to 1 CTR mode is selected otherwise the core will use CBC. Currently this bit is unused and the core always uses CTR mode.
2	RESERVED
1	Interrupt enable (IE) - When set an interrupt will be generated when the orocessing of the current descriptor is finished and the interrupt enable bit in the control register is set.
0	Enable (EN) -

Table 393. GRAES_DMA descriptor word 1 (address offset 0x4)

31 0
Data input address

31: 0	Data input address - Memory address pointer where plaintext/ciphertext for encryption/descryption is located.

Table 394. GRAES_DMA descriptor word 2 (address offset 0x10)

31 2	1	0
Dataout address		

31: 2	Dataout address - Memory address where encrypted/decrypted data shall be stored. If the data should be stored at the same location as the input data (DO bit in word 0 is 0) then this word shall not be included in the descriptor.
1: 0	Reserved

Table 395. GRAES_DMA descriptor word 3(address offset 0xC)

31 2	1	0
IV address		

31: 2	Initialization vector address - Memory address where initialization vector is located. If a new initvector is not ueeded (IV bit in word 0 is 0) then this word shall not be included in the descriptor.
1: 0	Reserved

Table 396. GRAES_DMA descriptor word 4 (address offset 0x8)

31 2	1	0
Key address		

Table 396. GRAES_DMA descriptor word 4 (address offset 0x8)

31: 2	Key address - Memory address where key is located. If a new key is not ueeded (KE bit in word 0 is 0) then this word shall not be included in the descriptor.
1: 0	Reserved

Table 397. GRAES_DMA descriptor word 5 (address offset 0x14)

31 … 2	1	0
Next descriptor		

31: 2	Next descriptor address - Memory address to the next descriptor.
1: 0	Reserved

The words in the descriptor should always be written in the order listed above. If one or more words are not included the offsets of the following words should be adjusted accordingly.

38.3 Background

The Federal Information Processing Standards (FIPS) Publication Series of the National Institute of Standards and Technology (NIST) is the official series of publications relating to standards and guidelines adopted and promulgated under the provisions of the Information Technology Management Reform Act.

The Advanced Encryption Standard (AES) standard specifies the Rijndael algorithm, a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. Rijndael was designed to handle additional block sizes and key lengths, however they are not adopted in this standard.

38.4 Characteristics

The GRAES_DMA core has been synthesized for a Actel AX2000-std device with the following results:

- Combinational Cells: 9364 of 21504 (44%)
- Sequential Cells: 2374 of 10752 (22%)
- Total Cells: 11738 of 32256 (37%)
- Block Rams : 0 of 64 (0%)
- Frequency:60 MHz

38.5 Registers

The core is programmed through registers mapped into APB address space.

*Table 398.*GRSPW registers

APB address offset	Register
0x0	Control
0x4	Status
0x8	Descriptor address

Table 399. GRAES_DMA control register

31 ... 2	1	0
RESERVED	IE	EN

31: 2	RESERVED
1	Interrupt Enable (IE) - If set, an interrupt is generated each time a message has been decrypted . Reset value: ‘0’.
0	Enable (EN) - Write a one to this bit each time new descriptors are activated in the list. Writing a one will cause the core to read a new descriptor and perform the requested operation. This bit is automatically cleared when the core encounters a descriptor which is disabled. Reset value: ‘0’

Table 400. GRSPW status register

31 ... 0
RESERVED

31: 0	RESERVED

Table 401. GRSPW Descriptor address

31 ... 2	1	0
Descriptor address		

31: 2	Current descriptor address - Points to current descriptor. Can be initialized with a new pointer when the core is disabled. Is updated by the core while it is progressing through the list of descriptors.
1: 0	RESERVED

38.6 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x07B. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

38.7 Configuration options

Table 402 shows the configuration options of the core (VHDL generics).

*Table 402.*Configuration options

Generic	**Function**	**Allowed range**	**Default**
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB BAR	0 - 16#FFF#	0
pmask	Mask field of the APB BAR	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRAES	0 - NAHBIRQ-1	0

38.8 Signal descriptions

Table 403 shows the interface signals of the core (VHDL ports).

*Table 403.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-

* see GRLIB IP Library User's Manual

38.9 Library dependencies

Table 404 shows libraries used when instantiating the core (VHDL libraries).

*Table 404.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	CRYPTO	Component	GRAES component declarations

38.10 Instantiation

This example shows how the core can be instantiated.

```
entity graes_dma_tb is
   generic(
      hindex:        in    Integer := 0;
      pindex:        in    Integer := 0;
      paddr:         in    Integer := 0;
      pmask:         in    Integer := 16#fff#;
      pirq:          in    Integer := 1);

end entity graes_dma_tb;


signal   rstn:         std_ulogic := ’0’;
signal   clk:          std_ulogic := ’0’;
signal   apbi:         apb_slv_in_type;
signal   apbo:         apb_slv_out_vector := (others => apb_none);
signal   ahbmi:        ahb_mst_in_type;
signal   ahbmo:        ahb_mst_out_vector := (others => ahbm_none);

graes0: graes_dma
     generic map(
       hindex         => hindex,
       pindex         => pindex,
       paddr          => paddr,
       pmask          => pmask,
       pirq           => pirq)
     port map(
       rstn           => rstn,
       clk            => clk,
       ahbi           => ahbmi,
       ahbo           => ahbmo(hindex),
       apbi           => apbi,
       apbo           => apbo(pindex));
```

39 GRCAN - CAN 2.0 Controller with DMA

39.1 Overview

The CAN controller is assumed to operate in an AMBA bus system where both the AMBA AHB bus and the APB bus are present. The AMBA APB bus is used for configuration, control and status handling. The AMBA AHB bus is used for retrieving and storing CAN messages in memory external to the CAN controller. This memory can be located on-chip, as shown in the block diagram, or external to the chip.

The CAN controller supports transmission and reception of sets of messages by use of circular buffers located in memory external to the core. Separate transmit and receive buffers are assumed. Reception and transmission of sets of messages can be ongoing simultaneously.

After a set of message transfers has been set up via the AMBA APB interface the DMA controller initiates a burst of read accesses on the AMBA AHB bus to fetch messages from memory, which are performed by the AHB master. The messages are then transmitted by the CAN core. When a programmable number of messages have been transmitted, the DMA controller issues an interrupt.

After the reception has been set up via the AMBA APB interface, messages are received by the CAN core. To store messages to memory, the DMA controller initiates a burst of write accesses on the AMBA AHB bus, which are performed by the AHB master. When a programmable number of messages have been received, the DMA controller issues an interrupt.

The CAN controller can detect a SYNC message and generate an interrupt, which is also available as an output signal from the core. The SYNC message identifier is programmable via the AMBA APB interface. Separate synchronisation message interrupts are provided.

The CAN controller can transmit and receive messages on either of two CAN busses, but only on one at a time. The selection is programmable via the AMBA APB interface.

Note that it is not possible to receive a CAN message while transmitting one.

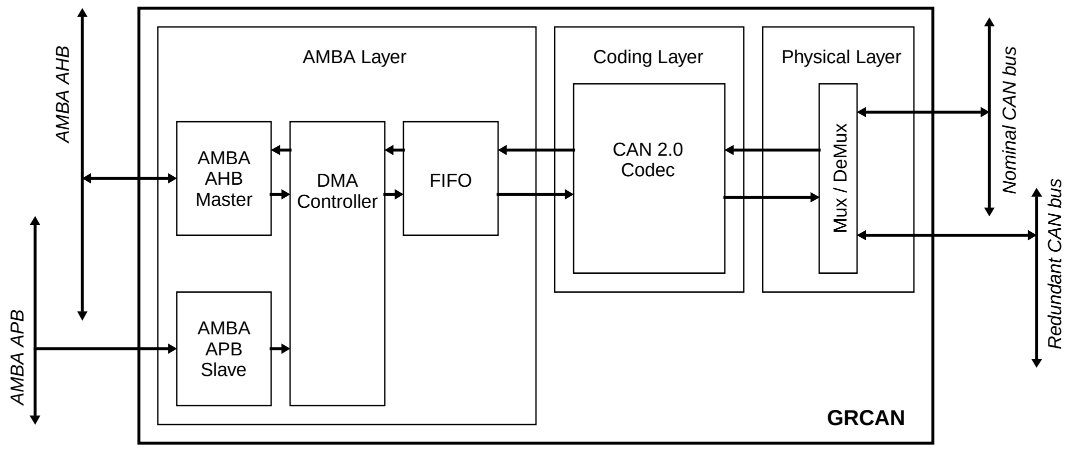


Figure 141. Block diagram

39.1.1 Function

The core implements the following functions:

- CAN protocol
- Message transmission
- Message filtering and reception

- SYNC message reception
- Status and monitoring
- Interrupt generation
- Redundancy selection

39.1.2 Interfaces

The core provides the following external and internal interfaces:

- CAN interface
- AMBA AHB master interface, with sideband signals as per [GRLIB] including:
- cacheability information
- interrupt bus
- configuration information
- diagnostic information
- AMBA APB slave interface, with sideband signals as per [GRLIB] including:
- interrupt bus
- configuration information
- diagnostic information

39.1.3 Hierarchy

The CAN controller core can be partitioned in the following hierarchical elements:

- CAN 2.0 Core
- Redundancy Multiplexer / De-multiplexer
- Direct Memory Access controller
- AMBA APB slave
- AMBA AHB master

39.2 Interface

The external interface towards the CAN bus features two redundant pairs of transmit output and receive input (i.e. 0 and 1).

The active pair (i.e. 0 or 1) is selectable by means of a configuration register bit. Note that all reception and transmission is made over the active pair.

For each pair, there is one enable output (i.e. 0 and 1), each being individually programmable. Note that the enable outputs can be used for enabling an external physical driver. Note that both pairs can be enabled simultaneously. Note that the polarity for the enable/inhibit inputs on physical interface drivers differs, thus the meaning of the enable output is undefined.

Redundancy is implemented by means of Selective Bus Access. Note that the active pair selection above provides means to meet this requirement.

39.3 Protocol

The CAN protocol is based on a CAN 2.0 controller VHDL core. The CAN controller complies with CAN Specification Version 2.0 Part B, except for the overload frame generation.

Note that there are three different CAN types generally defined:

- 2.0A, which considers 29 bit ID messages as an error

- 2.0B Passive, which ignores 29 bit ID messages
- 2.0B Active, which handles 11 and 29 bit ID messages

Only 2.0B Active is implemented.

39.4 Status and monitoring

The CAN interface incorporates status and monitoring functionalities. This includes:

- Transmitter active indicator
- Bus-Off condition indicator
- Error-Passive condition indicator
- Over-run indicator
- 8-bit Transmission error counter
- 8-bit Reception error counter

The status is available via a register and is also stored in a circular buffer for each received message.

39.5 Transmission

The transmit channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The transmit channel can be enabled or disabled.

39.5.1 Circular buffer

The transmit channel operates on a circular buffer located in memory external to the CAN controller. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the AMBA AHB master interface.

Each CAN message occupies 4 consecutive 32-bit words in memory. Each CAN message is aligned to 4 words address boundaries (i.e. the 4 least significant byte address bits are zero for the first word in a CAN message).

The size of the buffer is defined by the CanTxSIZE.SIZE field, specifying the number of CAN messages * 4 that fit in the buffer.

E.g. CanTxSIZE.SIZE =2 means 8 CAN messages fit in the buffer.

Note however that it is not possible to fill the buffer completely, leaving at least one message position in the buffer empty. This is to simplify wrap-around condition checking.

E.g. CanTxSIZE.SIZE =2 means that 7 CAN messages fit in the buffer at any given time.

39.5.2 Write and read pointers

The write pointer (CanTxWR.WRITE) indicates the position+1 of the last CAN message written to the buffer. The write pointer operates on number of CAN messages, not on absolute or relative addresses.

The read pointer (CanTxRD.READ) indicates the position+1 of the last CAN message read from the buffer. The read pointer operates on number of CAN messages, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of CAN messages available in the buffer for transmission. The difference is calculated using the buffer size, specified by the CanTxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 CAN messages available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE=2 and CanTxRD.READ=0.
- There are 2 CAN messages available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE =0 and CanTxRD.READ =6.
- There are 2 CAN messages available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE =1 and CanTxRD.READ =7.
- There are 2 CAN messages available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE =5 and CanTxRD.READ =3.

When a CAN message has been successfully transmitted, the read pointer (CanTxRD.READ) is automatically incremented, taking wrap around effects of the circular buffer into account. Whenever the write pointer CanTxWR.WRITE and read pointer CanTxRD.READ are equal, there are no CAN messages available for transmission.

39.5.3 Location

The location of the circular buffer is defined by a base address (CanTxADDR.ADDR), which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

39.5.4 Transmission procedure

When the channel is enabled (CanTxCTRL.ENABLE=1), as soon as there is a difference between the write and read pointer, a message transmission will be started. Note that the channel should not be enabled if a potential difference between the write and read pointers could be created, to avoid the message transmission to start prematurely.

A message transmission will begin with a fetch of the complete CAN message from the circular buffer to a local fetch-buffer in the CAN controller. After a successful data fetch, a transmission request will be forwarded to the CAN core. If there is at least an additional CAN message available in the circular buffer, a prefetch of this CAN message from the circular buffer to a local prefetch-buffer in the CAN controller will be performed. The CAN controller can thus hold two CAN messages for transmission: one in the fetch buffer, which is fed to the CAN core, and one in the prefetch buffer.

After a message has been successfully transmitted, the prefetch-buffer contents are moved to the fetch buffer (provided that there is message ready). The read pointer (CanTxRD.READ) is automatically incremented after a successful transmission, i.e. after the fetch-buffer contents have been transmitted, taking wrap around effects of the circular buffer into account. If there is at least an additional CAN message available in the circular buffer, a new prefetch will be performed.

If the write and read pointers are equal, no more prefetches and fetches will be performed, and transmission will stop.

If the single shot mode is enabled for the transmit channel (CanTxCTRL.SINGLE=1), any message for which the arbitration is lost, or failed for some other reason, will lead to the disabling of the channel (CanTxCTRL.ENABLE=0), and the message will not be put up for re-arbitration.

Interrupts are provided to aid the user during transmission, as described in detail later in this section. The main interrupts are the Tx, TxEmpty and TxIrq which are issued on the successful transmission of a message, when all messages have been transmitted successfully and when a predefined number of messages have been transmitted successfully. The TxLoss interrupt is issued whenever transmission arbitration has been lost, could also be caused by a communications error. The TxSync interrupt is issued when a message matching the SYNC Code Filter Register.SYNC and SYNC Mask Filter Register.MASK registers is successfully transmitted. Additional interrupts are provided to signal error conditions on the CAN bus and AMBA bus.

39.5.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (CanTxADDR.ADDR) field.

While the channel is disabled, the read pointer (CanTxRD.READ) can be changed to an arbitrary value pointing to the first message to be transmitted, and the write pointer (CanTxWR.WRITE) can be changed to an arbitrary value.

When the channel is enabled, the transmission will start from the read pointer and continue to the write pointer.

39.5.6 AMBA AHB error

Definition:

- a message fetch occurs when no other messages is being transmitted
- a message prefetch occurs when a previously fetched message is being transmitted
- the local fetch buffer holds the message being fetched
- the local prefetch buffer holds the message being prefetched
- the local fetch buffer holds the message being transmitted by the CAN core
- a successfully prefetched message is copied from the local prefetch buffer to the local fetch buffer when that buffer is freed after a successful transmission.

An AHB error response occurring on the AMBA AHB bus while a CAN message is being fetched will result in a TxAHBErr interrupt.

If the CanCONF.ABORT bit is set to 0b, the channel causing the AHB error will skip the message being fetched from memory and will increment the read pointer. No message will be transmitted.

If the CanCONF.ABORT bit is set to 1b, the channel causing the AHB error will be disabled (CanTx-CTRL.ENABLE is cleared automatically to 0 b). The read pointer can be used to determine which message caused the AHB error. Note that it could be any of the four word accesses required to read a message that caused the AHB error.

If the CanCONF.ABORT bit is set to 1b, all accesses to the AMBA AHB bus will be disabled after an AMBA AHB error occurs, as indicated by the CanSTAT.AHBErr bit being 1b. The accesses will be disabled until the CanSTAT register is read, and automatically clearing bit CanSTAT.AHBErr.

An AHB error response occurring on the AMBA AHB bus while a CAN message is being prefetched will not cause an interrupt, but will stop the ongoing prefetch and further prefetch will be prevented temporarily. The ongoing transmission of a CAN message from the fetch buffer will not be affected. When the fetch buffer is freed after a successful transmission, a new fetch will be initiated, and if this fetch results in an AHB error response occurring on the AMBA AHB bus, this will be handled as for the case above. If no AHB error occurs, prefetch will be allowed again.

39.5.7 Enable and disable

When an enabled transmit channel is disabled (CanTxCTRL.ENABLE=0b), any ongoing CAN message transfer request will not be aborted until a CAN bus arbitration is lost or the message has been sent successfully. If the message is sent successfully, the read pointer (CanTxRD.READ) is automatically incremented. Any associated interrupts will be generated.

The progress of the any ongoing access can be observed via the CanTxCTRL.ONGOING bit. The CanTxCTRL.ONGOING must be 0b before the channel can be re-configured safely (i.e. changing address, size or read pointer). It is also possible to wait for the Tx and TxLoss interrupts described hereafter.

The channel can be re-enabled again without the need to re-configure the address, size and pointers.

Priority inversion is handled by disabling the transmitting channel, i.e. setting CanTxCTRL.ENABLE=0b as described above, and observing the progress, i.e. reading via the CanTxCTRL.ONGOING bit as described above. When the transmit channel is disabled, it can be re-configured and a higher priority message can be transmitted. Note that the single shot mode does not require the channel to be disabled, but the progress should still be observed as above.

No message transmission is started while the channel is not enabled.

39.5.8 Interrupts

During transmission several interrupts can be generated:

• TxLoss:	Message arbitration lost for transmit (could be caused by communications error, as indicated by other interrupts as well)
• TxErrCntr:	Error counter incremented for transmit
• TxSync:	Synchronization message transmitted
• Tx:	Successful transmission of one message
• TxEmpty:	Successful transmission of all messages in buffer
• TxIrq:	Successful transmission of a predefined number of messages
• TxAHBErr:	AHB access error during transmission
• Off:	Bus-off condition
• Pass:	Error-passive condition

The Tx, TxEmpty and TxIrq interrupts are only generated as the result of a successful message transmission, after the CanTxRD.READ pointer has been incremented.

39.6 Reception

The receive channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The receive channel can be enabled or disabled.

39.6.1 Circular buffer

The receive channel operates on a circular buffer located in memory external to the CAN controller. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the AMBA AHB master interface.

Each CAN message occupies 4 consecutive 32-bit words in memory. Each CAN message is aligned to 4 words address boundaries (i.e. the 4 least significant byte address bits are zero for the first word in a CAN message).

The size of the buffer is defined by the CanRxSIZE.SIZE field, specifying the number of CAN messages * 4 that fit in the buffer.

E.g. CanRxSIZE.SIZE=2 means 8 CAN messages fit in the buffer.

Note however that it is not possible to fill the buffer completely, leaving at least one message position in the buffer empty. This is to simplify wrap-around condition checking.

E.g. CanRxSIZE.SIZE=2 means that 7 CAN messages fit in the buffer at any given time.

39.6.2 Write and read pointers

The write pointer (CanRxWR.WRITE) indicates the position+1 of the last CAN message written to the buffer. The write pointer operates on number of CAN messages, not on absolute or relative addresses.

The read pointer (CanRxRD.READ) indicates the position+1 of the last CAN message read from the buffer. The read pointer operates on number of CAN messages, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of CAN message positions available in the buffer for reception. The difference is calculated using the buffer size, specified by the CanRxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 CAN messages available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE=2 and CanRxRD.READ=0.
- There are 2 CAN messages available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE =0 and CanRxRD.READ=6.
- There are 2 CAN messages available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE =1 and CanRxRD.READ=7.
- There are 2 CAN messages available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE =5 and CanRxRD.READ=3.

When a CAN message has been successfully received and stored, the write pointer (CanRxWR.WRITE) is automatically incremented, taking wrap around effects of the circular buffer into account. Whenever the read pointer CanRxRD.READ equals (CanRxWR.WRITE+1) modulo (CanRxSIZE.SIZE*4), there is no space available for receiving another CAN message.

The error behavior of the CAN core is according to the CAN standard, which applies to the error counter, buss-off condition and error-passive condition.

39.6.3 Location

The location of the circular buffer is defined by a base address (CanRxADDR.ADDR), which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

39.6.4 Reception procedure

When the channel is enabled (CanRxCTRL.ENABLE=1), and there is space available for a message in the circular buffer (as defined by the write and read pointer), as soon as a message is received by the CAN core, an AMBA AHB store access will be started. The received message will be temporarily stored in a local store-buffer in the CAN controller. Note that the channel should not be enabled until the write and read pointers are configured, to avoid the message reception to start prematurely

After a message has been successfully stored the CAN controller is ready to receive a new message. The write pointer (CanRxWR.WRITE) is automatically incremented, taking wrap around effects of the circular buffer into account.

Interrupts are provided to aid the user during reception, as described in detail later in this section. The main interrupts are the Rx, RxFull and RxIrq which are issued on the successful reception of a message, when the message buffer has been successfully filled and when a predefined number of messages have been received successfully. The RxMiss interrupt is issued whenever a message has been received but does not match a message filtering setting, i.e. neither for the receive channel nor for the SYNC message described hereafter.

The RxSync interrupt is issued when a message matching the SYNC Code Filter Register.SYNC and SYNC Mask Filter Register.MASK registers has been successfully received. Additional interrupts are provided to signal error conditions on the CAN bus and AMBA bus.

39.6.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (CanRxADDR.ADDR) field.

While the channel is disabled, the write pointer (CanRxWR.WRITE) can be changed to an arbitrary value pointing to the first message to be received, and the read pointer (CanRxRD.READ) can be changed to an arbitrary value.

When the channel is enabled, the reception will start from the write pointer and continue to the read pointer.

39.6.6 AMBA AHB error

An AHB error response occurring on the AMBA AHB bus while a CAN message is being stored will result in an RxAHBErr interrupt.

If the CanCONF.ABORT bit is set to 0b, the channel causing the AHB error will skip the received message, not storing it to memory. The write pointer will be incremented.

If the CanCONF.ABORT bit is set to 1b, the channel causing the AHB error will be disabled (CanRxCTRL.ENABLE is cleared automatically to 0b). The write pointer can be used to determine which message caused the AHB error. Note that it could be any of the four word accesses required to writ a message that caused the AHB error.

If the CanCONF.ABORT bit is set to 1b, all accesses to the AMBA AHB bus will be disabled after an AMBA AHB error occurs, as indicated by the CanSTAT.AHBErr bit being 1b. The accesses will be disabled until the CanSTAT register is read, and automatically clearing bit CanSTAT.AHBErr.

39.6.7 Enable and disable

When an enabled receive channel is disabled (CanRxCTRL.ENABLE=0b), any ongoing CAN message storage on the AHB bus will not be aborted, and no new message storage will be started. Note that only complete messages can be received from the CAN core. If the message is stored successfully, the write pointer (CanRxWR.WRITE) is automatically incremented. Any associated interrupts will be generated.

The progress of the any ongoing access can be observed via the CanRxCTRL.ONGOING bit. The CanRxCTRL.ONGOING must be 0b before the channel can be re-configured safely (i.e. changing address, size or write pointer). It is also possible to wait for the Rx and RxMiss interrupts described hereafter.

The channel can be re-enabled again without the need to re-configure the address, size and pointers.

No message reception is performed while the channel is not enabled

39.6.8 Interrupts

During reception several interrupts can be generated:

- RxMiss: Message filtered away for receive
- RxErrCntr: Error counter incremented for receive
- RxSync: Synchronization message received
- Rx: Successful reception of one message
- RxFull: Successful reception of all messages possible to store in buffer

- RxIrq: Successful reception of a predefined number of messages
- RxAHBErr: AHB access error during reception
- OR: Over-run during reception
- OFF: Bus-off condition
- PASS: Error-passive condition

The Rx, RxFull and RxIrq interrupts are only generated as the result of a successful message reception, after the CanRxWR.WRITE pointer has been incremented.

The OR interrupt is generated when a message is received while a previously received message is still being stored. A full circular buffer will lead to OR interrupts for any subsequently received messages. Note that the last message stored which fills the circular buffer will not generate an OR interrupt. The overrun is also reported with the CanSTAT.OR bit, which is cleared when reading the register.

The error behavior of the CAN core is according to the CAN standard, which applies to the error counter, buss-off condition and error-passive condition.

39.7 Global reset and enable

When the CanCTRL.RESET bit is set to 1b, a reset of the core is performed. The reset clears all the register fields to their default values. Any ongoing CAN message transfer request will be aborted, potentially violating the CAN protocol.

When the CanCTRL.ENABLE bit is cleared to 0b, the CAN core is reset and the configuration bits CanCONF.SCALER, CanCONF.PS1, CanCONF.PS2, CanCONF.RSJ and CanCONF.BPR may be modified. When disabled, the CAN controller will be in sleep mode not affecting the CAN bus by only sending recessive bits. Note that the CAN core requires that 10 recessive bits are received before any reception or transmission can be initiated. This can be caused either by no unit sending on the CAN bus, or by random bits in message transfers.

39.8 Interrupt

Three interrupts are implemented by the CAN interface:

Index:	Name:	Description:
0	IRQ	Common output from interrupt handler
1	TxSYNC	Synchronization message transmitted (optional)
2	RxSYNC	Synchronization message received (optional)

The interrupts are configured by means of the *pirq* VHDL generic and the *singleirq* VHDL generic.

39.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 405.*GRCAN registers

APB address offset	Register
16#000#	Configuration Register
16#004#	Status Register
16#008#	Control Register
16#018#	SYNC Mask Filter Register
16#01C#	SYNC Code Filter Register
16#100#	Pending Interrupt Masked Status Register
16#104#	Pending Interrupt Masked Register
16#108#	Pending Interrupt Status Register
16#10C#	Pending Interrupt Register
16#110#	Interrupt Mask Register
16#114#	Pending Interrupt Clear Register
16#200#	Transmit Channel Control Register
16#204#	Transmit Channel Address Register
16#208#	Transmit Channel Size Register
16#20C#	Transmit Channel Write Register
16#210#	Transmit Channel Read Register
16#214#	Transmit Channel Interrupt Register
16#300#	Receive Channel Control Register
16#304#	Receive Channel Address Register
16#308#	Receive Channel Size Register
16#30C#	Receive Channel Write Register
16#310#	Receive Channel Read Register
16#314#	Receive Channel Interrupt Register
16#318#	Receive Channel Mask Register
16#31C#	Receive Channel Code Register

39.9.1 Configuration Register [CanCONF] R/W

*Table 406.*Configuration Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SCALER								PS1				PS2			
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	RSJ					BPR				SAM	Silent	Select	Enable1	Enable0	Abort

31-24:	SCALER	Prescaler setting, 8-bit: system clock / (SCALER +1)
23-20:	PS1	Phase Segment 1, 4-bit: (valid range 1 to 15)
19-16:	PS2	Phase Segment 2, 4-bit: (valid range 2 to 8)
14-12:	RSJ	ReSynchronization Jumps, 3-bit: (valid range 1 to 4)
9:8:	BPR	Baud rate, 2-bit:
		00b = system clock / (SCALER +1) / 1
		01b = system clock / (SCALER +1) / 2
		10b = system clock / (SCALER +1) / 4
		11b = system clock / (SCALER +1) / 8

5:	SAM	Single sample when 0b. Triple sample when 1b.
4:	SILENT	Listen only to the CAN bus, send recessive bits.
3:	SELECT	Selection receiver input and transmitter output: Select receive input 0 as active when 0b, Select receive input 1 as active when 1b Select transmit output 0 as active when 0b, Select transmit output 1 as active when 1b
2:	ENABLE1	Set value of output 1 enable
1:	ENABLE0	Set value of output 0 enable
0:	ABORT	Abort transfer on AHB ERROR

All bits are cleared to 0 at reset.

Note that constraints on PS1, PS2 and RSJ are defined as:

- PS1 +1 >= PS2
- PS1 > PS2
- PS2 >= RSJ

Note that CAN standard TSEG1 is defined by PS1+1.

Note that CAN standard TSEG2 is defined by PS2.

Note that the SCALER setting defines the CAN time quantum, together with the BPR setting:

system clock / ((SCALER+1) * BPR)

where SCALER is in range 0 to 255, and the resulting division factor due to BPR is 1, 2, 4 or 8.

For a quantum equal to one system clock period, an additional quantum is added to the node delay. Note that for minimizing the node delay, then set either SCALER > 0 or BRP > 0.

Note that the resulting bit rate is:

system clock / ((SCALER+1) * BPR * (1+ PS1+1 + PS2))

where PS1 is in the range 1 to 15, and PS2 is in the range 2 to 8.

Note that RSJ defines the number of allowed re-synchronization jumps according to the CAN standard, being in the range 1 to 4.

For SAM = 0b (single), the bus is sampled once; recommended for high speed buses (SAE class C).

For SAM = 1b (triple), the bus is sampled three times; recommended for low/medium speed buses (SAE class A and B) where filtering spikes on the bus line is beneficial.

Note that the transmit or receive channel active during the AMBA AHB error is disabled if the ABORT bit is set to 1b. Note that all accesses to the AMBA AHB bus will be disabled after an AMBA AHB error occurs while the ABORT bit is set to 1b. The accesses will be disabled until the CanSTAT register is read.

39.9.2 Status Register [CanSTAT] R

*Table 407.*Status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TxChannels				RxChannels				TxErrCntr							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RxErrCntr											Active	AHB Err	OR	Off	Pass

31-28:	TxChannels	Number of TxChannels -1, 4-bit
27-24:	RxChannels	Number of RxChannels -1, 4-bit

23-16:	TxErrCntr	Transmission error counter, 8-bit
15-8:	RxErrCntr	Reception error counter, 8-bit
4:	ACTIVE	Transmission ongoing
3:	AHBErr	AMBA AHB master interface blocked due to previous AHB error
2:	OR	Overrun during reception
1:	OFF	Bus-off condition
0:	PASS	Error-passive condition

All bits are cleared to 0 at reset.

The OR bit is set if a message with a matching ID is received and cannot be stored via the AMBA AHB bus, this can be caused by bandwidth limitations or when the circular buffer for reception is already full.

The OR and AHBErr status bits are cleared when the register has been read.

Note that TxErrCntr and RxErrCntr are defined according to CAN protocol.

Note that the AHBErr bit is only set to 1b if an AMBA AHB error occurs while the CanCONF.ABORT bit is set to 1b.

39.9.3 Control Register [CanCTRL] R/W

*Table 408.*Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
														Reset	Enable

1:	RESET	Reset complete core when 1
0:	ENABLE	Enable CAN controller, when 1. Reset CAN controller, when 0

All bits are cleared to 0 at reset.

Note that RESET is read back as 0b.

Note that ENABLE should be cleared to 0b to while other settings are modified, ensuring that the CAN core is properly synchronized.

Note that when ENABLE is cleared to 0b, the CAN interface is in sleep mode, only outputting recessive bits.

Note that the CAN core requires that 10 recessive bits be received before receive and transmit operations can begin.

39.9.4 SYNC Code Filter Register [CanCODE] R/W

*Table 409.*SYNC Code Filter Register

31	30	29	28 — 0
			SYNC

28-0:	SYNC	Message Identifier

All bits are cleared to 0 at reset.

Note that Base ID is bits 28 to 18 and Extended ID is bits 17 to 0.

39.9.5 SYNC Mask Filter Register [CanMASK] R/W

*Table 410.*SYNC Mask Filter Register

31	30	29	28-0
			MASK

28-0:	MASK	Message Identifier

All bits are set to 1 at reset.

Note that Base ID is bits 28 to 18 and Extended ID is bits 17 to 0.

A RxSYNC message ID is matched when:

((Received-ID XOR CanCODE.SYNC) AND CanMASK.MASK) = 0

A TxSYNC message ID is matched when:

((Transmitted-ID XOR CanCODE.SYNC) AND CanMASK.MASK) = 0

39.9.6 Transmit Channel Control Register [CanTxCTRL] R/W

*Table 411.*Transmit Channel Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

15	14	13	12	11	10	9-8	7	6	5	4	3	2	1	0
												Sin-gle	Ong oing	Ena ble

2:	SINGLE	Single shot mode
1:	ONGOING	Transmission ongoing
0:	ENABLE	Enable channel

All bits are cleared to 0 at reset.

Note that if the SINGLE bit is 1b, the channel is disabled (i.e. the ENABLE bit is cleared to 0b) if the arbitration on the CAN bus is lost.

Note that in the case an AHB bus error occurs during an access while fetching transmit data, and the CanCONF.ABORT bit is 1b, then the ENABLE bit will be reset automatically.

At the time the ENABLE is cleared to 0b, any ongoing message transmission is not aborted, unless the CAN arbitration is lost or communication has failed.

Note that the ONGOING bit being 1b indicates that message transmission is ongoing and that configuration of the channel is not safe.

39.9.7 Transmit Channel Address Register [CanTxADDR] R/W

*Table 412.*Transmit Channel Address Register

31-10	9-0
ADDR	

31-10:	ADDR	Base address for circular buffer

All bits are cleared to 0 at reset.

39.9.8 Transmit Channel Size Register [CanTxSIZE] R/W

*Table 413.*Transmit Channel Size Register

31 21	20 6	5 0
	SIZE	

20-6:	SIZE	The size of the circular buffer is SIZE*4 messages

All bits are cleared to 0 at reset.

Valid SIZE values are between 0 and 16384.

Note that each message occupies four 32-bit words.

Note that the resulting behavior of invalid SIZE values is undefined.

Note that only (SIZE*4)-1 messages can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

The width of the SIZE field may be made configurable by means of a VHDL generic. In this case it should be set to 16-1 bits width.

39.9.9 Transmit Channel Write Register [CanTxWR] R/W

*Table 414.*Transmit Channel Write Register

31 20	19 4	3 0
	WRITE	

19-4:	WRITE	Pointer to last written message +1

All bits are cleared to 0 at reset.

The WRITE field is written to in order to initiate a transfer, indicating the position +1 of the last message to transmit.

Note that it is not possible to fill the buffer. There is always one message position in buffer unused. Software is responsible for not over-writing the buffer on wrap around (i.e. setting WRITE=READ).

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

39.9.10 Transmit Channel Read Register [CanTxRD] R/W

*Table 415.*Transmit Channel Read Register

31 20	19 4	3 0
	READ	

19-4:	READ	Pointer to last read message +1

All bits are cleared to 0 at reset.

The READ field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last message transmitted.

Note that the READ field can be use to read out the progress of a transfer.

Note that the READ field can be written to in order to set up the starting point of a transfer. This should only be done while the transmit channel is not enabled.

Note that the READ field can be automatically incremented even if the transmit channel has been disabled, since the last requested transfer is not aborted until CAN bus arbitration is lost.

When the Transmit Channel Read Pointer catches up with the Transmit Channel Write Register, an interrupt is generated (TxEmpty). Note that this indicates that all messages in the buffer have been transmitted.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

39.9.11 Transmit Channel Interrupt Register [CanTxIRQ] R/W

*Table 416.*Transmit Channel Interrupt Register

31 20	19 4	3 0
	IRQ	

19-4:	IRQ	Interrupt is generated when CanTxRD.READ=IRQ, as a consequence of a message transmission

All bits are cleared to 0 at reset.

Note that this indicates that a programmed number of messages have been transmitted.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

39.9.12 Receive Channel Control Register [CanRxCTRL] R/W

*Table 417.*Receive Channel Control Register

31 2	1	0
	OnGoing	Enable

1:	ONGOING	Reception ongoing (read-only)
0:	ENABLE	Enable channel

All bits are cleared to 0 at reset.

Note that in the case an AHB bus error occurs during an access while fetching transmit data, and the CanCONF.ABORT bit is 1b, then the ENALBE bit will be reset automatically.

At the time the ENABLE is cleared to 0b, any ongoing message reception is not aborted

Note that the ONGOING bit being 1b indicates that message reception is ongoing and that configuration of the channel is not safe.

39.9.13 Receive Channel Address Register [CanRxADDR] R/W

*Table 418.*Receive Channel Address Register

31 10	9 0
ADDR	

31-10:	ADDR	Base address for circular buffer

All bits are cleared to 0 at reset.

39.9.14 Receive Channel Size Register [CanRxSIZE] R/W

*Table 419.*Receive Channel Size Register

31 21	20 6	5 0
	SIZE	

20-6:	SIZE	The size of the circular buffer is SIZE*4 messages

All bits are cleared to 0 at reset.

Valid SIZE values are between 0 and 16384.

Note that each message occupies four 32-bit words.

Note that the resulting behavior of invalid SIZE values is undefined.

Note that only (SIZE*4)-1 messages can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

The width of the SIZE field may be made configurable by means of a VHDL generic. In this case it should be set to 16-1 bits width.

39.9.15 Receive Channel Write Register [CanRxWR] R/W

*Table 420.*Receive Channel Write Register

31 20	19 4	3 0
	WRITE	

19-4:	WRITE	Pointer to last written message +1

All bits are cleared to 0 at reset.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

The WRITE field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last message received.

Note that the WRITE field can be use to read out the progress of a transfer.

Note that the WRITE field can be written to in order to set up the starting point of a transfer. This should only be done while the receive channel is not enabled.

39.9.16 Receive Channel Read Register [CanRxRD] R/W

*Table 421.*Receive Channel Read Register

31 20	19 4	3 0
	READ	

19-4:	READ	Pointer to last read message +1

All bits are cleared to 0 at reset.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

The READ field is written to in order to release the receive buffer, indicating the position +1 of the last message that has been read out.

Note that it is not possible to fill the buffer. There is always one message position in buffer unused. Software is responsible for not over-reading the buffer on wrap around (i.e. setting WRITE=READ).

39.9.17 Receive Channel Interrupt Register [CanRxIRQ] R/W

*Table 422.*Receive Channel Interrupt Register

31 20	19 4	3 0
	IRQ	

19-4:	IRQ	Interrupt is generated when CanRxWR.WRITE=IRQ, as a consequence of a message reception

All bits are cleared to 0 at reset.

Note that this indicates that a programmed number of messages have been received.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

39.9.18 Receive Channel Mask Register [CanRxMASK] R/W

*Table 423.*Receive Channel Mask Register

31	30	29	28 0
			AM

28-0:	AM	Acceptance Mask, bits set to 1b are taken into account in the comparison between the received message ID and the CanRxCODE.AC field

All bits are set to 1 at reset.

Note that Base ID is bits 28 to 18 and Extended ID is bits 17 to 0.

39.9.19 Receive Channel Code Register [CanRxCODE] R/W

*Table 424.*Receive Channel Code Register

31	30	29	28 0
			AC

28-0:	AC	Acceptance Code, used in comparison with the received message

All bits are cleared to 0at reset.

Note that Base ID is bits 28 to 18 and Extended ID is bits 17 to 0.

A message ID is matched when:

((Received-ID XOR CanRxCODE.AC) AND CanRxMASS.AM) = 0

39.9.20 Interrupt registers

The interrupt registers give complete freedom to the software, by providing means to mask interrupts, clear interrupts, force interrupts and read interrupt status.

When an interrupt occurs the corresponding bit in the Pending Interrupt Register is set. The normal sequence to initialize and handle a module interrupt is:

- Set up the software interrupt-handler to accept an interrupt from the module.
- Read the Pending Interrupt Register to clear any spurious interrupts.
- Initialize the Interrupt Mask Register, unmasking each bit that should generate the module interrupt.
- When an interrupt occurs, read the Pending Interrupt Status Register in the software interrupt-handler to determine the causes of the interrupt.
- Handle the interrupt, taking into account all causes of the interrupt.
- Clear the handled interrupt using Pending Interrupt Clear Register.

Masking interrupts: After reset, all interrupt bits are masked, since the Interrupt Mask Register is zero. To enable generation of a module interrupt for an interrupt bit, set the corresponding bit in the Interrupt Mask Register.

Clearing interrupts: All bits of the Pending Interrupt Register are cleared when it is read or when the Pending Interrupt Masked Register is read. Reading the Pending Interrupt Masked Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register. Selected bits can be cleared by writing ones to the bits that shall be cleared to the Pending Interrupt Clear Register.

Forcing interrupts: When the Pending Interrupt Register is written, the resulting value is the original contents of the register logically OR-ed with the write data. This means that writing the register can force (set) an interrupt bit, but never clear it.

Reading interrupt status: Reading the Pending Interrupt Status Register yields the same data as a read of the Pending Interrupt Register, but without clearing the contents.

Reading interrupt status of unmasked bits: Reading the Pending Interrupt Masked Status Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register, but without clearing the contents.

The interrupt registers comprise the following:

• Pending Interrupt Masked Status Register	[CanPIMSR]	R
• Pending Interrupt Masked Register	[CanPIMR]	R
• Pending Interrupt Status Register	[CanPISR]	R
• Pending Interrupt Register	[CanPIR]	R/W
• Interrupt Mask Register	[CanIMR]	R/W
• Pending Interrupt Clear Register	[CanPICR]	W

*Table 425.*Interrupt registers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
															Tx Loss

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rx Miss	Tx Err Cntr	Rx Err Cntr	Tx Syn c	Rx Syn c	Tx	Rx	Tx Emp ty	Rx Full	Tx IRQ	Rx IRQ	Tx AH B Err	Rx AH B Err	OR	Off	Pass

16:	TxLoss	Message arbitration lost during transmission (could be caused by communications error, as indicated by other interrupts as well)
15:	RxMiss	Message filtered away during reception
14:	TxErrCntr	Transmission error counter incremented
13:	RxErrCntr	Reception error counter incremented
12:	TxSync	Synchronization message transmitted
11:	RxSync	Synchronization message received
10:	Tx	Successful transmission of message
9:	Rx	Successful reception of message
8:	TxEmpty	Successful transmission of all messages in buffer
7:	RxFull	Successful reception of all messages possible to store in buffer
6:	TxIRQ	Successful transmission of a predefined number of messages
5:	RxIRQ	Successful reception of a predefined number of messages
4:	TxAHBErr	AHB error during transmission
3:	RxAHBErr	AHB error during reception
2:	OR	Over-run during reception
1:	OFF	Bus-off condition

0:	PASS	Error-passive condition

All bits in all interrupt registers are reset to 0b after reset.

Note that the TxAHBErr interrupt is generated in such way that the corresponding read and write pointers are valid for failure analysis. The interrupt generation is independent of the CanCONF.ABORT field setting.

Note that the RxAHBErr interrupt is generated in such way that the corresponding read and write pointers are valid for failure analysis. The interrupt generation is independent of the CanCONF.ABORT field setting.

39.10 Memory mapping

The CAN message is represented in memory as shown in table 426.

*Table 426.*CAN message representation in memory.

AHB addr

0x0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	IDE	RT R	-	bID											eID	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	eID															

0x4	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	DLC				-	-	-	-	TxErrCntr							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RxErrCntr								-	-	-	-	Ahb Err	OR	Off	Pass

0x8	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Byte 0 (first transmitted)								Byte 1							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Byte 2								Byte 3							

0xC	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Byte 4								Byte 5							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Byte 6								Byte 7 (last transmitted)							

Values:	Levels according to CAN standard:	1b is recessive, 0b is dominant	
Legend:	Naming and number in according to CAN standard		
IDE	Identifier Extension:	1b for Extended Format, 0b for Standard Format	
RTR	Remote Transmission Request:	1b for Remote Frame, 0b for Data Frame	
bID	Base Identifier		
eID	Extended Identifier		
DLC	Data Length Code, according to CAN standard:		
		0000b	0 bytes
		0001b	1 byte
		0010b	2 bytes
		0011b	3 bytes

		0100b	4 bytes
		0101b	5 bytes
		0110b	6 bytes
		0111b	7 bytes
		1000b	8 bytes
		OTHERS	illegal
TxErrCntr	Transmission Error Counter		
RxErrCntr	Reception Error Counter		
AHBErr	AHB interface blocked due to AHB Error when 1b		
OR	Reception Over run when 1b		
OFF	Bus Off mode when 1b		
PASS	Error Passive mode when 1b		
Byte 00 to 07	Transmit/Receive data, Byte 00 first Byte 07 last		

39.11 Vendor and device identifiers

The module has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x03D. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

39.12 Configuration options

Table 427 shows the configuration options of the core (VHDL generics).

*Table 427.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFC#
pirq	Interrupt line used by the GRCAN.	0 - NAHBIRQ-1	0
singleirq	Implement only one common interrupt	0 - 1	0
txchannels	Number of transmit channels	1 - 1	1
rxchannels	Number of receive channels	1 - 1	1
ptrwidth	Width of message pointers	16 - 16	16

39.13 Signal descriptions

Table 428 shows the interface signals of the core (VHDL ports).

*Table 428.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-
CANI	Rx[1:0]	Input	Receive lines	-
CANO	Tx[1:0]	Output	Transmit lines	-
	En[1:0]		Transmit enables	-

* see GRLIB IP Library User’s Manual

39.14 Library dependencies

Table 429 shows the libraries used when instantiating the core (VHDL libraries).

*Table 429.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	CAN	Signals, component	GRCAN component and signal declarations.

39.15 Instantiation

This example shows how the core can be instantiated.

```
library  ieee;
use      ieee.std_logic_1164.all;

library  gaisler;
use      gaisler.can.all;

entity example is
   generic (
      padtech:       in    integer := 0);
   port (
      -- CAN interface
      cantx:         out   std_logic_vector(1 downto 0);
      canrx:         in    std_logic_vector(1 downto 0);
      canen:         out   std_logic_vector(1 downto 0);

...

   -- Signal declarations
   signal   rstn:          std_ulogic;
   signal   clk:           std_ulogic;

   signal   ahbmo:         ahb_mst_out_vector := (others => ahbm_none);
   signal   ahbmi:         ahb_mst_in_type;

   signal   apbi:          apb_slv_in_type;
   signal   apbo:          apb_slv_out_vector := (others => apb_none);

   signal   cani0:         can_in_type;
   signal   cano0:         can_out_type;

...

   -- Component instantiation
   grcan0: grcan
      generic map (
         hindex         => 1,
         pindex         => 1,
         paddr          => 16#00C",
         pmask          => 16#FFC",
         pirq           => 1,
         txchannels     => 1,
         rxchannels     => 1,
         ptrwidth       => 16)
      port map (
         rstn           => rstn,
         clk            => clk,
         apbi           => apbi,
         apbo           => apbo(1),
         ahbi           => ahbmi,
         ahbo           => ahbmo(1),
         cani           => cani0,
         cano           => cano0);
```

```
cantx0_pad : outpad
   generic map (tech => padtech) port map (cantx(0), cani0.tx(0));

canrx0_pad : inpad
   generic map (tech => padtech) port map (canrx(0), cani0.rx(0));

canen0_pad : outpad
   generic map (tech => padtech) port map (canen(0), cani0.en(0));

cantx1_pad : outpad
   generic map (tech => padtech) port map (cantx(1), cani0.tx(1));

canrx1_pad : inpad
   generic map (tech => padtech) port map (canrx(1), cani0.rx(1));

canen1_pad : outpad
   generic map (tech => padtech) port map (canen(1), cani0.en(1));
```

40 GRCLKGATE - Clock gating unit

40.1 Overview

The clock gating unit provides a mean to save power by disabling the clock to unused functional blocks. The core provides a mechanism to automatically disabling the clock to LEON processors in power-down mode, and optionally also to disable the clock for shared floating-point units.

The core provides a register interface via its APB slave bus interface.

40.2 Operation

The operation of the clock gating unit is controlled through four registers: the unlock, clock enable, core reset and CPU/FPU override registers. The clock enable register defines if a clock is enabled or disabled. A ‘1’ in a bit location will enable the corresponding clock, while a ‘0’ will disable the clock. The core reset register allows to generate a reset signal for each generated clock. A reset will be generated as long as the corresponding bit is set to ‘1’. The bits in clock enable and core reset registers can only be written when the corresponding bit in the unlock register is 1. If a bit in the unlock register is 0, the corresponding bits in the clock enable and core reset registers cannot be written.

To gate the clock for a core, the following procedure should be applied:

1. Disable the core through software to make sure it does not initialize any AHB accesses

2. Write a 1 to the corresponding bit in the unlock register

3. Write a 0 to the corresponding bit in the clock enable register

4. Write a 0 to the corresponding bit in the unlock register

To enable the clock for a core, the following procedure should be applied

1. Write a 1 to the corresponding bit in the unlock register

2. Write a 1 to the corresponding bit in the core reset register

3. Write a 1 to the corresponding bit in the clock enable register

4. Write a 1 to the corresponding bit in the core reset register

5. Write a 0 to the corresponding bit in the unlock register

The clock gating unit also provides gating for the processor core and, optionally, floating-point units. A processor core will be automatically gated off when it enters power-down mode. Any shared FPU will be gated off when all processor cores connected to the FPU have floating-point disabled or when all connected processor cores are in power-down mode.

Processor/FPU clock gating can be disabled by writing ‘1’ to bit 0 of the CPU/FPU override register.

40.2.1 Shared FPU

For systems with shared FPU, a processor may be clock gated off while the connected FPU continues to be clocked. The power-down instruction may overtake a previously issued floating-point instruction and cause the processor to be gated off before the floating-point operation has completed. This can in turn lead to the processor not reacting to the completion of the floating-point operation and to a subsequent processor freeze after the processor wakes up and continues to wait for the completion of the floating-point operation.

In order to avoid this, software must make sure that all floating-point operations have completed before the processor enters power-down. This is generally not a problem in real-world applications as the power-down instruction is typically used in a idle loop and floating-point results have been stored to memory before entering the idle loop. To make sure that there are no floating-point operations pending, software should perform a store of the %fsr register before the power-down instruction.

40.2.2 Scan test support

When scan test support is configured into the core and the scanen signal is active, all clock gates are set to pass-through. Also, all registers in the core are clocked on the rising edge of the clock. The scan-enable signal is provided via the APB input record.

A separate ungate active-high input signal that also sets all clock gates to pass-through can be enabled in the core.

40.3 Registers

The core's registers are mapped into APB address space.

Table 430. Clock gate unit registers

APB address offset	**Register**
0x00	Unlock register
0x04	Clock enable register
0x08	Core reset register
0x0C	CPU/FPU override register

40.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x02C. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

40.5 Configuration options

Table 431 shows the configuration options of the core (VHDL generics).

*Table 431.*Configuration options

Generic	Function	Allowed range	Default
tech	Clock/fabrication technology	0 to NTECH-1	0
pindex	Selects which APB select signal (PSEL) will be used to access the unit		
paddr	The 12-bit MSB APB address	0 to 16#FFF#	0
pmask	The APB address mask	0 to 16#FFF#	16#FFF#
ncpu	Number of processors that will connect to the unit	-	1
nclks	Number of peripheral units (clock/reset pairs) in addition to any processors and floating-point units that will connect to the unit.	0 - 31	8
emask	Bit mask where bit n (0 is the least significant bit) decides if a unit should be enabled (1) or disabled (0) after system reset.	0 - 16#FFFFFFFF#	0
extemask	If this generic is set to a non-zero value then the after-reset-enable-mask will be taken from the input signal epwen.	0 - 1	0
scantest	Enable scan test support	0 - 1	0
edges	Extra clock edges provided by the clock gate unit after reset completes. CPUs get *edges* + 3 rising edges after reset and other cores get *edges* + 1 rising edges after system reset.	-	0
noinv	Do not use inverted clock for clock gate enable register. This generic can be set to one for technologies that have glitch free clock gates.	0 - 1	0
fpush	Selects FPU configuration 0: System has processors without, or with dedicated, FPUs 1: System has one FPU shared between all processors 3: System has one FPU for each parir of processors. (FPU0 is connected to CPU0 and CPU1, FPU1 is connected to CPU2 and CPU3, ...)	0 - 2	0
ungateen	Enable separate ungate input for asynchronous un-gating of all clocks.		

40.6 Signal descriptions

Table 432 shows the interface signals of the core (VHDL ports).

*Table 432.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLKIN	N/A	Input	Clock	-
PWD	N/A	Input	Power-down signal from processor cores	High
FPEN	N/A	Input	Floating-point enable signal from processor cores, only used in configurations with shared FPU.	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GCLK[nclks-1:0]	N/A	Output	Clock(s) to peripheral unit	-
RESET[nclks-1:0]	N/A	Output	Reset(s) to peripheral units	Low
CLKAHB	N/A	Output	Clock to non-gated units	-
CLKCPU[ncpu-1:0]	N/A	Output	Clock to processor cores	-
ENABLE[nclks-1:0]	N/A	Output	Enable signal(s) for peripheral units	High
CLKFPU[nfpu**:0]	N/A	Output	Clock to shared floating-point units, only used in configurations with shared FPU.	-
EPWEN	N/A	Input	External enable reset vector	High
UNGATE	N/A	Input	Ungate all clocks for test mode (only used if enabled in configuration)	High

* see GRLIB IP Library User's Manual
** where nfpu = (fpush/2)*(ncpu/2-1)

40.7 Library dependencies

Table 433 shows libraries used when instantiating the core (VHDL libraries).

*Table 433.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

40.8 Instantiation

This example shows how the core can be instantiated.

```
clkg0: grclkgate
      generic map (
        tech     => fabtech,
        pindex   => 4,
        paddr    => 16#040#,
        pmask    => 16#fff#,
        ncpu     => CFG_NCPU,
        nclks    => NCLKS,
        emask    => 0,                -- Don't care
        extemask => 1,                -- Reset value defined by input vector (epwen below)
        scantest => scantest,
        edges    => CG_EDGES,
        noinv    => CG_NOINV,
        fpush    => CFG_GRFPUSH)
```

```
port map(
  rst      => rstn,      -- from reset generator
  clkin    => ahb_clk,   -- from clock generator
  pwd      => pwd,  -- from processors, typically dsuo.pwd(CFG_NCPU-1 downto 0)
  fpen     => fpen, -- from processors, if shared FPU is used
  apbi     => apbi,
  apbo     => apbo(4),
  gclk     => gclk,      -- clock to (gated) peripheral cores
  reset    => grst,      -- reset to (gated) peripheral cores
  clkahb   => clkm,      -- clock to AMBA system (not gated)
  clkcpu   => cpuclk,    -- clock to processor cores
  enable   => clkenable, -- enable(n) signals that peripheral n is enabled
  clkfpu   => fpuclk,    -- clock to any shared FPU cores
  epwen    => pwenmask,  -- signal to set enable-after-reset
  ungate   => gnd);
```

41 GRECC - Elliptic Curve Cryptography

41.1 Overview

Elliptic Curve Cryptography (ECC) is used as a public key mechanism. The computational burden that is inhibited by ECC is less than the one of RSA. ECC provides the same level of security as RSA but with a significantly shorter key length. ECC is well suited for application in mobile communication.

The GRECC core implements encryption and decryption for an elliptic curve based on 233-bit key and point lengths. The implemented curve is denoted as *sect233r1* or *B-233*.

The *sect233r1* elliptic curve domain parameters are specified in the "Standards for Efficient Cryptography (SEC) - SEC2: Recommended Elliptic Curve Domain Parameters" document. The document is established by the Standards for Efficient Cryptography Group (SECG).

The *B-233* elliptic curve domain parameters are specified in the "Digital Signature Standard (DSS)" document, Federal Information Processing Standards (FIPS) Publication 186-2. The document is established by the National Institute of Standards and Technology (NIST).

The GRECC can be used with algorithms such as:

- Elliptic Curve Digital Signature Algorithm DSA (ECDSA), which appears in FIPS 186-2, IEEE 1363-2000 and ISO/IEC 15946-2
- Elliptic Curve El Gamal Method (key exchange protocol)
- Elliptic Curve Diffie-Hellman (ECDH) (key agreement protocol)

The core provides the following internal AMBA APB slave interface, with sideband signals as per [GRLIB] including:

- interrupt bus
- configuration information
- diagnostic information

The core can be partition in the following hierarchical elements:

- Elliptic Curve Cryptography (ECC) core
- AMBA APB slave
- GRLIB plug&play wrapper

Note that the core can also be used without the GRLIB plug&play information.

41.2 Operation

Elliptic Curve Cryptography (ECC) is an asymmetric cryptographic approach (also known as public key cryptography) that applies different keys for encryption and decryption. The most expensive operation during both encryption and decryption is the elliptic curve point multiplication. Hereby, a point on the elliptic curve is multiplied with a long integer (*k*P* multiplication). The bit sizes of the coordinates of the point *P*=(*x*, *y*) and the factor *k* have a length of hundreds of bits.

In this implementation the key and the point lengths are 233 bit, so that for every key there are 8 write cycles necessary and for every point (consisting of *x* and *y*) there are 16 write cycles necessary. After at least 16700 clock cycles the result can be read out.

The key is input via eight registers. The input point $P_{in}=(x, y)$ is written via eight registers for x and eight registers for y. After the last y input register is written, the encryption or decryption is started. The progress can be observed via the status register. When the operation is completed, an interrupt is generated. The output point $P_{out}=(x, y)$ is then read out via eight registers for x and eight registers for y.

41.3 Advantages

The main operation in ECC is the k*P multiplication. One k*P multiplication requires about 1500 field multiplications in the base field, which is the most expensive base operation. The complexity of a field multiplication can be reduced by applying the Karatsuba method. Normally the Karatsuba approach is applied recursively. The GRECC core includes an iterative implementation of the Karatsuba method which allows to realize area efficient hardware accelerators for the *k*P* multiplication. Hardware accelerators which are realized applying an iterative approach need up to 60 per cent less area and about 30 per cent less energy per multiplication than the recursive variants.

41.4 Background

The Standards for Efficient Cryptography Group (SECG) was initiated by Certicom Corporation to address the difficulty vendors and users face when building and deploying interoperable security solutions. The SECG is a broad international coalition comprised of leading technology companies and key industry players in the information security industry. One of the goals is to enable the effective incorporation of Elliptic Curve Cryptographic (ECC) technology into these various cryptographic solutions.

The Standards for Efficient Cryptography Group (SECG) has develop two sets of documents. The first set, under the name SEC, specifies interoperable cryptographic technologies and solutions. The second set, Guidelines for Efficient Cryptography (GEC), provides background information on elliptic curve cryptography and recommendations for ECC parameter and curve selection.

The Federal Information Processing Standards Publication Series of the National Institute of Standards and Technology (NIST) is the official series of publications relating to standards and guidelines adopted under the provisions of the Information Technology Management Reform Act.

This Digital Signature Standard (DSS) specifies a suite of algorithms which can be used to generate a digital signature. Digital signatures are used to detect unauthorized modifications to data and to authenticate the identity of the signatory. In addition, the recipient of signed data can use a digital signature in proving to a third party that the signature was in fact generated by the signatory. This is known as nonrepudiation since the signatory cannot, at a later time, repudiate the signature.

41.5 233-bit elliptic curve domain parameters

The core implements the 233-bit elliptic curve domain parameters *sect233r1*, or the equivalent *B-233*, which are verifiably random parameters. The following specification is established in “Standards for Efficient Cryptography (SEC) - SEC 2: Recommended Elliptic Curve Domain Parameters”. The verifiably random elliptic curve domain parameters over F_{2^m} are specified by the septuple T = (m; f (x); a; b; G; n; h) where m = 233 and the representation of $F_{2^{233}}$ is defined by:

$f(x) = x^{233}+x^{74}+1$

The curve E: $y^2+xy = x^3+ax^2+b$ over F_{2^m} is defined by:

a = 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000001

b = 0066 647EDE6C 332C7F8C 0923BB58 213B333B 20E9CE42 81FE115F 7D8F90AD

The base point G in compressed form is:

G = 0300FA C9DFCBAC 8313BB21 39F1BB75 5FEF65BC 391F8B36 F8F8EB73 71FD558B

and in uncompressed form is:

G = 04 00FAC9DF CBAC8313 BB2139F1 BB755FEF 65BC391F 8B36F8F8
EB7371FD 558B0100 6A08A419 03350678 E58528BE BF8A0BEF F867A7CA
36716F7E 01F81052

Finally the order *n* of *G* and the cofactor are:

n = 0100 00000000 00000000 00000000 0013E974 E72F8A69 22031D26 03CFE0D7

h = 02

41.6 Throughput

The data throughput for the GRECC core is around 233/16700 bits per clock cycle, i.e. approximately 13.9 kbits per MHz.

The underlaying EEC core has been implemented in a dual crypto chip on 250 nm technology as depicted in the figure below. The throughput at 33 MHz operating frequency was 850 kbit/s, the power consumption was 56,8 mW, and the size was 48,5 kgates.

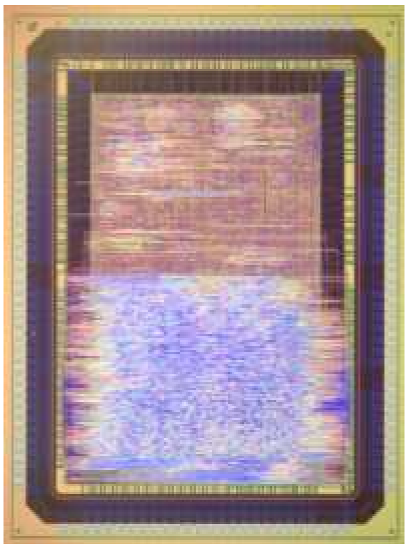


Figure 142. Dual Crypto Chip

41.7 Characteristics

The GRECC core has been synthesized for a Xilinx Virtex-2 XC2V6000-4 devices with the following results:

- LUTs: 12850 (19%)
- Frequency:93 MHz

41.8 Registers

The core is programmed through registers mapped into APB address space.

*Table 434.*GRECC registers

APB address offset	Register
16#020#	Key 0 Register
16#024#	Key 1 Register
16#028#	Key 2 Register
16#02C#	Key 3 Register
16#030#	Key 4 Register
16#034#	Key 5 Register
16#038#	Key 6 Register
16#03C#	Key 7 Register
16#040#	Point X Input 0 Register
16#044#	Point X Input 1 Register
16#048#	Point X Input 2 Register
16#04C#	Point X Input 3 Register
16#050#	Point X Input 4 Register
16#054#	Point X Input 5 Register
16#058#	Point X Input 6 Register
16#05C#	Point X Input 7 Register
16#060#	Point Y Input 0 Register
16#064#	Point Y Input 1 Register
16#068#	Point Y Input 2 Register
16#06C#	Point Y Input 3 Register
16#070#	Point Y Input 4 Register
16#074#	Point Y Input 5 Register
16#078#	Point Y Input 6 Register
16#07C#	Point Y Input 7 Register
16#0A0#	Point X Output 0 Register
16#0A4#	Point X Output 1 Register
16#0A8#	Point X Output 2 Register
16#0AC#	Point X Output 3 Register
16#0B0#	Point X Output 4 Register
16#0B4#	Point X Output 5 Register
16#0B8#	Point X Output 6 Register
16#0BC#	Point X Output 7 Register
16#0C0#	Point Y Output 0 Register
16#0C4#	Point Y Output 1 Register
16#0C8#	Point Y Output 2 Register
16#0CC#	Point Y Output 3 Register
16#0D0#	Point Y Output 4 Register
16#0D4#	Point Y Output 5 Register
16#0D8#	Point Y Output 6 Register
16#0DC#	Point Y Output 7 Register
16#0FC#	Status Register

41.8.1 Key 0 to 7 Registers (W)

*Table 435.*Key 0 Register (least significant)

31 0
KEY(31 downto 0)

*Table 436.*Key 1 Register

31 0
KEY(63 downto32)

*Table 437.*Key 2 Register

31 0
KEY(95 downto 64)

*Table 438.*Key 3 Register

31 0
KEY(127 downto 96)

*Table 439.*Key 4 Register

31 0
KEY(159 downto 128)

*Table 440.*Key 5 Register

31 0
KEY(191 downto 160)

*Table 441.*Key 6 Register

31 0
KEY(223 downto 192)

*Table 442.*Key 7 Register (most significant)

31 9	8 0
-	KEY(232 downto 224)

41.8.2 Point X Input 0 to 7 Registers (W)

*Table 443.*Point X Input 0 Register (least significant)

31 0
X(31 downto 0)

*Table 444.*Point X Input 1 Register

31 0
X(63 downto32)

*Table 445.*Point X Input 2 Register

31 0
X(95 downto 64)

*Table 446.*Point X Input 3 Register

31 0
X(127 downto 96)

*Table 447.*Point X Input 4 Register

31 0
X(159 downto 128)

*Table 448.*Point X Input 5 Register

31 0
X(191 downto 160)

*Table 449.*Point X Input 6 Register

31 0
X(223 downto 192)

*Table 450.*Point X Input 7 Register (most significant)

31 9	8 0
-	X(232 downto 224)

41.8.3 Point Y Input 0 to 7 Registers (W)

*Table 451.*Point Y Input 0 Register (least significant)

31 0
Y(31 downto 0)

*Table 452.*Point Y Input 1 Register

31 0
Y(63 downto32)

*Table 453.*Point Y Input 2 Register

31 0
Y(95 downto 64)

*Table 454.*Point Y Input 3 Register

31 0
Y(127 downto 96)

*Table 455.*Point Y Input 4 Register

31 0
Y(159 downto 128)

*Table 456.*Point Y Input 5 Register

31 0
Y(191 downto 160)

*Table 457.*Point Y Input 6 Register

31 0
Y(223 downto 192)

*Table 458.*Point Y Input 7 Register (most significant)

31 9	8 0
-	Y(232 downto 224)

The encryption or decryption operation is started when the Point Y Input 7 Register is written.

41.8.4 Point X Output 0 to 7 Registers (R)

*Table 459.*Point X Output 0 Register (least significant)

31 0
X(31 downto 0)

*Table 460.*Point X Output 1 Register

31 0
X(63 downto32)

*Table 461.*Point X Output 2 Register

31 0
X(95 downto 64)

*Table 462.*Point X Output 3 Register

31 0
X(127 downto 96)

*Table 463.*Point X Output 4 Register

31 0
X(159 downto 128)

*Table 464.*Point X Output 5 Register

31 0
X(191 downto 160)

*Table 465.*Point X Output 6 Register

31 0
X(223 downto 192)

*Table 466.*Point X Output 7 Register (most significant)

31 9	8 0
-	X(232 downto 224)

41.8.5 Point Y Output 0 to 7 Registers (R)

*Table 467.*Point Y Output 0 Register (least significant)

31 0
Y(31 downto 0)

*Table 468.*Point Y Output 1 Register

31 0
Y(63 downto32)

*Table 469.*Point Y Output 2 Register

31 0
Y(95 downto 64)

*Table 470.*Point Y Output 3 Register

31 0
Y(127 downto 96)

*Table 471.*Point Y Output 4 Register

31 0
Y(159 downto 128)

*Table 472.*Point Y Output 5 Register

31 0
Y(191 downto 160)

*Table 473.*Point Y Output 6 Register

31 0
Y(223 downto 192)

*Table 474.*Point Y Output 7 Register (most significant)

31 9	8 0
-	Y(232 downto 224)

41.8.6 Status Register (R)

*Table 475.*Status Register

31 1	0
.	FS M

31-1:	-	Unused
0:	FSM	0 when ongoing, 1 when idle or ready

41.9 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x074. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

41.10 Configuration options

Table 476 shows the configuration options of the core (VHDL generics).

*Table 476.*Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB BAR	0 - 16#FFF#	0
pmask	Mask field of the APB BAR	0 - 16#FFF#	16#FFC#
pirq	Interrupt line used by the GRECC	0 - NAHBIRQ-1	0

41.11 Signal descriptions

Table 477 shows the interface signals of the core (VHDL ports).

*Table 477.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
DEBUG[10:0]	N/A	Output	Debug information	-

* see GRLIB IP Library User’s Manual

Note that the ECC core can also be used without the GRLIB plug&play information. The AMBA APB signals are then provided as IEEE Std_Logic_1164 compatible scalars and vectors.

41.12 Library dependencies

Table 478 shows libraries used when instantiating the core (VHDL libraries).

*Table 478.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	CRYPTO	Component	GRECC component declarations

41.13 Instantiation

This example shows how the core can be instantiated.

```
library  ieee;
use      ieee.std_logic_1164.all;

library  grlib;
use      grlib.amba.all;

library  gaisler;
use      gaisler.crypto.all;
...
...
   signal debug: std_logic_vector(10 downto 0);
..
```

```
..
   grecc0: grecc
      generic map (
         pindex         => pindex,
         paddr          => paddr,
         pmask          => pmask,
         pirq           => pirq)
      port map (
         rstn           => rstn,
         clk            => clk,
         apbi           => apbi,
         apbo           => apbo(pindex),
         debug          => debug);
```

42 GRETH - Ethernet Media Access Controller (MAC) with EDCL support

42.1 Overview

Aeroflex Gaisler's Ethernet Media Access Controller (GRETH) provides an interface between an AMBA-AHB bus and an Ethernet network. It supports 10/100 Mbit speed in both full- and half-duplex. The AMBA interface consists of an APB interface for configuration and control and an AHB master interface which handles the dataflow. The dataflow is handled through DMA channels. There is one DMA engine for the transmitter and one for the receiver. Both share the same AHB master interface. The ethernet interface supports both the MII and RMII interfaces which should be connected to an external PHY. The GRETH also provides access to the MII Management interface which is used to configure the PHY.

Optional hardware support for the Ethernet Debug Communication Link (EDCL) protocol is also provided. This is an UDP/IP based protocol used for remote debugging.

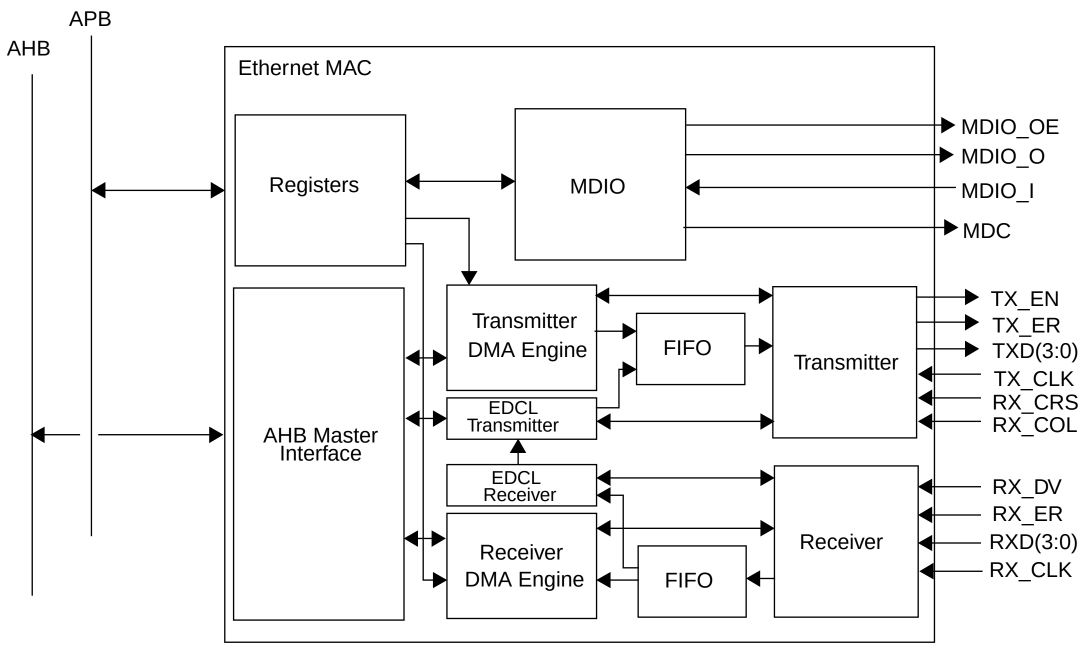


Figure 143. Block diagram of the internal structure of the GRETH.

42.2 Operation

42.2.1 System overview

The GRETH consists of 3 functional units: The DMA channels, MDIO interface and the optional Ethernet Debug Communication Link (EDCL).

The main functionality consists of the DMA channels which are used to transfer data between an AHB bus and an Ethernet network. There is one transmitter DMA channel and one Receiver DMA channel. The operation of the DMA channels is controlled through registers accessible through the APB interface.

The MDIO interface is used for accessing configuration and status registers in one or more PHYs connected to the MAC. The operation of this interface is also controlled through the APB interface.

The optional EDCL provides read and write access to an AHB bus through Ethernet. It uses the UDP, IP, ARP protocols together with a custom application layer protocol to accomplish this. The EDCL contains no user accessible registers and always runs in parallel with the DMA channels.

The Media Independent Interface (MII) is used for communicating with the PHY. There is an Ethernet transmitter which sends all data from the AHB domain on the Ethernet using the MII interface. Correspondingly, there is an Ethernet receiver which stores all data from the Ethernet on the AHB bus. Both of these interfaces use FIFOs when transferring the data streams. The GRETH also supports the RMII which uses a subset of the MII signals.

The EDCL and the DMA channels share the Ethernet receiver and transmitter.

42.2.2 Protocol support

The GRETH is implemented according to IEEE standard 802.3-2002 and IEEE standard 802.3Q-2003. There is no support for the optional control sublayer. This means that packets with type 0x8808 (the only currently defined ctrl packets) are discarded. The support for 802.3Q is optional and need to be enabled via generics.

42.2.3 Clocking

GRETH has three clock domains: The AHB clock, Ethernet receiver clock and the Ethernet transmitter clock. The ethernet transmitter and receiver clocks are generated by the external ethernet PHY, and are inputs to the core through the MII interface. The three clock domains are unrelated to each other and all signals crossing the clock regions are fully synchronized inside the core.

Both full-duplex and half-duplex operating modes are supported and both can be run in either 10 or 100 Mbit. The minimum AHB clock for 10 Mbit operation is 2.5 MHz, while 18 MHz is needed for 100 Mbit. Using a lower AHB clock than specified will lead to excessive packet loss.

42.2.4 RAM debug support

Support for debug accesses the core’s internal RAM blocks can be optionally enabled using the ramdebug VHDL generic. Setting it to 1 enables accesses to the transmitter and receiver RAM buffers and setting it to 2 enables accesses to the EDCL buffer in addition to the previous two buffers.

The transmitter RAM buffer is accessed starting from APB address offset 0x10000 which corresponds to location 0 in the RAM. There are 512 32-bit wide locations in the RAM which results in the last address being 0x107FC corresponding to RAM location 511 (byte addressing used on the APB bus).

Correspondingly the receiver RAM buffer is accessed starting from APB address offset 0x20000. The addresses, width and depth is the same.

The EDCL buffers are accessed starting from address 0x30000. The number of locations depend on the configuration and can be from 256 to 16384. Each location is 32-bits wide so the maximum address is 0x3FC and 0xFFFC correspondingly.

Before any debug accesses can be made the ramdebugen bit in the control register has to be set. During this time the debug interface controls the RAM blocks and normal operations is stopped. EDCL packets are not received. The MAC transmitter and receiver could still operate if enabled but the RAM buffers would be corrupt if debug accces are made simultaneously. Thus they MUST be disabled before the RAM debug mode is enabled.

42.2.5 Multibus version

There is a version of the core which has an additional master interface that can be used for the EDCL. Otherwise this version is identical to the basic version. The additional master interface is enabled with the edclsepahb VHDL generic. Then the ethi.edclsepahb signal control whether EDCL accesses are done on the standard master interface or the additional interface. Setting the signal to ‘0’ makes the EDCL use the standard master interface while ‘1’ selects the additional master. This signal is only sampled at reset and changes to this signal have no effect until the next reset.

42.3 Tx DMA interface

The transmitter DMA interface is used for transmitting data on an Ethernet network. The transmission is done using descriptors located in memory.

42.3.1 Setting up a descriptor.

A single descriptor is shown in table 479 and 480. The number of bytes to be sent should be set in the length field and the address field should point to the data. The address must be word-aligned. If the interrupt enable (IE) bit is set, an interrupt will be generated when the packet has been sent (this requires that the transmitter interrupt bit in the control register is also set). The interrupt will be generated regardless of whether the packet was transmitted successfully or not. The Wrap (WR) bit is also a control bit that should be set before transmission and it will be explained later in this section.

Table 479. GRETH transmit descriptor word 0 (address offset 0x0)

31 16	15	14	13	12	11	10 0
RESERVED	AL	UE	IE	WR	EN	LENGTH

31: 16	RESERVED
15	Attempt Limit Error (AL) - The packet was not transmitted because the maximum number of attempts was reached.
14	Underrun Error (UE) - The packet was incorrectly transmitted due to a FIFO underrun error.
13	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when the packet from this descriptor has been sent provided that the transmitter interrupt enable bit in the control register is set. The interrupt is generated regardless if the packet was transmitted successfully or if it terminated with an error.
12	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
11	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
10: 0	LENGTH - The number of bytes to be transmitted.

Table 480. GRETH transmit descriptor word 1 (address offset 0x4)

31 2	1 0
ADDRESS	RES

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

To enable a descriptor the enable (EN) bit should be set and after this is done, the descriptor should not be touched until the enable bit has been cleared by the GRETH.

42.3.2 Starting transmissions

Enabling a descriptor is not enough to start a transmission. A pointer to the memory area holding the descriptors must first be set in the GRETH. This is done in the transmitter descriptor pointer register. The address must be aligned to a 1 kB boundary. Bits 31 to 10 hold the base address of descriptor area while bits 9 to 3 form a pointer to an individual descriptor.The first descriptor should be located at the base address and when it has been used by the GRETH the pointer field is incremented by 8 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 1 kB boundary has been reached (the descriptor at address offset 0x3F8 has been used). The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 1 kB boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when a transmission is active.

The final step to activate the transmission is to set the transmit enable bit in the control register. This tells the GRETH that there are more active descriptors in the descriptor table. This bit should always be set when new descriptors are enabled, even if transmissions are already active. The descriptors must always be enabled before the transmit enable bit is set.

42.3.3 Descriptor handling after transmission

When a transmission of a packet has finished, status is written to the first word in the corresponding descriptor. The Underrun Error bit is set if the FIFO became empty before the packet was completely transmitted while the Attempt Limit Error bit is set if more collisions occurred than allowed. The packet was successfully transmitted only if both of these bits are zero. The other bits in the first descriptor word are set to zero after transmission while the second word is left untouched.

The enable bit should be used as the indicator when a descriptor can be used again, which is when it has been cleared by the GRETH. There are three bits in the GRETH status register that hold transmission status. The Transmitter Error (TE) bit is set each time an transmission ended with an error (when at least one of the two status bits in the transmit descriptor has been set). The Transmitter Interrupt (TI) is set each time a transmission ended successfully.

The transmitter AHB error (TA) bit is set when an AHB error was encountered either when reading a descriptor or when reading packet data. Any active transmissions were aborted and the transmitter was disabled. The transmitter can be activated again by setting the transmit enable register.

42.3.4 Setting up the data for transmission

The data to be transmitted should be placed beginning at the address pointed by the descriptor address field. The GRETH does not add the Ethernet address and type fields so they must also be stored in the data buffer. The 4 B Ethernet CRC is automatically appended at the end of each packet. Each descriptor will be sent as a single Ethernet packet. If the size field in a descriptor is greater than defined by maxsize generic + header size bytes, the packet will not be sent.

42.4 Rx DMA interface

The receiver DMA interface is used for receiving data from an Ethernet network. The reception is done using descriptors located in memory.

42.4.1 Setting up descriptors

A single descriptor is shown in table 481 and 482. The address field should point to a word-aligned buffer where the received data should be stored. The GRETH will never store more than defined by the maxisize generic + header size bytes to the buffer. If the interrupt enable (IE) bit is set, an interrupt will be generated when a packet has been received to this buffer (this requires that the receiver interrupt bit in the control register is also set). The interrupt will be generated regardless of whether the packet was received successfully or not. The Wrap (WR) bit is also a control bit that should be set before the descriptor is enabled and it will be explained later in this section.

Table 481. GRETH receive descriptor word 0 (address offset 0x0)

31 27	26	25 19	18	17	16	15	14	13	12	11	10 0
RESERVED	MC	RESERVED	LE	OE	CE	FT	AE	IE	WR	EN	LENGTH

31: 27	RESERVED
26	Multicast address (MC) - The destination address of the packet was a multicast address (not broadcast).
25: 19	RESERVED

Table 481. GRETH receive descriptor word 0 (address offset 0x0)

18	Length error (LE) - The length/type field of the packet did not match the actual number of received bytes.
17	Overrun error (OE) - The frame was incorrectly received due to a FIFO overrun.
16	CRC error (CE) - A CRC error was detected in this frame.
15	Frame too long (FT) - A frame larger than the maximum size was received. The excessive part was truncated.
14	Alignment error (AE) - An odd number of nibbles were received.
13	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when a packet has been received to this descriptor provided that the receiver interrupt enable bit in the control register is set. The interrupt is generated regardless if the packet was received successfully or if it terminated with an error.
12	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
11	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
10: 0	LENGTH - The number of bytes received to this descriptor.

Table 482. GRETH receive descriptor word 1 (address offset 0x4)

31 … 2	1 … 0
ADDRESS	RES

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

42.4.2 Starting reception

Enabling a descriptor is not enough to start reception. A pointer to the memory area holding the descriptors must first be set in the GRETH. This is done in the receiver descriptor pointer register. The address must be aligned to a 1 kB boundary. Bits 31 to 10 hold the base address of descriptor area while bits 9 to 3 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the GRETH the pointer field is incremented by 8 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 1 kB boundary has been reached (the descriptor at address offset 0x3F8 has been used). The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 1 kB boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when reception is active.

The final step to activate reception is to set the receiver enable bit in the control register. This will make the GRETH read the first descriptor and wait for an incoming packet.

42.4.3 Descriptor handling after reception

The GRETH indicates a completed reception by clearing the descriptor enable bit. The other control bits (WR, IE) are also cleared. The number of received bytes is shown in the length field. The parts of the Ethernet frame stored are the destination address, source address, type and data fields. Bits 17-14 in the first descriptor word are status bits indicating different receive errors. All four bits are zero after a reception without errors. The status bits are described in table 481.

Packets arriving that are smaller than the minimum Ethernet size of 64 B are not considered as a reception and are discarded. The current receive descriptor will be left untouched an used for the first packet arriving with an accepted size. The TS bit in the status register is set each time this event occurs.

If a packet is received with an address not accepted by the MAC, the IA status register bit will be set.

Packets larger than maximum size cause the FT bit in the receive descriptor to be set. The length field is not guaranteed to hold the correct value of received bytes. The counting stops after the word containing the last byte up to the maximum size limit has been written to memory.

The address word of the descriptor is never touched by the GRETH.

42.4.4 Reception with AHB errors

If an AHB error occurs during a descriptor read or data store, the Receiver AHB Error (RA) bit in the status register will be set and the receiver is disabled. The current reception is aborted. The receiver can be enabled again by setting the Receive Enable bit in the control register.

42.4.5 Accepted MAC addresses

In the default configuration the core receives packets with either the unicast address set in the MAC address register or the broadcast address. Multicast support can also be enabled and in that case a hash function is used to filter received multicast packets. A 64-bit register, which is accessible through the APB interface, determines which addresses should be received. Each address is mapped to one of the 64 bits using the hash function and if the bit is set to one the packet will be received. The address is mapped to the table by taking the 6 least significant bits of the 32-bit Ethernet crc calculated over the destination address of the MAC frame. A bit in the receive descriptor is set if a packet with a multicast address has been received to it.

42.5 MDIO Interface

The MDIO interface provides access to PHY configuration and status registers through a two-wire interface which is included in the MII interface. The GRETH provided full support for the MDIO interface. If it is not needed in a design it can be removed with a VHDL generic.

The MDIO interface can be used to access from 1 to 32 PHY containing 1 to 32 16-bit registers. A read transfer i set up by writing the PHY and register addresses to the MDIO Control register and setting the read bit. This caused the Busy bit to be set and the operation is finished when the Busy bit is cleared. If the operation was successful the Linkfail bit is zero and the data field contains the read data. An unsuccessful operation is indicated by the Linkfail bit being set. The data field is undefined in this case.

A write operation is started by writing the 16-bit data, PHY address and register address to the MDIO Control register and setting the write bit. The operation is finished when the busy bit is cleared and it was successful if the Linkfail bit is zero.

42.5.1 PHY interrupts

The core also supports status change interrupts from the PHY. A level sensitive interrupt signal can be connected on the mdint input. The mdint_pol vhdl generic can be used to select the polarity. The PHY status change bit in the status register is set each time an event is detected in this signal. If the PHY status interrupt enable bit is set at the time of the event the core will also generate an interrupt on the AHB bus.

42.6 Ethernet Debug Communication Link (EDCL)

The EDCL provides access to an on-chip AHB bus through Ethernet. It uses the UDP, IP and ARP protocols together with a custom application layer protocol. The application layer protocol uses an ARQ algorithm to provide reliable AHB instruction transfers. Through this link, a read or write transfer can be generated to any address on the AHB bus. The EDCL is optional and must be enabled with a generic.

42.6.1 Operation

The EDCL receives packets in parallel with the MAC receive DMA channel. It uses a separate MAC address which is used for distinguishing EDCL packets from packets destined to the MAC DMA channel. The EDCL also has an IP address which is set through generics. Since ARP packets use the Ethernet broadcast address, the IP-address must be used in this case to distinguish between EDCL ARP packets and those that should go to the DMA-channel. Packets that are determined to be EDCL packets are not processed by the receive DMA channel.

When the packets are checked to be correct, the AHB operation is performed. The operation is performed with the same AHB master interface that the DMA-engines use. The replies are automatically sent by the EDCL transmitter when the operation is finished. It shares the Ethernet transmitter with the transmitter DMA-engine but has higher priority.

42.6.2 EDCL protocols

The EDCL accepts Ethernet frames containing IP or ARP data. ARP is handled according to the protocol specification with no exceptions.

IP packets carry the actual AHB commands. The EDCL expects an Ethernet frame containing IP, UDP and the EDCL specific application layer parts. Table 483 shows the IP packet required by the EDCL. The contents of the different protocol headers can be found in TCP/IP literature.

*Table 483.*The IP packet expected by the EDCL.

Ethernet Header	IP Header	UDP Header	2 B Offset	4 B Control word	4 B Address	Data 0 - 242 4B Words	Ethernet CRC

The following is required for successful communication with the EDCL: A correct destination MAC address as set by the generics, an Ethernet type field containing 0x0806 (ARP) or 0x0800 (IP). The IP-address is then compared with the value determined by the generics for a match. The IP-header checksum and identification fields are not checked. There are a few restrictions on the IP-header fields. The version must be four and the header size must be 5 B (no options). The protocol field must always be 0x11 indicating a UDP packet. The length and checksum are the only IP fields changed for the reply.

The EDCL only provides one service at the moment and it is therefore not required to check the UDP port number. The reply will have the original source port number in both the source and destination fields. UDP checksum are not used and the checksum field is set to zero in the replies.

The UDP data field contains the EDCL application protocol fields. Table 484 shows the application protocol fields (data field excluded) in packets received by the EDCL. The 16-bit offset is used to align the rest of the application layer data to word boundaries in memory and can thus be set to any value. The R/W field determines whether a read (0) or a write(1) should be performed. The length

*Table 484.*The EDCL application layer fields in received frames.

16-bit Offset	14-bit Sequence number	1-bit R/W	10-bit Length	7-bit Unused

field contains the number of bytes to be read or written. If R/W is one the data field shown in table 483 contains the data to be written. If R/W is zero the data field is empty in the received packets. Table 485 shows the application layer fields of the replies from the EDCL. The length field is always zero for replies to write requests. For read requests it contains the number of bytes of data contained in the data field.

*Table 485.*The EDCL application layer fields in transmitted frames.

16-bit Offset	14-bit sequence number	1-bit ACK/NAK	10-bit Length	7-bit Unused

The EDCL implements a Go-Back-N algorithm providing reliable transfers. The 14-bit sequence number in received packets are checked against an internal counter for a match. If they do not match, no operation is performed and the ACK/NAK field is set to 1 in the reply frame. The reply frame contains the internal counter value in the sequence number field. If the sequence number matches, the operation is performed, the internal counter value is stored in the sequence number field, the ACK/NAK field is set to 0 in the reply and the internal counter is incremented, . The length field is always set to 0 for ACK/NAK=1 frames. The unused field is not checked and is copied to the reply. It can thus be set to hold for example some extra identifier bits if needed.

42.6.3 EDCL IP and Ethernet address settings

The default value of the EDCL IP and MAC addresses are set by `ipaddrh, ipaddrl, macaddrh and macaddrl` generics. The IP address can later be changed by software, but the MAC address is fixed. To allow several EDCL enabled GRETH controllers on the same sub-net, the 4 LSB bits of the IP and MAC address can optionally be set by an input signal. This is enabled by setting the `edcl` generic = 2, and driving the 4-bit LSB value on ethi.edcladdr.

42.6.4 EDCL buffer size

The EDCL has a dedicated internal buffer memory which stores the received packets during processing. The size of this buffer is configurable with a VHDL generic to be able to obtain a suitable compromise between throughput and resource utilization in the hardware. Table 486 lists the different buffer configurations. For each size the table shows how many concurrent packets the EDCL can handle, the maximum size of each packet including headers and the maximum size of the data payload. Sending more packets before receiving a reply than specified for the selected buffer size will lead to dropped packets. The behavior is unspecified if sending larger packets than the maximum allowed.

*Table 486.*EDCL buffer sizes

Total buffer size (kB)	Number of packet buffers	Packet buffer size (B)	Maximum data payload (B)
1	4	256	200
2	4	512	456
4	8	512	456
8	8	1024	968
16	16	1024	968
32	32	1024	968
64	64	1024	968

42.7 Media Independent Interfaces

There are several interfaces defined between the MAC sublayer and the Physical layer. The GRETH supports two of them: The Media Independent Interface (MII) and the Reduced Media Independent Interface (RMII).

The MII was defined in the 802.3 standard and is most commonly supported. The ethernet interface have been implemented according to this specification. It uses 16 signals.

The RMII was developed to meet the need for an interface allowing Ethernet controllers with smaller pin counts. It uses 6 (7) signals which are a subset of the MII signals. Table 487 shows the mapping between the RMII signals and the GRLIB MII interface.

*Table 487.*Signal mappings between RMII and the GRLIB MII interface.

RMII	MII
txd[1:0]	txd[1:0]
tx_en	tx_en
crs_dv	rx_crs
rxd[1:0]	rxd[1:0]
ref_clk	rmii_clk
rx_er	not used

42.8 Software drivers

Drivers for the GRETH MAC is provided for the following operating systems: RTEMS, eCos, uClinux and Linux-2.6. The drivers are freely available in full source code under the GPL license from Aeroflex Gaisler’s web site (http://gaisler.com/).

42.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 488.*GRETH registers

APB address offset	**Register**
0x0	Control register
0x4	Status/Interrupt-source register
0x8	MAC Address MSB
0xC	MAC Address LSB
0x10	MDIO Control/Status
0x14	Transmit descriptor pointer
0x18	Receiver descriptor pointer
0x1C	EDCL IP
0x20	Hash table msb
0x24	Hash table lsb
0x28	EDCL MAC address MSB
0x2C	EDCL MAC address LSB
0x10000 - 0x107FC	Transmit RAM buffer debug access
0x20000 - 0x207FC	Receiver RAM buffer debug access
0x30000 - 0x3FFFC	EDCL buffer debug access

Table 489. GRETH control register

31	30 28	27	26	25	24 15	14	13	12	11	10	9 8	7	6	5	4	3	2	1	0
ED	BS		MA	MC	RESERVED	ED	RD	DD	ME	PI	RES	SP	RS	PM	FD	RI	TI	RE	TE

31	EDCL available (ED) - Set to one if the EDCL is available.
30: 28	EDCL buffer size (BS) - Shows the amount of memory used for EDCL buffers. 0 = 1 kB, 1 = 2 kB, ..., 6 = 64 kB.
27	RESERVED

Table 489. GRETH control register

26	MDIO interrupts available (MA) - Set to one when the core supports mdio interrupts. Read only.
25	Multicast available (MC) - Set to one when the core supports multicast address reception. Read only.
24: 15	RESERVED
14	EDCL Disable (ED) - Set to one to disable the EDCL and zero to enable it. Reset value taken from the ethi.edcldisable signal. Only available if the EDCL hardware is present in the core.
13	RAM debug enable (RD) - Set to one to enable the RAM debug mode. Reset value: ‘0’. Only available if the VHDL generic ramdebug is nonzero.
12	Disable duplex detection (DD) - Disable the EDCL speed/duplex detection FSM. If the FSM cannot complete the detection the MDIO interface will be locked in busy mode. If software needs to access the MDIO the FSM can be disabled here and as soon as the MDIO busy bit is 0 the interface is available. Note that the FSM cannot be reenabled again.
11	Multicast enable (ME) - Enable reception of multicast addresses. Reset value: ‘0’.
10	PHY status change interrupt enable (PI) - Enables interrupts for detected PHY status changes.
9: 8	RESERVED
7	Speed (SP) - Sets the current speed mode. 0 = 10 Mbit, 1 = 100 Mbit. Only used in RMII mode (rmii = 1). A default value is automatically read from the PHY after reset. Reset value: ‘1’.
6	Reset (RS) - A one written to this bit resets the GRETH core. Self clearing. No other accesses should be done .to the slave interface other than polling this bit until it is cleared.
5	Promiscuous mode (PM) - If set, the GRETH operates in promiscuous mode which means it will receive all packets regardless of the destination address. Reset value: ‘0’.
4	Full duplex (FD) - If set, the GRETH operates in full-duplex mode otherwise it operates in half-duplex. Reset value: ‘0’.
3	Receiver interrupt (RI) - Enable Receiver Interrupts. An interrupt will be generated each time a packet is received when this bit is set. The interrupt is generated regardless if the packet was received successfully or if it terminated with an error. Reset value: ‘0’.
2	Transmitter interrupt (TI) - Enable Transmitter Interrupts. An interrupt will be generated each time a packet is transmitted when this bit is set. The interrupt is generated regardless if the packet was transmitted successfully or if it terminated with an error. Reset value: ‘0’.
1	Receive enable (RE) - Should be written with a one each time new descriptors are enabled. As long as this bit is one the GRETH will read new descriptors and as soon as it encounters a disabled descriptor it will stop until RE is set again. This bit should be written with a one after the new descriptors have been enabled. Reset value: ‘0’.
0	Transmit enable (TE) - Should be written with a one each time new descriptors are enabled. As long as this bit is one the GRETH will read new descriptors and as soon as it encounters a disabled descriptor it will stop until TE is set again. This bit should be written with a one after the new descriptors have been enabled. Reset value: ‘0’.

Table 490. GRETH status register

31 9	8	7	6	5	4	3	2	1	0
RESERVED	PS	IA	TS	TA	RA	TI	RI	TE	RE

8	PHY status changes (PS) - Set each time a PHY status change is detected.
7	Invalid address (IA) - A packet with an address not accepted by the MAC was received. Cleared when written with a one. Reset value: ‘0’.
6	Too small (TS) - A packet smaller than the minimum size was received. Cleared when written with a one. Reset value: ‘0’.
5	Transmitter AHB error (TA) - An AHB error was encountered in transmitter DMA engine. Cleared when written with a one. Not Reset.

Table 490. GRETH status register

4	Receiver AHB error (RA) - An AHB error was encountered in receiver DMA engine. Cleared when written with a one. Not Reset.
3	Transmitter interrupt (TI) - A packet was transmitted without errors. Cleared when written with a one. Not Reset.
2	Receiver interrupt (RI) - A packet was received without errors. Cleared when written with a one. Not Reset.
1	Transmitter error (TE) - A packet was transmitted which terminated with an error. Cleared when written with a one. Not Reset.
0	Receiver error (RE) - A packet has been received which terminated with an error. Cleared when written with a one. Not Reset.

Table 491. GRETH MAC address MSB.

31 16	15 0
RESERVED	Bit 47 downto 32 of the MAC address

31: 16	RESERVED
15: 0	The two most significant bytes of the MAC Address. Not Reset.

Table 492. GRETH MAC address LSB.

31 0
Bit 31 downto 0 of the MAC address

31: 0	The four least significant bytes of the MAC Address. Not Reset.

Table 493. GRETH MDIO ctrl/status register.

31 16	15 11	10 6	5	4	3	2	1	0
DATA	PHYADDR	REGADDR		NV	BU	LF	RD	WR

31: 16	Data (DATA) - Contains data read during a read operation and data that is transmitted is taken from this field. Reset value: 0x0000.
15: 11	PHY address (PHYADDR) - This field contains the address of the PHY that should be accessed during a write or read operation. Reset value: “00000”.
10: 6	Register address (REGADDR) - This field contains the address of the register that should be accessed during a write or read operation. Reset value: “00000”.
5	RESERVED
4	Not valid (NV) - When an operation is finished (BUSY = 0) this bit indicates whether valid data has been received that is, the data field contains correct data. Reset value: ‘0’.
3	Busy (BU) - When an operation is performed this bit is set to one. As soon as the operation is finished and the management link is idle this bit is cleared. Reset value: ‘0’.
2	Linkfail (LF) - When an operation completes (BUSY = 0) this bit is set if a functional management link was not detected. Reset value: ‘1’.
1	Read (RD) - Start a read operation on the management interface. Data is stored in the data field. Reset value: ‘0’.
0	Write (WR) - Start a write operation on the management interface. Data is taken from the Data field. Reset value: ‘0’.

Table 494. GRETH transmitter descriptor table base address register.

31 10	9 3	2 0
BASEADDR	DESCPNT	RES

Table 494. GRETH transmitter descriptor table base address register.

31: 10	Transmitter descriptor table base address (BASEADDR) - Base address to the transmitter descriptor table.Not Reset.
9: 3	Descriptor pointer (DESCPNT) - Pointer to individual descriptors. Automatically incremented by the Ethernet MAC.
2: 0	RESERVED

Table 495. GRETH receiver descriptor table base address register.

31 10	9 3	2 0
BASEADDR	DESCPNT	RES

31: 10	Receiver descriptor table base address (BASEADDR) - Base address to the receiver descriptor table.Not Reset.
9: 3	Descriptor pointer (DESCPNT) - Pointer to individual descriptors. Automatically incremented by the Ethernet MAC.
2: 0	RESERVED

Table 496. GRETH EDCL IP register

31 0
EDCL IP ADDRESS

31: 0	EDCL IP address. Reset value is set with the ipaddrh and ipaddrl generics.

Table 497. GRETH Hash table msb register

31 0
Hash table (64:32)

31: 0	Hash table msb. Bits 64 downto 32 of the hash table.

Table 498. GRETH Hash table lsb register

31 0
Hash table (64:32)

31: 0	Hash table lsb. Bits 31downto 0 of the hash table.

Table 499. GRETH EDCL MAC address MSB.

31 16	15 0
RESERVED	Bit 47 downto 32 of the EDCL MAC Address

31: 16	RESERVED
15: 0	The two most significant bytes of the EDCL MAC Address. Hardcoded reset value set with the VHDL generic macaddrh.

Table 500. GRETH EDCL MAC address LSB.

31 ... 0
Bit 31 downto 0 of the EDCL MAC Address

31: 0	The 4 least significant bytes of the EDCL MAC Address. Hardcoded reset value set with the VHDL generics macaddrh and macaddrl. If the VHDL generic edcl is set to 2 bits 3 downto 0 are set with the edcladdr input signal.

42.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x1D. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

42.11 Configuration options

Table 501 shows the configuration options of the core (VHDL generics).

*Table 501.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRETH.	0 - NAHBIRQ-1	0
memtech	Memory technology used for the FIFOs.	0 - NTECH	inferred
ifg_gap	Number of ethernet clock cycles used for one interframe gap. Default value as required by the standard. Do not change unless you know what you are doing.	1 - 255	24
attempt_limit	Maximum number of transmission attempts for one packet. Default value as required by the standard.	1 - 255	16
backoff_limit	Limit on the backoff size of the backoff time. Default value as required by the standard. Sets the number of bits used for the random value. Do not change unless you know what your doing.	1 - 10	10
slot_time	Number of ethernet clock cycles used for one slot- time. Default value as required by the ethernet standard. Do not change unless you know what you are doing.	1 - 255	128
mdcscaler	Sets the divisor value use to generate the mdio clock (mdc). The mdc frequency will be clk/(2*(mdcs-caler+1)).	0 - 255	25
enable_mdio	Enable the Management interface,	0 - 1	0
fifosize	Sets the size in 32-bit words of the receiver and transmit-ter FIFOs.	4 - 32	8
nsync	Number of synchronization registers used.	1 - 2	2
edcl	Enable EDCL. 0 = disabled. 1 = enabled. 2 = enabled and 4-bit LSB of IP and ethernet MAC address pro-grammed by ethi.edcladdr, 3=in addition to features for value 2 the reset value for the EDCL disable bit is taken from the ethi.edcldisable signal instead of being hard-coded to 0.	0 - 3	0
edclbufsz	Select the size of the EDCL buffer in kB.	1 - 64	1

*Table 501.*Configuration options

Generic	Function	Allowed range	Default
macaddrh	Sets the upper 24 bits of the EDCL MAC address. Not all addresses are allowed and most NICs and protocol implementations will discard frames with illegal addresses silently. Consult network literature if unsure about the addresses.	0 - 16#FFFFFF#	16#00005E#
macaddrl	Sets the lower 24 bits of the EDCL MAC address. Not all addresses are allowed and most NICs and protocol implementations will discard frames with illegal addresses silently. Consult network literature if unsure about the addresses.	0 - 16#FFFFFF#	16#000000#
ipaddrh	Sets the upper 16 bits of the EDCL IP address reset value.	0 - 16#FFFF#	16#C0A8#
ipaddrl	Sets the lower 16 bits of the EDCL IP address reset value.	0 - 16#FFFF#	16#0035#
phyrstadr	Sets the reset value of the PHY address field in the MDIO register.	0 - 31	0
rmii	Selects the desired PHY interface. 0 = MII, 1 = RMII.	0 - 1	0
oepol	Selects polarity on output enable (ETHO.MDIO_OE). 0 = active low, 1 = active high	0 - 1	0
mdint_pol	Selects polarity for level sensitive PHY interrupt line. 0 = active low, 1 = active high	0 - 1	0
enable_mdint	Enable mdio interrupts	0 - 1	0
multicast	Enable multicast support	0 - 1	0
ramdebug	Enables debug access to the core's RAM blocks through the APB interface. 1=enables access to the receiver and transmitter RAM buffers, 2=enables access to the EDCL buffers in addition to the functionality of value 1. Setting this generic to 2 will have no effect if the edcl generic is 0.	0 - 2	0
ehindex	AHB master index for the separate EDCL master interface. Only used if edclsepahb is 1.	0 - NAHBMST-1	0
edclsepahb	Enables separate EDCL AHB master interface. A signal determines if the separate interface or the common interface is used. Only available in the GRETH_GBIT_MB version of the core.	0 - 1	0
mdiohold	Set output hold time for MDIO in number of AHB cycles. Should be 10 ns or more.	1 - 30	1
maxsize	Set maximum length of the data field of Ethernet 802.3 frame. Values of 'maxsize' and below for this field indicate that the ethernet type field is used as the size of the payload of the Ethernet Frame while values of above 'maxsize' indicate that the field is used to represent EtherType. For 802.3q support set the length of the payload to 1504	64 - 2047	1500

42.12 Signal descriptions

Table 502 shows the interface signals of the core (VHDL ports).

*Table 502.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low

*Table 502.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
ETHI	gtx_clk	Input	Ethernet gigabit transmit clock.	-
	rmii_clk	Input	Ethernet RMII clock.	-
	tx_clk	Input	Ethernet transmit clock.	-
	rx_clk	Input	Ethernet receive clock.	-
	rxd	Input	Ethernet receive data.	-
	rx_dv	Input	Ethernet receive data valid.	High
	rx_er	Input	Ethernet receive error.	High
	rx_col	Input	Ethernet collision detected. (Asynchronous, sampled with tx_clk)	High
	rx_crs	Input	Ethernet carrier sense. (Asynchronous, sampled with tx_clk)	High
	mdio_i	Input	Ethernet management data input	-
	mdint	Input	Ethernet management interrupt	-
	phyrstaddr	Input	Reset address for GRETH PHY address field.	-
	edcladdr	Input	Sets the four least significant bits of the EDCL MAC address and the EDCL IP address when the edcl generic is set to 2.	-
	edclsepahb	Input	Selects AHB master interface for the EDCL. ‘0’ selects the common interface and ‘1’ selects the separate interface. Only available in the GRETH_GBIT_MB version of the core when the VHDL generic edclsepahb is set to 1.	-
	edcldisable	Input	Reset value for edcl disable register bit. Setting the signal to 1 disables the EDCL at reset and 0 enables it.	-
ETHO	reset	Output	Ethernet reset (asserted when the MAC is reset).	Low
	txd	Output	Ethernet transmit data.	-
	tx_en	Output	Ethernet transmit enable.	High
	tx_er	Output	Ethernet transmit error.	High
	mdc	Output	Ethernet management data clock.	-
	mdio_o	Output	Ethernet management data output.	-
	mdio_oe	Output	Ethernet management data output enable.	Set by the oepol generic.

* see GRLIB IP Library User’s Manual

42.13 Library dependencies

Table 503 shows libraries used when instantiating the core (VHDL libraries).

*Table 503.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	NET	Signals, components	GRETH component declaration

42.14 Instantiation

The first example shows how the non-mb version of the core can be instantiated and the second one show the mb version.

42.14.1 Non-MB version

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.ethernet_mac.all;

entity greth_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- ethernet signals
    ethi :: in  eth_in_type;
    etho :  in  eth_out_type
    );
end;

architecture rtl of greth_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

begin

  -- AMBA Components are instantiated here
  ...

  -- GRETH
  e1 : greth
    generic map(
     hindex       => 0,
     pindex       => 12,
     paddr        => 12,
     pirq         => 12,
     memtech      => inferred,
     mdcscaler    => 50,
     enable_mdio  => 1,
     fifosize     => 32,
     nsync        => 1,
     edcl         => 1,
     edclbufsz    => 8,
     macaddrh     => 16#00005E#,
     macaddrl     => 16#00005D#,
     ipaddrh      => 16#c0a8#,
     ipaddrl      => 16#0035#)
 port map(
   rst          => rstn,
   clk          => clk,
   ahbmi        => ahbmi,
   ahbmo        => ahbmo(0),
   apbi         => apbi,
   apbo         => apbo(12),
   ethi         => ethi,
   etho         => etho
   );
end;
```

42.14.2 MB version

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.ethernet_mac.all;

entity greth_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- ethernet signals
    ethi :: in  eth_in_type;
    etho :  in  eth_out_type
    );
end;

architecture rtl of greth_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

begin

  -- AMBA Components are instantiated here
  ...

  -- GRETH
  e1 : greth_mb
    generic map(
     hindex       => 0,
     pindex       => 12,
     paddr        => 12,
     pirq         => 12,
     memtech      => inferred,
     mdcscaler    => 50,
     enable_mdio  => 1,
     fifosize     => 32,
     nsync        => 1,
     edcl         => 1,
     edclbufsz    => 8,
     macaddrh     => 16#00005E#,
     macaddrl     => 16#00005D#,
     ipaddrh      => 16#c0a8#,
     ipaddrl      => 16#0035#,
     ehindex      => 1,
     edclsepahb   => 1)
```

```
 port map(
   rst          => rstn,
   clk          => clk,
   ahbmi        => ahbmi,
   ahbmo        => ahbmo(0),
   ahbmi2       => ahbmi,
   ahbmo2       => ahbmo(1),
   apbi         => apbi,
   apbo         => apbo(12),
   ethi         => ethi,
   etho         => etho
   );
end;
```

43 GRETH_GBIT - Gigabit Ethernet Media Access Controller (MAC) w. EDCL

43.1 Overview

Aeroflex Gaisler’s Gigabit Ethernet Media Access Controller (GRETH_GBIT) provides an interface between an AMBA-AHB bus and an Ethernet network. It supports 10/100/1000 Mbit speed in both full- and half-duplex. The AMBA interface consists of an APB interface for configuration and control and an AHB master interface which handles the dataflow. The dataflow is handled through DMA channels. There is one DMA engine for the transmitter and one for the receiver. Both share the same AHB master interface.

The ethernet interface supports the MII and GMII interfaces which should be connected to an external PHY. The GRETH also provides access to the MII Management interface which is used to configure the PHY. Optional hardware support for the Ethernet Debug Communication Link (EDCL) protocol is also provided. This is an UDP/IP based protocol used for remote debugging.

Some of the supported features for the DMA channels are Scatter Gather I/O and TCP/UDP over IPv4 checksum offloading for both receiver and transmitter. Software Drivers are provided for RTEMS, eCos, uClinux and Linux 2.6.

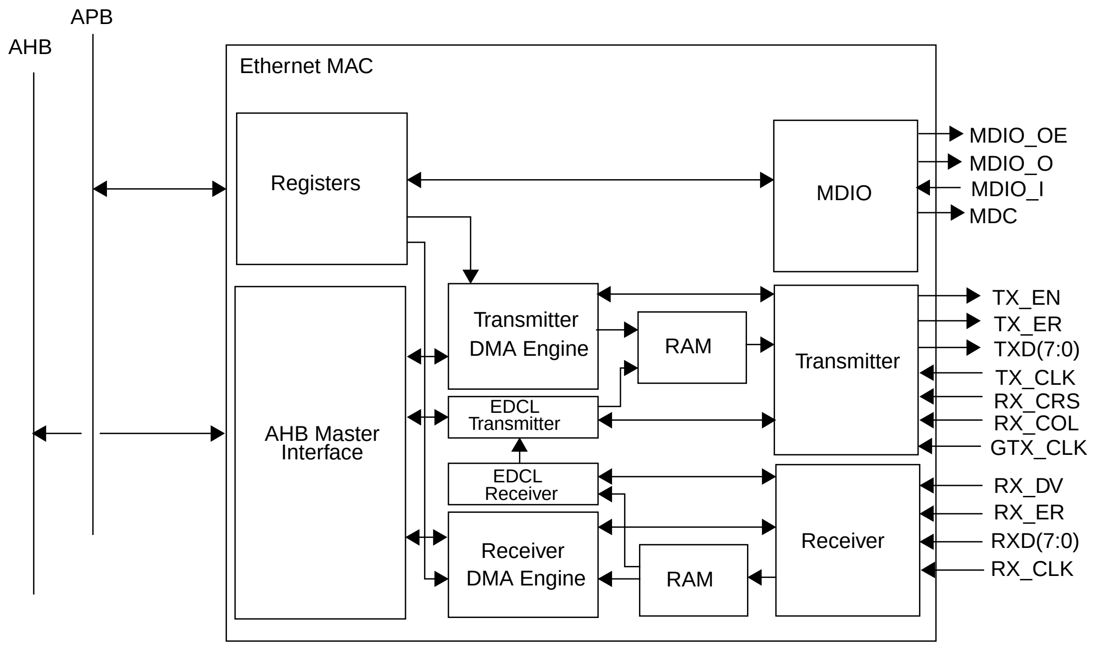


Figure 144. Block diagram of the internal structure of the GRETH_GBIT.

43.2 Operation

43.2.1 System overview

The GRETH_GBIT consists of 3 functional units: The DMA channels, MDIO interface and the optional Ethernet Debug Communication Link (EDCL).

The main functionality consists of the DMA channels which are used for transferring data between an AHB bus and an Ethernet network. There is one transmitter DMA channel and one Receiver DMA channel. The operation of the DMA channels is controlled through registers accessible through the APB interface.

The MDIO interface is used for accessing configuration and status registers in one or more PHYs connected to the MAC. The operation of this interface is also controlled through the APB interface.

The optional EDCL provides read and write access to an AHB bus through Ethernet. It uses the UDP, IP and ARP protocols together with a custom application layer protocol to accomplish this. The EDCL contains no user accessible registers and always runs in parallel with the DMA channels.

The Media Independent Interface (MII) and Gigabit Media Independent Interface (GMII) are used for communicating with the PHY. More information can be found in section 43.7.

The EDCL and the DMA channels share the Ethernet receiver and transmitter. More information on these functional units is provided in sections 43.3 - 43.6.

43.2.2 Protocol support

The GRETH_GBIT is implemented according to IEEE standard 802.3-2002. There is no support for the optional control sublayer. This means that packets with type 0x8808 (the only currently defined ctrl packets) are discarded.

43.2.3 Hardware requirements

The GRETH_GBIT is synthesisable with most Synthesis tools. There are three or four clock domains depending on if the gigabit mode is used. The three domains always present are the AHB clock, Ethernet Receiver clock (RX_CLK) and the 10/100 Ethernet transmitter clock (TX_CLK). If the gigabit mode is also used the fourth clock domain is the gigabit transmitter clock (GTX_CLK). Both full-duplex and half-duplex operating modes are supported and both can be run in either 10/100 or 1000 Mbit. The system frequency requirement (AHB clock) for 10 Mbit operation is 2.5 MHz, 18 MHz for 100 Mbit and 40 MHz for 1000 Mbit mode. The 18 MHz limit was tested on a Xilinx board with a DCM that did not support lower frequencies so it might be possible to run it on lower frequencies. It might also be possible to run the 10 Mbit mode on lower frequencies.

RX_CLK and TX_CLK are sourced by the PHY while GTX_CLK is sourced by the MAC according to the 802.3-2002 standard. The GRETH_GBIT does not contain an internal clock generator so GTX_CLK should either be generated in the FPGA (with a PLL/DLL) or with an external oscillator.

43.2.4 RAM debug support

Support for debug accesses the core’s internal RAM blocks can be optionally enabled using the ramdebug VHDL generic. Setting it to 1 enables accesses to the transmitter and receiver RAM buffers and setting it to 2 enables accesses to the EDCL buffer in addition to the previous two buffers.

The transmitter RAM buffer is accessed starting from APB address offset 0x10000 which corresponds to location 0 in the RAM. There are 512 32-bit wide locations in the RAM which results in the last address being 0x107FC corresponding to RAM location 511 (byte addressing used on the APB bus).

Correspondingly the receiver RAM buffer is accessed starting from APB address offset 0x20000. The addresses, width and depth is the same.

The EDCL buffers are accessed starting from address 0x30000. The number of locations depend on the configuration and can be from 256 to 16384. Each location is 32-bits wide so the maximum address is 0x3FC and 0xFFFC correspondingly.

Before any debug accesses can be made the ramdebugen bit in the control register has to be set. During this time the debug interface controls the RAM blocks and normal operations is stopped. EDCL packets are not received. The MAC transmitter and receiver could still operate if enabled but the RAM buffers would be corrupt if debug accces are made simultaneously. Thus they MUST be disabled before the RAM debug mode is enabled.

43.2.5 Multibus version

There is a version of the core which has an additional master interface that can be used for the EDCL. Otherwise this version is identical to the basic version. The additional master interface is enabled with the edclsepahb VHDL generic. Then the ethi.edclsepahb signal control whether EDCL accesses are done on the standard master interface or the additional interface. Setting the signal to ‘0’ makes the EDCL use the standard master interface while ‘1’ selects the additional master. This signal is only sampled at reset and changes to this signal have no effect until the next reset.

43.3 Tx DMA interface

The transmitter DMA interface is used for transmitting data on an Ethernet network. The transmission is done using descriptors located in memory.

43.3.1 Setting up a descriptor.

A single descriptor is shown in table 504 and 505. The number of bytes to be sent should be set in the length field and the address field should point to the data. There are no alignment restrictions on the address field. If the interrupt enable (IE) bit is set, an interrupt will be generated when the packet has been sent (this requires that the transmitter interrupt bit in the control register is also set). The interrupt will be generated regardless of whether the packet was transmitted successfully or not.

Table 504. GRETH_GBIT transmit descriptor word 0 (address offset 0x0)

31 21	20	19	18	17	16	15	14	13	12	11	10 0
RESERVED	UC	TC	IC	MO	LC	AL	UE	IE	WR	EN	LENGTH

31: 21	RESERVED
20	UDP checksum (UC) - Calculate and insert the UDP checksum for this packet. The checksum is only inserted if an UDP packet is detected.
19	TCP checksum (TC) - Calculate and insert the TCP checksum for this packet. The checksum is only inserted if an TCP packet is detected.
18	IP checksum (IC) - Calculate and insert the IP header checksum for this packet. The checksum is only inserted if an IP packet is detected.
17	More (MO) - More descriptors should be fetched for this packet (Scatter Gather I/O).
16	Late collision (LC) - A late collision occurred during the transmission (1000 Mbit mode only).
15	Attempt limit error (AL) - The packet was not transmitted because the maximum number of attempts was reached.
14	Underrun error (UE) - The packet was incorrectly transmitted due to a FIFO underrun error.
13	Interrupt enable (IE) - Enable Interrupts. An interrupt will be generated when the packet from this descriptor has been sent provided that the transmitter interrupt enable bit in the control register is set. The interrupt is generated regardless if the packet was transmitted successfully or if it terminated with an error.
12	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
11	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
10: 0	LENGTH - The number of bytes to be transmitted.

Table 505. GRETH_GBIT transmit descriptor word 1 (address offset 0x4)

31 0
ADDRESS

31: 0	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.

To enable a descriptor the enable (EN) bit should be set and after this is done, the descriptor should not be touched until the enable bit has been cleared by the GRETH_GBIT. The rest of the fields in the descriptor are explained later in this section.

43.3.2 Starting transmissions

Enabling a descriptor is not enough to start a transmission. A pointer to the memory area holding the descriptors must first be set in the GRETH_GBIT. This is done in the transmitter descriptor pointer register. The address must be aligned to a 1 kB boundary. Bits 31 to 10 hold the base address of descriptor area while bits 9 to 3 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the GRETH_GBIT the pointer field is incremented by 8 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 1 kB boundary has been reached (the descriptor at address offset 0x3F8 has been used). The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 1 kB boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when a transmission is active.

The final step to activate the transmission is to set the transmit enable bit in the control register. This tells the GRETH_GBIT that there are more active descriptors in the descriptor table. This bit should always be set when new descriptors are enabled, even if transmissions are already active. The descriptors must always be enabled before the transmit enable bit is set.

43.3.3 Descriptor handling after transmission

When a transmission of a packet has finished, status is written to the first word in the corresponding descriptor. The Underrun Error bit is set if the transmitter RAM was not able to provide data at a sufficient rate. This indicates a synchronization problem most probably caused by a low clock rate on the AHB clock. The whole packet is buffered in the transmitter RAM before transmission so underruns cannot be caused by bus congestion. The Attempt Limit Error bit is set if more collisions occurred than allowed. When running in 1000 Mbit mode the Late Collision bit indicates that a collision occurred after the slottime boundary was passed.

The packet was successfully transmitted only if these three bits are zero. The other bits in the first descriptor word are set to zero after transmission while the second word is left untouched.

The enable bit should be used as the indicator when a descriptor can be used again, which is when it has been cleared by the GRETH_GBIT. There are three bits in the GRETH_GBIT status register that hold transmission status. The Transmit Error (TE) bit is set each time an transmission ended with an error (when at least one of the three status bits in the transmit descriptor has been set). The Transmit Successful (TI) is set each time a transmission ended successfully.

The Transmit AHB Error (TA) bit is set when an AHB error was encountered either when reading a descriptor, reading packet data or writing status to the descriptor. Any active transmissions are aborted and the transmitter is disabled. The transmitter can be activated again by setting the transmit enable register.

43.3.4 Setting up the data for transmission

The data to be transmitted should be placed beginning at the address pointed by the descriptor address field. The GRETH_GBIT does not add the Ethernet address and type fields so they must also be stored in the data buffer. The 4 B Ethernet CRC is automatically appended at the end of each packet. Each descriptor will be sent as a single Ethernet packet. If the size field in a descriptor is greater than 1514 B, the packet will not be sent.

43.3.5 Scatter Gather I/O

A packet can be generated from data fetched from several descriptors. This is called Scatter Gather I/O. The More (MO) bit should be set to 1 to indicate that more descriptors should be used to generate the current packet. When data from the current descriptor has been read to the RAM the next descriptor is fetched and the new data is appended to the previous data. This continues until a descriptor with the MO bit set to 0 is encountered. The packet will then be transmitted.

Status is written immediately when data has been read to RAM for descriptors with MO set to 1. The status bits are always set to 0 since no transmission has occurred. The status bits will be written to the last descriptor for the packet (which had MO set to 0) when the transmission has finished.

No interrupts are generated for descriptors with MO set to 1 so the IE bit is don’t care in this case.

The checksum offload control bits (explained in section 43.3.6) must be set to the same values for all descriptors used for a single packet.

43.3.6 Checksum offloading

Support is provided for checksum calculations in hardware for TCP and UDP over IPv4. The checksum calculations are enabled in each descriptor and applies only to that packet (when the MO bit is set all descriptors used for a single packet must have the checksum control bits set in the same way).

The IP Checksum bit (IC) enables IP header checksum calculations. If an IPv4 packet is detected when transmitting the packet associated with the descriptor the header checksum is calculated and inserted. If TCP Checksum (TC) is set the TCP checksum is calculated and inserted if an TCP/IPv4 packet is detected. Finally, if the UDP Checksum bit is set the UDP checksum is calculated and inserted if a UDP/IPv4 packet is detected. In the case of fragmented IP packets, checksums for TCP and UDP are only inserted for the first fragment (which contains the TCP or UDP header).

43.4 Rx DMA interface

The receiver DMA interface is used for receiving data from an Ethernet network. The reception is done using descriptors located in memory.

43.4.1 Setting up descriptors

A single descriptor is shown in table 506 and 507. The address field points at the location where the received data should be stored. There are no restrictions on alignment. The GRETH_GBIT will never store more than 1518 B to the buffer (the tagged maximum frame size excluding CRC). The CRC field (4 B) is never stored to memory so it is not included in this number. If the interrupt enable (IE) bit is set, an interrupt will be generated when a packet has been received to this buffer (this requires that the receiver interrupt bit in the control register is also set). The interrupt will be generated regardless of whether the packet was received successfully or not.

The enable bit is set to indicate that the descriptor is valid which means it can be used by the to store a packet. After it is set the descriptor should not be touched until the EN bit has been cleared by the GRETH_GBIT.

The rest of the fields in the descriptor are explained later in this section..

Table 506. GRETH_GBIT receive descriptor word 0 (address offset 0x0)

31 27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10 0
RESERVED	MC	IF	TR	TD	UR	UD	IR	ID	LE	OE	CE	FT	AE	IE	WR	EN	LENGTH

31: 27	RESERVED
26	Multicast address (MC) - The destination address of the packet was a multicast address (not broadcast).
25	IP fragment (IF) - Fragmented IP packet detected.

Table 506. GRETH_GBIT receive descriptor word 0 (address offset 0x0)

24	TCP error (TR) - TCP checksum error detected.
23	TCP detected (TD) - TCP packet detected.
22	UDP error (UR) - UDP checksum error detected.
21	UDP detected (UD) - UDP packet detected.
20	IP error (IR) - IP checksum error detected.
19	IP detected (ID) - IP packet detected.
18	Length error (LE) - The length/type field of the packet did not match the actual number of received bytes.
17	Overrun error (OE) - The frame was incorrectly received due to a FIFO overrun.
16	CRC error (CE) - A CRC error was detected in this frame.
15	Frame too long (FT) - A frame larger than the maximum size was received. The excessive part was truncated.
14	Alignment error (AE) - An odd number of nibbles were received.
13	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when a packet has been received to this descriptor provided that the receiver interrupt enable bit in the control register is set. The interrupt is generated regardless if the packet was received successfully or if it terminated with an error.
12	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
11	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
10: 0	LENGTH - The number of bytes received to this descriptor.

Table 507. GRETH_GBIT receive descriptor word 1 (address offset 0x4)

31 0
ADDRESS

31: 0	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.

43.4.2 Starting reception

Enabling a descriptor is not enough to start reception. A pointer to the memory area holding the descriptors must first be set in the GRETH_GBIT. This is done in the receiver descriptor pointer register. The address must be aligned to a 1 kB boundary. Bits 31 to 10 hold the base address of descriptor area while bits 9 to 3 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the GRETH_GBIT the pointer field is incremented by 8 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 1 kB boundary has been reached (the descriptor at address offset 0x3F8 has been used). The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 1 kB boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when reception is active.

The final step to activate reception is to set the receiver enable bit in the control register. This will make the GRETH_GBIT read the first descriptor and wait for an incoming packet.

43.4.3 Descriptor handling after reception

The GRETH indicates a completed reception by clearing the descriptor enable bit. The other control bits (WR, IE) are also cleared. The number of received bytes is shown in the length field. The parts of the Ethernet frame stored are the destination address, source address, type and data fields. Bits 24-14 in the first descriptor word are status bits indicating different receive errors. Bits 18 - 14 are zero after a reception without link layer errors. The status bits are described in table 506 (except the checksum offload bits which are also described in section 43.4.6).

Packets arriving that are smaller than the minimum Ethernet size of 64 B are not considered as a reception and are discarded. The current receive descriptor will be left untouched an used for the first packet arriving with an accepted size. The TS bit in the status register is set each time this event occurs.

If a packet is received with an address not accepted by the MAC, the IA status register bit will be set.

Packets larger than maximum size cause the FT bit in the receive descriptor to be set. The length field is not guaranteed to hold the correct value of received bytes. The counting stops after the word containing the last byte up to the maximum size limit has been written to memory.

The address word of the descriptor is never touched by the GRETH.

43.4.4 Reception with AHB errors

If an AHB error occurs during a descriptor read or data store, the Receiver AHB Error (RA) bit in the status register will be set and the receiver is disabled. The current reception is aborted. The receiver can be enabled again by setting the Receive Enable bit in the control register.

43.4.5 Accepted MAC addresses

In the default configuration the core receives packets with either the unicast address set in the MAC address register or the broadcast address. Multicast support can also be enabled and in that case a hash function is used to filter received multicast packets. A 64-bit register, which is accessible through the APB interface, determines which addresses should be received. Each address is mapped to one of the 64 bits using the hash function and if the bit is set to one the packet will be received. The address is mapped to the table by taking the 6 least significant bits of the 32-bit Ethernet crc calculated over the destination address of the MAC frame. A bit in the receive descriptor is set if a packet with a multicast address has been received to it.

43.4.6 Checksum offload

Support is provided for checksum calculations in hardware for TCP/UDP over IPv4. The checksum logic is always active and detects IPv4 packets with TCP or UDP payloads. If IPv4 is detected the ID bit is set, UD is set if an UDP payload is detected in the IP packet and TD is set if a TCP payload is detected in the IP packet (TD and UD are never set if an IPv4 packet is not detected). When one or more of these packet types is detected its corresponding checksum is calculated and if an error is detected the checksum error bit for that packet type is set. The error bits are never set if the corresponding packet type is not detected. The core does not support checksum calculations for TCP and UDP when the IP packet has been fragmented. This condition is indicated by the IF bit in the receiver descriptor and when set neither the TCP nor the UDP checksum error indications are valid.

43.5 MDIO Interface

The MDIO interface provides access to PHY configuration and status registers through a two-wire interface which is included in the MII interface. The GRETH_GBIT provides full support for the MDIO interface.

The MDIO interface can be used to access from 1 to 32 PHY containing 1 to 32 16-bit registers. A read transfer i set up by writing the PHY and register addresses to the MDIO Control register and setting the read bit. This caused the Busy bit to be set and the operation is finished when the Busy bit is cleared. If the operation was successful the Linkfail bit is zero and the data field contains the read data. An unsuccessful operation is indicated by the Linkfail bit being set. The data field is undefined in this case.

A write operation is started by writing the 16-bit data, PHY address and register address to the MDIO Control register and setting the write bit. The operation is finished when the busy bit is cleared and it was successful if the Linkfail bit is zero.

43.5.1 PHY interrupts

The core also supports status change interrupts from the PHY. A level sensitive interrupt signal can be connected on the mdint input. The mdint_pol vhdl generic can be used to select the polarity. The PHY status change bit in the status register is set each time an event is detected in this signal. If the PHY status interrupt enable bit is set at the time of the event the core will also generate an interrupt on the AHB bus.

43.6 Ethernet Debug Communication Link (EDCL)

The EDCL provides access to an on-chip AHB bus through Ethernet. It uses the UDP, IP and ARP protocols together with a custom application layer protocol. The application layer protocol uses an ARQ algorithm to provide reliable AHB instruction transfers. Through this link, a read or write transfer can be generated to any address on the AHB bus. The EDCL is optional and must be enabled with a generic.

43.6.1 Operation

The EDCL receives packets in parallel with the MAC receive DMA channel. It uses a separate MAC address which is used for distinguishing EDCL packets from packets destined to the MAC DMA channel. The EDCL also has an IP address which is set through generics. Since ARP packets use the Ethernet broadcast address, the IP-address must be used in this case to distinguish between EDCL ARP packets and those that should go to the DMA-channel. Packets that are determined to be EDCL packets are not processed by the receive DMA channel.

When the packets are checked to be correct, the AHB operation is performed. The operation is performed with the same AHB master interface that the DMA-engines use. The replies are automatically sent by the EDCL transmitter when the operation is finished. It shares the Ethernet transmitter with the transmitter DMA-engine but has higher priority.

43.6.2 EDCL protocols

The EDCL accepts Ethernet frames containing IP or ARP data. ARP is handled according to the protocol specification with no exceptions.

IP packets carry the actual AHB commands. The EDCL expects an Ethernet frame containing IP, UDP and the EDCL specific application layer parts. Table 508 shows the IP packet required by the EDCL. The contents of the different protocol headers can be found in TCP/IP literature.

*Table 508.*The IP packet expected by the EDCL.

Ethernet Header	IP Header	UDP Header	2 B Offset	4 B Control word	4 B Address	Data 0 - 242 4B Words	Ethernet CRC

The following is required for successful communication with the EDCL: A correct destination MAC address as set by the generics, an Ethernet type field containing 0x0806 (ARP) or 0x0800 (IP). The IP-address is then compared with the value determined by the generics for a match. The IP-header checksum and identification fields are not checked. There are a few restrictions on the IP-header fields. The version must be four and the header size must be 5 B (no options). The protocol field must always be 0x11 indicating a UDP packet. The length and checksum are the only IP fields changed for the reply.

The EDCL only provides one service at the moment and it is therefore not required to check the UDP port number. The reply will have the original source port number in both the source and destination fields. UDP checksum are not used and the checksum field is set to zero in the replies.

The UDP data field contains the EDCL application protocol fields. Table 509 shows the application protocol fields (data field excluded) in packets received by the EDCL. The 16-bit offset is used to align the rest of the application layer data to word boundaries in memory and can thus be set to any value. The R/W field determines whether a read (0) or a write(1) should be performed. The length

*Table 509.*The EDCL application layer fields in received frames.

16-bit Offset	14-bit Sequence number	1-bit R/W	10-bit Length	7-bit Unused

field contains the number of bytes to be read or written. If R/W is one the data field shown in Table 508 contains the data to be written. If R/W is zero the data field is empty in the received packets. Table 510 shows the application layer fields of the replies from the EDCL. The length field is always zero for replies to write requests. For read requests it contains the number of bytes of data contained in the data field.

*Table 510.*The EDCL application layer fields in transmitted frames.

16-bit Offset	14-bit sequence number	1-bit ACK/NAK	10-bit Length	7-bit Unused

The EDCL implements a Go-Back-N algorithm providing reliable transfers. The 14-bit sequence number in received packets are checked against an internal counter for a match. If they do not match, no operation is performed and the ACK/NAK field is set to 1 in the reply frame. The reply frame contains the internal counter value in the sequence number field. If the sequence number matches, the operation is performed, the internal counter is incremented, the internal counter value is stored in the sequence number field and the ACK/NAK field is set to 0 in the reply. The length field is always set to 0 for ACK/NAK=1 frames. The unused field is not checked and is copied to the reply. It can thus be set to hold for example some extra id bits if needed.

43.6.3 EDCL IP and Ethernet address settings

The default value of the EDCL IP and MAC addresses are set by `ipaddrh, ipaddrl, macaddrh and macaddrl` generics. The IP address can later be changed by software, but the MAC address is fixed. To allow several EDCL enabled GRETH controllers on the same sub-net, the 4 LSB bits of the IP and MAC address can optionally be set by an input signal. This is enabled by setting the `edcl` generic = 2, and driving the 4-bit LSB value on ethi.edcladdr.

43.6.4 EDCL buffer size

The EDCL has a dedicated internal buffer memory which stores the received packets during processing. The size of this buffer is configurable with a VHDL generic to be able to obtain a suitable compromise between throughput and resource utilization in the hardware. Table 511 lists the different buffer configurations. For each size the table shows how many concurrent packets the EDCL can handle, the maximum size of each packet including headers and the maximum size of the data payload. Sending more packets before receiving a reply than specified for the selected buffer size will lead to dropped packets. The behavior is unspecified if sending larger packets than the maximum allowed.

*Table 511.*EDCL buffer sizes

Total buffer size (kB)	Number of packet buffers	Packet buffer size (B)	Maximum data payload (B)
1	4	256	200
2	4	512	456
4	8	512	456
8	8	1024	968
16	16	1024	968
32	32	1024	968
64	64	1024	968

43.7 Media Independent Interfaces

There are several interfaces defined between the MAC sublayer and the Physical layer. The GRETH_GBIT supports the Media Independent Interface (MII) and the Gigabit Media Independent Interface (GMII).

The GMII is used in 1000 Mbit mode and the MII in 10 and 100 Mbit. These interfaces are defined separately in the 802.3-2002 standard but in practice they share most of the signals. The GMII has 9 additional signals compared to the MII. Four data signals are added to the receiver and transmitter data interfaces respectively and a new transmit clock for the gigabit mode is also introduced.

*Table 512.*Signals in GMII and MII.

MII and GMII	**GMII Only**
txd[3:0]	txd[7:4]
tx_en	rxd[7:4]
tx_er	gtx_clk
rx_col	
rx_crs	
rxd[3:0]	
rx_clk	
rx_er	
rx_dv	

43.8 Registers

The core is programmed through registers mapped into APB address space.

*Table 513.*GRETH_GBIT registers

APB address offset	**Register**
0x0	Control register
0x4	Status/Interrupt-source register
0x8	MAC Address MSB
0xC	MAC Address LSB
0x10	MDIO Control/Status
0x14	Transmit descriptor pointer
0x18	Receiver descriptor pointer
0x1C	EDCL IP
0x20	Hash table msb
0x24	Hash table lsb
0x28	EDCL MAC address MSB
0x2C	EDCL MAC address LSB
0x10000 - 0x107FC	Transmit RAM buffer debug access
0x20000 - 0x207FC	Receiver RAM buffer debug access
0x30000 - 0x3FFFC	EDCL buffer debug access

Table 514. GRETH control register

31	30 28	27	26	25	24 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ED	BS	GA	MA	MC	RESERVED	ED	RD	DD	ME	PI	BM	GB	SP	RS	PR	FD	RI	TI	RE	TE

31	EDCL available (ED) - Set to one if the EDCL is available.
30: 28	EDCL buffer size (BS) - Shows the amount of memory used for EDCL buffers. 0 = 1 kB, 1 = 2 kB, ..., 6 = 64 kB.
27	Gigabit MAC available (GA) - This bit always reads as a 1 and indicates that the MAC has 1000 Mbit capability.
26	Mdio interrupts enabled (ME) - Set to one when the core supports mdio interrupts. Read only.
25	Multicast available (MC) - Set to one when the core supports multicast address reception. Read only.
24: 15	RESERVED
14	EDCL Disable(ED) - Set to one to disablethe EDCL and zero to enable it. Reset value taken from the ethi.edcldisable signal. Only available if the EDCL hardware is present in the core.
13	RAM debug enable (RD) - Set to one to enable the RAM debug mode. Reset value: ‘0’. Only available if the VHDL generic ramdebug is nonzero.
12	Disable duplex detection (DD) - Disable the EDCL speed/duplex detection FSM. If the FSM cannot complete the detection the MDIO interface will be locked in busy mode. If software needs to access the MDIO the FSM can be disabled here and as soon as the MDIO busy bit is 0 the interface is available. Note that the FSM cannot be reenabled again.
11	Multicast enable (ME) - Enable reception of multicast addresses. Reset value: ‘0’.
10	PHY status change interrupt enable (PI) - Enables interrupts for detected PHY status changes.
9	Burstmode (BM) - When set to 1, transmissions use burstmode in 1000 Mbit Half-duplex mode (GB=1, FD = 0). When 0 in this speed mode normal transmissions are always used with extension inserted. Operation is undefined when set to 1 in other speed modes. Reset value: ‘0’.
8	Gigabit (GB) - 1 sets the current speed mode to 1000 Mbit and when set to 0, the speed mode is selected with bit 7 (SP). Reset value: ‘0’.
7	Speed (SP) - Sets the current speed mode. 0 = 10 Mbit, 1 = 100 Mbit. Must not be set to 1 at the same time as bit 8 (GB). Reset valuie: ‘0’.
6	Reset (RS) - A one written to this bit resets the GRETH_GBIT core. Self clearing. No other accesses should be done .to the slave interface other than polling this bit until it is cleared.
5	Promiscuous mode (PM) - If set, the GRETH_GBIT operates in promiscuous mode which means it will receive all packets regardless of the destination address. Reset value: ‘0’.
4	Full duplex (FD) - If set, the GRETH_GBIT operates in full-duplex mode otherwise it operates in half-duplex. Reset value: ‘0’.
3	Receiver interrupt (RI) - Enable Receiver Interrupts. An interrupt will be generated each time a packet is received when this bit is set. The interrupt is generated regardless if the packet was received successfully or if it terminated with an error. Reset value: ‘0’.
2	Transmitter interrupt (TI) - Enable Transmitter Interrupts. An interrupt will be generated each time a packet is transmitted when this bit is set. The interrupt is generated regardless if the packet was transmitted successfully or if it terminated with an error. Reset value: ‘0’.
1	Receive enable (RE) - Should be written with a one each time new descriptors are enabled. As long as this bit is one the GRETH_GBIT will read new descriptors and as soon as it encounters a disabled descriptor it will stop until RE is set again. This bit should be written with a one after the new descriptors have been enabled. Reset value: ‘0’.
0	Transmit enable (TE) - Should be written with a one each time new descriptors are enabled. As long as this bit is one the GRETH_GBIT will read new descriptors and as soon as it encounters a disabled descriptor it will stop until TE is set again. This bit should be written with a one after the new descriptors have been enabled. Reset value: ‘0’.

Table 515. GRETH_GBIT status register.

31 9	8	7	6	5	4	3	2	1	0
RESERVED	PS	IA	TS	TA	RA	TI	RI	TE	RE

Table 515. GRETH_GBIT status register.

31: 9	RESERVED
8	PHY status changes (PS) - Set each time a PHY status change is detected.
7	Invalid address (IA) - A packet with an address not accepted by the MAC was received. Cleared when written with a one. Reset value: ‘0’.
6	Too small (TS) - A packet smaller than the minimum size was received. Cleared when written with a one. Reset value: ‘0’.
5	Transmitter AHB error (TA) - An AHB error was encountered in transmitter DMA engine. Cleared when written with a one. Not Reset.
4	Receiver AHB error (RA) - An AHB error was encountered in receiver DMA engine. Cleared when written with a one. Not Reset.
3	Transmit successful (TI) - A packet was transmitted without errors. Cleared when written with a one. Not Reset.
2	Receive successful (RI) - A packet was received without errors. Cleared when written with a one. Not Reset.
1	Transmitter error (TE) - A packet was transmitted which terminated with an error. Cleared when written with a one. Not Reset.
0	Receiver error (RE) - A packet has been received which terminated with an error. Cleared when written with a one. Not Reset.

Table 516. GRETH_GBIT MAC address MSB.

31 16	15 0
RESERVED	Bit 47 downto 32 of the MAC Address

31: 16	RESERVED
15: 0	The two most significant bytes of the MAC Address. Not Reset.

Table 517. GRETH_GBIT MAC address LSB.

31 0
Bit 31 downto 0 of the MAC Address

31: 0	The 4 least significant bytes of the MAC Address. Not Reset.

Table 518. GRETH_GBIT MDIO control/status register.

31 16	15 11	10 6	5	4	3	2	1	0
DATA	PHYADDR	REGADDR		NV	BU	LF	RD	WR

31: 16	Data (DATA) - Contains data read during a read operation and data that is transmitted is taken from this field. Reset value: 0x0000.
15: 11	PHY address (PHYADDR) - This field contains the address of the PHY that should be accessed during a write or read operation. Reset value: “00000”.
10: 6	Register address (REGADDR) - This field contains the address of the register that should be accessed during a write or read operation. Reset value: ‘”00000”.
5	RESERVED
4	Not valid (NV) - When an operation is finished (BUSY = 0) this bit indicates whether valid data has been received that is, the data field contains correct data. Reset value: ‘0’.
3	Busy (BU) - When an operation is performed this bit is set to one. As soon as the operation is finished and the management link is idle this bit is cleared. Reset value: ‘0’.

Table 518. GRETH_GBIT MDIO control/status register.

2	Linkfail (LF) - When an operation completes (BUSY = 0) this bit is set if a functional management link was not detected. Reset value: ‘1’.
1	Read (RD) - Start a read operation on the management interface. Data is stored in the data field. Reset value: ‘0’.
0	Write (WR) - Start a write operation on the management interface. Data is taken from the Data field. Reset value: ‘0’.

Table 519. GRETH_GBIT transmitter descriptor table base address register.

31 - 10	9 - 3	2 - 0
BASEADDR	DESCPNT	RES

31: 10	Transmitter descriptor table base address (BASEADDR) - Base address to the transmitter descriptor table.Not Reset.
9: 3	Descriptor pointer (DESCPNT) - Pointer to individual descriptors. Automatically incremented by the Ethernet MAC.
2: 0	RESERVED

Table 520. GRETH_GBIT receiver descriptor table base address register.

31 - 10	9 - 3	2 - 0
BASEADDR	DESCPNT	RES

31: 10	Receiver descriptor table base address (BASEADDR) - Base address to the receiver descriptor table.Not Reset.
9: 3	Descriptor pointer (DESCPNT) - Pointer to individual descriptors. Automatically incremented by the Ethernet MAC.
2: 0	RESERVED

Table 521. GRETH_GBIT EDCL IP register

31 - 0
EDCL IP ADDRESS

31: 0	EDCL IP address. Reset value is set with the ipaddrh and ipaddrl generics.

Table 522. GRETH Hash table msb register

31 - 0
Hash table (64:32)

31: 0	Hash table msb. Bits 64 downto 32 of the hash table.

Table 523. GRETH Hash table lsb register

31 - 0
Hash table (64:32)

31: 0	Hash table lsb. Bits 31downto 0 of the hash table.

Table 524. GRETH_GBIT EDCL MAC address MSB.

31 16	15 0
RESERVED	Bit 47 downto 32 of the EDCL MAC Address

31: 16	RESERVED
15: 0	The two most significant bytes of the EDCL MAC Address. Hardcoded reset value set with the VHDL generic macaddrh.

Table 525. GRETH_GBIT EDCL MAC address LSB.

31 0
Bit 31 downto 0 of the EDCL MAC Address

31: 0	The 4 least significant bytes of the EDCL MAC Address. Hardcoded reset value set with the VHDL generics macaddrh and macaddrl. If the VHDL generic edcl is set to 2 bits 3 downto 0 are set with the edcladdr input signal.

43.9 Software drivers

Drivers for the GRETH_GBIT MAC is provided for the following operating systems: RTEMS, eCos, uClinux and Linux-2.6. The drivers are freely available in full source code under the GPL license from Aeroflex Gaisler's web site (http://www.gaisler.com/).

43.10 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x01D. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

43.11 Configuration options

Table 526 shows the configuration options of the core (VHDL generics).*) Not all addresses are allowed and most NICs and protocol implementations will discard frames with illegal addresses silently. Consult network literature if unsure about the addresses.

*Table 526.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRETH.	0 - NAHBIRQ-1	0
memtech	Memory technology used for the FIFOs.	0 - NTECH	inferred
ifg_gap	Number of ethernet clock cycles used for one interframe gap. Default value as required by the standard. Do not change unless you know what your doing.	1 - 255	24
attempt_limit	Maximum number of transmission attempts for one packet. Default value as required by the standard.	1 - 255	16

*Table 526.*Configuration options

Generic	Function	Allowed range	Default
backoff_limit	Limit on the backoff size of the backoff time. Default value as required by the standard. Sets the number of bits used for the random value. Do not change unless you know what your doing.	1 - 10	10
slot_time	Number of ethernet clock cycles used for one slot- time. Default value as required by the ethernet standard. Do not change unless you know what you are doing.	1 - 255	128
mdcscaler	Sets the divisor value use to generate the mdio clock (mdc). The mdc frequency will be clk/(2*(mdcscaler+1)).	0 - 255	25
nsync	Number of synchronization registers used.	1 - 2	2
edcl	Enable EDCL. 0 = disabled. 1 = enabled. 2 = enabled and 4-bit LSB of IP and ethernet MAC address programmed by ethi.edcladdr, 3=in addition to features for value 2 the reset value for the EDCL disable bit is taken from the ethi.edcldisable signal instead of being hardcoded to 0.	0 - 3	0
edclbufsz	Select the size of the EDCL buffer in kB.	1 - 64	1
burstlength	Sets the maximum burstlength used during DMA	4 - 128	32
macaddrh	Sets the upper 24 bits of the EDCL MAC address. Not all addresses are allowed and most NICs and protocol implementations will discard frames with illegal addresses silently. Consult network literature if unsure about the addresses.	0 - 16#FFFFFF#	16#00005E#
macaddrl	Sets the lower 24 bits of the EDCL MAC address. Not all addresses are allowed and most NICs and protocol implementations will discard frames with illegal addresses silently. Consult network literature if unsure about the addresses.	0 - 16#FFFFFF#	16#000000#
ipaddrh	Sets the upper 16 bits of the EDCL IP address reset value.	0 - 16#FFFF#	16#C0A8#
ipaddrl	Sets the lower 16 bits of the EDCL IP address reset value.	0 - 16#FFFF#	16#0035#
phyrstadr	Sets the reset value of the PHY address field in the MDIO register. When set to 32, the address is taken from the ethi.phyrstaddr signal.	0 - 32	0
sim	Set to 1 for simulations and 0 for synthesis. 1 selects a faster mdc clock to speed up simulations.	0 - 1	0
mdint_pol	Selects polarity for level sensitive PHY interrupt line. 0 = active low, 1 = active high	0 - 1	0
enable_mdint	Enables mdio interrupts.	0 - 1	0
multicast	Enables multicast support.	0 - 1	0
ramdebug	Enables debug access to the core's RAM blocks through the APB interface. 1=enables access to the receiver and transmitter RAM buffers, 2=enables access to the EDCL buffers in addition to the functionality of value 1. Setting this generic to 2 will have no effect if the edcl generic is 0.	0 - 2	0
ehindex	AHB master index for the separate EDCL master interface. Only used if edclsepahb is 1.	0 - NAHBMST-1	0
edclsepahb	Enables separate EDCL AHB master interface. A signal determines if the separate interface or the common interface is used. Only available in the GRETH_GBIT_MB version of the core.	0 - 1	0
mdiohold	Set output hold time for MDIO in number of AHB cycles. Should be 10 ns or more.	1 - 30	1

43.12 Signal descriptions

Table 527 shows the interface signals of the core (VHDL ports).

*Table 527.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
ETHI	gtx_clk	Input	Ethernet gigabit transmit clock.	-
	rmii_clk	Input	Ethernet RMII clock.	-
	tx_clk	Input	Ethernet transmit clock.	-
	rx_clk	Input	Ethernet receive clock.	-
	rxd	Input	Ethernet receive data.	-
	rx_dv	Input	Ethernet receive data valid.	High
	rx_er	Input	Ethernet receive error.	High
	rx_col	Input	Ethernet collision detected. (Asynchronous, sampled with tx_clk)	High
	rx_crs	Input	Ethernet carrier sense. (Asynchronous, sampled with tx_clk)	High
	mdio_i	Input	Ethernet management data input	-
	mdint	Input	Ethernet management interrupt	-
	phyrstaddr	Input	Reset address for GRETH PHY address field.	-
	edcladdr	Input	Sets the four least significant bits of the EDCL MAC address and the EDCL IP address when the edcl generic is set to 2.	-
	edclsepahb	Input	Selects AHB master interface for the EDCL. ‘0’ selects the common interface and ‘1’ selects the separate interface. Only available in the GRETH_GBIT_MB version of the core when the VHDL generic edclsepahb is set to 1.	-
	edcldisable	Input	Reset value for edcl disable register bit. Setting the signal to 1 disables the EDCL at reset and 0 enables it.	-
ETHO	reset	Output	Ethernet reset (asserted when the MAC is reset).	Low
	txd	Output	Ethernet transmit data.	-
	tx_en	Output	Ethernet transmit enable.	High
	tx_er	Output	Ethernet transmit error.	High
	mdc	Output	Ethernet management data clock.	-
	mdio_o	Output	Ethernet management data output.	-
	mdio_oe	Output	Ethernet management data output enable.	Set by the oepol generic.

* see GRLIB IP Library User’s Manual

43.13 Library dependencies

Table 528 shows libraries used when instantiating the core (VHDL libraries).

*Table 528.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	ETHERNET_MAC	Signals, component	GRETH_GBIT component declarations, GRETH_GBIT signals.
GAISLER	NET	Signals	Ethernet signals

43.14 Instantiation

The first example shows how the non-mb version of the core can be instantiated and the second one show the mb version.

43.14.1 Non-MB version

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.ethernet_mac.all;

entity greth_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- ethernet signals
    ethi :  in  eth_in_type;
    etho :  in  eth_out_type
    );
end;

architecture rtl of greth_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

begin

  -- AMBA Components are instantiated here
  ...

  -- GRETH
  e1 : greth_gbit
    generic map(
     hindex       => 0,
     pindex       => 12,
     paddr        => 12,
     pirq         => 12,
     memtech      => inferred,
     mdcscaler    => 50,
     burstlength  => 32,
     nsync        => 1,
     edcl         => 1,
     edclbufsz    => 8,
     macaddrh     => 16#00005E#,
     macaddrl     => 16#00005D#,
     ipaddrh      => 16#c0a8#,
     ipaddrl      => 16#0035#)
 port map(
   rst          => rstn,
   clk          => clk,
   ahbmi        => ahbmi,
   ahbmo        => ahbmo(0),
   apbi         => apbi,
   apbo         => apbo(12),
   ethi         => ethi,
   etho         => etho
   );
end;
```

43.14.2 MB version

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.ethernet_mac.all;

entity greth_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- ethernet signals
    ethi :  in  eth_in_type;
    etho :  in  eth_out_type
    );
end;

architecture rtl of greth_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
begin

  -- AMBA Components are instantiated here
  ...

  -- GRETH
  e1 : greth_gbit_mb
    generic map(
     hindex       => 0,
     pindex       => 12,
     paddr        => 12,
     pirq         => 12,
     memtech      => inferred,
     mdcscaler    => 50,
     burstlength  => 32,
     nsync        => 1,
     edcl         => 1,
     edclbufsz    => 8,
     macaddrh     => 16#00005E#,
     macaddrl     => 16#00005D#,
     ipaddrh      => 16#c0a8#,
     ipaddrl      => 16#0035#,
     ehindex      => 1
```

```
        edclsepahb   => 1)
    port map(
      rst          => rstn,
      clk          => clk,
      ahbmi        => ahbmi,
      ahbmo        => ahbmo(0),
      ahbmi2       => ahbmi,
      ahbmo2       => ahbmo(1),
      apbi         => apbi,
      apbo         => apbo(12),
      ethi         => ethi,
      etho         => etho
      );
   end;
```

44 GRFIFO - FIFO Interface

44.1 Overview

The FIFO interface is assumed to operate in an AMBA bus system where both the AMBA AHB bus and the APB bus are present. The AMBA APB bus is used for configuration, control and status handling. The AMBA AHB bus is used for retrieving and storing FIFO data in memory external to the FIFO interface. This memory can be located on-chip or external to the chip.

The FIFO interface supports transmission and reception of blocks of data by use of circular buffers located in memory external to the core. Separate transmit and receive buffers are assumed. Reception and transmission of data can be ongoing simultaneously.

After a data transfer has been set up via the AMBA APB interface, the DMA controller initiates a burst of read accesses on the AMBA AHB bus to fetch data from memory that are performed by the AHB master. The data are then written to the external FIFO. When a programmable amount of data has been transmitted, the DMA controller issues an interrupt.

After reception has been set up via the AMBA APB interface, data are read from the external FIFO. To store data to memory, the DMA controller initiates a burst of write accesses on the AMBA AHB bus that are performed by the AHB master. When a programmable amount of data has been received, the DMA controller issues an interrupt.

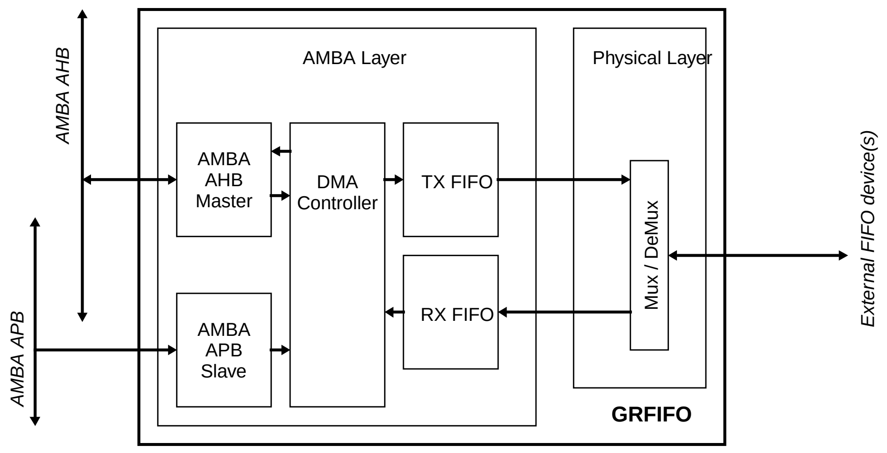


Figure 145. Block diagram

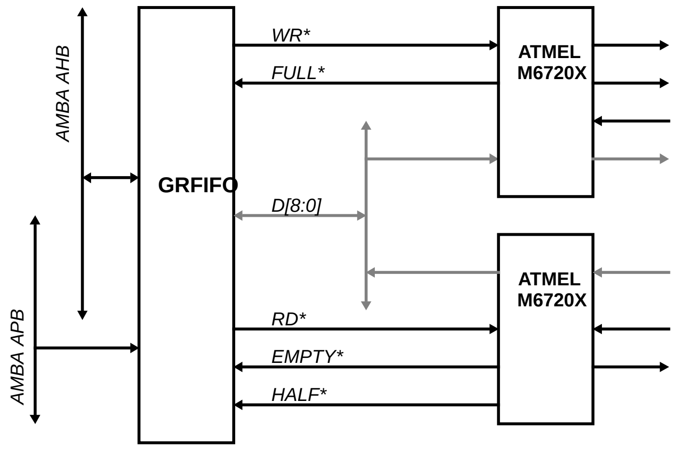


Figure 146. Example of usage, with shared external data bus

44.1.1 Function

The core implements the following functions:

- data transmission to external FIFO
- circular transmit buffer
- direct memory access for transmitter
- data reception from external FIFO
- circular receive buffer for receiver
- direct memory access
- automatic 8- and 16-bit data width conversion
- general purpose input output

44.1.2 Transmission

Data to be transferred via the FIFO interface are fetched via the AMBA AHB master interface from on-chip or off-chip memory. This is performed by means of direct memory access (DMA), implementing a circular transmit buffer in the memory. The transmit channel is programmable via the AMBA APB slave interface, which is also used for the monitoring of the FIFO and DMA status.

The transmit channel is programmed in terms of a base address and size of the circular transmit buffer. The outgoing data are stored in the circular transmit buffer by the system. A write address pointer register is then set by the system to indicate the last byte written to the circular transmit buffer. An interrupt address pointer register is used by the system to specify a location in the circular transmit buffer from which a data read should cause an interrupt to be generated.

The FIFO interface automatically indicates with a read address pointer register the location of the last fetched byte from the circular transmit buffer. Read accesses are performed as incremental bursts, except when close to the location specified by the interrupt pointer register at which point the last bytes might be fetched by means of single accesses.

Data transferred via the FIFO interface can be either 8- or 16-bit wide. The handling of the transmit channel is however the same. All transfers performed by the AMBA AHB master are 32-bit word based. No byte or half-word transfers are performed.

To handle the 8- and 16-bit FIFO data width, a 32-bit read access might carry less than four valid bytes. In such a case, the remaining bytes are ignored. When additional data are available in the circular transmit buffer, the previously fetched bytes will be re-read together with the newly written bytes to form the 32-bit data. Only the new bytes will be transmitted to the FIFO, not to transmit the same byte more than once. The aforementioned write address pointer indicates what bytes are valid.

An interrupt is generated when the circular transmit buffer is empty. The status of the external FIFO is observed via the AMBA APB slave interface, indicating Full Flag and Half-Full Flag.

44.1.3 Reception

Data received via the FIFO interface are stored via the AMBA AHB master interface to on-chip or off-chip memory. This is performed by means of direct memory access (DMA), implementing a circular receive buffer in the memory. The receive channel is programmable via the AMBA APB slave interface, which is also used for the monitoring of the FIFO and DMA status.

The receive channel is programmed in terms of a base address and size of the circular receive buffer. The incoming data are stored in the circular receive buffer. The interface automatically indicates with a write address pointer register the location of the last stored byte. A read address pointer register is used by the system to indicate the last byte read from the circular receive buffer. An interrupt address pointer register is used by the system to specify a location in the circular receive buffer to which a data write should cause an interrupt to be generated.

Write accesses are performed as incremental bursts, except when close to the location specified by the interrupt pointer register at which point the last bytes might be stored by means of single accesses.

Data transferred via the FIFO interface can be either 8- or 16-bit wide. The handling of the receive channel is however the same. All transfers performed by the AMBA AHB master are 32-bit word based. No byte or half-word transfers are performed.

To handle the 8- and 16-bit FIFO data width, a 32-bit write access might carry less than four valid bytes. In such a case, the remaining bytes will all be zero. When additional data are received from the FIFO interface, the previously stored bytes will be re-written together with the newly received bytes to form the 32-bit data. In this way, the previously written bytes are never overwritten. The aforementioned write address pointer indicates what bytes are valid.

An interrupt is generated when the circular receive buffer is full. If more FIFO data are available, they will not be moved to the circular receive buffer. The status of the external FIFO is observed via the AMBA APB slave interface, indicating Empty Flag and Half-Full Flag.

44.1.4 General purpose input output

Data input and output signals unused by the FIFO interface can be used as general purpose input output, providing 0, 8 or 16 individually programmable channels.

44.1.5 Interfaces

The core provides the following external and internal interfaces:

- FIFO interface
- AMBA AHB master interface, with sideband signals as per [GLRIB] including:
- cachability information
- interrupt bus
- configuration information
- diagnostic information
- AMBA APB slave interface, with sideband signals as per [GLRIB] including:
- interrupt bus
- configuration information
- diagnostic information

The interface is intended to be used with the following FIFO devices from ATMEL:

Name:Type:

M67204H4K x 9 FIFOESA/SCC 9301/049, SMD/5962-89568

M67206H16K x 9 FIFOESA/SCC 9301/048, SMD/5962-93177

M672061H16K x 9 FIFO ESA/SCC 9301/048, SMD/5962-93177

44.2 Interface

The external interface supports one or more FIFO devices for data output (transmission) and/or one or more FIFO devices for data input (reception). The external interface supports FIFO devices with 8- and 16-bit data width. Note that one device is used when 8-bit and two devices are used when 16-bit data width is needed. The data width is programmable. Note that this is performed commonly for both directions.

The external interface supports one parity bit over every 8 data bits. Note that there can be up to two parity bits in either direction. The parity is programmable in terms of odd or even parity. Note that odd parity is defined as an odd number of logical ones in the data bits and parity bit. Note that even parity is defined as an even number of logical ones in the data bits and parity bit. Parity is generated for write accesses to the external FIFO devices. Parity is checked for read accesses from the external FIFO devices and a parity failure results in an internal interrupt.

The external interface provides a Write Enable output signal. The external interface provides a Read Enable output signal. The timing of the access towards the FIFO devices is programmable in terms of wait states based on system clock periods.

The external interface provides an Empty Flag input signal, which is used for flow-control during the reading of data from the external FIFO, not reading any data while the external FIFO is empty. Note that the Empty Flag is sampled at the end of the read access to determine if the FIFO is empty. To determine when the FIFO is not empty, the Empty Flag is re-synchronized with Clk.

The external interface provides a Full Flag input signal, which is used for flow-control during the writing of data to the external FIFO, not writing any data while the external FIFO is full. Note that the Full Flag is sampled at the end of the write access to determine if the FIFO is full. To determine when the FIFO is not full, the Full Flag is re-synchronized with Clk.

The external interface provides a Half-Full Flag input signal, which is used as status information only.

The data input and output signals are possible to use as general purpose input output channels. This need is only realized when the data signals are not used by the FIFO interface. Each general purpose input output channel is individually programmed as input or output. The default reset configuration for each general purpose input output channel is as input. The default reset value each general purpose input output channel is logical zero. Note that protection toward spurious pulse commands during power up shall be mitigated as far as possible by means of I/O cell selection from the target technology.

44.3 Waveforms

The following figures show read and write accesses to the FIFO with 0 and 4 wait states, respectively.

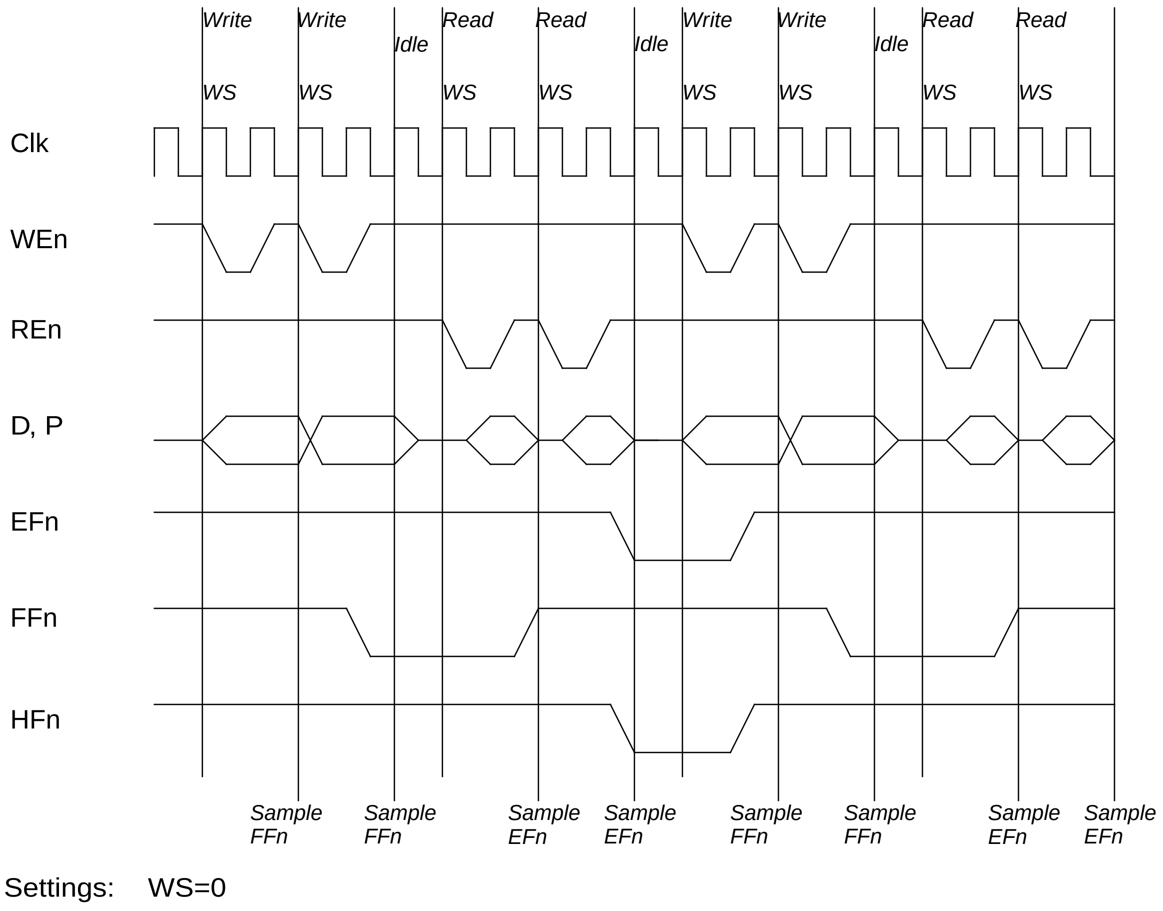
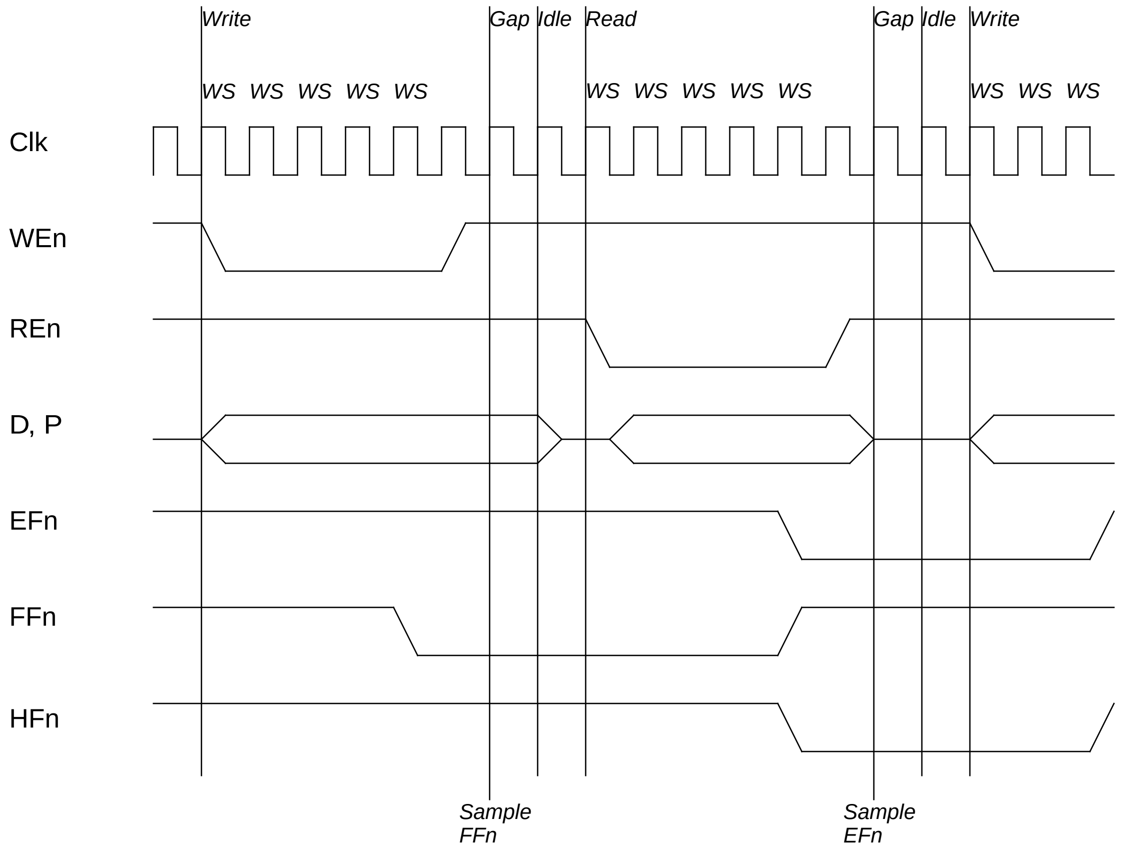


Figure 147. FIFO read and write access waveform, 0 wait states (WS)



Settings: WS=4 (with additional gap between accesses)

Figure 148. FIFO read and write access waveform, 4 wait states (WS)

44.4 Transmission

The transmit channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The transmit channel can be enabled or disabled.

44.4.1 Circular buffer

The transmit channel operates on a circular buffer located in memory external to the FIFO controller. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the AMBA AHB master interface.

The size of the buffer is defined by the FifoTxSIZE.SIZE field, specifying the number of 64 byte blocks that fit in the buffer.

E.g. FifoTxSIZE.SIZE = 1 means 64 bytes fit in the buffer.

Note however that it is not possible to fill the buffer completely, leaving at least one word in the buffer empty. This is to simplify wrap-around condition checking.

E.g. FifoTxSIZE.SIZE = 1 means that 60 bytes fit in the buffer at any given time.

44.4.2 Write and read pointers

The write pointer (FifoTxWR.WRITE) indicates the position+1 of the last byte written to the buffer. The write pointer operates on number of bytes, not on absolute or relative addresses.

The read pointer (FifoTxRD.READ) indicates the position+1 of the last byte read from the buffer. The read pointer operates on number of bytes, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of bytes available in the buffer for transmission. The difference is calculated using the buffer size, specified by the FifoTxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 bytes available for transmit when FifoTxSIZE.SIZE=1, FifoTxWR.WRITE=2 and FifoTxRD.READ=0.
- There are 2 bytes available for transmit when FifoTxSIZE.SIZE=1, FifoTxWR.WRITE =0 and FifoTxRD.READ =62.
- There are 2 bytes available for transmit when FifoTxSIZE.SIZE=1, FifoTxWR.WRITE =1 and FifoTxRD.READ =63.
- There are 2 bytes available for transmit when FifoTxSIZE.SIZE=1, FifoTxWR.WRITE =5 and FifoTxRD.READ =3.

When a byte has been successfully written to the FIFO, the read pointer (FifoTxRD.READ) is automatically incremented, taking wrap around effects of the circular buffer into account. Whenever the write pointer FifoTxWR.WRITE and read pointer FifoTxRD.READ are equal, there are no bytes available for transmission.

44.4.3 Location

The location of the circular buffer is defined by a base address (FifoTxADDR.ADDR), which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

44.4.4 Transmission procedure

When the channel is enabled (FifoTxCTRL.ENABLE=1), as soon as there is a difference between the write and read pointer, a transmission will be started. Note that the channel should not be enabled if a potential difference between the write and read pointers could be created, to avoid the data transmission to start prematurely.

A data transmission will begin with a fetch of the data from the circular buffer to a local buffer in the FIFO controller. After a successful fetch, a write access will be performed to the FIFO.

The read pointer (FifoTxRD.READ) is automatically incremented after a successful transmission, taking wrap around effects of the circular buffer into account. If there is at least one byte available in the circular buffer, a new fetch will be performed.

If the write and read pointers are equal, no more prefetches and fetches will be performed, and transmission will stop.

Interrupts are provided to aid the user during transmission, as described in detail later in this section. The main interrupts are the TxError, TxEmpty and TxIrq which are issued on the unsuccessful transmission of a byte due to an error condition on the AMBA bus, when all bytes have been transmitted successfully and when a predefined number of bytes have been transmitted successfully.

Note that 32-bit wide read accesses past the address of the last byte or halfword available for transmission can be performed as part of a burst operation, although no read accesses are made beyond the circular buffer size.

All accesses to the AMBA AHB bus are performed as two consecutive 32-bit accesses in a burst, or as a single 32-bit access in case of an AMBA AHB bus error.

44.4.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (FifoTxADDR.ADDR) field.

While the channel is disabled, the read pointer (FifoTxRD.READ) can be changed to an arbitrary value pointing to the first byte to be transmitted, and the write pointer (FifoTxWR.WRITE) can be changed to an arbitrary value.

When the channel is enabled, the transmission will start from the read pointer and continue to the write pointer.

44.4.6 AMBA AHB error

An AHB error response occurring on the AMBA AHB bus while data is being fetched will result in a TxError interrupt.

If the FifoCONF.ABORT bit is set to 0b, the channel causing the AHB error will re-try to read the data being fetched from memory till successful.

If the FifoCONF.ABORT bit is set to 1b, the channel causing the AHB error will be disabled (FifoTxCTRL.ENABLE is cleared automatically to 0 b). The read pointer can be used to determine which data caused the AHB error. The interface will not start any new write accesses to the FIFO. Any ongoing FIFO access will be completed and the FifoTxSTAT.TxOnGoing bit will be cleared. When the channel is re-enabled, the fetch and transmission of data will resume at the position where it was disabled, without losing any data.

44.4.7 Enable and disable

When an enabled transmit channel is disabled (FifoTxCTRL.ENABLE=0b), the interface will not start any new read accesses to the circular buffer by means of DMA over the AMBA AHB bus. No new write accesses to the FIFO will be started. Any ongoing FIFO access will be completed. If the data is written successfully, the read pointer (FifoTxRD.READ) is automatically incremented and the FifoTxSTAT.TxOnGoing bit will be cleared. Any associated interrupts will be generated.

Any other fetched or pre-fetched data from the circular buffer which is temporarily stored in the local buffer will be discarded, and will be fetched again when the transmit channel is re-enabled.

The progress of the any ongoing access can be observed via the FifoTxSTAT.TxOnGoing bit. The FifoTxSTAT.TxOnGoing must be 0b before the channel can be re-configured safely (i.e. changing address, size or read/write pointers). It is also possible to wait for the TxEmpty interrupt described hereafter.

The channel can be re-enabled again without the need to re-configure the address, size and pointers. No data transmission is started while the channel is not enabled.

44.4.8 Interrupts

During transmission several interrupts can be generated:

• TxEmpty:	Successful transmission of all data in buffer
• TxIrq:	Successful transmission of a predefined number of data
• TxError:	AHB access error during transmission

The TxEmpty and TxIrq interrupts are only generated as the result of a successful data transmission, after the FifoTxRD.READ pointer has been incremented.

44.5 Reception

The receive channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The receive channel can be enabled or disabled.

44.5.1 Circular buffer

The receive channel operates on a circular buffer located in memory external to the FIFO controller. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the AMBA AHB master interface.

The size of the buffer is defined by the FifoRxSIZE.SIZE field, specifying the number 64 byte blocks that fit in the buffer.

E.g. FifoRxSIZE.SIZE=1 means 64 bytes fit in the buffer.

Note however that it is not possible for the hardware to fill the buffer completely, leaving at least two words in the buffer empty. This is to simplify wrap-around condition checking.

E.g. FifoRxSIZE.SIZE=1 means that 56 bytes fit in the buffer at any given time.

44.5.2 Write and read pointers

The write pointer (FifoRxWR.WRITE) indicates the position+1 of the last byte written to the buffer. The write pointer operates on number of bytes, not on absolute or relative addresses.

The read pointer (FifoRxRD.READ) indicates the position+1 of the last byte read from the buffer. The read pointer operates on number of bytes, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of bytes available in the buffer for reception. The difference is calculated using the buffer size, specified by the FifoRxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 bytes available for read-out when FifoRxSIZE.SIZE=1, FifoRxWR.WRITE =2 and FifoRxRD.READ=0.
- There are 2 bytes available for read-out when FifoRxSIZE.SIZE=1, FifoRxWR.WRITE =0 and FifoRxRD.READ=62.
- There are 2 bytes available for read-out when FifoRxSIZE.SIZE=1, FifoRxWR.WRITE =1 and FifoRxRD.READ=63.
- There are 2 bytes available for read-out when FifoRxSIZE.SIZE=1, FifoRxWR.WRITE =5 and FifoRxRD.READ=3.

When a byte has been successfully received and stored, the write pointer (FifoRxWR.WRITE) is automatically incremented, taking wrap around effects of the circular buffer into account.

44.5.3 Location

The location of the circular buffer is defined by a base address (FifoRxADDR.ADDR), which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

44.5.4 Reception procedure

When the channel is enabled (FifoRxCTRL.ENABLE=1), and there is space available for data in the circular buffer (as defined by the write and read pointer), a read access will be started towards the FIFO, and then an AMBA AHB store access will be started. The received data will be temporarily stored in a local store-buffer in the FIFO controller. Note that the channel should not be enabled until the write and read pointers are configured, to avoid the data reception to start prematurely

After a datum has been successfully stored the FIFO controller is ready to receive new data. The write pointer (FifoRxWR.WRITE) is automatically incremented, taking wrap around effects of the circular buffer into account.

Interrupts are provided to aid the user during reception, as described in detail later in this section. The main interrupts are the RxError, RxParity, RxFull and RxIrq which are issued on the unsuccessful reception of data due to an AMBA AHB error or parity error, when the buffer has been successfully filled and when a predefined number of data have been received successfully.

All accesses to the AMBA AHB bus are performed as two consecutive 32-bit accesses in a burst, or as a single 32-bit access in case of an AMBA AHB bus error.

44.5.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (FifoRxADDR.ADDR) field.

While the channel is disabled, the write pointer (FifoRxWR.WRITE) can be changed to an arbitrary value pointing to the first data to be received, and the read pointer (FifoRxRD.READ) can be changed to an arbitrary value.

When the channel is enabled, the reception will start from the write pointer and continue to the read pointer.

44.5.6 AMBA AHB error

An AHB error response occurring on the AMBA AHB bus while data is being stored will result in an RxError interrupt.

If the FifoCONF.ABORT bit is set to 0b, the channel causing the AHB error will retry to store the received data till successful

If the FifoCONF.ABORT bit is set to 1b, the channel causing the AHB error will be disabled (FifoRx-CTRL.ENABLE is cleared automatically to 0b). The write pointer can be used to determine which address caused the AHB error. The interface will not start any new read accesses to the FIFO. Any ongoing FIFO access will be completed and the data will be stored in the local receive buffer. The FifoRxSTAT.ONGOING bit will be cleared. When the receive channel is re-enabled, the reception and storage of data will resume at the position where it was disabled, without losing any data.

44.5.7 Enable and disable

When an enabled receive channel is disabled (FifoRxCTRL.ENABLE=0b), any ongoing data storage on the AHB bus will not be aborted, and no new storage will be started. If the data is stored successfully, the write pointer (FifoRxWR.WRITE) is automatically incremented. Any associated interrupts will be generated. The interface will not start any new read accesses to the FIFO. Any ongoing FIFO access will be completed.

The channel can be re-enabled again without the need to re-configure the address, size and pointers. No data reception is performed while the channel is not enabled.

The progress of the any ongoing access can be observed via the FifoRxSTAT.ONGOING bit. Note that the there might be data left in the local store-buffer in the FIFO controller. This can be observed via the FifoRxSTAT.RxByteCntr field. The data will not be lost if the channel is not reconfigured before re-enabled.

To empty this data from the local store-buffer to the external memory, the channel needs to be ren-abled. By setting the FifoRxIRQ.IRQ field to match the value of the FifoRxWR.WRITE field plus the value of the FifoRxSTAT.RxByteCntr field, an emptying to the external memory is forced of any data temporarily stored in the local store-buffer. Note however that additional data could be received in the local store-buffer when the channel is re-enabled.

The FifoRxSTAT.ONGOING must be 0b before the channel can be re-configured safely (i.e. changing address, size or read/write pointers).

44.5.8 Interrupts

During reception several interrupts can be generated:

- RxFull: Successful reception of all data possible to store in buffer
- RxIrq: Successful reception of a predefined number of data
- RxError: AHB access error during reception
- RxParity: Parity error during reception

The RxFull and RxIrq interrupts are only generated as the result of a successful data reception, after the FifoRxWR.WRITE pointer has been incremented.

44.6 Operation

44.6.1 Global reset and enable

When the FifoCTRL.RESET bit is set to 1b, a reset of the core is performed. The reset clears all the register fields to their default values. Any ongoing data transfers will be aborted.

44.6.2 Interrupt

Seven interrupts are implemented by the FIFO interface:

Index:	Name:	Description:
0	TxIrq	Successful transmission of block of data
1	TxEmpty	Circular transmission buffer empty
2	TxError	AMBA AHB access error during transmission
3	RxIrq	Successful reception of block of data
4	RxFull	Circular reception buffer full
5	RxError	AMBA AHB access error during reception
6	RxParity	Parity error during reception

The interrupts are configured by means of the *pirq* VHDL generic. The setting of the *singleirq* VHDL generic results in a single interrupt output, instead of multiple, configured by the means of the *pirq* VHDL generic, and enables the read and write of the interrupt registers. When multiple interrupts are implemented, each interrupt is generated as a one system clock period long active high output pulse. When a single interrupt is implemented, it is generated as an active high level output.

44.6.3 Reset

After a reset the values of the output signals are as follows:

Signal:	Value after reset:
FIFOO.WEn	de-asserted
FIFOO.REn	de-asserted

44.6.4 Asynchronous interfaces

The following input signals are synchronized to Clk:

- FIFOI.EFn
- FIFOI.FFn
- FIFOI.HFn

44.7 Registers

The core is programmed through registers mapped into APB address space.

*Table 529.*GRFIFO registers

APB address offset	Register
0x000	Configuration Register
0x004	Status Register
0x008	Control Register
0x020	Transmit Channel Control Register
0x024	Transmit Channel Status Register
0x028	Transmit Channel Address Register
0x02C	Transmit Channel Size Register
0x030	Transmit Channel Write Register
0x034	Transmit Channel Read Register
0x038	Transmit Channel Interrupt Register
0x040	Receive Channel Control Register
0x044	Receive Channel Status Register
0x048	Receive Channel Address Register
0x04C	Receive Channel Size Register
0x050	Receive Channel Write Register
0x054	Receive Channel Read Register
0x058	Receive Channel Interrupt Register
0x060	Data Input Register
0x064	Data Output Register
0x068	Data Direction Register
0x100	Pending Interrupt Masked Status Register
0x104	Pending Interrupt Masked Register
0x108	Pending Interrupt Status Register
0x10C	Pending Interrupt Register
0x110	Interrupt Mask Register
0x114	Pending Interrupt Clear Register

44.7.1 Configuration Register [FifoCONF] R/W

*Table 530.*Configuration Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
									Abort	DW		Parity	WS		

Field:	Description:	
6:	ABORT	Abort transfer on AHB ERROR
5-4:	DW	Data width: 00b = none 01b = 8 bitFIFOO.Dout[7:0], FIFOI.Din[7:0] 10b = 16 bitFIFOO.Dout[15:0] FIFOI.Din[15:0]

		11b = spare/none
3:	PARITY	Parity type:
		0b = even
		1b = odd
2-0:	WS	Number of wait states, 0 to 7

All bits are cleared to 0 at reset.

Note that the transmit or receive channel active during the AMBA AHB error is disabled if the ABORT bit is set to 1b. Note that all accesses on the affected channel will be disabled after an AMBA AHB error occurs while the ABORT bit is set to 1b. The accesses will be disabled until the affected channel is re-enabled setting the FifoTxCTRL.ENABLE or FifoRxCTRL.ENABLE bit, respectively.

Note that a wait states corresponds to an additional clock cycle added to the period when the read or write strobe is asserted. The default asserted width is one clock period for the read or write strobe when WS=0. Note that an idle gap of one clock cycle is always inserted between read and write accesses, with neither the read nor the write strobe being asserted.

Note that an additional gap of one clock cycle with the read or write strobe de-asserted is inserted between two accesses when WS is equal to or larger than 100b.

44.7.2 Status Register [FifoSTAT] R

*Table 531.*Status register

31 28	27 24	23 16
TxChannels	RxChannels	-

15 6	5	4 0
-	SingleIrq	-

31-28:	TxChannels	Number of TxChannels -1, 4-bit
27-24:	RxChannels	Number of RxChannels -1, 4-bit
5:	SingleIrq	Single interrupt output and interrupt registers when set to 1

44.7.3 Control Register [FifoCTRL] R/W

*Table 532.*Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
														Rese t	

1:	RESET	Reset complete FIFO interface, all registers

All bits are cleared to 0 at reset.

Note that RESET is read back as 0b.

44.7.4 Transmit Channel Control Register [FifoTxCTRL] R/W

*Table 533.*Transmit Channel Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
															Enable

0:	ENABLE	Enable channel

All bits are cleared to 0 at reset.

Note that in the case of an AHB bus error during an access while fetching transmit data, and the FifoConf.ABORT bit is 1b, then the ENABLE bit will be reset automatically.

At the time the ENABLE is cleared to 0b, any ongoing data writes to the FIFO are not aborted.

44.7.5 Transmit Channel Status Register [FifoTxSTAT] R

*Table 534.*Transmit Channel Status Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
									TxOnGoing		TxIrq	TxEmpty	TxError	FF	HF

6:	TxOnGoing	Access ongoing
4:	TxIrq	Successful transmission of block of data
3:	TxEmpty	Transmission buffer has been emptied
2:	TxError	AMB AHB access error during transmission
1:	FF	FIFO Full Flag
0:	HF	FIFO Half-Full Flag

All bits are cleared to 0 at reset.

The following sticky status bits are cleared when the register has been read:

- TxIrq, TxEmpty and TxError.

44.7.6 Transmit Channel Address Register [FifoTxADDR] R/W

*Table 535.*Transmit Channel Address Register

31 - 10	9 - 0
ADDR	

31-10:	ADDR	Base address for circular buffer

All bits are cleared to 0 at reset.

44.7.7 Transmit Channel Size Register [FifoTxSIZE] R/W

*Table 536.*Transmit Channel Size Register

31 17	16 6	5 0
	SIZE	

16-6:	SIZE	Size of circular buffer, in number of 64 bytes block

All bits are cleared to 0 at reset.

Valid SIZE values are 0, and between 1 and 1024. Note that the resulting behavior of invalid SIZE values is undefined.

Note that only SIZE*64-4 bytes can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

The width of the SIZE field is configurable indirectly by means of the VHDL generic (ptrwidth) which sets the width of the read and write data pointers. In the above example VHDL generic ptrwidth=16, making the SIZE field 11 bits wide.

44.7.8 Transmit Channel Write Register [FifoTxWR] R/W

*Table 537.*Transmit Channel Write Register

31 16	15 0
	WRITE

15-0:	WRITE	Pointer to last written byte + 1

All bits are cleared to 0 at reset.

The WRITE field is written to in order to initiate a transfer, indicating the position +1 of the last byte to transmit.

Note that it is not possible to fill the buffer. There is always one word position in buffer unused. Software is responsible for not over-writing the buffer on wrap around (i.e. setting WRITE=READ).

Note that the LSB may be ignored for 16-bit wide FIFO devices.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

44.7.9 Transmit Channel Read Register [FifoTxRD] R/W

*Table 538.*Transmit Channel Read Register

31 16	15 0
	READ

15-0:	READ	Pointer to last read byte + 1

All bits are cleared to 0 at reset.

The READ field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last byte transmitted.

Note that the READ field can be used to read out the progress of a transfer.

Note that the READ field can be written to in order to set up the starting point of a transfer. This should only be done while the transmit channel is not enabled.

Note that the READ field can be automatically incremented even if the transmit channel has been disabled, since the last requested transfer is not aborted until completed.

Note that the LSB may be ignored for 16-bit wide FIFO devices.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

44.7.10 Transmit Channel Interrupt Register [FifoTxIRQ] R/W

*Table 539.*Transmit Channel Interrupt Register

31 … 16	15 … 0
	IRQ

15-0:	IRQ	Pointer+1 to a byte address from which the read of transmitted data shall result in an interrupt

All bits are cleared to 0 at reset.

Note that this indicates that a programmed amount of data has been sent. Note that the LSB may be ignored for 16-bit wide FIFO devices.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

44.7.11 Receive Channel Control Register [FifoRxCTRL] R/W

*Table 540.*Receive Channel Control Register

31 … 2	1	0
		Enable

0:	ENABLE	Enable channel

All bits are cleared to 0 at reset.

Note that in the case of an AHB bus error during an access while storing receive data, and the Fifo-Conf.ABORT bit is 1b, then the ENABLE bit will be reset automatically.

At the time the ENABLE is cleared to 0b, any ongoing data reads from the FIFO are not aborted.

44.7.12 Receive Channel Status Register [FifoRxSTAT] R

*Table 541.*Receive Channel Status Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					RxByteCntr				RxOnGoing	RxParity	RxIrq	RxFull	RxError	EF	HF

10-8:	RxByteCntr	Number of bytes in local buffer
6:	RxOnGoing	Access ongoing
5:	RxParity	Parity error during reception
4:	RxIrq	Successful reception of block of data
3:	RxFull	Reception buffer has been filled
2:	RxError	AMB AHB access error during reception
1:	EF	FIFO Empty Flag
0:	HF	FIFO Half-Full Flag

All bits are cleared to 0 at reset.

The following sticky status bits are cleared when the register has been read:

- RxParity, RxIrq, RxFull and RxError.

The circular buffer is considered as full when there are two words or less left in the buffer.

44.7.13 Receive Channel Address Register [FifoRxADDR] R/W

*Table 542.*Receive Channel Address Register

31 10	9 0
ADDR	

31-10:	ADDR	Base address for circular buffer

All bits are cleared to 0 at reset.

44.7.14 Receive Channel Size Register [FifoRxSIZE] R/W

*Table 543.*Receive Channel Size Register

31 17	16 6	5 0
	SIZE	

16-6:	SIZE	Size of circular buffer, in number of 64 byte blocks

All bits are cleared to 0 at reset.

Valid SIZE values are 0, and between 1 and 1024. Note that the resulting behavior of invalid SIZE values is undefined.

Note that only SIZE*64-8 bytes can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

The width of the SIZE field is configurable indirectly by means of the VHDL generic (ptrwidth) which sets the width of the read and write data pointers. In the above example VHDL generic ptrwidth=16, making the SIZE field 11 bits wide.

44.7.15 Receive Channel Write Register [FifoRxWR] R/W

*Table 544.*Receive Channel Write Register

31 16	15 0
	WRITE

15-0:	WRITE	Pointer to last written byte +1

All bits are cleared to 0 at reset.

The field is implemented as relative to the buffer base address (scaled with SIZE field).

The WRITE field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last byte received.

Note that the WRITE field can be used to read out the progress of a transfer.

Note that the WRITE field can be written to in order to set up the starting point of a transfer. This should only be done while the transmit channel is not enabled.

Note that the LSB may be ignored for 16-bit wide FIFO devices.

44.7.16 Receive Channel Read Register [FifoRxRD] R/W

*Table 545.*Receive Channel Read Register

31 16	15 0
	READ

15-0:	READ	Pointer to last read byte +1

All bits are cleared to 0 at reset.

The field is implemented as relative to the buffer base address (scaled with SIZE field).

The READ field is written to in order to release the receive buffer, indicating the position +1 of the last byte that has been read out.

Note that it is not possible to fill the buffer. There is always one word position unused in the buffer. Software is responsible for not over-reading the buffer on wrap around (i.e. setting WRITE=READ).

Note that the LSB may be ignored for 16-bit wide FIFO devices

44.7.17 Receive Channel Interrupt Register [FifoRxIRQ] R/W

*Table 546.*Receive Channel Interrupt Register

31 16	15 0
	IRQ

15-0:	IRQ	Pointer+1 to a byte address to which the write of received data shall result in an interrupt

All bits are cleared to 0 at reset.

Note that this indicates that a programmed amount of data has been received.

The field is implemented as relative to the buffer base address (scaled with SIZE field).

Note that the LSB may be ignored for 16-bit wide FIFO devices.

Note that by setting the IRQ field to match the value of the Receive Channel Write Register.WRITE field plus the value of the Receive Channel Status Register.RxByteCntr field, an emptying to the external memory is forced of any data temporarily stored in the local buffer.

44.7.18 Data Input Register [FifoDIN] R

*Table 547.*Data Input Register

31 16	15 0
	DIN

15-0:	DIN	Input data	*FIFOI.Din[15:0]*

All bits are cleared to 0 at reset.

Note that only the part of FIFOI.Din[15:0] not used by the FIFO can be used as general purpose input output, see FifoCONF.DW.

Note that only bits dwidth-1 to 0 are implemented.

44.7.19 Data Output Register [FifoDOUT] R/W

*Table 548.*Data Output Register

31 16	15 0
	DOUT

15-0:	DOUT	Output data	*FIFOO.Dout[15:0]*

All bits are cleared to 0 at reset.

Note that only the part of FIFOO.Dout[15:0] not used by the FIFO can be used as general purpose input output, see FifoCONF.DW.

Note that only bits dwidth-1 to 0 are implemented.

44.7.20 Data Register [FifoDDIR] R/W

*Table 549.*Data Direction Register

31 16	15 0
	DDIR

15-0:	DDIR	Direction: 0b = input = high impedance, 1b = output = driven	*FIFOO.Dout[15:0]*

All bits are cleared to 0 at reset.

Note that only the part of FIFOO.Dout[15:0] not used by the FIFO can be used as general purpose input output, see FifoCONF.DW.

Note that only bits dwidth-1 to 0 are implemented.

44.7.21 Interrupt registers

The interrupt registers give complete freedom to the software, by providing means to mask interrupts, clear interrupts, force interrupts and read interrupt status.

When an interrupt occurs the corresponding bit in the Pending Interrupt Register is set. The normal sequence to initialize and handle a module interrupt is:

- Set up the software interrupt-handler to accept an interrupt from the module.
- Read the Pending Interrupt Register to clear any spurious interrupts.
- Initialize the Interrupt Mask Register, unmasking each bit that should generate the module interrupt.
- When an interrupt occurs, read the Pending Interrupt Status Register in the software interrupt-handler to determine the causes of the interrupt.
- Handle the interrupt, taking into account all causes of the interrupt.
- Clear the handled interrupt using Pending Interrupt Clear Register.

Masking interrupts: After reset, all interrupt bits are masked, since the Interrupt Mask Register is zero. To enable generation of a module interrupt for an interrupt bit, set the corresponding bit in the Interrupt Mask Register.

Clearing interrupts: All bits of the Pending Interrupt Register are cleared when it is read or when the Pending Interrupt Masked Register is read. Reading the Pending Interrupt Masked Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register. Selected bits can be cleared by writing ones to the bits that shall be cleared to the Pending Interrupt Clear Register.

Forcing interrupts: When the Pending Interrupt Register is written, the resulting value is the original contents of the register logically OR-ed with the write data. This means that writing the register can force (set) an interrupt bit, but never clear it.

Reading interrupt status: Reading the Pending Interrupt Status Register yields the same data as a read of the Pending Interrupt Register, but without clearing the contents.

Reading interrupt status of unmasked bits: Reading the Pending Interrupt Masked Status Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register, but without clearing the contents.

The interrupt registers comprise the following:

- Pending Interrupt Masked Status Register[FifoPIMSR]R
- Pending Interrupt Masked Register[FifoPIMR]R
- Pending Interrupt Status Register[FifoPISR]R
- Pending Interrupt Register[FifoPIR]R/W
- Interrupt Mask Register[FifoIMR]R/W
- Pending Interrupt Clear Register[FifoPICR]W

*Table 550.*Interrupt registers

31 7	6	5	4	3	2	1	0
-	RxParity	RxError	RxFull	RxIrq	TxError	TxEmpty	TxIrq

6:	RxParity	Parity error during reception
5:	RxError	AMBA AHB access error during reception
4:	RxFull	Circular reception buffer full
3:	RxIrq	Successful reception of block of data
2:	TxError	AMBA AHB access error during transmission
1:	TxEmpty	Circular transmission buffer empty
0:	TxIrq	Successful transmission of block of data

All bits in all interrupt registers are reset to 0b after reset.

44.8 Vendor and device identifiers

The module has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x035. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

44.9 Configuration options

Table 551 shows the configuration options of the core (VHDL generics).

*Table 551.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRFIFO.	0 - NAHBIRQ-1	0
dwidth	Data width	16	16
ptrwidth	Width of data pointers	16 - 16	16
singleirq	Single interrupt output. A single interrupt is assigned to the AMBA APB interrupt bus instead of multiple separate ones. The single interrupt output is controlled by the interrupt registers which are also enabled with this VHDL generic.	0, 1	0
oepol	Output enable polarity	0, 1	1

44.10 Signal descriptions

Table 552 shows the interface signals of the core (VHDL ports).

*Table 552.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-
FIFOI	DIN[31:0]	Input	Data input	-
	PIN[3:0]		Parity input	-
	EFN		Empty flag	Low
	FFN		Full flag	Low
	HFN		Half flag	Low
FIFOO	DOUT[31:0]	Output	Data output	-
	DEN[31:0]		Data output enable	-
	POUT[3:0]		Parity output	-
	PEN[3:0]		Parity output enable	-
	WEN		Write enable	Low
	REN		Read enable	Low

* see GRLIB IP Library User’s Manual

44.11 Library dependencies

Table 553 shows the libraries used when instantiating the core (VHDL libraries).

*Table 553.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AMBA signal definitions
GRLIB	AMBA	Signals, component	DMA2AHB definitions
GAISLER	MISC	Signals, component	Component declarations, signals.

44.12 Instantiation

This example shows how the core can be instantiated.

```
TBD
```

45 GRADCDAC - ADC / DAC Interface

45.1 Overview

The block diagram shows a possible partitioning of the combined analogue-to-digital converter (ADC) and digital-to-analogue converter (DAC) interface.

The combined analogue-to-digital converter (ADC) and digital-to-analogue converter (DAC) interface is assumed to operate in an AMBA bus system where an APB bus is present. The AMBA APB bus is used for data access, control and status handling.

The ADC/DAC interface provides a combined signal interface to parallel ADC and DAC devices. The two interfaces are merged both at the pin/pad level as well as at the interface towards the AMBA bus. The interface supports simultaneously one ADC device and one DAC device in parallel.

Address and data signals unused by the ADC and the DAC can be used for general purpose input output, providing 0, 8, 16 or 24 channels.

The ADC interface supports 8 and 16 bit data widths. It provides chip select, read/convert and ready signals. The timing is programmable. It also provides an 8-bit address output. The ADC conversion can be initiated either via the AMBA interface or by internal or external triggers. An interrupt is generated when a conversion is completed.

The DAC interface supports 8 and 16 bit data widths. It provides a write strobe signal. The timing is programmable. It also provides an 8-bit address output. The DAC conversion is initiated via the AMBA interface. An interrupt is generated when a conversion is completed.

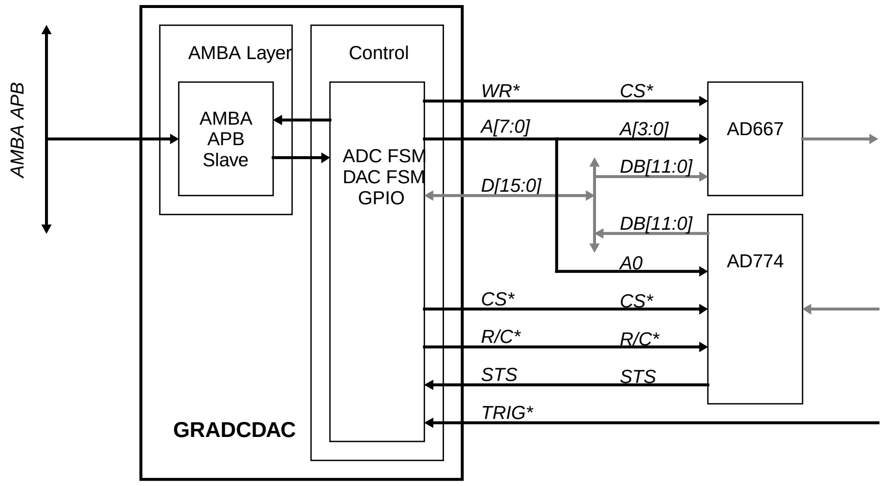


Figure 149. Block diagram and usage example

45.1.1 Function

The core implements the following functions:

- ADC interface conversion:
    - ready feed-back, or
    - timed open-loop
- DAC interface conversion:
    - timed open-loop
- General purpose input output:
    - unused data bus, and

- unused address bus
- Status and monitoring:
- on-going conversion
- completed conversion
- timed-out conversion

Note that it is not possible to perform ADC and DAC conversions simultaneously. On only one conversion can be performed at a time.

45.1.2 Interfaces

The core provides the following external and internal interfaces:

- Combined ADC/DAC interface
- programmable timing
- programmable signal polarity
- programmable conversion modes
- AMBA APB slave interface

The ADC interface is intended for amongst others the following devices:

Name:Width:Type:

AD574 12-bit R/C*, CE, CS*, RDY*, tri-state

AD674 12-bit R/C*, CE, CS*, RDY*, tri-state

AD774 12-bit R/C*, CE, CS*, RDY*, tri-state

AD670 8-bit R/W*, CE*, CS*, RDY, tri-state

AD571 10-bit Blank/Convert*, RDY*, tri-state

AD1671 12-bit Encode, RDY*, non-tri-state

LTC141414-bitConvert*, RDY, non-tri-state

The DAC interface is intended for amongst others the following devices:

Name:Width:Type:

AD56110-bitParallel-Data-in-Analogue-out

AD56512-bitParallel-Data-in-Analogue-out

AD66712-bitParallel-Data-in-Analogue-out, CS*

AD76712-bitParallel-Data-in-Analogue-out, CS*

DAC08 8-bit Parallel-Data-in-Analogue-out

45.2 Operation

45.2.1 Interfaces

The internal interface on the on-chip bus towards the core is a common AMBA APB slave for data access, configuration and status monitoring, used by both the ADC interface and the DAC interface.

The ADC address output and the DAC address output signals are shared on the external interface. The address signals are possible to use as general purpose input output channels. This is only realized when the address signals are not used by either ADC or DAC.

The ADC data input and the DAC data output signals are shared on the external interface. The data input and output signals are possible to use as general purpose input output channels. This is only realized when the data signals are not used by either ADC or DAC.

Each general purpose input output channel shall be individually programmed as input or output. This applies to both the address bus and the data bus. The default reset configuration for each general purpose input output channel is as input. The default reset value each general purpose input output channel is logical zero.

Note that protection toward spurious pulse commands during power up shall be mitigated as far as possible by means of I/O cell selection from the target technology.

45.2.2 Analogue to digital conversion

The ADC interface supports 8 and 16 bit wide input data.

The ADC interface provides an 8-bit address output, shared with the DAC interface. Note that the address timing is independent of the acquisition timing.

The ADC interface shall provide the following control signals:

- Chip Select
- Read/Convert
- Ready

The timing of the control signals is made up of two phases:

- Start Conversion
- Read Result

The Start Conversion phase is initiated by one of the following sources, provided that no other conversion is ongoing:

- Event on one of three separate trigger inputs
- Write access to the AMBA APB slave interface

Note that the trigger inputs can be connected to internal or external sources to the ASIC incorporating the core. Note that any of the trigger inputs can be connected to a timer to facilitate cyclic acquisition. The selection of the trigger source is programmable. The trigger inputs is programmable in terms of: Rising edge or Falling edge. Triggering events are ignored if ADC or DAC conversion is in progress.

The transition between the two phases is controlled by the Ready signal. The Ready input signal is programmable in terms of: Rising edge or Falling edge. The Ready input signaling is protected by means of a programmable time-out period, to assure that every conversion terminates. It is also possible to perform an ADC conversion without the use of the Ready signal, by means of a programmable conversion time duration. This can be seen as an open-loop conversion.

The Chip Select output signal is programmable in terms of:

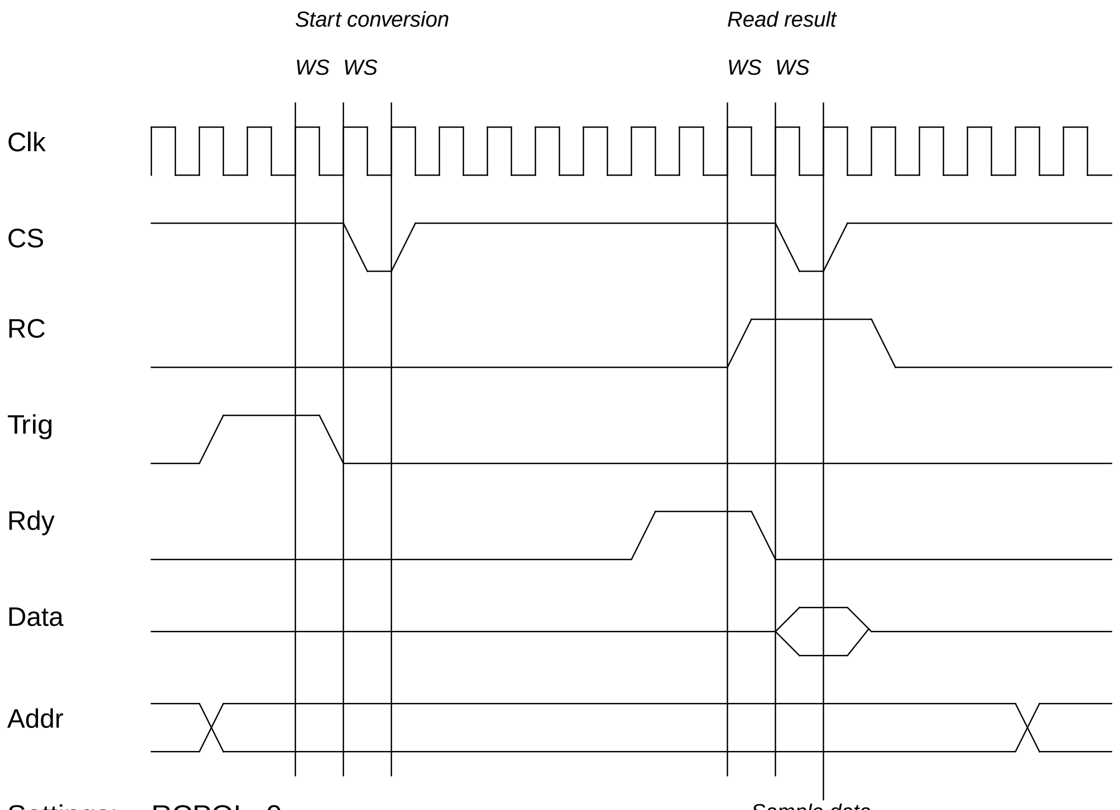
- Polarity
- Number of assertions during a conversion, either
- Pulsed once during Start Conversion phase only,
- Pulsed once during Start Conversion phase and once during Read Result phase, or
- Asserted at the beginning of the Start Conversion phase and de-asserted at the end of the Read Result phase

The duration of the asserted period is programmable, in terms of system clock periods, for the Chip Select signal when pulsed in either of two phases.

The Read/Convert signal is de-asserted during the Start Conversion phase, and asserted during the Read Result phase. The Read/Convert output signal is programmable in terms of: Polarity. The setup timing from Read/Convert signal being asserted till the Chip Select signal is asserted is programmable, in terms of system clock periods. Note that the programming of Chip Select and Read/Convert timing is implemented as a common parameter.

At the end of the Read Result phase, an interrupt is generated, indicating that data is ready for readout via the AMBA APB slave interface. The status of an on-going conversion is possible to read out via the AMBA APB slave interface. The result of a conversion is read out via the AMBA APB slave interface. Note that this is independent of what trigger started the conversion.

An ADC conversion is non-interruptible. It is possible to perform at least 1000 conversions per second.



Settings: RCPOL=0
CSPOL=0
RDYPOL=1
TRIGPOL=1
RDYMODE=1
CSMODE=00
ADCWS=0

Figure 150. Analogue to digital conversion waveform, 0 wait states (WS)

45.2.3 Digital to analogue conversion

The DAC interface supports 8 and 16 bit wide output data. The data output signal is driven during the conversion and is placed in high impedance state after the conversion.

The DAC interface provides an 8-bit address output, shared with the ADC interface. Note that the address timing is independent of the acquisition timing.

The DAC interface provides the following control signal: Write Strobe. Note that the Write Strobe signal can also be used as a chip select signal. The Write Strobe output signal is programmable in terms of: Polarity. The Write Strobe signal is asserted during the conversion. The duration of the asserted period of the Write Strobe is programmable in terms of system clock periods.

At the end the conversion, an interrupt is generated. The status of an on-going conversion is possible to read out via the AMBA APB slave interface. A DAC conversion is non-interruptible.

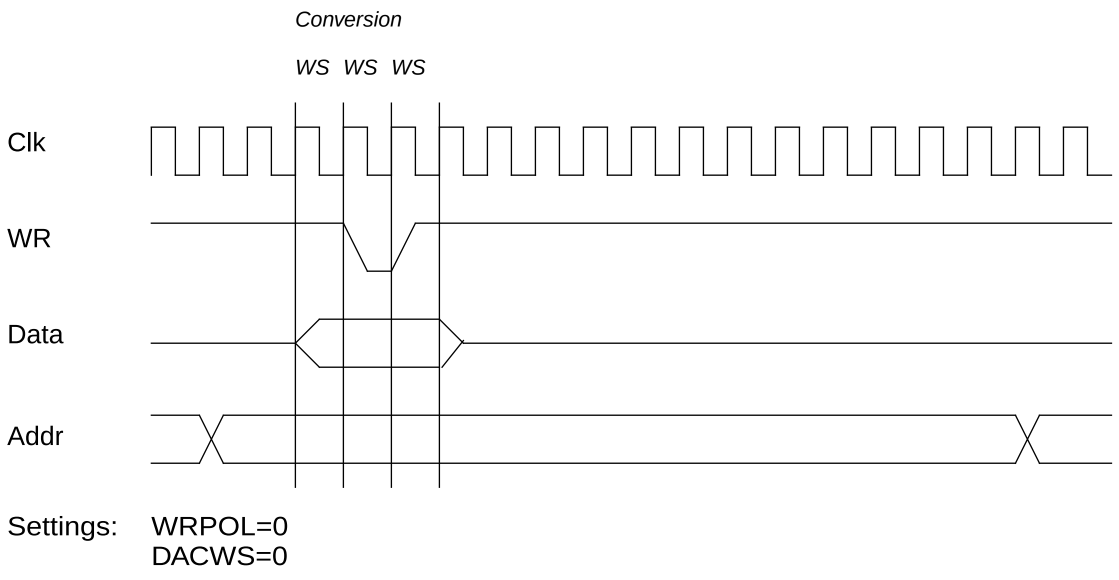


Figure 151. Digital to analogue conversion waveform, 0 wait states (WS)

45.3 Operation

45.3.1 Interrupt

Two interrupts are implemented by the ADC/DAC interface:

Index:	Name:	Description:
0	ADC	ADC conversion ready
1	DAC	DAC conversion ready

The interrupts are configured by means of the *pirq* VHDL generic.

45.3.2 Reset

After a reset the values of the output signals are as follows:

Signal:	Value after reset:
ADO.Aout[7:0]	de-asserted
ADO.Aen[7:0]	de-asserted
ADO.Dout[15:0]	de-asserted
ADO.Den[15:0]	de-asserted
ADO.WR	de-asserted (logical one)
ADO.CS	de-asserted (logical one)
ADO.RC	de-asserted (logical one)

45.3.3 Asynchronous interfaces

The following input signals are synchronized to Clk:

- ADI.Ain[7:0]
- ADI.Din[15:0]
- ADI.RDY
- ADI.TRIG[2:0]

45.4 Registers

The core is programmed through registers mapped into APB address space.

*Table 554.*GRADCDAC registers

APB address offset	**Register**
16#000#	Configuration Register
16#004#	Status Register
16#010#	ADC Data Input Register
16#014#	DAC Data Output Register
16#020#	Address Input Register
16#024#	Address Output Register
16#028#	Address Direction Register
16#030#	Data Input Register
16#034#	Data Output Register
16#038#	Data Direction Register

45.4.1 Configuration Register [ADCONF] R/W

*Table 555.*Configuration register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								DACWS					WR POL	DACDW	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCWS					RCP OL	CSMODE		CSP OL	RD YM OD E	RD YP OL	TRI GP OL	TRIG-MODE		ADCDW	

23-19:	DACWS	Number of DAC wait states, 0 to 31 [5 bits]
18:	WRPOL	Polarity of DAC write strobe:
		0b = active low
		1b = active high
17-16:	DACDW	DAC data width
		00b = none
		01b = 8 bit ADO.Dout[7:0]
		10b = 16 bit ADO.Dout[15:0]
		11b = none/spare
15-11:	ADCWS	Number of ADC wait states, 0 to 31 [5 bits]
10:	RCPOL	Polarity of ADC read convert:
		0b = active low read
		1b = active high read
9-8:	CSMODE	Mode of ADC chip select:
		00b = asserted during conversion and read phases

		01b = asserted during conversion phase
		10b = asserted during read phase
		11b = asserted continuously during both phases
7:	CSPOL	Polarity of ADC chip select:0b = active low
		1b = active high
6:	RDYMODE:	Mode of ADC ready:
		0b = unused, i.e. open-loop
		1b = used, with time-out
5:	RDYPOL	Polarity of ADC ready:
		0b = falling edge
		1b = rising edge
4:	TRIGPOL	Polarity of ADC triggers:
		0b = falling edge
		1b = rising edge
3-2:	TRIGMODE	ADC trigger source:
		00b = none
		01b = ADI.TRIG[0]
		10b = ADI.TRIG[1]
		11b = ADI.TRIG[2]
1-0:	ADCDW	ADC data width:
		00b = none
		01b = 8 bit ADI.Din[7:0]
		10b = 16 bit ADI.Din[15:0]
		11b = none/spare

The ADCDW field defines what part of ADI.Din[15:0] is read by the ADC.

The DACDW field defines what part of ADO.Dout[15:0] is written by the DAC.

Parts of the data input/output signals used neither by ADC nor by DAC are available for the general purpose input output functionality.

Note that an ADC conversion can be initiated by means of a write access via the AMBA APB slave interface, thus not requiring an explicit ADC trigger source setting.

The ADCONF.ADCWS field defines the number of system clock periods, ranging from 1 to 32, for the following timing relationships between the ADC control signals:

- ADO.RC stable before ADO.CS period
- ADO.CS asserted period, when pulsed
- ADO.TRIG[2:0] event until ADO.CS asserted period
- Time-out period for ADO.RDY: 2048 * (1+ADCONF.ADCWS)
- Open-loop conversion timing: 512 * (1+ADCONF.ADCWS)

The ADCONF.DACWS field defines the number of system clock periods, ranging from 1 to 32, for the following timing relationships between the DAC control signals:

- ADO.Dout[15:0] stable before ADO.WR period
- ADO.WR asserted period
- ADO.Dout[15:0] stable after ADO.WR period

45.4.2 Status Register [ADSTAT] R

*Table 556.*Status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
									DACNO	DACRDY	DACON	ADCTO	ADCNO	ADCRDY	ADCON

6:	DACNO	DAC conversion request rejected (due to ongoing DAC or ADC conversion)
5:	DACRDY	DAC conversion completed
4:	DACON	DAC conversion ongoing
3:	ADCTO	ADC conversion timeout
2:	ADCNO	ADC conversion request rejected (due to ongoing ADC or DAC conversion)
1:	ADCRDY	ADC conversion completed
0:	ADCON	ADC conversion ongoing

When the register is read, the following sticky bit fields are cleared:

- DACNO, DACRDY,
- ADCTO, ADCNO, and ADCRDY.

Note that the status bits can be used for monitoring the progress of a conversion or to ascertain that the interface is free for usage.

45.4.3 ADC Data Input Register [ADIN] R/W

*Table 557.*ADC Data Input Register

31-16	15-0
	ADCIN

15-0:	ADCIN	ADC input data	*ADI.Din[15:0]*

A write access to the register initiates an analogue to digital conversion, provided that no other ADC or DAC conversion is ongoing (otherwise the request is rejected).

A read access that occurs before an ADC conversion has been completed returns the result from a previous conversion.

Note that any data can be written and that it cannot be read back, since not relevant to the initiation of the conversion.

Note that only the part of ADI.Din[15:0] that is specified by means of bit ADCONF.ADCDW is used by the ADC. The rest of the bits are read as zeros.

Note that only bits dwidth-1 to 0 are implemented.

45.4.4 DAC Data Output Register [ADOUT] R/W

*Table 558.*DAC Data Output Register

31-16	15-0
	DACOUT

15-0:	DACOUT	DAC output data	*ADO.Dout[15:0]*

A write access to the register initiates a digital to analogue conversion, provided that no other DAC or ADC conversion is ongoing (otherwise the request is rejected).

Note that only the part of ADO.Dout[15:0] that is specified by means of ADCONF.DACDW is driven by the DAC. The rest of the bits are not driven by the DAC during a conversion.

Note that only the part of ADO.Dout[15:0] which is specified by means of ADCONF.DACDW can be read back, whilst the rest of the bits are read as zeros.

Note that only bits dwidth-1 to 0 are implemented.

45.4.5 Address Input Register [ADAIN] R

*Table 559.*Address Input Register

31 8	7 0
	AIN

7-0:	AIN	Input address	*ADI.Ain[7:0]*

All bits are cleared to 0 at reset.

Note that only bits awidth-1 to 0 are implemented.

45.4.6 Address Output Register [ADAOUT] R/W

*Table 560.*Address Output Register

31 8	7 0
	AOUT

7-0:	AOUT	Output address	*ADO.Aout[7:0]*

All bits are cleared to 0 at reset.

Note that only bits awidth-1 to 0 are implemented.

45.4.7 Address Direction Register [ADADIR] R/W

*Table 561.*Address Direction Register

31 8	7 0
	ADIR

7-0:	ADIR	Direction: 0b = input = high impedance, 1b = output = driven	*ADO.Aout[7:0]*

All bits are cleared to 0 at reset.

Note that only bits awidth-1 to 0 are implemented.

45.4.8 Data Input Register [ADDIN] R

*Table 562.*Data Input Register

31 16	15 0
	DIN

15-0:	DIN	Input data	*ADI.Din[15:0]*

All bits are cleared to 0 at reset.

Note that only the part of ADI.Din[15:0] not used by the ADC can be used as general purpose input output, see ADCONF.ADCDW.

Note that only bits dwidth-1 to 0 are implemented.

45.4.9 Data Output Register [ADDOUT] R/W

*Table 563.*Data Output Register

31 16	15 0
	DOUT

15-0:	DOUT	Output data	*ADO.Dout[15:0]*

All bits are cleared to 0 at reset.

Note that only the part of ADO.Dout[15:0] neither used by the DAC nor the ADC can be used as general purpose input output, see ADCONF.DACDW and ADCONF. ADCDW.

Note that only bits dwidth-1 to 0 are implemented.

45.4.10 Data Register [ADDDIR] R/W

*Table 564.*Data Direction Register

31 16	15 0
	DDIR

15-0:	DDIR	Direction: 0b = input = high impedance, 1b = output = driven	*ADO.Dout[15:0]*

All bits are cleared to 0 at reset.

Note that only the part of ADO.Dout[15:0] not used by the DAC can be used as general purpose input output, see ADCONF.DACDW.

Note that only bits dwidth-1 to 0 are implemented.

45.5 Vendor and device identifiers

The module has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x036. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

45.6 Configuration options

Table 565 shows the configuration options of the core (VHDL generics).

*Table 565.*Configuration options

Generic	**Function**	**Allowed range**	**Default**
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRADCDAC.	0 - NAHBIRQ-1	1
nchannel	Number of input/outputs	1 - 32	24
npulse	Number of pulses	1 - 32	8
invertpulse	Invert pulse output when set	1 - 32	0
cntrwidth	Pulse counter width	4 to 32	20
oepol	Output enable polarity	0, 1	1

45.7 Signal descriptions

Table 566 shows the interface signals of the core (VHDL ports).

*Table 566.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
ADI	ADI.Ain[7:0]	Input	Common Address input	-
	ADI.Din[15:0]		ADC Data input	
	ADI.RDY		ADC Ready input	
	ADI.TRIG[2:0]		ADC Trigger inputs	
ADO	ADO.Aout[7:0]	Output	Common Address output	-
	ADO.Aen[7:0]		Common Address output enable	
	ADO.Dout[15:0]		DAC Data output	-
	ADO.Den[15:0]		DAC Data output enable	
	ADO.WRDAC		Write Strobe	
	ADO.CSADC		Chip Select	
	ADO.RCADC		Read/Convert	

* see GRLIB IP Library User’s Manual

Note that the VHDL generic oepol is used for configuring the logical level of ADO.Den and ADO.Aen while asserted.

45.8 Signal definitions and reset values

The signals and their reset values are described in table 567.

*Table 567.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
a[]	Input/Output	Address	High	Tri-state
d[]	Input/Output	Data	High	Tri-state
wr	Output	DAC Write Strobe	-	Logical 0
cs	Output	ADC Chip Select	-	Logical 0
rc	Output	ADC Read/Convert	-	Logical 0
rdy	Input	ADC Ready	-	-
trig[]	Input	ADC Trigger	-	-

45.9 Timing

The timing waveforms and timing parameters are shown in figure 152 and are defined in table 568. Note that the input and output polarities of control and response signals are programmable. The figures shows operation where there are zero wait states. Note also that the address timing has no direct correlation with the ADC and DAC accesses, since controlled by a separate set of registers.

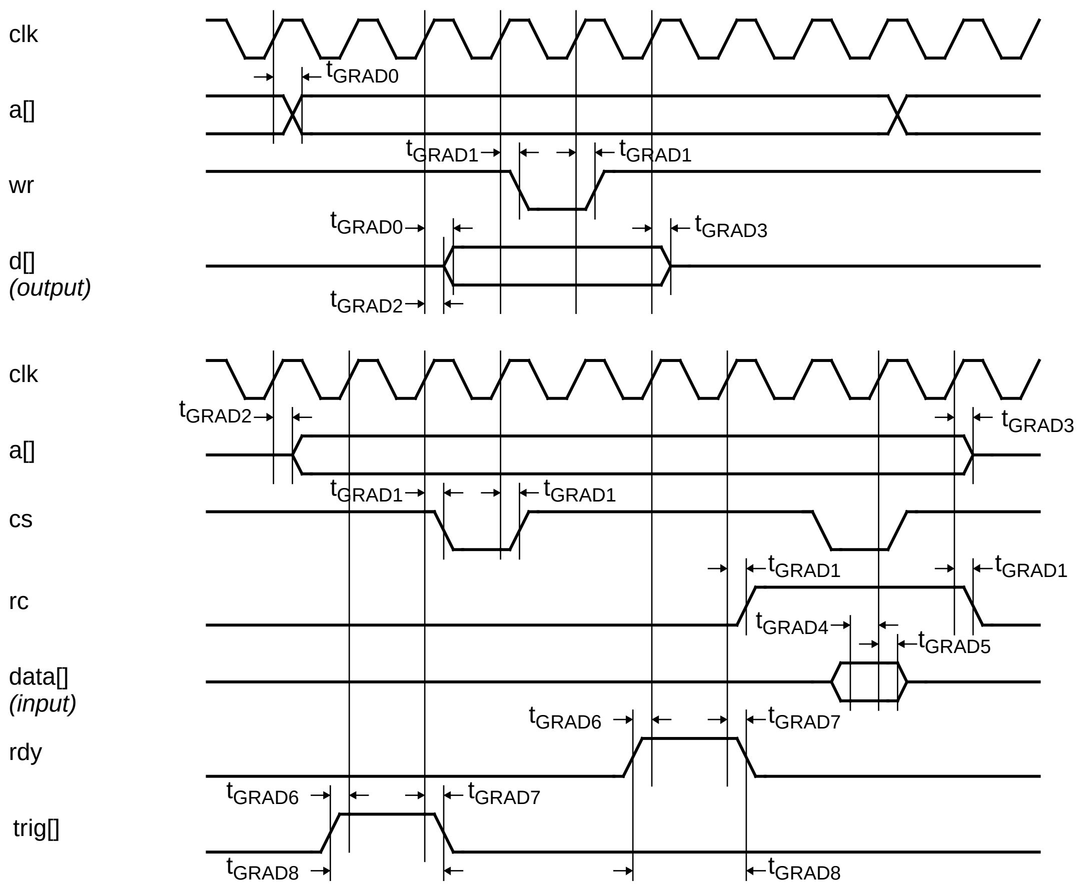


Figure 152. Timing waveforms

*Table 568.*Timing parameters

Name	**Parameter**	**Reference edge**	**Min**	**Max**	**Unit**
t_{GRAD0}	a/d clock to output delay	rising *clk* edge	-	TBD	ns
t_{GRAD1}	clock to output delay	rising *clk* edge	-	TBD	ns
t_{GRAD2}	clock to a/d non-tri-state delay	rising *clk* edge	TBD	-	ns
t_{GRAD3}	a/d clock to data tri-state delay	rising *clk* edge	-	TBD	ns
t_{GRAD4}	a/d input to clock setup	rising *clk* edge	TBD	-	ns
t_{GRAD5}	a/d input from clock hold	rising *clk* edge	TBD	-	ns
t_{GRAD6}	input to clock setup	rising *clk* edge	-	TBD	ns
t_{GRAD7}	input from clock hold	rising *clk* edge	TBD	-	ns
t_{GRAD8}	input assertion duration	-	TBD	-	*clk* periods

45.10 Library dependencies

Table 569 shows the libraries used when instantiating the core (VHDL libraries).

*Table 569.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Signals	GRADCDAC component declaration

45.11 Instantiation

This example shows how the core can be instantiated.

```
TBD
```

46 GRFPU - High-performance IEEE-754 Floating-point unit

46.1 Overview

GRFPU is a high-performance FPU implementing floating-point operations as defined in the IEEE Standard for Binary Floating-Point Arithmetic (IEEE-754) and the SPARC V8 standard (IEEE-1754). Supported formats are single and double precision floating-point numbers. The advanced design combines two execution units, a fully pipelined unit for execution of the most common FP operations and a non-blocking unit for execution of divide and square-root operations.

The logical view of the GRFPU is shown in figure 153.

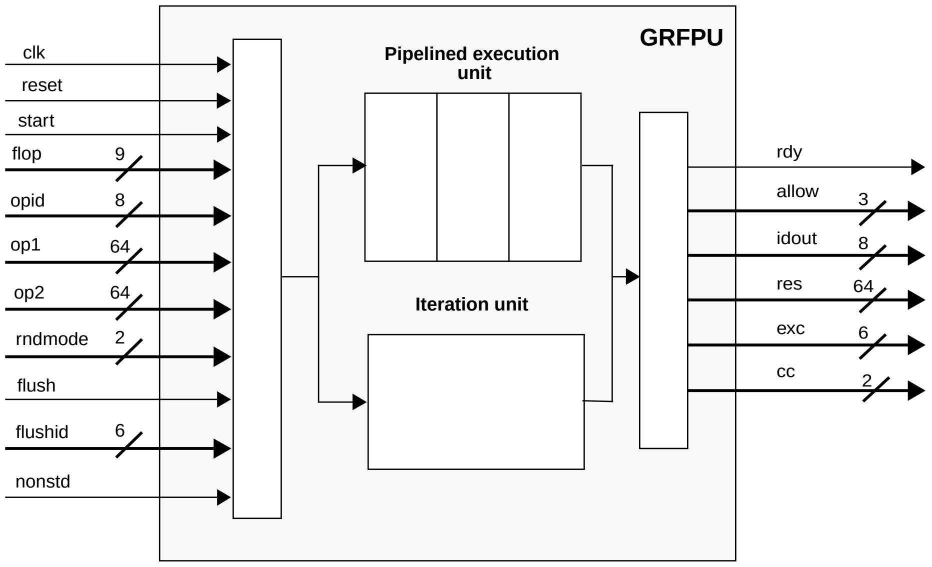


Figure 153. GRFPU Logical View

This document describes GRFPU from functional point of view. Chapter “Functional description” gives details about GRFPU’s implementation of the IEEE-754 standard including FP formats, operations, opcodes, operation timing, rounding and exceptions. “Signals and timing” describes the GRFPU interface and its signals. “GRFPU Control Unit” describes the software aspects of the GRFPU integration into a LEON processor through the GRFPU Control Unit - GRFPC. For implementation details refer to the white paper, “GRFPU - High Performance IEEE-754 Floating-Point Unit” (available at *www.gaisler.com*).

46.2 Functional description

46.2.1 Floating-point number formats

GRFPU handles floating-point numbers in single or double precision format as defined in the IEEE-754 standard with exception for denormalized numbers. See section 46.2.5 for more information on denormalized numbers.

46.2.2 FP operations

GRFPU supports four types of floating-point operations: arithmetic, compare, convert and move. The operations implement all FP instructions specified by SPARC V8 instruction set, and most of the operations defined in IEEE-754. All operations are summarized in table 570, with their opcodes, operands, results and exception codes. Throughputs and latencies and are shown in table 570.

Table 570.: GRFPU operations

Operation	**OpCode[8:0]**	**Op1**	**Op2**	**Result**	**Exceptions**	**Description**
Arithmetic operations						
FADDS FADDD	001000001 001000010	SP DP	SP DP	SP DP	UNF, NV, OF, UF, NX	Addition
FSUBS FSUBD	001000101 001000110	SP DP	SP DP	SP DP	UNF, NV, OF, UF, NX	Subtraction
FMULS FMULD FSMULD	001001001 001001010 001101001	SP DP SP	SP DP SP	SP DP DP	UNF, NV, OF, UF, NX UNF, NV, OF, UF, NX UNF, NV, OF, UF	Multiplication, FSMULD gives exact double-precision product of two single-precision operands.
FDIVS FDIVD	001001101 001001110	SP DP	SP DP	SP DP	UNF, NV, OF, UF, NX, DZ	Division
FSQRTS FSQRTD	000101001 000101010	- -	SP DP	SP DP	UNF, NV, NX	Square-root
Conversion operations						
FITOS FITOD	011000100 011001000	-	INT	SP DP	NX -	Integer to floating-point conversion
FSTOI FDTOI	011010001 011010010	-	SP DP	INT	UNF, NV, NX	Floating-point to integer conversion. The result is rounded in round-to-zero mode.
FSTOI_RND FDTOI_RND	111010001 111010010	-	SP DP	INT	UNF, NV, NX	Floating-point to integer conversion. Rounding according to RND input.
FSTOD FDTOS	011001001 011000110	-	SP DP	DP SP	UNF, NV UNF, NV, OF, UF, NX	Conversion between floating-point formats
Comparison operations						
FCMPS FCMPD	001010001 001010010	SP DP	SP DP	CC	NV	Floating-point compare. Invalid exception is generated if either operand is a signaling NaN.
FCMPES FCMPED	001010101 001010110	SP DP	SP DP	CC	NV	Floating point compare. Invalid exception is generated if either operand is a NaN (quiet or signaling).
Negate, Absolute value and Move						
FABSS	000001001	-	SP	SP	-	Absolute value.
FNEGS	000000101	-	SP	SP	-	Negate.
FMOVS	000000001		SP	SP	-	Move. Copies operand to result output.

SP - single precision floating-point number	CC - condition codes, see table 573
DP - double precision floating-point number	UNF, NV, OF, UF, NX - floating-point exceptions, see section 46.2.3
INT - 32 bit integer	

Arithmetic operations include addition, subtraction, multiplication, division and square-root. Each arithmetic operation can be performed in single or double precision formats. Arithmetic operations have one clock cycle throughput and a latency of four clock cycles, except for divide and square-root operations, which have a throughput of 16 - 25 clock cycles and latency of 16 - 25 clock cycles (see table 571). Add, sub and multiply can be started on every clock cycle, providing high throughput for these common operations. Divide and square-root operations have lower throughput and higher latency due to complexity of the algorithms, but are executed in parallel with all other FP operations in a non-blocking iteration unit. Out-of-order execution of operations with different latencies is easily handled through the GRFPU interface by assigning an id to every operation which appears with the result on the output once the operation is completed (see section 46.4).

Table 571.: Throughput and latency

Operation	Throughput	Latency
FADDS, FADDD, FSUBS, FSUBD, FMULS, FMULD, FSMULD	1	4
FITOS, FITOD, FSTOI, FSTOI_RND, FDTOI, FDTOI_RND, FSTOD, FDTOS	1	4
FCMPS, FCMPD, FCMPES, FCMPED	1	4
FDIVS	16	16
FDIVD	16.5 (15/18)*	16.5 (15/18)*
FSQRTS	24	24
FSQRTD	24.5 (23/26)*	24.5 (23/26)*

* Throughput and latency are data dependant with two possible cases with equal statistical possibility.

Conversion operations execute in a pipelined execution unit and have throughput of one clock cycle and latency of four clock cycles. Conversion operations provide conversion between different floating-point numbers and between floating-point numbers and integers.

Comparison functions offering two different types of quiet Not-a-Numbers (QNaNs) handling are provided. Move, negate and absolute value are also provided. These operations do not ever generate unfinished exception (unfinished exception is never signaled since compare, negate, absolute value and move handle denormalized numbers).

46.2.3 Exceptions

GRFPU detects all exceptions defined by the IEEE-754 standard. This includes detection of Invalid Operation (NV), Overflow (OF), Underflow (UF), Division-by-Zero (DZ) and Inexact (NX) exception conditions. Generation of special results such as NaNs and infinity is also implemented. Overflow (OF) and underflow (UF) are detected before rounding. If an operation underflows the result is flushed to zero (GRFPU does not support denormalized numbers or gradual underflow). A special Unfinished exception (UNF) is signaled when one of the operands is a denormalized number which is not handled by the arithmetic and conversion operations.

46.2.4 Rounding

All four rounding modes defined in the IEEE-754 standard are supported: round-to-nearest, round-to-+inf, round-to--inf and round-to-zero.

46.2.5 Denormalized numbers

Denormalized numbers are not handled by the GRFPU arithmetic and conversion operations. A system (microprocessor) with the GRFPU could emulate rare cases of operations on denormals in software using non-FPU operations. A special Unfinished exception (UNF) is used to signal an arithmetic or conversion operation on the denormalized numbers. Compare, move, negate and absolute value operations can handle denormalized numbers and do not raise the unfinished exception. GRFPU does not generate any denormalized numbers during arithmetic and conversion operations on normalized numbers. If the infinitely precise result of an operation is a tiny number (smaller than minimum value representable in normal format) the result is flushed to zero (with underflow and inexact flags set).

46.2.6 Non-standard Mode

GRFPU can operate in a non-standard mode where all denormalized operands to arithmetic and conversion operations are treated as (correctly signed) zeroes. Calculations are performed on zero operands instead of the denormalized numbers obeying all rules of the floating-point arithmetics including rounding of the results and detecting exceptions.

46.2.7 NaNs

GRFPU supports handling of Not-a-Numbers (NaNs) as defined in the IEEE-754 standard. Operations on signaling NaNs (SNaNs) and invalid operations (e.g. inf/inf) generate the Invalid exception and deliver QNaN_GEN as result. Operations on Quiet NaNs (QNaNs), except for FCMPES and FCMPED, do not raise any exceptions and propagate QNaNs through the FP operations by delivering NaN-results according to table 572. QNaN_GEN is 0x7fffe00000000000 for double precision results and 0x7fff0000 for single precision results.

Table 572.: Operations on NaNs

	Operand 2			
		FP	QNaN2	SNaN2
Operand 1	none	FP	QNaN2	QNaN_GEN
	FP	FP	QNaN2	QNaN_GEN
	QNaN1	QNaN1	QNaN2	QNaN_GEN
	SNaN1	QNaN_GEN	QNaN_GEN	QNaN_GEN

46.3 Signal descriptions

Table 573 shows the interface signals of the core (VHDL ports). All signals are active high except for RST which is active low.

Table 573.: Signal descriptions

Signal	I/O	Description
CLK	I	Clock
RESET	I	Reset
START	I	Start an FP operation on the next rising clock edge
NONSTD	I	Nonstandard mode. Denormalized operands are converted to zero.
FLOP[8:0]	I	FP operation. For codes see table 570.
OP1[63:0] OP2[63:0]	I	FP operation operands are provided on these one or both of these inputs. All 64 bits are used for IEEE-754 double precision floating-point numbers, bits [63:32] are used for IEEE-754 single precision floating-point numbers and 32-bit integers.
OPID[7:0]	I	FP operation id. Every operation is associated with an id which will appear on the RESID output when the FP operation is completed. This value shall be incremented by 1 (with wrap-around) for every started FP operation. If flushing is used, FP operation id is 6 -bits wide (OPID[5:0] are used for id, OPID[7:6] are tied to "00"). If flushing is not used (input signal FLUSH is tied to '0'), all 8-bits (OPID[7:0]) are used.
FLUSH	I	Flush FP operation with FLUSHID.
FLUSHID[5:0]	I	Id of the FP operation to be flushed.
RNDMODE[1:0]	I	Rounding mode. 00 - rounding-to-nearest, 01 - round-to-zero, 10 - round-to-+inf, 11 - round-to--inf.
RES[63:0]	O	Result of an FP operation. If the result is double precision floating-point number all 64 bits are used, otherwise single precision or integer result appears on RESULT[63:32].
EXC[5:0]	O	Floating-point exceptions generated by an FP operation. EXC[5] - Unfinished FP operation. Generated by an arithmetic or conversion operation with denormalized input(s). EXC[4] - Invalid exception. EXC[3] - Overflow. EXC[2] - Underflow. EXC[1] - Division by zero. EXC[0] - Inexact.
ALLOW[2:0]	O	Indicates allowed FP operations during the next clock cycle. ALLOW[0] - FDIVS, FDIVD, FSQRTS and FSQRTD allowed ALLOW[1] - FMULS, FMULD, FSMULD allowed ALLOW[2] - all other FP operations allowed
RDY	O	The result of a FP operation will be available at the end of the next clock cycle.
CC[1:0]	O	Result (condition code) of an FP compare operation. 00 - equal 01 - operand1 < operand2 10 - operand1 > operand2 11 - unordered
IDOUT[7:0]	O	Id of the FP operation whose result appears at the end of the next clock cycle.

46.4 Timing

An FP operation is started by providing the operands, opcode, rounding mode and id before rising edge. The operands need to be provided a small set-up time before a rising edge while all other signals are latched on rising edge.

The FPU is fully pipelined and a new operation can be started every clock cycle. The only exceptions are divide and square-root operations which require 16 to 26 clock cycles to complete, and which are not pipelined. Division and square-root are implemented through iterative series expansion algorithm. Since the algorithms basic step is multiplication the floating-point multiplier is shared between multiplication, division and square-root. Division and square-root do not occupy the multiplier during the whole operation and allow multiplication to be interleaved and executed parallelly with division or square-root.

One clock cycle before an operation is completed, the output signal RDY is asserted to indicate that the result of an FPU operation will appear on the output signals at the end of the next cycle. The id of the operation to be completed and allowed operations are reported on signals RESID and ALLOW. During the next clock cycle the result appears on RES, EXCEPT and CC outputs.

Figure 154 shows signal timing during four arithmetic operations on GRFPU.

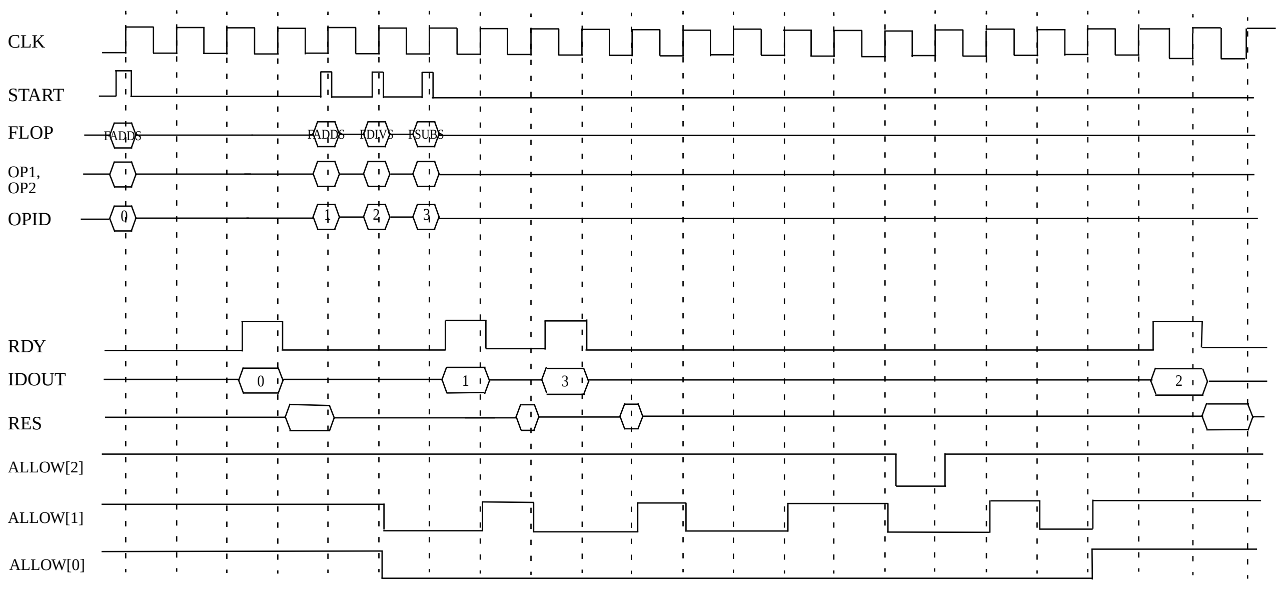


Figure 154. Signal timing

46.5 Shared FPU

46.5.1 Overview

In multi-processor systems, a single GRFPU can be shared between multiple CPU cores providing an area efficient solution. In this configuration, the GRFPU is extended with a wrapper. Each CPU core issues a request to execute a specific FP operation to the wrapper, which performs fair arbitration using the round-robin algorithm. When a CPU core has started a divide or square-root operation, the FPU is not able to accept a new division or square-root until the current operation has finished. Also, during the execution of a division or square-root, other operations cannot be accepted during certain cycles. This can lead to the, currently, highest prioritized CPU core being prevented from issuing an operation to the FPU. If this happens, the next CPU core that has a operation that can be started will be allowed to access the FPU and the current arbitration order will be saved. The arbitration order will be restored when the operation type that was prevented can be started. This allows the FPU resource the be fairly shared between several CPU cores while at the same time allowing maximum utilization of the FPU.

In shared FPU configuration, GRFPU uses an 8 bit wide id for each operation. The three high-order bits are used to identify the CPU core which issued the FP operation, while the five low-order bits are used to enumerate FP operations issued by one core. FP operation flushing is not possible in shared FPU configuration.

46.5.2 Shared FPU and clock gating

Clock gating of LEON processors is typically implemented so that the clock for a processor core is gated off when the processor is idle. The clock for a shared FPU is typically gated off when the connected processors are all idle or have floating-point disabled.

This means that, in a shared FPU configuration, a processor may be clock gated off while the connected FPU continues to be clocked. The power-down instruction may overtake a previously issued floating-point instruction and cause the processor to be gated off before the floating-point operation has completed. This can in turn lead to the processor not reacting to the completion of the floating-point operation and to a subsequent processor freeze after the processor wakes up and continues to wait for the completion of the floating-point operation.

In order to avoid this, software must make sure that all floating-point operations have completed before the processor enters power-down. This is generally not a problem in real-world applications as the power-down instruction is typically used in a idle loop and floating-point results have been stored to memory before entering the idle loop. To make sure that there are no floating-point operations pending, software should perform a store of the %fsr register before the power-down instruction.

47 GRFPC - GRFPU Control Unit

The GRFPU Control Unit (GRFPC) is used to attach the GRFPU to the LEON integer unit (IU). GRFPC performs scheduling, decoding and dispatching of the FP operations to the GRFPU as well as managing the floating-point register file, the floating-point state register (FSR) and the floating-point deferred-trap queue (FQ). Floating-point operations are executed in parallel with other integer instructions, the LEON integer pipeline is only stalled in case of operand or resource conflicts.

In the FT-version, all registers are protected with TMR and the floating-point register file is protected using parity coding.

47.1 Floating-Point register file

The GRFPU floating-point register file contains 32 32-bit floating-point registers (%f0-%f31). The register file is accessed by floating-point load and store instructions (LDF, LDDF, STD, STDF) and floating-point operate instructions (FPop).

47.2 Floating-Point State Register (FSR)

The GRFPC manages the floating-point state register (FSR) containing FPU mode and status information. All fields of the FSR register as defined in SPARC V8 specification are implemented and managed by the GRFPU conforming to the SPARC V8 specification and the IEEE-754 standard. Implementation-specific parts of the FSR managing are the NS (non-standard) bit and *ftt* field.

If the NS (non-standard) bit of the FSR register is set, all floating-point operations will be performed in non-standard mode as described in section 46.2.6. When the NS bit is cleared all operations are performed in standard IEEE-compliant mode.

Following floating-point trap types never occur and are therefore never set in the ftt field:
- unimplemented_FPop: all FPop operations are implemented
- hardware_error: non-resumable hardware error
- invalid_fp_register: no check that double-precision register is 0 mod 2 is performed

GRFPU implements the *qne* bit of the FSR register which reads 0 if the floating-point deferred-queue (FQ) is empty and 1 otherwise.

The FSR is accessed using LDFSR and STFSR instructions.

47.3 Floating-Point Exceptions and Floating-Point Deferred-Queue

GRFPU implements the SPARC deferred trap model for floating-point exceptions (fp_exception). A floating-point exception is caused by a floating-point instruction performing an operation resulting in one of following conditions:

- an operation raises IEEE floating-point exception (ftt = IEEE_754_exception) e.g. executing invalid operation such as 0/0 while the NVM bit of the TEM field id set (invalid exception enabled).
- an operation on denormalized floating-point numbers (in standard IEEE-mode) raises unfinished_FPop floating-point exception
- sequence error: abnormal error condition in the FPU due to the erroneous use of the floating-point instructions in the supervisor software.

The trap is deferred to one of the floating-point instructions (FPop, FP load/store, FP branch) following the trap-inducing instruction (note that this may not be next floating-point instruction in the program order due to exception-detecting mechanism and out-of-order instruction execution in the GRFPC). When the trap is taken the floating-point deferred-queue (FQ) contains the trap-inducing instruction and up to seven FPop instructions that were dispatched in the GRFPC but did not complete.

After the trap is taken the *qne* bit of the FSR is set and remains set until the FQ is emptied. The STDFQ instruction reads a double-word from the floating-point deferred queue, the first word is the address of the instruction and the second word is the instruction code. All instructions in the FQ are FPop type instructions. The first access to the FQ gives a double-word with the trap-inducing instruction, following double-words contain pending floating-point instructions. Supervisor software should emulate FPops from the FQ in the same order as they were read from the FQ.

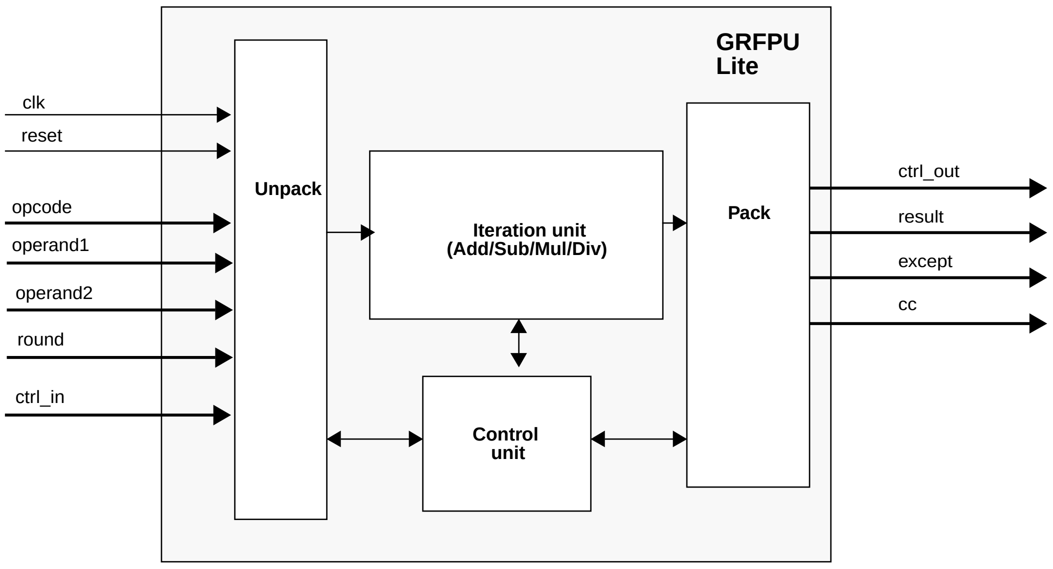
Note that instructions in the FQ may not appear in the same order as the program order since GRFPU executes floating-point instructions out-of-order. A floating-point trap is never deferred past an instruction specifying source registers, destination registers or condition codes that could be modified by the trap-inducing instruction. Execution or emulation of instructions in the FQ by the supervisor software gives therefore the same FPU state as if the instructions were executed in the program order.

48 GRFPU Lite - IEEE-754 Floating-Point Unit

48.1 Overview

The GRFPU Lite floating-point unit implements floating-point operations as defined in IEEE Standard for Binary Floating-Point Arithmetic (IEEE-754) and SPARC V8 standard (IEEE-1754).

Supported formats are single and double precision floating-point numbers. The floating-point unit is not pipelined and executes one floating-point operation at a time.



48.2 Functional Description

48.2.1 Floating-point number formats

The floating-point unit handles floating-point numbers in single or double precision format as defined in IEEE-754 standard.

48.2.2 FP operations

The floating-point unit supports four types of floating-point operations: arithmetic, compare, convert and move. The operations, summarised in the table below, implement all FP instructions specified by the SPARC V8 instruction set except FSMULD and instructions with quadruple precision.

Table 574.:Floating-point operations

Operation	Op1	Op2	Result	Exceptions	Description
Arithmetic operations					
FADDS FADDD	SP DP	SP DP	SP DP	NV, OF, UF, NX	Addition
FSUBS FSUBD	SP DP	SP DP	SP DP	NV, OF, UF, NX	Subtraction
FMULS FMULD	SP DP	SP DP	SP DP	NV, OF, UF, NX NV, OF, UF, NX	Multiplication
FDIVS FDIVD	SP DP	SP DP	SP DP	NV, OF, UF, NX, DZ	Division
FSQRTS FSQRTD	- -	SP DP	SP DP	NV, NX	Square-root
Conversion operations					
FITOS FITOD	-	INT	SP DP	NX -	Integer to floating-point conversion
FSTOI FDTOI	-	SP DP	INT	NV, NX	Floating-point to integer conversion. The result is rounded in round-to-zero mode.
FSTOD FDTOS	-	SP DP	DP SP	NV NV, OF, UF, NX	Conversion between floating-point formats
Comparison operations					
FCMPS FCMPD	SP DP	SP DP	CC	NV	Floating-point compare. Invalid exception is generated if either operand is a signaling NaN.
FCMPES FCMPED	SP DP	SP DP	CC	NV	Floating point compare. Invalid exception is generated if either operand is a NaN (quiet or signaling).
Negate, Absolute value and Move					
FABSS	-	SP	SP	-	Absolute value.
FNEGS	-	SP	SP	-	Negate.
FMOVS		SP	SP	-	Move. Copies operand to result output.

SP - single precision floating-point number

DP - double precision floating-point number

INT - 32 bit integer

CC - condition codes

NV, OF, UF, NX - floating-point exceptions, see section 48.2.3

Below is a table of worst-case throughput of the floating point unit.

*Table 575.*Worst-case instruction timing

Instruction	Throughput	Latency
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	8	8
FDIVS	31	31
FDIVD	57	57
FSQRTS	46	46
FSQRTD	65	65

48.2.3 Exceptions

The floating-point unit detects all exceptions defined by the IEEE-754 standard. This includes detection of Invalid Operation (NV), Overflow (OF), Underflow (UF), Division-by-Zero (DZ) and Inexact (NX) exception conditions. Generation of special results such as NaNs and infinity is also implemented.

48.2.4 Rounding

All four rounding modes defined in the IEEE-754 standard are supported: round-to-nearest, round-to-+inf, round-to--inf and round-to-zero.

49 GRLFPC - GRFPU Lite Floating-point unit Controller

49.1 Overview

The GRFPU Lite Floating-Point Unit Controller (GRLFPC) is used to attach the GRFPU Lite floating-point unit (FPU) to the LEON integer unit (IU). It performs decoding and dispatching of the floating-point (FP) operations to the floating-point unit as well as managing the floating-point register file, the floating-point state register (FSR) and the floating-point deferred-trap queue (FQ).

The GRFPU Lite floating-point unit is not pipelined and executes only one instruction at a time. To improve performance, the controller (GRLFPC) allows the GRFPU Lite floating-point unit to execute in parallel with the processor pipeline as long as no new floating-point instructions are pending.

49.2 Floating-Point register file

The floating-point register file contains 32 32-bit floating-point registers (%f0-%f31). The register file is accessed by floating-point load and store instructions (LDF, LDDF, STD, STDF) and floating-point operate instructions (FPop).

In the FT-version, the floating-point register file is protected using 4-bit parity per 32-bit word. The controller is capable of detecting and correcting one bit error per byte. Errors are corrected using the instruction restart function in the IU.

49.3 Floating-Point State Register (FSR)

The controller manages the floating-point state register (FSR) containing FPU mode and status information. All fields of the FSR register as defined in SPARC V8 specification are implemented and managed by the controller conform to the SPARC V8 specification and IEEE-754 standard.

The non-standard bit of the FSR register is not used, all floating-point operations are performed in standard IEEE-compliant mode.

Following floating-point trap types never occur and are therefore never set in the ftt field:
- unimplemented_FPop: all FPop operations are implemented
- unfinished_FPop: all FPop operation complete with valid result
- invalid_fp_register: no check that double-precision register is 0 mod 2 is performed

The controller implements the *qne* bit of the FSR register which reads 0 if the floating-point deferred-queue (FQ) is empty and 1 otherwise. The FSR is accessed using LDFSR and STFSR instructions.

49.4 Floating-Point Exceptions and Floating-Point Deferred-Queue

The floating-point unit implements the SPARC deferred trap model for floating-point exceptions (fp_exception). A floating-point exception is caused by a floating-point instruction performing an operation resulting in one of following conditions:

- an operation raises IEEE floating-point exception (ftt = IEEE_754_exception) e.g. executing invalid operation such as 0/0 while the NVM bit of the TEM field id set (invalid exception enabled).
- sequence error: abnormal error condition in the FPU due to the erroneous use of the floating-point instructions in the supervisor software.
- hardware_error: uncorrectable parity error is detected in the FP register file

The trap is deferred to the next floating-point instruction (FPop, FP load/store, FP branch) following the trap-inducing instruction. When the trap is taken the floating-point deferred-queue (FQ) contains the trap-inducing instruction.

After the trap is taken the *qne* bit of the FSR is set and remains set until the FQ is emptied. STDFQ instruction reads a double-word from the floating-point deferred queue, the first word is the address of the instruction and the second word is the instruction code.

50 GRGPIO - General Purpose I/O Port

50.1 Overview

The general purpose input output port core is a scalable and provides optional interrupt support. The port width can be set to 2 - 32 bits through the *nbits* VHDL generic (i.e. *nbits* = 16). Interrupt generation and shaping is only available for those I/O lines where the corresponding bit in the *imask* VHDL generic has been set to 1.

Each bit in the general purpose input output port can be individually set to input or output, and can optionally generate an interrupt. For interrupt generation, the input can be filtered for polarity and level/edge detection.

It is possible to share GPIO pins with other signals. The output register can then be bypassed through the bypass register.

The figure 155 shows a diagram for one I/O line.

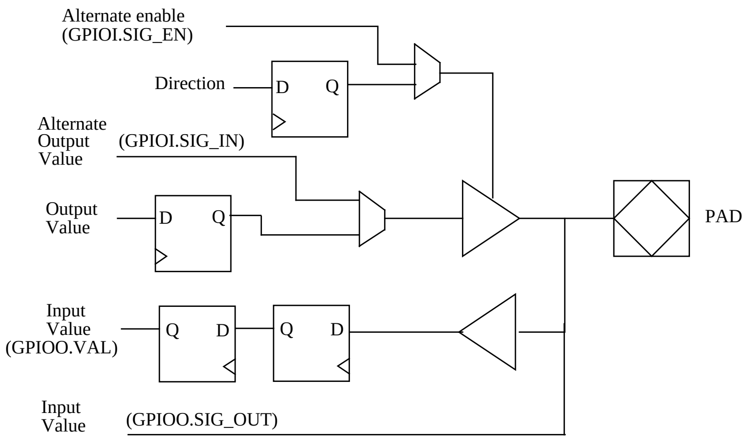


Figure 155. General Purpose I/O Port diagram

50.2 Operation

The I/O ports are implemented as bi-directional buffers with programmable output enable. The input from each buffer is synchronized by two flip-flops in series to remove potential meta-stability. The synchronized values can be read-out from the I/O port data register. They are also available on the GPIOO.VAL signals. The output enable is controlled by the I/O port direction register. A ‘1’ in a bit position will enable the output buffer for the corresponding I/O line. The output value driven is taken from the I/O port output register.

The core can be implemented with one of three different alternatives for interrupt generation. Either each I/O line can drive a separate interrupt line on the APB interrupt bus, the interrupt line to use can be assigned dynamically for each I/O line, or one interrupt line can be shared for all I/O lines. In the fixed mapping with a separate interrupt line for each I/O line, the interrupt number is equal to the I/O line index plus an offset given by the first interrupt line assigned to the core, *pirq*, (PIO[1] = interrupt *pirq*+1, etc.). If the core has been implemented to support dynamic mapping of interrupts, each I/O line can be mapped using the Interrupt map register(s) to an interrupt line starting at interrupt *pirq*. When the core is implemented to drive one, fixed, shared interrupt line for all I/O lines, the core will drive interrupt line *pirq* only. The value of *pirq* can be read out from the core’s AMBA plug’n’play information.

Interrupt generation is controlled by three registers: interrupt mask, polarity and edge registers. To enable an interrupt, the corresponding bit in the interrupt mask register must be set. If the edge register is ‘0’, the interrupt is treated as level sensitive. If the polarity register is ‘0’, the interrupt is active low. If the polarity register is ‘1’, the interrupt is active high. If the edge register is ‘1’, the interrupt is edge-triggered. The polarity register then selects between rising edge (‘1’) or falling edge (‘0’).

A GPIO pin can be shared with other signals. The ports that should have the capability to be shared are specified with the *bypass* generic (the corresponding bit in the generic must be 1). The unfiltered inputs are available through GPIOO.SIG_OUT and the alternate output value must be provided in GPIOI.SIG_IN. The bypass register then controls whether the alternate output is chosen. The direction of the GPIO pin can also be shared, if the corresponding bit is set in the *bpdir* generic. In such case, the output buffer is enabled when GPIOI.SIG_EN is active.

50.3 Registers

The core is programmed through registers mapped into APB address space.

Table 576. General Purpose I/O Port registers

APB address offset	**Register**
0x00	I/O port data register
0x04	I/O port output register
0x08	I/O port direction register
0x0C	Interrupt mask register
0x10	Interrupt polarity register
0x14	Interrupt edge register
0x18	Bypass register
0x1C	Capability register
0x20 - 0x3C	Interrupt map register(s). Address 0x20 + 4*n contains interrupt map registers for IO[4*n : 3+4+n], if implemented.

Table 577. I/O port data register

31 16	16-1 0
“000..0”	I/O port input value

16-1: 0 I/O port input value

Table 578. I/O port output register

31 16	16-1 0
“000..0”	I/O port output value

16-1: 0 I/O port output value

Table 579. I/O port direction register

31 16	16-1 0
“000..0”	I/O port direction value

16-1: 0 I/O port direction value (0=output disabled, 1=output enabled)

Table 580. Interrupt mask register

31 16	16-1 0
“000..0”	Interrupt mask

16-1: 0 Interrupt mask (0=interrupt masked, 1=intrrupt enabled)

Table 581. Interrupt polarity register

31 16	16-1 0
“000..0”	Interrupt polarity

Table 581. Interrupt polarity register

16-1:	0	Interrupt polarity (0=low/falling, 1=high/rising)

Table 582. Interrupt edge register

31 16	16-1 0
“000..0”	Interrupt edge

16-1:	0	Interrupt edge (0=level, 1=edge)

Table 583. Bypass register

31 16	16-1 0
“000..0”	Bypass

16-1:	0	Bypass. (0=normal output, 1=alternate output)

Table 584. Capability register

31 13	12 8	7 5	4 0
“000..0”	IRQGEN		NLINES

12	8	IRQGEN: Interrupt generation setting: If irqgen = 0, I/O line n will drive interrupt line pirq + n, up to NAHBIRQ-1. No Interrupt map registers will be implemented. This is the default, and traditional, implementation of the core. If irqgen = 1, all I/O lines capable of generating interrupts will use interrupt pirq and no Interrupt map registers are implemented. If irqgen > 1, the core will include Interrupt map registers allowing software to dynamically map which lines that should drive interrupt lines [pirq : pirq+irqgen-1]. The value of pirq can be read out from the core’s plug&play information. This field is available in revision 2 and above of the GPIO port. This field is read-only.
4:	0	NLINES. Number of pins in GPIO port - 1. Compatibility note: This field is available in revision 2 and above of the GPIO port. This field is read-only.

Table 585. Interrupt map register(s)

31 29	28 24	23 21	20 16	15 13	12 8	7 6	4 0
“000..0”	IRQMAP[4*n]	“000..0”	IRQMAP[4*n+1]	“000..0”	IRQMAP[4*n+2]	“000..0”	IRQMAP[4*n+3]

31:	0	IRQMAP[i] : The field IRQMAP[i] determines to which interrupt I/O line i is connected. If IRQMAP[i] is set to *x*, IO[i] will drive interrupt *pirq*+*x*. Where *pirq* is the first interrupt assigned to the core. Several I/O can be mapped to the same interrupt. The core has one IRQMAP field per I/O line. The Interrupt map register at offset 0x20+*4*n* contains the IRQMAP fields for IO[*4*n* : *4*n+3*]. This means that the fields for IO[0:3] are located on offset 0x20, IO[4:7] on offset 0x24, IO[8:11] on offset 0x28, and so on. An I/O line’s interrupt generation must be enabled in the Interrupt mask register in order for the I/O line to drive the interrupt specified by the IRQMAP field. The Interrupt map register(s) can only be written if the core was implemented with support for interrupt mapping.

50.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x01A. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

50.5 Configuration options

Table 586 shows the configuration options of the core (VHDL generics).

*Table 586.*Configuration options

Generic	**Function**	**Allowed range**	**Default**
pindex	Selects which APB select signal (PSEL) will be used to access the GPIO unit	0 to NAHBIRQ-1	0
paddr	The 12-bit MSB APB address	0 to 16#FFF#	0
pmask	The APB address mask	0 to 16#FFF#	16#FFF#
nbits	Defines the number of bits in the I/O port	1 to 32	8
imask	Defines which I/O lines are provided with interrupt generation and shaping	0 - 16#FFFF#	0
oepol	Select polarity of output enable signals. 0 = active low, 1 = active high.	0 - 1	0
syncrst	Selects between synchronous (1) or asynchronous (0) reset during power-up.	0 - 1	0
bypass	Defines which I/O lines are provided bypass capabilities	0 - 16#7FFFFFFF#	0
scantest	Enable scan support for asyncronous-reset flip-flops	0 - 1	0
bpdir	Defines which I/O lines are provided output enable bypass capabilities	0 - 16#7FFFFFFF#	0
pirq	First interrupt line that the core will drive. The core will only drive interrupt lines up to line NAHBIRQ-1. If NAHBIRQ is set to 32 and *pirq* is set to 16, the core will only be able to generate interrupts for I/O lines 0 - 15.	0 - NAHBIRQ-1	0
irqgen	This generic configures interrupt generation. If *irqgen* = 0, I/O line *n* will drive interrupt line *pirq* + *n*, up to NAHBIRQ-1. No Interrupt map registers will be implemented. This is the default, and traditional, implementation of the core. If *irqgen* = 1, all I/O lines capable of generating interrupts will use interrupt *pirq* and no Interrupt map registers will be implemented. If *irqgen* > 1, the core will include Interrupt map registers allowing software to dynamically map which lines that should drive interrupt lines [*pirq* : *pirq*+*irqgen*-1]	0 - NAHBIRQ-1	0

50.6 Signal descriptions

Table 587 shows the interface signals of the core (VHDL ports).

*Table 587.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GPIOO	OEN[31:0]	Output	I/O port output enable	see oepol
	DOUT[31:0]	Output	I/O port outputs	-
	VAL[31:0]	Output	The current (synchronized) value of the GPIO signals	-
	SIG_OUT[31:0]	Output	The current (unsynchronized) value of the GPIO signals	
GPIOI	DIN[31:0]	Input	I/O port inputs	-
	SIG_IN[31:0]	Input	Alternate output	-
	SIG_EN[31:0]	Input	Alternate output enable	High

* see GRLIB IP Library User’s Manual

50.7 Library dependencies

Table 588 shows libraries used when instantiating the core (VHDL libraries).

*Table 588.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Signals, component	Component declaration

50.8 Component declaration

The core has the following component declaration.

```
ibrary gaisler;
use gaisler.misc.all;

entity grgpio is
  generic (
    pindex   : integer := 0;
    paddr    : integer := 0;
    pmask    : integer := 16#fff#;
    imask    : integer := 16#0000#;
    nbits    : integer := 16-- GPIO bits

  );
  port (
    rst    : in  std_ulogic;
    clk    : in  std_ulogic;
    apbi   : in  apb_slv_in_type;
    apbo   : out apb_slv_out_type;
    gpioi  : in  gpio_in_type;
    gpioo  : out gpio_out_type
  );
```

```
end;
```

50.9 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

signal gpti : gptimer_in_type;

begin

gpio0 : if CFG_GRGPIO_EN /= 0 generate     -- GR GPIO unit
    grgpio0: grgpio
      generic map( pindex => 11, paddr => 11, imask => CFG_GRGPIO_IMASK, nbits => 8)
      port map( rstn, clkm, apbi, apbo(11), gpioi, gpioo);

      pio_pads : for i in 0 to 7 generate
        pio_pad : iopad generic map (tech => padtech)
            port map (gpio(i), gpioo.dout(i), gpioo.oen(i), gpioi.din(i));
      end generate;
end generate;
```

51 GRGPREG - General Purpose Register

51.1 Overview

The core provides a programmable register that drives an output vector that can be used to control miscellaneous options in a design.

51.2 Operation

The core contains one register of configurable length that is mapped into APB address space. The value of this register is propagated to an output vector. The reset value of the register can be specified via VHDL generics, or via an input vector.

51.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 589.*General purpose register registers

APB address offset	**Register**
0x00	General purpose register bits 31:0
0x04	General purpose register bits 63:0

Table 590. General purpose register

31 nbits	nbits-1 0
RESERVED	REGISTER BITS

31:nbits	RESERVED (not present if nbits >= 32)
nbits-1:0	Register bits. Position i corresponds to bit i in the core's output vector

Table 591. General purpose register (extended)

31 nbits-32	nbits-33 0
RESERVED	REGISTER BITS

31:nbits-32	RESERVED (not present if nbits = 64 or nbits <= 32)
nbits-33:0	Register bits. Position i corresponds to bit 31+i in the core's output vector (not present if nbits < 33)

51.4 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x087. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

51.5 Configuration options

Table 592 shows the configuration options of the core (VHDL generics).

*Table 592.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
nbits	Number of register bits	1 - 64	16
rstval	Reset value for bits 31:0	0 - 16#FFFFFFFF#	0
rstval2	Reset value for bits 63:32	0 - 16#FFFFFFFF#	0
extrst	Use input vector *resval* to specify reset value. If this generic is 0 the register reset value is determined by VHDL generics *rstval* and *rstval2*. If this generic is 1, the reset value is specified by the input vector *resval*.	0 - 1	0

51.6 Signal descriptions

Table 593 shows the interface signals of the core (VHDL ports).

*Table 593.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GRGPREGO	N/A	Output	Value of register mapped into APB address space	-
RESVAL	N/A	Input	(Optionally) specifes register reset value	-

* see GRLIB IP Library User’s Manual

51.7 Library dependencies

Table 594 shows the libraries used when instantiating the core (VHDL libraries).

*Table 594.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component, signals	Component declaration, I2C signal definitions

51.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;

library gaisler;
use gaisler.misc.all;

entity grgpreg_ex is
```

```
  port (
    clkm  : in std_ulogic;
    rstn : in std_ulogic;

    -- I2C signals
    iic_scl : inout std_ulogic;
    iic_sda : inout std_ulogic
    );
end;

architecture rtl of i2c_ex is
  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- Width of general purpose register
  constant GRGPREG_NBITS : integer := 9;

  signal gprego           : std_logic_vector(GRGPREG_NBITS-1 downto 0);
  signal gpregresval      : std_logic_vector(GRGPREG_NBITS-1 downto 0);
begin

  -- AMBA Components are instantiated here
  ...

  -- General purpose register
  grgpreg0 : grgpreg                  -- General purpose register
    generic map (
      pindex  => 10,
      paddr   => 16#0a0#,
      pmask   => 16#fff#,
      nbits   => GRGPREG_NBITS,
      rstval  => 0,                   -- Not used
      rstval2 => 0,                   -- Not used
      extrst  => 1)                   -- Use input vector for reset value
    port map (
      rst     => rstn,
      clk     => clkm,
      apbi    => apbi,
      apbo    => apbo(10),
      gprego  => gprego,
      resval  => gpregresval);
end;
```

52 GRIOMMU - AHB/AHB bridge with access protection and address translation

52.1 Overview

The core is used to connect two AMBA AHB buses clocked by synchronous clocks with any frequency ratio. The two buses are connected through an interface pair consisting of an AHB slave and an AHB master interface. AHB transfer forwarding is performed in one direction, where AHB transfers to the slave interface are forwarded to the master interface. The core can be configured to provide access protection and address translation for AMBA accesses traversing over the core. Access protection can be provided using a bit vector to restrict access to memory. Access protection and address translation can also be provided using page tables in main memory, providing full IOMMU functionality. Both protection strategies allow devices to be placed into a configurable number of groups that share data structures located in main memory. The protection and address translation functionality provides protection for memory assigned to processes and operating systems from unwanted accesses by units capable of direct memory access.

Applications of the core include system partitioning, clock domain partitioning, system expansion and secure software partitioning.

Features offered by the core include:

- Single and burst AHB transfer forwarding
- Access protection and address translation that can provide full IOMMU functionality
- Devices can be placed into groups where a group shares page tables / access restriction vectors
- Hardware table-walk
- Efficient bus utilization through (optional) use of SPLIT response, data prefetching and posted writes
- Read and write combining, improves bus utilization and allows connecting cores with differing AMBA access size restrictions.
- Deadlock detection logic enables use of two uni-directional bridges to build a bi-directional bridge. The core can be connected with an another instance of the core, or with a uni-directional AHB/AHB bridge core (AHB2AHB), to form a bi-directional bridge.

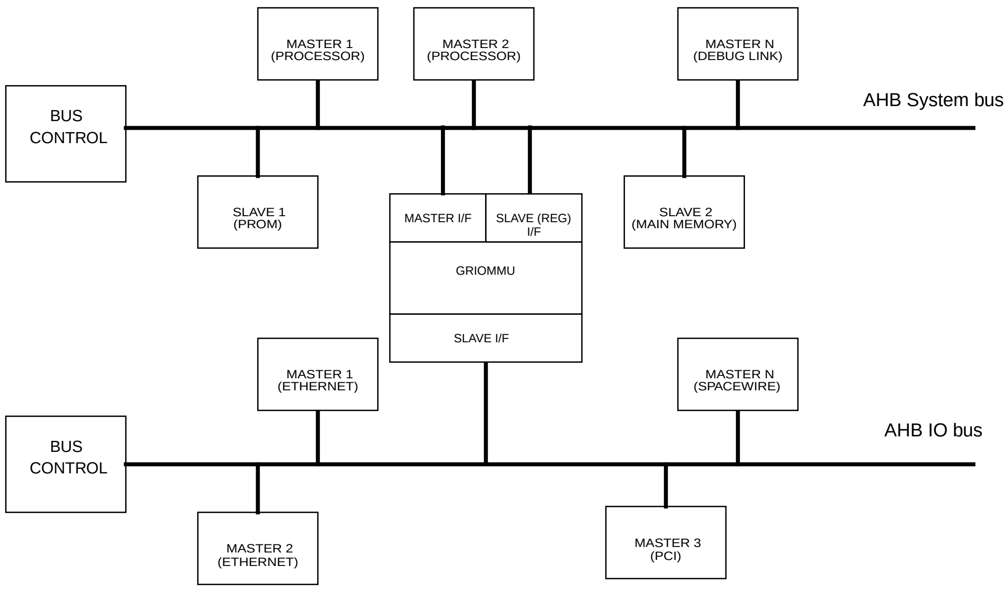


Figure 156. System with core providing access restricion/address translation for masters on AHB IO bus

52.2 Bridge operation

52.2.1 General

The first sub sections below describe the general AHB bridge function. The functionality providing access restriction and address translation is described starting with section 52.3. In the description of AHB accesses below the core propagates accesses from the IO bus to the System bus, see figure 156.

The address space occupied by the core on the IO bus is configurable and determined by Bank Address Registers in the slave interface's AHB Plug&Play configuration record.

The core is capable of handling single and burst transfers of all burst types. Supported transfer sizes (HSIZE) are BYTE, HALF-WORD, WORD, DWORD, 4WORD and 8WORD.

For AHB write transfers write data is always buffered in an internal FIFO implementing posted writes. For AHB read transfers the core uses GRLIB's AMBA Plug&Play information to determine whether the read data will be prefetched and buffered in an internal FIFO. If the target address for an AHB read burst transfer is a prefetchable location the read data will be prefetched and buffered.

The core can be implemented to use SPLIT responses or to insert wait states when handling an access. With SPLIT responses enabled, an AHB master initiating a read transfer to the core is always splitted on the first transfer attempt to allow other masters to use the slave bus while the core performs read transfer on the master bus. The descriptions of operation in the sections below assume that the core has been implemented with support for AMBA SPLIT responses. The effects of disabling support for AMBA SPLIT responses are described in section 52.2.11.

If interrupt forwarding is enabled the interrupts on the IO bus interrupt lines will be forwarded to the system bus and vice versa.

52.2.2 AHB read transfers

When a read transfer is registered on the slave interface connected to the IO bus, the core gives a SPLIT response. The master that initiated the transfer will be de-granted allowing other bus masters to use the slave bus while the core performs a read transfer on the master side. The master interface then requests the bus and starts the read transfer on the master side. Single transfers on the slave side are normally translated to single transfers with the same AHB address and control signals on the master side, however read combining can translate one access into several smaller accesses. Translation of burst transfers from the slave to the master side depends on the burst type, burst length, access size and core configuration.

If the read FIFO is enabled and the transfer is a burst transfer to a prefetchable location, the master interface will prefetch data in the internal read FIFO. If the splitted burst on the slave side was an incremental burst of unspecified length (INCR), the length of the burst is unknown. In this case the master interface performs an incremental burst up to a specified address boundary (determined by the VHDL generic *rburst)*. The core can be configured to recognize an INCR read burst marked as instruction fetch (indicated on HPROT signal). In this case the prefetching on the master side is completed at the end of a cache line (the cache line size is configurable through the VHDL generic *iburst*). When the burst transfer is completed on the master side, the splitted master that initiated the transfer (on the slave side) is allowed in bus arbitration by asserting the appropriate HSPLIT signal to the AHB controller. The splitted master re-attempts the transfer and the core will return data with zero wait states.

If the read FIFO is disabled, or the burst is to non-prefetchable area, the burst transfer on the master side is performed using sequence of NONSEQ, BUSY and SEQ transfers. The first access in the burst on the master side is of NONSEQ type. Since the master interface can not decide whether the splitted burst will continue on the slave side or not, the system bus is held by performing BUSY transfers. On the slave side the splitted master that initiated the transfer is allowed in bus arbitration by asserting the HSPLIT signal to the AHB controller. The first access in the transfer is completed by returning read data. The next access in the transfer on the slave side is extended by asserting HREADY low. On the master side the next access is started by performing a SEQ transfer (and then holding the bus using BUSY transfers). This sequence is repeated until the transfer is ended on the slave side.

In case of an ERROR response on the master side the ERROR response will be given for the same access (address) on the slave side. SPLIT and RETRY responses on the master side are re-attempted until an OKAY or ERROR response is received.

52.2.3 AHB write transfers

The core implements posted writes. During the AHB write transfer on the slave side the data is buffered in the internal write FIFO and the transfer is completed on the slave side by always giving an OKAY response. The master interface requests the bus and performs the write transfer when the master bus is granted. If the burst transfer crosses the write burst boundary (defined by VHDL generic *wburst*), a SPLIT response is given. When the core has written the contents of the FIFO out on the master side, the core will allow the master on the slave side to perform the remaining accesses of the write burst transfer.

Writes are accepted with zero wait states if the core is idle and the incoming access is not locked. If the incoming access is locked, each access will have one wait state. If write combining is disabled a non-locked BUSY cycle will lead to a flush of the write FIFO. If write combining is enabled or if the incoming access is locked, the core will not flush the write FIFO during the BUSY cycle.

52.2.4 Deadlock conditions

When two cores are used to form a bi-directional bridge, a deadlock situation can occur if the cores are simultaneously accessed from both buses. The core that has been configured as a slave contains deadlock detection logic which will resolve a deadlock condition by giving a RETRY response, or by issuing SPLIT complete followed by a new SPLIT response. When the core resolves a deadlock while prefetching data, any data in the prefetch buffer will be dropped when the core’s slave interface issues the AMBA RETRY response. When the access is retried it may lead to the same memory locations being read twice.

Deadlock detection logic for bi-directional configurations may lead to deadlocks in other parts of the system. Consider the case where a processor on bus A on one side of the bidirectional bridge needs to perform an instruction fetch over the bridge before it can release a semaphore located in memory on bus A. Another processor on bus B, on the other side of the bridge, may spin on the semaphore waiting for its release. In this scenario, the accesses from the processor on bus B could, depending on system configuration, continuously trigger a deadlock condition where the core will drop data in, or be prevented from initiating, the instruction fetch for the processor on bus A. Due to scenarios of this kind the bridge should not be used in bi-directional configurations where dependencies as the one described above exist between the buses connected by the bridge.

Other deadlock conditions exist with locked transfers, see section 52.2.5.

52.2.5 Locked transfers

The core supports locked transfers. The master bus will be locked when the bus is granted and remain locked until the transfer completes on the slave side. Locked transfers can lead to deadlock conditions, the core’s VHDL generic *lckdac* determines if and how the deadlock conditions are resolved.

With the VHDL generic *lckdac* set to 0, locked transfers may *not* be made after another read access which received SPLIT until the first read access has received split complete. This is because the core will return split complete for the first access first and wait for the first master to return. This will cause deadlock since the arbiter is not allowed to change master until a locked transfer has been completed. The AMBA specification requires that the locked transfer is handled before the previous transfer, which received a SPLIT response, is completed.

With *lckdac* set to 1, the core will respond with an AMBA ERROR response to locked access that is made while an ongoing read access has received a SPLIT response. With *lckdac* set to 2 the core will save state for the read access that received a SPLIT response, allow the locked access to complete, and then complete the first access. All non-locked accesses from other masters will receive SPLIT responses until the saved data has been read out.

If the core is used to create a bi-directional bridge there is one more deadlock condition that may arise when locked accesses are made simultaneously in both directions. If the VHDL generic *lckdac* is set to 0 the core will deadlock. If *lckdac* is set to a non-zero value the slave bridge will resolve the deadlock condition by issuing an AMBA ERROR response to the incoming locked access.

52.2.6 Read and write combining

Read and write combining allows the core to assemble or split AMBA accesses on the core's slave interface into one or several accesses on the master interface. This functionality can improve bus utilization and also allows cores that have differing AMBA access size restrictions to communicate with each other. The functionality attained by read and write combining depends on the VHDL generics *rdcomb* (defines type of read combining), *wrcomb* (defines type of write combining), *slvmstaccsz* (defines maximum AHB access size supported by the core's slave interface) and *mstmaccsz* (defines maximum AHB access size that can be used by core's master interface). These VHDL generics are described in section 52.13. The table below shows the effect of different settings. BYTE and HALF-WORD accesses are special cases. The table does not list illegal combinations, for instance *mstmaccsz* /= *slvmaccsz* requires that *wrcomb* /= 0 and *rdcomb* /= 0.

*Table 595.*Read and write combining

Access on slave interface	**Access size**	**wrcomb**	**rdcomb**	**Resulting access(es) on master interface**
BYTE or HALF-WORD single read access to any area	-	-	-	Single access of same size
BYTE or HALF-WORD read burst to prefetchable area	-	-	-	Incremental read burst of same access size as on slave interface, the length is the same as the number of 32-bit words in the read buffer, but will not cross the read burst boundary.
BYTE or HALF-WORD read burst to non-prefetchable area	-	-	-	Incremental read burst of same access size as on slave interface, the length is the same as the length of the incoming burst. The master interface will insert BUSY cycles between the sequential accesses.
BYTE or HALF-WORD single write	-	-	-	Single access of same size
BYTE or HALF-WORD write burst	-	-	-	Incremental write burst of same size and length, the maximum length is the number of 32-bit words in the write FIFO.
Single read access to any area	Access size <= mstmaccsz	-	-	Single access of same size
Single read access to any area	Access size > mstmaccsz	-	1	Sequence of single accesses of mstmaccsz. Number of accesses: (access size)/mstmaccsz
Single read access to any area	Access size > mstmaccsz	-	2	Burst of accesses of size mstmaccsz. Length of burst: (access size)/mstmaccsz
Read burst to prefetchable area	-	-	0	Burst of accesses of incoming access size up to address boundary defined by rburst.
Read burst to prefetchable area	-	-	1 or 2	Burst of accesses of size mstmaccsz up to address boundary defined by rburst.
Read burst to non-prefetchable area	Access size <= mstmaccsz	-	-	Incremental read burst of same access size as on slave interface, the length is the same as the length of the incoming burst. The master interface will insert BUSY cycles between the sequential accesses.

*Table 595.*Read and write combining

Access on slave interface	Access size	wrcomb	rdcomb	Resulting access(es) on master interface
Read burst to non-prefetch-able area	Access size > mstmaccsz	-	1 or 2	Burst of accesses of size mstmaccsz. Length of burst: (incoming burst length)*(access size)/mstmaccsz
Single write	Access size <= mstmaccsz	-	-	Single write access of same size
Single write	Access size > mstmaccsz	1	-	Sequence of single access of mstmaccsz. Number of accesses: (access size)/mstmaccsz.
Single write	Access size > mstmaccsz	2	-	Burst of accesses of mstmaccsz. Length of burst: (access size)/mstmaccsz.
Write burst	-	0	-	Burst of same size as incoming burst, up to address boundary defined by VHDL generic wburst.
Write burst	-	1 or 2	-	Burst write of maximum possible size. The core will use the maximum size (up to mstmaccsz) that it can use to empty the write buffer.

Read and write combining prevents the bridge from propagating fixed length bursts and wrapping bursts. See section 52.2.7 for a discussion on burst operation.

Read and write combining with VHDL generics *wrcomb/rdcomb* set to 1 cause the core to use single accesses when dividing an incoming access into several smaller accesses. This means that another master on the bus may write or read parts of the memory area to be accessed by the core before the core has read or written all the data. In bi-directional configurations, an incoming access on the master core may cause a collision that aborts the operation on the slave core. This may cause the core to read the same memory locations twice. This is normally not a problem when accessing memory areas. The same issues apply when using an AHB arbiter that performs early burst termination. The standard GRLIB AHBCTRL core does not perform early burst termination.

To ensure that the core does not re-read an address, and that all data in an access from the core's slave interface is propagated out on the master interface without interruption the VHDL generics *rdcomb* and *wrcomb* should both be set to 0 or 2. In addition to this, the AHB arbiter may not perform early burst termination (early burst termination is not performed by the GRLIB AHBCTRL arbiter).

Read and write combining can be limited to specified address ranges. See description of the *comb-mask* VHDL generic for more information. Note that if the core is implemented with support for prefetch and read combining, it will not obey combmask for prefetch operations (burst read to prefetchable areas). Prefetch operations will always be performed with the maximum allowed size on the master interface.

52.2.7 Burst operation

The core can be configured to support all AMBA 2.0 burst types (single access, incrementing burst of unspecified length, fixed length incrementing bursts and wrapping bursts). Single accesses and incrementing bursts of unspecified length have previously been discussed in this document. An incoming single access will lead to one access, or multiple accesses for some cases with read/write combining, on the other side of the bridge. An incoming incrementing burst of unspecified length to a prefetchable area will lead to the prefetch buffer (if available) being filled using the same access size, or the maximum allowed access size if read/write combining is enabled, on the master interface.

If the core is used in a system where no fixed length bursts or incremental bursts will be used in accesses to the bridge, then set the *allbrst* generic to 0 and skip the remainder of this section.

The VHDL generic *allbrst* controls if the core will support fixed length and wrapping burst accesses. If *allbrst* is set to 0, the core will treat all burst accesses as incrementing of unspecified length. For fixed length and wrapping bursts this can lead to performance penalties and malfunctions. Support for fixed length and wrapping bursts is enabled by setting *allbrst* to 1 or 2. Table 52.2.7 describes how the core will handle different burst types depending on the setting of *allbrst*.

*Table 596.*Burst handling

Value of allbrst generic	**Access type***	**Undefined length incrementing burst INCR**	**Fixed length incrementing burst INCR{4,8,16}**	**Wrapping burst WRAP{4,8,16}**
0	Reads to non-prefetchable area	Incrementing burst with BUSY cycles inserted. Same behaviour with read and write combining.	Fixed length burst with BUSY cycles inserted. If the burst is short then the burst may end with a BUSY cycle. If access combining is used the HBURST signal will get incorrect values.	Malfunction. Not supported
	Reads to prefetchable area	Incrementing burst of maximum allowed size, filling prefetch buffer, starting at address boundary defined by prefetch buffer.		Malfunction. Not supported
	Write burst	Incrementing burst	Incrementing burst, if write combining is enabled, and triggered, the burst will be translated to an incrementing burst of undefined length. VHDL generic *wrcomb* should not be set to 1 (but to 0 or 2) in this case	Write combining is not supported. Same access size will be used on both sides of the bridge.
1	Reads to non-prefetchable area	Incrementing burst with BUSY cycles inserted. Same behaviour with read and write combining.	Same burst type with BUSY cycles inserted. If read combining is enabled, and triggered by the incoming access size, an incremental burst of unspecified length will be used. If the burst is short then the burst may end with a BUSY cycle.	Same burst type with BUSY cycles inserted. If read combining is enabled, and triggered by the incoming access size, an incremental burst of unspecified length will be used. This will cause AMBA violations if the wrapping burst does not start from offset 0.
	Reads to prefetchable area	Incrementing burst of maximum allowed size, filling prefetch buffer.	For reads, the core will perform full (or part that fits in prefetch buffer) fixed/wrapping burst on master interface and then respond with data. No BUSY cycles are inserted. If the access made to the slave interface is larger than the maximum supported access size on the master interface then a incrementing burst of unspecified length will be used to fill the prefetch buffer. This (read combining) is not supported for wrapping bursts.	
	Write burst	Same as for allbrst = 0		
2	Reads to non-prefetchable area	Incrementing burst with BUSY cycles inserted. Same behaviour with read and write combining.	Reads are treated as a prefetchable burst. See below.	
	Reads to prefetchable area	Incrementing burst of maximum allowed size, filling prefetch buffer, starting at address boundary defined by prefetch buffer.	Core will perform full (or part that fits in prefetch buffer) fixed/wrapping burst on master interface and then respond with data. No BUSY cycles are inserted. If the access made to the slave interface is larger than the maximum supported access size on the master interface then a incrementing burst of unspecified length will be used to fill the prefetch buffer. This (read combining) is not supported for wrapping bursts.	
	Write burst	Same as for allbrst = 0		

* Access to prefetchable area where the core’s prefetch buffer is ised (VHDL generic pfen /= 0).

52.2.8 Transaction ordering, starvation and AMBA arbitration schemes

The core is configured at implementation to use one of two available schemes to handle incoming accesses. The core will issue SPLIT responses when it is busy and on incoming read accesses. If the core has been configured to use first-come, first-served ordering it will keep track of the order of incoming accesses and serve the requests in the same order. If first-come, first-served ordering is disabled the core will give some advantage to the master it has a response for and then allow all masters in to arbitration simultaneously, moving the decision on which master that should be allowed to access the core to the bus arbitration.

When designing a system containing a core the expected traffic patterns should be analyzed. The designer must be aware how SPLIT responses affect arbitration and how the selected transaction ordering in the core will affect the system. The two different schemes are further described in sections 52.2.9 and 52.2.10.

52.2.9 First-come, first-served ordering

First-come, first served ordering is used when the VHDL generic *fcfs* is non-zero.

With first-come, first-served ordering the core will keep track of the order of incoming accesses. The accesses will then be served in the same order. For instance, if master 0 initiates an access to the core, followed by master 3 and then master 5, the core will propagate the access from master 0 (and respond with SPLIT on a read access) and then respond with SPLIT to the other masters. When the core has a response for master 0, this master will be allowed in arbitration again by the core asserting HSPLIT. When the core has finished serving master 0 it will allow the next queued master in arbitration, in this case master 3. Other incoming masters will receive SPLIT responses and will not be allowed in arbitration until all previous masters have been served.

An incoming locked access will always be given precedence over any other masters in the queue.

A burst that has initiated a pre-fetch operation will receive SPLIT and be inserted last in the master queue if the burst is longer than the maximum burst length that the core has been configured for.

It should be noted that first-come, first-served ordering may not work well in systems where an AHB master needs to have higher priority compared to the other masters. The core will not prioritize any master, except for masters performing locked accesses.

52.2.10 Bus arbiter ordering

Bus arbiter ordering is used when VHDL generic *fcfs* is set to zero.

When several masters have received SPLIT and the core has a response for one of these masters, the master with the queued response will be allowed in to bus arbitration by the core asserting the corresponding HSPLIT signal. In the following clock cycle, all other masters that have received SPLIT responses will also be allowed in bus arbitration as the core asserts their HSPLIT signals simultaneously. By doing this the core defers the decision on the master to be granted next to the AHB arbiter. The core does not show any preference based on the order in which it issued SPLIT responses to masters, except to the master that initially started a read or write operation. Care has been taken so that the core shows a consistent behavior when issuing SPLIT responses. For instance, the core could be simplified if it could issue a SPLIT response just to be able to change state, and not initiate a new operation, to an access coming after an access that read out prefetched data. When the core entered its idle state it could then allow all masters in bus arbitration and resume normal operation. That solution could lead to starvation issues such as:

T0: Master 1 and Master 2 have received SPLIT responses, the core is prefetching data for Master 1

T1: Master 1 is allowed in bus arbitration by setting the corresponding HSPLIT

T2: Master 1 reads out prefetch data, Master 2 HSPLIT is asserted to let Master 2 in to bus arbitration

T3: Master 2 performs an access, receives SPLIT, however the core does not initiate an access, it just stalls in order to enter its idle state.

T4: Master 2 is allowed in to bus arbitration, Master 1 initiates an access that leads to a prefetch and Master 1 receives a SPLIT response

T5: Master 2 performs an access, receives SPLIT since the core is prefetching data for master 1

T6: Go back to T0

This pattern will repeat until Master 1 backs away from the bus and Master 2 is able to make an access that starts an operation over the core. In most systems it is unlikely that this behavior would introduce a bus lock. However, the case above could lead to an unexpectedly long time for Master 2 to complete its access. Please note that the example above is illustrative and the problem does not exist in the core as the core does not issue SPLIT responses to (non-locked) accesses in order to just change state but a similar pattern could appear as a result of decisions taken by the AHB arbiter if Master 1 is given higher priority than Master 2.

In the case of write operations the scenario is slightly different. The core will accept a write immediately and will not issue a SPLIT response. While the core is busy performing the write on the master side it will issue SPLIT responses to all incoming accesses. When the core has completed the write operation on the master side it will continue to issue SPLIT responses to any incoming access until there is a cycle where the core does not receive an access. In this cycle the core will assert HSPLIT for all masters that have received a SPLIT response and return to its idle state. The first master to access the core in the idle state will be able to start a new operation. This can lead to the following behavior:

T0: Master 1 performs a write operation, does NOT receive a SPLIT response

T1: Master 2 accesses the core and receives a SPLIT response

T2: The core now switches state to idle since the write completed and asserts HSPLIT for Master 2.

T3: Master 1 is before Master 2 in the arbitration order and we are back at T0.

In order to avoid this last pattern the core would have to keep track of the order in which it has issued SPLIT responses and then assert HSPLIT in the same order. This is done with first-come, first-served ordering described in section 52.2.9.

52.2.11 AMBA SPLIT support

Support for AMBA SPLIT responses is enabled/disabled through the VHDL generic *split*. SPLIT support should be enabled for most systems. The benefits of using SPLIT responses is that the bus on the core's slave interface side can be free while the core is performing an operation on the master side. This will allow other masters to access the bus and generally improve system performance. The use of SPLIT responses also allows First-come, first-served transaction ordering.

For configurations where the core is the only slave interface on a bus, it can be beneficial to implement the core without support for AMBA SPLIT responses. Removing support for SPLIT responses reduces the area used by the core and may also reduce the time required to perform accesses that traverse the core. It should be noted that building a bi-directional core without support for SPLIT responses will increase the risk of access collisions.

If SPLIT support is disabled the core will insert wait states where it would otherwise issue a SPLIT response. This means that the arbitration ordering will be left to the bus arbiter and the core cannot be implemented with the First-come, first-served transaction ordering scheme. The core will still issue RETRY responses to resolve dead lock conditions, to split up long burst and also when the core is busy emptying it's write buffer on the master side.

The core may also be implemented with dynamic SPLIT support, this allows the use of SPLIT responses to be configurable via the core's register interface (see SP field in the core's Control register).

52.2.12 Core latency

The delay incurred when performing an access over the core depends on several parameters such as core configuration, the operating frequency of the AMBA buses, AMBA bus widths and memory access patterns. This section deals with latencies in the core's bridge function. Access protection mechanisms may add additional delays, please refer to the description of access protection for a description of any additional delays.

Table 597 below shows core behavior in a system where both AMBA buses are running at the same frequency and the core has been configured to use AMBA SPLIT responses. Table 598 further down shows core behavior in the same system without support for SPLIT responses.

*Table 597.*Example of single read with FFACT = 1, and SPLIT support

Clock cycle	Core slave side activity	Core master side activity
0	Discovers access and transitions from idle state	Idle
1	Slave side waits for master side, SPLIT response is given to incoming access, any new incoming accesses also receive SPLIT responses.	Discovers slave side transition. Master interface output signals are assigned.
2		If bus access is granted, perform address phase. Otherwise wait for bus grant.
3		Register read data and transition to data ready state.
4	Discovers that read data is ready, assign read data output and assign SPLIT complete	Idle
5	SPLIT complete output is HIGH	
6	Typically a wait cycle for the SPLIT:ed master to be allowed into arbitration. Core waits for master to return. Other masters receive SPLIT responses.	
7	Master has been allowed into arbitration and performs address phase. Core keeps HREADY high	
8	Access data phase. Core has returned to idle state.	

*Table 598.*Example of single read with FFACT = 1, without SPLIT support

Clock cycle	Core slave side activity	Core master side activity
0	Discovers access and transitions from idle state	Idle
1	Slave side waits for master side, wait states are inserted on the AMBA bus.	Discovers slave side transition. Master interface output signals are assigned.
2		Bus access is granted, perform address phase.
3		Register read data and transition to data ready state.
4	Discovers that read data is ready, assign HREADY output register and data output register.	Idle
5	HREADY is driven on AMBA bus. Core has returned to idle state	

While the transitions shown in tables 597 and 598 are simplified they give an accurate view of the core delay. If the master interface needs to wait for a bus grant or if the read operation receives wait states, these cycles must be added to the cycle count in the tables. The behavior of the core with a frequency factor of two between the buses is shown in tables 599 and 600 (best case, delay may be larger depending on which slave clock cycle an access is made to the core).

*Table 599.*Example of single read with FFACT = 2, Master freq. > Slave freq, without SPLIT support

Slave side clock cycle	**Core slave side activity**	**Master side clock cycle**	**Core master side activity**
0	Discovers access and transitions from idle state	0	Discovers slave side transition. Master interface output signals are assigned.
1	Slave side waits for master side, wait states are inserted on the AMBA bus.		
2		1	Bus access is granted, perform address phase.
3			
4		2	Register read data and transition to data ready state.
5			
6	Discovers that read data is ready, assign HREADY output register and data output register.	3	Idle
7	HREADY is driven on AMBA bus. Core has returned to idle state		

*Table 600.*Example of single read with FFACT = 2, Master freq. > Slave freq, without SPLIT support

Slave side clock cycle	**Core slave side activity**	**Master side clock cycle**	**Core master side activity**
0	Discovers access and transitions from idle state	0	Idle
		1	
1	Slave side waits for master side, wait states are inserted on the AMBA bus.	2	Discovers slave side transition. Master interface output signals are assigned.
		3	Bus access is granted, perform address phase.
2	Discovers that read data is ready, assign HREADY output register and data output register.	4	Register read data and transition to data ready state.
		5	Idle
3	HREADY is driven on AMBA bus. Core has returned to idle state	6	
		7	

Table 601 below lists the delays incurred for single operations that traverse the bridge while the bridge is in its idle state. The second column shows the number of cycles it takes the master side to perform the requested access, this column assumes that the master slave gets access to the bus immediately and that each access is completed with zero wait states. The table only includes the delay incurred by traversing the core. For instance, when the access initiating master reads the core’s prefetch buffer, each additional read will consume one clock cycle. However, this delay would also have been present if the master accessed any other slave.

Write accesses are accepted with zero wait states if the bridge is idle, this means that performing a write to the idle core does not incur any extra latency. However, the core must complete the write operation on the master side before it can handle a new access on the slave side. If the core has not transitioned into its idle state, pending the completion of an earlier access, the delay suffered by an access be longer than what is shown in the tables in this section. Accesses may also suffer increased delays during collisions when the core has been instantiated to form a bi-directional bridge. Locked accesses that abort on-going read operations will also mean additional delays.

If the core has been implemented to use AMBA SPLIT responses there will be an additional delay where, typically, one cycle is required for the arbiter to react to the assertion of HSPLIT and one clock cycle for the repetition of the address phase.

Note that if the core has support for read and/or write combining, the number of cycles required for the master will change depending on the access size and length of the incoming burst access. For instance, in a system where the bus in the core's master side is wider than the bus on the slave side, write combining will allow the core to accept writes with zero wait states and then combine several accesses into one or several larger access. Depending on memory controller implementation this could reduce the time required to move data to external memory, and will reduce the load on the master side bus.

*Table 601.*Access latencies

Access	**Master acc. cycles**	**Slave cycles**	**Delay incurred by performing access over core**
Single read	3	1	1 * clk_{slv} + 3 * clk_{mst}
Burst read with prefetch	2 + (burst length)^{x}	2	2 * clk_{slv} + (2 + burst length)* clk_{mst}
Single write^{xx}	(2)	0	0
Burst write^{xx}	(2 + (burst length))	0	0

^{x} A prefetch operation ends at the address boundary defined by the prefetch buffer's size

^{xx} The core implements posted writes, the number of cycles taken by the master side can only affect the next access.

52.3 General access protection and address translation

52.3.1 Overview

The core provides two types of access protection. The first option is to use a bit vector to implement access restriction on a memory page basis. The second option is to use a page-table to provide access restriction and address translation. Regardless of the protection strategy, the core provides means to assign masters on the IO bus in groups where each group can be associated with a data structure (access restriction vector or page table) in memory. The core can be implemented to support a dynamically configurable page size from 4 to 512 KiB, or a fixed page size of 4 KiB.

When a master on the IO bus initiates an access to be propagated by the core, the core will first look at the incoming master's group assignment setting to determine to which group the master belongs. When the group is known, the core can propagate or inhibit the access based on the group's attributes, or determine the address of the in-memory data structures to use for access checks (and possibly address translation). The in-memory data structure may be cached by the core, otherwise the information will be fetched from main memory.

Once the core has the necessary information to process the incoming access, the access will be either allowed to propagate through the core or, in case the access is to a restricted memory location, be inhibited. If the access is inhibited, the core will issue an AMBA ERROR response to the master if the incoming access is a read access. The core implements posted writes, therefore write operations will not receive an AMBA ERROR response. An interrupt can, optionally, be asserted when an access is inhibited. The AHB failing access register can be configured to log the first or most recent access that was inhibited.

When enabled, the core always checks access permissions when a master initiates an access. For the access protection and translation operation to be effective the masters are required to adhere to the AMBA 2.0 specification and not issue burst transfers that cross a 1 KiB address boundary.

It is possible for masters to access the core's register interface through the core. In this case the core will perform an access to itself on the System AHB bus. For configurations where the core is used to form a bi-directional core, any data structures read by the core must be located on the same bus as the core's master interface. The core cannot access data structures that are placed on the same bus as masters that the core protects against, in other words data structures must be accessible on a slave on the System bus, see figure 156.

52.3.2 Delays incurred from access protection

The time required for the core's master interface to start an access may be delayed by access protection checks. Table 602 below shows the added delays, please refer to section 52.2.12 for a description of delays from the core's bridge operation.

*Table 602.*Access protection check latencies

Protection mode	**Delay in clock cycles on master side**
Disabled	0
Write-protection only and read access	0
Master assigned to group in passthrough or inactive group	1
Access Protection Vector, cache hit	1
Access Protection Vector cache miss, cache disabled/not implemented	Minimum^x 4 clock cycles
IOMMU Protection, cache hit	1
IOMMU Protection, TLB miss, TLB disabled/not implemented	Minimum^x 4 clock cycles

^{x} The core may suffer additional AMBA bus delays when accessing the vector in memory. 4 cycles is the minim time required and assumes that the core is instantly granted access to the bus and that data is delivered with zero wait states.

52.4 Access Protection Vector

The Access Protection Vector (APV) consists of a continuous bit vector where each bit determines the access rights to a memory page. The bit vector provides access restriction on the full 4 GiB AMBA address space. The required size of the bit vector depends on the page size used by the core, see table below:

*Table 603.*Bit vector size vs. page size

Page size	**Bit vector size**
4 KiB	128 KiB
8 KiB	64 KiB
16 KiB	32 KiB
32 KiB	16 KiB
64 KiB	8 KiB
128 KiB	4 KiB
256 KiB	2 KiB
512 KiB	1 KiB

Each group can have a bit vector with a base address specified by a field in the group's Group Control Register. When a master performs an access to the core, the master's group number is used to select one of the available bit vectors. The AMBA access size used to fetch the vector is fixed at implementation time and can be read out from the core's Capability register 1. If the AMBA access size to use is 32-bits (WORD sized) and the page size is 4 KiB, bits 31:17 of the incoming address (HADDR) are used to index a word in the bit vector, and bits HADDR[16:12] are used to select one of the 32 bits in the word. For each increase in AMBA access size (DWORD, 4WORD, 8WORD), one bit less of the physical address is used to index the vector and this bit is instead used to select one specific bit in the data read from memory. Similarly, for each increase in page size one bit less of the physical address is used.

The lowest page is protected by the most significant bit in the bit vector. This means that page 0 is protected by the most significant bit in byte 0 read from the bit vector's base address (using big endian addressing). When performing WORD accesses, the lowest page is protected by bit 31 in the accessed word (using the bit numbering convention used throughout this document).

If the bit at the selected position is '0', the access to the page is allowed and the core will propagate the access. If the selected bit is '1', and the access is an read access, an AMBA ERROR response is given to the master initiating the access. If the selected bit is '1', and the access is a write access, the write is inhibited (not propagated through the core).

52.4.1 Access Protection Vector cache

The core can be implemented with an internal memory that caches the Access Protection Vector. The cache size is configurable at implementation time and depends on a number of parameters that can be read out via Capability registers 0 and 1. If the core has been implemented with IOMMU functionality and a IOMMU Translation Lookaside Buffer (TLB), the RAMs in the APV cache will be shared with the IOMMU TLB.

The cache is implemented as a direct-mapped cache built up of one data RAM and one tag RAM. The number of locations in each RAM is the number of lines in the cache. The width of the data RAM (cache line size) is the same as the size of the AMBA accesses used to fetch the APV from main memory. The width and contents of the tag RAM depends on the number of supported groups, cache line size and number of lines in the cache.

The address used to select a position in the RAMs, called the set address, must have *log2(number of lines in the cache)* bits. The number of address bits taken from the physical address required to uniquely address one position in the bit vector depends on the cache line size. The number of required bits for each allowed cache line size is shown in table 604 below.

*Table 604.*Cache line size vs. physical address bits

Cache line size in bits	**Bits of physical address needed to identify one position depending on page size**							
	4 KiB	**8 KiB**	**16 KiB**	**32 KiB**	**64 KiB**	**128 KiB**	**256 KiB**	**512 KiB**
32	15	14	13	12	11	10	9	8
64	14	13	12	11	10	9	8	7
128	13	12	11	10	9	8	7	6
256	12	11	10	9	8	7	6	5

If the core has support for more than one group, the cache must also be tagged with the group ID. The number of bits required to uniquely select one group is *log2(number of groups)*.

This means that in order to be able to cache the full bit vectors for all supported groups the cache address (set address) must have *log2(number of groups) + (required physical address bits)* address bits. The number of required lines in the cache to be able to hold all vectors is:

*cache lines = (number of groups) * (2^{20} / (cache line size))*

If the cache size is not large enough to hold a copy of each position in the bit vector, part of the physical address and group will be placed in the cache tag RAM instead. If the number of lines in the cache allows keeping a cached data of all positions in all bit vectors, the set address and tag data arrangement shown in table 605 will be used.

For the set address/tag RAM tables below the following values are used:

SB = Set address bits = log2(cache line size)
HB = Required number of bits of physical address = See table 604 above.
GB = Required number of bits to select one group = *log2(number of groups)*

Table 605. Set address bits = (group ID bits) + (Physical address bits)

Set address:

31	(HB+BG-1) HB	(HB-1) 0
Not present	Group ID	Physical address

Contents of Tag RAM:

31	0
Not present	V

0	Valid (V) - Signals that addressed position in cache contains valid data

If the number of lines in the cache allows part of the group ID to be part of the set address, the arrangement will be:

Table 606. Set address bits < (group ID bits) + (Physical address bits)

Set address:

31	SB HB	HB-1 0
Not present	Part of Group ID	Physical address

Contents of Tag RAM:

31		0
Not present	Part of Group ID	V

0	Valid (V) - Signals that addressed position in cache contains valid data

If the number of lines in the cache only allows part of the required physical address to be part of the set address, the arrangement will be:

Table 607. Set address bits < (group ID bits) + (Physical address bits)

Set address:

31	SB 0
Not present	Low bits of physical address

Contents of Tag RAM:

31		HB-SB 1	0
Not present	Group ID	High bits of physical address	V

0	Valid (V) - Signals that addressed position in cache contains valid data

In the first arrangement, where *(set address bits) = (group ID bits) + (physical address bits)*, there will never be a collision in the cache. In the two other arrangements there is not room for all positions in the bit vector(s). This means that a cached copy for one memory page can be replaced with the bit vector for another memory page. Since the physical address is used as the set address, accesses from a master assigned to one group may evict cached bit vector data belonging to another group. This may not be wanted in systems where interference between groups of masters should be minimized. In order to minimize inter-group interference, the core can be implemented with support for using as much of the group ID as possible in the set address, this functionality is called group-set-addressing.

The core has support for group-set-addressing if the CA field in Capability register 0 is non-zero. If the number of set address bits (cache lines) is large enough to cache all bit vectors, the set address and tag RAM arrangement will be as described by table 605. If the number of set address bits will allow the whole group ID to be part of the set address, the arrangement will be:

Table 608. Group set address: Set address bits < (group ID bits) + (Physical address bits)

Set address:

31	SB	GB-1 0
Not present	Low bits of physical address	Group ID

Contents of Tag RAM:

31	1	0
Not present	High bits of physical address	V

0	Valid (V) - Signals that addressed position in cache contains valid data

If only part of the group ID can be used for the set address, the arrangement will be:

Table 609. Group set address: Set address bits < (group ID bits)

Set address:

31	GB-SB-1 0
Not present	Low part of Group ID

Contents of Tag RAM:

31		1	0
Not present	Physical address	High part of group ID	V

0	Valid (V) - Signals that addressed position in cache contains valid data

Group-set-addressing is enabled via the GS field in the core’s Control register.

52.4.2 Constraining the memory area covered by the APV cache

In a typical system, the normal case for an AMBA master core is to perform accesses to main memory. In order to reduce latency, the protection data for these accesses is ideally cached within the core. However, main memory is not likely to occupy the full AMBA address range. If accesses outside a certain access range is expected to be rare, and if it is not critical if these accesses suffer a higher latency, it can be beneficial to restrict the memory range for which the core caches the Access Protection Vector. The benefit of this is that the cache size can be reduced while the same hit rate is kept for the specified memory area, alternatively the hit rate could possibly be increased while keeping the cache size constant.

The core can be configured at implementation to only cache the bit vector for a specified memory range. Capability register 1 contains an address and a mask that describes this area. Bit vector data for the specified memory range will be cached by the core. Bit vector data for accesses made outside the memory range will not be placed in the cache, and will instead be fetched for memory on each access. The impact of having a non-zero mask in Capability register 1 is that for each ‘1’ in the mask, one physical address bit can be removed from the cache set address in the examples given earlier in this section.

52.4.3 Access Protection Vector cache flush operation

If the contents of a vector is modified the core cache must be flushed by writing to the TLB/Cache Flush Register. The TLB/Cache Flush register contains fields to flush the entire cache or to flush the lines belonging to a specified group. In order to flush entries for a specific group, group-set-addressing must be implemented and enabled. Performing a group flush without group-set-addressing may only flush part of the cache and can lead to unexpected behavior.

The core will not propagate any transfers while a cache flush operation is in progress.

52.5 IO Memory Management Unit (IOMMU) functionality

The IOMMU functionality of the core provides address translation and access protection on the full 4 GiB AMBA address space. The size of the address range where addresses are translated is specified by the IOMMU Translation Range (ITR) field in the core's Control register:

*Size of translated address range in MiB = 16 MiB * 2^{ITR}*

The maximum allowed value of the ITR field is eight, which means that the IOMMU can provide address translation to an area of size $16*2^8$ = 4096 MiB, which is the full 32-bit address space. When ITR is set to eight and a page size of 4 KiB is used, bits 31:12 of the incoming IO address are translated to physical addresses, using IO Page Tables entries describes below. Bits 11:0 of the incoming access are propagated through the IOMMU. For each increase in page size one more bit will be directly propagated through the IOMMU instead of being translated.

If ITR is less then eight the most significant bits of the IO address must match the value of the TMASK field in Capability register 2. If an access is outside the range specified by TMASK the access will be inhibited. Table 610 shows the the effect of different ITR values. As an example, with ITR set to 2, the IOMMU will perform address translation for a range that spans 64 MiB. This range will be located at offset TMASK[31:26]. Accesses to addresses that do not have their most significant bits set to match TMASK[31:26] will be inhibited. The table also shows the number of pages within the decoded range and the memory required to hold the translation information (page tables) in main memory. The *pgsz* value is the value of the PGSZ field in the control register.

*Table 610.*Effects of IOMMU Translation Range setting

ITR	Size of translated range	TMASK bits used	Number of pages	Size of page tables
0	16 MiB	TMASK[31:24]	4096 / 2^{pgsz}	16 / 2^{pgsz} KiB
1	32 MiB	TMASK[31:25]	8192 / 2^{pgsz}	32 / 2^{pgsz} KiB
2	64 MiB	TMASK[31:26]	16384 / 2^{pgsz}	64 / 2^{pgsz}KiB
3	128 MiB	TMASK[31:27]	32768 / 2^{pgsz}	128 / 2^{pgsz} KiB
4	256 MiB	TMASK[31:28]	655536 / 2^{pgsz}	256 / 2^{pgsz} KiB
5	512 MiB	TMASK[31:29]	131072 / 2^{pgsz}	512 / 2^{pgsz} KiB
6	1024 MiB	TMASK[31:30]	262144 / 2^{pgsz}	1 / 2^{pgsz} MiB
7	2048 MiB	TMASK[31]	524288 / 2^{pgsz}	2 / 2^{pgsz} MiB
8	4096 MiB	TMASK not used	1048576 / 2^{pgsz}	4 / 2^{pgsz} MiB

52.5.1 IO Page Table Entry

Address translation is performed by looking up translation information in a one-level table present in main memory. Part of the incoming address is used to index the table that consists of IO Page Table Entries. The format of an IO Page Table Entry (IOPTE) is shown in table 611 below.

Table 611. IOMMU Page Table Entry (IOPTE)

31 8	7	6 5 4 3	2	1	0
PPAGE	C	RESERVED	W	V	R

Table 611. IOMMU Page Table Entry (IOPTE)

31:8	Physical Page (PPAGE) - Bits 27:8 of this field corresponds to physical address bits 31:12 of the page. With a 4 KiB page size, PPAGE[27:8] is concatenated with the incoming IO address bits [11:0] to form the translated address. For each increase in page size one bit less of PPAGE is used and one bit more of the incoming IO address is used: this means that with a 16 KiB page size , PPAGE[27:10] will be concatenated with the incoming IO address bits [13:0] to form the translated address. Bits 31:27 of this field are currently discarded by the IOMMU and are present in the data structure for forward compatibility with systems using 36-bit AMBA address space.
7	Cacheable (C) - This field is currently not used by the IOMMU
6:3	RESERVED
2	Writeable (W) - If this field is ‘1’ write access is allowed to the page. If this field is ‘0’, only read accesses are allowed.
1	Valid (V) - If this field is ‘1’ the PTE is valid. If this field is ‘0’, accesses to the page covered by this PTE will be inhibited.
0	RESERVED

When the core has IOMMU protection enabled all, incoming accesses from masters belonging to an active group, which is not in pass-through mode, will be matched against TMASK. If an access is outside the range specified by ITR/TMASK, the access will be inhibited and may receive an AMBA ERROR response (not applicable when the access is a posted write).

If the incoming access is within the range specified by ITR/TMASK, the core will use the incoming IO address to index the page table containing the address translation information for the master/IO address. The core may be implemented with an Translation Lookaside Buffer (TLB) that may hold a cached copy of the translation information. Otherwise the translation information will be fetched from main memory. The base address of the page table to use is given by the Group Configuration register to which the master performing the access is assigned. Please see the register description of the Group Configuration register for constraints on the page table base address. The core will use bits X:Y to index the table, where X depends on the value of the ITR field in the core’s Control register, and Y depends on the page size (Y = 12 + PGSZ field in Control register).

When the core has fetched the translation information (IOPTE) for the accesses page it will check the IOPTE’s Valid (V) and Writeable (W) fields. If the IOPTE is invalid, the access will be inhibited. If the Writeable (W) field is unset and the access is a write access, the access will be inhibited. Otherwise the core will, for a page size of 4 KiB, use the IOPTE field PPAGE, bits 27:8, and bits 11:0 of the incoming IO address to form the physical address to use when the access is propagated by the core (physical address: PPAGE[27:8] & IOADDR[11:0]).

If the valid (V) bit of the IOPTE is ‘0’ the core may or may not store the IOPTE in the TLB (if implemented). This is controlled via the SIV field in the core’s Control register.

52.5.2 Prefetch operations and IOMMU protection

During normal bridge operation, and with Access Protection Vector protection, the core determines if data for an access can be prefetched by looking at the IO address and the System bus plug and play information. This operation cannot be done without introducing additional delays when the core is using IOMMU protection. The incoming IO address must first be translated before it can be determined if the access is to a memory area that can be prefetched. In order to minimize delays the core makes the assumption that any incoming burst access is to a prefetchable area. The result is that when using IOMMU protection all burst accesses will result in the core performing a prefetch operation.

52.5.3 Translation Lookaside Buffer operation

The core can be implemented with an internal memory that caches IO Page Table Entries. This memory is referred to as a Translation Lookaside Buffer (TLB). The TLB size is configurable at implementation time and depends on a number of parameters that can be read out via Capability registers 0 and 2. If the core has been implemented with Access Protection Vector functionality and an APV cache, the RAMs for the APV cache will be shared with the IOMMU TLB.

The TLB is implemented as a direct-mapped cache built up of one data RAM and one tag RAM. The number of locations in each RAM is the number of entries in the TLB. The width of the data RAM (entry size) is the same as the size of the AMBA accesses used to fetch page table entries from main memory. The width and contents of the tag RAM depends on the number of supported groups, entry size and number of entries in the TLB.

The address used to select a position in the RAMs, called the set address, must have *log2(number of entries in the TLB)* bits. The number of address bits taken from the physical address required to uniquely address one position in the TLB depends on the entry size. The number of required bits for each allowed entry size is shown in table 604 below, the values in the third column is the number of address bits that must be used to accommodate the largest translatable range (maximum value of ITR field in the core's Control register). Note that an entry size larger than 32 bits results in an TLB that holds multiple IOPTEs per entry.

*Table 612.*TLB entry size vs. physical address bits

Entry size in bits	**Entry size in IOPTEs**	**Bits of physical address needed to identify one position depending on page size**							
		4 KiB	**8 KiB**	**16 KiB**	**32 KiB**	**64 KiB**	**128 KiB**	**256 KiB**	**512 KiB**
32	1	20	19	18	17	16	15	14	13
64	2	19	18	17	16	15	14	13	12
128	4	18	17	16	15	14	13	12	11
256	8	17	16	15	14	13	12	11	10

If the core has support for more than one group, the TLB entries must also be tagged with the group ID. The number of bits required to uniquely select one group is *log2(number of groups)*.

This means that in order to be able to cache the page tables for all supported groups the TLB address (set address) must have *log2(number of groups) + (required physical address bits)* address bits. The number of required entries in the TLB to be able to hold all vectors is:

*TLB entries = (number of groups) * (2^{20} / (entry size))*

If the TLB is not large enough to hold a copy of each position in the page table, part of the physical address and group will be placed in the tag RAM. The core will implement the TLB depending on the parameters mentioned above. If the number of entries in the TLB allows keeping a copy of all positions in all page tables, the set address and tag data arrangement shown in table 613 will be used.

For the set address/tag RAM tables below the following values are used:

SB = Set address bits = log2(number of TLB entries)
HB = Required number of bits of physical address = See table 612 above.
GB = Required number of bits to select one group = *log2(number of groups)*

Table 613. Set address bits = (group ID bits) + (Physical address bits)

Set address:

31	(HB+BG-1) HB	(HB-1) 0
Not present	Group ID	Physical address

Contents of Tag RAM:

31	0
Not present	V

0	Valid (V) - Signals that addressed position in cache contains valid data

If the number of entries in the TLB allows part of the group ID to be part of the set address, the arrangement will be:

Table 614. Set address bits < (group ID bits) + (Physical address bits)

Set address:

31	SB HB	HB-1 0
Not present	Part of Group ID	Physical address

Contents of Tag RAM:

31		0
Not present	Part of Group ID	V

0	Valid (V) - Signals that addressed position in cache contains valid data

If the number of entries in the TLB only allows part of the required physical address to be part of the set address, the arrangement will be:

Table 615. Set address bits < (group ID bits) + (Physical address bits)

Set address:

31	SB 0
Not present	Low bits of physical address

Contents of Tag RAM:

31		HB-SB 1	0
Not present	Group ID	High bits of physical address	V

0	Valid (V) - Signals that addressed position in cache contains valid data

In the first arrangement, where *(set address bits) = (group ID bits) + (physical address bits)*, there will never be a collision in the TLB. In the two other arrangements there is not room for all entries in the page table(s). This means that a cached IOPTE for one memory page can be replaced with the IOPTE for another memory page. Since the physical address is used as the set address, accesses from a master assigned to one group may evict cached IOPTE's belonging to another group. This may not be wanted in systems where interference between groups of masters should be minimized. In order to minimize inter-group interference, the core can be implemented with support for using as much of the group ID as possible in the set address, this functionality is called group-set-addressing.

The core has support for group-set-addressing if the IT field in Capability register 0 is non-zero. If the number of set address bits (TLB entries) is large enough to cache all page tables, the set address and tag RAM arrangement will be as described by table 613. If the number of set address bits will allow the whole group ID to be part of the set address, the arrangement will be:

Table 616. Group set address: Set address bits < (group ID bits) + (Physical address bits)

Set address:

31	SB	(GB-1) 0
Not present	Low bits of physical address	Group ID

Contents of Tag RAM:

31	1	0
Not present	High bits of physical address	V

0	Valid (V) - Signals that addressed position in cache contains valid data

If only part of the group ID can be used for the set address, the arrangement will be:

Table 617. Group set address: Set address bits < (group ID bits)

Set address:

31	GB-SB-1	0

Table 617. Group set address: Set address bits < (group ID bits)

Not present	Low part of group ID

Contents of Tag RAM:

31			1	0
Not present	Physical address	High part of Group ID		V

0	Valid (V) - Signals that addressed position in cache contains valid data

Group-set-addressing is enabled via the GS field in the core's Control register.

52.5.4 TLB flush operation

If the contents of a page table is modified the TLB must be flushed by writing to the TLB/Cache Flush Register. The TLB/Cache Flush register contains fields to flush the entire TLB or to flush the entries belonging to a specified group. In order to flush entries for a specific group, group-set-addressing must be implemented and enabled. Performing a group flush without group-set-addressing may only flush part of the TLB and can lead to unexpected behavior.

When working in IOMMU mode, the core can be configured to not store a IOPTE in the TLB if the IOPTE's valid (V) bit is cleared. This behavior is controller via the SIV field in the core's Control register.

The core will not propagate any transfers while a flush operation is in progress.

52.6 Fault-tolerance

In order to attain fault-tolerance the core should be implemented with inferred memory technology for the read buffer, write buffer and any FCFS buffer. This will implement the buffers in flip-flops and the core must then be implemented using techniques such as radiation hardened registers or TMR insertion.

The Access Protection Vector cache and IOMMU TLB can be implemented with the same options as above or using non-protected memory cells. When using non-protected memory cells the core can be implemented to use byte-parity to protect entries in the cache/TLB. If an error is detected it will be processed as a cache/TLB miss and the data will be re-read from main memory. A detected error will also be reported via the core's status register and the core also has support for signaling errors via its statistic output.

Errors can be injected in the Access Protection Vector cache and IOMMU TLB via the Data and Tag RAM Error Injection registers.

52.7 Statistics

In order to record statistics, a LEON4 Statistics Unit should be connected to the core. The core has the following statistics outputs:

*Table 618.*IOMMU Statistics

Output	**Description**
hit	High for one cycle during TLB/cache hit.
miss	High for one cycle during TLB/cache miss
pass	High for one cycle during passthrough access
accok	High for one cycle during access allowed
accerr	High for one cycle during access denied
walk	High while core is busy performing a table walk or accessing the access protection vector
lookup	High while core is performing cache lookup/table walk
perr	High for one cycle when core detects a parity error in the APV cache

52.8 Multi-bus bridge

The core can be instantiated in a version with two AHB master interfaces. These interfaces can be connected to separate AHB buses. The top-level entity griommu_mb contains additional signals for the second AHB master interface. Using the griommu_mb entity will enable bus select fields in the core's master configuration registers and the LB field in the core's control register. The bus select fields in the Master configuration registers allows the user to select which AHB master interface that should be used for accesses initiated by a specific master. The control register field LB selects which AHB master interfaces that should be used when the core fetches IOPTEs or APV bit vector data from memory.

52.9 ASMP support

In some systems there may be a need to have separated instances of software each controlling a group of masters. In this case, sharing of the IOMMU register interface may not be wanted as it would allow software to modify the protection settings for a group of masters that belongs to another software instance. The core can be implemented with ASMP support to support systems where software entities are separated by address space. In this case, the core's register interface is mirrored on different 4 KiB pages. Different write protection settings can be set for each mirrored block of registers. This allows use of a memory management unit to control that software running can write to one, and only one, subset of registers.

When ASMP support is enabled, the field NARB in Capability register 0 is non-zero. The value of NARB tells how many ASMP register blocks that are available. Each ASMP register block mirrors the standard register set described in section 52.10 with the addition that some registers may be write protected. Table 619 contains a column that shows if a register is writable when accessed from an ASMP register block. The core's Control register, Master configuration register(s), Diagnostic cache registers, the ASMP access control register(s) can never be written via ASMP register block. These registers are only available in the first register set starting at the core register set base address. ASMP register block *n* is mapped at an offset *n**0x1000 from the core's register base address.

Software should first set up the IOMMU and assign the masters into groups. Then the ASMP control registers should be configured to constrain which registers that can be written from each ASMP block. After this initialization is done, other parts of the software environment can be brought up.

As an example, consider the case where OS A will control masters 0, 1 and 4 while OS B will control masters 2 and 3. In this case it may be appropriate to map masters 0, 1 and 4 to group 0 and master 2 and 3 to group 1. The ASMP access control registers can then be configured to only allow accesses to the Group control register for group 0 from ASMP register block 1 and likewise only allow accesses to the Group control register for group 1 from ASMP register block 2.

OS A will then map in ASMP register block 1 (registers within page located at core base offset + 0x1000) and OS B will then map in ASMP register block 2 (registers within page located at core base offset + 0x2000). This way OS a will be able to change the base address and the properties of group 0, containing its masters, without being able to change the protection mechanisms of group 1 belonging to OS B. Note that since an OS is able to flush the TLB/cache it is able to impact the I/O performance of masters assigned to other OS instances. Also note that care must be taken when clearing status bits and setting the mask register that controls interrupt generation.

52.10 Registers

The core is programmed through registers mapped into AHB I/O address space. All accesses to register address space must be made with word (32-bit) accesses.

*Table 619.*GRIOMMU registers

AHB address offset	**Register**	**Writable in ASMP block**
0x00	Capability register 0	No
0x04	Capability register 1	No
0x08	Capability register 2	No
0x0C	Reserved	-
0x10	Control register	No
0x14	TLB/cache flush register	Yes, protected**
0x18	Status register	Yes, protected**
0x1C	Interrupt mask register	Yes, protected**
0x20	AHB Failing Access register	No
0x24 - 0x3C	Reserved, must not be accessed	-
0x40 - 0x7C	Master configuration registers. Master n configuration register is located at offset 0x40 + n*0x4.	No
0x80-0xBC	Group control registers. Group n’s control register is located at offset 0x80 + n*0x4.	Yes, protected**
0xC0	Diagnostic cache access register	No
0xC4 - 0xE0	Diagnostic cache access data registers 0 - 7	No
0xE4	Diagnostic cache access tag register	No
0xE8	Data RAM error injection register	No
0xEC	Tag RAM error injection register	No
0xF0 - 0xFF	Reserved, must not be accessed	No
0x100 - 0x13F	ASMP access control registers. The control register for ASMP block n is located at offset 0x100+n*0x4.	No

* Register is duplicated in ASMP register block at offset 0x1000 + register offset. The number of ASMP register blocks is given by the NARB field in Capability register 0. ASMP register block *n* starts at offset *n**0x1000. Register is only writable if allowed by the corresponding ASMP access control register field.

Table 620. GRIOMMU Capability register 0

31	30	29	28	27 24	23 20	19	18 17	16	15	14	13	12	11 9	8	7 4	3 0
A	AC	CA	CP	RESERVED	NARB	CS	FT	ST	I	IT	IA	IP	RESERVED	MB	GRPS	MSTS

31	Access Protection Vector (A) - If this bit is ‘1’, the core has support for Access Protection Vector
30	Access Protection Vector Cache (AC) - If this bit is ‘1’, the core has a internal cache for Access Protection vector lookups.
29	Access Protection Vector Cache Addressing (CA): 0: Core only supports standard addressing, group number is used as tag 1: Core supports using group ID as part of cache set address
28	Access Protection Vector Cache Pipeline (CP) - If this bit is set to ‘1’ the core has a pipeline stage added on the APV cache’s address. This means one cycle additional latency.
27:24	RESERVED
23:20	ASMP Register Blocks (NARB) - This field contains the number of ASMP register blocks that the core implements. If this field is non-zero the core has NARB ASMP register blocks with the first block starting at offset 0x1000 and the last block starting at offset NARB*0x1000.

Table 620. GRIOMMU Capability register 0

19	Configurable Page Size (CS) - If this bit is ‘1’ the core supports several page sizes and the size is set via the Control register field PGSZ. If this bit is ‘0’, a fixed page size of 4 KiB is used.
18:17	Fault Tolerance (FT) - “00” - No fault tolerance, “01” - APV cache and/or IOMMU TLB is protected by parity
16	Statistics (S) - If this field is ‘1’ the core collects statistics
15	IOMMU functionality enable (I) - If this bit is ‘1’, the core has support for IOMMU functionality.
14	IOMMU TLB (IT) - If this bit is ‘1’, the core has an IOMMU Translation Lookaside Buffer (TLB)
13	IOMMU Addressing (IA): 0: Core only supports standard addressing, group number is used as tag 1: Core supports using group ID as part of TLB set address
12	IOMMU TLB Address Pipeline (IP) - If this bit is set to ‘1’ the core has a pipeline stage added on the TLB’s address. This means one cycle additional latency.
11:9	RESERVED
8	Multi-bus (MB) - Set to 1 if core is connected to two system buses.
7:4	Number of groups (GRPS) - Number of groups that the core has been implemented to support - 1.
3:0	Numbers of masters (MSTS) - Number of masters that the core has been implemented to support - 1.

Reset value: Implementation dependent

Table 621. GRIOMMU Capability register 1

31 - 20	19 - 16	15 - 8	7 - 5	4 - 0
CADDR	CMASK	CTAGBITS	CISIZE	CLINES

31:20	Access Protection Vector Cacheable Address (CADDR) - If the CMASK field of this register is non-zero the CADDR and CMASK fields specify the base address of the memory area protected by the part of the bit vector that can be cached by the core.
19:16	Access Protection Vector Cacheable Mask (CMASK) - Number of ‘1’s in the Access Protection Vector Cachable mask. If the core is implemented with a Access Protection Vector cache and this value is non-zero, the CMASK field together with the CADDR field specify a memory area protected by a part of the bit vector that can be cached by the core. The CMASK value corresponds to the number of most significant bits of the CADDR field that are matched against the incoming AMBA address when determining if the protection bits for the memory area should be cached. As an example, if CMASK is 1 and CADDR is 0x000, the core will cache protection information for the address range 0x00000000 - 0x7FFFFFFF. With the same mask and CADDR = 0x800, the core would cache protection information for the address range 0x80000000 - 0xFFFFFFFF.
15:8	Access Protection Vector Cache Tag bits (CTAGBITS) - The width in bits of the Access Protection Vector cache’s tags.
7:5	Access Protection Vector Access size (CSIZE) - This field indicates the AMBA access size used when accessing the Access Protection Vector in main memory. This is also the cache line size for the APV cache (if enabled). The values are: 000: 32-bit (4 byte) 001: 64-bit (8 byte) 010: 128-bit (16 byte) 011: 256-bit (32-byte)
4:0	Access Protection Vector Cache Lines (CLINES) - Number of lines in the Access Protection Vector cache. The number of lines in the cache is 2^{CLINES}.

Reset value: implementation dependent

Table 622. GRIOMMU Capability register 2

31 - 24	23 - 20	19 - 18	17 - 16	15 - 8	7 - 5	4 - 0
TMASK	RESERVED	MTYPE	TTYPE	TTAGBITS	ISIZE	TLBENT

31:24	Translation Mask (TMASK) - The incoming IO address bits IOADDR[31:24] must match this field, depending on the setting of the ITR field in the core’s Control register, for an address translation operation to be performed.

Table 622. GRIOMMU Capability register 2

23:20	RESERVED
19:18	IOMMU Type (MTYPE) - Shows IOMMU implementation type. This field is always 0, other values are reserved for future versions of the core. If this field is non-zero, it should trigger a software alert as future versions of the core may not be backward compatible.
17:16	TLB Type (TTYPE) - Show implementation of Translation Lookaside Buffer. This field is always 0, other values are reserved for future versions of the core. If this field is non-zero, it should trigger a software alert as future versions of the core may not be backward compatible.
15:8	TLB Tag bits (TTAGBITS) - The width in bits of the TLB tag.
7:5	IOMMU Access size (ISIZE) - This field indicates the AMBA access size used when accessing page tables in main memory. This is also the line size for the TLB (if enabled). The values are: 000: 32-bit (4 byte) 001: 64-bit (8 byte) 010: 128-bit (16 byte) 011: 256-bit (32-byte)
4:0	TLB entries (TLBENT) - Number of entries in the TLB. The number of entries is 2^{TLBENT}.

Reset value: implementation dependent

Table 623. GRIOMMU Control register

31 - 21	20 - 18	17	16	15 - 12	11	10	9 - 8	7	6	5	4	3	2 - 1	0
RESERVED	PGSZ	LB	SP	ITR	DP	SIV	HPROT	AU	WP	DM	GS	CE	PM	EN

31:21	RESERVED
20:18	Page Size (PGSZ) - The value in this field determines the page size mapped by page table entries and bit vector positions. Valid values are: 000: 4 KiB 001: 8 KiB 010: 16 KiB 011: 32 KiB 100: 64 KiB 101: 128 KiB 110: 256 KiB 111: 512 KiB This field is only writable if the CS field in Capability register 0 is non-zero.
17	Lookup bus (LB) - The value of this bit controls AHB master interface to use for fetching bit vector and/or page table entries from memory when the core has been implemented with support for multiple buses (multiple AHB master interfaces). If this field is ‘0’, the first master interface will be used for vector/table lookups. If this field is ‘1’, the second master interface will be used for lookups. This field is only writable if the MB field in Capability register 0 is non-zero.
16	SPLIT support (SP) - The value of this bit controls if the core can issue AMBA SPLIT responses to masters on the IO bus. If this bit is ‘1’ the core will use AMBA SPLIT responses. If this bit is ‘0’, the core will insert waitstates and not issue AMBA SPLIT responses. This bit is read-only if the core has been implemented with support for only one response mode. If this bit is writable, software must make sure that the IO bus is free and that the core is not handling any ongoing accesses before changing the value of this bit. The core performs rudimentary checks in order to determine if the slave side is idle before changing SPLIT behavior. Therefore AMBA SPLIT responses may not be disabled or enabled immediately after this bit is written.
15:12	IOMMU Translation Range (ITR) - This field defines the size of the address range translated by the core’s IOMMU functionality. The size of the decoded address range is 16 MiB * 2^{ITR} and the decoded memory area is located on an address with the most significant bits specified by the TMASK field in Capability register 2, unless ITR = 8 in which case the whole address space is covered by the translated range.

Table 623. GRIOMMU Control register

11	Disable Prefetch (DP) - When this bit is ‘1’ the core will not perform any prefetch operations. This bit is read only if the core has been implemented without support for prefetching data. During normal operation prefetch of data improves performance and should be enabled (the value of this bit should be ‘0’). Prefetching may need to be disabled in scenarios where IOMMU protection is enabled, which leads to a prefetch operation on every incoming burst access, and when the core is used in bi-directional bridge configurations where dead locks may be resolved by the core dropping prefetch data.
10	Save Invalid IOPTE (SIV) - If this field is ‘1’ the core will save IOPTEs that have their valid (V) bit set to ‘0’ if the core has been implemented with a TLB. If this field is ‘0’ the core will not buffer an IOPTE with valid (V) set to ‘0’ and perform an page table lookup every time the page covered by the IOPTE is accessed. If the value of this field is changed, a TLB flush must be made to remove any existing IOPTEs from the core’s internal buffer. Also if this field is set to ‘0’, any diagnostic accesses to the TLB should not set the IOPTE valid bit to ‘0’ unless the Tag valid bit is also set to ‘0’. This field is only accessible if the core has support for IOMMU protection and is implemented with a Translation Lookaside Buffer (TLB).
9:8	HPROT encoding (HPROT) - The value of this field will be assigned to the AMBA AHB HPROT signal bits 3:2 when the core is fetching protection data from main memory. HPROT(3) signals if the access is cacheable and HPROT(2) signals if the access is bufferable. This field is only used when the core has been implemented with support for Access Protection Vector or IOMMU functionality.
7	Always Update (AU) - If this bit is set to ‘0’ the AHB failing access register will only be updated if the Access Denied (AD) bit in the Status register is ‘0’ when the access is denied. Otherwise the AHB failing access register will be updated each time an access is denied, regardless of the Access Denied (AD) bit’s value.
6	Write Protection only (WP) - If this bit is set to ‘1’ the core will only used the Access Protection Vector to protect against write accesses. Read accesses will be propagated over the core without any access restriction checks. This will improve the latency for read operations. This field has no effect when the core is using IOMMU protection (PM field = “01”). When using IOMMU protection all accesses to the range determined by TMASK and ITR will be checked against the page table, unless the access is from a master that is assigned to an inactive group or a group in pass-through mode.
5	Diagnostic Mode (DM) - If this bit is set to ‘1’ the core’s internal buffers can be accessed via the Diagnostic interface (see Diagnostic cache access register) when the DE field of the Status register has been set by the core. Set this bit to ‘0’ to leave Diagnostic mode. While in this mode the core will not forward any incoming AMBA accesses.
4	Group-Set-addressing (GS) - When this bit is set to ‘1’, the core will use the group number as part of the Access Protection Vector cache set address. This bit can only be set if fields A and CA, or I and IA, of Capability register 0 are non-zero.
3	Cache/TLB Enable (CE) - When this bit is set to ‘1’, the core’s internal cache/TLB is enabled. Note that the core can be implemented without internal cache/TLB. Capability register 0, fields AC and IT show if the core has internal cache.
2:1	Protection Mode (PM) - This value selects the protection mode to use. “00” selects Group Mode and/or Access Protection Vector mode (if available). “01” selects IOMMU mode. This field is read only if the core only has support for one mode setting.
0	Enable (EN) - Core enable. If this bit is set to 1 the core is enabled. If this bit is set to 0 the core is disabled and in pass-through mode. After writing this bit software should read back the value. The change has not taken effect before the value of this bit has changed. The bit transition may be blocked if the core is in diagnostic access mode or otherwise occupied.

Reset value: 0x00000000 if core has support for APV. 0x00000002 if core only supports IOMMU protection.

Table 624. GRIOMMU TLB/cache flush register

31 8	7 4	3 2	1	0
RESERVED	FGRP	RES	GF	F

31:1	RESERVED

Table 624. GRIOMMU TLB/cache flush register

7:4	Flush Group (FGRP) - This field specifies the group to be used for a Group Flush, see GF field below.
3:2	RESERVED
1	Group Flush (GF) - When this bit is written to ‘1’ the cache entries for the group selected by the FGRP field will be flushed. More precisely the core will use the FGRP field as (part of the) set address when performing the flush. This flush option is only available if the core has support for group set addressing (CA field of Capability register 1 is non-zero). This flush option must only be used if the GS bit in the Control register is set to ‘1’, otherwise old data may still be marked as valid in the Access Protection Vector cache or IOMMU TLB. This bit will be reset to ‘0’ when a flush operation has completed. A flush operation also affects the FL and FC fields in the Status register.
0	Flush (F) - When this bit is written to ‘1’ the core’s internal cache will be flushed. This bit will be reset to ‘0’ when a flush operation has completed. A flush operation also affects the FL and FC fields in the Status register.

Reset value: 0x00000000

Table 625. GRIOMMU Status register

31 6	5	4	3	2	1	0
RESERVED	PE	DE	FC	FL	AD	TE

31:6	RESERVED
5	Parity Error (PE) - The core sets this bit to ‘1’ when it detects a parity error in the tag or data RAM of the APV cache. This field is cleared by writing ‘1’ to this position, writes of ‘0’ have no effect.
4	Diagnostic Mode Enabled (DE) - If this bit is set to ‘1’ the core is in Diagnostic Mode where the core’s internal buffers can be accessed via the Diagnostic access registers. While in this mode the core will not forward any incoming AMBA accesses.
3	Flush Completed (FC) - The core sets this bit to ‘1’ when a flush operation completes. This field is cleared by writing ‘1’ to this position, writes of ‘0’ have no effect.
2	Flush started (FL) - The core sets this bit to ‘1’ when a Flush operation has started. This field is cleared by writing ‘1’ to this position, writes of ‘0’ have no effect.
1	Access Denied (AD) - The core denied an AMBA access. This field is cleared by writing ‘1’ to this position, writes of ‘0’ have no effect.
0	Translation Error (TE) - The core received an AMBA ERROR response while accessing the bit vector or page tables in memory. This also leads to the incoming AMBA access being inhibited. Depending on the status of the Control register’s AU field and this register’s AD field this may also lead to an update of the AHB Failing Access register.

Reset value: 0x00000000

Table 626. GRIOMMU Interrupt mask register

31 6	5	4	3	2	1	0
RESERVED	PEI	R	FCI	FLI	ADI	TEI

31:6	RESERVED
5	Parity Error Interrupt (PEI) - If this bit is set to ‘1’ an interrupt will be generated when the PE bit in the Status register transitions from ‘0’ to ‘1’.
4	RESERVED
3	Flush Completed Interrupt (FCI) - If this bit is set to ‘1’ an interrupt will be generated when the FC bit in the Status register transitions from ‘0’ to ‘1’.
2	Flush Started Interrupt (FLI) - If this bit is set to ‘1’ an interrupt will be generated when the FL bit in the Status register transitions from ‘0’ to ‘1’..
1	Access Denied Interrupt (ADI) - If this bit is set to ‘1’ an interrupt will be generated when the AD bit in the Status register transitions from ‘0’ to ‘1’.

Table 626. GRIOMMU Interrupt mask register

0	Translation Error Interrupt (TEI) - If this bit is set to ‘1’ an interrupt will be generated when the TE bit in the Status register transitions from ‘0’ to ‘1’.

Reset value: 0x00000000

Table 627. GRIOMMU AHB failing access register

31 5	4	3 2 1 0
FADDR[31:5]	FW	FMASTER

31:5	Failing Address (FADDR[31:5]) - Bits 31:5 of IO address in access that was inhibited by the core. This field is updated depending on the value of the Control register AU field and the Status register AD field.
4	Failing Write (FW) - If this bit is set to ‘1’ the failed access was a write access, otherwise the failed access was a read access. This field is updated depending on the value of the Control register AU field and the Status register AD field.
3:0	Failing Master (FMASTER) - Index of the master that initiated the failed access. This field is updated depending on the value of the Control register AU field and the Status register AD field.

Reset value: 0x00000000

Table 628. GRIOMMU Master configuration register(s)

31 24	23 12	11 5	4	3 0
VENDOR	DEVICE	RESERVED	BS	GROUP

31: 24	Vendor ID (VENDOR) - GRLIB Plug’n’play Vendor ID of master
23: 12	Device ID (DEVICE) - GRLIB Plug’n’play Device ID of master
11: 5	RESERVED
4	Bus select for master (BS) - Master n’s bus select register is located at register address offset 0x40 + n*0x4. This field specifies the the bus to use for accesses initiated by AHB master n. This field is only available if the MB field in Capability register 0 is non-zero.
3:0	Group assignment for master - Master n’s group assignment field is located at register address offset 0x40 + n*0x4. This field specifies the group to which a master is assigned.

Reset value: 0x00000000

Table 629. GRIOMMU Group control register(s)

31	1	0
BASE[31:2+SIZE]	P	AG

31: 2	Base address (BASE) - Group n’s control register is located at offset 0x80 + n*0x4. This field contains the base address of the data structure for the group. The number of bits writeable in the data structure base address depends on the access size used to fetch entries in the Access Protection Vector and/or the IOMMU page table. The access size is given in the ISIZE and CSIZE Capability register fields. This field is only writable if the core has been implemented with support for Access Protection Vector and/or IOMMU functionality.

Table 629. GRIOMMU Group control register(s)

1	Pass-through (P) - If this bit is set to '1' and the group is active (see bit 0 below) the core will pass-through all accesses made by master in this group and not use the address specified by BASE to perform look-ups in main memory. Note that this also means that the access will pass through untranslated when the core is using IOMMU protection (even if the access is outside the translated range defined by TMASK in Capability register 2). If this bit is set to '0', the core will use the contents in its cache, or in main memory, to perform checks and possibly address translation on incoming accesses. If the core has been implemented without support for Access Protection Vector and IOMMU, this field is disabled.
0	Active Group (AG) - Indicates if the group is active. If this bit is set to '0', all accesses made by masters assigned to this group will be blocked. If the core has been implemented without support for Access Protection Vector and IOMMU, accesses will be propagated if this bit is set to '1'. If the core has been implemented with support for Access Protection Vector and/or IOMMU the core will check the P field of this register and possibly also the in-memory data structure before allowing or blocking the access.

Reset value: 0x00000000

Table 630. GRIOMMU Diagnostic cache access register

31	30	29 - 22	21	20	19	18 - 0
DA	RW	RESERVED	DP	TP	R	SETADDR

31	Diagnostic Access (DA) - When this bit is set to '1' the core will perform a diagnostic operation to the cache address specified by the SETADDR field. When the operation has finished this bit will be reset to '0'.
30	Read/Write (RW) - If this bit is '1' and the A field is set to '1' the core will perform a read operation to the cache. The result will be available in the Diagnostic cache access tag and data register(s). If this bit is set to '0' and the A field is set to '1', the core will write the contents of the Diagnostic cache access tag and data registers to the internal cache.
29:22	RESERVED
21	Data Parity error (DP) - This bit is set to '1' if a parity error has been detected in the word read from the cache's data RAM. This bit can be set even if no diagnostic cache access has been made and it can also be set after a cache write operation. This bit is read-only.
20	Tag Parity error (TP) - This bit is set to '1' if a parity error has been detected in the word read from the cache's tag RAM. This bit can be set even if no diagnostic cache access has been made and it can also be set after a cache write operation. This bit is read-only.
19	RESERVED
18:0	Cache Set Address (SETADDR) - Set address to use for diagnostic cache access. When a read operation has been performed, this field should not be changed until all wanted data has been read from the Diagnostic cache access data and tag registers. Changing this field invalidates the contents of the data and tag registers.

* This register can only be accessed if the core has an internal cache and the DE bit in the Status register is set
Reset value: 0b0000000000UU0UUUUUUUUUUUUUUUUUUU, where U is undefined

Table 631. GRIOMMU Diagnostic cache access data register 0 - 7

31 - 0
CDATAn

31:0	Cache data word n (CDATAn) - The core has 8 Diagnostic cache access data registers. Diagnostic cache access data register n holds data bits [31+32*n:32*n] in the cache line.

* This register can only be accessed if the core has an internal cache and the DE bit in the Status register is set
Reset value: Undefined

Table 632. GRIOMMU Diagnostic cache access tag register

31	0
TAG	V

31:1	Cache tag (TAG) - The size of the tag depends on cache size. The contents of the tag depends on cache size and addressing settings.
0	Valid (V) - Valid bit of tag

* This register can only be accessed if the core has an internal cache and the DE bit in the Status register is set
Reset value: Undefined

Table 633. GRIOMMU Data RAM error injection register

31 0
DPERRINJ

31:0	Data RAM Parity Error Injection (DPERRINJ) - Bit DPERRINJ[n] in this register is XOR:ed with the parity bit for data bits [*7+8*n*:*8*n*] in the data RAM.

* This register can only be accessed if the core has an internal cache and the FT field in Capability register 0 is non-zero
Reset value: 0x00000000

Table 634. GRIOMMU Tag RAM error injection register

31 0
TPERRINJ

0	Tag RAM Parity Error Injection (TPERRINJ) - Bit TPERRINJ[n] in this register is XOR:ed with the parity bit for tag bits [*7+8*n*:*8*n*] in the tag RAM.

* This register can only be accessed if the core has an internal cache and the FT field in Capability register 0 is non-zero
Reset value: 0x00000000

Table 635. GRIOMMU ASMP access control register(s)

31 19	18	17	16	15 0
RESERVED	FC	SC	MC	GRPACCSZCTRL

31: 19	RESERVED
18	Flush register access control (FC) - If this bit is set to ‘1’ in the ASMP control register at offset 0x100 + n*0x4 then the TLB/cache flush register in ASMP register block n is writable. Otherwise writes to the TLB/cache flush register in ASMP register block n will be inhibited.
17	Status register access control (SC) - If this bit is set to ‘1’ in the ASMP control register at offset 0x100 + n*0x4 then the Status register in ASMP register block n is writable. Otherwise writes to the Status register in ASMP register block n will be inhibited.
16	Mask register access control (MC) - If this bit is set to ‘1’ in the ASMP control register at offset 0x100 + n*0x4 then the Master register in ASMP register block n is writable. Otherwise writes to the Mask register in ASMP register block n will be inhibited.
15:0	Group control register access control (GRPACCSZCTRL) - ASMP register block n’s group access control field is located at register address offset 0x100 + n*0x4. This field specifies which of the Group control registers that are writable from an ASMP register block. If GRPACCSZCTRL[i] in the ASMP access control register at offset 0x100 + n*0x4 is set to ‘1’ then Group control register i is writable from ASMP register block n.

Reset value: 0x00000000

52.11 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x04F. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

If implemented, the core's second AHB master interface has 0x01 (Aeroflex Gaisler) and device identifier 0x010.

52.12 Implementation

52.12.1 Technology mapping

The core has two technology mapping generics *memtech* and fcfsmtech. *memtech* selects which memory technology that will be used to implement the FIFO memories. *fcfsmtech* selects the memory technology to be used to implement the First-come, first-served buffer, if FCFS is enaled.

52.12.2 RAM usage

The core instantiates one or several *syncram_2p* blocks from the technology mapping library (TECHMAP). If prefetching is enabled max(*mstmaccsz*, *slvaccsz*)/32 *syncram_2p* block(s) with organization (max*(rburst*,*iburst*)-max(*mstmaccsz*, *slvaccsz*)/32) x 32 is used to implement read FIFO (max*(rburst*,*iburst*) is the size of the read FIFO in 32-bit words). max(*mstmaccsz*, *slvaccsz*)/32 *syncram_2p* block(s) with organization *(wburst* - max(*mstmaccsz*, *slvaccsz*)/32) x 32, is always used to implement the write FIFO (where *wburst* is the size of the write FIFO in 32-bit words).

If the core has support for first-come, first-served ordering then one *fcfs* x 4 *syncram_2p* block will be instantiated, using the technology specified by the VHDL generic *fcfsmtech*.

If the core has an Access Protection Vector cache and/or IOMMU TLB, the cache/TLB will be implemented using one *syncramft* block for the tag RAM and one *syncramft* block for the data RAM.

52.13 Configuration options

Table 636 shows the configuration options of the core (VHDL generics).

*Table 636.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
memtech	Memory technology		
iohsindex	Slave I/F AHB index on IO bus	0 to NAHBMST-1	0
syshmindex	Master I/F AHB index on System bus	0 to NAHBMST-1	0
syshmindex2	Master I/F AHB index for second AHB interface. Only available if the entity griommu_mb is instantiated.	0 to NAHBMST-1	0
syshsindex	Index for register slave AHB I/F connected to same bus as core Master I/F	0 to NAHBMST-1	0
syshmaddr	ADDR field of AHB slave BAR 0 on system bus	0 - 16#FFF#	0
syshmask	MASK field of AHB slave BAR 0 on system bus. The requied value of this generic depends on the setting of generic narb (see below).	0 - 16#FFF#	16#FFF#
syshirq	Interrupt line to use for IOMMU interrupts	1 - NAHBIRQ-1	1
dir	0 - clock frequency on the master bus is lower than or equal to the frequency on the slave bus 1 - clock frequency on the master bus is higher than or equal to the frequency on the slave bus (for VHDL generic *ffact* = 1 the value of dir does not matter)	0 - 1	0

*Table 636.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
ffact	Frequency scaling factor between AHB clocks on master and slave buses.	1 - 15	2
slv	Slave bridge. Used in bi-directional bridge configuration where *slv* is set to 0 for master bridge and 1 for slave bridge. When a deadlock condition is detected slave bridge (*slv*=1) will give RETRY response to current access, effectively resolving the deadlock situation. This generic must only be set to 1 for a bridge where the frequency of the bus connecting the master interface is higher or equal to the frequency of the AHB bus connecting to the bridge's slave interface. Otherwise a race condition during access collisions may cause the bridge to deadlock.	0 - 1	0
pfen	Prefetch enable. Enables read FIFO.	0 - 1	0
irqsync	Interrupt forwarding. Forward interrupts from slave interface to master interface and vice versa. 0 - no interrupt forwarding, 1 - forward interrupts 1 - 15, 2 - forward interrupts 0 - 31. Since interrupts are forwarded in both directions, interrupt forwarding should be enabled for one bridge only in a bi-directional AHB/AHB bridge.	0 - 2	0
wburst	Length of write bursts in 32-bit words. Determines write FIFO size and write burst address boundary. If the wburst generic is set to 2 the bridge will not perform write bursts over a 2x4=8 byte boundary. This generic must be set so that the buffer can contain two of the maximum sized accesses that the bridge can handle.	2 - 32	8
iburst	Instruction fetch burst length. This value is only used if the generic *ibrsten* is set to 1. Determines the length of prefetching instruction read bursts on the master side. The maximum of (iburst,rburst) determines the size of the core's read buffer FIFO.	4 - 8	8
rburst	Incremental read burst length. Determines the maximum length of incremental read burst of unspecified length (INCR) on the master interface. The maximum of *rburst* and *iburst* determine the read burst boundary. As an example, if the maximum value of these generics is 8 the bridge will not perform read bursts over a 8x4=32 byte boundary. This generic must be set so that the buffer can contain two of the maximum sized accesses that the bridge can handle. For systems where AHB masters perform fixed length burst (INCRx , WRAPx) *rburst* should not be less than the length of the longest fixed length burst.	4 - 32	8
bar0	Address area 0 decoded by the bridge's slave interface. Appears as memory address register (BAR0) on the slave interface. The generic has the same bit layout as bank address registers with bits [19:18] suppressed (use functions ahb2ahb_membar and ahb2ahb_iobar in gaisler.misc package to generate this generic).	0 - 1073741823	0
bar1	Address area 1 (BAR1)	0 - 1073741823	0
bar2	Address area 2 (BAR2)	0 - 1073741823	0
bar3	Address area 3 (BAR2)	0 - 1073741823	0

*Table 636.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
sbus	The number of the AHB bus to which the slave interface is connected. The value appears in bits [1:0] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	0
mbus	The number of the AHB bus to which the master interface is connected. The value appears in bits [3:2] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	0
ioarea	Address of the I/O area containing the configuration area for AHB bus connected to the bridge's master interface. This address appears in the bridge's slave interface user-defined register 1. In order for a master on the slave interface's bus to access the configuration area on the bus connected to the bridge's master interface, the I/O area must be mapped on one of the bridge's BARs. If this generic is set to 0, some tools, such as Aeroflex Gaisler's GRMON debug monitor, will not perform Plug'n'Play scanning over the bridge.	0 - 16#FFF#	0
ibrsten	Instruction fetch burst enable. If set, the bridge will perform bursts of *iburst* length for opcode access (HPROT[0] = '0'), otherwise bursts of *rburst* length will be used for both data and opcode accesses.	0 - 1	0
lckdac	Locked access error detection and correction. Locked accesses may lead to deadlock if a locked access is made while an ongoing read access has received a SPLIT response. The value of *lckdac* determines how the core handles this scenario: 0: Core will deadlock 1: Core will issue an AMBA ERROR response to the locked access 2: Core will allow both accesses to complete. If the core is used to create a bidirectional bridge, a deadlock condition may arise when locked accesses are made simultaneously in both directions. With *lckdac* set to 0 the core will deadlock. With *lckdac* set to a non-zero value the slave bridge will issue an ERROR response to the incoming locked access.	0 - 2	0
slvmaccsz	The maximum size of accesses that will be made to the bridge's slave interface. This value must equal *mstmaccsz* unless *rdcomb* /= 0 and *wrcomb* /= 0.	32 - 256	32
mstmaccsz	The maximum size of accesses that will be performed by the bridge's master interface. This value must equal *mstmaccsz* unless *rdcomb* /= 0 and *wrcomb* /= 0.	32 - 256	32

*Table 636.*Configuration options (VHDL generics)

Generic	**Function**	**Allowed range**	**Default**
rdcomb	Read combining. If this generic is set to a non-zero value the core will use the master interface's maximum AHB access size when prefetching data and allow data to be read out using any other access size supported by the slave interface. If slvmaccsz > 32 and mstmaccsz > 32 and an incoming single access, or access to a non-prefetchable area, is larger than the size supported by the master interface the bridge will perform a series of small accesses in order to fetch all the data. If this generic is set to 2 the core will use a burst of small fetches. If this generic is set to 1 the bridge will not use a burst unless the incoming access was a burst. Read combining is only supported for single accesses and incremental bursts of unspecified length.	0 - 2	0
wrcomb	Write combining. If this generic is set to a non-zero value the core may assemble several small write accesses (that are part of a burst) into one or more larger accesses or assemble one or more accesses into several smaller accesses. The settings are as follows: 0: No write combining 1: Combine if burst can be preserved 2: Combine if burst can be preserved and allow single accesses to be converted to bursts (only applicable if slv-maccsz > 32) Only supported for single accesses and incremental bursts of unspecified length	0 - 2	0
combmask	Read/write combining mask. This generic determines which ranges that the core can perform read/write com-bining to (only available when rdcomb respectively wrcomb are non-zero). The value given for combmask is treated as a 16-bit vector with LSB bit (right-most) indi-cating address 0x0 - 0x10000000. Making an access to an address in an area marked as '0' in combmask is equivalent to making an access over a bridge with rdcomb = 0 and wrcomb = 0. However, combmask is not taken into account when the core performs a prefetch operation (see pfen generic). When a prefetch operation is initiated, the core will always use the maximum sup-ported access size (when rdcomb /= 0).	0 - 16#FFFF#	16#FFFF#
allbrst	Support all burst types 2: Support all types of burst and always prefetch for wrapping and fixed length bursts. 1: Support all types of bursts 0: Only support incremental bursts of unspecified length See section 52.2.7 for more information. When allbrst is enabled, the core's read buffer (size set via rburst/iburst generics) must have at least 16 slots.	0 - 2	0

*Table 636.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
ifctrlen	Interface control enable. When this generic is set to 1 the input signals *ifctrl.mstifen* and *ifctrl.slvifen* can be used to force the AMBA slave respectively master interface into an idle state. This functionality is intended to be used when the clock of one interface has been gated-off and any stimuli on one side of the bridge should not be propagated to the interface on the other side of the bridge. When this generic is set to 0, the ifctrl.* input signals are unused.	0 - 1	0
fcfs	First-come, first-served operation. When this generic is set to a non-zero value, the core will keep track of the order of incoming accesses and handle the requests in the same order. If this generic is set to zero the bridge will not preserve the order and leave this up to bus arbitration. If FCFS is enabled the value of this generic must be higher or equal to the number of masters that may perform accesses over the bridge.	0 - NAHBMST	0
fcfsmtech	Memory technology to use for FCFS buffer. When VHDL generic *fcfs* is set to a non-zero value, the core will instantiate a 4 bit x *fcfs* buffer to keep track of the incoming master indexes. This generic decides the memory technology to use for the buffer.	0 - NTECH	0 (inferred)
scantest	Enable scan support	0 - 1	0
split	Use AMBA SPLIT responses. When this generic is set to 1 the core will issue AMBA SPLIT responses. When this generic is set to 0 the core will insert waitstates instead and may also issue AMBA RETRY responses. If this generic is set to 0, the *fcfs* generic must also be set to 0, otherwise a simulation failure will be asserted.	0 - 1	1
dynsplit	Dynamic SPLIT responses. If this generic is non-zero the Control register field SP will be writable. This allows software to control if the core should use AMBA SPLIT responses or waitstates on the IO bus. The VHDL generic *split* must be set to 1 if this generic is set to 1.	0 - 1	0
nummst	Number of masters connected to the bus that the core’s slave interface connects to.	1 - NAHBMST-1	1
numgrp	Number of groups	1 - NAHBMST-1	1
stat	Enable statistics outputs	0 - 1	0
apv	Include support for Access Protection Vector (APV). Setting this generic to 1 includes support.	0 - 1	1
apvc_en	Access Protection Vector cache. 0: disabled, 1: enabled.	0 - 1	0
apvc_ways	Number of ways in Access Protection Vector cache	1 - 1	1
apvc_lines	Number of lines in each way of the Access Protection vector cache. The total size of the data cache in bytes will be apvc_ways * tbw_accsz/8 * apvc_lines. This value must be a power of two. If the core is implemented with an IOMMU TLB, the maximum value of this generic and VHDL generic *tlb_num* determines the number of lines in the cache.		16
apvc_tech	Access Protection Vector cache memory technology. This generic decides the technology setting for the cache’s tag and data RAM.	0 - NTECH	0 (inferred)

*Table 636.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
apvc_gseta	Allow use of group ID as part of cache set address. This allows cache addressing scheme 2 to be used. The value 0 disables the use of group ID as part of the cache set address and 1 enables the functionality. If the core is implemented with an IOMMU TLB and VHDL generic *tlb_gseta* is set to non-zero value, this will also enable *apvc_gseta*. This generic may only be set to a non-zero value if VHDL generic *numgrp* > 1.	0 - 1	0
apvc_caddr	If generic *apvc_cmask* is non-zero this generic specifies the base address of the memory area that the core will cache protection information for.The area is specified in the same way as addresses for AHB slaves. To cache protection information for the block 0x40000000 - 0x7FFFFFFF, set this generic to 0x400 and *apvc_cmask* to 0xC00.	0 - 16#FFF#	0
apvc_cmask	Specifies size of memory block for which protection information will be cached by the core. If this generic is zero the core will cache protection information for the full AMBA address range. If this generic is 0x800 the core will cache information for half the AMBA address range, the base address for the cacheable area is specified by *apvc_caddr.*	0 - 16#FFF#	0
apvc_pipe	Insert pipelining registers on APV cache. The master -> group -> cache path may become critical in a design. If there are timing problem on the tag or cache RAM address inputs, set this generic to 1 and suffer one cycle in additional penalty on cache accesses. If the core is implemented with an IOMMU TLB and VHDL generic *tlb_pipe* is set to non-zero value, this will also enable *apvc_pipe*.	0 - 1	0
iommu	Enable IOMMU functionality	0 - 1	0
iommutype	Selects type of IOMMU functionality. Set to 0	0 - 1	0
tlb_num	Number of entries in the IOMMU translation lookaside buffer (TLB). A value of zero here implements the core without a TLB. The width of an entry is determined through the VHDL generic tbw_accsz. The total size of the TLB in bytes will be tbw_accsz/8 * tlb_num. This value must be a power of two. If the core has been implemented with an APV cache, the maximum value of this generic and VHDL generic *apvc_lines* determines the number of entries in the TLB.	0 - 64	0
tlb_type	Selects TLB implementation. Set to 0.	0 - 1	0
tlb_tech	TLB memory technology. This generic decides the technology setting to use for implementing the TLB. In the current version of the bridge this generic and VHDL generic *apvc_tech* must have the same value.	0 - NTECH	0 (inferred)

*Table 636.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
tlb_gseta	Allow use of group ID as part of TLB set address. This allows cache addressing scheme 2 to be used. The value 0 disables the use of group ID as part of the TLB set address and 1 enables the functionality. If the core is implemented with an APV cache and VHDL generic *apvc_gseta* is set to non-zero value, this will also enable *tlb_gseta*. This generic may only be set to a non-zero value if VHDL generic *numgrp* > 1.	0 - 1	0
tlb_pipe	Insert pipelining registers on TLB address. The master -> group -> TLB path may become critical in a design. If there are timing problem on the tag or cache RAM address inputs, set this generic to 1 and suffer one cycle in additional penalty on cache accesses. If the core is implemented with an APV cache and VHDL generic *apvc_pipe* is set to non-zero value, this will also enable *tlb_pipe*.	0 - 1	0
tmask	Translation mask. Specifies the value that the most significant bits of the IO address must have for an address to be translated. Bits 7:0 of this value specified TMASK[31:24]. The default value 0xff is recommended. However, this may not work well if the IO bus has a GRLIB plug'n'play area. Note that tmask must specify an address range that is covered by one of the core's memory bars. Otherwise the core will not be selected by the AHB bus controller when the tmask area is accessed.	0 - 16#ff#	16#ff#
tbw_accsz	AMBA access size to use when fetching entries of the Access Protection Vector and/or the IOMMU page table. This value also sets the Access Protection Vector cache line size and the TLB entry size. This value must not exceed the maximum access size for the AHB master interface.	32 - *mstmaccsz*	32
dpagesz	Support for dynamic page size. If this generic is set to 1 the core will support selecting the page size via the Control register. If this generic is set to 0, the page size is fixed to 4 KiB.	0 - 1	0
ft	Fault tolerance. This setting determines if the APV cache and/or TLB tag and data RAMs should be protected against faults. Possible values are: 0 - disabled, 1 - byte parity	0 - 1	0
narb	Number of ASMP register blocks. The core will be implemented with narb ASMP register blocks. the required syshmask settings for different narb values are: narb 0 : hmask 0xfff, narb 1 : hmask 0xfe, narb 2 -3 : hmask 0xfc, narb 4-7 : hmask 0xf80, narb 8 - 15 : hmask 0xf00		
multiirq	Enable interrupt propagation for second AHB master interface. This generic is only available if the the entity griommu_mb is instantiated. If this generic is set to '1', interrupt propagation, as configured via the irqsync generic, will also be done for the second AHB master interface.	0 - 1	0

52.14 Signal descriptions

Table 637 shows the interface signals of the core (VHDL ports).

*Table 637.*Signal descriptions (VHDL ports)

Signal name	Field	Type	Function	Active
RST		Input	Reset	Low
HCLKSYS		Input	AHB system bus clock	-
HCLKIO		Input	AHB IO bus clock	-
IO_AHBSI	*	Input	AHB slave input signals	-
IO_AHBSO	*	Output	AHB slave output signals	-
IO_AHBPNP	*	Input	AHB master output vector signals (on io/slave i/f side). Used to decode plug'n'play vendor/device ID of masters so that these values can be visible in the Master control register(s).	-
SYS_AHBMI	*	Input	AHB master input signals	-
SYS_AHBMO	*	Output	AHB master output signals	-
SYS_AHBPNP	*	Input	AHB slave input vector signals (on system/master i/f side). Used to decode cachability and prefetchability Plug&Play information on bus connected to the bridge's master interface.	-
SYS_AHBMI2	*	Input	AHB master input signals, second interface. Only available on griommu_mb entity.	-
SYS_AHBMO2	*	Output	AHB master output signals, second interface. Only available on griommu_mb entity.	-
SYS_AHBPNP2	*	Input	AHB slave input vector signals (on system/master i/f side). Used to decode cachability and prefetchability Plug&Play information on bus connected to the bridge's second master interface. Only available on griommu_mb entity.	-
SYS_AHBSI	*	Input	AHB slave input signals	-
SYS_AHBSO	*	Output	AHB slave output signals	-
WLK_AHBMI	*	Input	AHB master input signals	-
WLK_AHBMO	*	Output	AHB master output signals	-
LCKI	slck blck mlck	Input	Used in systems with multiple AHB/AHB bridges (e.g. bi-directional AHB/AHB bridge) to detect deadlock conditions. Tie to "000" in systems with only uni-directional AHB/AHB bus.	High
LCKO	slck blck mlck	Output	Indicates possible deadlock condition	High
STATO (clocked by HCLKSYS)	hit	Output	High for one cycle during TLB/cache hit.	High
	miss	Output	High for one cycle during TLB/cache miss	High
	pass	Output	High for one cycle during passthrough access	High
	accok	Output	High for one cycle during access allowed	High
	accerr	Output	High for one cycle during access OK	High
	walk	Output	High while core is busy performing a table walk or accessing the access protection vector	High
	lookup	Output	High while core is performing cache lookup/ table walk	High
	perr	Output	High for one cycle when core detects a parity error in the APV cache	High

*Table 637.*Signal descriptions (VHDL ports)

Signal name	Field	Type	Function	Active
IFCTRL	mstifen	Input	Enable master interface. This input signal is unused if the VHDL generic *ifctrlen* is 0. If VHDL generic *ifctrlen* is 1 this signal must be set to ‘1’ in order to enable the core’s AMBA master interface, otherwise the master interface will always be idle and will not respond to stimuli on the core’s AMBA slave interface.	High
	slvifen	Input	Enable slave interface. This input signal is unused if the VHDL generic *ifctrlen* is 0. If VHDL generic *ifctrlen* is 1 this signal must be set to ‘1’ in order to enable the core’s AMBA slave interface, otherwise the interface will always be ready and the bridge will not propagate stimuli on the core’s AMBA slave interface to the core’s AMBA master interface.	High

* see GRLIB IP Library User’s Manual

52.15 Library dependencies

Table 638 shows the libraries used when instantiating the core (VHDL libraries).

*Table 638.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

52.16 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.iommu.all;

entity griommu_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ...  -- other signals
    );
end;

architecture rtl of griommu_ex is

  -- AMBA signals, system bus
  signal proc_ahbsi       : ahb_slv_in_type;
  signal proc_ahbso       : ahb_slv_out_vector;
  signal proc_ahbmi       : ahb_mst_in_type;
  signal proc_ahbmo       : ahb_mst_out_vector;

  -- AMBA signals, IO bus
  signal io_ahbsi         : ahb_slv_in_type;
  signal io_ahbso         : ahb_slv_out_vector;
  signal io_ahbmi         : ahb_mst_in_type;
  signal io_ahbmo         : ahb_mst_out_vector;

  signal nolock           : griommu_ctrl_type;
  signal noifctrl         : griommu_ifctrl_type;
  signal dbgifctrl        : griommu_ifctrl_type;
  signal griommu_stato    : griommu_stat_type;

begin

nolock <= griommu_ctrl_none;
noifctrl <= griommu_ifctrl_none


-- Instantiate clock generators and AHBCTRL cores here
....
....
-- GRIOMMU
iommu: griommu
    generic map (
      memtech     => memtech,
      iohsindex   => 0,
      syshmindex  => 4,
      syshsindex  => 4,
      syshaddr    => 16#200#,
      syshmask    => 16#FFE#,
      syshirq     => 1,
      slv         => 0,
      dir         => 1,
      ffact       => 1,
      pfen        => 1,
      wburst      => 8,
      iburst      => 8,
      rburst      => 8,
      irqsync     => 0, -- No interrupt synchronization
      bar0        => ahb2ahb_membar(16#000#, '0', '0', 16#800#),
      bar1        => ahb2ahb_membar(16#800#, '0', '0', 16#800#),
      sbus        => 1,
      mbus        => 0,
      ioarea      => 16#FFF#,
      ibrsten     => 0,
      lckdac      => 0,
      slvmaccsz   => 32,  -- Maximum allowed access size by masters on io bus
      mstmaccsz   => 128, -- Maximum allowed access size on system bus
      rdcomb      => 2,
      wrcomb      => 2B,
      allbrst     => 0,
      ifctrlen    => 0,
      fcfs        => IO_NAHBM*CFG_IOMMU_FCFS,
      fcfsmtech   => 0,
      scantest    => scantest,
      split       => CFG_IOMMU_FCFS,
      nummst      => IO_NAHBM, -- Number of masters to support
      numgrp      => CFG_IOMMU_NUMGRP,
      stat        => CFG_IOMMU_STAT,
      apv         => CFG_IOMMU_APV,
      apv_accsz   => CFG_IOMMU_APVACCSZ,
      apvc_en     => CFG_IOMMU_APVCEN,
      apvc_ways   => 1,                 -- Only valid value
      apvc_lines  => CFG_IOMMU_APVCLINES,
      apvc_tech   => CFG_IOMMU_APVCTECH,
      apvc_gseta  => CFG_IOMMU_APVCGSETA,
      apvc_caddr  => CFG_IOMMU_APVCCADDR,
      apvc_cmask  => CFG_IOMMU_APVCCMASK,
      apvc_pipe   => CFG_IOMMU_APVCPIPE,
      iommu       => CFG_IOMMU_IOMMU,
      iommutype   => CFG_IOMMU_IOMMUTYPE,
      tlb_num     => CFG_IOMMU_TLBNUM,
      tlb_type    => CFG_IOMMU_TLBTYPE,
      tlb_tech    => CFG_IOMMU_TLBTECH,
      tlb_gseta   => CFG_IOMMU_TLBGSETA,
      tlb_pipe    => CFG_IOMMU_TLBPIPE,
      tmask       => 16#ff#,
      tbw_accsz   => CFG_IOMMU_TBWACCSZ,
      ft          => CFG_IOMMU_FT)
    port map (
      rstn        => rstn,
      hclksys     => clkm,
      hclkio      => clkm,
      io_ahbsi    => io_ahbsi,
      io_ahbso    => io_ahbso(0),
      io_ahbpnp   => io_ahbmo(IO_NAHBM-1 downto 0),
      sys_ahbmi   => sys_ahbmi,
      sys_ahbmo   => sys_ahbmo(4),
      sys_ahbpnp  => sys_ahbso,
      sys_ahbsi   => io_ahbsi,
      sys_ahbso   => io_ahbso(4),
      lcki        => nolock,
      lcko        => open,
      stato       => griommu_stato,
      ifctrl      => noifctrl);

end;
```

53 GRPULSE - General Purpose Input Output

53.1 Overview

The General Purpose Input Output interface is assumed to operate in an AMBA bus system where the APB bus is present. The AMBA APB bus is used for control and status handling.

The General Purpose Input Output interface provides a configurable number of channels. Each channel is individually programmed as input or output. Additionally, a configurable number of the channels are also programmable as pulse command outputs. The default reset configuration for each channel is as input. The default reset value each channel is logical zero.

The pulse command outputs have a common counter for establishing the pulse command length. The pulse command length defines the logical one (active) part of the pulse. It is possible to select which of the channels shall generate a pulse command. The pulse command outputs are generated simultaneously in phase with each other, and with the same length (or duration). It is not possible to generate pulse commands out of phase with each other.

Each channel can generate a separate internal interrupt. Each interrupt is individually programmed as enabled or disabled, as active high or active low level sensitive, or as rising edge or falling edge sensitive.

53.1.1 Function

The core implements the following functions:

- Input
- Output
- Output pulse commands
- Input interrupts
- Status and monitoring

53.1.2 Interfaces

The core provides the following external and internal interfaces:

- Discrete input and output interface
- AMBA APB slave interface, with sideband signals as per [GRLIB] including:
  - interrupt bus
  - configuration information
  - diagnostic information

53.2 Registers

The core is programmed through registers mapped into APB address space.

*Table 639.*GRPULSE registers

APB address offset	Register
16#000#	Input Register
16#004#	Output Register
16#008#	Direction Register
16#00C#	Interrupt Mask Register
16#010#	Interrupt Polarity Register
16#014#	Interrupt Edge Register
16#018#	Pulse Register
16#01C#	Pulse Counter Register

53.2.1 Input Register [GpioIN] R

*Table 640.*Input Register

31	30	29	28	27	26	25	24	23 0
								IN

23-0:	IN	Input Data

Note that only bits nchannel-1 to 0 are implemented.

53.2.2 Output Register [GpioOUT] R/W

*Table 641.*Output Register

31	30	29	28	27	26	25	24	23 0
								OUT

23-0:	OUT	Output Data

All bits are cleared to 0 at reset.

Note that only bits nchannel-1 to 0 are implemented.

53.2.3 Direction Register [GpioDIR] R/W

*Table 642.*Direction Register

31	30	29	28	27	26	25	24	23 0
								DIR

23-0:	DIR	Direction: 0b=input, 1b=output

All bits are cleared to 0 at reset.

Note that only bits nchannel-1 to 0 are implemented.

53.2.4 Pulse Register [GpioPULSE] R/W

*Table 643.*Pulse Register

31	30	29	28	27	26	25	24	23 … 0
								PULSE

23-0:	PULSE	Pulse enable: 0b=output, 1b=pulse command output

All bits are cleared to 0 at reset.

Only channels configured as outputs are possible to enable as command pulse outputs.

Note that only bits npulse-1 to 0 are implemented.

53.2.5 Pulse Counter Register [GpioCNTR] R/W

*Table 644.*Pulse Counter Register

31	30	29	28	27	26	25	24	23 … 0
								CNTR

23-0:	CNTR	Pulse counter value

All bits are cleared to 0 at reset.

The pulse counter is decremented each clock period, and does not wrap after reaching zero.

Command pulse channels, with the corresponding output data and pulse enable bits set, are (asserted) while the pulse counter is greater than zero.

Setting CNTR to 0 does not give a pulse.

Setting CNTR to 1 does give a pulse with of 1 Clk period.

Setting CNTR to 255 does give a pulse with of 255 Clk periods.

Note that only bits cntrwidth-1 to 0 need be implemented.

53.2.6 Interrupt Mask Register [GpioMASK] R/W

*Table 645.*Interrupt Mask Register

31 … 24	23 … 16	15 … 0
	MASK	

23-16:	MASK	Interrupt enable, 0b=disable, 1b=enable

Note that only bits that are enabled by the imask VHDL generic and that are in the range nchannel-1 to 0 are implemented.

53.2.7 Interrupt Polarity Register [GpioPOL] R/W

*Table 646.*Interrupt Polarity Register

31 … 24	23 … 16	15 … 0
	POL	

23-16:	POL	Interrupt polarity, 0b=active low or falling edge, 1b=active high or rising edge

Note that only bits that are enabled by the imask VHDL generic and that are in the range nchannel-1 to 0 are implemented.

53.2.8 Interrupt Edge Register [GpioEDGE] R/W

*Table 647.*Interrupt Edge Register

31 24	23 16	15 0
	EDGE	

23-16:	EDGE	Interrupt edge or level, 0b=level, 1b=edge

Note that only bits that are enabled by the imask VHDL generic and that are in the range nchannel-1 to 0 are implemented.

53.3 Operation

53.3.1 Interrupt

Two interrupts are implemented by the interface:

Index:	Name:	Description:
0	PULSE	Pulse command completed
31:0	IRQ	Filtered input interrupt

The PULSE interrupt is configured by means of the *pirq* VHDL generic.

The IRQ interrupts are configured by means of the *imask* and *ioffset* VHDL generics, where *imask* enables individually the input interrupts, and *ioffset* adds an offset to the resulting index on the interrupt bus.

53.3.2 Reset

After a reset the values of the output signals are as follows:

Signal:	Value after reset:
GPIOO.Dout[31:0]	de-asserted
GPIOO.OEn[31:0]	de-asserted

53.3.3 Asynchronous interfaces

The following input signals are synchronized to Clk:

- GPIOI.Din[31:0]

53.4 Vendor and device identifiers

The module has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x037. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

53.5 Configuration options

Table 648 shows the configuration options of the core (VHDL generics).

*Table 648.*Configuration options

Generic name	**Function**	**Allowed range**	**Default**
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRPULSE.	0 - NAHBIRQ-1	1
nchannel	Number of input/outputs	1 - 32	24
npulse	Number of pulses	1 - 32	8
imask	Interrupt mask	0 - 16#FFFFFFFF#	16#FF00#
ioffset	Interrupt offset	0-32	8
invertpulse	Invert pulse output when set	1 - 32	0
cntrwidth	Pulse counter width	4 to 32	20
oepol	Output enable polarity	0, 1	1

53.6 Signal descriptions

Table 649 shows the interface signals of the core (VHDL ports).

*Table 649.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GPIOI	*	Input		-
GPIOO	*	Output		-

* see GRLIB IP Library User’s Manual

53.7 Signal definitions and reset values

The signals and their reset values are described in table 650.

*Table 650.*Signal definitions and reset values

Signal name	**Type**	**Function**	**Active**	**Reset value**
gpio[]	Input/Output	General purpose input output	-	Tri-state

53.8 Timing

The timing waveforms and timing parameters are shown in figure 157 and are defined in table 651.

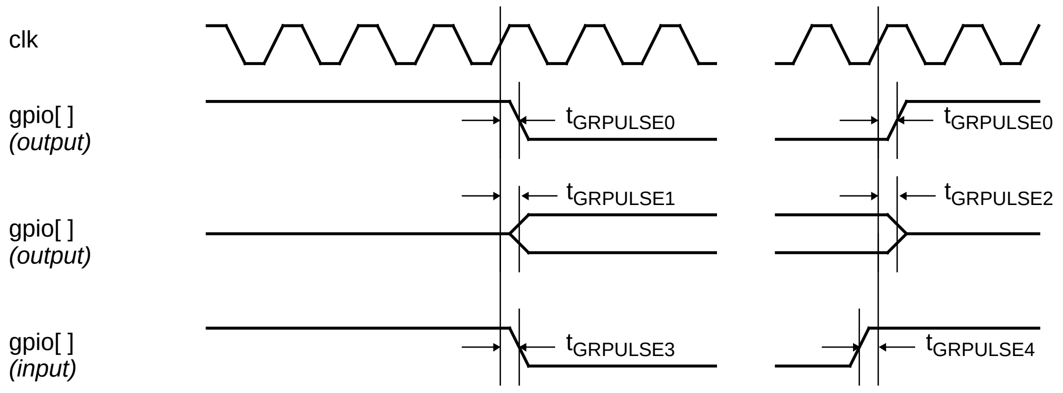


Figure 157. Timing waveforms

*Table 651.*Timing parameters

Name	**Parameter**	**Reference edge**	**Min**	**Max**	**Unit**
$t_{GRPULSE0}$	clock to output delay	rising *clk* edge	-	TBD	ns
$t_{GRPULSE1}$	clock to non-tri-state delay	rising *clk* edge	TBD	-	ns
$t_{GRPULSE2}$	clock to tri-state delay	rising *clk* edge	-	TBD	ns
$t_{GRPULSE3}$	input to clock hold	rising *clk* edge	TBD	-	ns
$t_{GRPULSE4}$	input to clock setup	rising *clk* edge	TBD	-	ns

53.9 Library dependencies

Table 652 shows the libraries used when instantiating the core (VHDL libraries).

*Table 652.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Signals, Components	Signal and component declarations

53.10 Instantiation

This example shows how the core can be instantiated.

```
TBD
```

54 GRPWM - Pulse Width Modulation Generator

54.1 Overview

GRPWM is a pulse width modulation (PWM) generator that supports several outputs, with different frequencies. The core is configured through a set of APB registers, described in section 54.3. The core supports both asymmetric and symmetric PWM generation. Each of the PWM outputs can be configured to be either a single PWM signal or a pair of PWM signals (where the two signals are each others' inverse), with configurable amount of dead band time in between them. The core also supports programming of the output polarity, setting the outputs to fixed values, and configurable interrupt schemes. Hardware support to simplify the generation of a PWM signal that emulate an arbitrary repetitive waveform is also included.

54.2 Operation

54.2.1 System clock scaling

In order to support a wide range of system clock and PWM frequencies the core includes programmable clock scalers. Each scaler is clocked by the system clock and decrement on each clock cycle. When a scaler underflows it is reloaded with the value of its reload register and a tick is generated. This tick can then be used to increment (or decrement) one or more PWM counters. The reload value(s) of the scaler(s) can be read and written through the APB register called *Scaler reload register*, described in section 54.3. The number of system clock scalers is configurable through the VHDL generic *nscalers* and the width of the scaler(s) is determined by the VHDL generic *sbits*.

54.2.2 Asymmetric and symmetric PWM generation

An asymmetric PWM is a pulse signal that is inactive at the beginning of its period and after a certain amount of time goes active, and then stays active for the rest of the period. A symmetric PWM is a pulse signal that is inactive for a certain amount of time at the beginning of the period and a certain amount of time at the end of the period, and stays active in between. The two inactive time periods are normally, but not necessarily, equally long.

For the core to generate a PWM, independent of whether asymmetric or symmetric method is used, software need to do the following (also see section 54.3 for more detailed description of register interface):

- Enable the core by writing the *en* bit in *Core control register.*
- Configure the scaler (see section 54.2.1) and set the PWM period in the *PWM period register.*
- Write the *PWM compare register* with the value at which the PWM's counter should match and switch the outputs.
- If dead band time should be generated, write the value at which the current PWM's dead band time counter should match to the *PWM dead band compare register.* Also set the *dben* bit in the *PWM control register* to 1. See section 54.2.4 for information on dead band time.
- Set the meth bit in PWM control register to either asymmetric och symmetric.
- Set the polarity of the PWM output be setting the *pol* bit in the *PWM control register.*
- If the PWM output should be paired with its inverse then set the *pair* bit in the *PWM control register* to 1, otherwise set it to 0. Note that each PWM always has two ouputs, but if the *pair* bit is set to 0 then the second output is constantly inactive.
- Program the interrupt, see section 54.2.5.
- Enable the PWM generation by writing the *en* bit in *PWM control register* to 1.
- If software wants the PWM output(s) to assume fix value(s) it can write the *fix* bits in the PWM *control register* appropriately.

Specific configuration required for symmetric PWM if dual compare mode should be used:

- If the core should update the PWM's compare register twice every PWM period, then set the *dcomp* bit in the *PWM control register* to a 1.
- If the *dcomp* bit in the *PWM control register* is set, and it is desired that the two inactive time periods are not of equal length, software needs to continuously update the *PWM compare register* with new values. Since the core updates its internal register at the start of and middle of the PWM period, software need to update the *PWM compare register* sometime during the first half of the period.

Note that the core's internal period register is updated from the *PWM period register* at the start of every period, both for asymmetric and symmetric PWM generation.

54.2.3 Waveform PWM generation

That, which in this document is referred to as a *waveform PWM* is not a PWM generated in a different way than the asymmetric or symmetric methods described above. In fact a waveform PWM is either generated asymmetrically or symmetrically. The difference is that when the compare registers are loaded with new values they are read from an internal RAM instead of the *PWM compare register.* The advantage with this is that if software wants to, for example, generate a PWM signals that emulates a sine wave, it can load a number of compare values into the RAM before starting the PWM generation. Once started, the core will read the RAM, increasing the address at every compare match, and generate the same pattern over and over without the need for software intervention. Note that any pattern that is loaded into the RAM is generated, the core is not limited to a sine wave. This feature is supported if the *wpwm* bit in *Capability register 2* is set to 1. The core only support one waveform PWM and it is always the PWM with the highest index. The index is determined by the VHDL generic *npwm*. If for example *npwm* = 4, then it is only PWM four that can be put in waveform mode. For details on how to configure the waveform mode and read/write the RAM please see the description of the *Waveform configuration register, Waveform RAM, w*ord *X* registers, and *PWM control register* in section 54.3.

54.2.4 Dead band time

It is often desired to have a delay between when one of the PWM signals of a PWM pair goes inactive and when the other signal goes active. This delay is called dead band time. By default the core does not generate any dead band time, but can be configured to do so by setting the *dben* bit in the *PWM control register* to 0b1. When dead band time is enabled the core will start a counter each time a PWM pair switch its outputs. The output going inactive is not delayed while the output going active is delayed until the counter matches the value in the *PWM dead band compare register.* To support a wide range of applications the amount of dead band time inserted is programmable. The number of bits used in the *PWM dead band compare register* is configurable through the VHDL generic *dbbits*, and also a four bit system clock scaler can be enabled for each PWM's dead band counter by setting the *dbscaler* VHDL generic to 1.

54.2.5 Interrupts

Interrupts can programmed individually for each PWM to be generated at PWM compare match, at PWM period match, or not generated at all. This is programmed in each PWM's *PWM control register.* Each PWM also has a 6-bit interrupt counter that can be used to scale down the frequency at which the interrupts occur. When an interrupt is generated the bit in the *Interrupt pending register* for the PWM in question is set. The bits in the *Interrupt pending register* stay set until software clears them by writing 1 to them. Through the *sepirq* and *npwm* VHDL generics the core supports several different interrupt numbers, this is described in section *54.6*.

When an interrupt is generated, or when the interrupt scaler counter is increased, an output tick is generated on the core's *tick* output signal. The output tick bit has the same index as the PWM in question.

54.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 653.*GRPWM registers

APB address offset	**Register**
0x00	Core control register
0x04	Scaler reload register
0x08	Interrupt pending register
0x0C	Capability register 1
0x10	Capability register 2
0x14*	Waveform configuration registers
0x18 - 0x1C	Reserved, always zero.
0x20**	PWM period register
0x24**	PWM compare register
0x28**	PWM dead band compare register
0x2C**	PWM control register
0x8000***	Waveform RAM, word 0
0x8004***	Waveform RAM, word 1
...	...
0xFFFC***	Waveform RAM, word 8191

* This register is only implemented if the wpwm bit (bit 0) in *Capability register 2* is set to 1.

** This register is implemented once for every PWM (value of *npwm* VHDL generic decides the number of registers), with an offset of 0x10 from the previous PWM’s register. The functionality is the same for each PWM.

*** The implementation of this register depends on if the *wpwm* bit (bit 0) in *Capability register 2* is set to 1 and if the waveform RAM is large enough (the value of the field *wabits* in *Capability register 2* reports the number of address bits - 1 that is used for the waveform RAM).

Table 654. Core control register

31 x+13	x+12 12	11	10 8	7 1	0
R	noup	R	scalersel	R	en

31:x+13	Reserved, always zero. x is the value of bits 2:0 in *Capability register 1*
12+x:12	No update bits for each PWM. x is the value of bits 2:0 in *Capability register 1*. Bit 12 is for the first PWM, bit 13 for the second etc. If a bit is set to 0b1 then that PWM’s internal period register, compare register, and dead band compare registers are not updated from the corresponding APB registers. These bits can be used by software if it wants to change more than one of the values and it is required that all values change in the same PWM period. It can also be used to synhronize the use of new values for different PWMs. Reset value 0b0..0.
11	Reserved, always zero.
10:8	System clock scaler select bits. These bits determine which of the implemented system clock scalers’ reload value that can be read/written from the *Scaler reload register.*These bits are only present if the *nscalers* generic is greater than 1. Reset value is 0b000
7:1	Reserved, always zero.
0	Core enable bit. 0b0 = Core is disabled, no operations are performed and all outputs are disabled. 0b1 = Core is enabled, PWM outputs can be generated. Reset value is 0b0.

Table 655. Scaler reload register

31 sbits	sbits-1 0
R	reload

Table 655. Scaler reload register

31:sbits	Reserved, always zero. If sbits = 32 then this field is not present. (sbits is the value of the *sbits* generic)
(sbits-1):0	The value of this field is used to reload the system clock scaler when it underflows. If the core is configured with more than one scaler (*nscalers* generic greater than 1) then the *scalersel* bits in the *Core control register* determine which of the scalers that is read/written. Reset value is 0b1..1 (all ones).

Table 656. Interrupt pending register

31 npwm	npwm-1 0
R	irq pending

31:npwm	Reserved, always zero.
(npwm-1):0	Interrup pending bits for the PWM(s). When an interrupt event for a specific PWM occurs the core sets the corresponding bit in the interrupt pending register and generates an interrupt. Software can read this register to see which PWM that generated the interrupt. The bits are cleared by writing 1 to them. Reset value is 0b0..0 (all zeroes).

Table 657. Capability register 1

31 29	28	27	26 25	24	23	22	21	20 16	15 13	12 8	7 3	2 0
R	def-pol	dcm ode	sepirq	R	sym pwm	asyp wm	dbsc aler	dbbits	nscalers	sbits	pbits	npwm

31:29	Reserved, always zero.
28	0 = Default polarity is active low (outputs are high after reset/power-up). 1 = Default polarity is active high (outputs are low after reset/power-up).
27	0 = Dual compare mode not implemented. 1 = Dual compare mode implemented.
26:25	Reports interrupt configuration. Value of *sepirq* VHDL generic. Read only.
24	Reserved, always zero.
23	0 = Symmetric PWM generation is not implemented. 1 = Symmetric PWM generation is implemented. Value of *sympwm* VHDL generic. Read only.
22	0 = Asymmetric PWM generation is not implemented. 1 = Asymmetric PWM generation is implemented. Value of *asympwm* VHDL generic. Read only.
21	0 = Dead band time scaler(s) is not implemented. 1 = Dead band time scaler(s) is implemented. Value of *dbscaler* VHDL generic. Read only.
20:16	Reports number of bits, -1, for the PWM's dead band time counters. Value of the *dbbits* VHDL generic - 1. Read only.
15:13	Reports number of implemented scalers, -1. Value of the *nscalers* VHDL generic - 1. Read only.
12: 8	Reports number of bits for the scalers, -1. Value of the *sbits* VHDL generic - 1. Read only.
7:3	Reports number of bits for the PWM counters, -1. Value of the *pbits* VHDL generic - 1. Read only.
2:0	Reports number of implemented PWMs. Value of the *npwm* VHDL generic - 1. Read only.

Table 658. Capability register 2

31 11	10	9 6	5 1	0
R	wsync	wabits	wdbits	wpwm

31:11	Reserved, always zero
10	1 if Waveform PWM synch signal generation is implemented, 0 if not. Value of VHDL generic *wsync*. Read only
9:6	Reports the number of address bits - 1 used for the waveform RAM. Value is log2(*wdepth*) - 1, where *wdepth* is the VHDL generic *wdepth.* Read only.
5:1	Reports number of bits -1 for each word in the waveform RAM. Value of VHDL generic *wbits* - 1. Read only
0	1 if waveform PWM generation is implemented, 0 if not. Value of VHDL generic *wavepwm*. Read only

Table 659. Waveform configuration register

31 30	29	28 wabits+17	wabits+16 16	15 wabits+1	wabits 0
wsynccfg	wsen	R	wsynccomp	R	wstopaddr

31:30	These bits are used to configure at which point in the PWM period matching the *wsynccomp* field (see description below) that the *wsync* ouput will be set high. 0b00 = The wsync ouput will be set at the start of the PWM period. 0b01 = The output will be set at the first compare match. 0b10 = If the *meth* bit in the *PWM Control Register* is set to one (symmetric) for the waveform PWM, then the output will be set at the middle of the PWM period. 0b11 = If the *meth* bit is set to one for the waveform PWM, then the output will be set at the second compare match.
29	Enables/disables the waveform sync signal. This bit is only present if the *wsync* bit in *Capability register 2* is set to 1. Reset value is 0b0
28:wabits+17	Reserved, always zero. wabits is the value of the *wabits* field in *Capability register 2.* Note that this field is not present if wabits is 12.
16+wabits:16	wabits is the value of the *wabits* field in *Capability register 2.* The number of words in the waveform RAM is the same as the maximum number of PWM periods that will occur before the waveform is restarted. The value of this field is used as an offset into the waveform PWM. A counter is increased every PWM period and when the counter matches this value the *wsync* output of the core will be set to 1 sometime during that period. These bits are only present if the *wsync* bit in *Capability register 2* is set to 1. Reset value is 0b0..0.
15:wabits+1	Reserved, always zero.
wabits:0	The value of this field is used by the core to wrap when accessing the waveform RAM. wabits is the value of the *wabits* field in *Capability register 2*. This field is reset to 0b1..1 (all ones) so that by default the core reads the whole RAM. If software wants to put a waveform in the RAM that does not fill the whole RAM it should set these bits to the address where the last waveform PWM compare value will be stored.

Table 660. PWM period register

31 pbits	pbits-1 0
R	per

31:pbits	Reserved, always zero. If pbits = 32 then this field is not present. (pbits is the value of the *pbits* generic)
(pbits-1):0	When the PWM counter reaches this value a PWM period has passed. Depending on the method used to generate the PWM the output could then be switched. When this register is written the actual PWM period value used inside the core is not updated immediately, instead a shadow register is used to hold the new value until a new PWM period starts. Reset value 0b0..0 (all zeroes).

Table 661. PWM compare register

31 pbits	pbits-1 0
R	comp

31:pbits	Reserved, always zero. If pbits = 32 then this field is not present. (pbits is the value of the *pbits* generic)
(pbits-1):0	When the PWM counter reaches this value the PWM output is switched. Depending on the method used to generate the PWM this register is used once or twice during each PWM period. When this register is written the actual PWM compare value used inside the core is not updated immediately, instead a shadow register is used to hold the new value until a new PWM period starts. Reset value 0b0..0 (all zeroes).

Table 662. PWM dead band compare register

31 dbbits	dbbits-1 0
R	dbcomp

Table 662. PWM dead band compare register

31:dbbits	Reserved, always zero. If dbbits = 32 then this field is not present. (dbbits is the value of the *dbbits* generic)
(dbbits-1):0	The dead band time has passed once the dead band counter reach the value of this field. When this register is written the actual compare value used inside the core is not updated immediately, instead a shadow register is used to hold the new value until a new PWM period starts. Reset value 0b0..0 (all zeroes).

Table 663. PWM control register

31 27	26	25 22	21	20 15	14	13	12 10	9	8	7	6	5 3	2	1	0
R	flip	dbscaler	dben	irqscaler	irqt	irqen	scalersel	wen	dcen	R	meth	fix	pair	pol	en

31:27	Reserved, always zero.
26	Output flip bit. When this bit is set to 0b1 the PWM outputs are flipped.
25:22	Dead band scaler. These bits are used to scale the system clock when generating dead band time. This field is only present if the *dbscaler* generic is set to 1. When these bits are written the dead band scaler register inside the core is not updated immediately. Instead these bits are written to a reload register which updates the actual scaler when it underflows. This is done in order to prevent the dead band scaler register to change during the actual dead band time. Reset value is 0b0..0 (all zeroes).
21	Dead band enable. 0b0 = Dead band time generation is disabled, no dead band time will be inserted when the PWM output switch from deactive to active. 0b1 = Dead band time will be inserted when the PWM output switch from deactive to active. Reset value is 0b0.
20:15	Interrupt scaler. Determines how many compare/period matches that need to occur before an interrupt is generated. All zeroes means that an interrupt will occur every compare/period match, a one means that an interrupt will occur every second match etc. Note that when generating a symmetric PWM two compare matches occur during a PWM period but when generating an asymmetric PWM only one compare match occur during a period. Reset value is 0b0..0 (all zeroes).
14	Interrupt type. 0b0 = Generate interrupt on PWM period match. 0b1 = Generate interrupt on PWM compare match. Reset value is 0b0.
13	Interrupt enable/disable bit. 0b0 = Interrupt is disabled. 0b1 = Interrupt is enabled. Reset value is 0b0.
12:10	Scaler select bits. These bits are used to select which of the system clock scalers that will be used when generating the current PWM. This field is only present when the *nscalers* generic is greater than 1. These bits can only be set if the PWM is disabled, i.e. *en* bit (see below) set to 0b0. Reset value is 0b000.
9	Waveform PWM enable. This bit can only be set if the current PWM is the PWM with the highest index (determined by the generic *npwm*) and if the *wavepwm* field in *Capability register 2* is set to 1. Also the PWM need to be disabled, i.e. *en* bit (see below) set to 0b0. When this bit is set the core will reload the internal PWM compare registers with values from the waveform RAM instead of values from the *PWM compare register.* Reset value is 0b0.
8	Dual compare mode enable. If this bit is set to 0b1 and the *meth* bit (see below) is set to 0b1 (symmetric) then the core will update its internal PWM compare register twice every PWM period, once when the counter is zero and once when a period match occur and the counter starts counting downwards again. In this way it is possible to have two different compare values, one when counter is counting upwards and one when counter is counting downwards. If this bit is 0b0 the compare register is only updated when the counter is zero. This bit has no effect if an asymmetric PWM is generated. Reset value is 0b0. This bit is only present if the *dcmode* bit in the *Capability register* is set.
7	Reserved, always zero.
6	PWM generation method select bit. This bit selects if an asymmetric or symmetric PWM will be generated, where 0b0 = asymmetric and 0b1 = symmetric. . The asymmetric and symmetric methods are only available if the generics *asympwm* and *sympwm* respectively are set to 1. This bit can only be set if the PWM is disabled, i.e. *en* bit (see below) set to 0b0. The core prevents software from setting this bit to an invalid value. Reset value is 0b0 if asymmetric PWM is supported otherwise 0b1.
5:3	PWM fix value select bits. These bits can be used to set the PWM output to a fix value. If bit 3 is set to 0b1 then bit 4 decides what value the PWM output will have. If the *pair* bit (see below) is set to 0b1 while bit 3 is set to 0b1 as well then bit 5 determines what value the complement output will have. Reset value is 0b000.

Table 663. PWM control register

2	PWM pair bit. If this bit is set to 0b1 a complement output for this PWM will be generated, creating a PWM pair instead of a single PWM. The complement output will be the first ouput's inverse, with the exception that dead band time might be added when the values switch from deactive to active. Reset value is 0b1.
1	PWM polarity select bit. 0b0 = PWM is active low, 0b1 = PWM is active high. This bit can only be set if the PWM is disabled, i.e. *en* bit (see below) set to 0b0. Reset value equals *defpol* bit in *Capability Register 1*.
0	PWM enable/disable bit. 0b0 = PWM is disabled. 0b1 = PWM is enabled. When this bit is set to 1 (from 0) and the *wen* bit (see bit 9 above) is set the core's internal address counter for the waveform RAM is reset. Reset value is 0b0.

Table 664. Waveform RAM, word X

31 … wbits+1	wbits … 0
R	waveform data

31:wbits+1	Reserved, always zero. wbits is the value of the *wdbits* field in *Capability register 2.* Note that this field is not present if wbits = 31.
wbits:0	wbits is the value of the wdbits field in *Capability register 2*. Data in the waveform RAM at the address which the current register maps to can be read/written through these bits. This register can only be read/written if either the *wen* bit or *en* bit in the associated PWM's PWM control register are set to 0.

54.4 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x04A. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

54.5 RAM usage

The core maps all usage of RAM on the *syncram* (or *syncramft* if *ft* generic is not set to 0) component from the technology mapping library (TECHMAP). RAM is only used if the core is configured with support for generation of a waveform PWM (*wavepwm* generic set to 1). The size of the instantiated RAM is determined by the *wbits* and *wdepth* generics. *wdepth* is the number of words that the RAM can hold, and *wbits* is the number of bits in each word. Fault tolerance - byte parity DMR or TMR - can be added to the RAM by setting the *ft* generic to 1 or 2. Note that the *ft* generic need to be set to 0 if the core is used together with the GPL version of GRLIB, since that version does not support any fault tolerance.

54.6 Configuration options

Table 665 shows the configuration options of the core (VHDL generics).

*Table 665.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR. Need to be set to 16#F00# or smaller if the generic *wavepwm* is set to 1.	0 - 16#FFF#	16#F00#
pirq	APB irq number.	0 - NAHBIRQ-1	0
memtech	Memory technology used for waveform buffer. This generic has no impact if *wavepwm* is 0.	0 - NTECH	inferred
npwm	Number of PWM outputs.	1 - 8	3
pbits	Number of bits used for each PWM.	1 - 32	16

*Table 665.*Configuration options

Generic name	Function	Allowed range	Default
sbits	Number of bits in the system clock scaler(s).	1 - 32	16
nscalers	Number of system clock scalers.	1 - 8	1
dbbits	Number of bits used for the dead band configuration for each PWM.	1 - 32	8
dbscaler	Decides if a scaler is implemeted for the dead band configuration for each PWM. 1 = A four bit system clock scaler is implemented for each PWM. 0 = No scaling of the system clock when calculating the dead band time is implemented.	0 - 1	0
asympwm	Decides if assymetric PWM generation is implemented. This generic can not be set to 0 if the *sympwm* generic is set to 0 as well.	0 - 1	1
sympwm	Decides if symmetric PWM generation is implemented. This generic can not be set to 0 if the *asympwm* generic is set to 0 as well.	0 - 1	1
dcmode	Enables dual compare mode. Core then supports updates of the PWM's compare registers twice during every (symmetric) PWM period. This generic has no effect if sympwm is set to 0.	0 - 1	0
wavepwm	Decides if the core implements support for generating a waveform PWM. If this generic is set to 1 then a RAM block of size *wdepth***wbits* bits will be instantiated.	0 - 1	1
wbits	The number of bits in each of the wdepth words in the internal RAM that holds the waveform. This generic has no impact if *wavepwm* is 0. This generic can not be larger than the *pbits* generic.	1 - 32	8
wdepth	The number of *wbits* wide words that need to fit in the internal buffer holding the waveform. If *wdepth* is not a power of two then the actual number of words that will fit in the buffer is the closest power of two above *wdepth*. This generic has no impact if *wavepwm* is 0.	1 - 8192	512
wsync	If this generic is set the core supports the generation of a synchronization signal. The synchronization signal can be configured to go active any time during the waveform PWM. This generic has no impact if *wavepwm* is 0.	0 - 1	0
sepirq	0 = One irq number (value of *pirq* generic) is used for all PWMs. 1 = Each PWM has it's own irq number, starting with the value of *pirq* and counting up to *pirq*+(*npwm*-1). 2 = The interrupt configuration depend on the *npwm* generic in the following way: If *npwm* < 3, each PWM has its own irq (*pirq* and possibly *pirq*+1). If *npwm* = 3 the PWMs share irq (*pirq*). If 3 < *npwm* < 6 the first three PWMs share irq (*pirq*) and the remaining PWM(s) have their own irq (*pirq*+1 and possibly *pirq*+2). If *npwm* >= 6 the first three PWMs share irq number *pirq*, the second three PWMs share irq number *pirq*+1, and (if implemented) the last two PWMs have their own irq (*pirq*+2 and *pirq*+3).	0 - 2	0
ft	This generic determines if fault tolerance should be added to the RAM that holds the waveform PWM. This generic has no impact if *wavepwm* is 0. 0 = no fault tolerance, 1 = Byte parity DMR, 2 = TMR. Note that this generic need to be set to 0 if the core is used together with the GPL verison of GRLIB, since that version does not include any fault tolerance.	0 - 2	0

*Table 665.*Configuration options

Generic name	Function	Allowed range	Default
defpol	This generic sets the default polarity of the PWM outputs. 0 = Active low polarity, outputs are high after reset/ power-up. 1 = Active high polarity, outputs are low after reset/power-up.	0 - 1	1

54.7 Signal descriptions

Table 666 shows the interface signals of the core (VHDL ports).

*Table 666.*Signal descriptions

Signal name	Field	Type	Function	Active
rst	N/A	Input	Reset	Logical 0
clk	N/A	Input	Clock	-
apbi	*	Input	APB slave input signals	-
apbo	*	Output	APB slave output signals	-
o	pwm(x:0)**	Output	PWM signals	***
	wavesync****	Output	Waveform PWM synchronization signal	Logical 1
	tick(y:0)*****	Output	PWM synchronization tick outputs	Logical 1

* see GRLIB IP Library User’s Manual

** The width depends on core configuration in the following way: x = <number of PWMs>*2-1 (<number of PWMs> = value of VHDL generic *npwm*)

*** Depends on core configuration.

**** Signal is only driven if the waveform PWM and and waveform sync functionality are implemented (VHDL generics *wavepwm* and *wsync* need to be set to 1).

***** The width depends on core configuration in the following way: y = <number of PWMs>-1 (<number of PWMs> = value of VHDL generic *npwm*)

54.8 Signal definitions and reset values

The signals and their reset values are described in table 667.

*Table 667.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
pwm(x:0)*	Output	PWM signals	**	**
wavesync***	Output	Optional synchronization signal	Logical 1	Logical 0
tick(y:0)****	Output	PWM synchronization tick outputs	Logical 1	Logical 0

* The width depends on core configuration in the following way: x = <number of PWMs>*2-1 (<number of PWMs> = value of VHDL generic *npwm*)

** Depends on core configuration.

*** Signal is only driven if the waveform PWM and and waveform sync functionality are implemented (VHDL generics *wavepwm* and *wsync* need to be set to 1).

**** The width depends on core configuration in the following way: y = <number of PWMs>-1 (<number of PWMs> = value of VHDL generic *npwm*)

54.9 Library dependencies

Table 668 shows the libraries used when instantiating the core (VHDL libraries).

*Table 668.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	PWM	Signals, component	Component declaration
TECHMAP	GENCOMP	Constants, components	Components etc. for technology mapping.

54.10 Timing

The timing waveforms and timing parameters are shown in figure 158 and are defined in table 669.

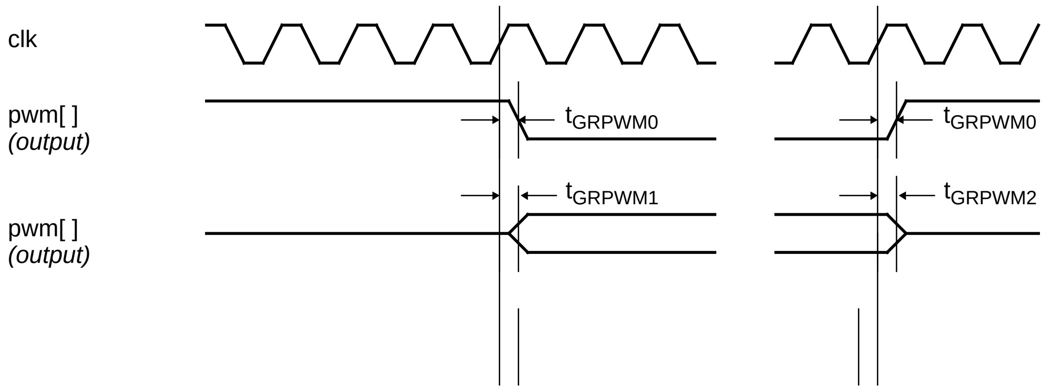


Figure 158. Timing waveforms

*Table 669.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{GRPWM0}	clock to output delay	rising *clk* edge	0	20	ns
t_{GRPWM1}	clock to non-tri-state delay	rising *clk* edge	-	-	ns
t_{GRPWM2}	clock to tri-state delay	rising *clk* edge	-	-	ns

54.11 Instantiation

This example shows how the core can be instantiated. The instantiated core has all its generics, except *pindex*, *paddr*, and *pirq* at their default values. The impact of the generics can be seen in table 665.

```
library ieee, grlib, gaisler;
use ieee.std_logic_1164.all;
use grlib.amba.all;
use gaisler.pwm.all;

entity grpwm_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;
    pwm : out std_logic_vector(5 downto 0)
    );
end;

architecture rtl of grpwm_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- GRPWM signals
  signal pwm : grpwm_out_type;

begin

  -- AMBA Components are instantiated here
  ...

  -- GRPWM core
  grpwm0 : grpwm
    generic map (pindex => 10, paddr => 10, pirq => 10)
    port map (rstn, clk, apbi, apbo(10), pwm);

  -- Pads for GRPWM core
  pwm_pad : outpadv generic map (tech => padtech, width => 6)
                    port map (pwmo, pwm.pwm);

end;
```

55 GRSPW - SpaceWire codec with AHB host Interface and RMAP target

55.1 Overview

The SpaceWire core provides an interface between the AHB bus and a SpaceWire network. It implements the SpaceWire standard (ECSS-E-ST-50-12C) with the protocol identification extension (ECSS-E-ST-50-51C). The optional Remote Memory Access Protocol (RMAP) target implements the ECSS standard (ECSS-E-ST-50-52C).

The core is configured through a set of registers accessed through an APB interface. Data is transferred through DMA channels using an AHB master interface.

Currently, there is one DMA channel but the core can easily be extended to use separate DMA channels for specific protocols. The core can also be configured to have either one or two ports.

There can be up to four clock domains: one for the AHB interface (system clock), one for the transmitter and one or two for the receiver depending on the number of configured ports. The receiver clock can be twice as fast and the transmitter clock four times as fast as the system clock whose frequency should be at least 10 MHz.

The core only supports byte addressed 32-bit big-endian host systems.

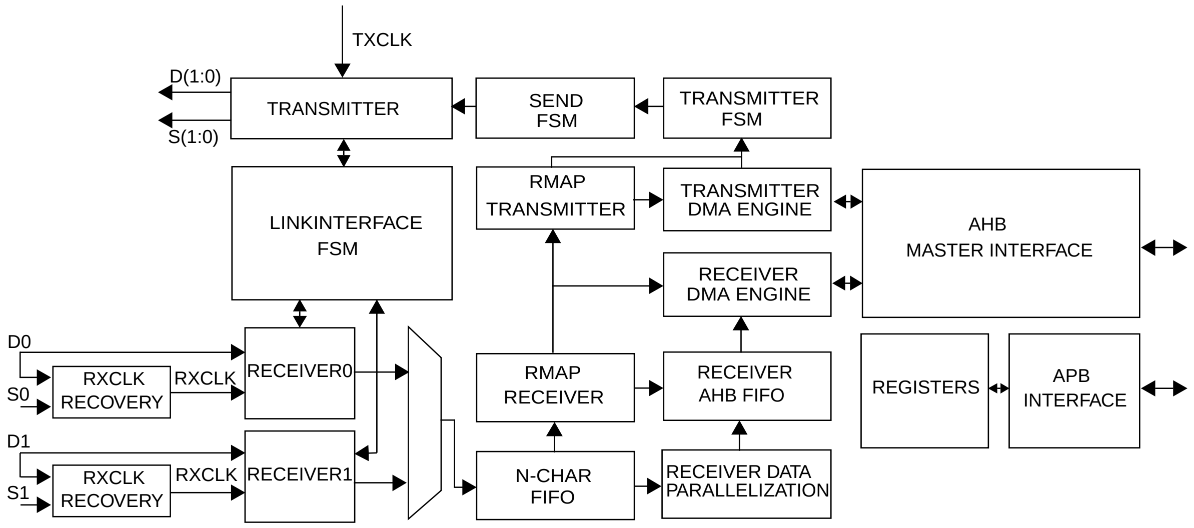


Figure 159. Block diagram

55.2 Operation

55.2.1 Overview

The main sub-blocks of the core are the link-interface, the RMAP target and the AMBA interface. A block diagram of the internal structure can be found in figure 159.

The link interface consists of the receiver, transmitter and the link interface FSM. They handle communication on the SpaceWire network. The AMBA interface consists of the DMA engines, the AHB master interface and the APB interface. The link interface provides FIFO interfaces to the DMA engines. These FIFOs are used to transfer N-Chars between the AMBA and SpaceWire domains during reception and transmission.

The RMAP target is an optional part of the core which can be enabled with a VHDL generic. The RMAP target handles incoming packets which are determined to be RMAP commands instead of the receiver DMA engine. The RMAP command is decoded and if it is valid, the operation is performed on the AHB bus. If a reply was requested it is automatically transmitted back to the source by the RMAP transmitter.

The core is controlled by writing to a set of user registers through the APB interface and three signals: tick-in, rmapen and clkdiv10. The controlled parts are clock-generation, DMA engines, RMAP target and the link interface.

The link interface, DMA engines, RMAP target and AMBA interface are described in section 55.3, 55.4, 55.6 and 55.7 respectively.

55.2.2 Protocol support

The core only accepts packets with a destination address corresponding to the one set in the node address register. Packets with address mismatch will be silently discarded (except in promiscuous mode which is covered in section 55.4.10). The node address register is initialized to the default address 254 during reset. It can then be changed to other values by writing to the register.

The core has support for the protocol ID specified in ECSS-E-ST-50-51C. It is used for identifying RMAP commands that should be received by the RMAP target. Other packets are stored to the DMA channel. This is only applicable if the RMAP target is present and enabled. When the RMAP target is not present or disabled all the bytes after the address are treated as normal cargo.

RMAP commands are identified using the protocol ID (0x01) and the instruction field. They are handled separately from other packets if the hardware RMAP target is enabled. When enabled, all RMAP commands are processed, executed and replied in hardware. RMAP replies received are always stored to the DMA channel. If the RMAP target is disabled, all packets are stored to the DMA channel. More information on the RMAP protocol support is found in section 55.6.

RMAP packets arriving with the extended protocol ID (0x000001) are stored to the DMA channel which means that the hardware RMAP target will not work if the incoming RMAP packets use the extended protocol ID. Note also that packets with the reserved extended protocol identifier (ID = 0x000000) are not ignored by the core. It is up to the client receiving the packets to ignore them.

When transmitting packets, the address and protocol-ID fields must be included in the buffers from where data is fetched. They are *not* automatically added by the core.

Figure 160 shows the packet types supported by the core. The core also allows reception and transmission with extended protocol identifiers but without support for RMAP CRC calculations and the RMAP target.

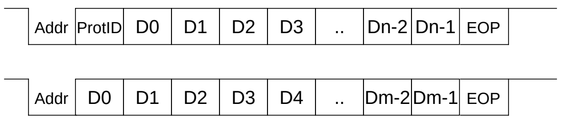


Figure 160. The SpaceWire packet types supported by the GRSPW.

55.3 Link interface

The link interface handles the communication on the SpaceWire network and consists of a transmitter, receiver, a FSM and FIFO interfaces. An overview of the architecture is found in figure 159.

55.3.1 Link interface FSM

The FSM controls the link interface (a more detailed description is found in the SpaceWire standard). The low-level protocol handling (the signal and character level of the SpaceWire standard) is handled by the transmitter and receiver while the FSM in the host domain handles the exchange level.

The link interface FSM is controlled through the control register. The link can be disabled through the link disable bit, which depending on the current state, either prevents the link interface from reaching the started state or forces it to the error-reset state. When the link is not disabled, the link interface FSM is allowed to enter the started state when either the link start bit is set or when a NULL character has been received and the autostart bit is set.

The current state of the link interface determines which type of characters are allowed to be transmitted which together with the requests made from the host interfaces determine what character will be sent.

Time-codes are sent when the FSM is in the run-state and a request is made through the time-interface (described in section 55.3.5).

When the link interface is in the connecting- or run-state it is allowed to send FCTs. FCTs are sent automatically by the link interface when possible. This is done based on the maximum value of 56 for the outstanding credit counter and the currently free space in the receiver N-Char FIFO. FCTs are sent as long as the outstanding counter is less than or equal to 48 and there are at least 8 more empty FIFO entries than the counter value.

N-Chars are sent in the run-state when they are available from the transmitter FIFO and there are credits available. NULLs are sent when no other character transmission is requested or the FSM is in a state where no other transmissions are allowed.

The credit counter (incoming credits) is automatically increased when FCTs are received and decreased when N-Chars are transmitted. Received N-Chars are stored to the receiver N-Char FIFO for further handling by the DMA interface. Received Time-codes are handled by the time-interface.

55.3.2 Transmitter

The state of the FSM, credit counters, requests from the time-interface and requests from the DMA-interface are used to decide the next character to be transmitted. The type of character and the character itself (for N-Chars and Time-codes) to be transmitted are presented to the low-level transmitter which is located in a separate clock-domain.

This is done because one usually wants to run the SpaceWire link on a different frequency than the host system clock. The core has a separate clock input which is used to generate the transmitter clock. More information on transmitter clock generation is found in section 55.8.1. Since the transmitter often runs on high frequency clocks (> 100 MHz) as much logic as possible has been placed in the system clock domain to minimize power consumption and timing issues.

The transmitter logic in the host clock domain decides what character to send next and sets the proper control signal and presents any needed character to the low-level transmitter as shown in figure 161. The transmitter sends the requested characters and generates parity and control bits as needed. If no requests are made from the host domain, NULLs are sent as long as the transmitter is enabled. Most of the signal and character levels of the SpaceWire standard is handled in the transmitter. External LVDS drivers are needed for the data and strobe signals.

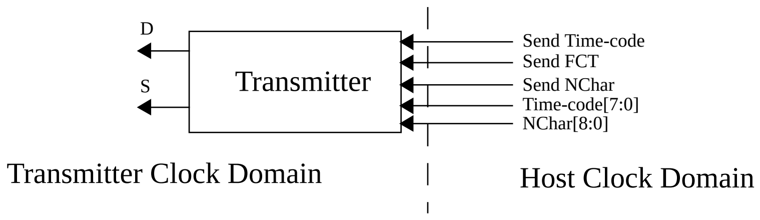


Figure 161. Schematic of the link interface transmitter.

A transmission FSM reads N-Chars for transmission from the transmitter FIFO. It is given packet lengths from the DMA interface and appends EOPs/EEPs and RMAP CRC values if requested. When it is finished with a packet the DMA interface is notified and a new packet length value is given.

55.3.3 Receiver

The receiver detects connections from other nodes and receives characters as a bit stream on the data and strobe signals. It is also located in a separate clock domain which runs on a clock generated from the received data and strobe signals. More information on the clock-generation can be found in section 55.8.1.

The receiver is activated as soon as the link interface leaves the error reset state. Then after a NULL is received it can start receiving any characters. It detects parity, escape and credit errors which causes the link interface to enter the error reset state. Disconnections are handled in the link interface part in the system clock domain because no receiver clock is available when disconnected.

Received Characters are flagged to the host domain and the data is presented in parallel form. The interface to the host domain is shown in figure 162. L-Chars are the handled automatically by the host domain link interface part while all N-Chars are stored in the receiver FIFO for further handling. If two or more consecutive EOPs/EEPs are received all but the first are discarded.

There are no signals going directly from the transmitter clock domain to the receiver clock domain and vice versa. All the synchronization is done to the system clock.

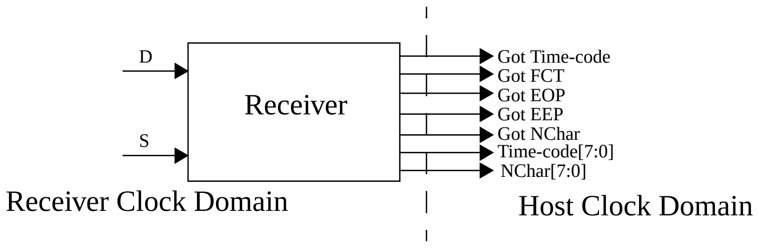


Figure 162. Schematic of the link interface receiver.

55.3.4 Dual port support

The core can be configured to include an additional SpaceWire port. With dual ports the transmitter drives an additional pair of data/strobe output signals and one extra receiver is added to handle a second pair of data/strobe input signals.

One of the ports is set as active (how the active port is selected is explained below) and the transmitter drives the data/strobe signals of the active port with the actual output values as explained in section 55.3.2. The inactive port is driven with zero on both data and strobe.

Both receivers will always be active but only the active port's interface signals (see figure 162) will be propagated to the link interface FSM. Each time the active port is changed, the link will be reset so that the new link is started in a controlled manner.

When the noportforce register is zero the portsel register bit selects the active link and when set to one it is determined by the current link activity. In the latter mode the port is changed when no activity is seen on the currently active link while there is activity on the deselected receive port. Activity is defined as a detected null. This definition is selected so that glitches (e.g. port unconnected) do not cause unwanted port switches.

55.3.5 Time interface

The time interface is used for sending Time-codes over the SpaceWire network and consists of a time-counter register, time-ctrl register, tick-in signal, tick-out signal, tick-in register field and a tick-out register field. There are also two control register bits which enable the time receiver and transmitter respectively.

Each Time-code sent from the core is a concatenation of the time-ctrl and the time-counter register. There is a timetxen bit which is used to enable Time-code transmissions. It is not possible to send time-codes if this bit is zero.

Received Time-codes are stored to the same time-ctrl and time-counter registers which are used for transmission. The timerxen bit in the control register is used for enabling time-code reception. No time-codes will be received if this bit is zero.

The two enable bits are used for ensuring that a node will not (accidentally) both transmit and receive time-codes which violates the SpaceWire standard. It also ensures that a the master sending time-codes on a network will not have its time-counter overwritten if another (faulty) node starts sending time-codes.

The time-counter register is set to 0 after reset and is incremented each time the tick-in signal is asserted for one clock-period and the timetxen bit is set. This also causes the link interface to send the new value on the network. Tick-in can be generated either by writing a one to the register field or by asserting the tick-in signal. A Tick-in should not be generated too often since if the time-code after the previous Tick-in has not been sent the register will not be incremented and no new value will be sent. The tick-in field is automatically cleared when the value has been sent and thus no new ticks should be generated until this field is zero. If the tick-in signal is used there should be at least 4 system-clock and 25 transmit-clock cycles between each assertion.

A tick-out is generated each time a valid time-code is received and the timerxen bit is set. When the tick-out is generated the tick-out signal will be asserted one clock-cycle and the tick-out register field is asserted until it is cleared by writing a one to it.

The current time counter value can be read from the time register. It is updated each time a Time-code is received and the timerxen bit is set. The same register is used for transmissions and can also be written directly from the APB interface.

The control bits of the Time-code are always stored to the time-ctrl register when a Time-code is received whose time-count is one more than the nodes current time-counter register. The time-ctrl register can be read through the APB interface. The same register is used during time-code transmissions.

It is possible to have both the time-transmission and reception functions enabled at the same time.

55.4 Receiver DMA engine

The receiver DMA engine handles reception of data from the SpaceWire network to different DMA channels. Currently there is only one receive DMA channel available but the core has been written so that additional channels can be easily added if needed.

55.4.1 Basic functionality

The receiver DMA engine reads N-Chars from the N-Char FIFO and stores them to a DMA channel. Reception is based on descriptors located in a consecutive area in memory that hold pointers to buffers where packets should be stored. When a packet arrives at the core it reads a descriptor from memory and stores the packet to the memory area pointed to by the descriptor. Then it stores status to the same descriptor and increments the descriptor pointer to the next one.

55.4.2 Setting up the core for reception

A few registers need to be initialized before reception can take place. First the link interface need to be put in the run state before any data can be sent. The DMA channel has a maximum length register which sets the maximum size of packet that can be received to this channel. Larger packets are truncated and the excessive part is spilled. If this happens an indication will be given in the status field of the descriptor. The minimum value for the receiver maximum length field is 4 and the value can only be incremented in steps of four bytes. If the maximum length is set to zero the receiver will *not* function correctly.

The node address register needs to be set to hold the address of this SpaceWire node. Packets received with the incorrect address are discarded. Finally, the descriptor table and control register must be initialized. This will be described in the two following sections.

55.4.3 Setting up the descriptor table address

The core reads descriptors from an area in memory pointed to by the receiver descriptor table address register. The register consists of a base address and a descriptor selector. The base address points to the beginning of the area and must start on a 1 kbytes aligned address. It is also limited to be 1 kbytes in size which means the maximum number of descriptors is 128.

The descriptor selector points to individual descriptors and is increased by 1 when a descriptor has been used. When the selector reaches the upper limit of the area it wraps to the beginning automatically. It can also be set to wrap automatically by setting a bit in the descriptors. The idea is that the selector should be initialized to 0 (start of the descriptor area) but it can also be written with another 8 bytes aligned value to start somewhere in the middle of the area. It will still wrap to the beginning of the area.

If one wants to use a new descriptor table the receiver enable bit has to be cleared first. When the rxactive bit for the channel is cleared it is safe to update the descriptor table register. When this is finished and descriptors are enabled the receiver enable bit can be set again.

55.4.4 Enabling descriptors

As mentioned earlier one or more descriptors must be enabled before reception can take place. Each descriptor is 8 byte in size and the layout can be found in the tables below. The descriptors should be written to the memory area pointed to by the receiver descriptor table address register. When new descriptors are added they must always be placed after the previous one written to the area. Otherwise they will not be noticed.

A descriptor is enabled by setting the address pointer to point at a location where data can be stored and then setting the enable bit. The WR bit can be set to cause the selector to be set to zero when reception has finished to this descriptor. IE should be set if an interrupt is wanted when the reception has finished. The DMA control register interrupt enable bit must also be set for this to happen.

The descriptor packet address should be word aligned. All accesses on the bus are word accesses so complete words will always be overwritten regardless of whether all 32-bit contain received data. Also if the packet does not end on a word boundary the complete word containing the last data byte will be overwritten. If the *rxunaligned* or *rmap* VHDL generic is set to 1 this restriction is removed and any number of bytes can be received to any packet address without excessive bytes being overwritten.

Table 670. GRSPW receive descriptor word 0 (address offset 0x0)

31	30	29	28	27	26	25	24 – 0
TR	DC	HC	EP	IE	WR	EN	PACKETLENGTH

31	Truncated (TR) - Packet was truncated due to maximum length violation.
30	Data CRC (DC) - Unused. 1 if a CRC error was detected for the data and 0 otherwise.
29	Header CRC (HC) - Unused. 1 if a CRC error was detected for the header and 0 otherwise.

Table 670. GRSPW receive descriptor word 0 (address offset 0x0)

28	EEP termination (EP) - This packet ended with an Error End of Packet character.
27	Interrupt enable (IE) - If set, an interrupt will be generated when a packet has been received if the receive interrupt enable bit in the DMA channel control register is set.
26	Wrap (WR) - If set, the next descriptor used by the GRSPW will be the first one in the descriptor table (at the base address). Otherwise the descriptor pointer will be increased with 0x8 to use the descriptor at the next higher memory location. The descriptor table is limited to 1 kbytes in size and the pointer will be automatically wrap back to the base address when it reaches the 1 kbytes boundary.
25	Enable (EN) - Set to one to activate this descriptor. This means that the descriptor contains valid control values and the memory area pointed to by the packet address field can be used to store a packet.
24: 0	Packet length (PACKETLENGTH) - The number of bytes received to this buffer. Only valid after EN has been set to 0 by the GRSPW.

Table 671. GRSPW receive descriptor word 1 (address offset 0x4)

31 ... 0
PACKETADDRESS

31: 0	Packet address (PACKETADDRESS) - The address pointing at the buffer which will be used to store the received packet. If the rxunaligned and rmap VHDL generics are both set to zero only bit 31 to 2 are used.

55.4.5 Setting up the DMA control register

The final step to receive packets is to set the control register in the following steps: The receiver must be enabled by setting the rxen bit in the DMA control register (see section 55.9). This can be done anytime and before this bit is set nothing will happen. The rxdescav bit in the DMA control register is then set to indicate that there are new active descriptors. This must always be done after the descriptors have been enabled or the core might not notice the new descriptors. More descriptors can be activated when reception has already started by enabling the descriptors and writing the rxdescav bit. When these bits are set reception will start immediately when data is arriving.

55.4.6 The effect to the control bits during reception

When the receiver is disabled all packets going to the DMA-channel are discarded. If the receiver is enabled the next state is entered where the rxdescav bit is checked. This bit indicates whether there are active descriptors or not and should be set by the external application using the DMA channel each time descriptors are enabled as mentioned above. If the rxdescav bit is ‘0’ and the nospill bit is ‘0’ the packets will be discarded. If nospill is one the core waits until rxdescav is set.

When rxdescav is set the next descriptor is read and if enabled the packet is received to the buffer. If the read descriptor is not enabled, rxdescav is set to ‘0’ and the packet is spilled depending on the value of nospill.

The receiver can be disabled at any time and will cause all packets received afterwards to be discarded. If a packet is currently received when the receiver is disabled the reception will still be finished. The rxdescav bit can also be cleared at any time. It will not affect any ongoing receptions but no more descriptors will be read until it is set again. Rxdescav is also cleared by the core when it reads a disabled descriptor.

55.4.7 Address recognition and packet handling

When the receiver N-Char FIFO is not empty, N-Chars are read by the receiver DMA engine. The first character is interpreted as the logical address which is compared to the node address register. If it does not match, the complete packet is discarded (up to and including the next EOP/EEP). If the address matches the packet will or will not be received to the DMA channel depending on the conditions mentioned in the previous section. If received, the complete packet including address and protocol ID but excluding EOP/EEP is stored to the address indicated in the descriptor, otherwise the complete packet is discarded.

If the address matches the next action taken depends on whether RMAP is enabled or not. If RMAP is disabled all packets are stored to the DMA channel and depending on the conditions mentioned in the previous section, the packet will be received or not. If the packet is received the complete packet including address and protocol ID but excluding EOP/EEP is stored to the address indicated in the descriptor, otherwise the complete packet is discarded.

If RMAP is enabled the protocol ID and 3rd byte in the packet is first checked before any decisions are made. If incoming packet is an RMAP packet (ID = 0x01) and the command type field is 01b the packet is processed by the RMAP command handler which is described in section 55.6. Otherwise the packet is processed by the DMA engine as when RMAP is disabled.

At least 2 non EOP/EEP N-Chars need to be received for a packet to be stored to the DMA channel. If it is an RMAP packet 3 N-Chars are needed since the command byte determines where the packet is processed. Packets smaller than the minimum size are discarded.

55.4.8 Status bits

When the reception of a packet is finished the enable bit in the current descriptor is set to zero. When enable is zero, the status bits are also valid and the number of received bytes is indicated in the length field. The DMA control register contains a status bit which is set each time a packet has been received. The core can also be made to generate an interrupt for this event as mentioned in section 55.4.4.

RMAP CRC logic is included in the implementation if the *rmapcrc* or *rmap* VHDL generic set to 1. The RMAP CRC calculation is always active for all received packets and all bytes except the EOP/EEP are included. The packet is always assumed to be a RMAP packet and the length of the header is determined by checking byte 3 which should be the command field. The calculated CRC value is then checked when the header has been received (according to the calculated number of bytes) and if it is non-zero the HC bit is set indicating a header CRC error.

The CRC value is not set to zero after the header has been received, instead the calculation continues in the same way until the complete packet has been received. Then if the CRC value is non-zero the DC bit is set indicating a data CRC error. This means that the core can indicate a data CRC error even if the data field was correct when the header CRC was incorrect. However, the data should not be used when the header is corrupt and therefore the DC bit is unimportant in this case. When the header is not corrupted the CRC value will always be zero when the calculation continues with the data field and the behaviour will be as if the CRC calculation was restarted

If the received packet is not of RMAP type the header CRC error indication bit cannot be used. It is still possible to use the DC bit if the complete packet is covered by a CRC calculated using the RMAP CRC definition. This is because the core does not restart the calculation after the header has been received but instead calculates a complete CRC over the packet. Thus any packet format with one CRC at the end of the packet calculated according to RMAP standard can be checked using the DC bit.

If the packet is neither of RMAP type nor of the type above with RMAP CRC at the end, then both the HC and DC bits should be ignored.

55.4.9 Error handling

If a packet reception needs to be aborted because of congestion on the network, the suggested solution is to set link disable to ‘1’. Unfortunately, this will also cause the packet currently being transmitted to be truncated but this is the only safe solution since packet reception is a passive operation depending on the transmitter at the other end. A channel reset bit could be provided but is not a satisfactory solution since the untransmitted characters would still be in the transmitter node. The next character (somewhere in the middle of the packet) would be interpreted as the node address which would probably cause the packet to be discarded but not with 100% certainty. Usually this action is performed when a reception has stuck because of the transmitter not providing more data. The channel reset would not resolve this congestion.

If an AHB error occurs during reception the current packet is spilled up to and including the next EEP/EOP and then the currently active channel is disabled and the receiver enters the idle state. A bit in the channels control/status register is set to indicate this condition.

55.4.10 Promiscuous mode

The core supports a promiscuous mode where all the data received is stored to the DMA channel regardless of the node address and possible early EOPs/EEPs. This means that all non-eop/eep N-Chars received will be stored to the DMA channel. The rxmaxlength register is still checked and packets exceeding this size will be truncated.

RMAP commands will still be handled by the RMAP target when promiscuous mode is enabled if the rmapen bit is set. If it is cleared, RMAP commands will also be stored to the DMA channel.

55.5 Transmitter DMA engine

The transmitter DMA engine handles transmission of data from the DMA channel to the SpaceWire network. There is one DMA channel available but the core has been written so that additional DMA channels can be easily added if needed.

55.5.1 Basic functionality

The transmit DMA engine reads data from the AHB bus and stores them in the transmitter FIFO for transmission on the SpaceWire network. Transmission is based on the same type of descriptors as for the receiver and the descriptor table has the same alignment and size restrictions. When there are new descriptors enabled the core reads them and transfer the amount data indicated.

55.5.2 Setting up the core for transmission

Four steps need to be performed before transmissions can be done with the core. First the link interface must be enabled and started by writing the appropriate value to the ctrl register. Then the address to the descriptor table needs to be written to the transmitter descriptor table address register and one or more descriptors must also be enabled in the table. Finally, the txen bit in the DMA control register should be written with a one which triggers the transmission. These steps will be covered in more detail in the next sections.

55.5.3 Enabling descriptors

The descriptor table address register works in the same way as the receiver’s corresponding register which was covered in section 55.4.

To transmit packets one or more descriptors have to be initialized in memory which is done in the following way: The number of bytes to be transmitted and a pointer to the data has to be set. There are two different length and address fields in the transmit descriptors because there are separate pointers for header and data. If a length field is zero the corresponding part of a packet is skipped and if both are zero no packet is sent. The maximum header length is 255 bytes and the maximum data length is 16 Mbyte - 1. When the pointer and length fields have been set the enable bit should be set to enable the descriptor. This must always be done last. The other control bits must also be set before enabling the descriptor.

The transmit descriptors are 16 bytes in size so the maximum number in a single table is 64. The different fields of the descriptor together with the memory offsets are shown in the tables below.

The HC bit should be set if RMAP CRC should be calculated and inserted for the header field and correspondingly the DC bit should be set for the data field. This field is only used by the core when the CRC logic is available (*rmap* or *rmapcrc* VHDL generic set to 1). The header CRC will be calculated from the data fetched from the header pointer and the data CRC is generated from data fetched from the data pointer. The CRCs are appended after the corresponding fields. The NON-CRC bytes field is set to the number of bytes in the beginning of the header field that should not be included in the CRC calculation. The CRCs are sent even if the corresponding length is zero.

When both header and data length are zero no packet is sent not even an EOP.

55.5.4 Starting transmissions

When the descriptors have been initialized, the transmit enable bit in the DMA control register has to be set to tell the core to start transmitting. New descriptors can be activated in the table on the fly (while transmission is active). Each time a set of descriptors is added the transmit enable register bit should be set. This has to be done because each time the core encounters a disabled descriptor this register bit is set to 0.

Table 672. GRSPW transmit descriptor word 0 (address offset 0x0)

31 18	17	16	15	14	13	12	11 8	7 0
RESERVED	DC	HC	LE	IE	WR	EN	NONCRCLEN	HEADERLEN

31: 18	RESERVED
17	Append data CRC (DC) - Unused. Append CRC calculated according to the RMAP specification after the data sent from the data pointer. The CRC covers all the bytes from this pointer. A null CRC will be sent if the length of the data field is zero.
16	Append header CRC (HC) - Unused. Append CRC calculated according to the RMAP specification after the data sent from the header pointer. The CRC covers all bytes from this pointer except a number of bytes in the beginning specified by the non-crc bytes field. The CRC will not be sent if the header length field is zero.
15	Link error (LE) - A Link error occurred during the transmission of this packet.
14	Interrupt enable (IE) - If set, an interrupt will be generated when the packet has been transmitted and the transmitter interrupt enable bit in the DMA control register is set.
13	Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x10 to use the descriptor at the next higher memory location.
12	Enable (EN) - Enable transmitter descriptor. When all control fields (address, length, wrap and crc) are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the transmission. The GRSPW clears this bit when the transmission has finished.
11: 8	Non-CRC bytes (NONCRCLEN)- Unused. Sets the number of bytes in the beginning of the header which should not be included in the CRC calculation. This is necessary when using path addressing since one or more bytes in the beginning of the packet might be discarded before the packet reaches its destination.
7: 0	Header length (HEADERLEN) - Header Length in bytes. If set to zero, the header is skipped.

Table 673. GRSPW transmit descriptor word 1 (address offset 0x4)

31 0
HEADERADDRESS

31: 0	Header address (HEADERADDRESS) - Address from where the packet header is fetched. Does not need to be word aligned.

Table 674. GRSPW transmit descriptor word 2 (address offset 0x8)

31 24	23 0
RESERVED	DATALEN

Table 674. GRSPW transmit descriptor word 2 (address offset 0x8)

31: 24	RESERVED
23: 0	Data length (DATALEN) - Length of data part of packet. If set to zero, no data will be sent. If both data- and header-lengths are set to zero no packet will be sent.

Table 675. GRSPW transmit descriptor word 3(address offset 0xC)

31 … 0
DATAADDRESS

31: 0	Data address (DATAADDRESS) - Address from where data is read. Does not need to be word aligned.

55.5.5 The transmission process

When the txen bit is set the core starts reading descriptors immediately. The number of bytes indicated are read and transmitted. When a transmission has finished, status will be written to the first field of the descriptor and a packet sent bit is set in the DMA control register. If an interrupt was requested it will also be generated. Then a new descriptor is read and if enabled a new transmission starts, otherwise the transmit enable bit is cleared and nothing will happen until it is enabled again.

55.5.6 The descriptor table address register

The internal pointer which is used to keep the current position in the descriptor table can be read and written through the APB interface. This pointer is set to zero during reset and is incremented each time a descriptor is used. It wraps automatically when the 1 kbytes limit for the descriptor table is reached or it can be set to wrap earlier by setting a bit in the current descriptor.

The descriptor table register can be updated with a new table anytime when no transmission is active. No transmission is active if the transmit enable bit is zero and the complete table has been sent or if the table is aborted (explained below). If the table is aborted one has to wait until the transmit enable bit is zero before updating the table pointer.

55.5.7 Error handling

Abort Tx

The DMA control register contains a bit called Abort TX which if set causes the current transmission to be aborted, the packet is truncated and an EEP is inserted. This is only useful if the packet needs to be aborted because of congestion on the SpaceWire network. If the congestion is on the AHB bus this will not help (This should not be a problem since AHB slaves should have a maximum of 16 wait-states). The aborted packet will have its LE bit set in the descriptor. The transmit enable register bit is also cleared and no new transmissions will be done until the transmitter is enabled again.

AHB error

When an AHB error is encountered during transmission the currently active DMA channel is disabled and the transmitter goes to the idle mode. A bit in the DMA channel's control/status register is set to indicate this error condition and, if enabled, an interrupt will also be generated. Further error handling depends on what state the transmitter DMA engine was in when the AHB error occurred. If the descriptor was being read the packet transmission had not been started yet and no more actions need to be taken.

If the AHB error occurs during packet transmission the packet is truncated and an EEP is inserted. Lastly, if it occurs when status is written to the descriptor the packet has been successfully transmitted but the descriptor is not written and will continue to be enabled (this also means that no error bits are set in the descriptor for AHB errors).

The client using the channel has to correct the AHB error condition and enable the channel again. No more AHB transfers are done again from the same unit (receiver or transmitter) which was active during the AHB error until the error state is cleared and the unit is enabled again.

Link error

When a link error occurs during the transmission the remaining part of the packet is discarded up to and including the next EOP/EEP. When this is done status is immediately written (with the LE bit set) and the descriptor pointer is incremented. The link will be disconnected when the link error occurs but the core will automatically try to connect again provided that the link-start bit is asserted and the link-disabled bit is deasserted. If the LE bit in the DMA channel's control register is not set the transmitter DMA engine will wait for the link to enter run-state and start a new transmission immediately when possible if packets are pending. Otherwise the transmitter will be disabled when a link error occurs during the transmission of the current packet and no more packets will be transmitted until it is enabled again.

55.6 RMAP

The Remote Memory Access Protocol (RMAP) is used to implement access to resources in the node via the SpaceWire Link. Some common operations are reading and writing to memory, registers and FIFOs. The core has an optional hardware RMAP target which is enabled with a VHDL generic. This section describes the basics of the RMAP protocol and the target implementation.

55.6.1 Fundamentals of the protocol

RMAP is a protocol which is designed to provide remote access via a SpaceWire network to memory mapped resources on a SpaceWire node. It has been assigned protocol ID 0x01. It provides three operations write, read and read-modify-write. These operations are posted operations which means that a source does not wait for an acknowledge or reply. It also implies that any number of operations can be outstanding at any time and that no timeout mechanism is implemented in the protocol. Time-outs must be implemented in the user application which sends the commands. Data payloads of up to 16 Mb - 1 is supported in the protocol. A destination can be requested to send replies and to verify data before executing an operation. A complete description of the protocol is found in the RMAP standard.

55.6.2 Implementation

The core includes a taget for RMAP commands which processes all incoming packets with protocol ID = 0x01 and type field (bit 7 and 6 of the 3rd byte in the packet) equal to 01b. When such a packet is detected it is not stored to the DMA channel, instead it is passed to the RMAP receiver.

The core implements all three commands defined in the standard with some restrictions. First of all the optional error code 12 is not implemented and support is only provided for 32-bit big-endian systems. This means that the first byte received is the msb in a word. The command handler will not receive RMAP packets using the extended protocol ID which are always dumped to the DMA channel.

The RMAP receiver processes commands. If they are correct and accepted the operation is performed on the AHB bus and a reply is formatted. If an acknowledge is requested the RMAP transmitter automatically send the reply. RMAP transmissions have priority over DMA channel transmissions.

Packets with a mismatching destination logical address are never passed to the RMAP target. There is a user accessible destination key register which is compared to destination key field in incoming packets. If there is a mismatch and a reply has been requested the error code in the reply is set to 3. Replies are sent if and only if the ack field is set to '1'.

Detection of all error codes except code 12 is supported. When a failure occurs during a bus access the error code is set to 1 (General Error). There is predetermined order in which error-codes are set in the case of multiple errors in the core. It is shown in table 676.

*Table 676.*The order of error detection in case of multiple errors in the GRSPW. The error detected first has number 1.

Detection Order	Error Code	Error
1	2	Unused RMAP packet type or command code
2	3	Invalid destination key
3	9	Verify buffer overrun
4	11	RMW data length error
5	10	Authorization failure
6*	1	General Error (AHB errors during non-verified writes)
7	5/7	Early EOP / EEP (if early)
8	4	Invalid Data CRC
9	1	General Error (AHB errors during verified writes or RMW)
10	7	EEP
11	6	Cargo Too Large
*The AHB error is not guaranteed to be detected before Early EOP/EEP or Invalid Data CRC. For very long accesses the AHB error detection might be delayed causing the other two errors to appear first.		

Read accesses are performed on the fly, that is they are not stored in a temporary buffer before transmission. This means that the error code 1 will never be seen in a read reply since the header has already been sent when the data is read. If the AHB error occurs the packet will be truncated and ended with an EEP.

Errors up to and including Invalid Data CRC (number 8) are checked before verified commands. The other errors do not prevent verified operations from being performed.

The details of the support for the different commands are now presented. All defined commands which are received but have an option set which is not supported in this specific implementation will not be executed and a possible reply is sent with error code 10.

55.6.3 Write commands

The write commands are divided into two subcategories when examining their capabilities: verified writes and non-verified writes. Verified writes have a length restriction of 4 B and the address must be aligned to the size. That is 1 B writes can be done to any address, 2 B must be halfword aligned, 3 B are not allowed and 4 B writes must be word aligned. Since there will always be only one AHB operation performed for each RMAP verified write command the incrementing address bit can be set to any value.

Non-verified writes have no restrictions when the incrementing bit is set to 1. If it is set to 0 the number of bytes must be a multiple of 4 and the address word aligned. There is no guarantee how many words will be written when early EOP/EEP is detected for non-verified writes.

55.6.4 Read commands

Read commands are performed on the fly when the reply is sent. Thus if an AHB error occurs the packet will be truncated and ended with an EEP. There are no restrictions for incrementing reads but non-incrementing reads have the same alignment restrictions as non-verified writes. Note that the “Authorization failure” error code will be sent in the reply if a violation was detected even if the length field was zero. Also note that no data is sent in the reply if an error was detected i.e. if the status field is non-zero.

55.6.5 RMW commands

All read-modify-write sizes are supported except 6 which would have caused 3 B being read and written on the bus. The RMW bus accesses have the same restrictions as the verified writes. As in the verified write case, the incrementing bit can be set to any value since only one AHB bus operation will be performed for each RMW command. Cargo too large is detected after the bus accesses so this error will not prevent the operation from being performed. No data is sent in a reply if an error is detected i.e. the status field is non-zero.

55.6.6 Control

The RMAP command handler mostly runs in the background without any external intervention, but there are a few control possibilities.

There is an enable bit in the control register of the core which can be used to completely disable the RMAP command handler. When it is set to ‘0’ no RMAP packets will be handled in hardware, instead they are all stored to the DMA channel.

There is a possibility that RMAP commands will not be performed in the order they arrive. This can happen if a read arrives before one or more writes. Since the command handler stores replies in a buffer with more than one entry several commands can be processed even if no replies are sent. Data for read replies is read when the reply is sent and thus writes coming after the read might have been performed already if there was congestion in the transmitter. To avoid this the RMAP buffer disable bit can be set to force the command handler to only use one buffer which prevents this situation.

The last control option for the command handler is the possibility to set the destination key which is found in a separate register.

*Table 677.*GRSPW hardware RMAP handling of different packet type and command fields.

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Command**	**Action**
Reserved	**Command / Response**	**Write / Read**	**Verify data before write**	**Acknow-ledge**	**Increment Address**		
0	0	-	-	-	-	Response	Stored to DMA-channel.
0	1	0	0	0	0	Not used	Does nothing. No reply is sent.
0	1	0	0	0	1	Not used	Does nothing. No reply is sent.
0	1	0	0	1	0	Read single address	Executed normally. Address has to be word aligned and data size a multiple of four. Reply is sent. If alignment restrictions are violated error code is set to 10.
0	1	0	0	1	1	Read incrementing address.	Executed normally. No restrictions. Reply is sent.
0	1	0	1	0	0	Not used	Does nothing. No reply is sent.
0	1	0	1	0	1	Not used	Does nothing. No reply is sent.
0	1	0	1	1	0	Not used	Does nothing. Reply is sent with error code 2.
0	1	0	1	1	1	Read-Modify-Write incrementing address	Executed normally. If length is not one of the allowed rmw values nothing is done and error code is set to 11. If the length was correct, alignment restrictions are checked next. 1 byte can be rmw to any address. 2 bytes must be halfword aligned. 3 bytes are not allowed. 4 bytes must be word aligned. If these restrictions are violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	0	0	0	Write, single-address, do not verify before writing, no acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done. No reply is sent.
0	1	1	0	0	1	Write, incrementing address, do not verify before writing, no acknowledge	Executed normally. No restrictions. No reply is sent.
0	1	1	0	1	0	Write, single-address, do not verify before writing, send acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.

*Table 677.*GRSPW hardware RMAP handling of different packet type and command fields.

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Command**	**Action**
Reserved	**Command / Response**	**Write / Read**	**Verify data before write**	**Acknow-ledge**	**Increment Address**		
0	1	1	0	1	1	Write, incre-menting address, do not verify before writ-ing, send acknowledge	Executed normally. No restric-tions. If AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	0	0	Write, single address, ver-ify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restric-tions apply as for rmw. No reply is sent.
0	1	1	1	0	1	Write, incre-menting address, ver-ify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restric-tions apply as for rmw. If they are violated nothing is done. No reply is sent.
0	1	1	1	1	0	Write, single address, ver-ify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are vio-lated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	1	1	Write, incre-menting address, ver-ify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are vio-lated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
1	0	-	-	-	-	Unused	Stored to DMA-channel.
1	1	-	-	-	-	Unused	Stored to DMA-channel.

55.7 AMBA interface

The AMBA interface consists of an APB interface, an AHB master interface and DMA FIFOs. The APB interface provides access to the user registers which are described in section 55.9. The DMA engines have 32-bit wide FIFOs to the AHB master interface which are used when reading and writ-ing to the bus.

The transmitter DMA engine reads data from the bus in bursts which are half the FIFO size in length. A burst is always started when the FIFO is half-empty or if it can hold the last data for the packet. The burst containing the last data might have shorter length if the packet is not an even number of bursts in size.

The receiver DMA works in the same way except that it checks if the FIFO is half-full and then performs a burst write to the bus which is half the fifo size in length. The last burst might be shorter. If the *rmap* or *rxunaligned* VHDL generics are set to 1 the interface also handles byte accesses. Byte accesses are used for non word-aligned buffers and/or packet lengths that are not a multiple of four bytes. There might be 1 to 3 single byte writes when writing the beginning and end of the received packets.

55.7.1 APB slave interface

As mentioned above, the APB interface provides access to the user registers which are 32-bits in width. The accesses to this interface are required to be aligned word accesses. The result is undefined if this restriction is violated.

55.7.2 AHB master interface

The core contains a single master interface which is used by both the transmitter and receiver DMA engines. The arbitration algorithm between the channels is done so that if the current owner requests the interface again it will always acquire it. This will not lead to starvation problems since the DMA engines always deassert their requests between accesses.

The AHB accesses are always word accesses (HSIZE = 0x010) of type incremental burst with unspecified length (HBURST = 0x001) if VHDL generics rmap and rxunaligned are disabled. The AHB accesses can be of size byte, halfword and word (HSIZE = 0x000, 0x001, 0x010) otherwise. Byte and halfword accesses are always NONSEQ. Note that read accesses are always word accesses (HSIZE = 0x010), which can result in destructive read.

The burst length will be half the AHB FIFO size except for the last transfer for a packet which might be smaller. Shorter accesses are also done during descriptor reads and status writes.

The AHB master also supports non-incrementing accesses where the address will be constant for several consecutive accesses. HTRANS will always be NONSEQ in this case while for incrementing accesses it is set to SEQ after the first access. This feature is included to support non-incrementing reads and writes for RMAP.

If the core does not need the bus after a burst has finished there will be one wasted cycle (HTRANS = IDLE).

BUSY transfer types are never requested and the core provides full support for ERROR, RETRY and SPLIT responses.

55.8 Synthesis and hardware

55.8.1 Clock-generation

Figure 163 shows the clock recovery scheme for the receiver. Data and strobe are coupled directly from their pads to an xor gate which generates the clock. The output from the xor is then connected to a clock network. The specific type of clock network depends on the technology used. The xor gate is actually all that logically belongs to the Rx clock recovery module in figure 163.

The clock output drives all flip-flops in the receiver module found in figure 159. The data signal which is used for generating the clock is also coupled to the data inputs of several flip-flops clocked by the Rx clock as seen in figure 163. Care must be taken so that the delay from the data and strobe signals through the clock network are longer than the delay to the data input + setup time.

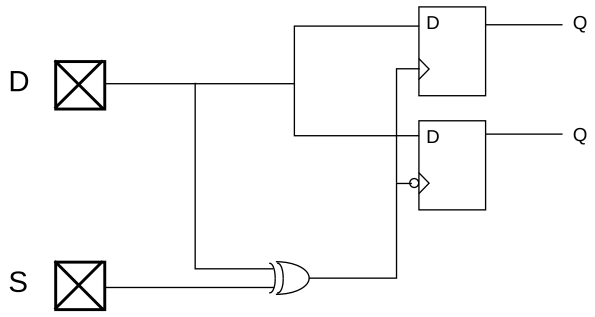


Figure 163. The clocking scheme for the receiver. The clock is

The transmitter clock is generated from the txclk input. A separate clock input is used to allow the transmitter to be run at much higher frequencies than the system clock. The SpaceWire node contains a clock-divider which divides the txclk signal to the wanted frequency. The transmitter clock should be 10 MHz during initialization and any frequency above 2 MHz in the run-state.

There is an input signal called clkdiv10 which sets the clock divisor value during initialization and the reset value for the user accessible clock divisor register. The user register value will be used in run-state. The resulting tx clock frequency will be txclk/(clock divisor value+1). So if no clock division is wanted, the clock divisor should be set to 0.

Since only integer values are allowed for the clock division and the required init-frequency is 10 Mhz the frequency of the txclk input must be a multiple of 10 MHz. The clock divisor value is 8-bits wide so the maximum txclk frequency supported is 2.56 GHz (note that there is also a restriction on the relation between the system and transmit clock frequencies).

55.8.2 Timers

There are two timers in the core: one for generating the 6.4/12.8 us periods and one for disconnect timing. They run on the system (AMBA) clock and the frequency must be at least 10 MHz to guarantee disconnect timing limits.

There are two user accessible registers which are used to the set the number of clock cycles used for the timeout periods. These registers are described in section 55.9.

The reset value for the timer registers can be set in two different ways selected by the usegen VHDL generic. If usegen is set to 1, the sysfreq VHDL generic is used to generate reset values for the disconnect, 6.4 us and 12.8 us timers. Otherwise, the input signals dcrstval and timerrstval will be used as reset values. If the system clock frequency is 10 MHz or above the disconnect time will be within the limits specified in the SpaceWire standard.

55.8.3 Synchronization

The VHDL generic nsync selects how many synchronization registers are used between clock domains. The default is one and should be used when maximum performance is needed. It allows the transmitter to be clocked 4 times faster than the system clock and the receiver 2 times faster. These are theoretical values without consideration for clock skew and jitter. Note also that the receiver clocks data at both negative and positive edges. Thus, the bitrate is twice as high as the clock-rate.

The synchronization limits the Tx and Rx clocks to be at most 4 and 2 times faster than the system clock. But it might not be possible to achieve such high clock rates for the Tx and Rx clocks for all technologies.

The asynchronous reset to the receiver clock domain has to have a maximum delay of one receiver clock cycle to ensure correct operation. This is needed because the receiver uses has a completely asynchronous reset. To make sure that nothing bad happens the is a synchronous reset guard which prevents any signals from being assigned before all registers have their reset signals released.

55.8.4 Fault-tolerance

The core can optionally be implemented with fault-tolerance against SEU errors in the FIFO memories. The fault-tolerance is enabled through the *ft* VHDL generic. Possible options are byte parity protection (*ft = 1*) or TMR registers (*ft = 2*). Note: the GPL version of GRLIB does not include fault-tolerance, and the core will not work unless the *ft* VHDL generic is 0.

55.8.5 Synthesis

Since the receiver and transmitter may run on very high frequency clocks their clock signals have been coupled through a clock buffer with a technology wrapper. This clock buffer will utilize a low skew net available in the selected technology for the clock.

The clock buffer will also enable most synthesis tools to recognize the clocks and it is thus easier to find them and place constraints on them. The fact there are three clock domains in the GRSPW of which all are possibly high frequency clocks makes it necessary to declare all paths between the clock domains as false paths.

In Synplify this is most easily done by declaring all the clocks to be in different clockgroups in the sdc file (if Synplify does not automatically put them in different groups). This will disable any timing considerations between the clock domains and these constraints will also propagate to the place and route tool.

The type of clock buffer is selectable with a VHDL generic and the value zero provides a normal feed through which lets the synthesis tool infer the type of net used.

55.8.6 Technology mapping

The core has three generics for technology mapping: *tech, techfifo* and *memtech. Tech* selects the technology used for the clock buffers and also adds reset to some registers for technologies where they would otherwise cause problems with gate-level simulations. *Techfifo* selects whether *memtech* should be used to select the technology for the FIFO memories (the RMAP buffer is not affected by the this generic) or if they should be inferred. *Tech* and *memtech* can be set to any value from 0 to NTECH as defined in the GRLIB.TECH package.

55.8.7 RAM usage

The core maps all RAM memories on the syncram_2p component if the *ft* generic is 0 and to the syncram_2pft component for other values. The syncrams are located in the technology mapping library (TECHMAP). The organization of the different memories are described below. If techfifo and/or memtech is set to 0 the synthesis tool will infer the memories. Either RAM blocks or flip-flops will be used depending on the tool and technology. The number of flip-flops used is *syncram depth x syncram width* for all the different memories. The receiver AHB FIFO with fifosize 32 will for example use 1024 flips-flops.

Receiver ahb FIFO

The receiver AHB fifo consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 678 shows the syncram organization for the allowed configurations.

*Table 678.*syncram_2p sizes for GRSPW receiver AHB FIFO.

Fifosize	Syncram_2p organization
4	4x32
8	8x32
16	16x32
32	32x32

Transmitter ahb FIFO

The transmitter AHB fifo consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 679 shows the syncram organization for the allowed configurations.

*Table 679.*syncram_2p sizes for transmitter AHB FIFO.

Fifosize	Syncram_2p organization
4	4x32
8	8x32
16	16x32
32	32x32

Receiver N-Char FIFO

The receiver N-Char fifo consists of one syncram_2p block with a width of 9-bits. The depth is determined by the configured FIFO depth. Table 680 shows the syncram organization for the allowed configurations.

*Table 680.*syncram_2p sizes for the receiver N-Char FIFO.

Fifosize	Syncram_2p organization
16	16x9
32	32x9
64	64x9

RMAP buffer

The RMAP buffer consists of one syncram_2p block with a width of 8-bits. The depth is determined by the number of configured RMAP buffers. Table 681 shows the syncram organization for the allowed configurations.

*Table 681.*syncram_2p sizes for RMAP buffer memory.

RMAP buffers	Syncram_2p organization
2	64x8
4	128x8
8	256x8

55.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 682.*GRSPW registers

APB address offset	**Register**
0x0	Control
0x4	Status/Interrupt-source
0x8	Node address
0xC	Clock divisor
0x10	Destination key
0x14	Time
0x18	Timer and Disconnect
0x20	DMA channel 1 control/status
0x24	DMA channel 1 rx maximum length
0x28	DMA channel 1 transmit descriptor table address.
0x2C	DMA channel 1 receive descriptor table address.

Table 683. GRSPW control register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RA	RX	RC			PO	RESERVED				PS	NP			RD	RE	RESERVED				TR	TT	LI	TQ		RS	PM	TI	IE	AS	LS	LD

31	RMAP available (RA) - Set to one if the RMAP command handler is available. Only readable.
30	RX unaligned access (RX) - Set to one if unaligned writes are available for the receiver. Only readable.
29	RMAP CRC available (RC) - Set to one if RMAP CRC is enabled in the core. Only readable.
28: 27	RESERVED
26	Number of ports (PO) - The number of available SpaceWire ports minus one. Only readable.
25: 22	RESERVED
21	Port select (PS) - Selects the active port when the no port force bit is zero. ‘0’ selects the port connected to data and strobe on index 0 while ‘1’ selects index 1. Only available if the ports VHDL generic is set to 2. Reset value: ‘0’.
20	No port force (NP) - Disable port force. When disabled the port select bit cannot be used to select the active port. Instead, it is automatically selected by checking the activity on the respective receive links. Only available if the ports VHDL generic is set to 2. Reset value: ‘0’ if the RMAP command handler is not available. If available the reset value is set to the value of the rmapen input signal.

Table 683. GRSPW control register

19: 18	RESERVED
17	RMAP buffer disable (RD) - Unused. If set only one RMAP buffer is used. This ensures that all RMAP commands will be executed consecutively. Only available if the rmap VHDL generic is set to 1. Reset value: ‘0’.
16	RMAP Enable (RE) - Unused. Enable RMAP command handler. Only available if rmap VHDL generic is set to 1. Reset value: ‘1’.
15: 12	RESERVED
11	Time Rx Enable (TR) - Enable time-code receptions. Reset value: ‘0’.
10	Time Tx Enable (TT) - Enable time-code transmissions. Reset value: ‘0’.
9	Link error IRQ (LI) - Generate interrupt when a link error occurs. Not reset.
8	Tick-out IRQ (TQ) - Generate interrupt when a valid time-code is received. Not reset.
7	RESERVED
6	Reset (RS) - Make complete reset of the SpaceWire node. Self clearing. Reset value: ‘0’.
5	Promiscuous Mode (PM) - Enable Promiscuous mode. Reset value: ‘0’.
4	Tick In (TI) - The host can generate a tick by writing a one to this field. This will increment the timer counter and the new value is transmitted after the current character is transferred. A tick can also be generated by asserting the tick_in signal. Reset value: ‘0’.
3	Interrupt Enable (IE) - If set, an interrupt is generated when one or both of bit 8 to 9 is set and its corresponding event occurs. Reset value: ‘0’.
2	Autostart (AS) - Automatically start the link when a NULL has been received. Reset value: ‘0’ if the RMAP command handler is not available. If available the reset value is set to the value of the rmapen input signal.
1	Link Start (LS) - Start the link, i.e. allow a transition from ready to started state. Reset value: ‘0’.
0	Link Disable (LD) - Disable the SpaceWire codec. Reset value: ‘0’.

Table 684. GRSPW status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED								LS			RESERVED											AP	EE	IA	WE		PE	DE	ER	CE	TO

31: 24	RESERVED
23: 21	Link State (LS) - The current state of the start-up sequence. 0 = Error-reset, 1 = Error-wait, 2 = Ready, 3 = Started, 4 = Connecting, 5 = Run. Reset value: 0.
20: 10	RESERVED
9	Active port (AP) - Shows the currently active port. ‘0’ = Port 0 and ‘1’ = Port 1 where the port numbers refer to the index number of the data and strobe signals. Only available if the ports generic is set to 2.
8	Early EOP/EEP (EE) - Set to one when a packet is received with an EOP after the first byte for a non-rmap packet and after the second byte for a RMAP packet. Cleared when written with a one. Reset value: ‘0’.
7	Invalid Address (IA) - Set to one when a packet is received with an invalid destination address field, i.e it does not match the nodeaddr register. Cleared when written with a one. Reset value: ‘0’.
6	Write synchronization Error (WE) - A synchronization problem has occurred when receiving N-Chars. Cleared when written with a one. Reset value: ‘0’.
5	RESERVED
4	Parity Error (PE) - A parity error has occurred. Cleared when written with a one. Reset value: ‘0’.
3	Disconnect Error (DE) - A disconnection error has occurred. Cleared when written with a one. Reset value: ‘0’.
2	Escape Error (ER) - An escape error has occurred. Cleared when written with a one. Reset value: ‘0’.
1	Credit Error (CE) - A credit has occurred. Cleared when written with a one. Reset value: ‘0’.
0	Tick Out (TO) - A new time count value was received and is stored in the time counter field. Cleared when written with a one. Reset value: ‘0’.

Table 685. GRSPW node address register

31 8	7 0
RESERVED	NODEADDR

31: 8	RESERVED
7: 0	Node address (NODEADDR) - 8-bit node address used for node identification on the SpaceWire network. Reset value: 254 (taken from the nodeaddr VHDL generic when /= 255, else from the rmapnodeaddr input signal)

Table 686. GRSPW clock divisor register

31 16	15 8	7 0
RESERVED	CLKDIVSTART	CLKDIVRUN

31: 16	RESERVED
15: 8	Clock divisor startup (CLKDIVSTART) - 8-bit Clock divisor value used for the clock-divider during startup (link-interface is in other states than run). The actual divisor value is Clock Divisor register + 1. Reset value: clkdiv10 input signal.
7: 0	Clock divisor run (CLKDIVRUN) - 8-bit Clock divisor value used for the clock-divider when the link-interface is in the run-state. The actual divisor value is Clock Divisor register + 1. Reset value: clkdiv10 input signal.

Table 687. GRSPW destination key

31 8	7 0
RESERVED	DESTKEY

31: 8	RESERVED
7: 0	Destination key (DESTKEY) - RMAP destination key. Only available if the rmap VHDL generic is set to 1. Reset value: 0 (taken from the deskey VHDL generic)

Table 688. GRSPW time register

31 8	7 6	5 0
RESERVED	TCTRL	TIMECNT

31: 8	RESERVED
7: 6	Time control flags (TCTRL) - The current value of the time control flags. Sent with time-code resulting from a tick-in. Received control flags are also stored in this register. Reset value: '0'.
5: 0	Time counter (TIMECNT) - The current value of the system time counter. It is incremented for each tick-in and the incremented value is transmitted. The register can also be written directly but the written value will not be transmitted. Received time-counter values are also stored in this register. Reset value: '0'.

Table 689. GRSPW timer and disconnect register.

31 22	21 12	11 0
RESERVED	DISCONNECT	TIMER64

31: 22	RESERVED

Table 689. GRSPW timer and disconnect register.

21: 12	Disconnect (DISCONNECT) - Used to generate the 850 ns disconnect time period. The disconnect period is the number is the number of clock cycles in the disconnect register + 3. So to get a 850 ns period, the smallest number of clock cycles that is greater than or equal to 850 ns should be calculated and this values - 3 should be stored in the register. Reset value is set with VHDL generics or with input signals depending on the value of the usegen VHDL generic.
11: 0	6.4 us timer (TIMER64) - Used to generate the 6.4 and 12.8 us time periods. Should be set to the smallest number of clock cycles that is greater than or equal to 6.4 us. Reset value is set with VHDL generics or with input signals depending on the value of the usegen VHDL generic.

Table 690. GRSPW dma control register

31 - 17	16	15 - 13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED	LE	RESERVED	NS	RD	RX	AT	RA	TA	PR	PS	AI	RI	TI	RE	TE

31: 17	RESERVED
16	Link error disable (LE) - Disable transmitter when a link error occurs. No more packets will be transmitted until the transmitter is enabled again. Reset value: ‘0’.
15: 13	RESERVED
12	No spill (NS) - If cleared, packets will be discarded when a packet is arriving and there are no active descriptors. If set, the GRSPW will wait for a descriptor to be activated.
11	Rx descriptors available (RD) - Set to one, to indicate to the GRSPW that there are enabled descriptors in the descriptor table. Cleared by the GRSPW when it encounters a disabled descriptor: Reset value: ‘0’.
10	RX active (RX) - Is set to ‘1’ if a reception to the DMA channel is currently active otherwise it is ‘0’. Only readable.
9	Abort TX (AT) - Set to one to abort the currently transmitting packet and disable transmissions. If no transmission is active the only effect is to disable transmissions. Self clearing. Reset value: ‘0’.
8	RX AHB error (RA) - An error response was detected on the AHB bus while this receive DMA channel was accessing the bus. Cleared when written with a one. Reset value: ‘0’.
7	TX AHB error (TA) - An error response was detected on the AHB bus while this transmit DMA channel was accessing the bus. Cleared when written with a one. Reset value: ‘0’.
6	Packet received (PR) - This bit is set each time a packet has been received. never cleared by the SW-node. Cleared when written with a one. Reset value: ‘0’.
5	Packet sent (PS) - This bit is set each time a packet has been sent. Never cleared by the SW-node. Cleared when written with a one. Reset value: ‘0’.
4	AHB error interrupt (AI) - If set, an interrupt will be generated each time an AHB error occurs when this DMA channel is accessing the bus. Not reset.
3	Receive interrupt (RI) - If set, an interrupt will be generated each time a packet has been received. This happens both if the packet is terminated by an EEP or EOP. Not reset.
2	Transmit interrupt (TI) - If set, an interrupt will be generated each time a packet is transmitted. The interrupt is generated regardless of whether the transmission was successful or not. Not reset.
1	Receiver enable (RE) - Set to one when packets are allowed to be received to this channel. Reset value: ‘0’.
0	Transmitter enable (TE) - Write a one to this bit each time new descriptors are activated in the table. Writing a one will cause the SW-node to read a new descriptor and try to transmit the packet it points to. This bit is automatically cleared when the SW-node encounters a descriptor which is disabled. Reset value: ‘0’.

Table 691. GRSPW RX maximum length register.

31 - 25	24 - 0
RESERVED	RXMAXLEN

Table 691. GRSPW RX maximum length register.

31: 25	RESERVED
24: 0	RX maximum length (RXMAXLEN) - Receiver packet maximum length in bytes. Only bits 24 - 2 are writable. Bits 1 - 0 are always 0. Not reset.

Table 692. GRSPW transmitter descriptor table address register.

31 : 10	9 : 4	3 : 0
DESCBASEADDR	DESCSEL	RESERVED

31: 10	Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. Not reset.
9: 4	Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPW. For each new descriptor read, the selector will increase with 16 and eventually wrap to zero again. Reset value: 0.
3: 0	RESERVED

Table 693. GRSPW receiver descriptor table address register.

31 : 10	9 : 3	2 : 0
DESCBASEADDR	DESCSEL	RESERVED

31: 10	Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. Not reset.
9: 3	Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPW. For each new descriptor read, the selector will increase with 8 and eventually wrap to zero again. Reset value: 0.
2: 0	RESERVED

55.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x1F. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

55.11 Configuration options

Table 694 shows the configuration options of the core (VHDL generics).

*Table 694.*Configuration options

Generic	Function	Allowed range	Default
tech	Technology for clock buffers	0 - NTECH	inferred
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by GRSPW.	0 - NAHBIRQ-1	0
sysfreq	Frequency of clock input “clk” in kHz.	-	10000
usegen	Use values calculated from sysfreq generic as reset values for 6.4 us timer and disconnect timer.	0 - 1	1
nsync	Number of synchronization registers. Warning: Value 2 only to be used when bit rate is equal or less than the system clock frequency.	1 - 2	1
rmap	Include hardware RMAP target. RMAP CRC logic will also be added. If set to 2 the core will only implement the RMAP target, provide a limited APB interface, enable time code reception and its interrupt.	0 - 2	0
rmapcrc	Enable RMAP CRC logic.	0 - 1	0
fifosize1	Sets the number of entries in the 32-bit receiver and transmitter AHB fifos.	4 - 32	32
fifosize2	Sets the number of entries in the 9-bit receiver fifo (N-Char fifo).	16 - 64	64
rxclkbuftype	Select clock buffer type for receiver clock. 0 does not select a buffer, instead i connects the input directly to the output (synthesis tools may still infer a buffer). 1 selects hardwired clock while 2 selects routed clock.	0 - 2	0
rxunaligned	Receiver unaligned write support. If set, the receiver can write any number of bytes to any start address without writing any excessive bytes.	0 - 1	0
rmapbufs	Sets the number of buffers to hold RMAP replies.	2 - 8	4
ft	Enable fault-tolerance against SEU errors	0 - 2	0
scantest	Enable support for scan test	0 - 1	0
techfifo	Implement FIFO with RAM cells (1) or flip-flops (0)	0 - 1	1
netlist	Use netlist rather then RTL code	0 - 1	0
ports	Sets the number of ports	1 - 2	1
memtech	Technology for RAM blocks	0 - NTECH	inferred
nodeaddr	Sets the reset value for the core’s node address. Value 255 enables rmapnodeaddr input instead.	0 - 254 255	254
destkey	Sets the reset value for the core’s destination key.	0 - 255	0

55.12 Signal descriptions

Table 695 shows the interface signals of the core (VHDL ports). As indicated in the table the core consists of two different entities, called GRSPW and GRSPW_PHY. The GRSPW entity is the main part and includes most core’s functionality, while the GRSPW_PHY only handles the receiver clock generation and the lower parts of the PHY layer. One GRSPW_PHY entity is used per port. See section 55.16 for information on how to interface GRSPW with GRSPW_PHY.

Table 695. Signal descriptions

Entity	**Signal name**	**Field**	**Type**	**Function**	**Active**
GRSPW	RST	N/A	Input	Reset	Low
	CLK	N/A	Input	Clock	-
	RXCLK[1:0]	N/A	Input	Receiver clock. One clock per port.	-
	TXCLK	N/A	Input	Transmitter default run-state clock	-
	AHBMI	*	Input	AHB master input signals	-
	AHBMO	*	Output	AHB master output signals	-
	APBI	*	Input	APB slave input signals	-
	APBO	*	Output	APB slave output signals	-
	SWNI	D[1:0]	Input	Data input synchronous to RXCLK (rising edge). One bit per port.	-
		ND[9:0]	Input	Data input synchronous to RXCLK (falling edge). Five bits per port.	
		TICKIN	Input	Time counter tick input	High
		CLKDIV10	Input	Clock divisor value used during initialization and as reset value for the clock divisor register	-
		RMAPEN	Input	Reset value for the rmapen control register bit	-
		RMAPNODEADDR	Input	Reset value for nodeaddr register bits when nodeaddr VHDL generic /= 255	-
		DCRSTVAL	Input	Reset value for disconnect timer. Used if usegen VHDL generic is set to 0.	-
		TIMERRSTVAL	Input	Reset value for 6.4 us timer. Used if usegen VHDL generic is set to 0.	-
		DCONNECT[3:0]	Input	Disconnect strobes. Two bits per port.	
	SWNO	D[1:0]	Output	SpaceWire data output. One bit per port.	-
		S[1:0]	Output	SpaceWire strobe output. One bit per port.	-
		TICKOUT	Output	Time counter tick output	High
		LINKDIS	Output	Linkdisabled status	High
		RMAPACT	Output	RMAP command processing active	High
		RXRST	Output	Receiver reset.	Low

Table 695. Signal descriptions

Entity	Signal name	Field	Type	Function	Active
GRSPW_PHY	RXRST	N/A	Input	Receiver reset.	Low
	DI	N/A	Input	SpaceWire data input.	-
	SI	N/A	Input	SpaceWire strobe input.	-
	RXCLKO	N/A	Output	Receiver clock recovered from data and strobe input.	-
	DO	N/A	Ouput	Recovered data, synchronous to RXCLKO (rising edge).	-
	NDO[4:0]	N/A	Ouput	Recovered data, synchronous to RXCLKO (falling edge)	-
	DCON-NECT[1:0]	N/A	Ouput	Disconnect strobe signals.	-
	TESTEN	N/A	Input	Scan test enable	High
	TESTCLK	N/A	Input	Scan test clock. Used inside the GRSPW_PHY entity instead of recov-ered RXCLK when TESTEN is active.	-
	* see GRLIB IP Library User’s Manual				

55.13 Signal definitions and reset values

The signals and their reset values are described in table 696.

*Table 696.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
spw_clk	Input	Transmitter default run-state clock	Rising edge	-
spw_rxd	Input, LVDS	Data input, positive	High	-
spw_rxdn	Input, LVDS	Data input, negative	Low	-
spw_rxs	Input, LVDS	Strobe input, positive	High	-
spw_rxsn	Input, LVDS	Strobe input, negative	Low	-
spw_txd	Output, LVDS	Data output, positive	High	Logical 0
spw_txdn	Output, LVDS	Data output, negative	Low	Logical 1
spw_txs	Output, LVDS	Strobe output, positive	High	Logical 0
spw_txsn	Output, LVDS	Strobe output, negative	Low	Logical 1

55.14 Timing

The timing waveforms and timing parameters are shown in figure 164 and are defined in table 697.

The SpaceWire jitter and skew timing waveforms and timing parameters are shown in figure 165 and are defined in table 698.

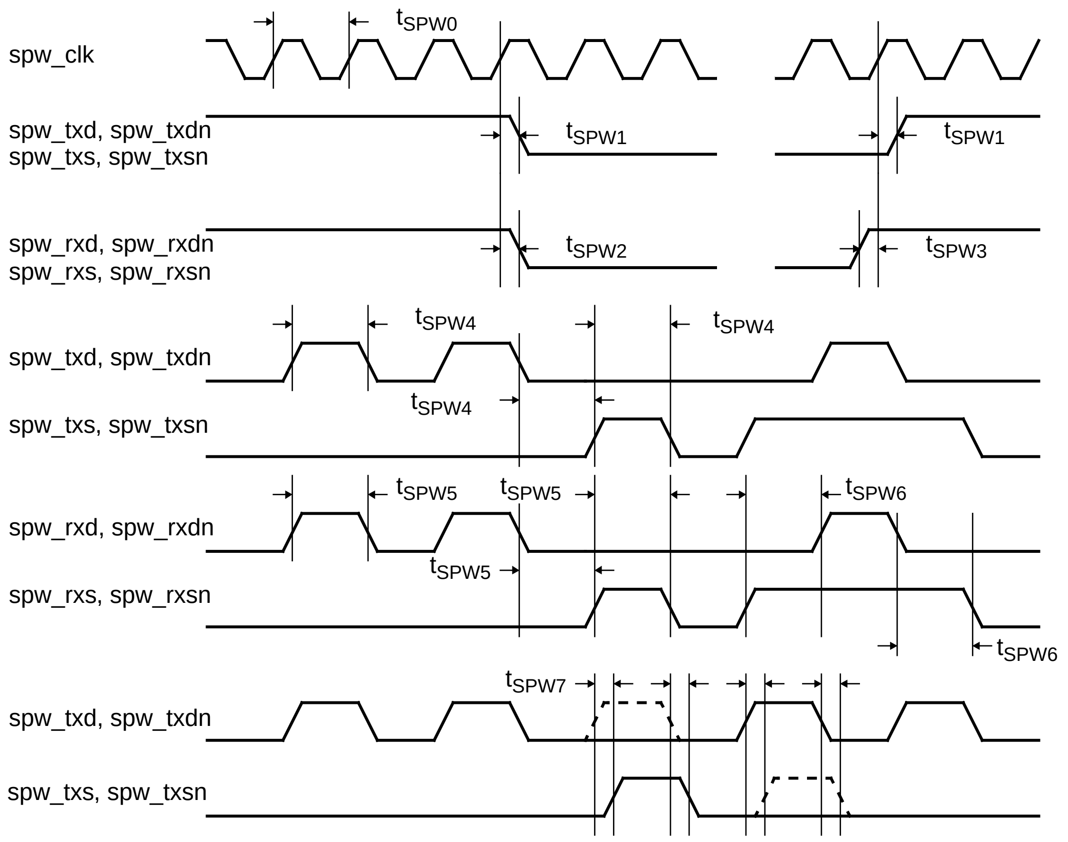


Figure 164. Timing waveforms

*Table 697.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{SPW0}	transmit clock period	-	20	-	ns
t_{SPW1}	clock to output delay	rising *spw_clk* edge	0	20	ns
t_{SPW2}	input to clock hold	-	-	-	not applicable
t_{SPW3}	input to clock setup	-	-	-	not applicable
t_{SPW4}	output data bit period	-	-		*clk* periods
		-	t_{SPW0} -5	tSPW0 +5	ns
t_{SPW5}	input data bit period	-	20	-	ns
t_{SPW6}	data & strobe edge separation	-	10	-	ns
t_{SPW7}	data & strobe output skew	-	-	5	ns

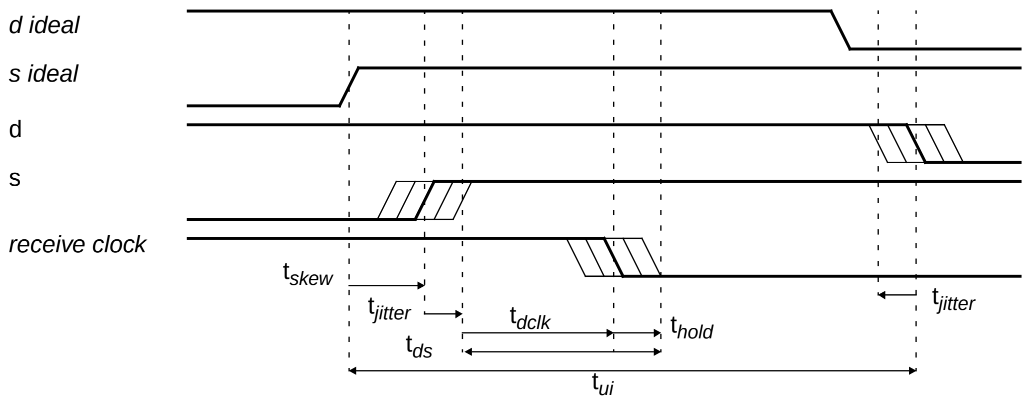


Figure 165. Skew and jitter timing waveforms

*Table 698.*Skew and jitter timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{skew}	skew between data and strobe	-	-	TBD	ns
t_{jitter}	jitter on data or strobe	-	-	TBD	ns
t_{ds}	minimum separation between data and strobe edges	-	TBD	-	ns
t_{dclk}	delay from edge of data or strobe to the receiver flip-flop	-	-	TBD	ns
t_{hold}	hold timer on receiver flip-flop	-	TBD	-	ns
t_{ui}	unit interval (bit period)	-	TBD	-	ns

55.15 Library dependencies

Table 699 shows libraries used when instantiating the core (VHDL libraries).

*Table 699.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPACEWIRE	Signals, component	Component and record declarations.

55.16 Instantiation

This example shows how the core can be instantiated.

Normally di, si, do and so should be connected to input and output pads configured with LVDS drivers. How this is done is technology dependent.

The GRSPW in the example is a 2-port core configured with non-ft memories of size 4, 64 and 8 entries for AHB FIFOs, N-Char FIFO and RMAP buffers respectively. The system frequency (clk) is 40 MHz and the transmitter frequency (txclk) is 20 MHz.

The memory technology is inferred which means that the synthesis tool will select the appropriate components. The rx clk buffer uses a hardwired clock.

The hardware RMAP command handler is enabled which also automatically enables rxunaligned and rmapcrc. Finally, the DMA channel interrupt line is 2 and the number of synchronization registers is 1.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.spacewire.all;

entity spacewire_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- spacewire signals
    di :  in  std_logic_vector(1 downto 0);
    si :  in  std_logic_vector(1 downto 0);
    do :  out std_logic_vector(1 downto 0);
    so :  out std_logic_vector(1 downto 0)
    );
end;

architecture rtl of spacewire_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- Spacewire signals
  signal swni : grspw_in_type;
  signal swno : grspw_out_type;
  signal rxclk : std_logic_vector(1 downto 0);

begin

  -- AMBA Components are instantiated here
  ...

  -- GRSPW
  sw0 : grspw
  generic map (tech => inferred, hindex => 5, pindex => 7, paddr => 7, nsync => 1,
    rmap => 1, rxunaligned => 0, rmapcrc => 0, rxclkbuftype => 0, sysfreq => 40000,
    pirq => 2, fifosize1 => 4, fifosize2 => 64, rmapbufs => 8, ft => 0, ports => 2)
  port map (rstn, clk, rxclk, apbi, apbo(7), ahbmi, ahbmo(5), swni, swno);

  phy0 : grspw_phy
  generic map (tech => inferred, rxclkbuftype => 0, scantest => 0)
  port map (rxrst => swno.rxrst, di => di(0), si => si(0),
    rxclko => rxclk(0), do => swni.d(0), ndo => swni.nd(4 downto 0),
    dconnect => swni.dconnect(1 downto 0));

  phy1 : grspw_phy
  generic map (tech => inferred, rxclkbuftype => 0)
  port map (rxrst => swno.rxrst, di => di(1), si => si(1),
    rxclko => rxclk(1), do => swni.d(1), ndo => swni.nd(9 downto 5),
    dconnect => swni.dconnect(3 downto 2));

  swni.rmapen   <= ‘1’;
  swni.clkdiv10 <= “00000001”;
  swni.tickin   <= ‘0’;
  do(0)         <= swno.d(0);
  so(0)         <= swno.s(0);
  do(1)         <= swno.d(1);
  so(1)         <= swno.s(1);
end;
```

55.17 API

A simple Application Programming Interface (API) is provided together with the GRSPW. The API is located in $(GRLIB)/software/spw. The files are rmapapi.c, spwapi.c, rmapapi.h, spwapi.h. The spwapi.h file contains the declarations of the functions used for configuring the GRSPW and transferring data. The corresponding definitions are located in spwapi.c. The rmapapi is structured in the same manner and contains a function for building RMAP packets.

These functions could be used as a simple starting point for developing drivers for the GRSPW. The different functions are described in this section.

55.17.1 GRSPW Basic API

The basic GRSPW API is based on a struct spwvars which stores all the information for a single GRSPW core. The information includes its address on the AMBA bus as well as SpaceWire parameters such as node address and clock divisor. A pointer to this struct is used as a input parameter to all the functions. If several cores are used, a separate struct for each core is created and used when the specific core is accessed.

*Table 700.*The spwvars struct

Field	**Description**	**Allowed range**
regs	Pointer to the GRSPW	-
nospill	The nospill value used for the core.	0 - 1
rmap	Indicates whether the core is configured with RMAP. Set by spw_init.	0 - 1
rxunaligned	Indicates whether the core is configured with rxunaligned support. Set by spw_init.	0 - 1
rmapcrc	Indicates whether the core is configured with RMAPCRC support. Set by spw_init.	0 - 1
clkdiv	The clock divisor value used for the core.	0 - 255
nodeaddr	The node address value used for the core.	0 - 255
destkey	The destination key value used for the core.	0 - 255
rxmaxlen	The Receiver maximum length value used for the core.	0 - 33554431
rxpnt	Pointer to the next receiver descriptor.	0 - 127
rxchkpnt	Pointer to the next receiver descriptor that will be polled.	0 - 127
txpnt	Pointer to the next transmitter descriptor.	0 - 63
txchkpnt	Pointer to the next transmitter descriptor that will be polled.	0 - 63
timetxen	The timetxen value used for this core.	0 - 1
timerxen	The timerxen value used for this core.	0 - 1
txd	Pointer to the transmitter descriptor table.	-
rxd	Pointer to the receiver descriptor table	-

The following functions are available in the basic API:

```
int spw_setparam(int nodeaddr, int clkdiv, int destkey, int nospill, int timetxen, int
timerxen, int rxmaxlen, int spwadr, struct spwvars *spw);
```

Used for setting the different parameters in the spwvars struct. Should always be run first after creating a spwvars struct. This function only initializes the struct. Does not write anything to the SpaceWire core.

*Table 701.*Return values for spw_setparam

Value	Description
0	The function completed successfully
1	One or more of the parameters had an illegal value

*Table 702.*Parameters for spw_setparam

Parameter	Description	Allowed range
nodeaddr	Sets the node address value of the struct spw passed to the function.	0-255
clkdiv	Sets the clock divisor value of the struct spw passed to the function.	0-255
destkey	Sets the destination key of the struct spw passed to the function.	0-255
nospill	Sets the nospill value of the struct spw passed to the function.	0 - 1
timetxen	Sets the timetxen value of the struct spw passed to the function.	0 - 1
timerxen	Sets the timerxen value of the struct spw passed to the function.	0 - 1
rxmaxlen	Sets the receiver maximum length field of the struct spw passed to the function.	0 - 2^{25}-1
spwadr	Sets the address to the GRSPW core which will be associated with the struct passed to the function.	0 - 2^{32}-1

```
int spw_init(struct spwvars *spw);
```

Initializes the GRSPW core located at the address set in the struct spw. Sets the following registers: node address, destination key, clock divisor, receiver maximum length, transmitter descriptor table address, receiver descriptor table address, ctrl and dmactrl. All bits are set to the values found in the spwvars struct. If a register bit is not present in the struct it will be set to zero. The descriptor tables are allocated to an aligned area using malloc. The status register is cleared and lastly the link interface is enabled. The run state frequency will be set according to the value in clkdiv.

*Table 703.*Return values for spw_init

Value	Description
0	The function completed successfully
1	One or more of the parameters could not be set correctly or the link failed to initialize.

*Table 704.*Parameters for spw_init

Parameter	Description	Allowed range
spw	The spwvars struct associated with the GRSPW core that should be initialized.	-

```
int set_txdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the transmitter descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_tx and spw_checktx (Explained in the section for those functions).

*Table 705.*Return values for spw_txdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

*Table 706.*Parameters for spw_txdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	$0 - 2^{32}-1$
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int set_rxdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the Receiver descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_rx and spw_checkrx (Explained in the section for those functions).

*Table 707.*Return values for spw_rxdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

*Table 708.*Parameters for spw_rxdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	$0 - 2^{32}-1$
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_disable(struct spwvars *spw);
```

Disables the GRSPW core (the link disable bit is set to ‘1’).

*Table 709.*Parameters for spw_disable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_enable(struct spwvars *spw);
```

Enables the GRSPW core (the link disable bit is set to ‘0’).

*Table 710.*Parameters for spw_enable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_start(struct spwvars *spw);
```

Starts the GRSPW core (the link start bit is set to ‘1’).

*Table 711.*Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_stop(struct spwvars *spw);
```

Stops the GRSPW core (the link start bit is set to ‘0’).

*Table 712.*Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_setclockdiv(struct spwvars *spw);
```

Sets the clock divisor register with the clock divisor value stored in the spwvars struct.

*Table 713.*Return values for spw_setclockdiv

Value	Description
0	The function completed successfully
1	The new clock divisor value is illegal.

*Table 714.*Parameters for spw_setclockdiv

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_set_nodeadr(struct spwvars *spw);
```

Sets the node address register with the node address value stored in the spwvars struct.

*Table 715.*Return values for spw_set_nodeadr

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

*Table 716.*Parameters for spw_set_nodeadr

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_set_rxmaxlength(struct spwvars *spw);
```

Sets the Receiver maximum length register with the rxmaxlen value stored in the spwvars struct.

*Table 717.*Return values for spw_set_rxmaxlength

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

*Table 718.*Parameters for spw_set_rxmaxlength

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_tx(int crc, int skipcrcsize, int hsize, char *hbuf, int dsize, char *dbuf, struct
spwvars *spw);
```

Transmits a packet. Separate header and data buffers can be used. If CRC logic is available the GSPW inserts RMAP CRC values after the header and data fields if crc is set to one. This function only sets a descriptor and initiates the transmission. Spw_checktx must be used to check if the packet has been transmitted. A pointer into the descriptor table is stored in the spwvars struct to keep track of the next location to use. It is incremented each time the function returns 0.

*Table 719.*Return values for spw_tx

Value	Description
0	The function completed successfully
1	There are no free transmit descriptors currently available
2	There was illegal parameters passed to the function

*Table 720.*Parameters for spw_tx

Parameter	Description	Allowed range
crc	Set to one to append RMAP CRC after the header and data fields. Only available if hardware CRC is available in the core.	0 - 1
skipcrcsize	The number of bytes in the beginning of a packet that should not be included in the CRC calculation	0 - 15
hsize	The size of the header in bytes	0 - 255
hbuf	Pointer to the header data	-
dsize	The size of the data field in bytes	0 - 2^{24}-1
dbuf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW core that should transmit the packet	-

```
int spw_rx(char *buf, struct spwvars *spw);
```

Enables a descriptor for reception. The packet will be stored to buf. Spw_checkrx must be used to check if a packet has been received. A pointer in the spwvars struct is used to keep track of the next location to use in the descriptor table. It is incremented each time the function returns 0.

*Table 721.*Return values for spw_rx

Value	Description
0	The function completed successfully
1	There are no free receive descriptors currently available

*Table 722.*Parameters for spw_rx

Parameter	Description	Allowed range
buf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW core that should receive the packet	-

```
int spw_checkrx(int *size, struct rxstatus *rxs, struct spwvars *spw);
```

Checks if a packet has been received. When a packet has been received the size in bytes will be stored in the size parameter and status is found in the rxs struct. A pointer in the spwvars struct is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

*Table 723.*Return values for spw_checkrx

Value	Description
0	No packet has been received
1	A packet has been received

*Table 724.*Parameters for spw_checkrx

Parameter	Description	Allowed range
size	When the function returns 1 this variable holds the number of bytes received	-
rxs	When the function returns 1 this variable holds status information	-
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

*Table 725.*The rxstatus struct

Field	Description	Allowed range
truncated	Packet was truncated	0 - 1
dcrcerr	Data CRC error bit was set. Only indicates an error if the packet received was an RMAP packet.	0 - 1
hcrcerr	Header CRC error bit was se.t. Only indicates an error if the packet received was an RMAP packet.	0 - 1
eep	Packet was terminated with EEP	0 - 1

```
int spw_checktx(struct spwvars *spw);
```

Checks if a packet has been transmitted. A pointer is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

*Table 726.*Return values for spw_checktx

Value	Description
0	No packet has been transmitted
1	A packet has been correctly transmitted
2	A packet has been incorrectly transmitted

*Table 727.*Parameters for spw_checktx

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
void send_time(struct spwvars *spw);
```

Sends a new time-code. Increments the time-counter in the GRSPW and transmits the value.

*Table 728.*Parameters for send time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
int check_time(struct spwvars *spw);
```

Check if a new time-code has been received.

*Table 729.*Return values for check_time

Value	Description
0	No time-code has been received
1	A new time-code has been received

*Table 730.*Parameters for check_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
int get_time(struct spwvars *spw);
```

Get the current time counter value.

*Table 731.*Return values for get_time

Value	Description
0 - 63	Returns the current time counter value

*Table 732.*Parameters for get_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
void spw_reset(struct spwvars *spw);
```

Resets the GRSPW.

*Table 733.*Parameters for spw_reset

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be reset	-

```
void spw_rmapen(struct spwvars *spw);
```

Enables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW.

*Table 734.*Parameters for spw_rmapen

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

```
void spw_rmapdis(struct spwvars *spw);
```

Disables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW

*Table 735.*Parameters for spw_rmapdis

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

```
int spw_setdestkey(struct spwvars *spw);
```

Set the destination key of the GRSPW. Has no effect if the RMAP command handler is not available. The value from the spwvars struct is used.

*Table 736.*Return values for spw_setdestkey

Value	Description
0	The function completed successfully
1	The destination key parameter in the spwvars struct contains an illegal value

*Table 737.*Parameters for spw_setdestkey

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set.	-

55.17.2 GRSPW RMAP API

The RMAP API contains only one function which is used for building RMAP headers.

```
int build_rmap_hdr(struct rmap_pkt *pkt, char *hdr, int *size);
```

Builds a RMAP header to the buffer pointed to by hdr. The header data is taken from the rmap_pkt struct.

*Table 738.*Return values for build_rmap_hdr

Value	Description
0	The function completed successfully
1	One or more of the parameters contained illegal values

*Table 739.*Parameters for build_rmap_hdr

Parameter	Description	Allowed range
pkt	Pointer to a rmap_pkt struct which contains the data from which the header should be built	
hdr	Pointer to the buffer where the header will be built	
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

*Table 740.*rmap_pkt struct fields

Field	**Description**	**Allowed Range**
type	Selects the type of packet to build.	writecmd, readcmd, rmwcmd, writerep, readrep, rmwrep
verify	Selects whether the data should be verified before writing	yes, no
ack	Selects whether an acknowledge should be sent	yes, no
incr	Selects whether the address should be incremented or not	yes, no
destaddr	Sets the destination address	0 - 255
destkey	Sets the destination key	0 - 255
srcaddr	Sets the source address	0 - 255
tid	Sets the transaction identifier field	0 - 65535
addr	Sets the address of the operation to be performed. The extended address field is currently always set to 0.	0 - 2^{32}-1
len	The number of bytes to be writte, read or read-modify-written	0 - 2^{24}-1
status	Sets the status field	0 - 11
dstspalen	Number of source path address bytes to insert before the destination address	0 - 228
dstspa	Pointer to memory holding the destination path address bytes	-
srcspalen	Number of source path address bytes to insert in a command. For a reply these bytes are placed before the return address	0 - 12
srcspa	Pointer to memory holding the source path address bytes	-

55.18 Appendix A Clarifications of the GRSPW implementation of the standard

6.3.1 page 9 RMAP draft F

"The user application at destination will be informed that there was an error

in the data transferred. The source will be informed of the data error if the

acknowledge bit in the command has been set."

We view the RMAP command handler as both protocol parser and user application. All commands are parsed in the first stage and various (internal) status bits are set. The next step (which can be viewed as the user application) will make the decision of how to act upon the received command from these bits. Therefore, the various errors that can occur are not externally observable.

If an error occurs when the command handler is accessing the AHB bus through the DMA interface errors will be externally observable using the AHB status register in GRLIB.

6.3.6 page 13 RMAP draft F

“The Write Command packet arrives at the destination and its header is found to be in error. This fact is added to the error statistics in the destination node.”

This text does not state how and if these statistics should be observable. At the moment the error handling is internal to the RMAP command handler and therefore no statistics are internally observable. A counter for this particular error might be added in the future.

6.3.6 page 15 RMAP draft F

"These various errors will be reported to the user application running on the

destination node (Write Data Error Indication)."

Again the RMAP command handler is the user application and all these errors are handled internally.

6.5.6 page 31 RMAP draft F

"The source user application, in fact immediately rejects this as an authorisation

failure as the command is trying to RMW an area of protected memory."

It should probably be destination user application instead of source. It is unclear what immediately means. Should it be rejected before any accesses are done on the bus and thus requiring the RMAP command handler to include a complete bus decoding. The GRSPW does (probably) not comply to this paragraph at the moment. If a bus error occurs a general error code will be returned.

6.5.6 page 32

"If the header of the RMW reply packet is received intact but the data field is

corrupted as indicated by an incorrect data field length (too long or too short)

or by a CRC error, then an error can be flagged to the application immediately

(RMW Data Failure) without having to wait for an application timeout."

This is not applicable to the GRSPW since it does not handle replies. However this is practically an unnecessary comment since it is not specified in the standard in which manner received replies are indicated to the higher layers.

56 GRSPW2 - SpaceWire codec with AHB host Interface and RMAP target

56.1 Overview

The SpaceWire core provides an interface between the AHB bus and a SpaceWire network. It implements the SpaceWire standard (ECSS-E-ST-50-12C) with the protocol identification extension (ECSS-E-ST-50-51C). The optional Remote Memory Access Protocol (RMAP) target implements the ECSS standard (ECSS-E-ST-50-52C).

The SpaceWire interface is configured through a set of registers accessed through an APB interface. Data is transferred through DMA channels using an AHB master interface. The number of DMA channels is configurable from one to four.

The core can also be configured with two SpaceWire ports with manual or automatic switching between them.

There can be up to four clock domains: one for the AHB interface (system clock), one for the transmitter and one or two for the receiver depending on the number of configured ports.

The core only supports byte addressed 32-bit big-endian host systems. Transmitter outputs can be either Single Data Rate (SDR) or Double Data Rate (DDR). The receiver can be connected either to an Aeroflex SpaceWire transceiver or recover the data itself using a self-clocking scheme or sampling (SDR or DDR).

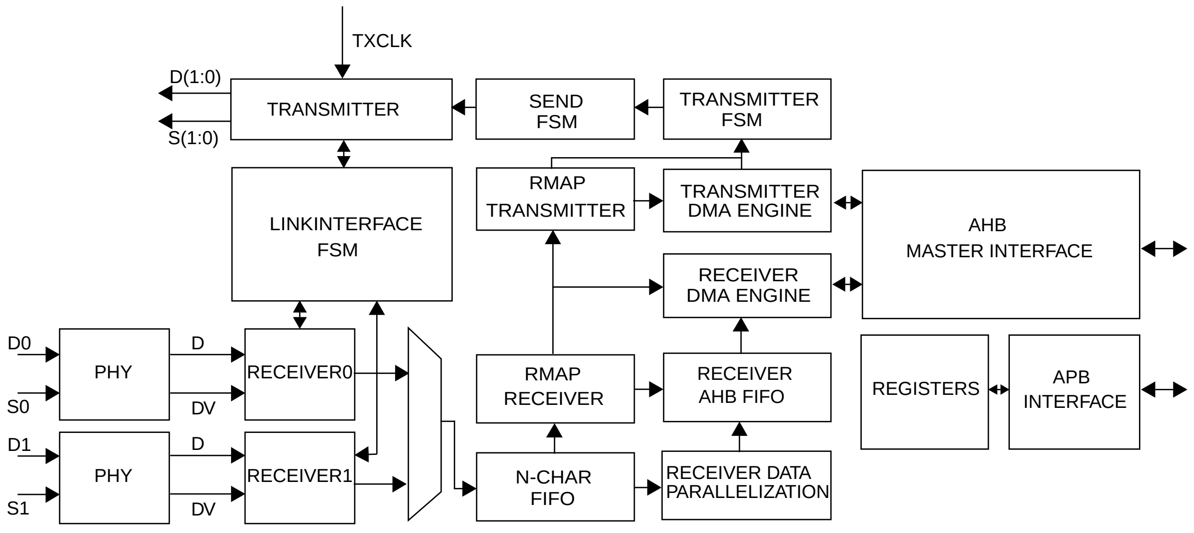


Figure 166. Block diagram

56.2 Operation

56.2.1 Overview

The main sub-blocks of the core are the link interface, the RMAP target and the AMBA interface. A block diagram of the internal structure can be found in figure 166.

The link interface consists of the receiver, transmitter and the link interface FSM. They handle communication on the SpaceWire network. The PHY block provides a common interface for the receiver to the four different data recovery schemes and is external to this core. A short description is found in section 56.3.5. The complete documentation is found in the GRSPW2_PHY section. The AMBA interface consists of the DMA engines, the AHB master interface and the APB interface. The link interface provides FIFO interfaces to the DMA engines. These FIFOs are used to transfer N-Chars between the AMBA and SpaceWire domains during reception and transmission.

The RMAP target is an optional part of the core which can be enabled with a VHDL generic. The RMAP target handles incoming packets which are determined to be RMAP commands instead of the receiver DMA engine. The RMAP command is decoded and if it is valid, the operation is performed on the AHB bus. If a reply was requested it is automatically transmitted back to the source by the RMAP transmitter.

The core is controlled by writing to a set of user registers through the APB interface and three signals: tick-in, rmapen and clkdiv10.

The link interface, DMA engines, RMAP target and AMBA interface are described in section 56.3, 56.4, 56.6 and 56.7 respectively.

56.2.2 Protocol support

The core only accepts packets with a valid destination address in the first received byte. Packets with address mismatch will be silently discarded (except in promiscuous mode which is covered in section 56.4.10).

The second byte is sometimes interpreted as a protocol ID a desrcibed hereafter. The RMAP protocol (ID=0x1) is the only protocol handled separately in hardware while other packets are stored to a DMA channel. If the RMAP target is present and enabled all RMAP commands will be processed, executed and replied automatically in hardware. Otherwise RMAP commands are stored to a DMA channel in the same way as other packets. RMAP replies are always stored to a DMA channel. More information on the RMAP protocol support is found in section 56.6. When the RMAP target is not present or disabled, there is no need to include a protocol ID in the packets and the data can start immediately after the address.

All packets arriving with the extended protocol ID (0x00) are stored to a DMA channel. This means that the hardware RMAP target will not work if the incoming RMAP packets use the extended protocol ID. Note also that packets with the reserved extended protocol identifier (ID = 0x000000) are not ignored by the core. It is up to the client receiving the packets to ignore them.

When transmitting packets, the address and protocol-ID fields must be included in the buffers from where data is fetched. They are *not* automatically added by the core.

Figure 167 shows the packet types accepted by the core. The core also allows reception and transmission with extended protocol identifiers but without support for RMAP CRC calculations and the RMAP target.

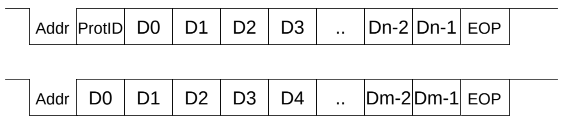


Figure 167. The SpaceWire packet types supported by the core.

56.3 Link interface

The link interface handles the communication on the SpaceWire network and consists of a transmitter, receiver, a FSM and FIFO interfaces. An overview of the architecture is found in figure 166.

56.3.1 Link interface FSM

The FSM controls the link interface (a more detailed description is found in the SpaceWire standard). The low-level protocol handling (the signal and character level of the SpaceWire standard) is handled by the transmitter and receiver while the FSM handles the exchange level.

The link interface FSM is controlled through the control register. The link can be disabled through the link disable bit, which depending on the current state, either prevents the link interface from reaching the started state or forces it to the error-reset state. When the link is not disabled, the link interface FSM is allowed to enter the started state when either the link start bit is set or when a NULL character has been received and the autostart bit is set.

The current state of the link interface determines which type of characters are allowed to be transmitted which together with the requests made from the host interfaces determine what character will be sent.

Time-codes are sent when the FSM is in the run-state and a request is made through the time-interface (described in section 56.3.6).

When the link interface is in the connecting- or run-state it is allowed to send FCTs. FCTs are sent automatically by the link interface when possible. This is done based on the maximum value of 56 for the outstanding credit counter and the currently free space in the receiver N-Char FIFO. FCTs are sent as long as the outstanding counter is less than or equal to 48 and there are at least 8 more empty FIFO entries than the counter value.

N-Chars are sent in the run-state when they are available from the transmitter FIFO and there are credits available. NULLs are sent when no other character transmission is requested or the FSM is in a state where no other transmissions are allowed.

The credit counter (incoming credits) is automatically increased when FCTs are received and decreased when N-Chars are transmitted. Received N-Chars are stored to the receiver N-Char FIFO for further handling by the DMA interface. Received Time-codes are handled by the time-interface.

56.3.2 Transmitter

The state of the FSM, credit counters, requests from the time-interface and requests from the DMA-interface are used to decide the next character to be transmitted. The type of character and the character itself (for N-Chars and Time-codes) to be transmitted are presented to the low-level transmitter which is located in a separate clock-domain.

This is done because one usually wants to run the SpaceWire link on a different frequency than the host system clock. The core has a separate clock input which is used to generate the transmitter clock. More information on transmitter clock generation is found in section 56.8.1. Since the transmitter often runs on high frequency clocks (> 100 MHz) as much logic as possible has been placed in the system clock domain to minimize power consumption and timing issues.

The transmitter logic in the host clock domain decides what character to send next and sets the proper control signal and presents any needed character to the low-level transmitter as shown in figure 168. The transmitter sends the requested characters and generates parity and control bits as needed. If no requests are made from the host domain, NULLs are sent as long as the transmitter is enabled. Most of the signal and character levels of the SpaceWire standard is handled in the transmitter. External LVDS drivers are needed for the data and strobe signals. The outputs can be configured as either single- or double data rate. The latter increases maximum bitrate significantly but is not available for all techonologies.

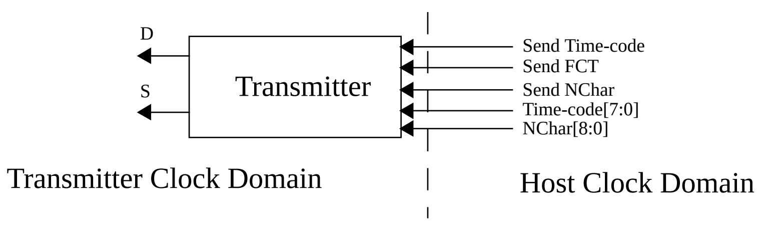


Figure 168. Schematic of the link interface transmitter.

A transmission FSM reads N-Chars for transmission from the transmitter FIFO. It is given packet lengths from the DMA interface and appends EOPs/EEPs and RMAP CRC values if requested. When it is finished with a packet the DMA interface is notified and a new packet length value is given.

56.3.3 Receiver

The receiver detects connections from other nodes and receives characters as a bit stream recovered from the data and strobe signals by the PHY module which presents it as a data and data-valid signal. Both the receiver and PHY are located in a separate clock domain which runs on a clock generated by the PHY. More information on the clock-generation can be found in section 56.8.1.

The receiver is activated as soon as the link interface leaves the error reset state. Then after a NULL is received it can start receiving any characters. It detects parity, escape and credit errors which causes the link interface to enter the error reset state. Disconnections are handled in the link interface part in the tx clock domain because no receiver clock is available when disconnected.

Received Characters are flagged to the host domain and the data is presented in parallel form. The interface to the host domain is shown in figure 169. L-Chars are the handled automatically by the host domain link interface part while all N-Chars are stored in the receiver FIFO for further handling. If two or more consecutive EOPs/EEPs are received all but the first are discarded.

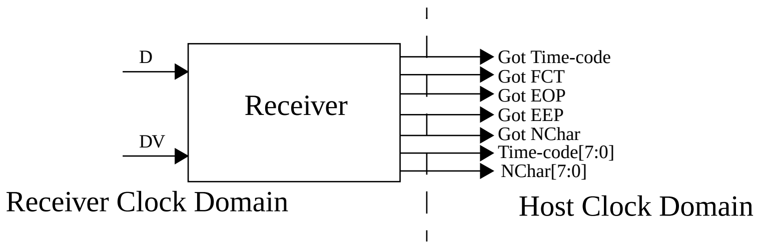


Figure 169. Schematic of the link interface receiver.

56.3.4 Dual port support

The core can be configured to include an additional SpaceWire port. With dual ports the transmitter drives an additional pair of data/strobe output signals and one extra receiver is added to handle a second pair of data/strobe input signals.

One of the ports is set as active (how the active port is selected is explained below) and the transmitter drives the data/strobe signals of the active port with the actual output values as explained in section 56.3.2. The inactive port is driven with zero on both data and strobe.

Both receivers will always be active but only the active port’s interface signals (see figure 169) will be propagated to the link interface FSM. Each time the active port is changed, the link will be reset so that the new link is started in a controlled manner.

When the noportforce register is zero the portsel register bit selects the active link and when set to one it is determined by the current link activity. In the latter mode the port is changed when no activity is seen on the currently active link while there is activity on the deselected receive port. Activity is defined as a detected null. This definition is selected so that glitches (e.g. port unconnected) do not cause unwanted port switches.

56.3.5 Receiver PHY

The receiver supports four different input data recovery schemes: self-clocking (xor), sampling SDR, sampling DDR and the Aeroflex SpaceWire transceiver. These four recovery types are handled in the PHY module and data is presented to the receiver as a data and data-valid signal. This part of the receiver must often be constrained and placing it in a separate module makes this process easier with the most common synthesis tools. The input type is selected using a VHDL generic. More information about the PHY can be found in the GRSPW2_PHY section of the grip manual.

56.3.6 Time interface

The time interface is used for sending Time-codes over the SpaceWire network and consists of a time-counter register, time-ctrl register, tick-in signal, tick-out signal, tick-in register field and a tick-out register field. There are also two control register bits which enable the time receiver and transmitter respectively.

Each Time-code sent from the grspw is a concatenation of the time-ctrl and the time-counter register. There is a timetxen bit which is used to enable Time-code transmissions. It is not possible to send time-codes if this bit is zero.

Received Time-codes are stored to the same time-ctrl and time-counter registers which are used for transmission. The timerxen bit in the control register is used for enabling time-code reception. No time-codes will be received if this bit is zero.

The two enable bits are used for ensuring that a node will not (accidentally) both transmit and receive time-codes which violates the SpaceWire standard. It also ensures that a the master sending time-codes on a network will not have its time-counter overwritten if another (faulty) node starts sending time-codes.

The time-counter register is set to 0 after reset and is incremented each time the tick-in signal is asserted for one clock-period and the timetxen bit is set. This also causes the link interface to send the new value on the network. Tick-in can be generated either by writing a one to the register field or by asserting the tick-in signal. A Tick-in should not be generated too often since if the time-code after the previous Tick-in has not been sent the register will not be incremented and no new value will be sent. The tick-in field is automatically cleared when the value has been sent and thus no new ticks should be generated until this field is zero. If the tick-in signal is used there should be at least 4 system-clock and 25 transmit-clock cycles between each assertion.

A tick-out is generated each time a valid time-code is received and the timerxen bit is set. When the tick-out is generated the tick-out signal will be asserted one clock-cycle and the tick-out register field is asserted until it is cleared by writing a one to it.

The current time counter value can be read from the time register. It is updated each time a Time-code is received and the timerxen bit is set. The same register is used for transmissions and can also be written directly from the APB interface.

The control bits of the Time-code are stored to the time-ctrl register when a Time-code is received whose time-count is one more than the nodes current time-counter register. The time-ctrl register can be read through the APB interface. The same register is used during time-code transmissions.

It is possible to have both the time-transmission and reception functions enabled at the same time.

56.4 Receiver DMA channels

The receiver DMA engine handles reception of data from the SpaceWire network to different DMA channels.

56.4.1 Address comparison and channel selection

Packets are received to different channels based on the address and whether a channel is enabled or not. When the receiver N-Char FIFO contains one or more characters, N-Chars are read by the receiver DMA engine. The first character is interpreted as the logical address and is compared with the addresses of each channel starting from 0. The packet will be stored to the first channel with an matching address. The complete packet including address and protocol ID but excluding EOP/EEP is stored to the memory address pointed to by the descriptors (explained later in this section) of the channel.

Each SpaceWire address register has a corresponding mask register. Only bits at an index containing a zero in the corresponding mask register are compared. This way a DMA channel can accept a range of addresses. There is a default address register which is used for address checking in all implemented DMA channels that do not have separate addressing enabled and for RMAP commands in the RMAP target. With separate addressing enabled the DMA channels' own address/mask register pair is used instead.

If an RMAP command is received it is only handled by the target if the default address register (including mask) matches the received address. Otherwise the packet will be stored to a DMA channel if one or more of them has a matching address. If the address does not match neither the default address nor one of the DMA channels' separate register, the packet is still handled by the RMAP target if enabled since it has to return the invalid address error code. The packet is only discarded (up to and including the next EOP/EEP) if an address match cannot be found and the RMAP target is disabled.

Packets, other than RMAP commands, that do not match neither the default address register nor the DMA channels' address register will be discarded. Figure 170 shows a flowchart of packet reception.

At least 2 non EOP/EEP N-Chars needs to be received for a packet to be stored to the DMA channel unless the promiscuous mode is enabled in which case 1 N-Char is enough. If it is an RMAP packet with hardware RMAP enabled 3 N-Chars are needed since the command byte determines where the packet is processed. Packets smaller than these sizes are discarded.

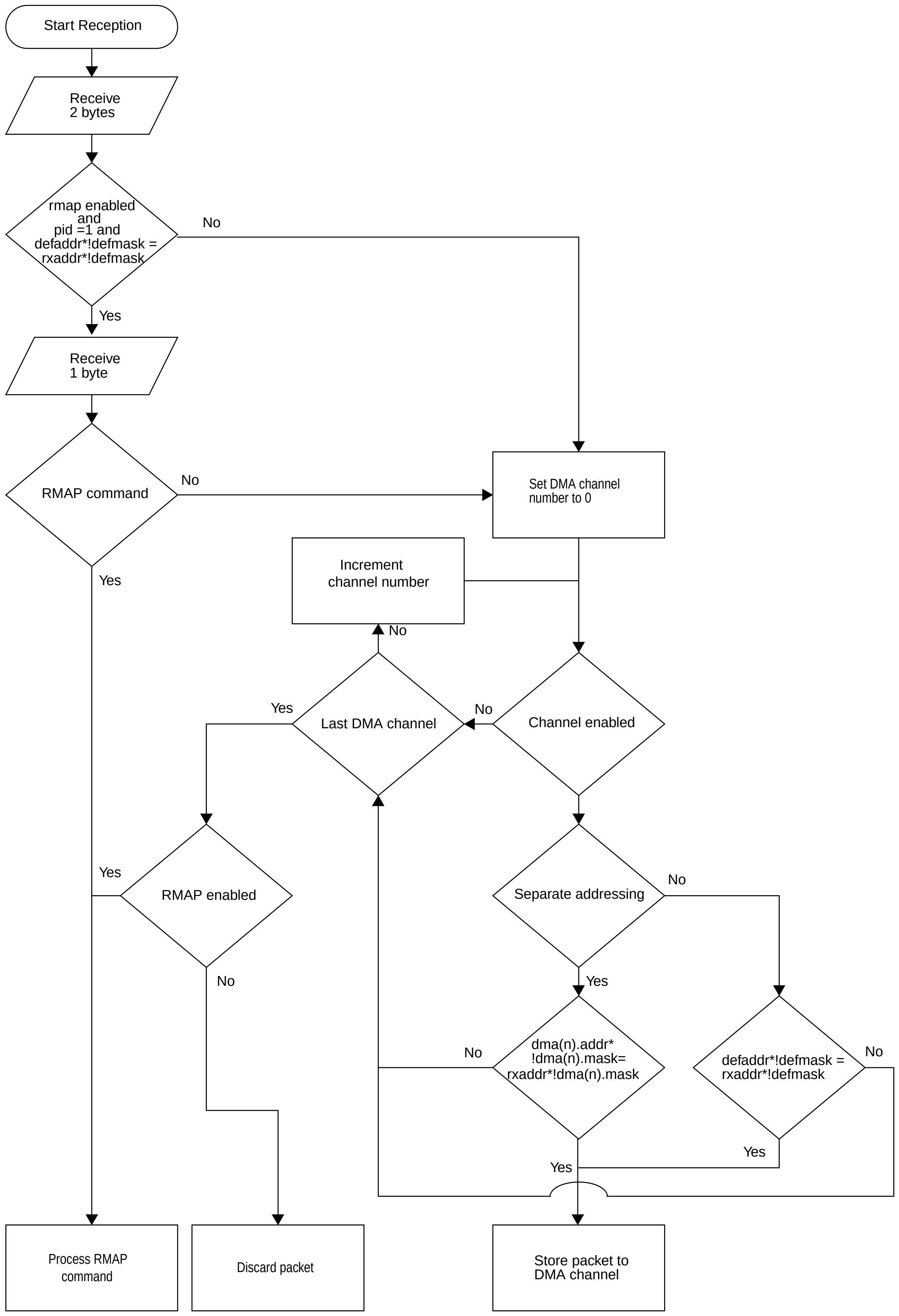


Figure 170. Flow chart of packet reception.

56.4.2 Basic functionality of a channel

Reception is based on descriptors located in a consecutive area in memory that hold pointers to buffers where packets should be stored. When a packet arrives at the core the channel which should receive it is first determined as described in the previous section. A descriptor is then read from the channels' descriptor area and the packet is stored to the memory area pointed to by the descriptor. Lastly, status is stored to the same descriptor and increments the descriptor pointer to the next one. The following sections will describe DMA channel reception in more detail.

56.4.3 Setting up the core for reception

A few registers need to be initialized before reception to a channel can take place. First the link interface need to be put in the run state before any data can be sent. The DMA channel has a maximum length register which sets the maximum packet size in bytes that can be received to this channel. Larger packets are truncated and the excessive part is spilled. If this happens an indication will be given in the status field of the descriptor. The minimum value for the receiver maximum length field is 4 and the value can only be incremented in steps of four bytes up to the maximum value 33554428. If the maximum length is set to zero the receiver will *not* function correctly.

Either the default address register or the channel specific address register (the accompanying mask register must also be set) needs to be set to hold the address used by the channel. A control bit in the DMA channel control register determines whether the channel should use default address and mask registers for address comparison or the channel's own registers. Using the default register the same address range is accepted as for other channels with default addressing and the RMAP target while the separate address provides the channel its own range. If all channels use the default registers they will accept the same address range and the enabled channel with the lowest number will receive the packet.

Finally, the descriptor table and control register must be initialized. This will be described in the two following sections.

56.4.4 Setting up the descriptor table address

The core reads descriptors from an area in memory pointed to by the receiver descriptor table address register. The register consists of a base address and a descriptor selector. The base address points to the beginning of the area and must start on a 1024 bytes aligned address. It is also limited to be 1024 bytes in size which means the maximum number of descriptors is 128 since the descriptor size is 8 bytes.

The descriptor selector points to individual descriptors and is increased by 1 when a descriptor has been used. When the selector reaches the upper limit of the area it wraps to the beginning automatically. It can also be set to wrap at a specific descriptor before the upper limit by setting the wrap bit in the descriptor. The idea is that the selector should be initialized to 0 (start of the descriptor area) but it can also be written with another 8 bytes aligned value to start somewhere in the middle of the area. It will still wrap to the beginning of the area.

If one wants to use a new descriptor table the receiver enable bit has to be cleared first. When the rxactive bit for the channel is cleared it is safe to update the descriptor table register. When this is finished and descriptors are enabled the receiver enable bit can be set again.

56.4.5 Enabling descriptors

As mentioned earlier one or more descriptors must be enabled before reception can take place. Each descriptor is 8 byte in size and the layout can be found in the tables below. The descriptors should be written to the memory area pointed to by the receiver descriptor table address register. When new descriptors are added they must always be placed after the previous one written to the area. Otherwise they will not be noticed.

A descriptor is enabled by setting the address pointer to point at a location where data can be stored and then setting the enable bit. The WR bit can be set to cause the selector to be set to zero when reception has finished to this descriptor. IE should be set if an interrupt is wanted when the reception has finished. The DMA control register interrupt enable bit must also be set for an interrupt to be generated.

The descriptor packet address should be word aligned. All accesses on the bus are word accesses so complete words will always be overwritten regardless of whether all 32-bit contain received data. Also if the packet does not end on a word boundary the complete word containing the last data byte will be overwritten. If the rxunaligned or rmap VHDL generic is set to 1 this restriction is removed and any number of bytes can be received to any packet address without excessive bytes being overwritten.

Table 741. GRSPW receive descriptor word 0 (address offset 0x0)

31	30	29	28	27	26	25	24 ... 0
TR	DC	HC	EP	IE	WR	EN	PACKETLENGTH

31	Truncated (TR) - Packet was truncated due to maximum length violation.
30	Data CRC (DC) - 1 if a CRC error was detected for the data and 0 otherwise.
29	Header CRC (HC) - 1 if a CRC error was detected for the header and 0 otherwise.
28	EEP termination (EP) - This packet ended with an Error End of Packet character.
27	Interrupt enable (IE) - If set, an interrupt will be generated when a packet has been received if the receive interrupt enable bit in the DMA channel control register is set.
26	Wrap (WR) - If set, the next descriptor used by the GRSPW will be the first one in the descriptor table (at the base address). Otherwise the descriptor pointer will be increased with 0x8 to use the descriptor at the next higher memory location. The descriptor table is limited to 1 kbytes in size and the pointer will be automatically wrap back to the base address when it reaches the 1 kbytes boundary.
25	Enable (EN) - Set to one to activate this descriptor. This means that the descriptor contains valid control values and the memory area pointed to by the packet address field can be used to store a packet.
24: 0	Packet length (PACKETLENGTH) - The number of bytes received to this buffer. Only valid after EN has been set to 0 by the GRSPW.

Table 742. GRSPW receive descriptor word 1 (address offset 0x4)

31 ... 0
PACKETADDRESS

31: 0	Packet address (PACKETADDRESS) - The address pointing at the buffer which will be used to store the received packet. If the rxunaligned and rmap VHDL generics are both set to zero only bit 31 to 2 are used.

56.4.6 Setting up the DMA control register

The final step to receive packets is to set the control register in the following steps: The receiver must be enabled by setting the rxen bit in the DMA control register (see section 56.9). This can be done anytime and before this bit is set nothing will happen. The rxdescav bit in the DMA control register is then set to indicate that there are new active descriptors. This must always be done after the descriptors have been enabled or the core might not notice the new descriptors. More descriptors can be activated when reception has already started by enabling the descriptors and writing the rxdescav bit. When these bits are set reception will start immediately when data is arriving.

56.4.7 The effect to the control bits during reception

When the receiver is disabled all packets going to the DMA-channel are discarded if the packet’s address does not fall into the range of another DMA channel. If the receiver is enabled and the address falls into the accepted address range, the next state is entered where the rxdescav bit is checked. This bit indicates whether there are active descriptors or not and should be set by the external application using the DMA channel each time descriptors are enabled as mentioned above. If the rxdescav bit is ‘0’ and the nospill bit is ‘0’ the packets will be discarded. If nospill is one the grspw waits until rxdescav is set and the characters are kept in the N-Char fifo during this time. If the fifo becomes full further N-char transmissions are inhibited by stopping the transmission of FCTs.

When rxdescav is set the next descriptor is read and if enabled the packet is received to the buffer. If the read descriptor is not enabled, rxdescav is set to ‘0’ and the packet is spilled depending on the value of nospill.

The receiver can be disabled at any time and will stop packets from being received to this channel. If a packet is currently received when the receiver is disabled the reception will still be finished. The rxdescav bit can also be cleared at any time. It will not affect any ongoing receptions but no more descriptors will be read until it is set again. Rxdescav is also cleared by the core when it reads a disabled descriptor.

56.4.8 Status bits

When the reception of a packet is finished the enable bit in the current descriptor is set to zero. When enable is zero, the status bits are also valid and the number of received bytes is indicated in the length field. The DMA control register contains a status bit which is set each time a packet has been received. The core can also be made to generate an interrupt for this event.

RMAP CRC logic is included in the implementation if the rmapcrc or rmap VHDL generic set to 1. The RMAP CRC calculation is always active for all received packets and all bytes except the EOP/EEP are included. The packet is always assumed to be a RMAP packet and the length of the header is determined by checking byte 3 which should be the command field. The calculated CRC value is then checked when the header has been received (according to the calculated number of bytes) and if it is non-zero the HC bit is set indicating a header CRC error.

The CRC value is not set to zero after the header has been received, instead the calculation continues in the same way until the complete packet has been received. Then if the CRC value is non-zero the DC bit is set indicating a data CRC error. This means that the core can indicate a data CRC error even if the data field was correct when the header CRC was incorrect. However, the data should not be used when the header is corrupt and therefore the DC bit is unimportant in this case. When the header is not corrupted the CRC value will always be zero when the calculation continues with the data field and the behaviour will be as if the CRC calculation was restarted

If the received packet is not of RMAP type the header CRC error indication bit cannot be used. It is still possible to use the DC bit if the complete packet is covered by a CRC calculated using the RMAP CRC definition. This is because the core does not restart the calculation after the header has been received but instead calculates a complete CRC over the packet. Thus any packet format with one CRC at the end of the packet calculated according to RMAP standard can be checked using the DC bit.

If the packet is neither of RMAP type nor of the type above with RMAP CRC at the end, then both the HC and DC bits should be ignored.

56.4.9 Error handling

If a packet reception needs to be aborted because of congestion on the network, the suggested solution is to set link disable to ‘1’. Unfortunately, this will also cause the packet currently being transmitted to be truncated but this is the only safe solution since packet reception is a passive operation depending on the transmitter at the other end. A channel reset bit could be provided but is not a satisfactory solution since the untransmitted characters would still be in the transmitter node. The next character (somewhere in the middle of the packet) would be interpreted as the node address which would probably cause the packet to be discarded but not with 100% certainty. Usually this action is performed when a reception has stuck because of the transmitter not providing more data. The channel reset would not resolve this congestion.

If an AHB error occurs during reception the current packet is spilled up to and including the next EEP/EOP and then the currently active channel is disabled and the receiver enters the idle state. A bit in the channels control/status register is set to indicate this condition.

56.4.10 Promiscuous mode

The core supports a promiscuous mode where all the data received is stored to the first DMA channel enabled regardless of the node address and possible early EOPs/EEPs. This means that all non-eop/eep N-Chars received will be stored to the DMA channel. The rxmaxlength register is still checked and packets exceeding this size will be truncated.

RMAP commands will still be handled by it when promiscuous mode is enabled if the rmapen bit is set. If it is cleared, RMAP commands will also be stored to a DMA channel.

56.5 Transmitter DMA channels

The transmitter DMA engine handles transmission of data from the DMA channels to the SpaceWire network. Each receive channel has a corresponding transmit channel which means there can be up to 4 transmit channels. It is however only necessary to use a separate transmit channel for each receive channel if there are also separate entities controlling the transmissions. The use of a single channel with multiple controlling entities would cause them to corrupt each other’s transmissions. A single channel is more efficient and should be used when possible.

Multiple transmit channels with pending transmissions are arbitrated in a round-robin fashion.

56.5.1 Basic functionality of a channel

A transmit DMA channel reads data from the AHB bus and stores them in the transmitter FIFO for transmission on the SpaceWire network. Transmission is based on the same type of descriptors as for the receiver and the descriptor table has the same alignment and size restrictions. When there are new descriptors enabled the core reads them and transfer the amount data indicated.

56.5.2 Setting up the core for transmission

Four steps need to be performed before transmissions can be done with the core. First the link interface must be enabled and started by writing the appropriate value to the ctrl register. Then the address to the descriptor table needs to be written to the transmitter descriptor table address register and one or more descriptors must also be enabled in the table. Finally, the txen bit in the DMA control register is written with a one which triggers the transmission. These steps will be covered in more detail in the next sections.

56.5.3 Enabling descriptors

The descriptor table address register works in the same way as the receiver’s corresponding register which was covered in section 56.4. The maximum size is 1024 bytes as for the receiver but since the descriptor size is 16 bytes the number of descriptors is 64.

To transmit packets one or more descriptors have to be initialized in memory which is done in the following way: The number of bytes to be transmitted and a pointer to the data has to be set. There are two different length and address fields in the transmit descriptors because there are separate pointers for header and data. If a length field is zero the corresponding part of a packet is skipped and if both are zero no packet is sent. The maximum header length is 255 bytes and the maximum data length is 16 Mbyte - 1. When the pointer and length fields have been set the enable bit should be set to enable the descriptor. This must always be done last. The other control bits must also be set before enabling the descriptor.

The transmit descriptors are 16 bytes in size so the maximum number in a single table is 64. The different fields of the descriptor together with the memory offsets are shown in the tables below.

The HC bit should be set if RMAP CRC should be calculated and inserted for the header field and correspondingly the DC bit should be set for the data field. This field is only used by the GRSPW when the CRC logic is available (*rmap* or *rmapcrc* VHDL generic set to 1). The header CRC will be calculated from the data fetched from the header pointer and the data CRC is generated from data fetched from the data pointer. The CRCs are appended after the corresponding fields. The NON-CRC bytes field is set to the number of bytes in the beginning of the header field that should not be included in the CRC calculation.

The CRCs are sent even if the corresponding length is zero, but when both lengths are zero no packet is sent not even an EOP.

56.5.4 Starting transmissions

When the descriptors have been initialized, the transmit enable bit in the DMA control register has to be set to tell the core to start transmitting. New descriptors can be activated in the table on the fly (while transmission is active). Each time a set of descriptors is added the transmit enable register bit should be set. This has to be done because each time the core encounters a disabled descriptor this register bit is set to 0.

Table 743. GRSPW transmit descriptor word 0 (address offset 0x0)

31 – 18	17	16	15	14	13	12	11 – 8	7 – 0
RESERVED	DC	HC	LE	IE	WR	EN	NONCRCLEN	HEADERLEN

31: 18	RESERVED
17	Append data CRC (DC) - Append CRC calculated according to the RMAP specification after the data sent from the data pointer. The CRC covers all the bytes from this pointer. A null CRC will be sent if the length of the data field is zero.
16	Append header CRC (HC) - Append CRC calculated according to the RMAP specification after the data sent from the header pointer. The CRC covers all bytes from this pointer except a number of bytes in the beginning specified by the non-crc bytes field. The CRC will not be sent if the header length field is zero.
15	Link error (LE) - A Link error occurred during the transmission of this packet.
14	Interrupt enable (IE) - If set, an interrupt will be generated when the packet has been transmitted and the transmitter interrupt enable bit in the DMA control register is set.
13	Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x10 to use the descriptor at the next higher memory location.
12	Enable (EN) - Enable transmitter descriptor. When all control fields (address, length, wrap and crc) are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the transmission. The GRSPW clears this bit when the transmission has finished.
11: 8	Non-CRC bytes (NONCRCLEN)- Sets the number of bytes in the beginning of the header which should not be included in the CRC calculation. This is necessary when using path addressing since one or more bytes in the beginning of the packet might be discarded before the packet reaches its destination.
7: 0	Header length (HEADERLEN) - Header Length in bytes. If set to zero, the header is skipped.

Table 744. GRSPW transmit descriptor word 1 (address offset 0x4)

31 … 0
HEADERADDRESS

31: 0	Header address (HEADERADDRESS) - Address from where the packet header is fetched. Does not need to be word aligned.

Table 745. GRSPW transmit descriptor word 2 (address offset 0x8)

31 … 24	23 … 0
RESERVED	DATALEN

31: 24	RESERVED
23: 0	Data length (DATALEN) - Length in bytes of data part of packet. If set to zero, no data will be sent. If both data- and header-lengths are set to zero no packet will be sent.

Table 746. GRSPW transmit descriptor word 3(address offset 0xC)

31 … 0
DATAADDRESS

31: 0	Data address (DATAADDRESS) - Address from where data is read. Does not need to be word aligned.

56.5.5 The transmission process

When the txen bit is set the core starts reading descriptors immediately. The number of bytes indicated are read and transmitted. When a transmission has finished, status will be written to the first field of the descriptor and a packet sent bit is set in the DMA control register. If an interrupt was requested it will also be generated. Then a new descriptor is read and if enabled a new transmission starts, otherwise the transmit enable bit is cleared and nothing will happen until it is enabled again.

56.5.6 The descriptor table address register

The internal pointer which is used to keep the current position in the descriptor table can be read and written through the APB interface. This pointer is set to zero during reset and is incremented each time a descriptor is used. It wraps automatically when the 1024 bytes limit for the descriptor table is reached or it can be set to wrap earlier by setting a bit in the current descriptor.

The descriptor table register can be updated with a new table anytime when no transmission is active. No transmission is active if the transmit enable bit is zero and the complete table has been sent or if the table is aborted (explained below). If the table is aborted one has to wait until the transmit enable bit is zero before updating the table pointer.

56.5.7 Error handling

Abort Tx

The DMA control register contains a bit called Abort TX which if set causes the current transmission to be aborted, the packet is truncated and an EEP is inserted. This is only useful if the packet needs to be aborted because of congestion on the SpaceWire network. If the congestion is on the AHB bus this will not help (This should not be a problem since AHB slaves should have a maximum of 16 wait-states). The aborted packet will have its LE bit set in the descriptor. The transmit enable register bit is also cleared and no new transmissions will be done until the transmitter is enabled again.

AHB error

When an AHB error is encountered during transmission the currently active DMA channel is disabled and the transmitter goes to the idle mode. A bit in the DMA channel’s control/status register is set to indicate this error condition and, if enabled, an interrupt will also be generated. Further error handling depends on what state the transmitter DMA engine was in when the AHB error occurred. If the descriptor was being read the packet transmission had not been started yet and no more actions need to be taken.

If the AHB error occurs during packet transmission the packet is truncated and an EEP is inserted. Lastly, if it occurs when status is written to the descriptor the packet has been successfully transmitted but the descriptor is not written and will continue to be enabled (this also means that no error bits are set in the descriptor for AHB errors).

The client using the channel has to correct the AHB error condition and enable the channel again. No more AHB transfers are done again from the same unit (receiver or transmitter) which was active during the AHB error until the error state is cleared and the unit is enabled again.

Link error

When a link error occurs during the transmission the remaining part of the packet is discarded up to and including the next EOP/EEP. When this is done status is immediately written (with the LE bit set) and the descriptor pointer is incremented. The link will be disconnected when the link error occurs but the grspw will automatically try to connect again provided that the link-start bit is asserted and the link-disabled bit is deasserted. If the LE bit in the DMA channel’s control register is not set the transmitter DMA engine will wait for the link to enter run-state and start a new transmission immediately when possible if packets are pending. Otherwise the transmitter will be disabled when a link error occurs during the transmission of the current packet and no more packets will be transmitted until it is enabled again immediately when possible if packets are pending.

56.6 RMAP

The Remote Memory Access Protocol (RMAP) is used to implement access to resources in the node via the SpaceWire Link. Some common operations are reading and writing to memory, registers and FIFOs. The core has an optional hardware RMAP target which is enabled with a VHDL generic. This section describes the basics of the RMAP protocol and the target implementation.

56.6.1 Fundamentals of the protocol

RMAP is a protocol which is designed to provide remote access via a SpaceWire network to memory mapped resources on a SpaceWire node. It has been assigned protocol ID 0x01. It provides three operations write, read and read-modify-write. These operations are posted operations which means that a source does not wait for an acknowledge or reply. It also implies that any number of operations can be outstanding at any time and that no timeout mechanism is implemented in the protocol. Time-outs must be implemented in the user application which sends the commands. Data payloads of up to 16 Mb - 1 is supported in the protocol. A destination can be requested to send replies and to verify data before executing an operation. A complete description of the protocol is found in the RMAP standard.

56.6.2 Implementation

The core includes a target for RMAP commands which processes all incoming packets with protocol ID = 0x01, type field (bit 7 and 6 of the 3rd byte in the packet) equal to 01b and an address falling in the range set by the default address and mask register. When such a packet is detected it is not stored to the DMA channel, instead it is passed to the RMAP receiver.

The core implements all three commands defined in the standard with some restrictions. Support is only provided for 32-bit big-endian systems. This means that the first byte received is the msb in a word. The target will not receive RMAP packets using the extended protocol ID which are always dumped to the DMA channel.

The RMAP receiver processes commands. If they are correct and accepted the operation is performed on the AHB bus and a reply is formatted. If an acknowledge is requested the RMAP transmitter automatically send the reply. RMAP transmissions have priority over DMA channel transmissions.

There is a user accessible destination key register which is compared to destination key field in incoming packets. If there is a mismatch and a reply has been requested the error code in the reply is set to 3. Replies are sent if and only if the ack field is set to ‘1’.

When a failure occurs during a bus access the error code is set to 1 (General Error). There is predetermined order in which error-codes are set in the case of multiple errors in the core. It is shown in table 747.

*Table 747.*The order of error detection in case of multiple errors in the GRSPW. The error detected first has number 1.

Detection Order	Error Code	Error
1	12	Invalid destination logical address
2	2	Unused RMAP packet type or command code
3	3	Invalid destination key
4	9	Verify buffer overrun
5	11	RMW data length error
6	10	Authorization failure
7*	1	General Error (AHB errors during non-verified writes)
8	5/7	Early EOP / EEP (if early)
9	4	Invalid Data CRC
10	1	General Error (AHB errors during verified writes or RMW)
11	7	EEP
12	6	Cargo Too Large
*The AHB error is not guaranteed to be detected before Early EOP/EEP or Invalid Data CRC. For very long accesses the AHB error detection might be delayed causing the other two errors to appear first.		

Read accesses are performed on the fly, that is they are not stored in a temporary buffer before transmitting. This means that the error code 1 will never be seen in a read reply since the header has already been sent when the data is read. If the AHB error occurs the packet will be truncated and ended with an EEP.

Errors up to and including Invalid Data CRC (number 8) are checked before verified commands. The other errors do not prevent verified operations from being performed.

The details of the support for the different commands are now presented. All defined commands which are received but have an option set which is not supported in this specific implementation will not be executed and a possible reply is sent with error code 10.

56.6.3 Write commands

The write commands are divided into two subcategories when examining their capabilities: verified writes and non-verified writes. Verified writes have a length restriction of 4 bytes and the address must be aligned to the size. That is 1 byte writes can be done to any address, 2 bytes must be halfword aligned, 3 bytes are not allowed and 4 bytes writes must be word aligned. Since there will always be only on AHB operation performed for each RMAP verified write command the incrementing address bit can be set to any value.

Non-verified writes have no restrictions when the incrementing bit is set to 1. If it is set to 0 the number of bytes must be a multiple of 4 and the address word aligned. There is no guarantee how many words will be written when early EOP/EEP is detected for non-verified writes.

56.6.4 Read commands

Read commands are performed on the fly when the reply is sent. Thus if an AHB error occurs the packet will be truncated and ended with an EEP. There are no restrictions for incrementing reads but non-incrementing reads have the same alignment restrictions as non-verified writes. Note that the “Authorization failure” error code will be sent in the reply if a violation was detected even if the length field was zero. Also note that no data is sent in the reply if an error was detected i.e. if the status field is non-zero.

56.6.5 RMW commands

All read-modify-write sizes are supported except 6 which would have caused 3 B being read and written on the bus. The RMW bus accesses have the same restrictions as the verified writes. As in the verified write case, the incrementing bit can be set to any value since only one AHB bus operation will be performed for each RMW command. Cargo too large is detected after the bus accesses so this error will not prevent the operation from being performed. No data is sent in a reply if an error is detected i.e. the status field is non-zero.

56.6.6 Control

The RMAP target mostly runs in the background without any external intervention, but there are a few control possibilities.

There is an enable bit in the control register of the core which can be used to completely disable the RMAP target. When it is set to ‘0’ no RMAP packets will be handled in hardware, instead they are all stored to the DMA channel.

There is a possibility that RMAP commands will not be performed in the order they arrive. This can happen if a read arrives before one or more writes. Since the target stores replies in a buffer with more than one entry several commands can be processed even if no replies are sent. Data for read replies is read when the reply is sent and thus writes coming after the read might have been performed already if there was congestion in the transmitter. To avoid this the RMAP buffer disable bit can be set to force the target to only use one buffer which prevents this situation.

The last control option for the target is the possibility to set the destination key which is found in a separate register.

*Table 748.*GRSPW hardware RMAP handling of different packet type and command fields.

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Command**	**Action**
Reserved	**Command / Response**	**Write / Read**	**Verify data before write**	**Acknow-ledge**	**Increment Address**		
0	0	-	-	-	-	Response	Stored to DMA-channel.
0	1	0	0	0	0	Not used	Does nothing. No reply is sent.
0	1	0	0	0	1	Not used	Does nothing. No reply is sent.
0	1	0	0	1	0	Read single address	Executed normally. Address has to be word aligned and data size a multiple of four. Reply is sent. If alignment restrictions are violated error code is set to 10.
0	1	0	0	1	1	Read incrementing address.	Executed normally. No restrictions. Reply is sent.
0	1	0	1	0	0	Not used	Does nothing. No reply is sent.
0	1	0	1	0	1	Not used	Does nothing. No reply is sent.
0	1	0	1	1	0	Not used	Does nothing. Reply is sent with error code 2.
0	1	0	1	1	1	Read-Modify-Write incrementing address	Executed normally. If length is not one of the allowed rmw values nothing is done and error code is set to 11. If the length was correct, alignment restrictions are checked next. 1 byte can be rmw to any address. 2 bytes must be halfword aligned. 3 bytes are not allowed. 4 bytes must be word aligned. If these restrictions are violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	0	0	0	Write, single-address, do not verify before writing, no acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done. No reply is sent.
0	1	1	0	0	1	Write, incrementing address, do not verify before writing, no acknowledge	Executed normally. No restrictions. No reply is sent.
0	1	1	0	1	0	Write, single-address, do not verify before writing, send acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.

*Table 748.*GRSPW hardware RMAP handling of different packet type and command fields.

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Command**	**Action**
Reserved	**Command / Response**	**Write / Read**	**Verify data before write**	**Acknow-ledge**	**Increment Address**		
0	1	1	0	1	1	Write, incrementing address, do not verify before writing, send acknowledge	Executed normally. No restrictions. If AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	0	0	Write, single address, verify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restrictions apply as for rmw. No reply is sent.
0	1	1	1	0	1	Write, incrementing address, verify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restrictions apply as for rmw. If they are violated nothing is done. No reply is sent.
0	1	1	1	1	0	Write, single address, verify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	1	1	Write, incrementing address, verify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
1	0	-	-	-	-	Unused	Stored to DMA-channel.
1	1	-	-	-	-	Unused	Stored to DMA-channel.

56.7 AMBA interface

The AMBA interface consists of an APB interface, an AHB master interface and DMA FIFOs. The APB interface provides access to the user registers which are described in section 56.9. The DMA engines have 32-bit wide FIFOs to the AHB master interface which are used when reading and writing to the bus.

The transmitter DMA engine reads data from the bus in bursts which are half the FIFO size in length. A burst is always started when the FIFO is half-empty or if it can hold the last data for the packet. The burst containing the last data might have shorter length if the packet is not an even number of bursts in size.

The receiver DMA works in the same way except that it checks if the FIFO is half-full and then performs a burst write to the bus which is half the fifo size in length. The last burst might be shorter. If the rmap or rxunaligned VHDL generics are set to 1 the interface also handles byte accesses. Byte accesses are used for non word-aligned buffers and/or packet lengths that are not a multiple of four bytes. There might be 1 to 3 single byte writes when writing the beginning and end of the received packets.

56.7.1 APB slave interface

As mentioned above, the APB interface provides access to the user registers which are 32-bits in width. The accesses to this interface are required to be aligned word accesses. The result is undefined if this restriction is violated.

56.7.2 AHB master interface

The core contains a single master interface which is used by both the transmitter and receiver DMA engines. The arbitration algorithm between the channels is done so that if the current owner requests the interface again it will always acquire it. This will not lead to starvation problems since the DMA engines always deassert their requests between accesses.

The AHB accesses are always word accesses (HSIZE = 0x010) of type incremental burst with unspecified length (HBURST = 0x001) if VHDL generics rmap and rxunaligned are disabled. The AHB accesses can be of size byte, halfword and word (HSIZE = 0x000, 0x001, 0x010) otherwise. Byte and halfword accesses are always NONSEQ. Note that read accesses are always word accesses (HSIZE = 0x010), which can result in destructive read.

The burst length will be half the AHB FIFO size except for the last transfer for a packet which might be smaller. Shorter accesses are also done during descriptor reads and status writes.

The AHB master also supports non-incrementing accesses where the address will be constant for several consecutive accesses. HTRANS will always be NONSEQ in this case while for incrementing accesses it is set to SEQ after the first access. This feature is included to support non-incrementing reads and writes for RMAP.

If the core does not need the bus after a burst has finished there will be one wasted cycle (HTRANS = IDLE).

BUSY transfer types are never requested and the core provides full support for ERROR, RETRY and SPLIT responses.

56.8 Synthesis and hardware

56.8.1 Clock-generation

The receiver module found in figure 166 should be clocked with a clock generated by the grspw2_phy module. See the example instantiation in this section and the grspw2_phy section of the grip manual for more information on how to connect this clock.

The transmitter clock is generated from the txclk input. A separate clock input is used to allow the transmitter to be run at much higher frequencies than the system clock. The SpaceWire node contains a clock-divider which divides the txclk signal to the wanted frequency. The transmitter clock should be 10 MHz during initialization and any frequency above 2 MHz in the run-state.

There is an input signal called clkdiv10 which sets the reset values for the user accessible clock divisor registers. There is one register value which is used during initialisation and one which is used in run-state The resulting tx clock frequency will be txclk/(clock divisor value+1). So if no clock division is wanted, the clock divisor should be set to 0.

Since only integer values are allowed for the clock division and the required init-frequency is 10 Mhz the frequency of the txclk input must be a multiple of 10 MHz. The clock divisor value is 8-bits wide so the maximum txclk frequency supported is 2.56 GHz (note that there is also a restriction on the relation between the system and transmit clock frequencies).

56.8.2 Timers

There are two timers in the grspw: one for generating the 6.4/12.8 us periods and one for disconnect timing.

The timeout periods are generated from the tx clock whose frequency must be at least 10 MHz to guarantee disconnect timing limits. The same clock divisor is used as for the tx clock during initialisation so it must be set correctly for the link timing to work.

56.8.3 Synchronization

The transmitter and receiver bit rates can be eight times higher than the system clock frequency. This includes a large margin for clock skew and jitter so it might be possible to run at even higher rate differences. Note also that the receiver clocks data at both negative and positive edges for the input modes 0 and 1 so the bitrate is twice the clock frequency. There is no direct relationship between bitrate and frequency for the sampling modes.

The clock synchronization is just one limiting factor for the clock frequency, it might for example not be possible to achieve the highest possible frequency for certain technologies.

The asynchronous reset to the receiver clock domain has to have a maximum delay of one receiver clock cycle to ensure correct operation. This is needed because the receiver uses a completely asynchronous reset. To make sure that nothing bad happens the is a synchronous reset guard which prevents any signals from being assigned before all registers have their reset signals released.

In the sampling modes this asynchronous reset can be removed if both the receiver and transmitter runs on the same clock. In that case set the RXTX_SAMECLK generic to 1.

56.8.4 Fault-tolerance

The core can optionally be implemented with fault-tolerance against SEU errors in the FIFO memories. The fault-tolerance is enabled through the *ft* VHDL generic. Possible options are byte parity protection (*ft = 1*) or TMR registers (*ft = 2*). Note: the GPL version of GRLIB does not include fault-tolerance, and the core will not work unless the *ft* VHDL generic is 0.

56.8.5 Synthesis

Since the receiver and transmitter may run on very high frequency clocks their clock signals have been coupled through a clock buffer with a technology wrapper. This clock buffer will utilize a low skew net available in the selected technology for the clock.

The clock buffer will also enable most synthesis tools to recognize the clocks and it is thus easier to find them and place constraints on them. The fact there are three clock domains in the core of which all are possibly high frequency clocks makes it necessary to declare all paths between the clock domains as false paths.

In Synplify this is most easily done by declaring all the clocks to be in different clockgroups in the sdc file (if Synplify does not automatically put them in different groups). This will disable any timing considerations between the clock domains and these constraints will also propagate to the place and route tool.

The type of clock buffer is selectable with a VHDL generic and the value zero provides a normal feed through which lets the synthesis tool infer the type of net used.

56.8.6 Technology mapping

The core has three generics for technology mapping: *tech*, *techfifo* and *memtech*. *Tech* selects the technology used for the clock buffers and also adds reset to some registers for technologies where they would otherwise cause problems with gate-level simulations. *Techfifo* selects whether *memtech* should be used to select the technology for the FIFO memories (the RMAP buffer is not affected by the this generic) or if they should be inferred. *Tech* and *memtech* can be set to any value from 0 to NTECH as defined in the GRLIB.TECH package.

56.8.7 RAM usage

The core maps all RAM memories on the syncram_2p component if the *ft* generic is 0 and to the syncram_2pft component for other values. The syncrams are located in the technology mapping library (TECHMAP). The organization of the different memories are described below. If techfifo and/or memtech is set to 0 the synthesis tool will infer the memories. Either RAM blocks or flip-flops will be used depending on the tool and technology. The number of flip-flops used is *syncram depth x syncram width* for all the different memories. The receiver AHB FIFO with fifosize 32 will for example use 1024 flips-flops.

Receiver ahb FIFO

The receiver AHB fifo consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 749 shows the syncram organization for the allowed configurations.

*Table 749.*syncram_2p sizes for GRSPW receiver AHB FIFO.

Fifosize	**Syncram_2p organization**
4	4x32
8	8x32
16	16x32
32	32x32

Transmitter ahb FIFO

The transmitter AHB fifo consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 750 shows the syncram organization for the allowed configurations.

*Table 750.*syncram_2p sizes for transmitter AHB FIFO.

Fifosize	**Syncram_2p organization**
4	4x32
8	8x32
16	16x32
32	32x32

Receiver N-Char FIFO

The receiver N-Char fifo consists of one syncram_2p block with a width of 9-bits. The depth is determined by the configured FIFO depth. Table 751 shows the syncram organization for the allowed configurations.

*Table 751.*syncram_2p sizes for the receiver N-Char FIFO.

Fifosize	**Syncram_2p organization**
16	16x9
32	32x9
64	64x9

RMAP buffer

The RMAP buffer consists of one syncram_2p block with a width of 8-bits. The depth is determined by the number of configured RMAP buffers. Table 752 shows the syncram organization for the allowed configurations.

*Table 752.*syncram_2p sizes for RMAP buffer memory.

RMAP buffers	**Syncram_2p organization**
2	64x8
4	128x8
8	256x8

56.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 753.*GRSPW registers

APB address offset	**Register**
0x0	Control
0x4	Status/Interrupt-source
0x8	Node address
0xC	Clock divisor
0x10	Destination key
0x14	Time
0x20	DMA channel 1 control/status
0x24	DMA channel 1 rx maximum length
0x28	DMA channel 1 transmit descriptor table address.
0x2C	DMA channel 1 receive descriptor table address.
0x30	DMA channel 1 address register
0x34	Unused
0x38	Unused
0x3C	Unused
0x40 - 0x5C	DMA channel 2 registers
0x60 - 0x7C	DMA channel 3 registers
0x80 - 0x9C	DMA channel 4 registers

Table 754. GRSPW control register

31	30	29	28 27	26	25 24 23 22	21	20	19 18	17	16	15 14 13 12	11	10	9	8	7	6	5	4	3	2	1	0
RA	RX	RC	NCH	PO	RESERVED	PS	NP		RD	RE	RESERVED	TR	TT	LI	TQ		RS	PM	TI	IE	AS	LS	LD

31	RMAP available (RA) - Set to one if the RMAP target is available. Only readable.
30	RX unaligned access (RX) - Set to one if unaligned writes are available for the receiver. Only readable.
29	RMAP CRC available (RC) - Set to one if RMAP CRC is enabled in the core. Only readable.
28: 27	Number of DMA channels (NCH) - The number of available DMA channels minus one (Number of channels = NCH+1).
26	Number of ports (PO) - The number of available SpaceWire ports minus one.
25: 23	RESERVED
22	Loop-back enable. The value of this bit is driven on the swno.loobpack output.
21	Port select (PS) - Selects the active port when the no port force bit is zero. ‘0’ selects the port connected to data and strobe on index 0 while ‘1’ selects index 1. Only available if the ports VHDL generic is set to 2. Reset value: ‘0’.
20	No port force (NP) - Disable port force. When disabled the port select bit cannot be used to select the active port. Instead, it is automatically selected by checking the activity on the respective receive links. Only available if the ports VHDL generic is set to 2. Reset value: ‘0’ if the RMAP command handler is not available. If available the reset value is set to the value of the rmapen input signal.
19: 18	RESERVED
17	RMAP buffer disable (RD) - If set only one RMAP buffer is used. This ensures that all RMAP commands will be executed consecutively. Only available if the rmap VHDL generic is set to 1. Reset value: ‘0’.
16	RMAP Enable (RE) - Enable RMAP target. Only available if rmap VHDL generic is set to 1. Reset value: ‘1’.
15: 12	RESERVED
11	Time Rx Enable (TR) - Enable time-code receptions. Reset value: ‘0’.
10	Time Tx Enable (TT) - Enable time-code transmissions. Reset value: ‘0’.
9	Link error IRQ (LI) - Generate interrupt when a link error occurs. Not reset.
8	Tick-out IRQ (TQ) - Generate interrupt when a valid time-code is received. Not reset.
7	RESERVED
6	Reset (RS) - Make complete reset of the SpaceWire node. Self clearing. Reset value: ‘0’.
5	Promiscuous Mode (PM) - Enable Promiscuous mode. Reset value: ‘0’.
4	Tick In (TI) - The host can generate a tick by writing a one to this field. This will increment the timer counter and the new value is transmitted after the current character is transferred. A tick can also be generated by asserting the tick_in signal. Reset value: ‘0’.
3	Interrupt Enable (IE) - If set, an interrupt is generated when one of bit 8 to 10 is set and its corresponding event occurs. Reset value: ‘0’.
2	Autostart (AS) - Automatically start the link when a NULL has been received. Reset value: ‘0’ if the RMAP target is not available. If available the reset value is set to the value of the rmapen input signal.
1	Link Start (LS) - Start the link, i.e. allow a transition from ready to started state. Reset value: ‘0’.
0	Link Disable (LD) - Disable the SpaceWire codec. Reset value: ‘0’.

Table 755. GRSPW status register

31-24	23-21	20-10	9	8	7	6-5	4	3	2	1	0
RESERVED	LS	RESERVED	AP	EE	IA		PE	DE	ER	CE	TO

31: 24	RESERVED
23: 21	Link State (LS) - The current state of the start-up sequence. 0 = Error-reset, 1 = Error-wait, 2 = Ready, 3 = Started, 4 = Connecting, 5 = Run. Reset value: 0.
20: 10	RESERVED
9	Active port (AP) - Shows the currently active port. ‘0’ = Port 0 and ‘1’ = Port 1 where the port numbers refer to the index number of the data and strobe signals. Only available if the ports generic is set to 2.
8	Early EOP/EEP (EE) - Set to one when a packet is received with an EOP after the first byte for a non-rmap packet and after the second byte for a RMAP packet. Cleared when written with a one. Reset value: ‘0’.
7	Invalid Address (IA) - Set to one when a packet is received with an invalid destination address field, i.e it does not match the nodeaddr register. Cleared when written with a one. Reset value: ‘0’.
6: 5	RESERVED
4	Parity Error (PE) - A parity error has occurred. Cleared when written with a one. Reset value: ‘0’.
3	Disconnect Error (DE) - A disconnection error has occurred. Cleared when written with a one. Reset value: ‘0’.
2	Escape Error (ER) - An escape error has occurred. Cleared when written with a one. Reset value: ‘0’.
1	Credit Error (CE) - A credit has occurred. Cleared when written with a one. Reset value: ‘0’.
0	Tick Out (TO) - A new time count value was received and is stored in the time counter field. Cleared when written with a one. Reset value: ‘0’.

Table 756. GRSPW default address register

31-16	15-8	7-0
RESERVED	DEFMASK	DEFADDR

31: 8	RESERVED
15: 8	Default mask (DEFMASK) - Default mask used for node identification on the SpaceWire network. This field is used for masking the address before comparison. Both the received address and the DEFADDR field are anded with the inverse of DEFMASK before the address check.
7: 0	Default address (DEFADDR) - Default address used for node identification on the SpaceWire network. Reset value: 254 (taken from the nodeaddr VHDL generic when /= 255, else from the rmapnodeaddr input signal)

Table 757. GRSPW clock divisor register

31-16	15-8	7-0
RESERVED	CLKDIVSTART	CLKDIVRUN

31: 16	RESERVED
15: 8	Clock divisor startup (CLKDIVSTART) - Clock divisor value used for the clock-divider during startup (link-interface is in other states than run). The actual divisor value is Clock Divisor register + 1. Reset value: clkdiv10 input signal.
7: 0	Clock divisor run (CLKDIVRUN) - Clock divisor value used for the clock-divider when the link-interface is in the run-state. The actual divisor value is Clock Divisor register + 1. Reset value: clkdiv10 input signal.

Table 758. GRSPW destination key

31 ... 8	7 ... 0
RESERVED	DESTKEY

31: 8	RESERVED
7: 0	Destination key (DESTKEY) - RMAP destination key. Only available if the rmap VHDL generic is set to 1. Reset value: 0. (taken from the deskey VHDL generic)

Table 759. GRSPW time register

31 ... 8	7 6	5 ... 0
RESERVED	TCTRL	TIMECNT

31: 8	RESERVED
7: 6	Time control flags (TCTRL) - The current value of the time control flags. Sent with time-code resulting from a tick-in. Received control flags are also stored in this register. Reset value: '0'.
5: 0	Time counter (TIMECNT) - The current value of the system time counter. It is incremented for each tick-in and the incremented value is transmitted. The register can also be written directly but the written value will not be transmitted. Received time-counter values are also stored in this register. Reset value: '0'.

Table 760. GRSPW DMA control register

31 ... 17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED	LE	SP	SA	EN	NS	RD	RX	AT	RA	TA	PR	PS	AI	RI	TI	RE	TE

31: 17	RESERVED
16	Link error disable (LE) - Disable transmitter when a link error occurs. No more packets will be transmitted until the transmitter is enabled again. Reset value: '0'.
15	Strip pid (SP) - Remove the pid byte (second byte) of each packet. The address byte (first byte) will also be removed when this bit is set independent of the SA bit. Reset value: '0'.
14	Strip addr (SA) - Remove the addr byte (first byte) of each packet. Reset value: '0'.
13	Enable addr (EN) - Enable separate node address for this channel. Reset value: '0'.
12	No spill (NS) - If cleared, packets will be discarded when a packet is arriving and there are no active descriptors. If set, the GRSPW will wait for a descriptor to be activated.
11	Rx descriptors available (RD) - Set to one, to indicate to the GRSPW that there are enabled descriptors in the descriptor table. Cleared by the GRSPW when it encounters a disabled descriptor: Reset value: '0'.
10	RX active (RX) - Is set to '1' if a reception to the DMA channel is currently active otherwise it is '0'. Only readable.
9	Abort TX (AT) - Set to one to abort the currently transmitting packet and disable transmissions. If no transmission is active the only effect is to disable transmissions. Self clearing. Reset value: '0'.
8	RX AHB error (RA) - An error response was detected on the AHB bus while this receive DMA channel was accessing the bus. Cleared when written with a one. Reset value: '0'.
7	TX AHB error (TA) - An error response was detected on the AHB bus while this transmit DMA channel was accessing the bus. Cleared when written with a one. Reset value: '0'.
6	Packet received (PR) - This bit is set each time a packet has been received. never cleared by the SW-node. Cleared when written with a one. Reset value: '0'.
5	Packet sent (PS) - This bit is set each time a packet has been sent. Never cleared by the SW-node. Cleared when written with a one. Reset value: '0'.
4	AHB error interrupt (AI) - If set, an interrupt will be generated each time an AHB error occurs when this DMA channel is accessing the bus. Not reset.
3	Receive interrupt (RI) - If set, an interrupt will be generated each time a packet has been received. This happens both if the packet is terminated by an EEP or EOP. Not reset.

Table 760. GRSPW DMA control register

2	Transmit interrupt (TI) - If set, an interrupt will be generated each time a packet is transmitted. The interrupt is generated regardless of whether the transmission was successful or not. Not reset.
1	Receiver enable (RE) - Set to one when packets are allowed to be received to this channel. Reset value: ‘0’.
0	Transmitter enable (TE) - Write a one to this bit each time new descriptors are activated in the table. Writing a one will cause the SW-node to read a new descriptor and try to transmit the packet it points to. This bit is automatically cleared when the SW-node encounters a descriptor which is disabled. Reset value: ‘0’.

Table 761. GRSPW RX maximum length register.

31 - 25	24 - 0
RESERVED	RXMAXLEN

31: 25	RESERVED
24: 0	RX maximum length (RXMAXLEN) - Receiver packet maximum length in bytes. Only bits 24 - 2 are writable. Bits 1 - 0 are always 0. Not reset.

Table 762. GRSPW transmitter descriptor table address register.

31 - 10	9 - 4	3 - 0
DESCBASEADDR	DESCSEL	RESERVED

31: 10	Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. Not reset.
9: 4	Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPW. For each new descriptor read, the selector will increase with 16 and eventually wrap to zero again. Reset value: 0.
3: 0	RESERVED

Table 763. GRSPW receiver descriptor table address register.

31 - 10	9 - 3	2 - 0
DESCBASEADDR	DESCSEL	RESERVED

31: 10	Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. Not reset.
9: 3	Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPW. For each new descriptor read, the selector will increase with 8 and eventually wrap to zero again. Reset value: 0.
2: 0	RESERVED

Table 764. GRSPW DMA channel address register

31 - 16	15 - 8	7 - 0
RESERVED	MASK	ADDR

31: 8	RESERVED

Table 764. GRSPW DMA channel address register

15: 8	Mask (MASK) - Mask used for node identification on the SpaceWire network. This field is used for masking the address before comparison. Both the received address and the ADDR field are anded with the inverse of MASK before the address check.
7: 0	Address (ADDR) - Address used for node identification on the SpaceWire network for the corresponding dma channel when the EN bit in the DMA control register is set. Reset value: 254.

56.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x29. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

56.11 Configuration options

Table 765 shows the configuration options of the core (VHDL generics).

*Table 765.*Configuration options

Generic	Function	Allowed range	Default
tech	Selects technology for transmitter DDR registers (if output_type=1) and enables a reset of additional registers for ASIC technologies.	0 - NTECH	inferred
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by GRSPW.	0 - NAHBIRQ-1	0
rmap	Include hardware RMAP target. RMAP CRC logic will also be added. If set to 2 the core will only implement the RMAP target, provide a limited APB interface, enable time code reception and its interrupt.	0 - 2	0
rmapcrc	Enable RMAP CRC logic.	0 - 1	0
fifosize1	Sets the number of entries in the 32-bit receiver and transmitter AHB fifos.	4 - 32	32
fifosize2	Sets the number of entries in the 9-bit receiver fifo (N-Char fifo).	16 - 64	64
rxclkbuftype	Select clock buffer type for receiver clock. 0 does not select a buffer, instead i connects the input directly to the output (synthesis tools may still infer a buffer). 1 selects hardwired clock while 2 selects routed clock.	0 - 2	0
rxunaligned	Receiver unaligned write support. If set, the receiver can write any number of bytes to any start address without writing any excessive bytes.	0 - 1	0
rmapbufs	Sets the number of buffers to hold RMAP replies.	2 - 8	4
ft	Enable fault-tolerance against SEU errors	0 - 2	0
scantest	Enables scantest support	0 - 1	0
techfifo	Enables technology specific RAM blocks selected with memtech. When disabled the memtech generic will have no effect.	0 - 1	1
ports	Sets the number of ports	1 - 2	1
dmachan	Sets the number of DMA channels	1 - 4	1
memtech	Selects technology for RAM blocks.	0 - NTECH	DEFMEMTECH
input_type	Select receiver type. 0=self clocking (xor), 1 = interface for aeroflex spacewire transceiver, 2 = single data rate sampling, 3 and 4 = double data rate sampling.	0 - 4	0
output_type	Select transmitter type. 0 = single data rate, 1 = double data rate, 2 = unused	0 - 2	0
rxtx_sameclk	Set to one if the same clock net is connected to both the receiver and transmitter (which means this feature is only applicable when the receiver uses sampling). This will remove some unnecessary synchronization registers.	0 - 1	0
netlist	Select presynthesized netlist instead of synthesizing from source. When enabled the specific netlist is selected with the tech generic.	0 - 1	0
nodeaddr	Sets the reset value for the core’s node address. Value 255 enables rmapnodeaddr input instead.	0 - 254 255	254
destkey	Sets the reset value for the core’s destination key.	0 - 255	0

56.12 Signal descriptions

Table 766 shows the interface signals of the core (VHDL ports).

Table 766. Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
RXCLK0	N/A	Input	Receiver clock for port 0.	-
RXCLK1	N/A	Input	Receiver clock for port 1. Unused if the VHDL ports generic is 2.	-
TXCLK	N/A	Input	Transmitter default run-state clock	-
TXCLKN	N/A	Input	Transmitter inverted default run-state clock. Only used in DDR transmitter mode for technol-ogies not supporting local generation of inverted clock.	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SWNI	D	Input	Data input	-
	DV	Input	Data valid	-
	S	Input	Strobe input	-
	DCONNECT	Input	Disconnect	
	TICKIN	Input	Time counter tick input. Increments internal time-counter and transmits the new value.	High
	TICKINRAW	Input	Raw tick input. Send time-code from timein input.	High
	TIMEIN	Input	The time value sent when tickinraw is asserted.	
	CLKDIV10	Input	Clock divisor value used during initialization and as reset value for the clock divisor register	-
	RMAPEN	Input	Reset value for the rmapen control register bit	-
	RMAPNODEADDR	Input	Reset value for nodeaddr register bits when nodeaddr VHDL generic /= 255	-
	DCRSTVAL	Input	Disconnect timeout reset value. Unused for GRSPW2.	-
	TIMERRSTVAL	Input	Timer reset value. Unused for GRSPW2.	-

Table 766. Signal descriptions

Signal name	Field	Type	Function	Active
SWNO	D	Output	Data output	-
	S	Output	Strobe output	-
	TICKOUT	Output	Time counter tick output. Asserted when a valid time-code has been received.	High
	TICKOUTRAW	Output	Tick out which is always set when a time-code is received.	High
	TIMEOUT	Output	Contains the received time-code when tickinraw is asserted.	
	TICKINDONE	Output	Asserted when a time-code has been accepted for transmission when tickinraw is asserted. Tickin-raw must be deasserted the same clock cycle as tickindone is asserted.	High
	RXDAV	Output	Asserted each cycle a character has been receved on the SpaceWire link.	High
	RXDATAOUT	Output	Contains the received character when rxdav is asserted.	
	LINKDIS	Output	Asserted when the link is disabled	High
	LOOPBACK	Output	Reflects the value of the loopback bit on GRSPW2 control register. Can be use to control on-chip loop-back for test purposes.	High
* see GRLIB IP Library User’s Manual				

56.13 Library dependencies

Table 767 shows libraries used when instantiating the core (VHDL libraries).

*Table 767.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPACEWIRE	Signals, component	Component and record declarations.

56.14 Instantiation

This example shows how the core can be instantiated.

Normally di, si, do and so should be connected to input and output pads configured with LVDS drivers. How this is done is technology dependent.

The core in the example is configured with non-ft memories of size 4, 64 and 8 entries for AHB FIFOs, N-Char FIFO and RMAP buffers respectively.

The memory technology is inferred which means that the synthesis tool will select the appropriate components.

The hardware RMAP target is enabled which also automatically enables rxunaligned and rmapcrc.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.spacewire.all;

entity spacewire_ex is
```

```
  port (
    clk   : in  std_ulogic;
    rstn          : in  std_ulogic;

    -- spacewire signals
    spw_rxdp      : in  std_ulogic;
    spw_rxdn      : in  std_ulogic;
    spw_rxsp      : in  std_ulogic;
    spw_rxsn      : in  std_ulogic;
    spw_txdp      : out std_ulogic;
    spw_txdn      : out std_ulogic;
    spw_txsp      : out std_ulogic;
    spw_txsn      : out std_ulogic;

    spw_rxtxclk   : in  std_ulogic;
    spw_rxclkn   : in  std_ulogic
    );
end;

architecture rtl of spacewire_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- Spacewire signals
  signal swni      : grspw_in_type;
  signal swno      : grspw_out_type;
  signal dtmp      : std_ulogic;
  signal stmp      : std_ulogic;
  signal rxclko    : std_ulogic;
begin

  -- AMBA Components are instantiated here

  spw_phy0 : grspw2_phy
    generic map(
      scantest   => 0,
      tech       => 0,
      input_type => 3)
    port map(
      rstn       => rstn,
      rxclki     => spw_rxtxclk,
      rxclkin    => spw_rxclkn,
      nrxclki    => spw_rxtxclk,
      di         => dtmp,
      si         => stmp,
      do         => swni.d(1 downto 0),
      dov        => swni.dv(1 downto 0),
      dconnect   => swni.dconnect(1 downto 0),
      rxclko     => rxclko);

  spw_rxclk  <= rxclko & rxclko;

  sw0 : grspw2
  generic map(
    tech         => 0,
    hindex       => 0,
    pindex       => 10,
    paddr        => 10,
    pirq         => 10,
    ports        => 1,
    dmachan      => 1,
    rmap         => 1,
    rmapcrc      => 1,
    fifosize1    => 32,
    fifosize2    => 32,
    rxunaligned  => 1,
    rmapbufs     => 4,
    output_type  => 1,
    input_type   => 3,
    rxtx_sameclk => 1)
  port map(rstn, clkm, rxclko, rxclko, spw_rxtxclk, spw_rxtxclk, ahbmi,
       ahbmo(0), apbi, apbo(10), swni, swno);

  swni.tickin <= ’0’; swni.rmapen <= ’1’;
  swni.clkdiv10 <= conv_std_logic_vector(SPW_TX_FREQ_KHZ/10000-1, 8);

  spw_rxd_pad : inpad_ds generic map (padtech, lvds, x25v)
    port map (spw_rxdp, spw_rxdn, dtmp);
  spw_rxs_pad : inpad_ds generic map (padtech, lvds, x25v)
    port map (spw_rxsp, spw_rxsn, stmp);
  spw_txd_pad : outpad_ds generic map (padtech, lvds, x25v)
    port map (spw_txdp, spw_txdn, swno.d(0), gnd(0));
  spw_txs_pad : outpad_ds generic map (padtech, lvds, x25v)
    port map (spw_txsp, spw_txsn, swno.s(0), gnd(0));
  ...
```

56.15 API

A simple Application Programming Interface (API) is provided together with the GRSPW. The API is located in $(GRLIB)/software/spw. The files are rmapapi.c, spwapi.c, rmapapi.h, spwapi.h. The spwapi.h file contains the declarations of the functions used for configuring the GRSPW and transferring data. The corresponding definitions are located in spwapi.c. The rmapapi is structured in the same manner and contains a function for building RMAP packets.

These functions could be used as a simple starting point for developing drivers for the GRSPW. The different functions are described in this section.

56.15.1 GRSPW Basic API

The basic GRSPW API is based on a struct spwvars which stores all the information for a single GRSPW core. The information includes its address on the AMBA bus as well as SpaceWire parameters such as node address and clock divisor. A pointer to this struct is used as a input parameter to all the functions. If several cores are used, a separate struct for each core is created and used when the specific core is accessed.

*Table 768.*The spwvars struct

Field	Description	Allowed range
regs	Pointer to the GRSPW	-
nospill	The nospill value used for the core.	0 - 1
rmap	Indicates whether the core is configured with RMAP. Set by spw_init.	0 - 1
rxunaligned	Indicates whether the core is configured with rxunaligned support. Set by spw_init.	0 - 1
rmapcrc	Indicates whether the core is configured with RMAPCRC support. Set by spw_init.	0 - 1
clkdiv	The clock divisor value used for the core.	0 - 255
nodeaddr	The node address value used for the core.	0 - 255
destkey	The destination key value used for the core.	0 - 255
rxmaxlen	The Receiver maximum length value used for the core.	0 - 33554431
rxpnt	Pointer to the next receiver descriptor.	0 - 127
rxchkpnt	Pointer to the next receiver descriptor that will be polled.	0 - 127
txpnt	Pointer to the next transmitter descriptor.	0 - 63
txchkpnt	Pointer to the next transmitter descriptor that will be polled.	0 - 63
timetxen	The timetxen value used for this core.	0 - 1
timerxen	The timerxen value used for this core.	0 - 1
txd	Pointer to the transmitter descriptor table.	-
rxd	Pointer to the receiver descriptor table	-

The following functions are available in the basic API:

```
int spw_setparam(int nodeaddr, int clkdiv, int destkey, int nospill, int timetxen, int
timerxen, int rxmaxlen, int spwadr, struct spwvars *spw);
```

Used for setting the different parameters in the spwvars struct. Should always be run first after creating a spwvars struct. This function only initializes the struct. Does not write anything to the SpaceWire core.

*Table 769.*Return values for spw_setparam

Value	Description
0	The function completed successfully
1	One or more of the parameters had an illegal value

*Table 770.*Parameters for spw_setparam

Parameter	Description	Allowed range
nodeaddr	Sets the node address value of the struct spw passed to the function.	0-255
clkdiv	Sets the clock divisor value of the struct spw passed to the function.	0-255
destkey	Sets the destination key of the struct spw passed to the function.	0-255
nospill	Sets the nospill value of the struct spw passed to the function.	0 - 1
timetxen	Sets the timetxen value of the struct spw passed to the function.	0 - 1
timerxen	Sets the timerxen value of the struct spw passed to the function.	0 - 1
rxmaxlen	Sets the receiver maximum length field of the struct spw passed to the function.	0 - 2^{25}-1
spwadr	Sets the address to the GRSPW core which will be associated with the struct passed to the function.	0 - 2^{32}-1

```
int spw_init(struct spwvars *spw);
```

Initializes the GRSPW core located at the address set in the struct spw. Sets the following registers: node address, destination key, clock divisor, receiver maximum length, transmitter descriptor table address, receiver descriptor table address, ctrl and dmactrl. All bits are set to the values found in the spwvars struct. If a register bit is not present in the struct it will be set to zero. The descriptor tables are allocated to an aligned area using malloc. The status register is cleared and lastly the link interface is enabled. The run state frequency will be set according to the value in clkdiv.

*Table 771.*Return values for spw_init

Value	Description
0	The function completed successfully
1	One or more of the parameters could not be set correctly or the link failed to initialize.

*Table 772.*Parameters for spw_init

Parameter	Description	Allowed range
spw	The spwvars struct associated with the GRSPW core that should be initialized.	-

```
int set_txdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the transmitter descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_tx and spw_checktx (Explained in the section for those functions).

*Table 773.*Return values for spw_txdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

*Table 774.*Parameters for spw_txdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	0 - 2^{32}-1
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int set_rxdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the Receiver descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_rx and spw_checkrx (Explained in the section for those functions).

*Table 775.*Return values for spw_rxdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

*Table 776.*Parameters for spw_rxdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	0 - 2^{32}-1
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_disable(struct spwvars *spw);
```

Disables the GRSPW core (the link disable bit is set to ‘1’).

*Table 777.*Parameters for spw_disable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_enable(struct spwvars *spw);
```

Enables the GRSPW core (the link disable bit is set to ‘0’).

*Table 778.*Parameters for spw_enable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_start(struct spwvars *spw);
```

Starts the GRSPW core (the link start bit is set to ‘1’).

*Table 779.*Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_stop(struct spwvars *spw);
```

Stops the GRSPW core (the link start bit is set to ‘0’).

*Table 780.*Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_setclockdiv(struct spwvars *spw);
```

Sets the clock divisor register with the clock divisor value stored in the spwvars struct.

*Table 781.*Return values for spw_setclockdiv

Value	Description
0	The function completed successfully
1	The new clock divisor value is illegal.

*Table 782.*Parameters for spw_setclockdiv

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_set_nodeadr(struct spwvars *spw);
```

Sets the node address register with the node address value stored in the spwvars struct.

*Table 783.*Return values for spw_set_nodeadr

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

*Table 784.*Parameters for spw_set_nodeadr

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_set_rxmaxlength(struct spwvars *spw);
```

Sets the Receiver maximum length register with the rxmaxlen value stored in the spwvars struct.

*Table 785.*Return values for spw_set_rxmaxlength

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

*Table 786.*Parameters for spw_set_rxmaxlength

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_tx(int crc, int skipcrcsize, int hsize, char *hbuf, int dsize, char *dbuf, struct
spwvars *spw);
```

Transmits a packet. Separate header and data buffers can be used. If CRC logic is available the GSPW inserts RMAP CRC values after the header and data fields if crc is set to one. This function only sets a descriptor and initiates the transmission. Spw_checktx must be used to check if the packet has been transmitted. A pointer into the descriptor table is stored in the spwvars struct to keep track of the next location to use. It is incremented each time the function returns 0.

*Table 787.*Return values for spw_tx

Value	Description
0	The function completed successfully
1	There are no free transmit descriptors currently available
2	There was illegal parameters passed to the function

*Table 788.*Parameters for spw_tx

Parameter	Description	Allowed range
crc	Set to one to append RMAP CRC after the header and data fields. Only available if hardware CRC is available in the core.	0 - 1
skipcrcsize	The number of bytes in the beginning of a packet that should not be included in the CRC calculation	0 - 15
hsize	The size of the header in bytes	0 - 255
hbuf	Pointer to the header data	-
dsize	The size of the data field in bytes	0 - 2^{24}-1
dbuf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW core that should transmit the packet	-

```
int spw_rx(char *buf, struct spwvars *spw);
```

Enables a descriptor for reception. The packet will be stored to buf. Spw_checkrx must be used to check if a packet has been received. A pointer in the spwvars struct is used to keep track of the next location to use in the descriptor table. It is incremented each time the function returns 0.

*Table 789.*Return values for spw_rx

Value	Description
0	The function completed successfully
1	There are no free receive descriptors currently available

*Table 790.*Parameters for spw_rx

Parameter	Description	Allowed range
buf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW core that should receive the packet	-

```
int spw_checkrx(int *size, struct rxstatus *rxs, struct spwvars *spw);
```

Checks if a packet has been received. When a packet has been received the size in bytes will be stored in the size parameter and status is found in the rxs struct. A pointer in the spwvars struct is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

*Table 791.*Return values for spw_checkrx

Value	Description
0	No packet has been received
1	A packet has been received

*Table 792.*Parameters for spw_checkrx

Parameter	Description	Allowed range
size	When the function returns 1 this variable holds the number of bytes received	-
rxs	When the function returns 1 this variable holds status information	-
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

*Table 793.*The rxstatus struct

Field	Description	Allowed range
truncated	Packet was truncated	0 - 1
dcrcerr	Data CRC error bit was set. Only indicates an error if the packet received was an RMAP packet.	0 - 1
hcrcerr	Header CRC error bit was se.t. Only indicates an error if the packet received was an RMAP packet.	0 - 1
eep	Packet was terminated with EEP	0 - 1

```
int spw_checktx(struct spwvars *spw);
```

Checks if a packet has been transmitted. A pointer is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

*Table 794.*Return values for spw_checktx

Value	Description
0	No packet has been transmitted
1	A packet has been correctly transmitted
2	A packet has been incorrectly transmitted

*Table 795.*Parameters for spw_checktx

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
void send_time(struct spwvars *spw);
```

Sends a new time-code. Increments the time-counter in the GRSPW and transmits the value.

*Table 796.*Parameters for send time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
int check_time(struct spwvars *spw);
```

Check if a new time-code has been received.

*Table 797.*Return values for check_time

Value	Description
0	No time-code has been received
1	A new time-code has been received

*Table 798.*Parameters for check_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
int get_time(struct spwvars *spw);
```

Get the current time counter value.

*Table 799.*Return values for get_time

Value	Description
0 - 63	Returns the current time counter value

*Table 800.*Parameters for get_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
void spw_reset(struct spwvars *spw);
```

Resets the GRSPW.

*Table 801.*Parameters for spw_reset

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be reset	-

```
void spw_rmapen(struct spwvars *spw);
```

Enables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW.

*Table 802.*Parameters for spw_rmapen

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

```
void spw_rmapdis(struct spwvars *spw);
```

Disables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW

*Table 803.*Parameters for spw_rmapdis

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

```
int spw_setdestkey(struct spwvars *spw);
```

Set the destination key of the GRSPW. Has no effect if the RMAP command handler is not available. The value from the spwvars struct is used.

*Table 804.*Return values for spw_setdestkey

Value	Description
0	The function completed successfully
1	The destination key parameter in the spwvars struct contains an illegal value

*Table 805.*Parameters for spw_setdestkey

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set.	-

56.15.2 GRSPW RMAP API

The RMAP API contains only one function which is used for building RMAP headers.

```
int build_rmap_hdr(struct rmap_pkt *pkt, char *hdr, int *size);
```

Builds a RMAP header to the buffer pointed to by hdr. The header data is taken from the rmap_pkt struct.

*Table 806.*Return values for build_rmap_hdr

Value	Description
0	The function completed successfully
1	One or more of the parameters contained illegal values

*Table 807.*Parameters for build_rmap_hdr

Parameter	Description	Allowed range
pkt	Pointer to a rmap_pkt struct which contains the data from which the header should be built	
hdr	Pointer to the buffer where the header will be built	
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

*Table 808.*rmap_pkt struct fields

Field	**Description**	**Allowed Range**
type	Selects the type of packet to build.	writecmd, readcmd, rmwcmd, writerep, readrep, rmwrep
verify	Selects whether the data should be verified before writing	yes, no
ack	Selects whether an acknowledge should be sent	yes, no
incr	Selects whether the address should be incremented or not	yes, no
destaddr	Sets the destination address	0 - 255
destkey	Sets the destination key	0 - 255
srcaddr	Sets the source address	0 - 255
tid	Sets the transaction identifier field	0 - 65535
addr	Sets the address of the operation to be performed. The extended address field is currently always set to 0.	0 - 2^{32}-1
len	The number of bytes to be writte, read or read-modify-written	0 - 2^{24}-1
status	Sets the status field	0 - 11
dstspalen	Number of source path address bytes to insert before the destination address	0 - 228
dstspa	Pointer to memory holding the destination path address bytes	-
srcspalen	Number of source path address bytes to insert in a command. For a reply these bytes are placed before the return address	0 - 12
srcspa	Pointer to memory holding the source path address bytes	-

57 GRSYSMON - AMBA Wrapper for Xilinx System Monitor

57.1 Overview

The core provides an AMBA AHB interface to the Xilinx System Monitor present in Virtex-5 FPGAs. All Xilinx System Monitor registers are mapped into AMBA address space. The core also includes functionality for generating interrupts triggered by System Monitor outputs, and allows triggering of conversion start via a separate register interface.

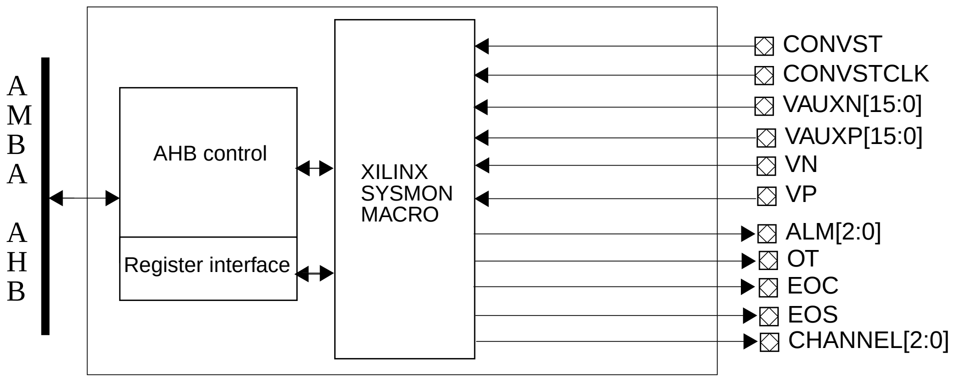


Figure 171. Block diagram

57.2 Operation

57.2.1 Operational model

The core has two I/O areas that can be accessed via the AMBA bus; the core configuration area and the System Monitor register area.

57.2.2 Configuration area

The configuration area, accessed via AHB I/O bank 0, contains two registers that provide status information and allow the user to generate interrupts from the Xilinx System Monitor's outputs. Write accesses to the configuration area have no AHB wait state and read accesses have one wait state. To ensure correct operation, only word (32-bit) sized accesses should be made to the configuration area.

57.2.3 System Monitor register area

The System Monitor register area is located in AHB I/O bank 1 and provides a direct-mapping to the System Monitor's Dynamic Reconfiguration Port. The System Monitor's first register is located at address offset 0x00000000 in this area.

Since the System Monitor documentation defines its addresses using half-word addressing, and AMBA uses byte-addressing, the addresses in the System Monitor documentation should be multiplied to get the correct offset in AMBA memory space. If the Configuration register bit WAL is '0' the address in System Monitor documentation should be multiplied by two to get the address mapped by the AMBA wrapper. A System Monitor register with address n is at AMBA offset 2*n. If the Configuration register bit WAL is '1', all registers start at a word boundary and the address in the System Monitor documentation should be multiplied by four to get the address mapped in AMBA address space. In this case, a System Monitor register with address n is at AMBA offset 4*n.

The wrapper always makes a single register access as the result of an access to the System Monitor register area. The size of the AMBA access is not checked and to ensure correct operation the mapped area should only be accessed using half-word (16-bit) accesses.

If the core has been implemented with AMBA split support, it will issue a SPLIT response to all accesses made to the mapped System Monitor registers. If the core is implemented without AMBA SPLIT support, wait states will be inserted until the System Monitor signals completion of a register access.

For a description of the System Monitor’s capabilities and configuration, please refer to the Xilinx Virtex-5 FPGA System Monitor User Guide.

57.3 Registers

The core is programmed through registers mapped into AHB address space.

*Table 809.*GRSYSMON registers

AHB address offset	**Register**
0x00	Configuration register
0x04	Status register

Table 810. GRSYSMON Configuration register

31	31 13	12	11 9	8 7	6	5	4	3	2	1	0
WAL	RESERVED	OT_IEN	ALM_IEN	RESERVED	CON-VST	EOS_IEN	EOC_IEN	BUSY_IEN	JB_IEN	JL_IEN	JM_IEN

31	Word aligned registers (WAL) - If this bit is set to ‘1’ each System Monitor memory mapped register start at a word boundary.
30 :13	RESERVED
12	Over temperature Interrupt Enable (OT_IEN) - If this bit is set to ‘1’ the core will generate an interrupt when the corresponding bit in the Status register is set to ‘1’. This bit is automatically cleared after the interrupt has been generated.
11:9	Alarm Interrupt Enable (ALM_IEN) - If a bit in this field is set to ‘1’ the core will generate an interrupt when the corresponding bit in the Status register is set to ‘1’. This bit is automatically cleared after the interrupt has been generated.
8:7	RESERVED
6	Conversion Start (CONVST) - If the core has been configured, at implementation, to use the an internal source for the Xilinx System Monitor CONVST signal, this bit can be written to ‘1’ to generate a pulse on the System Monitor’s CONVST input. This bit is automatically cleared after one clock cycle.
5	End of Sequence Interrupt Enable (EOS_IEN) - If this bit is set to ‘1’ the core will generate an interrupt when the corresponding bit in the Status register is set to ‘1’. This bit is automatically cleared after the interrupt has been generated.
4	End of Conversion Interrupt Enable (EOC_IEN) - If this bit is set to ‘1’ the core will generate an interrupt when the corresponding bit in the Status register is set to ‘1’. This bit is automatically cleared after the interrupt has been generated.
3	Busy Interrupt Enable (BUSY_IEN) - If this bit is set to ‘1’ the core will generate an interrupt when the corresponding bit in the Status register is set to ‘1’. This bit is automatically cleared after the interrupt has been generated.
2	JTAG Busy Interrupt Enable (JB_IEN) - If this bit is set to ‘1’ the core will generate an interrupt when the corresponding bit in the Status register is set to ‘1’. This bit is automatically cleared after the interrupt has been generated.
1	JTAG Locked Interrupt Enable (JL_IEN) - If this bit is set to ‘1’ the core will generate an interrupt when the corresponding bit in the Status register is set to ‘1’. This bit is automatically cleared after the interrupt has been generated.
0	JTAG Modified Interrupt Enable (JM_IEN) - .If this bit is set to ‘1’ the core will generate an interrupt when the corresponding bit in the Status register is set to ‘1’. This bit is automatically cleared after the interrupt has been generated.

Reset value: 0x00000000

Table 811. GRSYSMON Status register

31	30 — 13	12	11 — 9	8 — 6	5	4	3	2	1	0
WAL	RESERVED	OT	ALM	RESERVED	EOS	EOC	BUSY	JB	JL	JM

31	Word aligned registers (WAL) - If this bit is set to ‘1’ each System Monitor memory mapped register start at a word boundary.
30 :13	RESERVED
12	Over Temperature (OT) - Connected to the System Monitor’s Temperature Alarm output.
11:9	Alarm (ALM) - Connected to the System Monitor’s alarm outputs.
8:6	RESERVED
5	End of Sequence (EOS) - Connected to the System Monitor’s End of Sequence output.
4	End of Conversion (EOC) - Connected to the System Monitors End of Conversion output.
3	Busy (BUSY) - Connected to the System Monitor’s Busy output.
2	JTAG Busy (JB) - Connected to the System Monitor’s JTAG Busy output.
1	JTAG Locked (JL) - Connected to the System Monitor’s JTAG Locked output.
0	JTAG Modified (JM) - Connected to the System Monitor’s JTAG Modified output.

Reset value: See Xilinx System Monitor documentation

57.4 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x066. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

57.5 Implementation

57.5.1 Technology mapping

The core instantiates a SYSMON primitive.

57.5.2 RAM usage

The core does not use any RAM components.

57.6 Configuration options

Table 812 shows the configuration options of the core (VHDL generics).

*Table 812.*Configuration options

Generic name	Function	Allowed range	Default
tech	Target technology	0 - NTECH	0
hindex	AHB slave index	0 - (NAHBSLV-1)	0
hirq	Interrupt line	0 - (NAHBIRQ-1)	0
caddr	ADDR field of the AHB BAR0 defining configuration register address space.	0 - 16#FFF#	16#000#
cmask	MASK field of the AHB BAR0 defining configuration register address space.	0 - 16#FFF#	16#FFF#
saddr	ADDR field of the AHB BAR1 defining System Monitor register address space.	0 - 16#FFF#	16#001#
smask	MASK field of the AHB BAR1 defining System Monitor register space.	0 - 16#FFF#	16#FFF#

*Table 812.*Configuration options

Generic name	Function	Allowed range	Default
split	If this generic is set to 1 the core will issue AMBA SPLIT responses when it is busy performing an operation with the System Monitor. Otherwise the core will insert wait states until the operation completes.	0 - 1	0
extconvst	Connect CONVST input to System Monitor. If this generic is set to ‘0’ the System Monitor’s CONVST is controlled via the configuration register, otherwise the System Monitor CONVST input is taken from the core input signal.	0 - 1	0
wrdalign	Word align System Monitor registers. If this generic is set to 1 all System Monitor registers will begin on a word boundary. The first register will be mapped at offset 0x00, the second at 0x04. To translate a register access specified in the Xilinx System Monitor register documentation the register address should be multiplied by four to get the correct offset in AMBA address space. If this generic is set to 0, the register address should be multiplied by two to get the offset in AMBA address space.	0 - 1	0
INIT_40	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_41	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_42	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	16#0800#
INIT_43	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_44	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_45	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_46	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_47	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_48	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_49	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_4A	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_4B	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_4C	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_4D	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_4E	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_4F	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_50	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_51	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_52	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_53	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_54	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_55	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_56	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_57	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
SIM_MONITOR_ FILE	Simulation analog entry file. See Xilinx System Monitor documentation for a description of use and format.	-	“sysmon.txt”

57.7 Signal descriptions

Table 813 shows the interface signals of the core (VHDL ports).

*Table 813.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SYSMONI	CONVST	Input	Convert start input, connected to Xilinx System Monitor if the *extconvst* VHDL generic is set to ‘1’.	High
	CONVSTCLK	Input	Convert start input, connected to Xilinx System Monitor.	High
	VAUXN[15:0]	Input	Auxiliary analog input, connected to Xilinx System Monitor.	-
	VAUXP[15:0]	Input	Auxiliary analog input, connected to Xilinx System Monitor.	-
	VN	Input	Dedicated analog-input, connected to Xilinx System Monitor.	-
	VP	Input	Dedicated analog-input, connected to Xilinx System Monitor.	-
SYSMONO	ALM[2:0]	Output	Alarm outputs, connected to Xilinx System Monitor.	High
	OT	Output	Over-Temperature alarm output, connected to Xilinx System Monitor.	High
	EOC	Output	End of Conversion, connected to Xilinx System Monitor.	High
	EOS	Output	End of Sequence, connected to Xilinx System Monitor.	High
	CHANNEL[4:0]	Output	Channel selection, connected to Xilinx System Monitor.	-

* see GRLIB IP Library User’s Manual

57.8 Library dependencies

Table 814 shows the libraries used when instantiating the core (VHDL libraries).

*Table 814.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	MISC	Component, signals	Component and signal definitions
GRLIB	AMBA	Signals	AMBA signal definitions

57.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;
```

```
library gaisler;
use gaisler.misc.all;

entity grsysmon_ex is
  port (
    clk       : in  std_ulogic;
    rstn      : in  std_ulogic
    );
end;

architecture rtl of grsysmon_ex is
  -- AMBA signals
  signal ahbsi  : ahb_slv_in_type;
  signal ahbso  : ahb_slv_out_vector := (others => ahbs_none);
  ...
  -- GRSYSMON signals
  signal sysmoni : grsysmon_in_type;
  signal sysmono : grsysmon_out_type;

begin

  -- AMBA Components are instantiated here
  ...

  -- GRSYSMON core is instantiated below
  sysm0 : grsysmon generic map (tech => virtex5, hindex => 4,
     hirq => 4, caddr => 16#002#, cmask => 16#fff#,
     saddr => 16#003#, smask => 16#fff#, split => 1, extconvst => 0)
    port map (rstn, clk, ahbsi, ahbso(4), sysmoni, sysmono);
  sysmoni <= grsysmon_in_gnd; -- Inputs are all driven to ‘0’

end;
```

58 GRUSBDC - USB Device controller

58.1 Overview

The Universal Serial Bus Device Controller provides a USB 2.0 function interface accessible from an AMBA-AHB bus interface. The core must be connected to the USB through an external PHY (shown in figure 235) compliant to either UTMI, UTMI+ or ULPI. Both full-speed and high-speed mode are supported.

Endpoints are controlled through a set of registers accessed through an AHB slave interface. Each of the up to 16 IN and 16 OUT endpoints can be individually configured to any of the four USB transfer types.

USB data cargo is moved to the core's internal buffers using a master or a slave data interface. The data slave interface allows access directly to the internal buffers using AHB transactions and therefore does not need external memory. This makes it suitable for slow and simple functions. The data master interface requires an additional AHB master interface through which data is transferred autonomously using descriptor based DMA. This is suitable for functions requiring large bandwidth.

These two interfaces are mutually exclusive and cannot be present in the same implementation of the core.

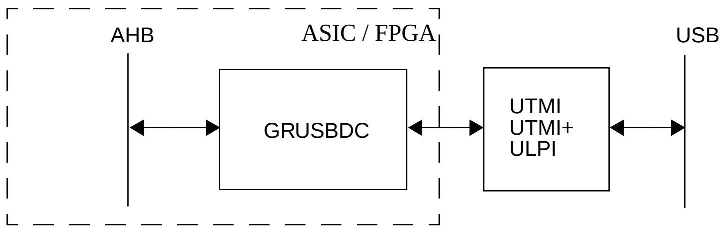


Figure 172. GRUSBDC connected to an external PHY device.

58.2 Operation

58.2.1 System overview

Figure 173 shows the internal structure of the core. This section briefly describes the function of the different blocks.

The Speed Negotiation Engine (SNE) detects connection by monitoring the VBUS signal on the USB connector. When a steady 5 V voltage is detected the SNE waits for a reset and then starts the High-speed negotiation. When the Speed negotiation and reset procedure is finished the selected speed mode (full-speed or high-speed) is notified to the Serial Interface Engine (SIE) which now can start operation. The SNE also detects and handles suspend and resume operations.

The SIE is enabled when the SNE notifies that the reset procedure has finished. It then waits for packets to arrive and processes them according to the USB 2.0 specification. The data cargo is stored to an internal buffer belonging to the recipient endpoint.

The AHB Interface Engine AIE is responsible for transferring USB data cargo from the endpoint's internal buffers to the AHB bus using descriptor based DMA through an AHB master interface when configured in master mode or by direct accesses to the AHB slave interface when configured in slave mode.

For received data it is then up to the external (to the device controller) function to continue processing of the USB data cargo after it has been transferred on the AHB bus. The function is the application specific core which determines the functionality of the complete USB device. It sets up endpoints in the device controller and notifies their existence through the appropriate USB descriptors. When the function wants to transmit a packet it either uses the slave interface to write to the endpoint buffers or establishes a DMA transfer.

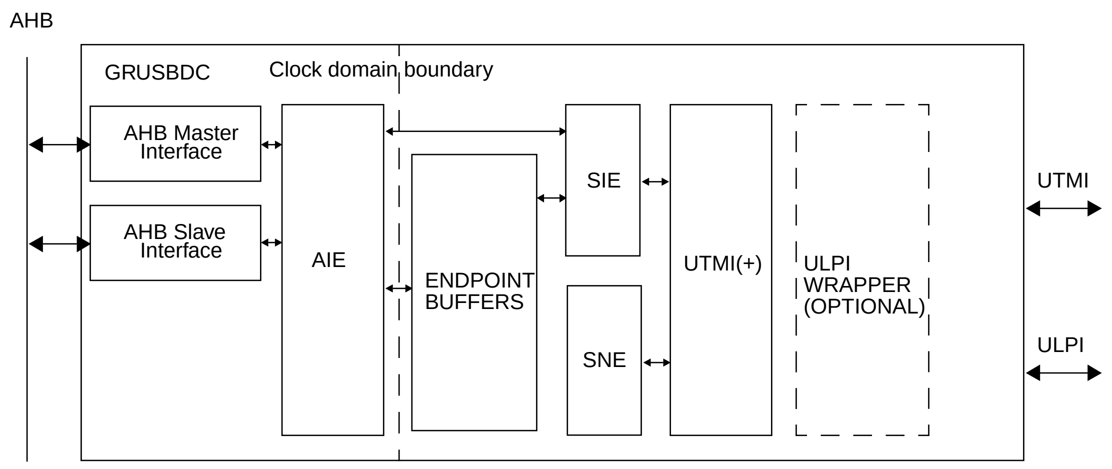


Figure 173. Block diagram of the internal structure of the core.

58.2.2 PHY interface

The core supports three different interfaces to the external PHY which is used to connect to the USB bus. The supported interfaces are UTMI, UTMI+ and ULPI. UTMI+ is an extension of the UTMI specification with optional additional support for host controllers and on-the-go devices while UTMI only supports devices. There are different so called levels in the UTMI+ specification, each with an added degree of support for hosts and on-the-go. The lowest level and common denominator for all the levels is identical to UTMI and uses the exact same signals. The core only supports devices and thus the support for UTMI+ refers to level 0. The data path to the UTMI/UTMI+ cores can be 8-bits or 16-bits wide and also uni- or bi-directional. All combinations are supported by the core.

The UTMI+ Low Pin Interface (ULPI) specifies a generic reduced pin interface and how it can be used to wrap a UTMI+ interface. The core has UTMI/UTMI+ as the main interface (they are identical) and when ULPI is used an extra conversion layer is added. When ULPI is enabled, the UTMI layer is always in 8-bit mode since this is what is required by the ULPI specification.

58.2.3 Speed Negotiation Engine (SNE)

The SNE detects attach, handles reset, high-speed handshake and suspend/resume operation. It also contains support for the various test-modes, which all USB device have to support.

The attached state is entered when a valid VBUS signal is detected. After this the core waits for a USB reset and then starts the high-speed handshake which determines whether full-speed or high-speed mode should be entered. No bus traffic will be accepted by the core until a valid reset has been detected.

The core supports soft connect/disconnect which means that the pull-up on the D+ line can be controlled from a user accessible register. The pull-up is disabled after reset and thus the function implementation has full control over when the device will be visible to the host.

The SNE also continuously monitors for the suspend condition (3 ms of idle on the USB bus) when the suspend state will be entered. The suspend state is left either through an USB reset or resume signaling. The resume signaling can come from either a downstream facing port (hub or host controller) or the device itself (Remote wakeup). The device controller core can generate remote wakeup signaling which is activated through an user accessible register. If this feature is used the function should indicate this in the descriptor returned to a device GetStatus request.

Transactions are only handled by the SIE if the SNE is in full-speed or high-speed mode. When in suspend, notattached, attached or during the reset process all transactions will be dropped.

The current status of the SNE such as VBUS valid, suspend active, USB reset received and current speed mode can be accessed through a status register. Each of these status bits have a corresponding interrupt enable bit which can be used to generate interrupts when a change occurs in the status bits.

If full-speed only mode is desired the core can be set to not perform the high-speed handshake through a register.

The different test-modes required by the USB standard are also enabled through user accessible registers. When enabled they can only be left by power-cycling the complete core or resetting the device controller (by using the rst signal to the core, not an USB reset).

The test modes are Test_SE0_NAK, Test_J, Test_K and Test_Packet.

In Test_SE0_NAK mode the high-speed receiver is enabled and only valid IN transactions (CRC correct, device and endpoint addresses match, PID is not corrupt) are responded to with a NAK.

Test_J continuously drives a high-speed J state.

Test_K continuously drives a high-speed K state.

Test_Packet repetitively sends a test packet. Please refer to the USB 2.0 standard for the packet contents. Minimum interpacket delay when device is sending two or more consecutive packets seems not to be specified in the standard. The core uses 192 bit times as its minimum delay which is the maximum value of the various minimum delays in the standard for any packet sequence and should therefore be compliant.

The core also supports a functional test-mode where all timeouts in the speed-negotiation engine have been shortened to eight clock cycles. This intended to be used in simulations and for ASIC testers where time and test-vector length respectively are important.

58.2.4 Serial Interface Engine (SIE)

The SIE handles transmission and reception of USB packets. The core will not respond to any transactions until a reset has been received and either full-speed or high-speed mode has been successfully entered.

The SIE always begins with waiting for a token packet. Depending on the type of token, data is either transferred from the core to the host or in the opposite direction. Special tokens are handled without any data transfers. The special tokens are PING and SOF which cause only a handshake to be sent or the frame number to be stored respectively. IN tokens initiate transfers to the host while SETUP and OUT tokens initiate transfers to the device. Packets received in the token stage with other PIDs than those mentioned in this paragraph are discarded.

A data packet is transmitted in the next stage if the token determined that data should be transferred to the host. When the data transmission is finished the core waits for a handshake before returning to the token stage.

If data was determined to be transferred to the device the core waits for and receives a data packet and then sends a handshake in return before entering the token stage again.

More detailed descriptions of the SIE and how it interacts with the core function are found in section 58.5.

58.2.5 Endpoint buffers

Each endpoint has two buffers to which packets are stored. The core automatically alternates between them when a packet has been received/transmitted so that data from one of the buffers can be transferred on the AHB bus while a new packet is being received/transmitted on the USB to/from the second buffer.

The state of the buffers affects the handshake sent to the host at the end of a transaction. In high-speed mode BULK OUT endpoints and CONTROL OUT endpoints which are not in the SETUP stage support the PING protocol. This means that at the end of an OUT transaction to one of these endpoints the device should return ACK if it could accept the current data and has space for another packet. For the USB device controller this is done if the second buffer for the endpoint is empty when the transaction is ready.

If the second buffer is non-empty a NYET is sent instead. If the current data could not be accepted (both buffers non-empty when the packet arrives) a NAK is returned.

For other endpoint types in high-speed mode and all endpoint types in full-speed mode an ACK is always returned if the data is accepted and a NAK if it could not be accepted.

An endpoint buffer can be configured to be larger than the maximum payload for that endpoint. For IN endpoints the writing of data larger than the maximum payload size to a buffer will result in a number of maximum sized packets being transferred ending with a packet smaller than or equal to the maximum size.

In the OUT direction larger buffers are only used for high-bandwidth endpoints where more than one transaction per microframe can occur for that endpoint. In that case the data from all packets during one microframe is stored in the order it arrives to a single buffer and is then handed over to the AHB interface. All non-high-bandwidth endpoints always store one packet data cargo to a buffer.

The endpoint buffers do not use separate physical RAM blocks in hardware instead they reside consecutively in the same memory space to avoid wasting memory.

58.2.6 AMBA Interface Engine (AIE)

The AIE can either be configured in slave mode or master mode. This is selected in synthesis process with a VHDL generic. Both cannot be present at the same time. The two interfaces will be described separately in this section.

Master interface

In master mode an AHB master interface is included in the core and handles all data transfers to and from the cores internal buffers using DMA operations. The DMA operation is described in detail in section 58.3. There is a separate DMA engine in the IN direction and the OUT direction respectively. They are multiplexed on the single master interface available for the core on the AHB bus. This scheme is used to limit the load on the AHB bus.

If both engines request the bus at the same time the owner will always be switched. That is, if the OUT direction DMA engine currently was allowed to make an access and when finished it still requests the bus for a new transaction and at the same time the IN direction engine also requests the bus, the IN direction will be granted access. If the situation is the same after the next access ownership will be switched back to the OUT engine etc.

The IN engine only reads data (note that this only applies to DATA, descriptor status is written) from the bus and always performs word transfers. Any byte alignment and length can still be used since this will only cause the core to skip the appropriate amount of leading and trailing bytes from the first and last words read.

The OUT engine writes data to the bus and performs both word and byte transfers. If the start transfer for an access is not word-aligned byte writes will be performed until a word boundary is reached. From then and onwards word writes are performed in burst mode until less than 4 bytes are left. If remaining number of bytes is not zero byte writes are performed for the last accesses. The byte accesses are always done as single accesses.

The bursts are of type incremental burst of unspecified length (refer to AMBA specification for more details). The core can only operate in big-endian mode that is the byte at the lowest address in a word is the most significant byte. This corresponds to bits 31 downto 24 in the GRLIB implementation. The first byte received on the USB will be stored to the msb location. In a single byte the lowest bit index corresponds to the first bit transmitted on the USB.

Slave interface

The slave interface is used for accessing registers and also for data transfers when the core is configured in slave mode. Byte, half-word and word accesses are supported. At least one waitstate is always inserted due to the pipelined nature of the interface but the upper limit is not fixed due to registers being accessed across clock domains. The maximum number of waitstates will thus depend on the difference in clock frequency between the USB and AHB clock domains. An upper bound can be calculated when the clock frequencies have been determined.

58.2.7 Synchronization

There are two clock domains in the core: the AHB clock domain and the USB clock domain. The AHB clock domain runs on the same clock as the AHB bus while the USB domain runs on the UTMI or ULPI clock. The boundary is between then AIE, SIE and Endpoint buffers. All signals between the two domains are synchronized and should be declared as false paths during synthesis.

58.2.8 Reset generation

The main reset (AMBA reset) resets AHB domain registers, synchronization registers between the USB and AHB clock domains, the USB PHY and USB SIE registers. Endpoint specific registers related to the state of the USB protocol in the SIE are reset when an UBS reset is received.

58.2.9 Synthesis

All number of endpoints with up to maximum size payloads cannot be supported due to limitations in the RAM block generator size in GRLIB. The maximum size also varies with technology. Note that large buffers can also have a large timing impact at least on FPGA since the a large RAM buffer will consist of several separate physical block RAMs located at different places causing large routing delays.

As mentioned in section 58.2.7, signals between the clock domains are synchronized and should be declared as false paths.

The complete AHB domain runs at the same frequency as the AHB bus and will be completely constrained by the bus frequency requirement.

The USB domain runs on different frequencies depending on the data path width. In 8-bit mode the frequency is 60 MHz and in 16-bit mode it is 30 MHz. Input and output constraints also need to be applied to the signals to and from the PHY. Please refer to the PHY documentation and/or UTMI/ULPI specification for the exact values of the I/O constraints.

58.2.10 Functional test-mode

A functional test-mode can be enabled in the core using the functesten VHDL generic. The functional test-mode is intended to reduce the number of required test-vectors during functional testing of an ASIC chip. During normal operation it would be required to go through the whole speed detection sequence before being able to start USB transactions. Since the speed detection takes a relatively long time this would make the test-vector amount very large often making it incompatible with existing test equipment.

In functional test-mode the core shortens the speed detection thus making it possible to test the functionality without a long initial delay. The test-mode can be disabled using the FT control register bit.

58.2.11 Scan test support

The VHDL generic *scantest* enables scan test support. If the core has been implemented with scan test support it will:

- disable the internal RAM blocks when the testen and scanen signals are asserted.
- use the testoen signal as output enable signal.
- use the testrst signal as the reset signal for those registers that are asynchronously reseted.

The testen, scanen, testrst, and testoen signals are routed via the AHB slave interface.

58.3 DMA operation

DMA operation is used when the core is configured in AHB master mode. Each IN and each OUT endpoint has a dedicated DMA channel which transfers data to and from the endpoint's internal buffers using descriptor based autonomous DMA. Each direction (IN and OUT) has its own DMA engine which requests the AHB master interface in contention with the other direction. Also each endpoint in a direction contends for the usage of the DMA engine with the other endpoints in the same direction. The arbitration is done in a round-robin fashion for all endpoints which are enabled and have data to send or receive.

The operation is nearly identical in both directions and the common properties will be explained here while the differences are outlined in the two following sub-sections.

The DMA operation is based on a linked list of descriptors located in memory. Each endpoint has its own linked list. The first word in a descriptor is the control word which contains an enable bit that determines whether the descriptor is active or not and other control bits. The following word is a pointer to a memory buffer where data should be written to or read from for this descriptor. The last word is a pointer to the location of the next descriptor. A bit in the control word determines if the next descriptor pointer is valid or not. If not valid the descriptor fetching stops after the current descriptor is processed and the DMA channel is disabled.

The DMA operation is started by first setting up a list with descriptors in memory and then writing a pointer to the first descriptor to the endpoint's descriptor pointer register in the core and setting the descriptor available bit. The pointer register is updated as the list is traversed and can be read through the AHB slave interface. When the list is ended with a descriptor that has its next descriptor available bit disabled the list must not be touched until the core has finished processing the list and the channel is disabled. Otherwise a deadlock situation might occur and behavior is undefined.

Another way to use the linked list is to always set the next descriptor available bit and instead make sure that the last descriptor is disabled. This way new descriptors can be added and enabled on the fly to the end of the list as long as the descriptor available bit is always set after the new descriptors have been written to memory. This ensures that no dead lock will occur and that no descriptors are missed. Figure 174 shows the structure of the descriptor linked list.

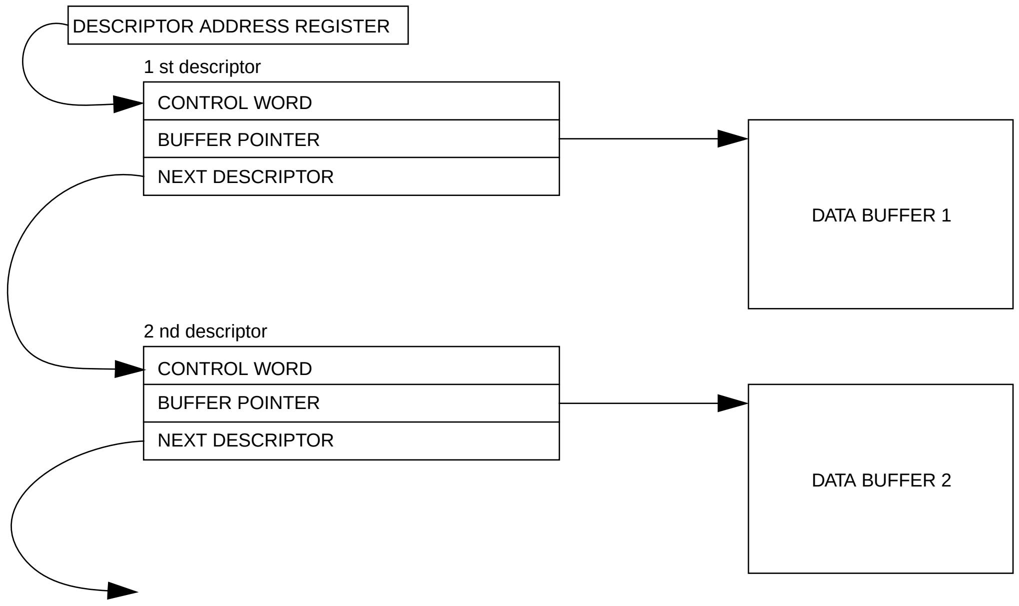


Figure 174. Example of the structure of a DMA descriptor linked list in memory.

58.3.1 OUT endpoints

The DMA operation for OUT endpoints conforms to the general description in the previous subsection. There are small differences in individual bits and the meaning of the length field. The contents of the different descriptor words can be found in the tables below.

When a descriptor has been enabled it will be fetched by the core when the descriptor available bit is set and as soon as a buffer for the corresponding endpoint contains data received from the USB it will be written to memory starting from the address specified in the buffer pointer word of the descriptor. The contents of a single internal memory buffer is always written to a single descriptor buffer. This always corresponds to a single USB packet except for high-bandwidth isochronous and interrupt endpoints. The number of bytes written is stored in the length field when writing is finished which is indicated by the enable bit being cleared. Then the SETUP status bit will also be valid. When the enable bit is cleared the memory location can be used again.

Interrupts are generated if requested as soon as the writing to memory is finished. The endpoint can also be configured to generate an interrupt immediately when a packet has been received to the internal buffers. This can not be enabled per packet since the core cannot associate a received packet with a specific descriptor in advance. This interrupt is enabled from the endpoint’s control register.

When the data has been fetched from the internal buffer it is cleared and can be used by the SIE again for receiving a new packet.

Table 815. OUT descriptor word 0 (address offset 0x0) ctrl word

31–18	17	16	15	14	13	12–0
RESERVED	SE	RE	IE	NX	EN	LENGTH

31: 18	RESERVED
17	Setup packet (SE) - The data was received from a SETUP packet instead of an OUT.
16	RESERVED

Table 815. OUT descriptor word 0 (address offset 0x0) ctrl word

15	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when the packet from this descriptor has been read to the internal buffers and handed over to the SIE. This does not mean that packet has also been transmitted.
14	Next descriptor available (NX) - The next descriptor field is valid and points to the next descriptor.
13	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
12: 0	LENGTH - The number of bytes received. Valid when the EN bit has been cleared by the core.

Table 816. OUT descriptor word 1 (address offset 0x4) Buffer pointer

31 … 0
ADDRESS

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

Table 817. OUT descriptor word 2 (address offset 0x8) Next descriptor pointer

31 … 2	1 0
NDP	RES

31: 2	Next descriptor pointer (NDP) - Pointer to the next descriptor.
1: 0	RESERVED

58.3.2 IN endpoints

The DMA operation for IN endpoints conforms to the general DMA description. There are small differences in individual bits and the meaning of the length field. The contents of the different descriptor words can be found in the tables below.

When a descriptor has been enabled and the descriptor available bit is set the core will start processing the descriptor and fetch the number of bytes indicated in the length field to an internal buffer belonging to the endpoint as soon as one is available. An interrupt will be generated if requested when data has been written to the internal buffer and status has been written back to the descriptor. The packet might not have been transmitted on the USB yet.

A separate interrupt is available which is generated when the packet has actually been transmitted. It needs to be enabled from the endpoint's control register and also in the descriptor (using the PI bit) for each packet that should generate the interrupt.

A descriptor with length zero will result in a packet with length zero being transmitted while a length larger than the maximum payload for the endpoint will result in two or more packets with all but the last being of maximum payload in length. The last transaction can be less than or equal to the maximum payload. If the length field is larger than the internal buffer size the data will not written to the internal buffer and status will be immediately written to the descriptor with an error bit set.

When the more bit is set the data from the current descriptor is written to the internal buffer and it then continues to the next descriptor without enabling the buffer for transmission. The next descriptor's data is also read to the same buffer and this continues until a descriptor is encountered which does not have more set.

If the total byte count becomes larger than the internal buffer size the packet is not sent (the data from the internal buffer is dropped) and the ML bit is set for the last descriptor. Then the descriptor fetching starts over again.

If the next bit is not set when the more bit is, the core will wait for a descriptor to be enabled without letting other endpoints access the AHB bus in between.

Table 818. IN descriptor word 0 (address offset 0x0) ctrl word

31 19	18	17	16	15	14	13	12 0
RESERVED	MO	PI	ML	IE	NX	EN	LENGTH

31: 19	RESERVED
18	More (MO) - The data from the next descriptor should be read to the same buffer.
17	Packet sent interrupt (PI) - Generate an interrupt when packet has been transmitted on the USB.
16	Maximum length violation (ML) - Attempted to transmit a data cargo amount larger than the buffer.
15	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when the packet from this descriptor has been read to the internal buffers and handed over to the SIE. This does not mean that packet has also been transmitted.
14	Next descriptor available (NX) - The next descriptor field is valid and points to the next descriptor.
13	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
12: 0	LENGTH - The number of bytes to be transmitted.

Table 819. IN descriptor word 1 (address offset 0x4) Buffer pointer

31 0
ADDRESS

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

Table 820. IN descriptor word 2 (address offset 0x8) Next descriptor pointer

31 2	1 0
NDP	RES

31: 2	Next descriptor pointer (NDP) - Pointer to the next descriptor.
1: 0	RESERVED

58.4 Slave data transfer interface operation

The AHB slave interface is used for data transfers instead of the DMA interface when the core is configured in AHB slave mode. This mode is selected by setting the aiface VHDL generic to 0. In this mode the core's internal buffers containing data to/from USB packets are accessed directly using AHB read and write transfers. This is usually much slower than DMA but much simpler and does not require any external memory, thus suitable for slow devices which need to be small and simple.

As for the DMA mode each endpoint is operated separately using four registers at the same addresses. Two of them, the control and status registers, are the same while the DMA control and the descriptor address registers have been replaced with the slave control and slave write/read data registers. The slave interface does not use descriptors so these four registers provides the complete control of the endpoint.

The details for data transfers in the two different endpoint directions will be explained in separate sections.

58.4.1 OUT slave endpoint

As stated earlier, in slave mode the core's buffers are accessed directly from the AHB bus through the slave interface. For OUT endpoints it has to be checked that data is available in the selected buffer and then reserve it. This is done using the slave control register by writing a one to the CB bit. The CB bit is always automatically cleared when the write access is finished and then the BS, DA and BUFCNT read-only bits/fields contain valid information. The BS bit is set to 0 if buffer 0 is currently selected and to 1 if buffer 1 is selected. DA is set to 1 if data is available in the buffer and in that case BUFCNT contains the number of bytes.

If data is available (DA is 1) it can be read from the slave data read register. One byte at a time is read using byte accesses, two bytes using half-word accesses and four bytes using word accesses. No other widths are supported. In each case data is available from bit 31 and downwards regardless of the value of the two least significant address bits. This is summarized in table 821. The BUFCNT field is continuously updated when reading the buffer so that it can be monitored how many bytes are left.

*Table 821.*AHB slave interface data transfer sizes

Size (byte)	AHB transfer size (HSIZE)	Data alignment (HRDATA)
1	byte (000)	31:24
2	half-word (001)	31:16
4	word (010)	31:0

When all the data has been read, a new buffer can be acquired by writing a one to CB. In this case when a buffer is currently reserved and the DA bit is set it will be released when CB is written. If a new buffer was available it will be reserved and DA is set to 1 again. If no new buffer is available DA will be 0 and the process has to be repeated. The current buffer (if one is selected) will be released regardless of whether a new one is available or not. Also, if all data has not been read yet when a buffer change request is issued the rest of the data will be lost.

The core does not have to be polled to determine whether a packet is available. A packet received interrupt is available which can be enabled from the control register and when set an interrupt will be generated each time a packet is stored to the internal buffers. The status of the buffers can also be read through the endpoint’s status register without actually reserving the buffer.

One buffer consists of the data payload from one single packet except for high-bandwidth interrupt and isochronous endpoints for which up to three packet data payloads can reside in a single buffer.

A buffer does not have to be read consecutively. Buffers for several endpoints can be acquired simultaneously and read interleaved with each other.

58.4.2 IN slave endpoint

The slave operation of IN endpoints is mostly identical to that for OUT. For IN endpoints it has to be checked that a buffer is free and then reserve it before writing data to it. This is done using the slave control register by writing a one to the EB bit. The EB bit is always automatically cleared when the write access is finished and then the BS, BA and BUFCNT fields are updated. The BS bit is set to 0 if buffer 0 is currently selected and to 1 if buffer 1 is selected. BA is set to 1 if a buffer is available to write data to. BUFCNT contains the number of bytes currently written to the selected buffer. It is cleared to zero when a new buffer is acquired.

If a buffer is available (BA is 1) data can be written through the slave data write register. One byte at a time is written using byte accesses, two bytes using half-word accesses and four bytes using word accesses. No other widths are supported. In each case data should be placed from bit 31 and downwards regardless of the value of the two least significant address bits. This is summarized in table 822.

*Table 822.*AHB slave interface data transfer sizes

Size (byte)	AHB transfer size (HSIZE)	Data alignment (HWDATA)
1	byte (000)	31:24
2	half-word (001)	31:16
4	word (010)	31:0

When all the data has been written, the buffer is enabled for transmission by writing a one to EB. If a new buffer was available it will be reserved and BA is set to 1 again. If no new buffer is available BA will be 0 and the process has to be repeated. The current buffer (if one is reserved) will be enabled for transmission regardless of whether a new one is available or not.

The core does not have to be polled to determine whether a buffer is available. A packet transmitted interrupt is available which can be enabled from the endpoint control register. Then when enabling a packet for transmission the PI bit in the slave control register can be set which will be cause an interrupt to be generated when this packet has been transmitted and cleared from the internal buffers. The status of the buffers can also be read through the endpoint's status register without actually reserving the buffer.

Maximum payload size packets will be generated from the buffer until the last packet which will contain the remaining bytes.

A buffer does not have to be written consecutively. Buffers for several endpoints can be acquired simultaneously and written interleaved with each other. This will however cause additional waitstates to be inserted.

58.5 Endpoints

An endpoint needs to have both its AHB and USB function configured before usage. The AHB configuration comprises the DMA operation described in the previous section. The USB configuration comprises enabling the endpoint for USB transactions, setting up transfer type (control, bulk, isochronous, interrupt), payload size, high-bandwidth among others. The configuration options are accessed through a register available from the AHB slave interface. See section 58.8 for the complete set of options.

When setting the configuration options the endpoint valid bit should be set. This will enable transfers to this endpoint as soon as a USB reset has been received. If the endpoint is reconfigured the valid bit must first be set to zero before enabling it again. Otherwise the endpoint will not be correctly initialized. When the endpoint is enabled the toggle scheme will be reset and data buffers cleared and buffer selectors set to buffer zero. The maximum payload, number of additional transactions and transfer type fields may only be changed when endpoint valid is zero or when setting endpoint valid to one again after being disabled. Other bits in the endpoint control register can be changed at any time.

No configuration options should be changed when the endpoint is enabled except the halt, control halt and disable bits.

An endpoint can also be halted by setting the halt bit of the endpoint. This will cause all transactions to receive a STALL handshake. When clearing the halt condition the toggle scheme will also be reset as required by the USB standard.

When the endpoint is setup, data transfers from and to the endpoint can take place. There is no difference in how data is transferred on the AHB bus depending on the selected transfer type. This only affects the transfers on the USB. For control endpoints some extra handling is required by the core user during error conditions which will be explained in the control endpoint section.

Packets that are received to an endpoint (independent of endpoint type) with a larger payload than the configured maximum value for the endpoint will receive a STALL handshake and cause the endpoint to enter halt mode.

When a USB reset is detected the CS, ED and EH bits of the endpoint control register will be cleared for all endpoints. The EV bit will also be cleared except for control endpoint 0.

The CB bit in the endpoint control register is for clearing the internal buffers of an endpoint. When set the data will be discarded from the buffers, the data available bits for the corresponding buffers will be cleared and the CB bit is cleared when it is done. This will however not work if a transaction is currently active to the same endpoint so this feature must be used with caution.

58.5.1 Control endpoints

Endpoint 0 must always be a control endpoint according to the USB standard and be accessible as soon as a USB reset is received. The core does not accept any transactions until a USB reset has been received so this endpoint can be enabled directly after power up. More control endpoints can be enabled as needed with the same constraints as the default control endpoint except that they should not be accessible until after configuration. If the function controlling the core is slow during startup it might not be able complete configuration of endpoint 0 before a USB reset is received. This problem is avoided in the core because its pull-up on D+ is disabled after reset which gives the function full control of when the device will be visible on the bus.

A control endpoint is a message pipe and therefore transfers data both in the IN and OUT direction. Thus control endpoints must use the endpoint in both directions with the same number in the device controller. This requires both to be configured in the same mode (same transfer type, payload ...). Otherwise device behavior is undefined.

A control transfer is always started with a SETUP transaction which will be received to the OUT endpoint. If the control transfer is a write the subsequent data phase will be in the OUT direction and this data will also be received to the OUT endpoint. The function should read both the setup data and the other data cargo and respond correspondingly. If the request was valid the function should enable a zero length packet for the endpoint in the IN direction which will lead to a valid status stage. If an error is detected it should instead halt the endpoint. There are two alternatives for this: A non-clearing halt which will last even after the next SETUP transaction or a clearing halt which will be removed when the next SETUP is received. The latter is the recommended behavior in the USB standard since the other will require the complete core to be reset to continue operation if the permanent halt appears on the default control endpoint.

The core can detect errors in single transactions which cause the endpoint to enter halt mode automatically. In this case the clearing halt feature will be used for control endpoints.

If a SETUP transaction indicates a control read the data phase will be in the IN direction. In that case the core user should enable data for the endpoint in the IN direction if the request was accepted otherwise the halt feature should be set. The transfer is finished when the host sends a zero length packet to the OUT endpoint.

Note that when entering halt for a control endpoint both the IN and OUT endpoints halt bits should be set.

Each time a control endpoint receives a setup token the buffers in the IN direction are emptied. This is done to prevent inconsistencies if the data and status stage were missing or corrupted and thus the data never fetched. The old data would still be in the buffer and the next setup transaction would receive erroneous data. The USB standard states that this can happen during error conditions and a new SETUP is transmitted before the previous transfer finished. The core user can also clear the buffer through the IN endpoint control register and is encouraged to do this when it detects a new SETUP before finishing the previous transfer. This must be done since the user might have enabled buffers after the core cleared them when receiving the new SETUP.

Whether data received to a descriptor for an OUT endpoint was from a SETUP transaction or an OUT transaction is indicated in a descriptor status bit.

58.5.2 Bulk endpoints

Bulk endpoints are stream pipes and therefore only use a single endpoint in either the IN or OUT direction. The endpoint with the same number in the other direction can be used independently. Data is accessed normally through the AHB interface and no special consideration need to be taken apart from the general endpoint guidelines.

58.5.3 Interrupt endpoints

Interrupt data is handled in the same manner as for bulk endpoints on the AHB interface. The differences only appear on the USB. These endpoints are also of stream type.

Interrupt endpoints support a high-bandwidth state which means that more than one transaction per microframe is performed. This necessitates buffers larger than the maximum payload size. The endpoint should be configured with a buffer larger or equal to the maximum payload times the number of transactions. All transactions will be received to/transmitted from the same buffer. The endpoint is configured as a high-bandwidth endpoint by setting the number of additional transactions to non-zero in the endpoint control register.

58.5.4 Isochronous endpoints

Isochronous endpoints are of stream type are identical to other endpoints regarding the handling on the AHB bus.

A big difference between isochronous endpoints and the other types is that they do not use handshakes. If no data is available when an IN token arrives to an Isochronous endpoint a data packet with length 0 is transmitted. This will indicate to the host that no error occurred but data was not ready. If no packet is sent the host will not know whether the packet was corrupted or not.

When not in high-bandwidth mode only one transaction in the OUT direction will be stored to a single buffer. In high-bandwidth mode all transactions during a microframe are stored to the same buffer.

In the IN direction data is always transferred from the same buffer until it is out of data. For high-bandwidth endpoints the buffer should be configured to be the maximum payload times the number of transactions in size.

Isochronous high-bandwidth endpoints use PID sequencing. When an error is detected in the PID sequence in the OUT direction no data is handed over to AHB domain for the complete microframe.

58.6 Device implementation example in master mode

This section will shortly describe how the USB device controller can be used in master mode.

A function controlling the device controller and implementing the actual application specific device will be needed. It can be either hardware, software or a combination. The only requirement is that it can control the device controller through the AHB bus.

The first thing needed for successful operation is a correctly configured PHY. This is automatically done by the device controller.

After this the device controller waits for attachment to the USB bus indicated by the VBUS becoming valid. This can be notified to the function either by polling or an interrupt. The time of attachment can be controlled by the function through the pull-up enable/disable bit in the core control register. When disabled the USB host will not notice the device even when it is plugged in.

After attachment a USB reset needs to be received before transactions are allowed to be accepted. This can also notified by polling or an interrupt.

Only control endpoint 0 should be accessible after reset. The function is responsible for enabling and configuring at the right time. It can wait until a USB reset has been received but it is easier to enable it immediately after power-up. This can be done since the device controller will not accept any transactions until USB reset has been received. When enabling the endpoint descriptors should also be enabled for both the IN and OUT direction and also the descriptor available bits should be set.

Then the endpoint is ready to accept packets and the function should wait for SETUP packets arriving. It can be notified of packets arriving either through polling or interrupts. The core should process the requests and return descriptors as requested. When a Set address request is received the function should write the new address to the device controller’s global control register. It will take effect after the next successful IN transaction for the control endpoint. This should correspond to the status stage of the Set address transfer.

When a Set configuration request is received the function should enable the appropriate interface and endpoints according to the selected configuration. This is done by writing to the various endpoint control registers. The function is responsible for advertising the configurations and interfaces through the descriptors requested by SETUP transactions.

When the endpoints for the selected configuration are enabled the function should also setup the DMA operation. Then it is ready to transmit and receive data through the application specific endpoints. Interrupts can be used to notify that new packets have transferred and then polling will determine which endpoint had a status change.

58.7 Device implementation example in slave mode

This section will shortly describe how the USB device controller can be used in slave mode.

A function controlling the device controller and implementing the actual application specific device will be needed. It can be either hardware, software or a combination. The only requirement is that it can control the device controller through the AHB bus.

The first thing needed for successful operation is a correctly configured PHY. This is automatically done by the device controller.

After this the device controller waits for attachment to the USB bus indicated by the VBUS becoming valid. This can be notified to the function either by polling or an interrupt. The time of attachment can be controlled by the function through the pull-up enable/disable bit in the core control register. When disabled the USB host will not notice the device even when it is plugged in.

After attachment an USB reset needs to be received before transactions are allowed to be accepted. This can also notified by polling or an interrupt.

Only control endpoint 0 should be accessible after reset. The function is responsible for enabling and configuring at the right time. It can wait until a USB reset has been received but it is easier to enable it immediately after power-up. This can be done since the device controller will not accept any transactions until USB reset has been received. When enabling the endpoint packet interrupts should be enabled or the function should start polling the buffer status so that it will notice when packets arrive.

Then the endpoint is ready to accept packets and the function should wait for SETUP packets arriving. When a packet arrives the core should process the requests and return USB descriptors as requested. When a Set address request is received the function should write the new address to the device controller’s global control register. It will take effect immediately. It should not be written until the status stage has been finished for the Set address request. It can be determined that the request has finished if a packet transmitted interrupt is enabled for the handshake packet of the request and an interrupt is received.

When a Set configuration request is received the function should enable the appropriate interface and endpoints according to the selected configuration. This is done by writing to the various endpoint control registers. The function is responsible for advertising the configurations and interfaces through the descriptors requested by SETUP transactions.

When the endpoints for the selected configuration are enabled the function should also enable interrupts or start polling these endpoints. Then it is ready to transmit and receive data through the application specific endpoints. Interrupts can be used to notify that new packets have been transferred and then status reads will determine which endpoint had a status change.

58.8 Registers

The core is programmed through registers mapped into AHB address space.

*Table 823.*GRUSBDC registers

AHB address offset	**Register**
0x00	OUT Endpoint 0 control register
0x04	OUT Endpoint 0 slave ctrl / DMA ctrl register
0x08	OUT Endpoint 0 slave data / DMA descriptor address register
0x0C	OUT Endpoint 0 status register
0x10-0x1C	OUT Endpoint 1
...	
0xF0-0xFC	OUT Endpoint 15
0x100-0x1FC	IN Endpoints 0-15
0x200	Global Ctrl register
0x204	Global Status register

Table 824. GRUSBDC OUT endpoint control register

31 21	20	19	18	17 7	6 5	4 3	2	1	0
BUFSZ	PI	CB	CS	MAXPL	NT	TT	EH	ED	EV

31: 21	Buffer size (BUFSZ) - Size/8 in bytes of one hardware buffer slot for this endpoint. Two slots are available for each endpoint.
20	Packet received interrupt (PI) - Generate an interrupt for each packet that is received on the USB for this endpoint (packet has been stored in the internal buffers). Reset value: ‘0’.
19	Clear buffers (CB) - Clears any buffers for the endpoint that contain data if the buffer is not currently active.
18	Control Stall (CS) - Return stall for data and status stages in a control transfer. Automatically cleared when the next setup token is received. Only used when the endpoint is configured as a control endpoint.
17: 7	Maximum payload (MAXPL) - Sets the maximum USB payload (maximum size of a single packet sent to/from the endpoint) size for the endpoint. All bits of the field are not always used. The maximum value for the maximum payload is determined with a generic for each endpoint. Not Reset.
6: 5	Number of transactions (NT) - Sets the number of additional transactions per microframe for high-speed endpoints and per frame for full-speed endpoints. Only valid for isochronous endpoints. Not Reset.
4: 3	Transfer type (TT) - Sets the transfer type for the endpoint. “00”=CTRL, “01” =ISOCH, “10”=BULK, “11”=INTERRUPT. Only OUT endpoints should be set to the CTRL type and then the IN endpoint with the same number will be automatically used. It is important not to use OUT endpoints that do not have a corresponding IN endpoint as a CTRL endpoint. Not Reset.
2	Endpoint halted (EH) - Halt the endpoint. If set, all transfers to this endpoint will receive a STALL handshake. Reset value: ‘0’.
1	Endpoint disabled (ED) - Disables the endpoint. If set, all transfers to this endpoint will receive a NAK handshake. Reset value: ‘0’.
0	Endpoint valid (EV) - Enables the endpoint. If not enabled, all transfers to this endpoint will be ignored and no handshake is sent. Reset value; ‘0’.

Table 825. GRUSBDC OUT slave control register.

31 17	16	15 3	2	1	0
RESERVED	SE	BUFCNT	DA	BS	CB

Table 825. GRUSBDC OUT slave control register.

31: 17	RESERVED
16	Setup packet (SE) - The data was received from a SETUP packet instead of an OUT.
15: 3	Buffer counter (BUFCNT) - The number bytes available(OUT)
2	Data available (DA) - Set to one if a valid packet was acquired when requested using the CB. If no valid packet was available it is set to zero. Reset value: ‘0’.
1	Buffer select (BS) - Current buffer selected. Read only.
0	Change or acquire buffer (CB) - If no buffer is currently active try to acquire a new one. If one is already acquired, free it and try to acquire a new one.

Table 826. GRUSBDC OUT slave buffer read register.

31 … 0
DATA

31: 0	Data (DATA) - In AHB slave mode, data is fetched directly from the internal buffer by reading from this register. Data always starts from bit 31. For word accesses bits 31-0 are valid, for half-word bits 31-16 and for byte accesses bits 31-24.

Table 827. GRUSBDC OUT DMA control register.

31 … 11	10	9 … 4	3	2	1	0
RESERVED	AE	RESERVED	AD	AI	IE	DA

31: 11	RESERVED
10	AHB error (AE) - An AHB error has occurred for this endpoint.
9: 4	RESERVED
3	Abort DMA (AD) - Disable descriptor processing (set DA to 0) and abort the current DMA transfer if one is active. Reset value: ‘0’.
2	AHB error interrupt (AI) - Generate interrupt when an AHB error occurs for this endpoint.
1	Interrupt enable (IE) - Enable DMA interrupts. Each time data has been received or transmitted to/ from a descriptor with its interrupt enable bit set an interrupt will be generated when this bit is set.
0	Descriptors available (DA) - Set to indicate to the GRUSBDC that one or more descriptors have been enabled.

Table 828. GRUSBDC OUT descriptor address register.

31 … 2	1 … 0
DESCADDR	RES

31: 2	Descriptor table address (DESCADDR) - Address to the next descriptor.Not Reset.
1: 0	RESERVED

Table 829. GRUSBDC OUT endpoint status register

31 30	29	28 … 16	15 … 3	2	1	0
RES	PR	B1CNT	B0CNT	B1	B0	BS

31: 30	RESERVED.
29	Packet received (PR) - Set each time a packet has been received (OUT) and stored in the internal buffers. Cleared when written with a ‘1’.
28: 16	Buffer 1 byte count (B1CNT) - Number of bytes in buffer one.
15: 3	Buffer 0 byte count (B0CNT) - Number of bytes in buffer zero.
2	Buffer 1 data valid (B1) - Set when buffer one contains valid data.

Table 829. GRUSBDC OUT endpoint status register

1	Buffer 0 data valid (B0) - Set when buffer zero contains valid data.
0	Buffer select (BS) - The currently selected buffer.

Table 830. GRUSBDC IN endpoint control register

31 21	20	19	18	17 7	6 5	4 3	2	1	0
BUFSZ	PI	CB	CS	MAXPL	NT	TT	EH	ED	EV

31: 21	Buffer size (BUFSZ) - Size/8 in bytes of one hardware buffer slot for this endpoint. Two slots are available for each endpoint.
20	Packet transmitted interrupt (PI) - Generate an interrupt each time a packet has been transmitted on the USB and the internal buffer is cleared. Reset value: ‘0’.
19	Clear buffers (CB) - Clears any buffers for the endpoint that contain data if the buffer is not currently active.
18	Control Stall (CS) - Return stall for data and status stages in a control transfer. Automatically cleared when the next setup token is received. Only used when the endpoint is configured as a control end-point.
17: 7	Maximum payload (MAXPL) - Sets the maximum USB payload (maximum size of a single packet sent to/from the endpoint) size for the endpoint. All bits of the field are not always used. The maxi-mum value for the maximum payload is determined with a generic for each endpoint. Not Reset.
6: 5	Number of transactions (NT) - Sets the number of additional transactions per microframe for high-speed endpoints and per frame for full-speed endpoints. Only valid for isochronous endpoints. Not Reset.
4: 3	Transfer type (TT) - Sets the transfer type for the endpoint. “00”=CTRL, “01” =ISOCH, “10”=BULK, “11”=INTERRUPT. Only OUT endpoints should be set to the CTRL type and then the IN endpoint with the same number will be automatically used. It is important not to use OUT end-points that do not have a corresponding IN endpoint as a CTRL endpoint. Not Reset.
2	Endpoint halted (EH) - Halt the endpoint. If set, all transfers to this endpoint will receive a STALL handshake. Reset value: ‘0’.
1	Endpoint disabled (ED) - Disables the endpoint. If set, all transfers to this endpoint will receive a NAK handshake. Reset value: ‘0’.
0	Endpoint valid (EV) - Enables the endpoint. If not enabled, all transfers to this endpoint will be ignored and no handshake is sent. Reset value; ‘0’.

Table 831. GRUSBDC IN slave control register.

31 17	16 4	3	2	1	0
RESERVED	BUFCNT	PI	BA	BS	EB

31: 17	RESERVED
16: 4	Buffer counter (BUFCNT) - The number of bytes written in the current buffer.
3	Packet interrupt enable (PI) - Generate interrupt when the activated packet has been transmitted. Should be set together with EB when enabling a packet for transmission. Reset value: ‘0’.
2	Buffer active (BA) - A free buffer was acquired and is available for use.
1	Buffer select (BS) - Current buffer selected. Read only.
0	Enable (EB) - Enable current buffer for transmission if one has been acquired and try to acquire a new buffer. If no data has been written to the buffer a zero length packet will be transmitted.

Table 832. GRUSBDC IN slave buffer read/write register.

31 0
DATA

Table 832. GRUSBDC IN slave buffer read/write register.

31: 0	Data (DATA) - Data written to this register is placed into the current buffer for transmission. Byte, Halfword and word sizes are allowed but only a word aligned address should be used. This means that data is always placed on 31-0 for word, 31-16 for half-word and 31-24 for byte.

Table 833. GRUSBDC IN DMA control register.

31 11	10	9 4	3	2	1	0
RESERVED	AE	RESERVED	AD	AI	IE	DA

31: 11	RESERVED
10	AHB error (AE) - An AHB has occurred for this endpoint.
9: 4	RESERVED
3	Abort DMA (AD) - Disable descriptor processing (set DA to 0) and abort the current DMA transfer if one is active. Reset value: ‘0’.
2	AHB error interrupt (AI) - Generate interrupt when an AHB error occurs for this endpoint.
1	Interrupt enable (IE) - Enable DMA interrupts. Each time data has been received or transmitted to/from a descriptor with its interrupt enable bit set an interrupt will be generated when this bit is set.
0	Descriptors available (DA) - Set to indicate to the GRUSBDC that one or more descriptors have been enabled.

Table 834. GRUSBDC IN descriptor address register.

31 2	1 0
DESCADDR	RES

31: 2	Descriptor table address (DESCADDR) - Address to the next descriptor.Not Reset.
1: 0	RESERVED

Table 835. GRUSBDC IN endpoint status register

31 30	29	28 16	15 3	2	1	0
RES	PT	B1CNT	B0CNT	B1	B0	BS

31: 30	RESERVED.
29	Packet transmitted (PT) - Packet has been transmitted and cleared from the internal buffers. Cleared when written with a ‘1’.
28: 16	Buffer 1 byte count (B1CNT) - Number of bytes in buffer one.
15: 3	Buffer 0 byte count (B0CNT) - Number of bytes in buffer zero.
2	Buffer 1 data valid (B1) - Set when buffer one contains valid data.
1	Buffer 0 data valid (B0) - Set when buffer zero contains valid data.
0	Buffer select (BS) - The currently selected buffer.

Table 836. GRUSBDC ctrl register

31	30	29	28	27	26 16	15	14	13	12	11 9	8	7 1	0
SI	UI	VI	SP	FI	RESERVED	FT	EP	DH	RW	TS	TM	UA	SU

31:	Suspend interrupt (SI) - Generate interrupt when suspend status changes. Reset value: ‘0’.
30	USB reset (UI) - Generate interrupt when USB reset is detected. Reset value: ‘0’.

Table 836. GRUSBDC ctrl register

29	VBUS valid interrupt (VI) - Generate interrupt when VBUS status changes. Reset value: ‘0’.
28	Speed mode interrupt (SP) - Generate interrupt when Speed mode changes. Reset value: ‘0’.
27	Frame number received interrupt (FI) - Generate interrupt when a new Start of frame (SOF) token is received. Reset value: ‘0’.
26: 16	RESERVED
15	Functional test mode (FT) - Enables functional test-mode which shortens all timer such as reset and chirp timers to 8 clock cycles.
14	Enable pull-up (EP) - Enable pull-up on the D+ line signaling a connect to the host. Reset value: ‘0’.
13	Disable High-speed (DH) - Disable high-speed handshake to make the core full-speed only.
12	Remote wakeup (RW) - Start remote wakeup signaling. It is self clearing and will be cleared when it has finished transmitting remote wakeup if it was currently in suspend mode. If not in suspend mode when set it will self clear immediately. Writes to this bit when it is already asserted are ignored. Reset value: ‘0’.
11: 9	Testmode selector (TS) - Select which testmode to enter. “001”= Test_J, “010”= Test_K, “011”= Test_SE0_NAK, “100”= Test_Packet.
8	Enable test mode (TM) - Set to one to enable test mode. Note that the testmode cannot be left without resetting or power-cycling the core and cannot be entered if hsdis is set to ‘1’. Reset value: ‘0’.
7: 1	USB address (UA) - The address assigned to the device on the USB bus.
0	Set USB address (SU) - Write with a one to set the usb address stored in the USB address field.

Table 837. GRUSBDC status register

31 28	27 24	23	22 18	17	16	15	14	13 11	10 0
NEPI	NEPO	DM	RESERVED	SU	UR	VB	SP	AF	FN

31: 28	Number of implemented IN endpoints (NEPI) - The number of configurable IN endpoints available in the core (including endpoint 0) minus one.
27: 24	Number of implemented OUT endpoints (NEPO) - The number of configurable OUT endpoints available in the core (including endpoint 0) minus one.
23	Data mode (DM) - 0 = core uses slave mode for data transfers, 1 = core uses master mode (DMA) for data transfers.
22: 18	RESERVED
17	Suspended (SU) - Set to ‘0’ when the device is suspended and ‘1’ when not suspended.
16	USB reset (UR) - Set each time an USB reset has been detected. Cleared when written with a ‘1’.
15	Vbus valid (VB) - Set to one when a valid voltage has been detected on the USB vbus line.
14	Speed (SP) - The current speed mode of the USB bus. ‘0’ = high-speed, ‘1’ = full-speed.
13: 11	Additional frames (AF) - Number of additional frames received with the current frame number.
10: 0	Frame number (FN) - The value of the last SOF token received.

58.9 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x021. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

58.10 Configuration options

Table 838 shows the configuration options of the core (VHDL generics). Buffer sizes are given in bytes and determines the sizes of one hardware buffer for the endpoint. Two buffers are available for each endpoint. The buffer size must be equal to or larger than the desired maximum payload size. In the case of high-bandwidth endpoints the buffer size must be equal to or larger than the maximum payload times the number of transactions per (micro) frame.

*Table 838.*Configuration options

Generic	**Function**	**Allowed range**	**Default**
hsindex	AHB slave index.	0 - NAHBSLV-1	0
hirq	AHB interrupt number	0 - NAHBIRQ-1	0
haddr	AHB slave address	-	0
hmask	AHB slave address mask	-	16#FFF#
hmindex	AHB master index	0 - NAHBMST-1	
aiface	0 selects the AHB slave interface for data transfer while 1 selects the AHB master interface.	0 - 1	0
memtech	Memory technology used for blockrams (endpoint buff-ers).	0 - NTECH	0
uiface	0 selects the UTMI interface while 1 selects ULPI.	0 - 1	0
dwidth	Selects the data path width for UTMI.	8 - 16	8
blen	Maximum number of beats in burst accesses on the AHB bus.	4 - 128	16
ninep	Number of IN endpoints	1 - 16	1
noutep	Number of OUT endpoints	1 - 16	1
i0	Buffer size for IN endpoint 0.	8, 16, 24, ... , 3072	1024
i1	Buffer size for IN endpoint 1.	8, 16, 24, ... , 3072	1024
i2	Buffer size for IN endpoint 2.	8, 16, 24, ... , 3072	1024
i3	Buffer size for IN endpoint 3.	8, 16, 24, ... , 3072	1024
i4	Buffer size for IN endpoint 4.	8, 16, 24, ... , 3072	1024
i5	Buffer size for IN endpoint 5.	8, 16, 24, ... , 3072	1024
i6	Buffer size for IN endpoint 6.	8, 16, 24, ... , 3072	1024
i7	Buffer size for IN endpoint 7.	8, 16, 24, ... , 3072	1024
i8	Buffer size for IN endpoint 8.	8, 16, 24, ... , 3072	1024
i9	Buffer size for IN endpoint 9.	8, 16, 24, ... , 3072	1024
i10	Buffer size for IN endpoint 10.	8, 16, 24, ... , 3072	1024
i11	Buffer size for IN endpoint 11.	8, 16, 24, ... , 3072	1024
i12	Buffer size for IN endpoint 12.	8, 16, 24, ... , 3072	1024
i13	Buffer size for IN endpoint 13.	8, 16, 24, ... , 3072	1024
i14	Buffer size for IN endpoint 14.	8, 16, 24, ... , 3072	1024
i15	Buffer size for IN endpoint 15.	8, 16, 24, ... , 3072	1024
o0	Buffer size for OUT endpoint 0.	8, 16, 24, ... , 3072	1024
o1	Buffer size for OUT endpoint 1.	8, 16, 24, ... , 3072	1024
o2	Buffer size for OUT endpoint 2.	8, 16, 24, ... , 3072	1024
o3	Buffer size for OUT endpoint 3.	8, 16, 24, ... , 3072	1024
o4	Buffer size for OUT endpoint 4.	8, 16, 24, ... , 3072	1024
o5	Buffer size for OUT endpoint 5.	8, 16, 24, ... , 3072	1024
o6	Buffer size for OUT endpoint 6.	8, 16, 24, ... , 3072	1024
o7	Buffer size for OUT endpoint 7.	8, 16, 24, ... , 3072	1024
o8	Buffer size for OUT endpoint 8.	8, 16, 24, ... , 3072	1024
o9	Buffer size for OUT endpoint 9.	8, 16, 24, ... , 3072	1024
o10	Buffer size for OUT endpoint 10.	8, 16, 24, ... , 3072	1024
o11	Buffer size for OUT endpoint 11.	8, 16, 24, ... , 3072	1024

*Table 838.*Configuration options

Generic	Function	Allowed range	Default
o12	Buffer size for OUT endpoint 12.	8, 16, 24, ... , 3072	1024
o13	Buffer size for OUT endpoint 13.	8, 16, 24, ... , 3072	1024
o14	Buffer size for OUT endpoint 14.	8, 16, 24, ... , 3072	1024
o15	Buffer size for OUT endpoint 15.	8, 16, 24, ... , 3072	1024
oepol	Select polarity of output enable signal. 1 selects active high and 0 selects active low.	0 - 1	0
syncprst	Use synchronously deasserted rst to PHY. This requires that the PHY generates a clock during reset.	0 - 1	0
prsttime	Reset time in microseconds needed for the PHY. Set to zero to disable this feature and use the reset time enforced by the reset to the GRUSBDC core.	>= 0	0
sysfreq	System frequency (HCLK input) in kHz. This is used together with prsttime to calculate how many clock cycles are needed for the PHY rst.	>= 0	50000
keepclk	This generic determines wheter or not the USB transceiver will be suspended and have its clock turned off during USB suspend. Set this generic to 1 if the clock should not be turned off. This might be needed for some technologies that can't handle that the USB clock is turned off for long periods of time.	0 - 1	0
sepirq*	Set this generic to 1 if three seperate interrupt lines should be used, one for status related interrupts, one for IN endpoint related interrupts, and one for OUT endpoint related interrupts. The irq number for the three different interrupts are set with the hirq (status), irqi (IN), and irqo (OUT) generics. If sepirq = 0 then only interrupts with irq number hirq will be generated.	0 - 1	0
irqi*	Sets the irq number for IN endpoint related interrupts. Only used if sepirq generic is set to 1.	0 - NAHBIRQ-1	1
irqo*	Sets the irq number for OUT endpoint related interrupts. Only used if sepirq generic is set to 1.	0 - NAHBIRQ-1	2
functesten	Enable functional test mode. This is used to skip the USB high-speed detection sequence to reduce the number of test vectors during functional testing.	0 - 1	0
scantest	Set this generic to 1 if scan test support should be implemented.	0 - 1	0

* The values of these generics are stored in the first User-Defined word of the core's AHB plug-n-play area as follows: bit 0 = sepirq, bits 7:4 = irqi, bits 11:8 = irqo. Please see the AHBCTRL section of GRLIB IP Core User's Manual.

58.11 Signal descriptions

Table 839 shows the interface signals of the core (VHDL ports).

*Table 839.*Signal descriptions

Signal name	Field	Type	Function	Active
UCLK	N/A	Input	USB UTMI/ULPI Clock	-
HCLK		Input	AMBA Clock	-
HRST		Input	AMBA Reset	Low
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
AHBSI	*	Input	AHB slave input signals	-

*Table 839.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
AHBSO	*	Output	AHB slave output signals	-
USBI	datain[15:0]	Input	UTMI/UTMI+/ULPI. Bits 15:8 are only used in 16-bit UTMI/UTMI+ mode.	-
	rxactive	Input	UTMI/UTMI+	High
	rxvalid	Input	UTMI/UTMI+	High
	rxvalidh	Input	UTMI/UTMI+ 16-bit	High
	rxerror	Input	UTMI/UTMI+	High
	txready	Input	UTMI/UTMI+	High
	linestate[1:0]	Input	UTMI/UTMI+	-
	nxt	Input	ULPI	High
	dir	Input	ULPI	High
	vbusvalid	Input	UTMI+	High
	urstdrive	Input	This input determines if the cores should drive the transceiver data lines low during USB transceiver reset, even if the dir input is High. This is needed for some transceivers, such as the NXP ISP1504. When this input is low the direction of the transceiver data lines are exclusively controlled by the dir signal from the transceiver. When this input is high the core will drive the data lines low during transceiver reset. Only applicable for ULPI transceivers.	High
USBO	dataout[15:0]	Output	UTMI/UTMI+/ULPI. Bits 15:8 are only used in 16-bit UTMI/UTMI+ mode.	-
	txvalid	Output	UTMI+	High
	txvalidh	Output	UTMI+ 16-bit	High
	opmode[1:0]	Output	UTMI+	-
	xcvrselect[1:0]	Output	UTMI/UTMI+. Bit 1 is constant low.	-
	termselect	Output	UTMI/UTMI+	-
	suspendm	Output	UTMI/UTMI+	Low
	reset	Output	Transceiver reset signal. Asserted asynchronously and deasserted synchrnously to the USB clock.	**
	stp	Output	ULPI	High
	oen	Output	Data bus direction control for ULPI and bi-directional UTMI/UTMI+ interfaces.	***
	databus16_8	Output	UTMI+. Constant high for 16-bit interface, constant low for 8-bit interface.	-
	dppulldown	Output	UTMI+. Constant low.	High
	dmpulldown	Output	UTMI+. Constant low.	High
	idpullup	Output	UTMI+. Constant low.	High
	drvvbus	Output	UTMI+. Constant low.	High
	dischrgvbus	Output	UTMI+. Constant low.	High
	chrgvbus	Output	UTMI+. Constant low.	High
	txbitstuffenable	Output	UTMI+. Constant low.	High
	txbitstuffenableh	Output	UTMI+. Constant low.	High
	fslsserialmode	Output	UTMI+. Constant low.	High
	tx_enable_n	Output	UTMI+. Constant high.	Low

*Table 839.*Signal descriptions

Signal name	Field	Type	Function	Active
	tx_dat	Output	UTMI+. Constant low.	High
	tx_se0	Output	UTMI+. Constant low.	High

* See GRLIB IP Library User's Manual.

** Depends on transceiver interface. Active high for UTMI/UTMI+ and active low for ULPI.

*** Implementation dependent.

58.12 Library dependencies

Table 840 shows libraries used when instantiating the core (VHDL libraries).

*Table 840.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	GRUSB	Signals, component	GRUSBDC component declarations, USB signals

58.13 Instantiation

This example shows how the core can be instantiated.

```
usbdc0: GRUSBDC
    generic map(
      hsindex       => 4,
      hirq          => 0,
      haddr         => 16#001#,
      hmask         => 16#FFF#,
      hmindex       => 14,
      aiface        => 1,
      memtech       => memtech,
      uiface        => 0,
      dwidth        => 8,
      nepi          => 16,
      nepo          => 16)
    port map(
      uclk          => uclk,
      usbi          => usbi,
      usbo          => usbo,
      hclk          => clkm,
      hrst          => rstn,
      ahbmi         => ahbmi,
      ahbmo         => ahbmo(14),
      ahbsi         => ahbsi,
      ahbso         => ahbso(4)
     );

usb_d_pads: for i in 0 to 15 generate
  usb_d_pad: iopad generic map(tech => padtech, slew => 1)
    port map (usb_d(i), usbo.dataout(i), usbo.oen, usbi.datain(i));
end generate;

usb_h_pad:iopad generic map(tech => padtech, slew => 1)
  port map (usb_validh, usbo.txvalidh, usbo.oen, usbi.rxvalidh);

usb_i0_pad : inpad generic map (tech => padtech) port map (usb_txready,usbi.txready);
usb_i1_pad : inpad generic map (tech => padtech) port map (usb_rxvalid,usbi.rxvalid);
usb_i2_pad : inpad generic map (tech => padtech) port map (usb_rxerror,usbi.rxerror);
usb_i3_pad : inpad generic map (tech => padtech) port map (usb_rxactive,usbi.rxactive);
usb_i4_pad : inpad generic map (tech => padtech) port map
(usb_linestate(0),usbi.linestate(0));
usb_i5_pad : inpad generic map (tech => padtech) port map
```

```
(usb_linestate(1),usbi.linestate(1));
usb_i6_pad : inpad generic map (tech => padtech) port map (usb_vbus, usbi.vbusvalid);

usb_o0_pad : outpad generic map (tech => padtech, slew => 1) port map (usb_reset,usbo.reset);
usb_o1_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_suspend,usbo.suspendm);
usb_o2_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_termsel,usbo.termselect);
usb_o3_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_xcvrsel,usbo.xcvrselect(0));
usb_o4_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_opmode(0),usbo.opmode(0));
usb_o5_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_opmode(1),usbo.opmode(1));
usb_o6_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_txvalid,usbo.txvalid);

usb_clk_pad : clkpad generic map (tech => padtech, arch => 2) port map (usb_clkout, uclk);

usbi.urstdrive <= ‘0’;
```

59 GRUSBHC - USB 2.0 Host Controller

59.1 Overview

The Aeroflex Gaisler USB 2.0 Host Controller provides a link between the AMBA AHB bus and the Universal Serial Bus. The host controller supports High-, Full-, and Low-Speed USB traffic. USB 2.0 High-Speed functionality is supplied by an enhanced host controller implementing the Enhanced Host Controller Interface revision 1.0. Full- and Low-Speed traffic is handled by up to 15 (USB 1.1) companion controllers implementing the Universal Host Controller Interface, revision 1.1. Each controller has its own AMBA AHB master interface. Configuration and control of the enhanced host controller is done via the AMBA APB bus. Companion controller registers are accessed via an AMBA AHB slave interface. Figure 175 shows a USB 2.0 host system and the organization of the controller types. Figure 176 shows an example with both host controller types present.

The controller supports both UTMI+ and ULPI transceivers and can handle up to 15 ports.

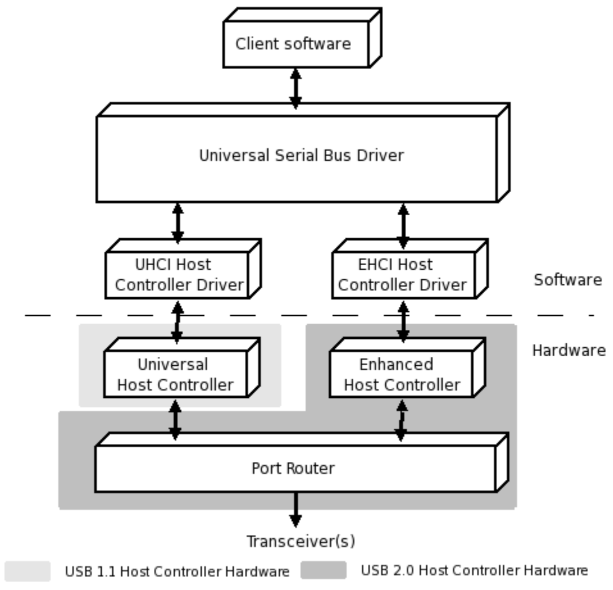

Client software
Universal Serial Bus Driver
UHCI Host Controller Driver
EHCI Host Controller Driver
Software
Hardware
Universal Host Controller
Enhanced Host Controller
Port Router
Transceiver(s)
USB 1.1 Host Controller Hardware
USB 2.0 Host Controller Hardware

Figure 175. Block diagram of USB 2.0 host system

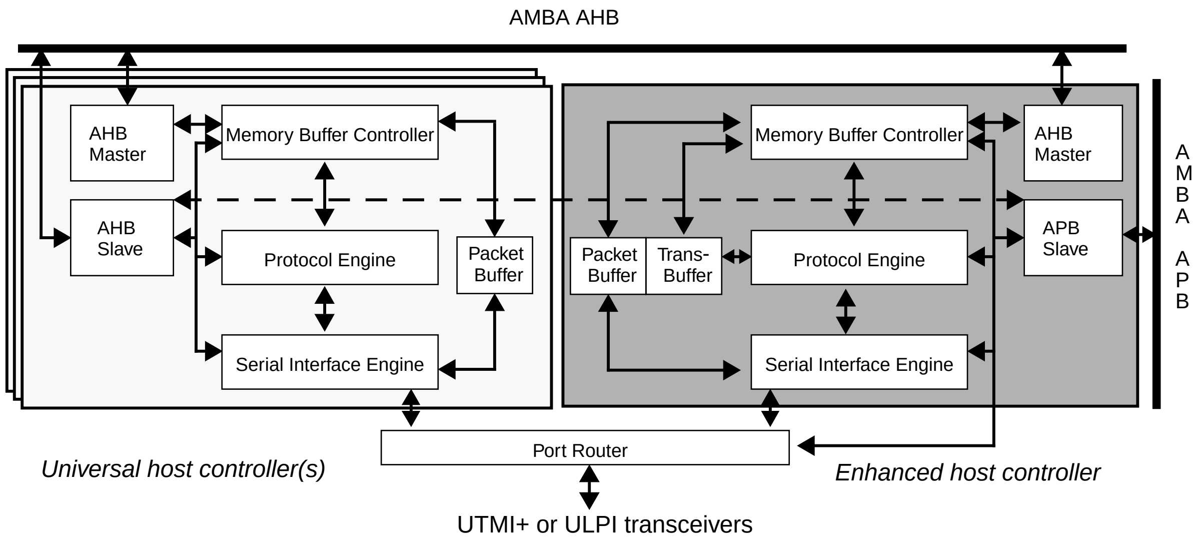


Figure 176. Block diagram of both host controller types

59.2 Operation

59.2.1 System overview

Depending on the core's configuration it may contain both controller types, one enhanced host controller, or up to 15 standalone universal host controllers. If both controller types are present, each universal host controller acts as a companion controller to the enhanced host controller.

The enhanced host controller complies with the Enhanced Host Controller Interface with the exception of the optional Light Host Controller Reset, which is not implemented.

The universal host controller complies with the Universal Host Controller Interface, with exceptions. The HCHalted field in the USB Command register is implemented as Read Only instead of Read/Write Clear. The Port Status/Control registers have been extended with Over Current and Over Current Change fields. Changes to both registers have been done in accordance with contemporary implementations of the interface. Both changes match the description of corresponding bits in the EHCI specification.

59.2.2 Protocol support

The enhanced host controller has full support for High-Speed traffic as defined in the USB Specification, revision 2.0. In addition Asynchronous Park Mode is supported, and the controller has a NAK counter.

The universal host controller supports Full- and Low-Speed traffic.

59.2.3 Descriptor and data buffering

The enhanced host controller prefetches one frame of isochronous descriptors. All payload data for a transaction is fetched before the transaction is executed. The enhanced host controller has a 2048 byte buffer for descriptors and a 2048 byte buffer for payload data, which can hold data for two transactions.

The universal host controller does not prefetch descriptors. Depending on controller configuration a transaction on the bus may be initiated before all payload data has been fetched from memory. Each universal host controller has a 1024 byte buffer for payload data. A transfer descriptor in UHCI may describe a transaction that has a payload of 1280 bytes. The USB specification limits the maximum allowed data payload to 1023 bytes and the controller will not transfer a larger payload than 1023 bytes. If a descriptor has a, legal, larger payload than 1023 bytes, the controller will only attempt to transfer the first 1023 bytes before the transaction is marked as completed.

In the event that the host controller has just one port, the universal host controller and the enhanced host controller will share the data payload buffer. Thus only two 2048 byte buffers are required.

59.2.4 Clocking and reset

The core has two clock domains; a system clock domain and a USB clock domain. The USB clock domain always operates in either 60 MHz or 30 MHz, depending on the transceiver interface. All signals that cross a clock domain boundary are synchronized to prevent meta-stability.

The reset input can be asserted and deasserted asynchronously or synchronously. All registers that in the system clock domain that require a reset value are synchronously reseted. It is assumed that the reset input is held asserted until the system clock input is stable. In order to insure that no unwanted USB activity take place, a few registers in the USB clock domain are asynchronously reseted. Once the USB clock starts to toggle, alla other registers in the USB domain that require a reset value will be reset as well.

From the reset input the core also generates reset to the USB transceivers. This generated reset is asserted asynchronously with respect to the USB clock, and it is deasserted synchronously or asynchronously depending on the value of the *syncprst* generic. Some transceivers require a synchronous deassert, while others turn off their clock output during reset, and therefore require an asynchronous deassert. The core can also time the reset to the transceiver(s). This is done with the *urst_time* and *sysfreq* generics.

Some ULPI transceivers require the data bus to be kept low by the core during transceiver reset, this behaviour is controlled by the *urstdrive* input signal.

59.2.5 Endianness

The core always accesses the least significant byte of a data payload at offset zero. Depending on the core's configuration, registers may be big endian, little endian, or byte swapped little endian.

59.2.6 RAM test facilities

The VHDL generic *ramtest* adds the possibility to test the RAM by mapping the core's internal buffers into the register space. If the core is implemented with RAM test facilities the universal host controller maps the packet buffer at offset 0x400 - 0x7FF. An enhanced host controller will map the packet buffer at offset 0x1000 - 0x17FF and the transaction buffer at 0x1800 - 0x1FFF. Note that the VHDL generics *uhchmask* and *ehcpmask* must be modified to allow access to the increased number of registers. The three least significant bits of the universal host controller's mask must be set to zero. The enhanced host controller's mask must have its five least significant bits set to zero.

When the *ramtest* generic is set to one an extra register called *RAM test control* register is added to both the universal and the enhanced controller. This register is described in section 59.9.1 and 59.9.2. To perform the RAM tests the user should first make sure that both the universal host controller and enhanced host controller are in their respective idle state. Note that if the core has only one port then the enhanced controller and the universal controller share the packet buffer. The shared buffer can be tested through both of the controllers but the controller performing the test must be the current owner of the port. For information on how to enter idle state and change ownership of the port please see section 59.9. When the controllers are in their idle states the enable bit in the *RAM test control* register should be set to one.

Once RAM test is enabled the whole RAM can be tested by first filling the buffers by writing to the corresponding register addresses and then setting the start bit in the *RAM test control* register. When the start bit is set the controller will, in order to access to the RAM from all its ports, read and write the whole packet buffer from the USB domain. If the core uses dual port RAM (see section 59.11 for more information on RAM usage) the same thing is done with the transaction buffer (if it is the enhanced controller that is performing the test). When the core uses double port RAM both the read and write port of the transaction buffer is located in the AHB domain, and therefore both the read and write port is tested during read and write accesses to register space. When the transfers are finished the core will clear the start bit and the data can then be read back through register space and be compared with the values that were written. When the core is implemented with dual port RAM individual addresses in the packet buffer and transaction buffer can be written and read without using the start bit. When using double port RAM only the transaction buffer can be read without using the start bit. However when individual addresses are accessed the buffers are only read/written from the AHB domain.

59.2.7 Scan test support

The VHDL generic *scantest* enables scan test support. If the core has been implemented with scan test support it will:

- disable the internal RAM blocks when the testen and scanen signals are asserted.
- use the testoen signal as output enable signal.
- clock all registers with the clk input (i.e. not use the USB clock).
- use the testrst signal as the reset signal for those registers that are asynchronously reseted.

The testen, scanen, testrst, and testoen signals are routed via the AHB master interface.

59.3 Port routing

Port routing is implemented according to the EHCI specification but functions regardless of whether the core is configured with or without an enhanced host controller. The VHDL generic prr enables or disables Port Routing Rules. With Port Routing Rules enabled, each port can be individually routed to a specific universal host controller via the VHDL generics portroute1 and portroute2. If Port Routing Rules is disabled the n_pcc lowest ports are routed to the first companion controller, the next n_pcc ports to the second companion controller, and so forth. The HCSP-PORTROUTE array is communicated via the portroute VHDL generics, which are calculated with the following algorithm:

portroute1 = $2^{26}*CC_8 + 2^{22}*CC_7 + 2^{18}*CC_6 + 2^{14}*CC_5 + 2^{10}*CC_4 + 2^6*CC_3 + 2^2*CC_2 + CC_1 / 4$

portroute1 = $2^{26}*CC_{15} + 2^{22}*CC_{14} + 2^{18}*CC_{13} + 2^{14}*CC_{12} + 2^{10}*CC_{11} + 2^6*CC_{10} + 2^2*CC_9 + CC_1 \bmod 4$

where CC_P is the companion controller that port P is routed to. Companion controllers are enumerated starting at 1.

When the enhanced host controller has not been configured by software, or when it is nonexistent, each port is routed to its companion controller. This allows a universal host controller to function even if the host system does not have support for the enhanced host controller. Please see the EHCI specification for a complete description of port routing.

59.4 DMA operations

Both host controller types have configurable DMA burst lengths. The burst length in words is defined by the VHDL generic bwrd. The value of bwrd limits how many words a controller may access in memory during a burst and not the number of memory operations performed after bus access has been granted. When writing a data payload back to memory that requires half-word or byte addressing the number of memory operations may exceed bwrd by one before the bus is released. If a host controller is given a byte-aligned data buffer its burst length may exceed the bwrd limit with one word when fetching payload data from memory.

The universal host controller uses a burst length of four words when fetching descriptors. This descriptor burst length is not affected by the bwrd VHDL generic. The universal host controller may be configured to start transactions on the USB before all data has been fetched from memory. The VHDL generic uhcblo specifies the number of words that must have been fetched from memory before a USB transaction is started. Since the USB traffic handled by the universal host controller can be expected to have significantly lower bandwidth than the system memory bus, this generic should be set to a low value.

59.5 Endianness

The core works internally with little endian. If the core is connected to a big endian bus, endian conversion must be enabled. When the VHDL generic endian_conv is set, all AMBA data lines are byte swapped. With endian_conv correctly set the core will start accessing data payloads from byte offset zero in the buffer, this is the first byte that is moved on the USB. The VHDL generic endian_conv must be set correctly for byte and halfword accesses to work. Therefore it is not possible to change the byte order of the buffer by configuring the controller for a little endian bus when it is connected to a big endian bus or vice versa.

The VHDL generics be_regs and be_desc are used to place the controller into big endian mode when endian conversion is enabled. These configuration options have no effect when the core is connected to a little endian bus, as determined by the value of VHDL generic endian_conv. The VHDL generic be_regs arranges the core’s registers in accordance with big endian addressing. In the enhanced host controller this will only affect the placement of the register fields CAPLENGTH and HCIVERSION, the HCSP-PORTROUTE array, and - if implemented - the PCI registers *Serial Bus Release Number Register* and *Frame Length Adjustment Register.* In the universal host controller be_regs will affect the placement of all registers. When be_regs is set, the bus to the register interface is never byte swapped. Tables 841 - 843 below illustrate the difference between big endian, little endian, and little endian layout with byte swapped (32 bit) WORDs on two 16 bit registers. Register R1 is located at address 0x00 and register R2 is located at address 0x02.

*Table 841.*R1 and R2 with big endian addressing

31 16	15 0
R1(15:0)	R2(15:0)

*Table 842.*R1 and R2 with little endian addressing

31 16	15 0
R2(15:0)	R1(15:0)

*Table 843.*R1 and R2 with little endian layout and byte swapped DWORD

31 24	23 16	15 8	7 0
R1(7:0)	R1(15:8)	R2(7:0)	R2(15:8)

The VHDL generic be_desc removes the byte swapping of descriptors on big endian systems. Tables 844 and 845 below list the effects of endian_conv and be_regs on a big endian and a little endian system respectively.

*Table 844.*Effect of endian_conv, be_regs, and be_desc on a big endian system

endian_conv	**be_regs**	**be_desc**	**System configuration**
0	-	-	Illegal. DMA will not function.
1	0	0	Host controller registers will be arranged according to little endian addressing and each DWORD will be byte swapped. In-memory transfer descriptors will also be byte swapped. This is the correct configuration for operating systems, such as Linux, that swap the bytes on big endian systems.
1	0	1	Host controller registers are arranged according to little endian addressing and will be byte swapped. Transfer descriptors will not be byte swapped.
1	1	0	Host controller registers will be arranged according to big endian addressing and will not be byte swapped. In memory transfer descriptors will be byte swapped.
1	1	1	Host controller registers will be arranged according to big endian addressing. In memory transfer descriptors will not be byte swapped.

*Table 845.*Effect of endian_conv, be_regs and be_desc on a little endian system

endian_conv	**be_regs**	**be_desc**	**System configuration**
0	-	-	Host controller registers will be placed as specified in the register interface specifications.
1	-	-	Illegal. DMA will not function.

59.6 Transceiver support

The controller supports UTMI+ 8-bit, UTMI+ 16-bit, and ULPI transceivers. All connected transceivers must be of the same type. Note that the transceiver type is fixed and the core can therefore not change between 8-bit and 16-bit UTMI+ interface during operation. Transceiver signals not belonging to the selected transceiver type are not connected and do not need to be driven. When using ULPI transceivers the default, and recommended, configuration is to use an external source for USB bus power (VBUS) as well as external VBUS fault detection. However the core can be configured to support configurations where the ULPI transceiver handles VBUS generation and fault detection internally, and configurations where VBUS generation is external to transceiver but fault detection is handled internally. Also the active level of the VBUS fault indicator can be configured. The configuration is handled by the vbusconf generic. If UTMI+ transceivers are used it does not matter to the core how VBUS generation and fault detection is handled as long as the VBUS enable signal and VBUS fault indicator are connected to the core's drvvbus and vbusvalid signals respectively. The UTMI+ specification defines these two signals to be active high, however in order to support different types of USB power switches and fault detectors the core can be configured to have active low drvvbus and vbusvalid signals. This configuration is also handled by the vbusconf generic. The UTMI+ interface is described in *USB 2.0 Transceiver Macrocell Interface (UTMI) Specification* and *UTMI+ Specification Revision 1.0*. The ULPI interface is described in *UTMI+ Low Pin Interface (ULPI) Specification Revision 1.1*.

59.7 PCI configuration registers and legacy support

The VHDL generic *pcidev* is used to configure the core to be used as a PCI device. If the core is configured to be used as a PCI device then the PCI registers *Serial Bus Release Number Register* and *Frame Length Adjustment Register* are implemented in the enhanced controller. The *Serial Bus Release Number Register* is also implemented for the universal controller. See *Enhanced Host Controller Interface Specification (EHCI) for Universal Serial Bus revision 1.0* and *Universal Host Controller Interface (UHCI) Design Guide revision 1.1* for details.

Legacy support is not implemented.

59.8 Software drivers

The core implements open interface standards and should function with available drivers. Aeroflex Gaisler supplies initialization code for both controllers for the Linux 2.6 kernel and VxWorks.

59.9 Registers

59.9.1 Enhanced host controller

The core is programmed through registers mapped into APB address space. The contents of each register is described in the *Enhanced Host Controller Interface Specification (EHCI) for Universal Serial Bus revision 1.0. A* register called *RAM test control* is added when the VHDL generic *ramtest* is set to one. The *RAM test control* register is not part of the EHCI interface and is described below. Also registers mapped to the packet buffer and transaction buffer are added if RAM test facilities are implemented.

*Table 846.*Enhanced Host Controller capability registers

APB address offset	**Register**
0x00	Capability Register Length
0x01	Reserved
0x02	Interface Version Number
0x04	Structural Parameters
0x08	Capability Parameters
0x0C	Companion Port Route Description

*Table 847.*Enhanced Host Controller operational registers

APB address offset	**Register**
0x14	USB Command*

*Table 847.*Enhanced Host Controller operational registers

APB address offset	**Register**
0x18	USB Status
0x1C	USB Interrupt Enable
0x20	USB Frame Index
0x24	4G Segment Selector (Reserved)
0x28	Frame List Base Address
0x2C	Next Asynchronous List Address
0x54	Configured Flag Register
0x58 - 0x90	Port Status/Control Registers**

*Light Host Controller reset is not implemented.

**One 32-bit register for each port

*Table 848.*Enhanced Host Controller RAM test registers

APB address offset	**Register**
0x100 - 0xFFF	RAM test control*
0x1000 - 0x17FF	Packet buffer**
0x1800 - 0x1FFF	Transaction buffer**

*Register is only present if *ramtest* generic is set to one. Accessible through any of the offsets specified.

**Registers are only present if *ramtest* generic is set to one.

Table 849. RAM test control

31 - 2	1	0
R	ST	EN

31: 2	R (Reserved): Always reads zero.
1	ST (Start): Starts the automatic RAM test. Can only be written to ‘1’. Cleared by the core when test is finished.
0	EN (Enable): Enable RAM test. Need to be set to ‘1’ in order to access the buffers.

*Table 850.*Enhanced Host Controller PCI registers

APB address offset	**Register**
0x2060	PCI registers *Serial Bus Release Number Register* and *Frame Length Adjustment Register**

*Only implemented if configured to be used as a PCI device.

59.9.2 Universal host controller

The core is programmed through registers mapped into AHB I/O address space. The contents of each register is described in the *Universal Host Controller Interface (UHCI) Design Guide revision 1.1. A* register called *RAM test control* is added when the VHDL generic *ramtest* is set to one. The *RAM test* *control* register is not part of the UHCI interface and is described below. Also registers mapped to the packet buffer are added when RAM test facilities are implemented.

*Table 851.*Universal Host Controller I/O registers

AHB address offset	Register
0x00	USB Command
0x02	USB Status*
0x04	USB Interrupt Enable
0x06	Frame Number
0x08	Frame List Base Address
0x0C	Start Of Frame Modify
0x10 - 0x2C	Port Status/Control**

*The HCHalted bit is implemented as Read Only and has the default value 1.

**Over Current and Over Current Change fields have been added. Each port has a 16-bit register.

Table 852. Changes to USB Status register

15 - 6	5	4 - 0
UHCI compliant	HCH	UHCI compliant

15: 6	UHCI compliant
5	Host Controller Halted (HCH) - Same behaviour as specified in the UHCI specification but the field has been changed from Read/Write Clear to Read Only and is cleared when Run/Stop is set. The default value of this bit has been changed to 1.
4:0	UHCI compliant

Table 853. Changes to Port Status/Control registers

15	11	10	9 - 0
UHCI compliant	OCC	OC	UHCI compliant

15: 12	UHCI compliant
11	Over Current Change (OCC) - Set to 1 when Over Current (OC) toggles. Read/Write Clear.
10	Over Current Active (OC) - Set to 1 when there is an over current condition. Read Only.
9:0	UHCI compliant

Table 854. Universal Host Controller RAM test registers

AHB address offset	Register
0x100 - 0x3FF	RAM test control*
0x400 - 0x7FF	Packet buffer**

*Register is only present if *ramtest* generic is set to one. Accessible through any of the offsets specified.

**Registers are only present if *ramtest* generic is set to one.

Table 855. RAM test control

31 - 2	1	0
R	ST	EN

31: 2	R (Reserved): Always reads zero.
1	ST (Start): Starts the automatic RAM test. Can only be written to ‘1’. Cleared by the core when test is finished.
0	EN (Enable): Enable RAM test. Need to be set to ‘1’ in order to access the buffers.

Table 856. Universal Host Controller PCI registers

AHB address offset	Register
0x60	PCI register *Serial Bus Release Number Register**

*Only implemented if configured to be used as a PCI device.

59.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler), the enhanced host controller has device identifier 0x026, the universal host controller has device identifier 0x027. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

59.11 RAM usage

The core maps all usage of RAM on either the *syncram_dp* component (dual port) or the *syncram_2p* component (double port), both from the technology mapping library (TECHMAP). Which component that is used can be configured with generics. The default, and recommended, configuration will use *syncram_dp*. A universal host controller requires one 256x32 *syncram_dp*, or two 256x32 *syncram_2p* for its packet buffer. The extra amount of *syncram_2p* needed comes from the fact that the packet buffer is both read and written from the AHB clock domain and the USB clock domain. It can not be guaranteed that a synchronization scheme would be fast enough and therefore one buffer for data beeing sent and one buffer for data beeing received are needed. An enhanced host controller requires one 512x32 *syncram_dp* (or two 512x32 *syncram_2p*, for the same reason discussed above) for its packet buffer and two 512x16 *syncram_dp/syncram_2p* for its transaction buffer. The transaction buffer is not doubled when using *syncram_2p*, instead synchronization and arbritration logic is added. When the core is instantiated with only one port, the enhanced host controller and universal host controller will share the packet buffer and the core only requires one 512x32 *syncram_dp* (or two 512x32 *syncram_2p*) for the packet buffer. Table 857 below shows RAM usage for all legal configurations.

*Table 857.*RAM usage for USB Host Controller core

Enhanced Host Controller present	Number of Universal Host Controllers	Number of ports	RAM component	RAM 256x32	RAM 512x32	RAM 512x16
No	x*	Don't care	syncram_dp	x*	0	0
No	x*	Don't care	syncram_2p	x**	0	0
Yes	1	1	syncram_dp	0	1	2
Yes	1	1	syncram_2p	0	2	2
Yes	x*	> 1	syncram_dp	x*	1	2
Yes	x*	> 1	syncram_2p	x**	2	2

* The number of required 256x32 *syncram_dp* equals the number of instantiated universal host controllers.

** The number of required 256x32 *syncram_2p* equals the double amount of instantiated universal host controllers.

59.12 Configuration options

Table 858 shows the configuration options of the core (VHDL generics).

*Table 858.*Configuration options

Generic name	Function	Allowed range	Default
ehchindex	Enhanced host controller AHB master index	0 - NAHBMST-1	0
ehcpindex	Enhanced host controller APB slave index	0 - NAPBSLV-1	0
ehcpaddr	Enhanced host controller ADDR field of the APB BAR.	0 - 16#FFF#	0

*Table 858.*Configuration options

Generic name	Function	Allowed range	Default
ehcpmask	Enhanced host controller MASK field of the APB BAR. Note that if the *ramtest* generic is set to 1 then the allowed range for this generic changes to 0 - 16#FE0#. If the pcidev generic is set to 1 then the allowed range for *ehcpmask* is 0 - 16#FC0#	0 - 16#FFF#	16#FFF#
ehcpirq	Enhanced host controller interrupt line	0 - NAHBIRQ-1	0
uhchindex	Universal host controller AHB master index. If the core contains more than one universal host controller the controllers will be assigned indexes from uhchindex to uhchindex+n_cc-1.	0 - NAHBMST-1	0
uhchsindex	Universal host controller AHB slave index. If the core contains more than one universal host controller the controllers will be assigned indexes from uhc_hsindex to uhchsindex+n_cc-1.	0 - NAHBSLV-n_cc	0
uhchaddr	Universal host controller ADDR field of the AHB BAR. If the core contains more than one universal host controller the controllers will be assigned the address space uhchaddr to uhchaddr + n_cc.	0 - 16#FFF#	0
uhchmask	Universal host controller MASK field of the AHB BAR. Note that if the *ramtest* generic is set to 1 then the allowed range for this generic changes to 0 - 16#FF8#	0 - 16#FFF#	16#FFF#
uhchirq	Universal host controller interrupt line. If the core contains more than one universal host controller the controller will be assigned interrupt lines uhc_hirq to uhchirq+n_cc-1.	0 - NAHBIRQ-1	0
tech	Technology for clock buffers	0 - NTECH	inferred
memtech	Memory Technology used for buffers.	0 - NTECH	inferred
nports	Number of USB ports	1 - 15	1
ehcgen	Enable enhanced host controller	0 - 1	1
uhcgen	Enable universal host controller(s)	0 - 1	1
n_cc	Number of universal host controllers. This value must be consistent with nports and n_pcc, or portroute1 and portroute2, depending on the value of the generic prr. This value must be at least 1, regardless the value of generic uhcgen.	1 - 15	1
n_pcc	Number of ports per universal host controller. This value must be consistent with n_cc and nports: nports <= (n_cc * n_pcc) < (nports + n_pcc) when Port Routing Rules is disabled. The only allowed deviation is if (nports mod n_cc) < n_pcc in which case the last universal host controller will get (nports mod n_cc) ports. This generic is not used then Port Routing Rules (prr) is enabled.	1 - 15	1
prr	Port Routing Rules. Determines if the core’s ports are routed to companion controller(s) with n_cc and n_pcc or with the help of portroute1 and portroute2.	0 - 1	0
portroute1	Defines part of the HCSP-PORTROUTE array	-	0
portroute2	Defines part of the HCSP-PORTROUTE array	-	0
endian_conv	Enable endian conversion. When set, all AMBA data lines are byte swapped. This generic must be set to 1 if the core is attached to a big endian bus, it must be set to 0 if the core is attached to a little endian bus.	0 - 1	1

*Table 858.*Configuration options

Generic name	**Function**	**Allowed range**	**Default**
be_regs	Arrange host controller registers according to big endian addressing. When set no endian conversion is made on the AMBA data lines connected to the host controller registers, regardless of endian_conv. Valid when endian_conv is enabled.	0 - 1	0
be_desc	Disable byte swapping of in-memory descriptors. Valid when endian_conv is enabled.	0 - 1	0
uhcblo	Universal Host Controller Buffer Limit Out. A universal host controller will start OUT bus transactions when uhcblo words of payload data has been fetched from memory. Note that if the core uses the UTMI+ 16 bit interface this generic must have a value larger than 2.	1 - 255	2
bwrd	Burst length in words. A universal host controller has a fixed, not affected by bwrd, burst length of four words when fetching transfer descriptors. See comments under section 59.2 DMA operations.	0 - 256	16
utm_type	Transceiver type: 0: UTMI+ 16 bit data bus 1: UTMI+ 8 bit data bus 2: ULPI	0 - 2	2
vbusconf*	Selects configuration for USB power source and fault detection (external and internal below is from the USB transceivers point of view): ULPI transceivers: 0: ULPI transceiver generates VBUS internally and no external fault indicator present 1: External power source but no external fault indicator. Transceiver implement the optional ULPI pin DrvVbu-sExternal but not ExternalVbusIndicator. 2: External power source and external active high fault indicator. Transceiver implement both the optional ULPI signals DrvvbusExternal and ExternalVbusIndicator. 3: External power source and external active low fault indicator. Transceiver implement both the optional sig-nals DrvvbusExternal and ExternalVbusIndicator. 4: External power source, but transceiver does not imple-ment the optional ULPI signal DrvVbusExternal. Active low drvvbus output from Host Controller will be used. Don’t care if ExternalVbusIndicator is implemented, not used. 5: External power source, but transceiver does not imple-ment the optional ULPI signal DrvVbusExternal. Active high drvvbus output from Host Controller will be used. Don’t care if ExternalVbusIndicator is implemented, not used. UTMI+ transceivers: 0: vbusvalid and drvvbus are both active low 1: vbusvalid is active low, drvvbus is active high 2: vbusvalid is active high, drvvbus is active low 3: vbusvalid and drvvbus are both active high	0 - 3	3
ramtest	When set each controller maps its internal buffers into the controller’s register space.**	0 - 1	0

*Table 858.*Configuration options

Generic name	Function	Allowed range	Default
urst_time	Amount of time (in ns) that the USB transceiver reset output need to be active. The actual length of the USB transceiver reset will be the length of the system reset plus the value of this generic. If set to zero, the host controller will not time the reset to the transceiver(s). Note that the sysfreq generic can not be set to zero either if reset timing is required.	-	0
oepol	The polarity of the output enable signal for the data input/output buffers, 0 means active low and 1 means active high.	0 - 1	0
scantest	Scan test support will be included if this generic is set to 1.	0 - 1	0
memsel	Selects if dual port or double port memories should be used for the host controllers' internal buffers. Dual port memories are used if this generic is set to 0 (or MEMSEL_DUALPORT). Double port memories are used if this generic is set to 1 (or MEMSEL_DOUBLEPORT). It is strongly recommended to use dual port memories in the host controllers. Double port memories should only be used on technologies that lack support for dual port memories or where the area overhead for dual port memories preventively large.***	0 - 1	0
syncprst	Some USB transceivers require that their reset input is deasserted synchronously to the USB clock. Set this generic to 1 if that is the case. Note that if the transceiver generates the USB clock and it keeps the clock output off during reset, then this generic must be set to 0. Otherwise the core will never leave its reset state.	0 - 1	0
sysfreq	This generic should be set to the system frequency (AHB domain clock frequency), in kHz. This generic is used to time reset to the USB transceivers. If set to zero, the host controller will not time the reset to the transceiver(s). Note that the urst_time generic can not be set to zero either if reset timing is required.	-	65000
pcidev	This generic should be set to one if the core is to be used as a PCI device. If set to 1 the core will hold its interrupt signal(s) high until cleared by software. Also a few PCI registers will be added. See section 59.7 and 59.9 for details on the registers. Note that if this generic is set to 1 then the allowed range for *ehcpmask* changes to 0 - 16#FC0#.	0 - 1	0

*see section 59.6 Transceiver support for more information

**see section 59.2.6 RAM test facilities for more information

***see section 59.11 RAM usage for more information

59.13 Signal descriptions

Table 859 shows the interface signals of the core (VHDL ports).

*Table 859.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
UCLK	N/A	Input	USB clock	-

*Table 859.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RST	N/A	Input	Reset	Low
APBI	*	Input	APB slave input signals	-
EHC_APBO	*	Output	APB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBSI	*	Input	AHB slave input signals	-
EHC_AHBMO	*	Output	AHB master output signals.	-
UHC_AHBMO[]	*	Output	AHB master output vector.	
UHC_AHBSO[]	*	Output	AHB slave output vector.	-
O[]	xcvrselect[1:0]	Output	UTMI+	-
	termselect	Output	UTMI+	-
	suspendm	Output	UTMI+	Low
	opmode[1:0]	Output	UTMI+	-
	txvalid	Output	UTMI+	High
	dataout[15:0]	Output	UTMI+/ULPI. Bits 15:8 are only used in 16-bit UTMI+ mode.	-
	txvalidh	Output	UTMI+ 16-bit	High
	stp	Output	ULPI	High
	reset	Output	Transceiver reset signal. Asserted asynchro-nously and deasserted synchrnously to the USB clock.	**
	oen	Output	Data bus direction control for ULPI and bi-direc-tional UTMI+ interfaces.	***
	databus16_8	Output	UTMI+ Constant high for 16-bit interface, con-stant low for 8-bit interface.	-
	dppulldown	Output	UTMI+ Constant high.	High
	dmpulldown	Output	UTMI+ Constant high.	High
	idpullup	Output	UTMI+ Constant low.	High
	drvvbus	Output	UTMI+/ULPI	***
	dischrgvbus	Output	UTMI+ Constant low.	High
	chrgvbus	Output	UTMI+ Constant low.	High
	txbitstuffenable	Output	UTMI+ Constant low.	High
	txbitstuffenableh	Output	UTMI+ Constant low.	High
	fslsserialmode	Output	UTMI+ Constant low.	High
	tx_enable_n	Output	UTMI+ Constant high.	High
	tx_dat	Output	UTMI+ Constant low.	High
	tx_se0	Output	UTMI+ Constant low.	High
I[]	linestate[1:0]	Input	UTMI+	-
	txready	Input	UTMI+	High
	rxvalid	Input	UTMI+	High
	rxactive	Input	UTMI+	High
	rxerror	Input	UTMI+	High
	vbusvalid	Input	UTMI+	***
	datain[15:0]	Input	UTMI+/ULPI. Bits 15:8 are only used in 16-bit UTMI+ interface.	-
	rxvalidh	Input	UTMI+ 16-bit	High
	hostdisconnect	Input	UTMI+	High

*Table 859.*Signal descriptions

Signal name	Field	Type	Function	Active
	nxt	Input	ULPI	High
	dir	Input	ULPI	-
	urstdrive	Input	This input determines if the cores should drive the transceiver data lines low during USB transceiver reset, even if the dir input is High. This is needed for some transceivers, such as the NXP ISP1504. When this input is low the direction of the transceiver data lines are exclusively controlled by the dir signal from the transceiver. When this input is high the core will drive the data lines low during transceiver reset. Only applicable for ULPI transceivers.	High

* See GRLIB IP Library User's Manual.

** Depends on transceiver interface. Active high for UTMI+ and active low for ULPI.

*** Implementation dependent.

59.14 Library dependencies

Table 860 shows the libraries used when instantiating the core (VHDL libraries).

*Table 860.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	GRUSB	Signals, component	Component declaration, USB signals

59.15 ASIC implementation details

When synthesizing the core for ASIC it might be required to use DC Ultra to reach the desired performance of the AMBA interface. The longest path might be as long as 200 gates when using DC-Expert, while its only around 10 gates when using DC Ultra. The core has successfully been synthesized with a 125 MHz AMBA interface, using TSMC 65nm CLN65LP standard library.

59.16 Instantiation

This example shows how the core can be instantiated.

```
library ieee, grlib, gaisler;
use ieee.std_logic_1164.all;
use grlib.amba.all;
use gaisler.grusb.all;

-- USB Host controller with 2 ports. One enhanced
-- host controller and two universal host controllers. Note that not all generics are set
-- in this example, many are kept at their defaulr values.

entity usbhc_ex is
  generic (
    tech    => tech;
    memtech => memtech;
    padtech => padtech);
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- USBHC signals
    usbh_clkin     : in std_ulogic;
    usbh_d         : inout std_logic_vector(15 downto 0);
    usbh_reset     : out std_logic_vector(1 downto 0);
    usbh_nxt       : in std_logic_vector(1 downto 0);
    usbh_stp       : out std_logic_vector(1 downto 0);
    usbh_dir       : in std_logic_vector(1 downto 0)
    );
end;

architecture rtl of usbhc_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);

  -- USBHC signals
  signal usbhci : grusb_in_vector(1 downto 0);
  signal usbhco : grusb_out_vector(1 downto 0);
  signal uhclk : std_ulogic;

begin

  -- AMBA Components are instantiated here
  ...

  -- Instantiate pads, one iteration for each port
  multi_pads: for i in 0 to 1 generate
    usbh_d_pad: iopadv
      generic map(tech => padtech, width => 8)
      port map (usbh_d((i*8+7) downto (i*8)),
                usbhco(i).dataout(7 downto 0), usbhco(i).oen,
                usbhci(i).datain(7 downto 0));
    usbh_nxt_pad : inpad generic map (tech => padtech)
      port map (usbh_nxt(i),usbhci(i).nxt);
    usbh_dir_pad : inpad generic map (tech => padtech)
      port map (usbh_dir(i),usbhci(i).dir);
    usbh_reset_pad : outpad generic map (tech => padtech)
      port map (usbh_reset(i),usbhco(i).reset);
    usbh_stp_pad : outpad generic map (tech => padtech)
      port map (usbh_stp(i),usbhco(i).stp);

    -- No need to drive ULPI data bus during USB reset
    usbhci(i).urstdrive <= ‘0’;
  end generate multi_pads;

  usbh_clkin_pad : clkpad:
    generic map (tech => padtech)
    port map(usbh_clkin, uhclk);

  usbhostcontroller0: grusbhc
    generic map (
      ehchindex => 5,
      ehcpindex => 14,
      ehcpaddr => 14,
      ehcpirq => 9,
      ehcpmask => 16#fff#,
      uhchindex => 6,
      uhchsindex => 3,
      uhchaddr => 16#A00#,
      uhchmask => 16#fff#,
      uhchirq => 10,
      tech => tech,
      memtech => memtech,
      nports => 2,
      ehcgen => 1,
      uhcgen => 1,
      n_cc => 2,
      n_pcc => 1,
      endian_conv => 1,
      utm_type => 2,
      vbusconf => 3)
    port map (
      clk => clk,
      uclk => uhclk,
      rst => rstn,
      apbi => apbi,
      ehc_apbo => apbo(14),
      ahbmi => ahbmi,
      ahbsi => ahbsi,
      ehc_ahbmo => ahbmo(5),
      uhc_ahbmo => ahbmo(7 downto 6),
      uhc_ahbso => ahbso(4 downto 3),
      o => usbhco,
      i => usbhci);
end;
```

60 GRVERSION - Version and Revision information register

60.1 Overview

The GRVERSION provides a register containing a 16 bit version field and a 16 bit revision field. The values for the two fields are taken from two corresponding VHDL generics. The register is available via the AMBA APB bus.

60.2 Registers

The core is programmed through registers mapped into APB address space.

*Table 861.*GRVERSION registers

APB address offset	Register
16#000#	Configuration Register

60.2.1 Configuration Register (R)

*Table 862.*Configuration Register

31 16	15 0
VERSION	REVISION

31-16:	VERSION	Version number
15- 0:	REVISION	Revision number

60.3 Vendor and device identifiers

The module has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x03A. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

60.4 Configuration options

Table 863 shows the configuration options of the core (VHDL generics).

*Table 863.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
versionnr	Version number	0 - 2^16-1	0
revisionnr	Revision number	0 - 2^16-1	0

60.5 Signal descriptions

Table 864 shows the interface signals of the core (VHDL ports).

*Table 864.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-

* see GRLIB IP Library User’s Manual

60.6 Library dependencies

Table 865 shows the libraries used when instantiating the core (VHDL libraries).

*Table 865.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

60.7 Instantiation

This example shows how the core can be instantiated.

```
TBD
```

61 I2C2AHB - I^2C to AHB bridge

61.1 Overview

The I^2C slave to AHB bridge is an I^2C slave that provides a link between the I^2C bus and AMBA AHB. The core is compatible with the Philips I^2C standard and external pull-up resistors must be supplied for both bus lines.

On the I^2C bus the slave acts as an I^2C memory device where accesses to the slave are translated to AMBA accesses. The core can translate I^2C accesses to AMBA byte, halfword or word accesses. The core makes use of I^2C clock stretching but can also be configured to use a special mode without clock stretching in order to support systems where master or physical layer limitations prevent stretching of the I^2C clock period.

GRLIB also contains another I^2C slave core, without an AHB interface, where the transfer of each individual byte is controlled by software via an APB interface, see the I2CSLV core documentation for more information.

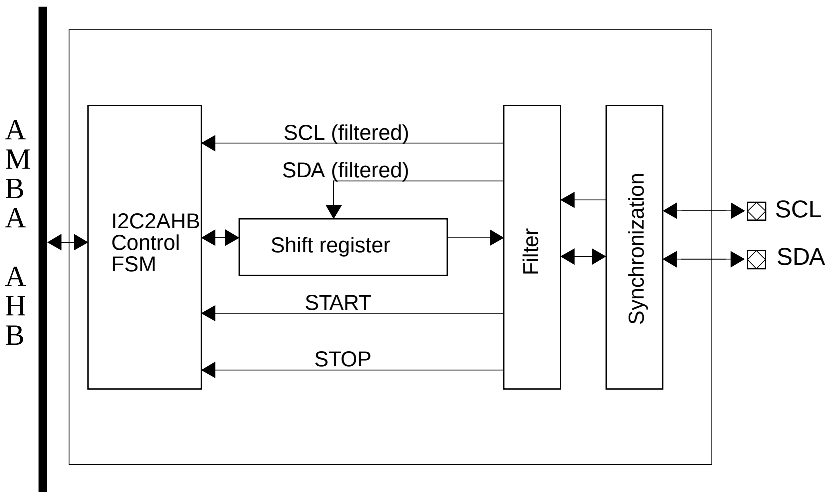


Figure 177. Block diagram, optional APB interface not shown

61.2 Operation

61.2.1 Transmission protocol

The I^2C-bus is a simple 2-wire serial multi-master bus with collision detection and arbitration. The bus consists of a serial data line (SDA) and a serial clock line (SCL). The I^2C standard defines three transmission speeds; Standard (100 kb/s), Fast (400 kb/s) and High speed (3.4 Mb/s).

A transfer on the I^2C-bus begins with a START condition. A START condition is defined as a high to low transition of the SDA line while SCL is high. Transfers end with a STOP condition, defined as a low to high transition of the SDA line while SCL is high. These conditions are always generated by a master. The bus is considered to be busy after the START condition and is free after a certain amount of time following a STOP condition. The bus free time required between a STOP and a START condition is defined in the I^2C-bus specification and is dependent on the bus bit rate.

Figure 178 shows a data transfer taking place over the I^2C-bus. The master first generates a START condition and then transmits the 7-bit slave address. The bit following the slave address is the $R/\overline{W}$ bit which determines the direction of the data transfer. In this case the $R/\overline{W}$ bit is zero indicating a write operation. After the master has transmitted the address and the $R/\overline{W}$ bit it releases the SDA line. The receiver pulls the SDA line low to acknowledge the transfer. If the receiver does not acknowledge the transfer, the master may generate a STOP condition to abort the transfer or start a new transfer by generating a repeated START condition.

After the address has been acknowledged the master transmits the data byte. If the R/$\overline{\text{W}}$ bit had been set to ‘1’ the master would have acted as a receiver during this phase of the transfer. After the data byte has been transferred the receiver acknowledges the byte and the master generates a STOP condition to complete the transfer.

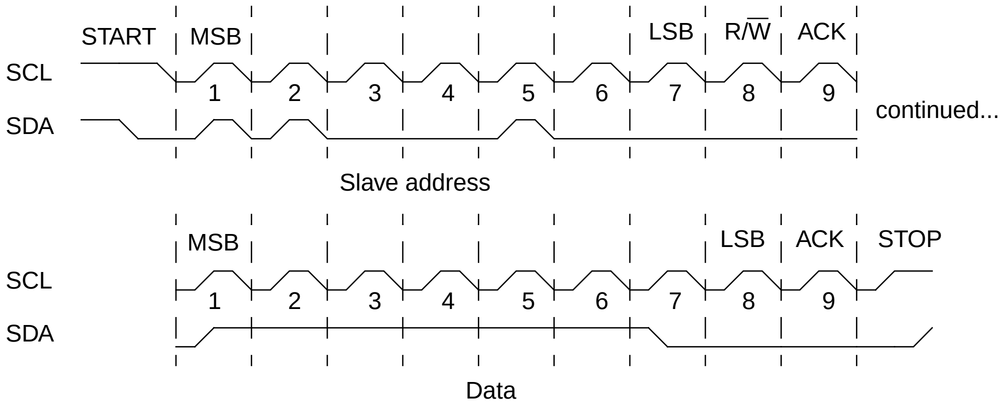


Figure 178. Complete I^2C data transfer

If the data bit rate is too high for a slave device or if the slave needs time to process data, it may stretch the clock period by keeping SCL low after the master has driven SCL low. Clock stretching is a configurable parameter of the core (see sections 61.2.4 and 61.2.6).

61.2.2 Slave addressing

The core’s I^2C addresses are set with VHDL generics at implementation time. If the core has been implemented with the optional APB interface, then the I^2C addresses can be changed via registers available via APB.

The core responds to two addresses on the I^2C bus. Accesses to the I2C memory address are translated to AMBA AHB accesses and accesses to the I^2C configuration address access the core’s configuration register.

61.2.3 System clock requirements and sampling

The core samples the incoming I^2C SCL clock and does not introduce any additional clock domains into the system. Both the SCL and SDA lines first pass through two stage synchronizers and are then filtered with a low pass filter consisting of four registers.

START and STOP conditions are detected if the SDA line, while SCL is high, is at one value for two system clock cycles, toggles and keeps the new level for two system clock cycles.

The synchronizers and filters constrain the minimum system frequency. The core requires the SCL signal to be stable for at least four system clock cycles before the core accepts the SCL value as the new clock value. The core’s reaction to transitions will be additionally delayed since both lines are taken through two-stage synchronizers before they are filtered. Therefore it takes the core over eight system clock cycles to discover a transition on SCL.

61.2.4 Configuration register access

The I^{2}C configuration register is accessed via a separate I^{2}C address (I^{2}C configuration address). The configuration register has the layout shown in table 866.

Table 866. I2C2AHB configuration register

7	6	5	4	3	2	1	0
Reserved		PROT	MEXC	DMAACT	NACK	HSIZE	

7:6	Reserved, always zero (read only)
5	Memory protection triggered (PROT) - ‘1’ if last AHB access was outside the allowed memory area. Updated after each AMBA access (read only)
4	Memory exception (MEXC) - ‘1’ if core receives AMBA ERROR response. Updated after each AMBA access (read only)
3	DMA active (DMAACT) - ‘1’ if core is currently performing a DMA operation.
2	NACK (NACK) - Use NACK instead of clock stretching. See documentation in section 61.2.6.
1:0	AMBA access size (HSIZE) - Controls the access size that the core will use for AMBA accesses. 0: byte, 1: halfword, 2: word. HSIZE = “11” is illegal.

Reset value: 0x02

Reads from the I^{2}C configuration address will return the current value of the configuration register. Writes to the I^{2}C configuration address will affect the writable bits in the configuration register.

61.2.5 AHB accesses

All AMBA accesses are done in big endian format. The first byte sent to or from the slave is the most significant byte.

To write a word on the AHB bus the following I2C bus sequence should be performed:

1. Generate START condition
2. Send I2C memory address with the R/$\overline{W}$ bit set to ‘0’.
3. Send four byte AMBA address, the most significant byte is transferred first
4. Send four bytes to write to the specified address
5. If more than four consecutive bytes should be written, continue to send additional bytes, otherwise go to 6.
6. Generate STOP condition

To perform a read access on the AHB bus, the following I2C bus sequence should be performed:

1. Generate START condition
2. Send I2C memory address with the R/$\overline{W}$ bit set to ‘0’.
3. Send four byte AMBA address, the most significant byte is transferred first
4. Generate (repeated) START condition
5. Send I2C memory address with the R/$\overline{W}$ bit set to ‘0’.
6. Read the required number of bytes and NACK the last byte
7. Generate stop condition

During consecutive read or write operations, the core will automatically increment the address. The access size (byte, halfword or word) used on AHB is set via the HSIZE field in the I2C2AHB configuration register.

The core always respects the access size specified via the HSIZE field. If a write operation writes fewer bytes than what is required to do an access of the specified HSIZE then the write data will be dropped, no access will be made on AHB. If a read operation reads fewer bytes than what is specified by HSIZE then the remaining read data will be dropped at a START or STOP condition. This means, for instance, that if HSIZE is “10” (word) the core will perform two word accesses if a master reads one byte, generates a repeated start condition, and reads one more byte. Between these two accesses the address will have been automatically increased, so the fist access will be to address *n* and the second to address *n+4*.

The automatic address increment means that it is possible to write data and then immediately read the data located at the next memory position. As an example, the following sequence will write a word to address 0 and then read a word from address 4:

1. Generate START condition
2. Send I2C memory address with the $R/\overline{W}$ bit set to ‘0’.
3. Send four byte AMBA address, all zero.
4. Send four bytes to write to the specified address
5. Generate (repeated) START condition
6. Send I2C memory address with the $R/\overline{W}$ bit set to ‘0’.
7. Read the required number of bytes and lack the last byte
8. Generate stop condition

The core will not mask any address bits. Therefore it is important that the I^2C master respects AMBA rules when performing halfword and word accesses. A halfword access must be aligned on a two byte address boundary (least significant bit of address must be zero) and a word access must be aligned on a four byte boundary (two least significant address bits must be zero).

The core can be configured to generate interrupt requests when an AHB access is performed if the core is implemented with the APB register interface, see the APB register documentation for details.

61.2.6 Clock stretching or NACK mode

The core has two main modes of operation for AMBA accesses. In one mode the core will use clock stretching while performing an AHB operation and in the other mode the core will not acknowledge bytes (abort the I^2C access) when the core is busy. Clock stretching is the preferred mode of operation. The NACK mode can be used in scenarios where the I^2C master or physical layer does not support clock stretching. The mode to use is selected via the NACK field in the I^2C configuration register.

When clock stretching is enabled (NACK field is ‘0’) the core will stretch the clock when the slave is accessed (via the I2C memory address) and the slave is busy processing a transfer. Clock stretching is also used when a data byte has been transmitted, or received, to keep SCL low until a DMA operation has completed. In the transmit (AMBA read) case SCL is kept low before the rising edge of the first byte. In the receive case (AMBA write) the ACK cycle for the previous byte is stretched.

When clock stretching is disabled (NACK field is ‘1’) the core will never stretch the SCL line. If the core is busy performing DMA when it is addressed, the address will not be acknowledged. If the core performs consecutive writes and the first write operation has not finished the core will now acknowledge the written byte. If the core performs a read operation and the read DMA operation has not finished when the core is supposed to deliver data then the core will go to its idle state and not respond to more accesses until a START condition is generated on the bus. This last part means that the NACK mode is practically unusable in systems where the AMBA access can take longer than one I^2C clock period. This can be compensated by using a very slow I^2C clock.

61.2.7 Memory protection

The core is configured at implementation time to only allow accesses to a specified AHB address range (which can be the full 4 GiB AMBA address range). If the core has been implemented with the optional APB register interface then the address range is soft configurable and the reset value is specified with VHDL generics.

The VHDL generics *ahbaddrh* and *ahbaddrl* define the base address for the allowed area. The VHDL generics *ahbmaskh* and *ahbmaskl* define the size of the area. The generics are used to assign the memory protection area's address and mask in the following way:

Protection address, bits 31:16 (*protaddr[31:16]*): ahbaddrh
Protection address, bits 15:0 (*protaddr[15:0]*): ahbaddrl
Protection address, bits 31:16 (*protmask[31:16]*): ahbmaskh
Protection address, bits 15:0 (*protmask[15:0]*): ahbmaskh

Before the core performs an AMBA access it will perform the check:

(((incoming address) xor (protaddr)) and protmask) /= 0x00000000

If the above expression is true (one or several bits in the incoming address differ from the protection address, and the corresponding mask bits are set to '1') then the access is inhibited. As an example, assume that *protaddr* is 0xA0000000 and *protmask* is 0xF0000000. Since *protmask* only has ones in the most significant nibble, the check above can only be triggered for these bits. The address range of allowed accessed will thus be 0xA0000000 - 0xAFFFFFFF.

The memory protection check is performed at the time when the core is to perform the AHB access. It is possible to start a write operation and transmit an illegal address to the core without any errors. If additional bytes are transmitted (so that a HSIZE access can be made) the core will NACK the byte that triggers the AHB access.

For a read operation the core will NACK the I^{2}C memory address of the first AHB access of the read in case the access would be to restricted memory. If consecutive bytes are read from the core and one of the later accesses lead to restricted memory being accessed, then the core will abort all operations and enter its idle state. In this case junk data will be returned and there is no way for the core to alert the master that memory protection has been triggered.

The core will set the configuration register bit PROT if an access is attempted outside the allowed address range. This bit is updated on each AHB access and will be cleared by an access inside the allowed range. Note that the (optional) APB status register has a PROT field with a slightly different behavior.

61.3 Registers

The core can optionally be implemented with an APB interface that provides registers mapped into APB address space.

*Table 867.*I^{2}C slave registers

APB address offset	**Register**
0x00	Control register
0x04	Status register
0x08	Protection address register
0x0C	Protection mask register
0x10	I2C slave memory address register
0x14	I2C slave configuration address register

Table 868. Control register

31 : 2	1	0
RESERVED	IRQEN	EN

31 : 2	RESERVED
1	Interrupt enable (IRQEN) - When this bit is set to '1' the core will generate an interrupt each time the DMA field in the status register transitions from '0' to '1'.

Table 868. Control register

0	Core enable (EN) - When this bit is set to ‘1’ the core is enabled and will respond to I2C accesses. Otherwise the core will not react to I2C traffic.

Reset value: Implementation dependent

Table 869. Status register

31 3	2	1	0
RESERVED	PROT	WR	DMA

31 : 3	RESERVED
2	Protection triggered (PROT) - Set to ‘1’ if an access has triggered the memory protection. This bit will remain set until cleared by writing ‘1’ to this position. Note that the other fields in this register will be updated on each AHB access while the PROT bit will remain at ‘1’ once set.
1	Write access (WR) - Last AHB access performed was a write access. This bit is read only.
0	Direct Memory Access (DMA) - This bit gets set to ‘1’ each time the core attempts to perform an AHB access. By setting the IRQEN field in the control register this condition can generate an interrupt. This bit can be cleared by software by writing ‘1’ to this position.

Reset value: 0x00000000

Table 870. Protection address register

31 0
PROTADDR

31 : 0	Protection address (PROTADDR) - Defines the base address for the memory area where the core is allowed to make accesses.

Reset value: Implementation dependent

Table 871. Protection mask register

31 3 2 1 0
PROTMASK

31 : 0	Protection mask (PROTMASK) - Selects which bits in the Protection address register that are used to define the protected memory area.

Reset value: Implementation dependent

Table 872. I2C slave memory address register

31 7	6 0
RESERVED	I2CSLVADDR

31 : 7	RESERVED
6 : 0	I2C slave memory address (I2CSLVADDR) - Address that slave responds to for AHB memory accesses

Reset value: Implementation dependent

Table 873. I2C slave configuration address register

31 7	6 0
RESERVED	I2CCFGADDR

Table 873. I2C slave configuration address register

31 : 7	RESERVED
6 : 0	I2C slave configuration address (I2CCFGADDR) - Address that slave responds to for configuration register accesses.

Reset value: Implementation dependent

61.4 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x00B. For a description of vendor and device identifiers see the GRLIB IP Library User's Manual.

61.5 Configuration options

Table 874 shows the configuration options of the core (VHDL generics). Two different top level entites for the core is available. One with the optional APB interface (i2c2ahb_apb) and one without the APB interface (i2c2ahb). The entity without the APB interface has fewer generics as indicated in the table below.

*Table 874.*Configuration options

Generic name	**Function**	**Allowed range**	**Default**
hindex	AHB master index	0 - NAHBMST	0
ahbaddrh	Defines bits 31:16 of the address used for the memory protection area	0 - 16#FFFF#	0
ahbaddrl	Defines bits 15:0 of the address used for the memory protection area	0 - 16#FFFF#	0
ahbmaskh	Defines bits 31:16 of the mask used for the memory protection area	0 - 16#FFFF#	0
ahbmaskl	Defines bits 15:0 of the mask used for the memory protection area	0 - 16#FFFF#	0
resen	Reset value for core enable bit (only available on the i2c2ahb_apb entity).	0 - 1	0
pindex	APB slave index (only available on the i2c2ahb_apb entity).	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR (only available on the i2c2ahb_apb entity).	0 - 16#FFF#	0
pmask	MASK field of the APB BAR (only available on the i2c2ahb_apb entity).	0 - 16#FFF#	16#FFF#
pirq	Interrupt line driven by APB interface (only available on the i2c2ahb_apb entity).	0 - NAHBIRQ-1	0
i2caddr	The slave's (initial) I^{2}C address. i2caddr specified the core's I2C memory address and (i2caddr+1) will be the cores I2C configuration address.	0 - 126	0
oepol	Output enable polarity	0 - 1	0

*Table 874.*Configuration options

Generic name	Function	Allowed range	Default
filter	Low-pass filter length. This generic should specify, in number of system clock cycles plus one, the time of the shortest pulse on the I2C bus to be registered as a valid value. For instance, to disregard any pulse that is 50 ns or shorter in a system with a system frequency of 54 MHz this generic should be set to: ((pulse time) / (clock period)) + 1 = (50 ns) / ((1/(54 MHz)) + 1 = 3.7 The value from this calculation should always be rounded up. In other words an appropriate filter length for a 54 MHz system is 4.	2 - 512	2

61.6 Signal descriptions

Table 875 shows the interface signals of the core (VHDL ports).

*Table 875.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
I2CI	SCL	Input	I^2C clock line input	-
	SDA	Input	I^2C data line input	-
I2CO	SCL	Output	I^2C clock line output	-
	SCLOEN	Output	I^2C clock line output enable	Low**
	SDA	Output	I^2C data line output	-
	SDAOEN	Output	I^2C data line output enable	Low**
	ENABLE	Output	High when core is enabled, low otherwise.	High

* see GRLIB IP Library User's Manual
** depends on value of OEPOL VHDL generic.

61.7 Library dependencies

Table 876 shows the libraries used when instantiating the core (VHDL libraries).

*Table 876.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	I2C	Component, signals	Component declaration, I2C signal definitions

61.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity i2c2ahb_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- I2C signals
    iic_scl : inout std_ulogic;
    iic_sda : inout std_ulogic
    );
end;

architecture rtl of i2c2ahb_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector;
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector;

  -- I2C signals
  signal i2ci : i2c_in_type;
  signal i2co : i2c_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- NOTE: There are also wrappers for the top-level entities that do not make use of VHDL
  --      records. These wrappers are called i2c2ahb_apb_gen and i2c2ahb_gen.

  -- I2C-slave, with APB interface
  i2c2ahb0 : i2c2ahb_apb
    generic map (
      hindex   => 1,
      ahbaddrh => ahbaddrh,
      ahbaddrl => ahbaddrl,
      ahbmaskh => ahbmaskh,
      ahbmaskl => ahbmaskl,
      resen    => 1,
      pindex   => 1,
      paddr    => 1,
      pmask    => 16#fff#,
      i2caddr  => i2caddr,
      oepol    => 0,
      filter   => I2C_FILTER)
    port map (rstn, clk, ahbmi, ahbmo(1), apbi, apbo(1),
              i2ci, i2co);
  scl_pad : iopad generic map (tech => padtech)
    port map (iic_scl, i2co.scl, i2co.scloen, i2ci.scl);
  sda_pad : iopad generic map (tech => padtech)
    port map (iic_sda, i2co.sda, i2co.sdaoen, i2ci.sda);
end;
```

62 I2CMST - I^{2}C-master

62.1 Overview

The I^{2}C-master core is a modified version of the OpenCores I^{2}C-Master with an AMBA APB interface. The core is compatible with Philips I^{2}C standard and supports 7- and 10-bit addressing. Standard-mode (100 kb/s) and Fast-mode (400 kb/s) operation are supported directly. External pull-up resistors must be supplied for both bus lines.

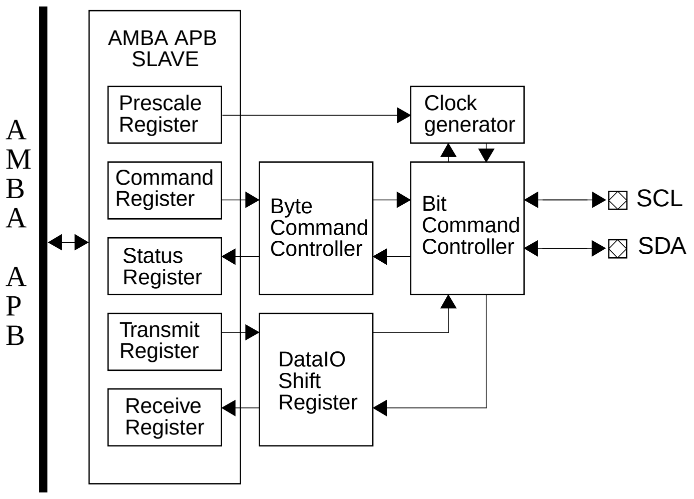


Figure 179. Block diagram

62.2 Operation

62.2.1 Transmission protocol

The I^{2}C-bus is a simple 2-wire serial multi-master bus with collision detection and arbitration. The bus consists of a serial data line (SDA) and a serial clock line (SCL). The I^{2}C standard defines three transmission speeds; Standard (100 kb/s), Fast (400 kb/s) and High speed (3.4 Mb/s).

A transfer on the I^{2}C-bus begins with a START condition. A START condition is defined as a high to low transition of the SDA line while SCL is high. Transfers end with a STOP condition, defined as a low to high transition of the SDA line while SCL is high. These conditions are always generated by a master. The bus is considered to be busy after the START condition and is free after a certain amount of time following a STOP condition. The bus free time required between a STOP and a START condition is defined in the I^{2}C-bus specification and is dependent on the bus bit rate.

Figure 180 shows a data transfer taking place over the I^{2}C-bus. The master first generates a START condition and then transmits the 7-bit slave address. The bit following the slave address is the R/$\overline{\text{W}}$ bit which determines the direction of the data transfer. In this case the R/$\overline{\text{W}}$ bit is zero indicating a write operation. After the master has transmitted the address and the R/$\overline{\text{W}}$ bit it releases the SDA line. The receiver pulls the SDA line low to acknowledge the transfer. If the receiver does not acknowledge the transfer, the master may generate a STOP condition to abort the transfer or start a new transfer by generating a repeated START condition.

After the first byte has been acknowledged the master transmits the data byte. If the R/$\overline{\text{W}}$ bit had been set to ‘1’ the master would have acted as a receiver during this phase of the transfer. After the data byte has been transferred the receiver acknowledges the byte and the master generates a STOP condition to complete the transfer. Section 62.2.3 contains three more example transfers from the perspective of a software driver.

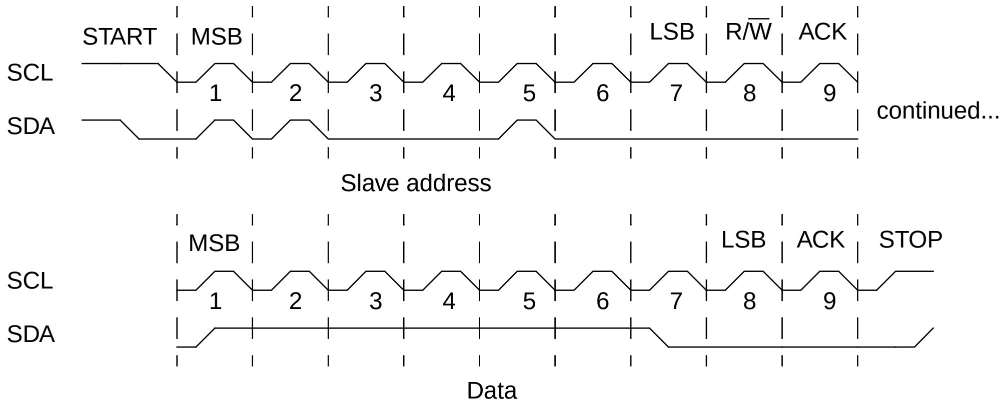


Figure 180. Complete I^{2}C data transfer

If the data bitrate is too high for a slave device, it may stretch the clock period by keeping SCL low after the master has driven SCL low.

62.2.2 Clock generation

The core uses the prescale register to determine the frequency of the SCL clock line and of the 5*SCL clock that the core uses internally. To calculate the prescale value use the formula:

$$Prescale = \frac{AMBAclockfrequency}{5 \cdot SCLfrequency} - 1$$

The *SCLfrequency* is 100 kHz for Standard-mode operation (100 kb/s) and 400 kHz for Fast mode operation. To use the core in Standard-mode in a system with a 60 MHz clock driving the AMBA bus the required prescale value is:

$$Prescale = \frac{60Mhz}{5 \cdot 100kHz} - 1 = 119 = 0x77$$

Note that the prescale register should only be changed when the core is disabled. The minimum recommended prescale value is 3 due to synchronization issues. This limits the minimum system frequency to 2 MHz for operation in Standard-mode (to be able to generate a 100 kHz SCL clock). However, a system frequency of 2 MHz will not allow the implementation fulfill the 100 ns minimum requirement for data setup time (required for Fast-mode operation). For compatibility with the I^{2}C Specification, in terms of minimum required data setup time, the minimum allowed system frequency is 20 MHz due to synchronization issues. If the core is run at lower system frequencies, care should be taken so that data from devices is stable on the bus one system clock period before the rising edge of SCL.

62.2.3 Software operational model

The core is initialized by writing an appropriate value to the clock prescale register and then setting the enable (EN) bit in the control register. Interrupts are enabled via the interrupt enable (IEN) bit in the control register.

To write a byte to a slave the I^{2}C-master must generate a START condition and send the slave address with the R/$\overline{W}$ bit set to ‘0’. After the slave has acknowledged the address, the master transmits the data, waits for an acknowledge and generates a STOP condition. The sequence below instructs the core to perform a write:

1. Left-shift the I^2C-device address one position and write the result to the transmit register. The least significant bit of the transmit register ($R/\overline{W}$) is set to ‘0’.

2. Generate START condition and send contents of transmit register by setting the STA and WR bits in the command register.

3. Wait for interrupt, or for TIP bit in the status register to go low.

4. Read RxACK bit in status register. If RxACK is low the slave has acknowledged the transfer, proceed to step 5. If RxACK is set the device did not acknowledge the transfer, go to step 1.

5. Write the slave-data to the transmit register.

6. Send the data to the slave and generate a stop condition by setting STO and WR in the command register.

7. Wait for interrupt, or for TIP bit in the status register to go low.

8. Verify that the slave has acknowledged the data by reading the RxACK bit in the status register. RxACK should not be set.

To read a byte from an I^2C-connected memory much of the sequence above is repeated. The data written in this case is the memory location on the I^2C slave. After the address has been written the master generates a repeated START condition and reads the data from the slave. The sequence that software should perform to read from a memory device:

1. Left-shift the I^2C-device address one position and write the result to the transmit register. The least significant bit of the transmit register (R/W) is set to ‘0’.

2. Generate START condition and send contents of transmit register by setting the STA and WR bits in the command register.

3. Wait for interrupt or for TIP bit in the status register to go low.

4. Read RxACK bit in status register. If RxACK is low the slave has acknowledged the transfer, proceed to step 5. If RxACK is set the device did not acknowledge the transfer, go to step 1.

5. Write the memory location to be read from the slave to the transmit register.

6. Set the WR bit in the command register. Note that a STOP condition is not generated here.

7. Wait for interrupt, or for TIP bit in the status register to go low.

8. Read RxACK bit in the status register. RxACK should be low.

9. Address the I^2C-slave again by writing its left-shifted address into the transmit register. Set the least significant bit of the transmit register (R/W) to ‘1’ to read from the slave.

10. Set the STA and WR bits in the command register to generate a repeated START condition.

11. Wait for interrupt, or for TIP bit in the status register to go low.

12. Read RxACK bit in the status register. The slave should acknowledge the transfer.

13. Prepare to receive the data read from the I^2C-connected memory. Set bits RD, ACK and STO on the command register. Setting the ACK bit NAKs the received data and signifies the end of the transfer.

14. Wait for interrupt, or for TIP in the status register to go low.

15. The received data can now be read from the receive register.

To perform sequential reads the master can iterate over steps 13 - 15 by not setting the ACK and STO bits in step 13. To end the sequential reads the ACK and STO bits are set. Consult the documentation of the I^2C-slave to see if sequential reads are supported.

The final sequence illustrates how to write one byte to an I^{2}C-slave which requires addressing. First the slave is addressed and the memory location on the slave is transmitted. After the slave has acknowledged the memory location the data to be written is transmitted without a generating a new START condition:

1. Left-shift the I^{2}C-device address one position and write the result to the transmit register. The least significant bit of the transmit register (R/W) is set to ‘0’.
2. Generate START condition and send contents of transmit register by setting the STA and WR bits in the command register.
3. Wait for interrupt or for TIP bit in the status register to go low.
4. Read RxACK bit in status register. If RxACK is low the slave has acknowledged the transfer, proceed to step 5. If RxACK is set the device did not acknowledge the transfer, go to step 1.
5. Write the memory location to be written from the slave to the transmit register.
6. Set the WR bit in the command register.
7. Wait for interrupt, or for TIP bit in the status register to go low.
8. Read RxACK bit in the status register. RxACK should be low.
9. Write the data byte to the transmit register.
10. Set WR and STO in the command register to send the data byte and then generate a STOP condition.
11. Wait for interrupt, or for TIP bit in the status register to go low.
12. Check RxACK bit in the status register. If the write succeeded the slave should acknowledge the data byte transfer.

The example sequences presented here can be generally applied to I^{2}C-slaves. However, some devices may deviate from the protocol above, please consult the documentation of the I^{2}C-slave in question. Note that a software driver should also monitor the arbitration lost (AL) bit in the status register.

62.2.4 Signal filters

The core is configured at implementation to use one of two possible filter strategies: a static filter or a dynamic filter, the selection between the two options is made with the *dynfilt* VHDL generic.

With a static filter (*dynfilt* = 0) the core will implement low-pass filters using simple shift registers. The number of shift registers is determined by the VHDL generic *filter*. When all bits in a shift register are equal, the core will consider the state of the input signal (SCL or SDA) to have changed. An appropriate value for the filter generic is calculated via:

$$filter = \frac{pulse\overline{time}}{systemclockperiod} + 1$$

To disregard any pulse that is 50 ns or shorter in a system with a system frequency of 54 MHz the *filter* generic should be set to: (50 ns) / ((1/(54 MHz)) + 1 = 3.7. The value from this calculation should always be rounded up. In other words an appropriate filter length for a 54 MHz system is 4.

With a dynamic filter (*dynfilt* = 1) the VHDL generic *filter* determines the number of bits implemented in a counter that controls the sample window. The reload value for the counter can then be specified by software by writing to the core’s dynamic filter register available via the APB interface. The number of bits required for the dynamic counter is calculated using (where system clock period is the shortest system clock period that the design will use):

$$filter = \log 2\left(\frac{pulse\overline{time}}{systemclockperiod} + 1\right)$$

When using dynamic filtering, the core will ignore all pulses shorter than the system clock period multiplied with the value of the FILT field in the core’s Dynamic Filter register and may also ignore pulses that are shorter than 2 * FILT * (system clock period) - 1.

62.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 877.*I^{2}C-master registers

APB address offset	Register
0x00	Clock prescale register
0x04	Control register
0x08	Transmit register*
0x08	Receive register**
0x0C	Command register*
0x0C	Status register**
0x10	Dynamic filter register***

* Write only

** Read only

*** Only available on some implementations

Table 878. I^{2}C-master Clock prescale register

31	16	15	7	6	5	4	3	2	1	0
RESERVED		Clock prescale								

31 : 16	RESERVED
15:0	Clock prescale - Value is used to prescale the SCL clock line. Do not change the value of this register unless the EN field of the control register is set to ‘0’. The minimum recommended value of this register is 0x0003. Lower values may cause the master to violate I^{2}C timing requirements due to synchronization issues.

Table 879. I^{2}C-master control register

31	8	7	6	5	0
RESERVED		EN	IEN	RESERVED	

31 : 8	RESERVED
7	Enable (EN) - Enable I^{2}C core. The core is enabled when this bit is set to ‘1’.
6	Interrupt enable (IEN) - When this bit is set to ‘1’ the core will generate interrupts upon transfer completion.
5:0	RESERVED

Table 880. I^{2}C-master transmit register

31	8	7	1	0
RESERVED		TDATA		RW

31 : 8	RESERVED
7:1	Transmit data (TDATA) - Most significant bits of next byte to transmit via I^{2}C
0	Read/Write (RW) - In a data transfer this is the data’s least significant bit. In a slave address transfer this is the RW bit. ‘1’ reads from the slave and ‘0’ writes to the slave.

Table 881. I^2C-master receive register

31	8	7	0
RESERVED		RDATA	

31 : 8	RESERVED
7:0	Receive data (RDATA) - Last byte received over I^2C-bus.

Table 882. I^2C-master command register

31	8	7	6	5	4	3	2	1	0
RESERVED		STA	STO	RD	WR	ACK	RESERVED		IACK

31 : 8	RESERVED
7	Start (STA) - Generate START condition on I^2C-bus. This bit is also used to generate repeated START conditions.
6	Stop (STO) - Generate STOP condition
5	Read (RD) - Read from slave
4	Write (WR) - Write to slave
3	Acknowledge (ACK) - Used when acting as a receiver. ‘0’ sends an ACK, ‘1’ sends a NACK.
2:1	RESERVED
0	Interrupt acknowledge (IACK) - Clears interrupt flag (IF) in status register.

Table 883. I^2C-master status register

31	8	7	6	5	4	3	2	1	0
RESERVED		RxACK	BUSY	AL	RESERVED			TIP	IF

31 : 8	RESERVED
7	Receive acknowledge (RxACK) - Received acknowledge from slave. ‘1’ when no acknowledge is received, ‘0’ when slave has acked the transfer.
6	I^2C-bus busy (BUSY) - This bit is set to ‘1’ when a start signal is detected and reset to ‘0’ when a stop signal is detected.
5	Arbitration lost (AL) - Set to ‘1’ when the core has lost arbitration. This happens when a stop signal is detected but not requested or when the master drives SDA high but SDA is low.
4:2	RESERVED
1	Transfer in progress (TIP) - ‘1’ when transferring data and ‘0’ when the transfer is complete. This bit is also set when the core will generate a STOP condition.
0	Interrupt flag (IF) - This bit is set when a byte transfer has been completed and when arbitration is lost. If IEN in the control register is set an interrupt will be generated. New interrupts will ge generated even if this bit has not been cleared.

Table 884. I^2C-master dynamic filter register

31	x	x-1	0
RESERVED		FILT	

31 : x	RESERVED
x-1 : 0	Dynamic filter reload value (FILT) - This field sets the reload value for the dynamic filter counter. The core will ignore all pulses on the bus shorter than FILT * (system clock period) and may also ignore pulses shorter than 2 * FILT * (system clock period) - 1. The reset value of this register is all ‘1’. This register is not available in all implementations, and only for core revisions higher than two (the core’s version number can be read from the plug’n’play area). When implemented, the number of bits in the FILT field is implementation dependent. Software can probe the precense of this register by writing 0x1 to the register location and reading back the value. If the read value is non-zero then the core has been implemented with a dynamic filter.

62.4 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x028. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

62.5 Configuration options

Table 885 shows the configuration options of the core (VHDL generics).

*Table 885.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by I^{2}C-master	0 - NAHBIRQ-1	0
oepol	Output enable polarity	0 - 1	0
filter	Low-pass filter length. This generic should specify, in number of system clock cycles plus one, the time of the shortest pulse on the I2C bus to be registered as a valid value. For instance, to disregard any pulse that is 50 ns or shorter in a system with a system frequency of 54 MHz this generic should be set to: ((pulse time) / (clock period)) + 1 = (50 ns) / ((1/(54 MHz)) + 1 = 3.7 The value from this calculation should always be rounded up. In other words an appropriate filter length for a 54 MHz system is 4. Note that the value of this generic changes meaning if the dynfilt generic described below is non-zero. See description below.	2 - 512	2
dynfilt	Dynamic low-pass filter length. If this generic is non-zero the core will be implemented with a configurable filter. If dynfilt is non-zero the filter generic, described above, specifies how many bits that will be implemented for the dynamic filter counter.	0 - 1	0

62.6 Signal descriptions

Table 886 shows the interface signals of the core (VHDL ports).

*Table 886.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
I2CI	SCL	Input	I^{2}C clock line input	-
	SDA	Input	I^{2}C data line input	-
I2CO	SCL	Output	I^{2}C clock line output	-
	SCLOEN	Output	I^{2}C clock line output enable	Low
	SDA	Output	I^{2}C data line output	-
	SDAOEN	Output	I^{2}C data line output enable	Low

* see GRLIB IP Library User’s Manual

62.7 Library dependencies

Table 887 shows the libraries used when instantiating the core (VHDL libraries).

*Table 887.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	I2C	Component, signals	Component declaration, I2C signal definitions

62.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity i2c_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- I2C signals
    iic_scl : inout std_ulogic;
    iic_sda : inout std_ulogic
    );
end;

architecture rtl of i2c_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  -- I2C signals
  signal i2ci : i2c_in_type;
  signal i2co : i2c_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- I2C-master
  i2c0 : i2cmst
    generic map (pindex => 12, paddr => 12, pmask => 16#FFF#,
                 pirq => 8, filter => (BUS_FREQ_in_kHz*5+50000)/100000+1)
    port map (rstn, clkm, apbi, apbo(12), i2ci, i2co);

  -- Using bi-directional pads:
  i2c_scl_pad : iopad generic map (tech => padtech)
    port map (iic_scl, i2co.scl, i2co.scloen, i2ci.scl);
  i2c_sda_pad : iopad generic map (tech => padtech)
    port map (iic_sda, i2co.sda, i2co.sdaoen, i2ci.sda);
  -- Note: Some designs may want to use a uni-directional pad for the clock. In this case the
  -- the clock should have a on-chip feedback like: i2ci.scl <= i2co.scloen (for OEPOL = 0)
  -- This feedback connection should have the same delay as i2co.sdaoen to i2ci.sda
end;
```

63 I2CSLV - I^{2}C slave

63.1 Overview

The I^{2}C slave core is a simple I^{2}C slave that provides a link between the I^{2}C bus and the AMBA APB. The core is compatible with Philips I^{2}C standard and supports 7- and 10-bit addressing with an optionally software programmable address. Standard-mode (100 kb/s) and Fast-mode (400 kb/s) operation are supported directly. External pull-up resistors must be supplied for both bus lines.

GRLIB also contains another I^{2}C slave core that has DMA capabilities, see the I2C2AHB core documentation for details.

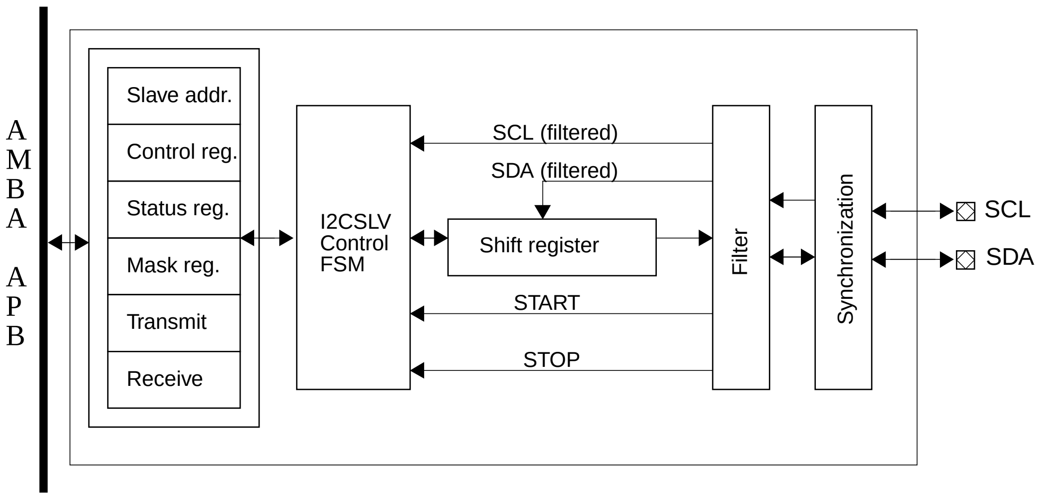


Figure 181. Block diagram

63.2 Operation

63.2.1 Transmission protocol

The I^{2}C-bus is a simple 2-wire serial multi-master bus with collision detection and arbitration. The bus consists of a serial data line (SDA) and a serial clock line (SCL). The I^{2}C standard defines three transmission speeds; Standard (100 kb/s), Fast (400 kb/s) and High speed (3.4 Mb/s).

A transfer on the I^{2}C-bus begins with a START condition. A START condition is defined as a high to low transition of the SDA line while SCL is high. Transfers end with a STOP condition, defined as a low to high transition of the SDA line while SCL is high. These conditions are always generated by a master. The bus is considered to be busy after the START condition and is free after a certain amount of time following a STOP condition. The bus free time required between a STOP and a START condition is defined in the I^{2}C-bus specification and is dependent on the bus bit rate.

Figure 182 shows a data transfer taking place over the I^{2}C-bus. The master first generates a START condition and then transmits the 7-bit slave address. I^{2}C also supports 10-bit addresses, which are discussed briefly below. The bit following the slave address is the R/$\overline{W}$ bit which determines the direction of the data transfer. In this case the R/$\overline{W}$ bit is zero indicating a write operation. After the master has transmitted the address and the R/$\overline{W}$ bit it releases the SDA line. The receiver pulls the SDA line low to acknowledge the transfer. If the receiver does not acknowledge the transfer, the master may generate a STOP condition to abort the transfer or start a new transfer by generating a repeated START condition.

After the address has been acknowledged the master transmits the data byte. If the R/$\overline{W}$ bit had been set to '1' the master would have acted as a receiver during this phase of the transfer. After the data byte has been transferred the receiver acknowledges the byte and the master generates a STOP condition to complete the transfer.

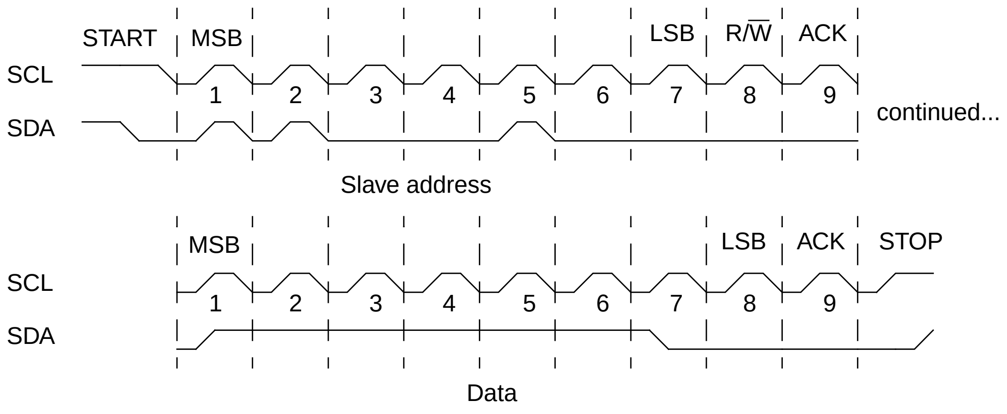


Figure 182. Complete I^{2}C data transfer

An I^{2}C slave may also support 10-bit addressing. In this case the master first transmits a pattern of five reserved bits followed by the two first bits of the 10-bit address and the R/$\overline{W}$ bit set to ‘0’. The next byte contains the remaining bits of the 10-bit address. If the transfer is a write operation the master then transmits data to the slave. To perform a read operation the master generates a repeated START condition and repeats the first part of the 10-bit address phase with the R/$\overline{W}$ bit set to ‘1’.

If the data bitrate is too high for a slave device or if the slave needs time to process data, it may stretch the clock period by keeping SCL low after the master has driven SCL low.

63.2.2 Slave addressing

The core’s addressing support is implementation dependent. The core may have a programmable address and may support 10-bit addresses. If the core has support for 10-bit addressing, the TBA bit of the Slave address register will be set to ‘1’ after reset. If the core’s address is programmable this bit is writable and is used by the core to determine if it should listen to a 7- or 10-bit address.

Software can determine the addressing characteristics of the core by writing and reading the Slave address register. The core supports 10-bit addresses if the TBA bit is, or can be set, to ‘1’. The core has a software programmable address if the SLVADDR field in the same register can be changed.

63.2.3 System clock requirements and sampling

The core samples the incoming I^{2}C SCL clock and does not introduce any additional clock domains into the system. Both the SCL and SDA lines first pass through two stage synchronizers and are then filtered with a low pass filter consisting of four registers.

START and STOP conditions are detected if the SDA line, while SCL is high, is at one value for two system clock cycles, toggles and keeps the new level for two system clock cycles.

The synchronizers and filters constrain the minimum system frequency. The core requires the SCL signal to be stable for at least four system clock cycles before the core accepts the SCL value as the new clock value. The core’s reaction to transitions will be additionally delayed since both lines are taken through two-stage synchronizers before they are filtered. Therefore it takes the core over eight system clock cycles to discover a transition on SCL. To use the slave in Standard-mode operation at 100 kHz the recommended minimum system frequency is 2 MHz. For Fast-mode operation at 400 kHz the recommended minimum system frequency is 6 MHz.

63.2.4 Operational model

The core has four main modes of operation and is configured to use one of these modes via the Control register bits Receive Mode (RMOD) and Transmit Mode (TMOD). The mode setting controls the core's behavior after a byte has been received or transmitted.

The core will always NAK a received byte if the receive register is full when the whole byte is received. If the receive register is free the value of RMOD determines if the core should continue to listen to the bus for the master's next action or if the core should drive SCL low to force the master into a wait state. If the value of the RMOD field is '0' the core will listen for the master's next action. If the value of the RMOD field is '1' the core will drive SCL low until the Receive register has been read and the Status register bit Byte Received (REC) has been cleared. Note that the core has not accepted a byte if it does not acknowledge the byte.

When the core receives a read request it evaluates the Transmit Valid (TV) bit in the Control register. If the Transmit Valid bit is set the core will acknowledge the address and proceed to transmit the data held in the Transmit register. After a byte has been transmitted the core assigns the value of the Control register bit Transmit Always Valid (TAV) to the Transmit Valid (TV) bit. This mechanism allows the same byte to be sent on all read requests without software intervention. The value of the Transmit Mode (TMOD) bit determines how the core acts after a byte has been transmitted and the master has acknowledged the byte, if the master NAKs the transmitted byte the transfer has ended and the core goes into an idle state. If TMOD is set to '0' when the master acknowledges a byte the core will continue to listen to the bus and wait for the master's next action. If the master continues with a sequential read operation the core will respond to all subsequent requests with the byte located in the Transmit Register. If TMOD is '1' the core will drive SCL low after a master has acknowledged the transmitted byte. SCL will be driven low until the Transmit Valid bit in the control register is set to '1'. Note that if the Transmit Always Valid (TAV) bit is set to '1' the Transmit Valid bit will immediately be set and the core will have show the same behavior for both Transmit modes.

When operating in Receive or Transmit Mode '1', the bus will be blocked by the core until software has acknowledged the transmitted or received byte. This may have a negative impact on bus performance and it also affects single byte transfers since the master is prevented to generate STOP or repeated START conditions when SCL is driven low by the core.

The core reports three types of events via the Status register. When the core NAKs a received byte, or its address in a read transfer, the NAK bit in the Status register will be set. When a byte is successfully received the core asserts the Byte Received (REC) bit. After transmission of a byte, the Byte Transmitted (TRA) bit is asserted. These three bits can be used as interrupt sources by setting the corresponding bits in the Mask register.

63.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 888.*I^{2}C slave registers

APB address offset	Register
0x00	Slave address register
0x04	Control register
0x08	Status register
0x0C	Mask register
0x10	Receive register
0x14	Transmit register

Table 889. Slave address register

31	30 ALEN	ALEN-1 0
TBA	RESERVED	SLVADDR

31	Ten-bit Address (TBA) - When this bit is set the core will interpret the value in the SLVADDR field as a 10-bit address. If the core has 10-bit address support this bit will have the reset value ‘1’.
30 : ALEN	RESERVED
(ALEN-1):0	Slave address (SLVADDR) - Contains the slave I2C address. The width of the slave address field, ALEN, is 7 bits (6:0) if the core only has support for 7-bit addresses. If the core has support for 10-bit addressing the width of SLVADDR is 10 bits. Depending on the hardware configuration this register may be read only. The core checks the length of the programmed address and will function with 7-bit addresses even if it has support for 10-bit addresses. I^{2}C addresses can be allocated by NXP, please see the link in the core’s overview section.

Reset value: Implementation dependent

Table 890. Control register

31 5	4	3	2	1	0
RESERVED	RMOD	TMOD	TV	TAV	EN

31 : 5	RESERVED
4	Receive Mode (RMOD) - Selects how the core handles writes: ‘0’: The slave accepts one byte and NAKs all other transfers until software has acknowledged the received byte by reading the Receive register. ‘1’: The slave accepts one byte and keeps SCL low until software has acknowledged the received byte by reading the Receive register.
3	Transmit Mode (TMOD) - Selects how the core handles reads: ‘0’: The slave transmits the same byte to all if the master requests more than one byte in the transfer. The slave then NAKs all read requests as long as the Transmit Valid (TV) bit is unset. ‘1’: The slave transmits one byte and then keeps SCL low until software has acknowledged that the byte has been transmitted by setting the Transmit Valid (TV) bit.
2	Transmit Valid (TV) - Software sets this bit to indicate that the data in the transmit register is valid. The core automatically resets this bit when the byte has been transmitted. When this bit is ‘0’ the core will either NAK or insert wait states on incoming read requests, depending on the Transmit Mode (TMOD).
1	Transmit Always Valid (TAV) - When this bit is set, the core will not clear the Transmit Valid (TV) bit when a byte has been transmitted.
0	Enable core (EN) - Enables core. When this bit is set to ‘1’ the core will react to requests to the address set in the Slave address register. If this bit is ‘0’ the core will keep both SCL and SDA inputs in Hi-Z state.

Reset value: 0b000000000000000000000000000UUUU0, where U is undefined.

Table 891. Status register

31 3	2	1	0
RESERVED	REC	TRA	NAK

31 : 3	RESERVED
2	Byte Received (REC) - This bit is set to ‘1’ when the core accepts a byte and is automatically cleared when the Receive register has been read.
1	Byte Transmitted (TRA) - This bit is set to ‘1’ when the core has transmitted a byte and is cleared by writing ‘1’ to this position. Writes of ‘0’ have no effect.
0	NAK Response (NAK) - This bit is set to ‘1’ when the core has responded with NAK to a read or write request. This bit does not get set to ‘1’ when the core responds with a NAK to an address that does not match the cores address. This bit is cleared by writing ‘1’ to this position, writes of ‘0’ have no effect.

Reset value: 0x00000000

Table 892. Mask register

31 … 3	2	1	0
RESERVED	RECE	TRAE	NAKE

31 : 3	RESERVED
2	Byte Received Enable (RECE) - When this bit is set the core will generate an interrupt when bit 2 in the Status register gets set.
1	Byte Transmitted Enable (TRAE) - When this bit is set the core will generate an interrupt when bit 1 in the Status register gets set.
0	NAK Response Enable (NAKE) - When this bit is set the core will generate an interrupt when bit 0 in the Status register gets set.

Reset value: Undefined

Table 893. Receive register

31 … 8	7 … 0
RESERVED	RECBYTE

31 : 8	RESERVED
7:0	Received Byte (RECBYTE) - Last byte received from master. This field only contains valid data if the Byte received (REC) bit in the status register has been set.

Reset value: Undefined

Table 894. Transmit register

31 … 8 … 8	7 … 0
RESERVED	TRABYTE

31 : 8	RESERVED
7:0	Transmit Byte (TRABYTE) - Byte to transmit on the next master read request.

Reset value: Undefined

63.4 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x03E. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

63.5 Configuration options

Table 895 shows the configuration options of the core (VHDL generics).

*Table 895.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by I^2C slave	0 - NAHBIRQ-1	0
hardaddr	If this generic is set to 1 the core uses the value of generic i2caddr as the hard coded address. If hardaddr is set to 0 the core’s address can be changed via the Slave address register.	0 - 1	0
tenbit	If this generic is set to 1 the core will support 10-bit addresses. Note that the core can still be configured to use a 7-bit address.	0 - 1	0
i2caddr	The slave’s (initial) I^2C address.	0 - 1023	0
oepol	Output enable polarity	0 - 1	0
filter	Low-pass filter length. This generic should specify, in number of system clock cycles plus one, the time of the shortest pulse on the I2C bus to be registered as a valid value. For instance, to disregard any pulse that is 50 ns or shorter in a system with a system frequency of 54 MHz this generic should be set to: ((pulse time) / (clock period)) + 1 = (50 ns) / ((1/(54 MHz)) + 1 = 3.7 The value from this calculation should always be rounded up. In other words an appropriate filter length for a 54 MHz system is 4.	2 - 512	2

63.6 Signal descriptions

Table 896 shows the interface signals of the core (VHDL ports).

*Table 896.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
I2CI	SCL	Input	I^2C clock line input	-
	SDA	Input	I^2C data line input	-
I2CO	SCL	Output	I^2C clock line output	-
	SCLOEN	Output	I^2C clock line output enable	Low**
	SDA	Output	I^2C data line output	-
	SDAOEN	Output	I^2C data line output enable	Low**
	ENABLE	Output	High when core is enabled, low otherwise	High

* see GRLIB IP Library User’s Manual
** Depends on OEPOL VHDL generic

63.7 Library dependencies

Table 897 shows the libraries used when instantiating the core (VHDL libraries).

*Table 897.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	I2C	Component, signals	Component declaration, I2C signal definitions

63.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity i2cslv_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- I2C signals
    iic_scl : inout std_ulogic;
    iic_sda : inout std_ulogic
    );
end;

architecture rtl of i2cslv_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- I2C signals
  signal i2ci : i2c_in_type;
  signal i2co : i2c_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- I2C-slave
  i2cslv0 : i2cslv
    generic map (pindex => 1, paddr => 1, pmask => 16#FFF#, pirq => 1,
                 hardaddr => 0, tenbit => 1, i2caddr => 16#50#)
    port map (rstn, clk, apbi, apbo(1), i2ci, i2co);
  i2cslv0_scl_pad : iopad generic map (tech => padtech)
    port map (iic_scl, i2co.scl, i2co.scloen, i2ci.scl);
  i2cslv0_sda_pad : iopad generic map (tech => padtech)
    port map (iic_sda, i2co.sda, i2co.sdaoen, i2ci.sda);
end;
```

64 IRQMP - Multiprocessor Interrupt Controller

64.1 Overview

The AMBA system in GRLIB provides an interrupt scheme where interrupt lines are routed together with the remaining AHB/APB bus signals, forming an interrupt bus. Interrupts from AHB and APB units are routed through the bus, combined together, and propagated back to all units. The multiprocessor interrupt controller core is attached to the AMBA bus as an APB slave, and monitors the combined interrupt signals.

The interrupts generated on the interrupt bus are all forwarded to the interrupt controller. The interrupt controller prioritizes, masks and propagates the interrupt with the highest priority to the processor. In multiprocessor systems, the interrupts are propagated to all processors.

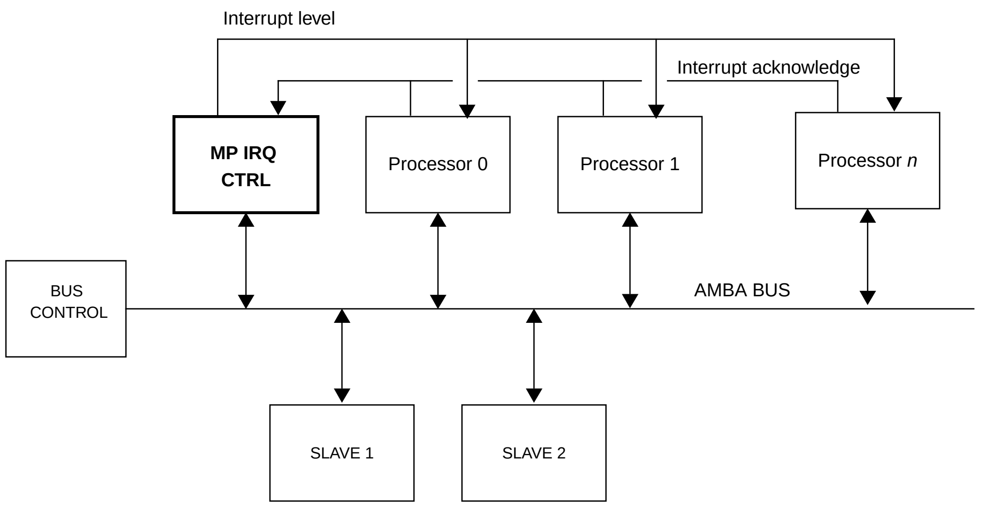


Figure 183. LEON multiprocessor system with Multiprocessor Interrupt controller

64.2 Operation

64.2.1 Interrupt prioritization

The interrupt controller monitors interrupt 1 - 15 of the interrupt bus (APBI.PIRQ[15:1]). When any of these lines are asserted high, the corresponding bit in the interrupt pending register is set. The pending bits will stay set even if the PIRQ line is de-asserted, until cleared by software or by an interrupt acknowledge from the processor.

Each interrupt can be assigned to one of two levels (0 or 1) as programmed in the interrupt level register. Level 1 has higher priority than level 0. The interrupts are prioritised within each level, with interrupt 15 having the highest priority and interrupt 1 the lowest. The highest interrupt from level 1 will be forwarded to the processor. If no unmasked pending interrupt exists on level 1, then the highest unmasked interrupt from level 0 will be forwarded. PIRQ[31:16] are not used by the IRQMP core.

Interrupts are prioritised at system level, while masking and forwarding of interrupts in done for each processor separately. Each processor in an multiprocessor system has separate interrupt mask and force registers. When an interrupt is signalled on the interrupt bus, the interrupt controller will prioritize interrupts, perform interrupt masking for each processor according to the mask in the corresponding mask register and forward the interrupts to the processors.

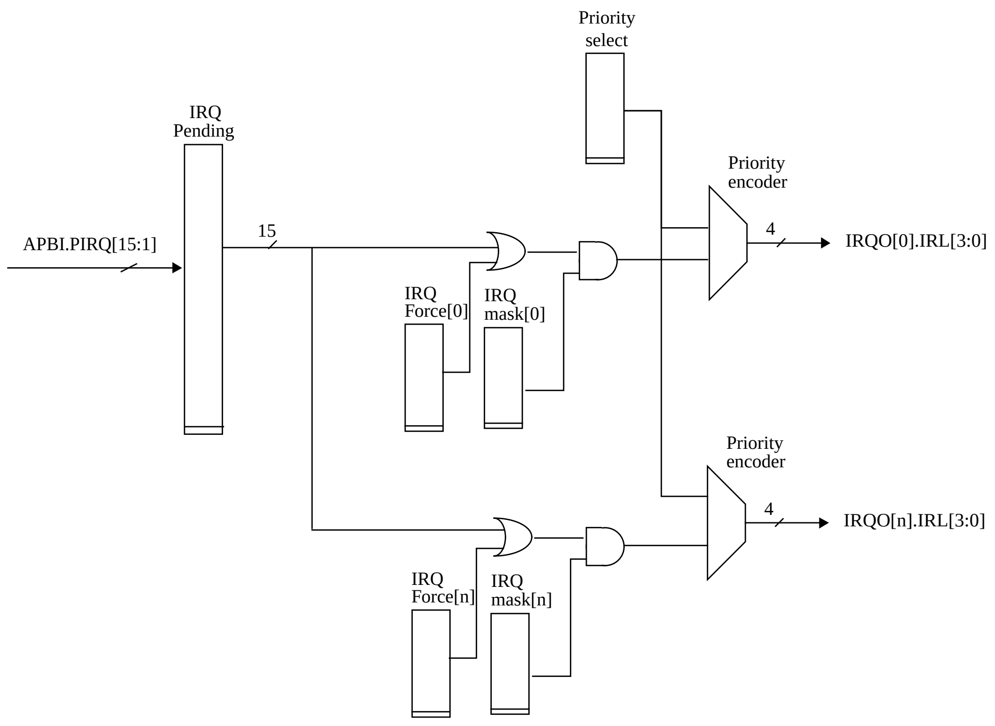


Figure 184. Interrupt controller block diagram

When a processor acknowledges the interrupt, the corresponding pending bit will automatically be cleared. Interrupt can also be forced by setting a bit in the interrupt force register. In this case, the processor acknowledgement will clear the force bit rather than the pending bit. After reset, the interrupt mask register is set to all zeros while the remaining control registers are undefined. Note that interrupt 15 cannot be maskable by the LEON processor and should be used with care - most operating systems do not safely handle this interrupt.

64.2.2 Extended interrupts

The AHB/APB interrupt consist of 32 signals ([31:0]), while the IRQMP only uses lines 1 - 15 in the nominal mode. To use the additional 16 interrupt lines (16-31), extended interrupt handling can be enabled by setting the VHDL generic *eirq* to a value between 1 - 15. The interrupt lines 16 - 31 will then also be handled by the interrupt controller, and the interrupt pending and mask registers will be extended to 32 bits. Since the processor only has 15 interrupt levels (1 - 15), the extended interrupts will generate one of the regular interrupts, indicated by the value of the *eirq* generic. When the interrupt is taken and acknowledged by the processor, the regular interrupt (*eirq*) and the extended interrupt pending bits are automatically cleared. The extended interrupt acknowledge register will identify which extended interrupt that was most recently acknowledged. This register can be used by software to invoke the appropriate interrupt handler for the extended interrupts.

64.2.3 Processor status monitoring

The processor status can be monitored through the Multiprocessor Status Register. The STATUS field in this register indicates if a processor is halted (‘1’) or running (‘0’). A halted processor can be reset and restarted by writing a ‘1’ to its status field. After reset, all processors except processor 0 are halted. When the system is properly initialized, processor 0 can start the remaining processors by writing to their STATUS bits.

64.2.4 Irq broadcasting

The Broadcast Register is activated when the generic *ncpu* is > 1. An incoming irq that has its bit set in the Broadcast Register is propagated to the force register of *all* CPUs rather than only to the Pending Register. This can be used to implement a timer that fires to all CPUs with that same irq.

64.3 Registers

The core is controlled through registers mapped into APB address space. The number of implemented registers depends on the number of processor in the multiprocessor system.

*Table 898.*Interrupt Controller registers

APB address offset	Register
0x00	Interrupt level register
0x04	Interrupt pending register
0x08	Interrupt force register (NCPU = 0)
0x0C	Interrupt clear register
0x10	Multiprocessor status register
0x14	Broadcast register
0x40	Processor interrupt mask register
0x44	Processor 1 interrupt mask register
0x40 + 4 * *n*	Processor *n* interrupt mask register
0x80	Processor interrupt force register
0x84	Processor 1 interrupt force register
0x80 + 4 * *n*	Processor *n* interrupt force register
0xC0	Processor extended interrupt acknowledge register
0xC4	Processor 1 extended interrupt acknowledge register
0xC0 + 4 * *n*	Processor *n* extended interrupt acknowledge register

Table 899. Interrupt Level Register

31 16	15 1	0
RESERVED	IL[15:1]	R

31:16	Reserved
15:1	Interrupt Level n (IL[n]) - Interrupt level for interrupt n
0	Reserved

Table 900. Interrupt Pending Register

31 16	15 1	0
EIP[31:16]	IP[15:1]	R

31:16	Extended Interrupt Pending n (EIP[n])
15:1	Interrupt Pending n (IP[n]) - Interrupt pending for interrupt n
0	Reserved

Table 901. Interrupt Force Register (NCPU = 0)

31 16	15 1	0
RESERVED	IF[15:1]	R

31:16	Reserved
15:1	Interrupt Force n (IF[n]) - Force interrupt nr n.
0	Reserved

Table 902. Interrupt Clear Register

31 16	15 1	0
EIC[31:16]	IC[15:1]	R

31:16	Extended Interrupt Clear n (EIC[n])
15:1	Interrupt Clear n (IC[n]) - Writing ‘1’ to IC[n] will clear interrupt n
0	Reserved

Table 903. Multiprocessor Status Register

31 28	27	26 20	19 16	15 0
NCPU	BA	RESERVED	EIRQ	STATUS[15:0]

31:28	Number of CPUs (NCPU) - Number of CPUs in the system - 1
27	Broadcast Available (BA) - Set to ‘1’ if NCPU > 0.
26:20	Reserved
19:16	Extended IRQ (EIRQ) - Interrupt number (1 - 15) used for extended interrupts. Fixed to 0 if extended interrupts are disabled.
15:0	Power-down status of CPU[n] (STATUS[n]) - ‘1’ = power-down, ‘0’ = running. Write STATUS[n] with ‘1’ to start processor n.

Table 904. Broadcast Register (NCPU > 0)

31 16	15 1	0
RESERVED	BM15:1]	R

31:16	Reserved
15:1	Broadcast Mask n (BM[n]) - If BM[n] = ‘1’ then interrupt n is broadcasted (written to the Force Register of all CPUs), otherwise standard semantic applies (Pending register)
0	Reserved

Table 905. Processor Interrupt Mask Register

31 16	15 1	0
EIM[31:16]	IM15:1]	R

31:16	Extended Interrupt Mask n (EIC[n]) - Interrupt mask for extended interrupts
15:1	Interrupt Mask n (IM[n]) - If IM[n] = ‘0’ then interrupt n is masked, otherwise it is enabled.
0	Reserved

Table 906. Processor Interrupt Force Register (NCPU > 0)

31 ... 17	16	15 ... 1	0
IFC[15:1]	R	IF15:1]	R

31:17	Interrupt Force Clear n (IFC[n]) - Interrupt force clear for interrupt n
16	Reserved
15:1	Interrupt Force n (IF[n]) - Force interrupt nr n
0	Reserved

Table 907. Extended Interrupt Acknowledge Register

31 ... 5	4 ... 0
RESERVED	EID[4:0]

31:5	Reserved
4:0	Extended interrupt ID (EID) - ID (16-31) of the most recent acknowledged extended interrupt. If this field is 0, and support for extended interrupts exist, the last assertion of interrupt *eirq* was not the result of an extended interrupt being asserted. If interrupt *eirq* is forced, or asserted, this field will be cleared unless one, or more, of the interrupts 31 - 16 are enabled and set in the pending register.

64.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x00D. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

64.5 Configuration options

Table 908 shows the configuration options of the core (VHDL generics).

*Table 908.*Configuration options

Generic	Function	Allowed range	Default
pindex	Selects which APB select signal (PSEL) that will be used to access the interrupt controller	0 to NAPBMAX-1	0
paddr	The 12-bit MSB APB address	0 to 4095	0
pmask	The APB address mask	0 to 4095	4095
ncpu	Number of processors in multiprocessor system	1 to 16	1
eirq	Enable extended interrupts	1 - 15	0

64.6 Signal descriptions

Table 909 shows the interface signals of the core (VHDL ports).

*Table 909.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
IRQI[n]	INTACK	Input	Processor *n* Interrupt acknowledge	High
	IRL[3:0]		Processor *n* interrupt level	High
	PWD		Unused	-
	FPEN		Unused	-
	IDLE		Unused	-
IRQO[n]	IRL[3:0]	Output	Processor *n* Input interrupt level	High
	RST		Reset power-down and error mode of processor *n*	High
	RUN		Start processor *n* after reset (SMP systems only)	High
	RSTVEC[31:12]		Always zero	-
	IACT		Always low	-
	INDEX[3:0]		CPU index	-
	HRDRST		Always low	-

* see GRLIB IP Library User's Manual

64.7 Library dependencies

Table 910 shows libraries that should be used when instantiating the core.

*Table 910.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	LEON3	Signals, component	Signals and component declaration

64.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon3.all;

entity irqmp_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ...  -- other signals
    );
end;
```

```
architecture rtl of irqmp_ex is
  constant NCPU : integer := 4;

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
  signal ahbsi : ahb_slv_in_type;

  -- GP Timer Unit input signals
  signal irqi   : irq_in_vector(0 to NCPU-1);
  signal irqo   : irq_out_vector(0 to NCPU-1);

  -- LEON3 signals
  signal leon3i : l3_in_vector(0 to NCPU-1);
  signal leon3o : l3_out_vector(0 to NCPU-1);

begin

  -- 4 LEON3 processors are instantiated here
  cpu : for i in 0 to NCPU-1 generate
    u0 : leon3s generic map (hindex => i)
    port map (clk, rstn, ahbmi, ahbmo(i), ahbsi,
irqi(i), irqo(i), dbgi(i), dbgo(i));
  end generate;

  -- MP IRQ controller
  irqctrl0 : irqmp
  generic map (pindex => 2, paddr => 2, ncpu => NCPU)
  port map (rstn, clk, apbi, apbo(2), irqi, irqo);
end
```

65 IRQ(A)MP - Multiprocessor Interrupt Controller with extended ASMP support

65.1 Overview

The AMBA system in GRLIB provides an interrupt scheme where interrupt lines are routed together with the remaining AHB/APB bus signals, forming an interrupt bus. Interrupts from AHB and APB units are routed through the bus, combined together, and propagated back to all units. The multiprocessor interrupt controller core is attached to the AMBA bus as an APB slave, and monitors the combined interrupt signals.

The interrupts generated on the interrupt bus are all forwarded to the interrupt controller. The interrupt controller prioritizes, masks and propagates the interrupt with the highest priority. The interrupt controller is configured at instantiation to implement one or several internal interrupt controllers. Each processor in a system can then be dynamically routed to one of the internal controllers. This allows safe Asymmetric Multiprocessing (ASMP) operation. For Symmetric Multiprocessor (SMP) operation, several processors can be routed to the same internal interrupt controller.

The IRQ(A)MP core is an extended version of the traditional multiprocessor interrupt controller. If a design does not need to have extended support for Asymmetric Multiprocessing, nor support for interrupt timestamping, it is recommended to use the IRQMP core instead.

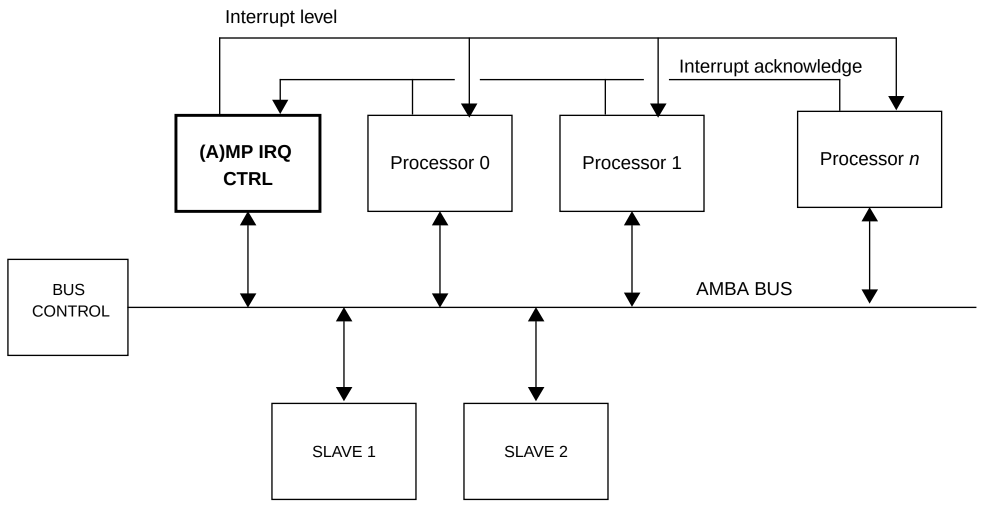


Figure 185. LEON multiprocessor system with Multiprocessor Interrupt controller

65.2 Operation

65.2.1 Support for Asymmetric Multiprocessing

Extended support for Asymmetric Multiprocessing (ASMP) is activated when the VHDL generic *nctrl* is > 1. Asymmetric Multiprocessing support means that parts of the interrupt controller are duplicated in order to provide safe ASMP operation. If the VHDL generic nctrl = 1 the core will have the same behavior as the normal IRQMP Multiprocessor interrupt controller core. If nctrl > 1, the core's register set will be duplicated on 4 KiB address boundaries. The core's register interface will also enable the use of three new registers, one Asymmetric Multiprocessing Control Register and two Interrupt Controller Select Registers.

Software can detect if the controller has been implemented with support for ASMP by reading the Asymmetric Multiprocessing Control register. If the field NCTRL is 0, the core was not implemented with ASMP extensions. If the value of NCTRL is non-zero, the core has NCTRL+1 sets of registers with additional underlying functionality. From a software view this is equivalent to having NCTRL interrupt controllers available and software can configure to which interrupt controller a processor should connect.

After system reset, all processors are connected to the first interrupt controller accessible at the core's base address. Software can then use the Interrupt Controller Select Registers to assign processors to other (internal) interrupt controllers. After assignments have been made, it is recommended to freeze the contents of the select registers by writing '1' to the lock bit in the Asymmetric Multiprocessing Control Register.

When a software driver for the interrupt controller is loaded, the driver should check the Asymmetric Multiprocessing Control Register and Interrupt Controller Select Registers to determine to which controller the current processor is connected. After software has determined that it has been assigned to controller n, software should only access the controller with registers at offset 0x1000 * n. Note that the controllers are enumerated with the first controller being n = 0.

The processor specific registers (mask, force, interrupt acknowledge) can be read from all interrupt controllers. However the processor specific mask and interrupt acknowledge registers can only be written from the interrupt controller to which the processor is assigned. This also applies to individual bits in the Multiprocessor Status Register. Interrupt Force bits in a processor's Interrupt Force Register can only be cleared through the controller to which the processor is assigned. If the ICF field in the Asymmetric Multiprocessing Control Register is set to '1', all bits in all Interrupt Force Registers can be set, but not cleared, from all controllers. If the ICF field is '0' the bits in a processor's Interrupt Force register can only be set from the controller to which the processor is assigned.

65.2.2 Interrupt prioritization

The interrupt controller monitors interrupt 1 - 15 of the interrupt bus (APBI.PIRQ[15:1]). When any of these lines are asserted high, the corresponding bit in the interrupt pending register is set. The pending bits will stay set even if the PIRQ line is de-asserted, until cleared by software or by an interrupt acknowledge from the processor.

Each interrupt can be assigned to one of two levels (0 or 1) as programmed in the interrupt level register. Level 1 has higher priority than level 0. The interrupts are prioritised within each level, with interrupt 15 having the highest priority and interrupt 1 the lowest. The highest interrupt from level 1 will be forwarded to the processor. If no unmasked pending interrupt exists on level 1, then the highest unmasked interrupt from level 0 will be forwarded. PIRQ[31:16] are not used by the IRQMP core.

Interrupts are prioritised at system level, while masking and forwarding of interrupts in done for each processor separately. Each processor in an multiprocessor system has separate interrupt mask and force registers. When an interrupt is signalled on the interrupt bus, the interrupt controller will prioritize interrupts, perform interrupt masking for each processor according to the mask in the corresponding mask register and forward the interrupts to the processors.

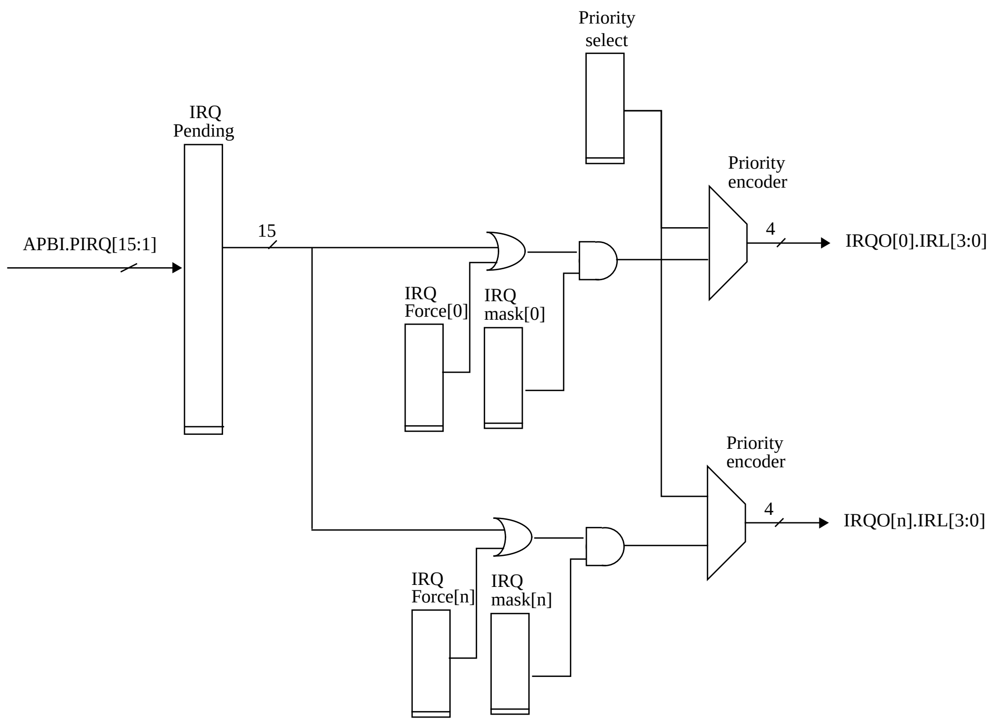


Figure 186. Interrupt controller block diagram

When a processor acknowledges the interrupt, the corresponding pending bit will automatically be cleared. Interrupt can also be forced by setting a bit in the interrupt force register. In this case, the processor acknowledgement will clear the force bit rather than the pending bit. After reset, the interrupt mask register is set to all zeros while the remaining control registers are undefined. Note that interrupt 15 cannot be maskable by the LEON processor and should be used with care - most operating systems do not safely handle this interrupt.

65.2.3 Extended interrupts

The AHB/APB interrupt consist of 32 signals ([31:0]), while the IRQMP only uses lines 1 - 15 in the nominal mode. To use the additional 16 interrupt lines (16-31), extended interrupt handling can be enabled by setting the VHDL generic *eirq* to a value between 1 - 15. The interrupt lines 16 - 31 will then also be handled by the interrupt controller, and the interrupt pending and mask registers will be extended to 32 bits. Since the processor only has 15 interrupt levels (1 - 15), the extended interrupts will generate one of the regular interrupts, indicated by the value of the *eirq* generic. When the interrupt is taken and acknowledged by the processor, the regular interrupt (*eirq*) and the extended interrupt pending bits are automatically cleared. The extended interrupt acknowledge register will identify which extended interrupt that was most recently acknowledged. This register can be used by software to invoke the appropriate interrupt handler for the extended interrupts.

65.2.4 Processor status monitoring

The processor status can be monitored through the Multiprocessor Status Register. The STATUS field in this register indicates if a processor is halted (‘1’) or running (‘0’). A halted processor can be reset and restarted by writing a ‘1’ to its status field. After reset, all processors except processor 0 are halted. When the system is properly initialized, processor 0 can start the remaining processors by writing to their STATUS bits.

The core can be implemented with support for specifying the processor reset start address dynamically. Please see section 65.2.9 for further information.

65.2.5 Irq broadcasting

The Broadcast Register is activated when the generic *ncpu* is > 1. An incoming irq that has its bit set in the Broadcast Register is propagated to the force register of *all* CPUs rather than only to the Pending Register. This can be used to implement a timer that fires to all CPUs with that same irq.

65.2.6 Interrupt timestamping description

Support for interrupt timestamping is implemented when the VHDL generic *tstamp* is > 0.

Interrupt timestamping is controlled via the Interrupt Timestamp Control register(s). Each Interrupt Timestamp Control register contains a field (TSTAMP) that contains the number of timestamp registers sets that the core implements. A timestamp register sets consist of one Interrupt Timestamp Counter register, one Interrupt Timestamp Control register, one Interrupt Assertion Timestamp register and one Interrupt Acknowledge Timestamp register.

Software enables timestamping for a specific interrupt via a Interrupt Timestamp Control Register. When the selected interrupt line is asserted, software will save the current value of the interrupt timestamp counter into the Interrupt Assertion Timestamp register and set the S1 field in the Interrupt Timestamp Control Register. When the processor acknowledges the interrupt, the S2 field of the Interrupt Timestamp Control register will be set and the current value of the timestamp counter will be saved in the Interrupt Acknowledge Timestamp Register. The difference between the Interrupt Assertion timestamp and the Interrupt Acknowledge timestamp is the number of system clock cycles that was required for the processor to react to the interrupt and divert execution to the trap handler.

The core can be configured to stamp only the first occurrence of an interrupt or to continuously stamp interrupts. The behavior is controlled via the Keep Stamp (KS) field in the Interrupt Timestamp Control Register. If KS is set, only the first assertion and acknowledge of an interrupt is stamped. Software must then clear the S1 and S2 fields for a new timestamp to be taken. If Keep Stamp is disabled (KS field not set), the controller will update the Interrupt Assertion Timestamp Register every time the selected interrupt line is asserted. In this case the controller will also automatically clear the S2 field and also update the Interrupt Acknowledge Timestamp register with the current value when the interrupt is acknowledged.

For controllers with extended ASMP support, each internal controller has a dedicated set of Interrupt timestamp registers. This means that the Interrupt Acknowledge Timestamp Register(s) on a specific controller will only be updated if and when the processor connected to the controller acknowledges the selected interrupt. The Interrupt Timestamp Counter is shared by all controllers and will be incremented when an Interrupt Timestamp Control register has the ITSEL field set to a non-zero value.

65.2.7 Interrupt timestamping usage guidelines

Note that KS = ‘0’ and a high interrupt rate may cause the Interrupt Assertion Timestamp register to be updated (and the S2 field reset) before the processor has acknowledged the first occurrence of the interrupt. When the processor then acknowledges the first occurrence, the Interrupt Acknowledge Timestamp register will be updated and the difference between the two Timestamp registers will not show how long it took the processor to react to the first interrupt request. If the interrupt frequency is expected to be high it is recommended to keep the first stamp (KS field set to ‘1’) in order to get reliable measurements. KS = ‘0’ should not be used in systems that include cores that use level interrupts, the timestamp logic will register each cycle that the interrupt line is asserted as an interrupt.

In order to measure the full interrupt handling latency in a system, software should also read the current value of the Interrupt Timestamp Counter when entering the interrupt handler. In the typical case, a software driver’s interrupt handler reads a status register and then determines the action to take. Adding a read of the timestamp counter before this status register read can give an accurate view of the latency during interrupt handling.

The core listens to the system interrupt vector when reacting to interrupt line assertions. This means that the Interrupt Assertion Timestamp Register(s) will not be updated if software writes directly to the pending or force registers. To measure the time required to serve a forced interrupt, read the value of the Interrupt Timestamp counter before forcing the interrupt and then read the Interrupt Acknowledge Timestamp and Interrupt Timestamp counter when the processor has reacted to the interrupt.

65.2.8 Watchdog

Support for watchdog inputs is implemented when the VHDL generic *wdogen* > 0, the number of watchdog input is determined by the VHDL generic *nwdog*.

The core can be implemented with support for asserting a bit in the controller’s Interrupt Pending Register when an external watchdog signal is asserted. This functionality can be used to implement a sort of soft watchdog for one or several processor cores. The controller’s Watchdog Control Register contains a field that shows the number of external watchdog inputs supported and fields for configuring which watchdog inputs that should be able to assert a bit in the Interrupt Pending Register. The pending register will be assigned in each cycle that a selected watchdog input is high. Therefore it is recommended that the watchdog inputs are connected to sources which send a one clock cycle long pulse when a watchdog expires. Otherwise software should make sure that the watchdog signal is deasserted before re-enabling interrupts during interrupt handling.

For controllers with extended ASMP support, each internal controller has a dedicated Watchdog Control register. Assertion of a watchdog input will only affect the pending register on the internal interrupt controllers that have enabled the watchdog input in their Watchdog Control Register.

65.2.9 Dynamic processor reset start address

Support for dynamically specifying the processor start address(es) is enabled when the VHDL generic *dynrstaddr* is non-zero (note that the processors must also be implemented to latch the start address from an external vector).

The core can be implemented with registers that are used to dynamically specify the reset start address for each CPU in the system. If implemented, the processor start address registers are available, one for each processor, starting at register offset 0x200. The reset value for all Processor Reset Start Address registers is specified at implementation time through the VHDL generic *rstaddr*. If software wishes to boot a processor from a different address, the processor’s start address register should be written (start address must be aligned on a 4 KiB address boundary) and the processor should then be enabled through the Processor boot register.

For controllers with extended ASMP support, the Processor Reset Start Address registers and boot register are visible and writable from the register space of all internal controllers.

65.3 Registers

The core is controlled through registers mapped into APB address space. The number of implemented registers depends on the number of processors in the multiprocessor system. The number of accessible register sets depend on the value of the NCTRL field in the Asymmetric Multiprocessing Control Register. The register set for controller n is accessed at offset 0x1000*n.

*Table 911.*Interrupt Controller registers

APB address offset	Register
0x000	Interrupt level register
0x004	Interrupt pending register
0x008	Interrupt force register (NCPU = 0)
0x00C	Interrupt clear register
0x010	Multiprocessor status register
0x014	Broadcast register
0x018	Reserved

*Table 911.*Interrupt Controller registers

APB address offset	**Register**
0x01C	Watchdog control register
0x020	Asymmetric multiprocessing control register
0x024	Interrupt controller select register for processor 0 - 7
0x028	Interrupt controller select register for processor 8 - 15
0x02C - 0x03C	Reserved
0x040	Processor interrupt mask register
0x044	Processor 1 interrupt mask register
0x040 + 0x4 * *n*	Processor *n* interrupt mask register
0x080	Processor interrupt force register
0x084	Processor 1 interrupt force register
0x080 + 0x4 * *n*	Processor *n* interrupt force register
0x0C0	Processor extended interrupt acknowledge register
0x0C4	Processor 1 extended interrupt acknowledge register
0x0C0 + 0x4 * *n*	Processor *n* extended interrupt acknowledge register
0x100	Interrupt timestamp counter register
0x104	Interrupt timestamp 0 control register
0x108	Interrupt assertion timestamp 0 register
0x10C	Interrupt acknowledge timestamp 0 register
0x100 + 0x10 * *n*	Interrupt timestamp counter register (mirrored in each set)
0x104 + 0x10 * *n*	Interrupt timestamp *n* control register
0x108 + 0x10 * *n*	Interrupt assertion timestamp *n* register
0x10C + 0x10 * *n*	Interrupt acknowledge timestamp *n* register
0x200 + 0x4* *n*	Processor n reset start address register
0x240	Processor boot register

Table 912. Interrupt Level Register

31 16	15 1	0
RESERVED	IL[15:1]	R

31:16	Reserved
15:1	Interrupt Level n (IL[n]) - Interrupt level for interrupt n
0	Reserved

Table 913. Interrupt Pending Register

31 16	15 1	0
EIP[31:16]	IP[15:1]	R

31:16	Extended Interrupt Pending n (EIP[n])
15:1	Interrupt Pending n (IP[n]) - Interrupt pending for interrupt n
0	Reserved

Table 914. Interrupt Force Register (NCPU = 0)

31 16	15 1	0
RESERVED	IF[15:1]	R

31:16	Reserved
15:1	Interrupt Force n (IF[n]) - Force interrupt nr n.
0	Reserved

Table 915. Interrupt Clear Register

31 16	15 1	0
EIC[31:16]	IC[15:1]	R

31:16	Extended Interrupt Clear n (EIC[n])
15:1	Interrupt Clear n (IC[n]) - Writing ‘1’ to IC[n] will clear interrupt n
0	Reserved

Table 916. Multiprocessor Status Register

31 28	27	26 20	19 16	15 0
NCPU	BA	RESERVED	EIRQ	STATUS[15:0]

31:28	Number of CPUs (NCPU) - Number of CPUs in the system - 1
27	Broadcast Available (BA) - Set to ‘1’ if NCPU > 0.
26:20	Reserved
19:16	Extended IRQ (EIRQ) - Interrupt number (1 - 15) used for extended interrupts. Fixed to 0 if extended interrupts are disabled.
15:0	Power-down status of CPU[n] (STATUS[n]) - ‘1’ = power-down, ‘0’ = running. Write STATUS[n] with ‘1’ to start processor n.

Table 917. Broadcast Register (NCPU > 0)

31 16	15 1	0
RESERVED	BM15:1]	R

31:16	Reserved
15:1	Broadcast Mask n (BM[n]) - If BM[n] = ‘1’ then interrupt n is broadcasted (written to the Force Register of all CPUs), otherwise standard semantic applies (Pending register)
0	Reserved

Table 918. Watchdog Control Register (NCPU > 0)

31 27	26 20	19 16	15 0
NWDOG	Reserved	WDOGIRQ	WDOGMSK

31:27	Number of watchdog inputs (NWDOG) - Number of watchdog inputs that the core supports.
26:20	Reserved
19:16	Watchdog interrupt (WDOGIRQ) - Selects the bit in the pending register to set when any line watchdog line selected by the WDOGMSK field is asserted.
15:0	Watchdog Mask n (WDOGMSK[n]) - If WDOGMSK[n] = ‘1’ then the assertion of watchdog input n will lead to the bit selected by the WDOGIRQ field being set in the controller’s Interrupt Pending Register.

Table 919. Asymmetric Multiprocessing Control Register

31 28	27 2	1	0
NCTRL	RESERVED	ICF	L

31:28	Number of internal controllers (NCTRL) - NCTRL + 1 is the number of internal interrupt controllers available.
27:2	Reserved
1	Inter-controller Force (ICF) - If this bit is set to ‘1’ all Interrupt Force Registers can be set from any internal controller. If this bit is ‘0’, a processor’s Interrupt Force Register can only be set from the controller to which the processor is connected. Bits in an Interrupt Force Register can only be cleared by the controller or by writing the Interrupt Force Clear field on the controller to which the processor is connected.
0	Lock (L) - If this bit is written to ‘1’, the contents of the Interrupt Controller Select registers is frozen. This bit can only be set if NCTRL > 0.

Table 920. Interrupt Controller Select Register for Processors 0 -7 (NCTRL > 0)

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
ICSEL0	ICSEL1	ICSEL2	ICSEL3	ICSEL4	ICSEL5	ICSEL6	ICSEL7

31:0	Interrupt controller select for processor n (ICSEL[n]) - The nibble ICSEL[n] selects the (internal) interrupt controller to connect to processor n.

Table 921. Interrupt Controller Select Register for Processors 8 - 15 (NCTRL > 0)

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
ICSEL8	ICSEL9	ICSEL10	ICSEL11	ICSEL12	ICSEL13	ICSEL14	ICSEL15

31:0	Interrupt controller select for processor n (ICSEL[n]) - The nibble ICSEL[n] selects the (internal) interrupt controller to connect to processor n.

Table 922. Processor Interrupt Mask Register

31 16	15 1	0
EIM[31:16]	IM15:1]	R

31:16	Extended Interrupt Mask n (EIC[n]) - Interrupt mask for extended interrupts
15:1	Interrupt Mask n (IM[n]) - If IM[n] = ‘0’ then interrupt n is masked, otherwise it is enabled.
0	Reserved

Table 923. Processor Interrupt Force Register (NCPU > 0)

31 17	16	15 1	0
IFC[15:1]	R	IF15:1]	R

31:17	Interrupt Force Clear n (IFC[n]) - Interrupt force clear for interrupt n
16	Reserved
15:1	Interrupt Force n (IF[n]) - Force interrupt nr n
0	Reserved

Table 924. Extended Interrupt Acknowledge Register

31 5	4 0
RESERVED	EID[4:0]

31:5	Reserved
4:0	Extended interrupt ID (EID) - ID (16-31) of the most recent acknowledged extended interrupt If this field is 0, and support for extended interrupts exist, the last assertion of interrupt *eirq* was not the result of an extended interrupt being asserted. If interrupt *eirq* is forced, or asserted, this field will be cleared unless one, or more, of the interrupts 31 - 16 are enabled and set in the pending register.

Table 925. Interrupt Timestamp Counter register(s)

31 0
TCNT

31:0	Timestamp Counter (TCNT) - Current value of timestamp counter. The counter increments whenever a TSISEL field in a Timestamp Control Register is non-zero. The counter will wrap to zero upon overflow and is read only.

Table 926. Timestamp n Control Register

31 27	26	25	24 6	5	4 0
TSTAMP	S1	S2	RESERVED	KS	TSISEL

31:27	Number of timestamp register sets (TSTAMP) - The number of available timestamp register sets.
26	Assertion Stamped (S1) - Set to ‘1’ when the assertion of the selected line has received a timestamp. This bit is cleared by writing ‘1’ to its position. Writes of ‘0’ have no effect.
25	Acknowledge Stamped (S2) - Set to ‘1’ when the processor acknowledge of the selected interrupt has received a timestamp. This bit can be cleared by writing ‘1’ to this position, writes of ‘0’ have no effect. This bit can also be cleared automatically by the core, see description of the KS field below.
24:6	RESERVED
5	Keep Stamp (KS) - If this bit is set to ‘1’ the core will keep the first stamp value for the first interrupt until the S1 and S2 fields are cleared by software. If this bit is set to ‘0’ the core will time stamp the most recent interrupt. This also has the effect that the core will automatically clear the S2 field whenever the selected interrupt line is asserted and thereby also stamp the next acknowledge of the interrupt.
4:0	Timestamp Interrupt Select (TSISEL) - This field selects the interrupt line (0 - 31) to timestamp when the EN field of this register has been set to ‘1’.

Table 927. Interrupt Assertion Timestamp register

31 0
TASSERTION

31:0	Timestamp of Assertion (TASSERTION) - The current Timestamp Counter value is saved in this register when timestamping is enabled and the interrupt line selected by TSISEL is asserted.

Table 928. Interrupt Acknowledge Timestamp register

31 0
TACKNOWLEDGE

31:0	Timestamp of Acknowledge (TACKNOWLEDGE) - The current Timestamp Counter value is saved in this register when timestamping is enabled, the Acknowledge Stamped (S2) field is ‘0’, and the interrupt selected by TSISEL is acknowledged by a processor connected to the interrupt controller.

Table 929. Processor n reset start address register

31 12	11 0
RSTADDR	RESERVED

31:12	Processor reset start address (RSTADDR) - If the core has been implemented to support dynamic assignment of the processor reset start address(es), then the Processor start address register at offset 0x200 + 4*n specifies the reset start address for processor n. Note that a processor must be reset before the new reset start address is valid. It is generally not possible to update the value in this register and then to correctly boot from the new address by only waking a processor via the Multiprocessor status register. Instead use the Processor boot register to boot or reset the processor.
11:0	RESERVED

Table 930. Processor boot register

31 16	15 0
RESET[n]	BOOT[n]

31:16	Processor reset (RESET): Writing bit *n* of this field to ‘1’ will reset, but not start, processor *n*. When the processor has been reset the bit will be reset to ‘0’. A processor can only be reset if it is currently idle (in power-down, error or debug mode), if a processor is running then the write to its bit in this field will be ignored. Multiple bits in this register may be set with one write but the register can only be written when all bits are zero.
15:0	Processor boot (BOOT): Writing bit *n* of this field to ‘1’ will reset and start processor *n*. When the processor has been booted the bit will be reset to ‘0’. A processor can only be reset if it is currently idle (in power-down, error or debug mode), if a processor is running then the write to its bit in this field will be ignored. Multiple bits in this register may be set with one write but the register can only be written when all bits are zero.

NOTE: This functionality is currently only supported when instantiating the LEON4FT entity. Contact Aeroflex Gaisler if you want to use the Processor boot register with LEON3.

65.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x00D (same as IRQMP core). For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

65.5 Configuration options

Table 931 shows the configuration options of the core (VHDL generics).

*Table 931.*Configuration options

Generic	Function	Allowed range	Default
pindex	Selects which APB select signal (PSEL) that will be used to access the interrupt controller	0 to NAPBSLV-1	0
paddr	The 12-bit MSB APB address	0 - 16#FFF#	0
pmask	The APB address mask. The mask determines the size of the memory area occupied by the core. The minimum required memory area based on the *nctrl* VHDL generic gives (nctrl : pmask) = 1 : 16#FFF#, 2: 16#FE0#, 3-4 : 16#FC0#, 5-8 : 16#F80#, 9-16 : 16#F00#. Note that even with nctrl = 1 the core may require a larger area than 256 bytes if the core has been implemented with support for timestamping and/or dynamic reset addresses.	0 - 16#FFF#	16#FFF#
ncpu	Number of processors in multiprocessor system	1 to 16	1
eirq	Enable extended interrupts	1 - 15	0
nctrl	Asymmetric multiprocessing system extension. This generic defines the number of internal interrupt controllers that will be implemented in the core.	1 - 16	1
tstamp	Interrupt timestamping. If this generic is non-zero the core will include a timestamp counter and *tstamp* set(s) of interrupt timestamp register(s).	0 - 16	0
wdogen	Enable watchdog inputs. If this generic is set to 1 the core will include logic to assert a selected interrupt when a watchdog input is asserted.	0 - 1	0
nwdog	Number of watchdog inputs	1 - 16	1
dynrstaddr	Enable dynamic reset start address assignments. If this generic is set to 1, the core will be implemented with registers that specify the reset address of each processor connected to the core. The processor(s) must also be implemented with support for dynamically assigning the reset start address.	0 - 1	0
rstaddr	Default reset start address for all Processor reset start address registers (if implemented)	0 - $(2^{20}-1)$	0
extrun	Use external run vector. If this generic is set to 1 the start of processors after reset will be controlled via the input signal cpurun. If this generic is set to 0, CPU 0 will be started after reset and the other CPUs will be put in power-down mode.	0 - 1	0

65.6 Signal descriptions

Table 932 shows the interface signals of the core (VHDL ports).

*Table 932.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
IRQI[n]	INTACK	Input	Processor *n* Interrupt acknowledge	High
	IRL[3:0]		Processor *n* interrupt level	High
	PWD		Unused	-
	FPEN		Unused	-
	IDLE		Processor idle	High
IRQO[n]	IRL[3:0]	Output	Processor *n* Input interrupt level	High
	RST		Reset power-down and error mode of processor *n*	High
	RUN		Start processor *n* after reset (SMP systems only)	High
	RSTVEC[31:12]		Reset start address for processor *n*	-
	IACT		Asserted when the IRL vector of processor 0 is non-zero.	High
	INDEX[3:0]		CPU index	-
	HRDRST		Processor reset	High
WDOG[]	N/A	Input	Watchdog input signals	High
NCPU[]	N/A	Input	If position n in this vector is set to ‘1’, processor n will be started after reset. Otherwise processor n will go into power-down. This signal is only used if VHDL generic extrun is /= 0.	High

* see GRLIB IP Library User’s Manual

65.7 Library dependencies

Table 933 shows libraries that should be used when instantiating the core.

*Table 933.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	LEON3	Signals, component	Signals and component declaration

65.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon3.all;

entity irqamp_ex is
```

```
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ...  -- other signals
    );
end;

architecture rtl of irqamp_ex is
  constant NCPU : integer := 4;

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
  signal ahbsi : ahb_slv_in_type;

  -- GP Timer Unit input signals
  signal irqi   : irq_in_vector(0 to NCPU-1);
  signal irqo   : irq_out_vector(0 to NCPU-1);

  -- LEON3 signals
  signal leon3i : l3_in_vector(0 to NCPU-1);
  signal leon3o : l3_out_vector(0 to NCPU-1);

begin

  -- 4 LEON3 processors are instantiated here
  cpu : for i in 0 to NCPU-1 generate
    u0 : leon3s generic map (hindex => i)
    port map (clk, rstn, ahbmi, ahbmo(i), ahbsi,
irqi(i), irqo(i), dbgi(i), dbgo(i));
  end generate;

  -- MP IRQ controller
  irqctrl0 : irqamp
  generic map (pindex => 2, paddr => 2, ncpu => NCPU, nctrl => NCPU)
  port map (rstn, clk, apbi, apbo(2), irqi, irqo);
end
```

66 L2C - Level 2 Cache controller

66.1 Overview

The L2C implements a Level-2 cache for processors with AHB interfaces. The L2C works as an AHB to AHB bridge, caching the data that is read or written via the bridge. A front-side AHB interface is connected to the processor bus, while a backend AHB interface is connected to the memory bus. Both front-side and backend buses can be individually configured to 32, 64 or 128 bits data width. The front-side bus and the backend bus must be clocked with the same clock. Figure 187 show a system block diagram for the cache controller.

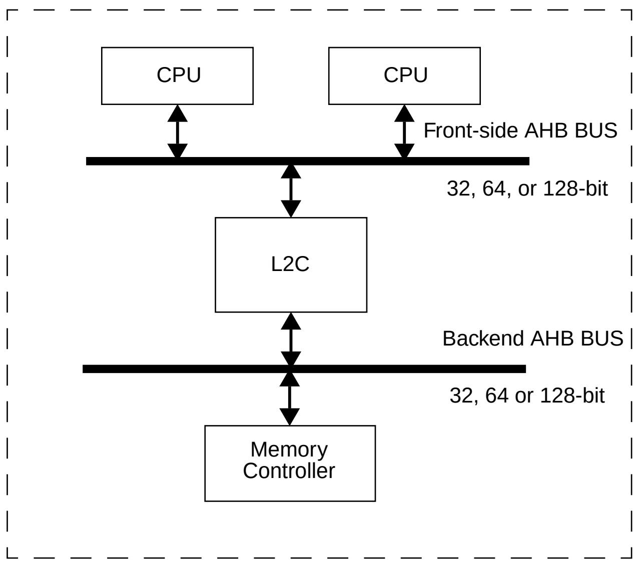


Figure 187. Block diagram

66.2 Configuration

The level-2 cache can be configured as direct-mapped or multi-way with associativity 2, 3 or 4. The replacement policy for a multi-way configuration can be configured as: LRU (least-recently-used), pseudo-random or master-index (where the way to replace is determine by the master index). The way size is configurable to 1 - 512 Kbyte with a line size of 32/64 bytes.

66.2.1 Replacement policy

The core can implements three different replacement policies: LRU (least-recently-used), (pseudo-) random and master-index. A VHDL generic REPL is used to define which replacement policy is configured as default. With the master-index replacement policy, master 0 would replace way 1, master 1 would replace way 2, and so on. With a master indexes corresponding to a way number larger then the number of implemented ways there is two options to determine which way to replace. One option is to map all these master index to a specific way. This is done by specify this way in the index-replace field in the control register and select this option in the replacement policy field also located in the control register. It is not allowed to select a locked way in the index-replace field. The second option is to replace way = ((master index) modulus (number of ways)). This option can be selected in the replacement policy field, but is only allowed with multi-way associativity 2 or 4.

66.2.2 Write policy

The cache can be configured to operate as write-through or copy-back cache. Before change the write policy to write-through, the cache has to be disabled and flushed (to write back dirty cache lines to memory. This can be done by setting the Cache disable bit when issue a flush all command). The write policy is controlled via the cache control register. A more fine-grained control can also be obtained by enabling the MTRR registers (see text below).

66.2.3 Memory type range registers

The memory type range registers (MTRR) are used to control the cache operation with respect to the address. Each MTRR can define an area in memory to be uncached, write-through or write-protected. The MTRR consist of a 14-bit address field, a 14-bit mask and two 2-bit control fields. The address field is compared to the 14 most significant bits of the cache address, masked by the mask field. If the unmasked bits are equal to the address, an MTRR hit is declared. The cache operation is then performed according to the control fields (see register descriptions). If no hit is declared or if the MTRR is disabled, cache operation takes place according to the cache control register. The number of MTRRs is configurable through the *mtrr* VHDL generic. When changing the value of any MTRR register, cache must be disabled and flushed (This can be done by setting the Cache disable bit when issue a flush all command).

Note that the write-protection provided via the MTRR registers is enforced even if the cache is disabled.

66.2.4 Cachability

The core uses a VHDL generic CACHED to determine which address range is cachable. Each bit in this 16-bit value defines the cachability of a 256 Mbyte address block on the AMBA AHB bus. A value of 16#00F3# will thus define cachable areas in 0 - 0x20000000 and 0x40000000 - 0x80000000. When the VHDL generic CACHED is 0, the cachable areas is defined by the plug&play information on the backend bus. The core can also be configured to use the HPROT signal to override the cachable area defined by VHDL generic CACHED. A access can only be redefined as non-cachable by the HPROT signal. See table 934 for information on how HPROT can change the access cachability within a cachable address area. The AMBA AHB signal HPROT[3] defines the access cacheable when active high and the AMBA AHB signal HPROT[2] defines the access bufferable when active high.

*Table 934.*Access cachability using HPROT.

HPROT:	**non-cachable, non-bufferable**	**non-cachable, bufferable**	**cacheable**
Read hit	Cache access*	Cache access	Cache access
Read miss	Memory access	Memory access	Cache allocation and Memory access
Write hit	Cache and Memory access	Cache access	Cache access
Write miss	Memory access	Memory access	Cache allocation

* When the HPROT-Read-Hit-Bypass bit is set in the cache control register this will generate a Memory access.

66.2.5 Cache tag entry

Table 935 show the different fields of the cache tag entry for a cache with set size equal to 1 kbyte. The number of bits implemented is depending on the cache configuration.

Table 935. L2C Cache tag entry

31 10	9 8	7 6	5	4 0
TAG	Valid	Dirty	RES	LRU

31 : 10	Address Tag (TAG) - Contains the address of the data held in the cache line.
9 : 8	Valid bits. When set, the corresponding sub-block of the cache line contains valid data. Valid bit 0 corresponds to the lower 16 bytes sub-block (with offset 1) in the cache line and valid bit 1 corresponds to the upper 16 bytes sub-block (with offset 0) in the cache line.
7 : 6	Dirty bits When set, this sub-block contains modified data.
5	RESERVED
4 : 0	LRU bits

66.2.6 AHB address mapping

The AHB slave interface occupies three AHB address ranges. The first AHB memory bar is used for memory/cache data access. The address and size of this bar is configured via VHDL generics. The second AHB memory bar is used for access to configuration registers and the diagnostic interface. This bar has a configurable address via VHDL generic but always occupy 4MB in the AHB address space. The last AHB memory bar is used to map the ioarea of the backend AHB bus (to access the plug&play information on that bus). The address and size of the this bar is configured via VHDL generics.

66.2.7 Memory protection and Error handling

The ft VHDL generic enables the implementation of the Error Detection And Correction (EDAC) protection for the data and tag memory. One error can be corrected and two error can be detected with the use of a (32, 7) BCH code. When implemented, the EDAC functionality can dynamically be enabled or disabled. Before being enabled the cache should be flushed. The dirty and valid bits fore each cache line is implemented with TMR. When EDAC error or backend AHB error or write-protection hit in a MTRR register is detected the error status register is updated to store the error type. The address which cause the error is also saved in the error address register. The error types is prioritised in the way that a uncorrected EDAC error will overwrite any other previously stored error in the error status register. In all other cases, the error status register has to be cleared before a new error can be stored. Each error type (correctable-, uncorrectable EDAC error, write-protection hit, backend AHB error) has a pending register bit. When set and this error is unmasked, a interrupt is generated. When uncorrectable error is detected in the read data the core will respond with a AHB error. AHB error response can also be enabled for a access whish match a stored error in the error status register. Error detection is done per cache line. The core also provide a correctable error counter accessible via the error status register.

*Table 936.*Cache action on detected EDAC error

Access/Error type	**Cache-line not dirty**	**Cache-line dirty**
Read, Correctable Tag error	Tag is corrected before read is handled, Error status is updated with a corretable error.	Tag is corrected before read is handled, Error status is updated with a corretable error.
Read, Uncorrectable Tag error	Cache-line invalidated before read is handled, Error status is updated with a corretable error.	Cache-line invalidated before read is handled, Error status is updated with a uncorrectable error. Cache data is lost.
Write, Correctable Tag error	Tag is corrected before write is handed, Error status is updated with a corretable error.	Tag is corrected before write is handled, Error status is updated with a corretable error.
Write, Uncorrectable Tag error	Cache-line invalidated before write is handled, Error status is updated with a correctable error.	Cache-line invalidated before write is handled, Error status is updated with a uncorrectable error. Cache data is lost.
Read, Correctable Data error	Cache-data is correted and updated, Error status is updated with a correctable error. AHB access is not affected.	Cache-data is correted and updated, Error status is updated with a correctable error. AHB access is not affected.
Read, Uncorrectable Data error	Cache-line is invalidated, Error status is updated with a correctable error. AHB access is terminated with retry.	Cache-line is invalidated, Error status is updated with a uncorrectable error. AHB access is terminated with error.
Write (<32-bit), Correctable Data error	Cache-data is correted and updated, Error status is updated with a correctable error. AHB access is not affected.	Cache-data is correted and updated, Error status is updated with a correctable error. AHB access is not affected.
Write (<32-bit), Uncorrectable Data error	Cache-line is re-fetched from memory, Error status is updated with a correctable error. AHB access is not affected.	Cache-line is invalidated, Error status is updated with a uncorrectable error. AHB access write data and cache data is lost.

66.2.8 Scrubber

When EDAC protection is implemented a cache scrubber is enabled. The scrubber is controlled via two register in the cache configuration interface. To scrub one specific cache line the index and way of the line is set in the scrub control register. To issue the scrub operation, the pending bit is set to 1. The scrubber can also be configured to continuously loop through and scrub each cache line by setting the enabled bit to 1. In this mode, the delay between the scrub operation on each cache line is determine by the scrub delay register (in clock cycles).

66.2.9 Locked way

One or more ways can be configured to be locked (not replaced). How many way that should be locked is configured by the locked-way field in the control register. The way to be locked is starting with the uppermost way (for a 4-way associative cache way 4 is the first locked way, way 3 the second, and so on). After a way is locked, this way has to be flushed with the “way flush” function to update the tag match the desired locked address. During this “way flush” operation, the data can also be fetched from memory.

66.3 Operation

66.3.1 Read

A cachable read access to the core result in a tag lookup to determine if the requested data is located in the cache memory. For a hit (requested data is in the cache) the data is read from the cache and no read access is issued to the memory. If the requested data is not in the cache (cache miss), the cache controller issue a read access to the memory controller to fetch the cache line including the requested data. The replacement policy determine which cache line in a multi-way configuration that should be replaced and its tag is updated. If the replaced cache line is modified (dirty) this data is stored in a write buffer and after the requested data is fetched from memory the replaced cache line is written to memory.

For a non-cachable read access to the core, the cache controller issue an single read access of the same size to the memory. The data is stored in a read buffer and the state of the cache is not modified in any way. When HPROT support is enabled, a bufferable (but non-cachable) read burst access will prefetch data up to the cache line boundary from memory.

66.3.2 Write

A cachable write access to the core result in a tag lookup to determine if the cache line is find in the cache. For a hit the cache line is updated. No access is issued to the memory for a copy-back configuration. When the core is configured as a write-through cache, each write access is also issued towards the memory. For a miss, the replacement policy determine which cache line i a multi-way configuration that should be replaced and updates its tag. If the replaced cache line is dirty, this is stored in a write buffer to be written back to the memory. The new cache line is updated with the data from the write access and for a non 128-bit access the rest of the cache line is fetched from memory. Last (when copy-back policy is used and the replaced cache line was marked dirty) the replaced cache line is written to memory. When the core is configured as a write-through cache, no cache lines are marked as dirty and no cache line needs to be written back to memory. Instead the write access is issued towards the memory as well. A new cache line is allocated on a miss for a cacheable write access independent of write policy (copy-back or write-through).

For a non-cachable write access to the core, the data is stored in a write buffer and the cache controller issue single write accesses to write the data to memory. The state of the cache is unmodified during this access.

66.3.3 Cache flushing

The cache can be flushed by accessing a cache flush register. There is three flushing modes: invalidate (reset valid bits), write back (write back dirty cache lines to memory, but no invalidation of the cache content) and flush (write back dirty cache lines to memory and invalidate the cache line). The flush command can be applied to the entire cache, one way or to only one cache line. The cache line to be flushed can be addresses in two ways: direct address (specify way and line address) and memory address (specify which memory address that should be flushed in the cache. The controller will make a cache lookup for the specified address and on a hit, flush that cache line). When the entire cache is flushed the Memory Address field should be set to zero. Invalidate a cache line takes 3 clock cycles. If the cache line needs to be written back to memory one additional clock cycle is needed plus the memory write latency. When the whole cache is flushed the invalidation of the first cache line takes 3 clock cycles, after this one line can be invalidate each clock cycle. When a cache line needs to be written back to memory this memory access will be stored in a access buffer. If the buffer is full the invalidation of the next cache line is stall until a slot in the buffer has opened up. If the cache also should be disabled after the flush is complete, it is recommended to set the cache disable bit together with the flush command instead of writing ‘0’ to the cache enable bit in the cache control register.

Note that after a processor (or any other AHB master) has initiated a flush the processor is not blocked by the flush unless it writes or requests data from the Level-2 cache. The cache blocks all accesses (responds with AMBA RETRY) until the flush is complete.

66.3.4 Disabling Cache

To be able to safely disable the cache when it is being accessed, the cache need to be disabled and flushed at the same time. This is accomplished by setting the cache disable bit when issue the flush command.

66.3.5 Diagnostic cache access

The diagnostic interface can be used for RAM block testing and direct access to the cache tag, cache data content and EDAC check bits. The read-check-bits filed in the error control register selects if data content or the EDAC check bits should be read out. On writes, the EDAC check bits can be selected from the data-check-bit or tag-check-bit register. These register can also be XOR:ed with the correct check bits on a write. See the error control register for how this is done.

66.3.6 Error injection

Except using the diagnostic interface, the EDAC check bits can also be manipulated on a regular cache access. By setting the xor-check-bit field in the error control register the data EDAC check bits will be XOR:ed with the data-check-bit register on the next write or the tag EDAC check bits will be XOR:ed with the tag-check-bit register on the next tag replacement. The tag check bit manipulation is only done if the tag-check-bit register is not zero. The xor-check-bit is reset on the next tag replacement or data write. Error can also be injected by writing a address together with the inject bit to the “Error injection” register. This will XOR the check-bits for the specified address with the data-check-bit register. If the specified address in not cached, the cache content will be unchanged.

66.3.7 AHB slave interface

The slave interface is the core’s connection to the CPU and the level 1 cache. The core can accept 8-bit(byte), 16-bit(half word), 32-bit(word), 64-bit, and 128-bit single accesses and also 32-bit, 64-bit, and 128-bit burst accesses. For an access during a flush operation, the core will respond with a AHB RETRY response. For a uncorrectable error or a backend AHB error on a read access, the core will respond with a AHB error.

66.3.8 AHB master interface

The master interface is the core’s connection to the memory controller. During cache line fetch, the controller can issue either a 32-bit, 64-bit or 128-bit burst access. For a non cachable access and in write-through mode the core can also issue a 8-bit(byte), 16-bit(half word), 32-bit(word), 64-bit, or 128-bit single write access. The bbuswidth VHDL generic controls the maximum bus access size on the master interface in the “wide-bus” address range. If set to 128 (default), the largest access will be 128-bit. If set to 64, the largest access will be 64-bit. If set to 32, the largest access will be 32-bit. The “wide-bus address range is defined by the wbmask VHDL generic. Each bit in this 16-bit value represents a 256 Mbyte address block on the AMBA AHB bus. The cache will only generate wide accesses (> 32-bit) to address ranges which wbmask bit is ‘1’. For address ranges which wbmask bit is ‘0’, wide accesses will be translated to 32-bit bursts.

The HBURST value during burst accesses will correspond to SINGLE, INCR, INCR4, INCR8 or INCR16, depending on burst type and AHB data bus width.

66.3.9 Latency

Table 937 defines the minimum latency for different access sequences. The latency is defined as core latency + memory latency (The memory latency in this table is for the synchronous version of the ddr2spa DDR controller with 64-bit DDR and 128-bit AHB bus).

*Table 937.*Access latency

	Previous Access		128-Bit						Non 128-Bit					
			Read			Write			Read			Write		
	Current Access	**None**	**Hit**	**Miss**	**Dirty miss**	**Hit**	**Miss**	**Dirty miss**	**Hit**	**Miss**	**Dirty miss**	**Hit**	**Miss**	**Dirty miss**
128- Bit	**Read hit**	3	3	3	3	4	4	6	3	3	3	6	6+6	6+6
	Read miss	5+7	5+7	5+7	5+11	6+7	6+7	9+13	6+6	6+6	6+11	7+6	9+12	11+12
	Read dirty miss	5+8	5+8	5+7	5+12	6+10	6+7	9+11	6+6	6+6	6+11	7+7	9+12	12+19
	Write hit	0	0	0	0	0	1	4+1	0	0	1	0	4+6	6+8
	Write miss	0	0	0	0	0	1	4+3	0	0	1	0	4+6	7+10
	Write dirty miss	0	0	0	0	0	1	5+2	0	0	1	0	4+10	7+6
Non 128-Bit	**Read hit**	3	3	3	3	4	4	6	3	3	4	4	5+7	8+7
	Read miss	5+8	6+6	6+6	6+11	7+6	7+6	9+21	5+7	5+7	6+11	6+7	7+15	9+24
	Read dirty miss	5+9	6+6	6+6	6+11	7+8	7+6	9+10	5+9	5+7	6+12	6+11	7+14	10+21
	Write hit	0	0	0	2	0	1	5	0	0	0	0	4+6	6+7
	Write miss	0	0	0	2	0	1	5+1	0	0	0	0	4+6	6+11
	Write dirty miss	0	0	0	1	0	1	5	0	0	0	0	4+11	6+7

+ Additional memory latency

The latency for a access that bypass the cache (the cache is disabled; the address is non-cachable; the HPROT signal defines the access not-cachable and not-bufferable) is the same as for a cache miss. Because the cache controller only issue single accesses towards the memory in this mode, a burst access will suffer this latency for each beat in the burst. If the access is bufferable and HRPOT support is enabled, the controller will prefetch data up to the cache line boundary from memory which then can be read out with no additional latency except for the first word. When EDAC is enabled, one additional latency cycle is added to read access returning cache data. For definition of the HPROT support (cachable and bufferable) see section 66.2.4.

66.3.10 Cache status

The cache controller has a status register which provide information on the cache configuration (multi-way configuration and set size). The core also provides an access counter and a hit counter via AHB mapped registers. These register can be used to calculate hit rate. The counters increments for each data access to core (i.e. a burst access is only counted as one access). When writing 0 to the access counter, the internal access/hit counters is cleared and its value is loaded to the registers accessible via the AHB interface. In wrapping mode both counters will be cleared when the access counter is wrapping at 0xFFFFFFFF. In shifting mode both counters will be shifted down 16 bits when the access counter reach 0xFFFFFFFF. In this mode the accessible counter registers is updated automatically when the access counter's 16 LSB reach the value of 0xFFFF.

The core can also implement a front-side bus usage counter. This counter records every clock cycle the bus is not in idle state. The registers accessible via the AHB interface is updated in the same way as for the hit counter registers. Writing 0 to the bus cycle counter register resets the bus usage counters. This counter also has a wrapping and shifting mode similar to the hit counter.

In addition to the counter registers, the core also provide output signals for: cache hit, cache miss, and cache access. These signals can be connected to external statistic counters.

66.4 Registers

The core is configured via registers mapped into the AHB memory address space.

*Table 938.*L2C: AHB registers

AHB address offset	**Register**
0x00	Control register
0x04	Status register
0x08	Flush (Memory address)
0x0C	Flush (set, index)
0x10	Access counter
0x14	Hit counter
0x18	Bus cycle counter
0x1C	Bus usage counter
0x20	Error status/control
0x24	Error address
0x28	TAG-check-bit
0x2C	Data-check-bit
0x30	Scrub Control/Status
0x34	Scrub Delay
0x38	Error injection
0x80 - 0xFC	MTRR registers
0x80000 - 0x8FFFC	Diagnostic interface (Tag) 0x80000: Tag 1, way-1 0x80004: Tag 1, way-2 0x80008: Tag 1, way-3 0x8000C: Tag 1, way-4 0x80010: Tag check-bits way-0,1,2,3 (Read only) bit[27:21] = check-bits for way-1. bit[20:14] = check-bits for way-2. bit[13:7] = check-bits for way-3. bit[6:0] = check-bits for way-4. 0x80020: Tag 2, way-1 0x80024: ...
0x200000 - 0x3FFFFC	Diagnostic interface (Data) 0x200000 - 0x27FFFC: Data or check-bits way-1 0x280000 - 0x2FFFFF: Data or check-bits way-2 0x300000 - 0x27FFFC: Data or check-bits way-3 0x380000 - 0x3FFFFF: Data or check-bits way-4 When check-bits are read out: Only 32-word at offset 0x0, 0x10, 0x20,... are valid check-bits. bit[27:21] = check-bits for data word at offset 0x0. bit[20:14] = check-bits for data word at offset 0x4. bit[13:7] = check-bits for data word at offset 0x8. bit[6:0] = check-bits for data word at offset 0xc.

Table 939. L2C Control register (address offset 0x00)

31	30	29 28	27 16	15 12	11 8	7 6	5	4	3	2	1	0
EN	EDAC	REPL	RES	INDEX-WAY	LOCK	RES	HPRHB	HPB	UC	HC	WP	HP

31	Cache enable. When set, the cache controller is enabled. When disabled, the cache is bypassed. Default disabled. Reset value is set by the cen VHDL generic, default value 0.

Table 939. L2C Control register (address offset 0x00)

30	EDAC enable
29 : 28	Replacement policy: (Reset value is set by the repl VHDL generic, default value 00) 00: LRU 01: (pseudo-) random 10: Master-index using index-replace field 11: Master-index using the modulus function
27 :16	RESERVED
15 : 12	Way to replace when Master-index replacement policy and master index is larger then number of ways in the cache (default value 0)
11 : 8	Number of locked ways (default value 0)
7 : 6	RESERVED
5	When set, a non-cacheable and non-bufferable read access will bypass the cache on a cache hit and return data from memory. Only used with HPROT support. (default value 0)
4	When HPROT is used to determine cachability and this bit is set, all accesses is marked bufferable. (default value 0)
3	Bus usage status mode. 0 = wrapping mode, 1 = shifting mode. Default value 0.
2	Hit rate status mode. 0 = wrapping mode, 1 = shifting mode. Default value 0.
1	Write policy. When set, the cache controller uses the write-through write policy. When not set, the write policy is copy-back. Default copy-back Reset value is set by the wp VHDL generic, default value 0.
0	When set, use HPROT to determine cachability. Reset value is set by the hprot VHDL generic, default value 0.

Table 940. L2C Status register (Read only) (address offset 0x04)

31 25	24	23	22	21 16	15 13	12 2	1 0
RESERVED	LSIZE	FTTIME	EDAC	MTRR	BBus width	Cache set size	Way

31 :25	RESERVED
24	Cache line size. 1 = 64 bytes, 0 = 32 bytes.
23	Access timing is simulated as if memory protection is implemented. (Read only)
22	Memory protection implemented (Read only)
21 : 16	Number of MTRR registers implemented (0 - 32) (Read only)
15 : 13	Backend bus width: 1 = 128-bit, 2 = 64-bit, 4 = 32-bit. (Read only)
12 : 2	Cache Set size configuration in kBytes (Read only)
1 : 0	Multi-Way configuration (Read only) “00“: Direct mapped “01“: 2-way “10“: 3-way “11“: 4-way

Table 941. L2C Flush (Memory address) register (address offset 0x08)

31 5	4	3	2 0
Memory Address	RES	DI	Flush

31 : 5	Memory Address (For flush all cache lines, this field should be set to zero)
4	RESERVED

Table 941. L2C Flush (Memory address) register (address offset 0x08)

3	Cache disable. Setting this bit to ‘1’ is equal to setting the Cache enable bit to ‘0’ in the Cache Control register.
2 : 0	Flush mode: “001“: Invalidate one line, “010”: Write-back one line, “011“: Invalidate & Write-back one line. “101“: Invalidate all lines, “110”: Write-back all lines, “111“: Invalidate & Write-back all lines. Only dirty cache lines are written back to memory.

Table 942. L2C Flush (set, index) register (address offset 0x0C)

31	16	..	10	9	8	7	6	5	4	3	2	1	0
Cache line index / TAG				Fetch	Valid	Dirty	RES	Way		DI	WF	Flush	

31 : 16	Cache line index, used when a specific cache line is flushed
31 : 10	TAG used when “way flush” is issued. If a specific cache line is flushed, bit 15 : 10 should be set to zero
9	If set to ‘1’ data is fetched form memory when a “way flush” is issued. If a specific cache line is flushed, this bit should be set to zero
8	Valid bit used when “way flush” is issued. If a specific cache line is flushed, this bit should be set to zero
7	Dirty bit used when “way flush” is issued. If a specific cache line is flushed, this bit should be set to zero
6	RESERVED
5 : 4	Cache way
3	Cache disable. Setting this bit to ‘1’ is equal to setting the Cache enable bit to ‘0’ in the Cache Control register.
2	Issue a Way-flush, If a specific cache line should be flushed, this bit should be set to zero
1 : 0	Flush mode (line flush): “01“: Invalidate one line “10”: Write-back one line (if line is dirty) “11“: Invalidate & Write-back one line (if line is dirty). Flush mode (way flush): “01“: Update Valid/Dirty bits according to register bit[8:7] “10”: Write-back dirty lines to memory “11“: Update Valid/Dirty bits according to register bit[8:7] & Write-back dirty lines to memory.

Table 943. L2C Access counter register (address offset 0x10)

31	0
Access counter	

31 : 0	Access counter. Write 0 to clear internal access/hit counter and update access/hit counter register.

Table 944. L2C Hit counter register (address offset 0x14)

31	0
Hit counter	

31 : 0	Hit counter.

Table 945. L2C Front-side bus cycle counter register (address offset 0x18)

31 : 0
Bus cycle counter

31 : 0 Bus cycle counter. Write 0 to clear internal bus cycle/usage counter and update bus cycle/usage counter register.

Table 946. L2C Front-side bus usage counter register (address offset 0x1C)

31 : 0
Bus usage counter

31 : 0 Bus usage counter.

Table 947. L2C Error status/control (address offset 0x20)

31 : 28	27	26 : 24	23	22	21	20	19	18 : 16	15 : 12	11 : 8	7 : 6	5 : 4	3	2	1	0
AHB master index	SCRUB	TYPE	TAG/DATA	COR/UCOR	MULTI	VALID	DISERESP	Correctable error counter	IRQ pending	IRQ mask	Select CB	Select TCB	XCB	RCB	COMP	RST

31: 28	AHB master that generated the access
27	Indicates that the error was trigged by the scrubber.
26: 24	Access/Error Type: (Read only) 000: cache read, 001: cache write, 010: memory fetch, 011: memory write, 100: Write-protection hit, 101: backend read AHB error, 110: backend write AHB error
23	0 tag error, 1: data error (Read only)
22	0: correctable error, 1: uncorrectable error (Read only)
21	Multiple error has occurred (Read only)
20	Error status register contains valid error (Read only)
19	Disable error responses for uncorrectable EDAC error.
18: 16	Correctable EDAC error counter (read only)
15: 12	Interrupt pending (read only) bit3: Backend AHB error bit2: Write-protection hit bit1: Uncorrectable EDAC error bit0: Correctable EDAC error
11: 8	Interrupt mask (if set this interrupt is unmasked) bit3: Backend AHB error bit2: Write-protection hit bit1: Uncorrectable EDAC error bit0: Correctable EDAC error
7: 6	Selects the data-check-bits for diagnostic data write: 00: use generated check-bits 01: use check-bits in the data-check-bit register 10: XOR check-bits with the data-check-bit register 11: use generated check-bits

Table 947. L2C Error status/control (address offset 0x20)

4: 5	Selects the tag-check-bits for diagnostic tag write: 00: use generated check-bits 01: use check-bits in the tag-check-bit register 10: XOR check-bits with the tag-check-bit register 11: use generated check-bits
3	If set, the check-bits for the next data write or tag replace will be XOR:ed withe the check-bit register. Default value is 0.
2	If set, a diagnostic read to the cache data area will return the check-bits related to that data Default value is 0.
1	If set, a read access matching a uncorrectable error stored in the error status register will generate a AHB error response. Default value is 0.
0	Resets the status register to be able to store a new error

Table 948. L2C Error address register (address offset 0x24)

31 : 0
Error address

31 : 0	Error address

Table 949. L2C Tag-check-bit register (address offset 0x28)

31 : 7	6 : 0
RESERVED	TCB

31 : 7	RESERVED
6 : 0	Check-bits which can be selected by the “Select check-bit“ field in the error status/control register for TAG updates

Table 950. L2C Data-check-bit register (address offset 0x2C)

31 : 28	27 : 0
RESERVED	DCB

31 : 28	RESERVED
27 : 0	Check-bits which can be selected by the “Select tag-check-bit“ field in the error status/control register for TAG updates

Table 951. L2C Scrub control/status register (address offset 0x30)

31 : 16	15 : 4	3 : 2	1	0
INDEX	RESERVED	Way	PEN	EN

31 :16	Index for the next line scrub operation
15 : 4	RESERVED
3:2	Way for the next line scrub operation
1	Indicates when a line scrub operation is pending. When the scrubber is disabled, writing ‘1’ to this bit scrubs one line.
0	Enables / disables the automatic scrub functionality.

Table 952. L2C Scrub delay register (address offset 0x34)

31 16	15 0
RESERVED	Delay

31 :16	RESERVED
15 : 0	Delay the scrubber waits before issue the next line scrub operation

Table 953. L2C Error injection register (address offset 0x38)

31 2	1	0
Address	RES	INJECT

31 :2	Address to inject error at.
1	RESERVED
0	Set to ‘1’ to inject a error at “address”.

Table 954. L2C Memory type range register (address offset 0x80-0xFC)

31 18	17 16	15 2	1 0
Address field	ACC	Address mask	CTRL

31 : 18	Address field to be compared to the cache address [31:18]
17 : 16	Access field. 00: uncached, 01: write-through
15 : 2	Address mask. Only bits set to 1 will be used during address comparison
1	Write-protection. 0: disabled, 1: enabled
0	Access control field. 0: disabled, 1: enabled

66.5 Configuration options

Table 955 shows the configuration options of the core (VHDL generics).

*Table 955.*Configuration options

Generic name	Function	Allowed range	Default
memtech	The memory technology used for the internal FIFOs.	0 - NTECH	0
hmstndx	AHB master index.	0 - NAHBMST-1	0
hslvndx	AHB slave index.	0 - NAHBSLV-1	0
haddr	ADDR filed of the AHB BAR (for data access).	0 - 16#FFF#	16#F00#
hmask	MASK filed of the AHB BAR.	0 - 16#FFF#	16#F00#
ioaddr	ADDR filed of the AHB BAR (for register and diagnostic access).	0 - 16#FFF#	16#F00#
cached	Fixed cachability mask.	0 - 16#FFFF#	16#0000#
hirq	Interrupt line used by the core.	0 - NAHBIRQ-1	0
cen	Reset value for cache enable. 1 = cache enabled.	0 - 1	0
hproten	Reset value for enabling hprot functionality	0 - 1	0
wp	Reset value for write-policy: 0 = copy-back, 1 = write-through	0 - 1	0
repl	Reset value for replacement policy: 0 = LRU, 1 = pseudo-random	0 - 1	0
ways	Number of cache ways	1 - 4	1
waysize	Size of each cache way in kBytes	1 - 512	1
linesize	Cache line size in bytes	32, 64	32
bbuswidth	Maximum bus width on master AHB interface	32, 64, 128	128
bioaddr	ADDR filed of the AHB BAR (for backend ioarea). Appears in the bridge's slave interface user-defined register 1.	0 - 16#FFF#	0
biomask	MASK filed of the AHB BAR.	0 - 16#FFF#	0
sbus	The number of the AHB bus to which the slave interface is connected. The value appears in bits [1:0] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	0
mbus	The number of the AHB bus to which the master interface is connected. The value appears in bits [3:2] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	1
stat	Enables the statistics counters. 0 all counters is disabled. 1 enables the access/hit counter. 2 enables the bus usage counter in addition to the access/hit counter.	0-2	0
arch	Selects between separate (0) or shared (1) RAM in multi-way configurations (see text below)	0 - 1	0
mtrr	Number of MTRR registers	0 - 32	0
edacen	Default value for the EDACEN field in the cache control register	0 - 1	0
rmw	Enables Read-Modify-Write for sub-word writes.	0 - 1	0
ft	Enables the memory protection (EDAC) implementation	0 - 1	0
fttiming	Simulate access timing as if memory protection was enabled. (Only for prototype testing)	0 - 1	0
wbmask	Wide-bus mask. Each bit in this value represent a 256Mbyte address range. To enabled wide accesses (>32-bit) to an address range, set the corresponding bit to '1'.	0 - 16#FFFF#	16#FFFF#

66.6 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x04B. For description of vendor and device identifier see GRLIB IP Library User's Manual

66.7 RAM usage

The L2C uses single-port RAM to implement both cache tags and data memory. The tags are implemented using the SYNCRAM core, with the width and depth depending on the cache size configuration. The data memory is implemented using the SYNCRAM_128BW or SYNCRAM_156BW core, which is a 128-bit or 156-bit wide RAM wrapper with byte enables. The SYNCRAM_156BW is used when memory protection (EDAC) is implemented. For multi-way caches, each way's tag is implemented with a separate SYNCRAM block. The data memory can be implemented with separate SYNCRAM_128BW/156BW cores, or merged into the same SYNCRAM_128BW/156BW if the ARCH generic is set to 1. This will reduce the number of SYNCRAM_128BW/156BW core in multi-ways cache to one. The valid/dirty bits are stored in a SYNCRAM_2PFT core.

66.8 Signal descriptions

Table 956 shows the interface signals of the core (VHDL ports).

*Table 956.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
AHBSOV	*	Input	Vector of all AHB slave outputs on the backend AHB bus.	
STO	bit[2]: Access bit[1]: Miss bit[0]: Hit	Output	Statistic output.	

*) see GRLIB IP Library User's Manual.

66.9 Library dependencies

Table 957 shows the libraries used when instantiating the core (VHDL libraries).

*Table 957.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	L2CACHE	Component	Component declaration

66.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
```

```
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
use grlib.tech.all;
library gaisler;
use gaisler.l2cache.all;

entity l2c_ex is
  port (
    clk : in std_ulogic;
    rst : in std_ulogic
  );
end;
.
.
signal ahbsi  : ahb_slv_in_type;
signal ahbso  : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi  : ahb_mst_in_type;
signal ahbmo  : ahb_mst_out_vector := (others => ahbm_none);
signal ahbsi2 : ahb_slv_in_type;
signal ahbso2 : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi2 : ahb_mst_in_type;
signal ahbmo2 : ahb_mst_out_vector := (others => ahbm_none);

architecture rtl of l2c_ex is

begin

...

  l2c0 : l2c
  generic map(hslvidx => 5, hmstidx => 1, cen => 0, haddr => 16#400#, hmask => 16#C00#,
              ioaddr => 16#FF4#, cached => 16#00F3#, repl => 0, ways => 1,
              linesize => 32, waysize => 512, memtech => 0, bbuswidth => 64)
  port map(rst => rst, clk => clk, ahbsi => ahbsi, ahbso => ahbso(5),
           ahbmi => ahbmi2, ahbmo => ahbmo2(1), ahbsov => ahbso2);

...

end;
```

67 L4STAT - LEON4 Statistics Unit

67.1 Overview

The LEON4 Statistics Unit (L4STAT) is used count events in the LEON4 processor and the AHB bus, in order to create performance statistics for various software applications.

L4STAT consists of a configurable number of 32-bit counters, which increment on a certain event. The counters roll over to zero when reaching their maximum value, but can also be automatically cleared on reading to facilitate statistics building over longer periods. Each counter has a control register where the event type is selected. In multi-processor systems, the control registers also indicates which particular processor core is monitored. The table 958 below shows the event types that can be monitored.

NOTE: L4STAT does currently not support double-clocked processor configurations. The processors and statistics unit must be run on the same frequency as the AMBA buses for L4STAT to function correctly.

*Table 958.*Event types and IDs

ID	Event description
Processor events:	
0x00	Instruction cache miss
0x01	Instruction MMU TLB miss
0x02	Inctruction cache hold
0x03	Instruction MMU hold
0x08	Data cache miss
0x09	Data MMU TLB miss
0x0A	Data cache hold
0x0B	Data MMU hold
0x10	Data write buffer hold
0x11	Total instruction count
0x12	Integer instructions
0x13	Floating-point unit instruction count
0x14	Branch prediction miss
0x15	Execution time, excluding debug mode
0x17	AHB utilization (per AHB master) (implementation depedent)
0x18	AHB utilization (total) (implementation dependent)
0x22	Integer branches
0x28	CALL instructions
0x30	Regular type 2 instructions
0x38	LOAD and STORE instructions
0x39	LOAD instructions
0x3A	STORE instructions
AHB events (only available if core is connected to a LEON4 Debug Support Unit):	
0x40	AHB IDLE cycles
0x41	AHB BUSY cycles
0x42	AHB NON-SEQUENTIAL transfers
0x43	AHB SEQUENTIAL transfers
0x44	AHB read accesses
0x45	AHB write accesses

*Table 958.*Event types and IDs

ID	**Event description**
0x46	AHB byte accesses
0x47	AHB half-word accesses
0x48	AHB word accesses
0x49	AHB double word accesses
0x4A	AHB quad word accesses
0x4B	AHB eight word accesses
0x4C	AHB waitstates
0x4D	AHB RETRY responses
0x4E	AHB SPLIT responses
0x4F	AHB SPLIT delay
0x50	AHB bus locked
0x51-0x5F	Reserved
Implementation specific events:	
0x60 - 0x6F	External event 0 - 15

Note that IDs 0x39 (LOAD instructions) and 0x3A (STORE instructions) will both count all LDST and SWAP instructions. The sum of events counted for 0x39 and 0x3A may therefore be larger than the number of events counted with ID 0x38 (LOAD and STORE instructions).

Event 0x00 - 0x3A can be counted of the core has been connected to one or several LEON4 processor cores. Counting of events 0x40 - 0x5F requires that the core is connected to a LEON4 Debug Support Unit (DSU). The core’s Counter control registers have a field that shows if the core has been implemented with this connection. The documentation for the Debug Support Unit contains more information on events 0x40 - 0x5F. Please note that the statistical outputs from the DSU may be subject to AHB trace buffer filters.

The core can also be implemented with support for counting up to 15 external events. These events can come from any source, but should be clocked by a clock which is synchronous with the AMBA clock used for the L4STAT core.

67.2 Multiple APB interfaces

The core can be implemented with two AMBA APB interfaces. The first APB interface always has precedence when both interfaces handle write operations to the same address.

67.3 Registers

The L4STAT core is programmed through registers mapped into APB address space.

Table 959. L4STAT counter control register

APB address offset	**Register**
0x00	Counter 0 value register
0x04	Counter 1 value register
4 * *n*	Counter *n* value register
0x80	Counter 0 control register
0x84	Counter 1 control register
0x80 + (4 * *n*)	Counter *n* control register

Table 960. Counter value register

31 0
CVAL

31: 0	Counter value (CVAL) - This register holds the current value of the counter. If the core has been implemented with support for keeping the maximum count (MC field of Counter control register is ‘1’) and the Counter control register field CD is ‘1’, then the value displayed by this register will be the maximum counter value reached with the settings in the counter’s control register. Writing to this register will write both to the counter and, if implemented, the hold register for the maximum counter value.

Table 961. Counter control register

31 28	27 23	22	21	20	19	18	17	16	15 14	13	12	11 4	3 0
NCPU	NCNT	MC	IA	DS	EE	R	EL	CD	SU	CL	EN	EVENT ID	CPU/AHBM

31: 28	Number of CPU (NCPU) - Number of supported processors - 1
27: 23	Number of counters (NCNT) - Number of implemented counters - 1
22	Maximum count (MC) - If this field is ‘1’ then this counter has support for keeping the maximum count value
21	Internal AHB count (IA) - If this field is ‘1’ the core supports events 0x17 and 0x18
20	DSU support (DS) - If this field is ‘1’ the core supports events 0x40-0x5F
19	External events (EE) - If this field is ‘1’ the core supports external events (events 0x60 - 0x6F)
18	Reserved for future use
17	Event Level (EL) - The value of this field determines the level where the counter keeps running when the CD field below has been set to ‘1’. If this field is ‘0’ the counter will count the time between event assertions. If this field is ‘1’ the counter will count the cycles where the event is asserted. This field can only be set if the MC field of this register is ‘1’.
16	Count maximum duration (CD) - If this bit is set to ‘1’ the core will save the maximum time the selected event has been at the level specified by the EL field. This also means that the counter will be reset when the event is activated or deactivated depending on the value of the EL field. When this bit is set to ‘1’, the value shown in the counter value register will be the maximum current value which may be different from the current value of the counter. This field can only be set if the MC field of this register is ‘1’.
15: 14	Supervisor/User mode filter (SU) - “01” - Only count supervisor mode events, “10” - Only count user mode events, others values - Count events regardless of user or supervisor mode. This setting only applies to events 0x0 - 0x3A
: 13	Clear counter on read (CL) - If this bit is set the counter will be cleared when the counter’s value is read. The register holding the maximum value will also be cleared, if implemented.
12	Enable counter (EN) - Enable counter
11: 4	Event ID to be counted
3: 0	CPU or AHB master to monitor.(CPU/AHBM) - The value of this field does not matter when selecting one of the events coming from the Debug Support Unit or one of the external events.

67.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x047. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

67.5 Configuration options

Table 962 shows the configuration options of the core (VHDL generics).

*Table 962.*Configuration options

Generic	Function	Allowed range	Default
pindex	Selects which APB select signal (PSEL) will be used to access the statistical unit	0 to NAPBMAX-1	0
paddr	The 12-bit MSB APB address	0 to 16#FFF#	0
pmask	The APB address mask	0 to 16#FFF#	16#FFF#
ncnt	Defines the number of counters	1 to 32	4
ncpu	Defines the number of CPUs monitored	1 - 16	1
nmax	If this generic is > 0, the core will include functionality for tracking the longest consecutive time that an event is active or inactive. The functionality will be available for the *nmax* first counters.	0 - 32	0
lahben	If this generic is 1, the core makes use of the AHBSI input for events 0x17 and 0x18, otherwise the AHBSI input is unused and events 0x17 and 0x18 will never increment a counter.	0 - 1	0
dsuen	If this generic is 1, the core makes use of the DSUO input for events 0x40 - 0x5F, otherwise the DSUO input is unused and events 0x40 - 0x5F will never increment a counter.	0 - 1	0
nextev	Defines the number of external events monitored	0 - 16	0
apb2en	Enables the second APB port on the core.	0 - 1	0
pindex2	Selects which APB select signal (PSEL) will be used to access the second interface of the statistical unit	0 to NAPBMAX-1	0
paddr2	The 12-bit MSB APB address for second interface	0 to 16#FFF#	0
pmask2	The APB address mask for second interface	0 to 16#FFF#	16#FFF#

67.6 Signal descriptions

Table 963 shows the interface signals of the core (VHDL ports).

*Table 963.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBSI	*	Input	AHB slave input signals	-
DBGO		Input	LEON4 debug output signals	-
DSUO	ASTAT	Input	DSU4 output signals	-
STATI	EVENT[15:0]	Input	Input for 16 user defined events	High
APB2I	*	Input	Secondary APB slave input signals	-
APB2O	*	Output	Secondary APB slave output signals	-

* see GRLIB IP Library User’s Manual

67.7 Library dependencies

Table 964 shows libraries used when instantiating the core (VHDL libraries).

*Table 964.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	LEON3	Signals	Signal definitions
GAISLER	LEON4	Signals, component	Component declaration

67.8 Component declaration

The core has the following component declaration.

```
library gaisler;
use gaisler.leon3.all;
use gausler.leon4.all;

entity l4stat is
  generic (
    pindex      : integer := 0;
    paddr       : integer := 0;
    pmask       : integer := 16#fff#;
    ncnt        : integer := 4;
    ncpu        : integer := 1
    );
  port (
    rstn   : in std_ulogic;
    clk    : in std_ulogic;
    apbi   : in apb_slv_in_type;
    apbo   : out apb_slv_out_type;
    ahbsi  : in  ahb_slv_in_type;
    dbgo   : in l4_debug_out_vector(0 to NCPU-1));
end;
```

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon4.all;

begin

l4sgen : if CFG_L4S_ENABLE = 1 generate
    l4stat0 : l4stat
      generic map (pindex => 11, paddr => 11, ncnt => CFG_L4S_CNT, ncpu => CFG_NCPU)
      port map (rstn, clkm, apbi, apbo(11), ahbsi, dbgo);
end generate;
```

68 LEON3/FT - High-performance SPARC V8 32-bit Processor

68.1 Overview

LEON3 is a 32-bit processor core conforming to the IEEE-1754 (SPARC V8) architecture. It is designed for embedded applications, combining high performance with low complexity and low power consumption.

The LEON3 core has the following main features: 7-stage pipeline with Harvard architecture, separate instruction and data caches, hardware multiplier and divider, on-chip debug support and multi-processor extensions.

The LEON3 processor can be enhanced with fault-tolerance against SEU errors (referred to as LEON3FT). The fault-tolerance is focused on the protection of on-chip RAM blocks, which are used to implement IU/FPU register files and the cache memory. Configuring the processor to implement fault-tolerance enables additional internal registers, register fields and changes the processor's plug&play device ID. This documentation describes both the LEON3 and LEON3FT versions of the processor.

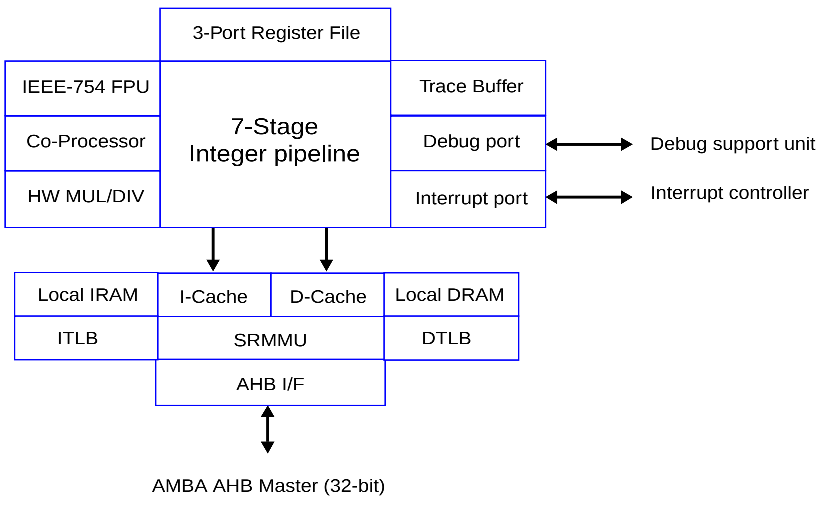


Figure 188. LEON3 processor core block diagram

Note: This manual describes the full functionality of the LEON3 core. Through the use of VHDL generics, parts of the described functionality can be suppressed or modified to generate a smaller or faster implementation.

68.1.1 Integer unit

The LEON3 integer unit implements the full SPARC V8 standard, including hardware multiply and divide instructions. The number of register windows is configurable within the limit of the SPARC standard (2 - 32), with a default setting of 8. The pipeline consists of 7 stages with a separate instruction and data cache interface (Harvard architecture).

68.1.2 Cache sub-system

LEON3 has a highly configurable cache system, consisting of a separate instruction and data cache. Both caches can be configured with 1 - 4 ways, 1 - 256 kbyte/way, 16 or 32 bytes per line. Sub-blocking is implemented with one valid bit per 32-bit word. The instruction cache uses streaming during line-refill to minimize refill latency. The data cache uses write-through policy and implements a double-word write-buffer. The data cache can also perform bus-snooping on the AHB bus. A local scratch pad ram can be added to both the instruction and data cache controllers to allow 0-waitstates access memory without data write back.

68.1.3 Floating-point unit and co-processor

The LEON3 integer unit provides interfaces for a floating-point unit (FPU), and a custom co-processor. Two FPU controllers are available, one for the high-performance GRFPU (available from Aeroflex Gaisler) and one for the Meiko FPU core (available from Sun Microsystems). The floating-point processors and co-processor execute in parallel with the integer unit, and does not block the operation unless a data or resource dependency exists.

68.1.4 Memory management unit

A SPARC V8 Reference Memory Management Unit (SRMMU) can optionally be enabled. The SRMMU implements the full SPARC V8 MMU specification, and provides mapping between multiple 32-bit virtual address spaces and physical memory. A three-level hardware table-walk is implemented, and the MMU can be configured to up to 64 fully associative TLB entries per implemented TLB.

68.1.5 On-chip debug support

The LEON3 pipeline includes functionality to allow non-intrusive debugging on target hardware. To aid software debugging, up to four watchpoint registers can be enabled. Each register can cause a breakpoint trap on an arbitrary instruction or data address range. When the (optional) debug support unit is attached, the watchpoints can be used to enter debug mode. Through a debug support interface, full access to all processor registers and caches is provided. The debug interfaces also allows single stepping, instruction tracing and hardware breakpoint/watchpoint control. An internal trace buffer can monitor and store executed instructions, which can later be read out over the debug interface.

68.1.6 Interrupt interface

LEON3 supports the SPARC V8 interrupt model with a total of 15 asynchronous interrupts. The interrupt interface provides functionality to both generate and acknowledge interrupts.

68.1.7 AMBA interface

The cache system implements an AMBA AHB master to load and store data to/from the caches. The interface is compliant with the AMBA-2.0 standard. During line refill, incremental burst are generated to optimise the data transfer.

68.1.8 Power-down mode

The LEON3 processor core implements a power-down mode, which halts the pipeline and caches until the next interrupt. This is an efficient way to minimize power-consumption when the application is idle, and does not require tool-specific support in form of clock gating. To implement clock-gating, a suitable clock-enable signal is produced by the processor.

68.1.9 Multi-processor support

LEON3 is designed to be used in multi-processor systems. Each processor has a unique index to allow processor enumeration. The write-through caches and snooping mechanism guarantees memory coherency in shared-memory systems.

68.2 LEON3 integer unit

68.2.1 Overview

The LEON3 integer unit implements the integer part of the SPARC V8 instruction set. The implementation is focused on high performance and low complexity. The LEON3 integer unit has the following main features:

- 7-stage instruction pipeline
- Separate instruction and data cache interface
- Support for 2 - 32 register windows
- Hardware multiplier with optional 16x16 bit MAC and 40-bit accumulator
- Radix-2 divider (non-restoring)
- Static branch prediction
- Single-vector trapping for reduced code size

Figure 189 shows a block diagram of the integer unit.

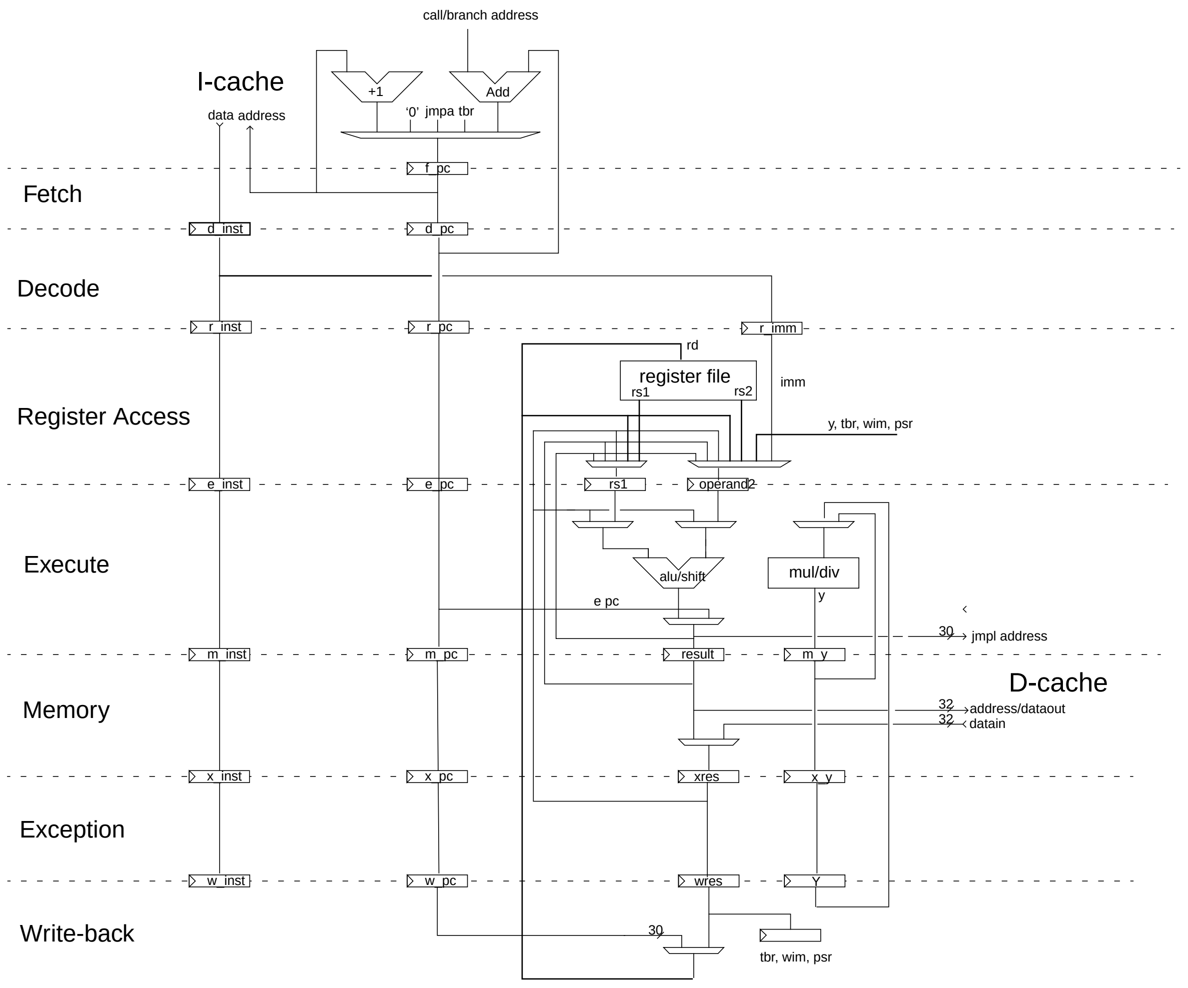


Figure 189. LEON3 integer unit datapath diagram

68.2.2 Instruction pipeline

The LEON3 integer unit uses a single instruction issue pipeline with 7 stages:

1. FE (Instruction Fetch): If the instruction cache is enabled, the instruction is fetched from the instruction cache. Otherwise, the fetch is forwarded to the AHB bus. The instruction is valid at the end of this stage and is latched inside the IU.
2. DE (Decode): The instruction is decoded and the CALL and Branch target addresses are generated.
3. RA (Register access): Operands are read from the register file or from internal data bypasses.
4. EX (Execute): ALU, logical, and shift operations are performed. For memory operations (e.g., LD) and for JMPL/ RETT, the address is generated.
5. ME (Memory): Data cache is read or written at this time.
6. XC (Exception) Traps and interrupts are resolved. For cache reads, the data is aligned as appropriate.
7. WR (Write): The result of any ALU, logical, shift, or cache operations are written back to the register file.

Table 965 lists the cycles per instruction (assuming cache hit and no icc or load interlock):

*Table 965.*Instruction timing

Instruction	**Cycles (MMU disabled)**	**Cycles (MMU fast-write)**	**Cycles (MMU slow-write)**
JMPL, RETT	3	3	3
Double load	2	2	2
Single store	2	2	4
Double store	3	3	5
SMUL/UMUL	1/4*	1/4*	1/4*
SDIV/UDIV	35	35	35
Taken Trap	5	5	5
Atomic load/store	3	3	5
All other instructions	**1**	**1**	**1**

* Multiplication cycle count is 1 clock (1 clock issue rate, 2 clock data latency), for the 32x32 multiplier and 4 clocks (issue rate, 4/5 clocks data latency for standard/pipelined version) for the 16x16 version.

The processor pipeline can be configured for one or two cycles load delay. A branch interlock occurs if an instruction that modifies the ICC bits in %psr is followed by a BICC or TICC instructions within two clocks, unless branch prediction has been enabled.

68.2.3 SPARC Implementor's ID

Aeroflex Gaisler is assigned number 15 (0xF) as SPARC implementor's identification. This value is hard-coded into bits 31:28 in the %psr register. The version number for LEON3 is 3, which is hard-coded in to bits 27:24 of the %psr.

68.2.4 Divide instructions

Full support for SPARC V8 divide instructions is provided (SDIV, UDIV, SDIVCC & UDIVCC). The divide instructions perform a 64-by-32 bit divide and produce a 32-bit result. Rounding and overflow detection is performed as defined in the SPARC V8 standard.

68.2.5 Multiply instructions

The LEON processor supports the SPARC integer multiply instructions UMUL, SMUL UMULCC and SMULCC. These instructions perform a 32x32-bit integer multiply, producing a 64-bit result. SMUL and SMULCC performs signed multiply while UMUL and UMULCC performs unsigned multiply. UMULCC and SMULCC also set the condition codes to reflect the result. The multiply instructions are performed using a 32x32 pipelined hardware multiplier, or a 16x16 hardware multiplier which is iterated four times. To improve the timing, the 16x16 multiplier can optionally be provided with a pipeline stage.

68.2.6 Multiply and accumulate instructions

To accelerate DSP algorithms, two multiply&accumulate instructions are implemented: UMAC and SMAC. The UMAC performs an unsigned 16-bit multiply, producing a 32-bit result, and adds the result to a 40-bit accumulator made up by the 8 lsb bits from the %y register and the %asr18 register. The least significant 32 bits are also written to the destination register. SMAC works similarly but performs signed multiply and accumulate. The MAC instructions execute in one clock but have two clocks latency, meaning that one pipeline stall cycle will be inserted if the following instruction uses the destination register of the MAC as a source operand.

Assembler syntax:

```
umacrs1, reg_imm, rd
smacrs1, reg_imm, rd
```

Operation:

```
prod[31:0] = rs1[15:0] * reg_imm[15:0]
result[39:0] = (Y[7:0] & %asr18[31:0]) + prod[31:0]
(Y[7:0] & %asr18[31:0]) = result[39:0]
rd = result[31:0]
```

%asr18 can be read and written using the RDASR and WRASR instructions.

68.2.7 Compare and Swap instruction (CASA)

LEON3 implements the SPARC V9 Compare and Swap Alternative (CASA) instruction. The CASA is enabled when the integer load delay is set to 1 and the NOTAG generic is 0. The CASA operates as described in the SPARC V9 manual. The instruction is privileged but setting ASI = 0xA (user data) will allow it to be used in user mode.

68.2.8 Branch prediction

Static branch prediction can be optionally be enabled, and reduces the penalty for branches preceded by an instruction that modifies the integer condition codes. The predictor uses a branch-always strategy, and starts fetching instruction from the branch address. On a prediction hit, 1 or 2 clock cycles are saved. No extra penalty incurs for mis-prediction. Branch prediction improves the performance with 10 - 20% on most control-type applications.

68.2.9 Register file data protection

Register file data protection is available for the fault-tolerant version of the LEON3 and is enabled via a VHDL generic. Register file data protection is described in section 68.8.

68.2.10 Hardware breakpoints

The integer unit can be configured to include up to four hardware breakpoints. Each breakpoint consists of a pair of application-specific registers (%asr24/25, %asr26/27, %asr28/29 and %asr30/31) registers; one with the break address and one with a mask:

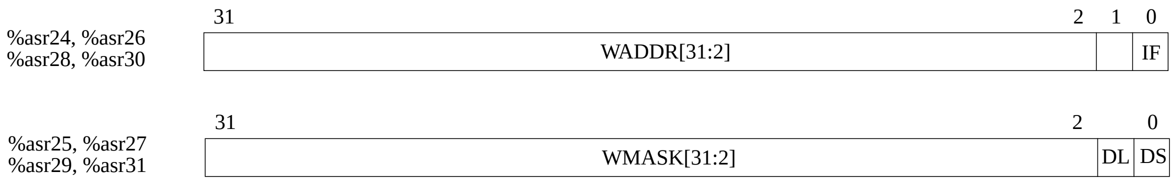


Figure 190. Watch-point registers

Any binary aligned address range can be watched - the range is defined by the WADDR field, masked by the WMASK field (WMASK[x] = 1 enables comparison). On a breakpoint hit, trap 0x0B is generated. By setting the IF, DL and DS bits, a hit can be generated on instruction fetch, data load or data store. Clearing these three bits will effectively disable the breakpoint function.

68.2.11 Instruction trace buffer

The instruction trace buffer consists of a circular buffer that stores executed instructions. The trace buffer operation is controlled through the debug support interface, and does not affect processor operation (see the DSU description). The size of the trace buffer is configurable from 1 to 64 kB through a VHDL generic. The trace buffer is 128 bits wide, and stores the following information:

- Instruction address and opcode
- Instruction result
- Load/store data and address
- Trap information
- 30-bit time tag

The operation and control of the trace buffer is further described in section 25.4. Note that in multiprocessor systems, each processor has its own trace buffer allowing simultaneous tracing of all instruction streams.

68.2.12 Processor configuration register

The application specific register 17 (%asr17) provides information on how various configuration options were set during synthesis. This can be used to enhance the performance of software, or to support enumeration in multi-processor systems. The register can be accessed through the RDASR instruction, and has the following layout:

Table 966. LEON3 configuration register (%asr17)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INDEX				DBP										CS	CF[1]
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CF[0]	DW	SV	LD	FPU		M	V8	NWP			NWIN				

31:28	Processor index (INDEX) - In multi-processor systems, each LEON core gets a unique index to support enumeration. The value in this field is identical to the *hindex* generic parameter in the VHDL model. Value in this field is identical to the *hindex* generic parameter in the VHDL model.
27	Disable Branch Prediction (DBP) - Disables branch prediction when set to ‘1’. Field is only available if the VHDL generic bp is set to the value 2.
26:18	Reserved for future implementations
17	Clock switching enabled (CS). If set, switching between AHB and CPU frequency is available.
16:15	CPU clock frequency (CF). CPU core runs at (CF+1) times AHB frequency.

Table 966. LEON3 configuration register (%asr17)

14	Disable write error trap (DWT). When set, a write error trap (tt = 0x2b) will be ignored. Set to zero after reset.
13	Single-vector trapping (SVT) enable. If set, will enable single-vector trapping. Fixed to zero if SVT is not implemented. Set to zero after reset.
12	Load delay. If set, the pipeline uses a 2-cycle load delay. Otherwise, a 1-cycle load delay i s used. Generated from the lddel generic parameter in the VHDL model.
11:10	FPU option. “00” = no FPU; “01” = GRFPU; “10” = Meiko FPU, “11” = GRFPU-Lite
9	If set, the optional multiply-accumulate (MAC) instruction is available
8	If set, the SPARC V8 multiply and divide instructions are available
7:5	Number of implemented watchpoints (NWP) (0 - 4)
4:0	Number of implemented registers windows corresponds to NWIN+1.

68.2.13 Exceptions

LEON3 adheres to the general SPARC trap model. The table below shows the implemented traps and their individual priority. When PSR (processor status register) bit ET=0, an exception trap causes the processor to halt execution and enter error mode, and the external error signal will then be asserted.

*Table 967.*Trap allocation and priority

Trap	**TT**	**Pri**	**Description**
reset	0x00	1	Power-on reset
write error	0x2b	2	write buffer error during data store
instruction_access_error	0x01	3	Error during instruction fetch
illegal_instruction	0x02	5	UNIMP or other un-implemented instruction
privileged_instruction	0x03	4	Execution of privileged instruction in user mode
fp_disabled	0x04	6	FP instruction while FPU disabled
cp_disabled	0x24	6	CP instruction while Co-processor disabled
watchpoint_detected	0x0B	7	Hardware breakpoint match
window_overflow	0x05	8	SAVE into invalid window
window_underflow	0x06	8	RESTORE into invalid window
register_hardware_error	0x20	9	register file EDAC error (LEON3FT only)
mem_address_not_aligned	0x07	10	Memory access to un-aligned address
fp_exception	0x08	11	FPU exception
cp_exception	0x28	11	Co-processor exception
data_access_exception	0x09	13	Access error during data load, MMU page fault
tag_overflow	0x0A	14	Tagged arithmetic overflow
divide_exception	0x2A	15	Divide by zero
interrupt_level_1	0x11	31	Asynchronous interrupt 1
interrupt_level_2	0x12	30	Asynchronous interrupt 2
interrupt_level_3	0x13	29	Asynchronous interrupt 3
interrupt_level_4	0x14	28	Asynchronous interrupt 4
interrupt_level_5	0x15	27	Asynchronous interrupt 5
interrupt_level_6	0x16	26	Asynchronous interrupt 6
interrupt_level_7	0x17	25	Asynchronous interrupt 7
interrupt_level_8	0x18	24	Asynchronous interrupt 8
interrupt_level_9	0x19	23	Asynchronous interrupt 9
interrupt_level_10	0x1A	22	Asynchronous interrupt 10
interrupt_level_11	0x1B	21	Asynchronous interrupt 11
interrupt_level_12	0x1C	20	Asynchronous interrupt 12
interrupt_level_13	0x1D	19	Asynchronous interrupt 13
interrupt_level_14	0x1E	18	Asynchronous interrupt 14
interrupt_level_15	0x1F	17	Asynchronous interrupt 15
trap_instruction	0x80 - 0xFF	16	Software trap instruction (TA)

68.2.14 Single vector trapping (SVT)

Single-vector trapping (SVT) is an SPARC V8e option to reduce code size for embedded applications. When enabled, any taken trap will always jump to the reset trap handler (%tbr.tba + 0). The trap type will be indicated in %tbr.tt, and must be decoded by the shared trap handler. SVT is enabled by setting bit 13 in %asr17. The model must also be configured with the SVT generic = 1.

68.2.15 Address space identifiers (ASI)

In addition to the address, a SPARC processor also generates an 8-bit address space identifier (ASI), providing up to 256 separate, 32-bit address spaces. During normal operation, the LEON3 processor accesses instructions and data using ASI 0x8 - 0xB as defined in the SPARC standard. Using the LDA/STA instructions, alternative address spaces can be accessed. The table shows the ASI usage for LEON. Only ASI[5:0] are used for the mapping, ASI[7:6] have no influence on operation.

*Table 968.*ASI usage

ASI	Usage
0x01	Forced cache miss
0x02	System control registers (cache control register)
0x08, 0x09, 0x0A, 0x0B	Normal cached access (replace if cacheable)
0x0C	Instruction cache tags
0x0D	Instruction cache data
0x0E	Data cache tags
0x0F	Data cache data
0x10	Flush instruction cache (and also data cache when system is implemented with MMU)
0x11	Flush data cache

68.2.16 Power-down

The processor can be configured to include a power-down feature to minimize power consumption during idle periods. The power-down mode is entered by performing a WRASR instruction to %asr19:

```
wr %g0, %asr19
```

During power-down, the pipeline is halted until the next interrupt occurs. Signals inside the processor pipeline and caches are then static, reducing power consumption from dynamic switching.

Note: %asr19 must always be written with the data value zero to ensure compatiblity with future extensions.

68.2.17 Processor reset operation

The processor is reset by asserting the RESET input for at least 4 clock cycles. The following table indicates the reset values of the registers which are affected by the reset. All other registers maintain their value (or are undefined).

*Table 969.*Processor reset values

Register	Reset value
Trap Base Register	Trap Base Address field reset (value given by RSTADDR VHDL generic)
PC (program counter)	0x0 (RSTADDR VHDL generic)
nPC (next program counter)	0x4 (RSTADDR VHDL generic + 4)
PSR (processor status register)	ET=0, S=1

By default, the execution will start from address 0. This can be overridden by setting the RSTADDR generic in the model to a non-zero value. The reset address is always aligned on a 4 kbyte boundary. If RSTADDR is set to 16#FFFFF#, then the reset address is taken from the signal IRQI.RSTVEC. This allows the reset address to be changed dynamically.

68.2.18 Multi-processor support

The LEON3 processor supports symmetric multi-processing (SMP) configurations, with up to 16 processors attached to the same AHB bus. In multi-processor systems, only the first processor will start. All other processors will remain halted in power-down mode. After the system has been initialized, the remaining processors can be started by writing to the 'MP status register', located in the multiprocessor interrupt controller. The halted processors start executing from the reset address (0 or RSTADDR generic). Enabling SMP is done by setting the *smp* generic to 1 or higher. Cache snooping should always be enabled in SMP systems to maintain data cache coherency between the processors.

68.2.19 Cache sub-system

The LEON3 processor implements a Harvard architecture with separate instruction and data buses, connected to two independent cache controllers. Both instruction and data cache controllers can be separately configured to implement a direct-mapped cache or a multi-way cache with set associativity of 2 - 4. The way size is configurable to 1 - 256 kbyte, divided into cache lines with 16 or 32 bytes of data. In multi-set configurations, one of three replacement policies can be selected: least-recently-used (LRU), least-recently-replaced (LRR) or (pseudo-) random. If the LRR algorithm can only be used when the cache is 2-way associative. A cache line can be locked in the instruction or data cache preventing it from being replaced by the replacement algorithm.

NOTE: The LRR algorithm uses one extra bit in tag rams to store replacement history. The LRU algorithm needs extra flip-flops per cache line to store access history. The random replacement algorithm is implemented through modulo-N counter that selects which line to evict on cache miss.

Cachability for both caches is controlled through the AHB plug&play address information. The memory mapping for each AHB slave indicates whether the area is cachable, and this information is used to (statically) determine which access will be treated as cacheable. This approach means that the cachability mapping is always coherent with the current AHB configuration. The AMBA plug&play cachability can be overridden using the CACHED generic. When this generic is not zero, it is treated as a 16-bit field, defining the cachability of each 256 Mbyte address block on the AMBA bus. A value of 16#00F3# will thus define cachable areas in 0 - 0x20000000 and 0x40000000 - 0x80000000.

68.2.20 AHB bus interface

The LEON3 processor uses one AHB master interface for all data and instruction accesses. Instructions are fetched with incremental bursts if the IB bit is set in the cache control register, otherwise single READ cycles are used.

Data is accessed using byte, half-word and word accesses. See section 68.5.11 for data cache behaviour. The HPROT signals of the AHB bus are driven to indicate if the accesses is instruction or data, and if it is a user or supervisor access.

68.3 Instruction cache

68.3.1 Operation

The instruction cache can be configured as a direct-mapped cache or as a multi-way cache with associativity of 2 - 4 implementing either LRU or random replacement policy or as 2-way associative cache implementing LRR algorithm. The way size is configurable to 1 - 64 kbyte and divided into cache lines of 16- 32 bytes. Each line has a cache tag associated with it consisting of a tag field, valid field with one valid bit for each 4-byte sub-block and optional LRR and lock bits. On an instruction cache miss to a cachable location, the instruction is fetched and the corresponding tag and data line updated. In a multi-way configuration a line to be replaced is chosen according to the replacement policy.

If instruction burst fetch is enabled in the cache control register (CCR) the cache line is filled from main memory starting at the missed address and until the end of the line. At the same time, the instructions are forwarded to the IU (streaming). If the IU cannot accept the streamed instructions due to internal dependencies or multi-cycle instruction, the IU is halted until the line fill is completed. If the IU executes a control transfer instruction (branch/CALL/JMPL/RETT/TRAP) during the line fill, the line fill will be terminated on the next fetch. If instruction burst fetch is enabled, instruction streaming is enabled even when the cache is disabled. In this case, the fetched instructions are only forwarded to the IU and the cache is not updated. During cache line refill, incremental burst are generated on the AHB bus.

If a memory access error occurs during a line fill with the IU halted, the corresponding valid bit in the cache tag will not be set. If the IU later fetches an instruction from the failed address, a cache miss will occur, triggering a new access to the failed address. If the error remains, an instruction access error trap (tt=0x1) will be generated.

68.3.2 Instruction cache tag

A instruction cache tag entry consists of several fields as shown in figure 191:

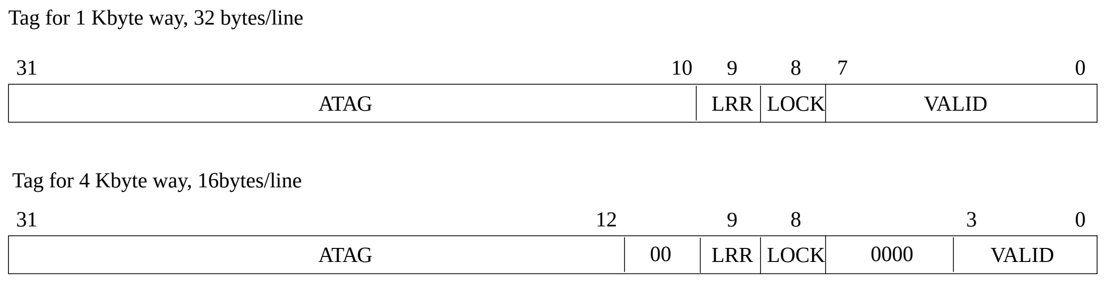

Tag for 1 Kbyte way, 32 bytes/line
31
10
9
8
7
0
ATAG
LRR
LOCK
VALID
Tag for 4 Kbyte way, 16bytes/line
31
12
9
8
3
0
ATAG
00
LRR
LOCK
0000
VALID

Figure 191. Instruction cache tag layout examples

Field Definitions:

[31:10]:	Address Tag (ATAG) - Contains the tag address of the cache line.
[9]:	LRR - Used by LRR algorithm to store replacement history, otherwise 0.
[8]:	LOCK - Locks a cache line when set. 0 if cache locking not implemented.
[7:0]:	Valid (V) - When set, the corresponding sub-block of the cache line contains valid data. These bits are set when a sub-block is filled due to a successful cache miss; a cache fill which results in a memory error will leave the valid bit unset. A FLUSH instruction will clear all valid bits. V[0] corresponds to address 0 in the cache line, V[1] to address 1, V[2] to address 2 and so on.

NOTE: only the necessary bits will be implemented in the cache tag, depending on the cache configuration. As an example, a 4 kbyte cache with 16 bytes per line would only have four valid bits and 20 tag bits. The cache rams are sized automatically by the ram generators in the model.

68.4 Data cache

68.4.1 Operation

The data cache can be configured as a direct-mapped cache or as a multi-way cache with associativity of 2 - 4 implementing either LRU or (pseudo-) random replacement policy or as 2-way associative cache implementing LRR algorithm. The way size is configurable to 1 - 64 kbyte and divided into cache lines of 16 - 32 bytes. Each line has a cache tag associated with it consisting of a tag field, valid field with one valid bit for each 4-byte sub-block and optional lock and LRR bits. On a data cache read-miss to a cachable location 4 bytes of data are loaded into the cache from main memory. The write policy for stores is write-through with no-allocate on write-miss. In a multi-way configuration a line to be replaced on read-miss is chosen according to the replacement policy. Locked AHB transfers are generated for LDST and SWAP instructions. If a memory access error occurs during a data load, the corresponding valid bit in the cache tag will not be set. and a data access error trap (tt=0x9) will be generated.

68.4.2 Write buffer

The write buffer (WRB) consists of three 32-bit registers used to temporarily hold store data until it is sent to the destination device. For half-word or byte stores, the stored data replicated into proper byte alignment for writing to a word-addressed device, before being loaded into one of the WRB registers. The WRB is emptied prior to a load-miss cache-fill sequence to avoid any stale data from being read in to the data cache.

Since the processor executes in parallel with the write buffer, a write error will not cause an exception to the store instruction. Depending on memory and cache activity, the write cycle may not occur until several clock cycles after the store instructions has completed. If a write error occurs, the currently executing instruction will take trap 0x2b.

Note: the 0x2b trap handler should flush the data cache, since a write hit would update the cache while the memory would keep the old value due the write error.

68.4.3 Data cache tag

A data cache tag entry consists of several fields as shown in figure 192:

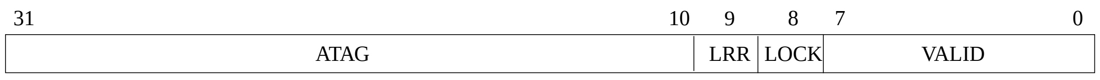


Figure 192. Data cache tag layout

Field Definitions:

[31:10]:	Address Tag (ATAG) - Contains the address of the data held in the cache line.
[9]:	LRR - Used by LRR algorithm to store replacement history. ‘0’ if LRR is not used.
[8]:	LOCK - Locks a cache line when set. ‘0’ if instruction cache locking was not enabled in the configuration.
[3:0]:	Valid (V) - When set, the corresponding sub-block of the cache line contains valid data. These bits are set when a sub-block is filled due to a successful cache miss; a cache fill which results in a memory error will leave the valid bit unset. V[0] corresponds to address 0 in the cache line, V[1] to address 1, V[2] to address 2 and V[3] to address 3.

NOTE: only the necessary bits will be implemented in the cache tag, depending on the cache configuration. As an example, a 2 kbyte cache with 32 bytes per line would only have eight valid bits and 21 tag bits. The cache rams are sized automatically by the ram generators in the model.

68.5 Additional cache functionality

68.5.1 Cache flushing

Both instruction and data cache are flushed by executing the FLUSH instruction. The instruction cache is also flushed by setting the FI bit in the cache control register, while the data cache is also flushed by setting the FD bit in the cache control register. When the processor is implemented with an MMU, both I and D caches can be flushed by writing to any location with ASI=0x10.

Cache flushing takes one cycle per cache line, during which the IU will not be halted, but during which the caches are disabled. When the flush operation is completed, the cache will resume the state (disabled, enabled or frozen) indicated in the cache control register. Diagnostic access to the cache is not possible during a FLUSH operation and will cause a data exception (trap=0x09) if attempted.

68.5.2 Diagnostic cache access

Tags and data in the instruction and data cache can be accessed through ASI address space 0xC, 0xD, 0xE and 0xF by executing LDA and STA instructions. Address bits making up the cache offset will be used to index the tag to be accessed while the least significant bits of the bits making up the address tag will be used to index the cache way.

Diagnostic read of tags is possible by executing an LDA instruction with ASI=0xC for instruction cache tags and ASI=0xE for data cache tags. A cache line and way are indexed by the address bits making up the cache offset and the least significant bits of the address bits making up the address tag. Similarly, the data sub-blocks may be read by executing an LDA instruction with ASI=0xD for instruction cache data and ASI=0xF for data cache data. The sub-block to be read in the indexed cache line and way is selected by A[4:2].

The tags can be directly written by executing a STA instruction with ASI=0xC for the instruction cache tags and ASI=0xE for the data cache tags. The cache line and way are indexed by the address bits making up the cache offset and the least significant bits of the address bits making up the address tag. D[31:10] is written into the ATAG field (see above) and the valid bits are written with the D[7:0] of the write data. Bit D[9] is written into the LRR bit (if enabled) and D[8] is written into the lock bit (if enabled). The data sub-blocks can be directly written by executing a STA instruction with ASI=0xD for the instruction cache data and ASI=0xF for the data cache data. The sub-block to be written in the indexed cache line and way is selected by A[4:2].

In multi-way caches, the address of the tags and data of the ways are concatenated. The address of a tag or data is thus:

ADDRESS = WAY & LINE & DATA & “00”

Examples: the tag for line 2 in way 1 of a 2x4 Kbyte cache with 16 byte line would be:

A[13:12]	= 1	(WAY)
A[11:5]	= 2	(TAG)

=> TAG ADDRESS = 0x1040

The data of this line would be at addresses 0x1040 - 0x104C

68.5.3 Cache line locking

In a multi-way configuration the instruction and data cache controllers can be configured with optional lock bit in the cache tag. Setting the lock bit prevents the cache line to be replaced by the replacement algorithm. A cache line is locked by performing a diagnostic write to the instruction tag on the cache offset of the line to be locked setting the Address Tag field to the address tag of the line to be locked, setting the lock bit and clearing the valid bits. The locked cache line will be updated on a read-miss and will remain in the cache until the line is unlocked. The first cache line on certain cache offset is locked in way 0. If several lines on the same cache offset are to be locked the locking is performed on the same cache offset and in ways in ascending order starting with way 0. The last way can not be locked and is always replaceable. Unlocking is performed in descending way order.

NOTE: Setting the lock bit in a cache tag and reading the same tag will show if the cache line locking was enabled during the LEON3 configuration: the lock bit will be set if the cache line locking was enabled otherwise it will be 0.

68.5.4 Local instruction ram

A local instruction ram can optionally be attached to the instruction cache controller. The size of the local instruction is configurable from 1-256 kB. The local instruction ram can be mapped to any 16 Mbyte block of the address space. When executing in the local instruction ram all instruction fetches are performed from the local instruction ram and will never cause IU pipeline stall or generate an instruction fetch on the AHB bus. Local instruction ram can be accessed through load/store integer word instructions (LD/ST). Only word accesses are allowed, byte, halfword or double word access to the local instruction ram will generate data exception.

68.5.5 Local scratch pad ram

Local scratch pad ram can optionally be attached to both instruction and data cache controllers. The scratch pad ram provides fast 0-waitstates ram memories for both instructions and data. The ram can be between 1 - 256 kbyte, and mapped on any 16 Mbyte block in the address space. Accessed performed to the scratch pad ram are not cached, and will not appear on the AHB bus. The scratch pads rams do not appear on the AHB bus, and can only be read or written by the processor. The instruction ram must be initialized by software (through store instructions) before it can be used. The default address for the instruction ram is 0x8e000000, and for the data ram 0x8f000000. See section 68.14 for additional configuration details. Note: local scratch pad ram can only be enabled when the MMU is disabled.

68.5.6 Data Cache snooping

To keep the data cache synchronized with external memory, cache snooping can be enabled through the *dsnoop* generic. When enabled, the data cache monitors write accesses on the AHB bus to cacheable locations. If an other AHB master writes to a cacheable location which is currently cached in the data cache, the corresponding cache line is marked as invalid.

68.5.7 Cache memory data protection

The cache memories (tags and data) can optionally be protected agains soft errors using byte-parity-codes. Enabling of the data protection is done through the FT generic and the functionality is only enabled for users that have licensed the fault-tolerant version of LEON3. Cache memory data protection is further described in section 68.9.

68.5.8 Cache Control Register

The operation of the instruction and data caches is controlled through a common Cache Control Register (CCR) (table 970). Each cache can be in one of three modes: disabled, enabled and frozen. If disabled, no cache operation is performed and load and store requests are passed directly to the memory controller. If enabled, the cache operates as described above. In the frozen state, the cache is accessed and kept in sync with the main memory as if it was enabled, but no new lines are allocated on read misses.

Table 970. LEON3 Cache Control Register (CCR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		RFT	PS	TB				DS	FD	FI	FT			ST	IB
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
IP	DP	ITE		IDE		DTE		DDE		DF	IF	DCS		ICS	

31:30	Reserved for future implementations
29	Register file test select (RFT). If set, will allow the read-out of IU register file checkbits via ASI 0x0F. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
28	Parity Select (PS) - if set diagnostic read will return 4 check bits in the lsb bits, otherwise tag or data word is returned. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
27:24	Test Bits (TB) - if set, check bits will be xored with test bits TB during diagnostic write. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
23	Data cache snoop enable (DS) - if set, will enable data cache snooping.
22	Flush data cache (FD). If set, will flush the instruction cache. Always reads as zero.
21	Flush Instruction cache (FI). If set, will flush the instruction cache. Always reads as zero.
20:19	FT scheme (FT) - “00” = no FT, “01” = 4-bit checking implemented
18	Reserved for future implementations
17	Separate snoop tags (ST). This read-only bit is set if separate snoop tags are implemented.
16	Instruction burst fetch (IB). This bit enables burst fill during instruction fetch.

Table 970. LEON3 Cache Control Register (CCR)

15	Instruction cache flush pending (IP). This bit is set when an instruction cache flush operation is in progress
14	Data cache flush pending (DP). This bit is set when an data cache flush operation is in progress.
13:12	Instruction Tag Errors (ITE) - Number of detected parity errors in the instruction tag cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
11:10	Instruction Data Errors (IDE) - Number of detected parity errors in the instruction data cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
9:8	Data Tag Errors (DTE) - Number of detected parity errors in the data tag cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
7:6	Data Data Errors (DDE) - Number of detected parity errors in the data data cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
5	Data Cache Freeze on Interrupt (DF) - If set, the data cache will automatically be frozen when an asynchronous interrupt is taken.
4	Instruction Cache Freeze on Interrupt (IF) - If set, the instruction cache will automatically be frozen when an asynchronous interrupt is taken.
3:2	Data Cache state (DCS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.
1:0	Instruction Cache state (ICS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.

If the DF or IF bit is set, the corresponding cache will be frozen when an asynchronous interrupt is taken. This can be beneficial in real-time system to allow a more accurate calculation of worst-case execution time for a code segment. The execution of the interrupt handler will not evict any cache lines and when control is returned to the interrupted task, the cache state is identical to what it was before the interrupt. If a cache has been frozen by an interrupt, it can only be enabled again by enabling it in the CCR. This is typically done at the end of the interrupt handler before control is returned to the interrupted task.

68.5.9 Cache configuration registers

The configuration of the two caches if defined in two registers: the instruction and data configuration registers. These registers are read-only and indicate the size and configuration of the caches.

Table 971. LEON3 Cache configuration register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CL		REPL		SN	WAYS			WSIZE				LR	LSIZE		
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
LRSIZE				LRSTART								M			

31:24	Reserved for future implementations
30	Cache locking (CL). Set if cache locking is implemented.
29:28	Cache replacement policy (REPL). 00 - no replacement policy (direct-mapped cache), 01 - least recently used (LRU), 10 - least recently replaced (LRR), 11 - random
27	Cache snooping (SN). Set if snooping is implemented.
26:24	Cache associativity (WAYS). Number of ways in the cache: 000 - direct mapped, 001 - 2-way associative, 010 - 3-way associative, 011 - 4-way associative
23:20	Way size (WSIZE). Indicates the size (Kbytes) of each cache way. Size = 2^{SIZE}
19	Local ram (LR). Set if local scratch pad ram is implemented.
18:16	Line size (LSIZE). Indicated the size (words) of each cache line. Line size = 2^{LSZ}
15:12	Local ram size (LRSZ). Indicates the size (Kbytes) of the implemented local scratch pad ram. Local ram size = 2^{LRSZ}
11:4	Local ram start address. Indicates the 8 most significant bits of the local ram start address.
3	MMU present. This bit is set to ‘1’ if an MMU is present.
2:0	Reserved for future implementations

All cache registers are accessed through load/store operations to the alternate address space (LDA/STA), using ASI = 2. The table below shows the register addresses:

*Table 972.*ASI 2 (system registers) address map

Address	**Register**
0x00	Cache control register
0x04	Reserved
0x08	Instruction cache configuration register
0x0C	Data cache configuration register

68.5.10 Software consideration

After reset, the caches are disabled and the cache control register (CCR) is 0. Before the caches may be enabled, a flush operation must be performed to initialized (clear) the tags and valid bits. A suitable assembly sequence could be:

```
flush
set 0x81000f, %g1
sta %g1, [%g0] 2
```

68.5.11 AMBA AHB interface

The processor uses one AHB master interface for all data and instruction accesses. Instructions are fetched with incremental bursts if the IB bit is set in the cache control register, otherwise single READ cycles are used.

Data is accessed using byte, half-word and word accesses. A double load/store data access will generate an incremental burst with two accesses. Cachable data is fetched by a burst of word accesses fetching the complete cache line. Earlier versions of the LEON3 processor made use of one word access for each accessed word. This manual describes LEON3v3 which is the only LEON3 version included by default in the GRLIB IP Library. The table below also show the behaviour of earlier versions of the processor.

Processor operation	**Area not cacheable**[1]	**LEON3FTv1/LEON3v1**		**LEON3FTv2/LEON3v3**	
		Area is cacheable[1]		**Area is cacheable**[1]	
		Cache enabled[2]	**Cache disabled**	**Cache enabled**[2]	**Cache disabled**
Data load <= 32-bit	*Read access with size specified by load instruction*	*Word access*	*Read access with size specified by load instruction*	Burst of 32-bit accesses to fetch full cache line.	Read access with size specified by load instruction
Data load 64-bit (LDD)	*Burst of two 32-bit accesses*				Burst of two 32-bit accesses
Data store <= 32-bit	Store access with size specified by store instruction.				
Data store 64-bit (STD)	Burst of two 32-bit store accesses				

[1] Cachability is determined by CACHED generic, if CACHED is zero then cachability is determined via AMBA PnP.

[2] Bus accesses for reads will only be made on L1 cache miss or forced cache miss.

The HPROT signals of the AHB bus are driven to indicate if the accesses is instruction or data, and if it is a user or supervisor access.

In the event of AMBA ERROR responses the cache line will not be written to the cache. If software accessed the memory address that resulted in an AMBA ERROR response then a trap will be generated.

68.6 Memory management unit

A SPARC V8 reference MMU (SRMMU) can optionally be enabled in the LEON3 configuration. For details on the SRMMU operation, see the SPARC V8 manual.

68.6.1 MMU/Cache operation

When the MMU is disabled, the MMU is bypassed and the caches operate with physical address mapping. When the MMU is enabled, the caches tags store the virtual address and also include an 8-bit context field. Both the tag address and context field must match to generate a cache hit.

If cache snooping is desired when the MMU is enabled, bit 2 of the *dsnoop* generic must be set. This will also store the physical address in each cache tag, which is then used for snooping. The size of each data cache way has to be smaller or equal to the MMU page size, which typically is 4 Kbyte (see below). This is necessary to avoid aliasing in the cache since the virtual tags are indexed with a virtual offset while the physical tags are indexed with a physical offset. Physical tags and snoop support is needed for SMP systems using the MMU (linux-2.6).

Because the cache is virtually tagged, no extra clock cycles are needed in case of a cache load hit. In case of a cache miss or store hit (write-through cache), 2 extra clock cycles are used to generate the physical address if there is a TLB hit. If there is a TLB miss the page table must be traversed, resulting in up to four AMBA read accesses and one possible writeback operation. If a combined TLB is used by the instruction cache, the translation is stalled until the TLB is free. If fast TLB operation is selected (tlb_type = 2), the TLB will be accessed simultaneously with tag access, saving 2 clocks on cache miss. This will increase the area somewhat, and may reduce the timing, but usually results in better overall throughput.

An MMU page fault will generate trap 0x09, and update the MMU status registers as defined in the SPARC V8 Manual. The cache and memory will not be modified on an MMU page fault.

68.6.2 Translation look-aside buffer (TLB)

The MMU can be configured to use a shared TLB, or separate TLB for instructions and data. The number of TLB entries (for each implemented TLB) can be set to 2 - 64 in the configuration record. The organisation of the TLB and number of entries is not visible to the software and does thus not require any modification to the operating system.

68.6.3 Variable minimum page sizes

The standard minimum page size for the SRMMU is 4 Kbyte. The minimum page size can also be configured to 8, 16 or 32 Kbyte in order to allow for large data cache ways. The page sizes for level 1, 2 and 3 is seen in the table below:

*Table 973.*MMU page size

Scheme	Level-1	Level-2	Level-3
4 Kbyte (default)	16 Mbyte	256 Kbyte	4 Kbyte
8 Kbyte	32 Mbyte	512 Kbyte	8 Kbyte
16 Kbyte	64 Mbyte	1 Mbyte	16 Kbyte
32 Kbyte	256 Mbyte	2 Mbyte	32 Kbyte

The layouts of the indexes are chosen so that PTE pagetables can be joined together inside one MMU page without leaving holes. The page size can optionally also be choose by the program at run-time by setting generic *mmupgsz* to 1. In this case the page size is choose by bit [17:16] in the MMU control register.

68.6.4 MMU registers

The following MMU registers are implemented:

*Table 974.*MMU registers (ASI = 0x19)

Address	**Register**
0x000	MMU control register
0x100	Context pointer register
0x200	Context register
0x300	Fault status register
0x400	Fault address register

The MMU control register layout can be seen in table 975, while the definition of the remaining MMU registers can be found in the SPARC V8 manual.

Table 975. LEON3 MMU control register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IMPL				VER				ITLB			DTLB			PSZ	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TD	ST													NF	E

31:28	MMU Implementation ID. Hardcoded to “0000”
27:24	MMU Version ID. Hardcoded to “0001”.
23:21	Number of ITLB entries. The number of ITLB entries is calculated as 2^{ITLB}. If the TLB is shared between instructions and data, this field indicates to total number of TLBs.
20:18	Number of DTLB entries. The number of DTLB entries is calculated as 2^{DTLB}. If the TLB is shared between instructions and data, this field is zero.
17:16	Page size. The size of the smallest MMU page. 0 = 4 Kbyte; 1 = 8 Kbyte; 2 = 16 Kbyte; 3 = 32 Kbyte. If the page size is programmable, this field is writable, otherwise it is read-only.
15	TLB disable. When set to 1, the TLB will be disabled and each data access will generate an MMU page table walk.
14	Separate TLB. This bit is set to 1 if separate instruction and data TLBs are implemented
13:2	Reserved for future implementations
1	No Fault. When NF= 0, any fault detected by the MMU causes FSR and FAR to be updated and causes a fault to be generated to the processor. When NF= 1, a fault on an access to ASI 9 is handled as when NF= 0; a fault on an access to any other ASI causes FSR and FAR to be updated but no fault is generated to the processor.
0	Enable MMU. 0 = MMU disabled, 1 = MMU enabled.

68.6.5 ASI mappings

When the MMU is used, the following ASI mappings are added:

*Table 976.*MMU ASI usage

ASI	**Usage**
0x10	Flush I and D cache
0x14	MMU diagnostic dcache context access
0x15	MMU diagnostic icache context access
0x18	Flush TLB and I/D cache
0x19	MMU registers
0x1C	MMU bypass
0x1D	MMU diagnostic access
0x1E	MMU snoop tags diagnostic access

68.6.6 Snoop tag diagnostic access

If the MMU has been configured to use separate snoop tags, they can be accessed via ASI 0x1E. This is primarily useful for RAM testing, and should not be performed during normal operation. The figure below shows the layout of the snoop tag for a 1 Kbyte data cache:

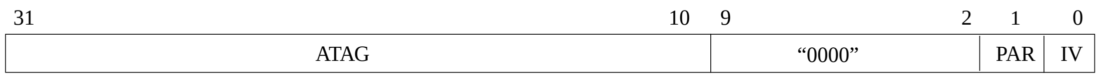


Figure 193. Snoop cache tag layout

[31:10] Address tag. The physical address tag of the cache line.
[1]: Parity. The odd parity over the data tag. LEON3FT only.
[0]: Invalid. When set, the cache line is not valid and will cause a cache miss if accessed by the processor. Only present if fast snooping is enabled.

68.7 Floating-point unit and custom co-processor interface

The SPARC V8 architecture defines two (optional) co-processors: one floating-point unit (FPU) and one user-defined co-processor. Two different FPU’s can be interfaced the LEON3 pipeline: Aeroflex Gaisler’s GRFPU and GRFPU-Lite. Selection of which FPU to use is done through the VHDL model’s generic map. The characteristics of the FPU’s are described in the next sections.

68.7.1 Aeroflex Gaisler’s floating-point unit (GRFPU)

The high-performance GRFPU operates on single- and double-precision operands, and implements all SPARC V8 FPU instructions. The FPU is interfaced to the LEON3 pipeline using a LEON3-specific FPU controller (GRFPC) that allows FPU instructions to be executed simultaneously with integer instructions. Only in case of a data or resource dependency is the integer pipeline held. The GRFPU is fully pipelined and allows the start of one instruction each clock cycle, with the exception is FDIV and FSQRT which can only be executed one at a time. The FDIV and FSQRT are however executed in a separate divide unit and do not block the FPU from performing all other operations in parallel.

All instructions except FDIV and FSQRT has a latency of three cycles, but to improve timing, the LEON3 FPU controller inserts an extra pipeline stage in the result forwarding path. This results in a latency of four clock cycles at instruction level. The table below shows the GRFPU instruction timing when used together with GRFPC:

*Table 977.*GRFPU instruction timing with GRFPC

Instruction	**Throughput**	**Latency**
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FSMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	1	4
FDIVS	14	16
FDIVD	15	17
FSQRTS	22	24
FSQRTD	23	25

The GRFPC controller implements the SPARC deferred trap model, and the FPU trap queue (FQ) can contain up to 7 queued instructions when an FPU exception is taken. When the GRFPU is enabled in the model, the version field in %fsr has the value of 2.

68.7.2 GRFPU-Lite

GRFPU-Lite is a smaller version of GRFPU, suitable for FPGA implementations with limited logic resources. The GRFPU-Lite is not pipelined and executes thus only one instruction at a time. To improve performance, the FPU controller (GRLFPC) allows GRFPU-Lite to execute in parallel with the processor pipeline as long as no new FPU instructions are pending. Below is a table of worst-case throughput of the GRFPU-Lite:

*Table 978.*GRFPU-Lite worst-case instruction timing with GRLFPC

Instruction	**Throughput**	**Latency**
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FSMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	8	8
FDIVS	31	31
FDIVD	57	57
FSQRTS	46	46
FSQRTD	65	65

When the GRFPU-Lite is enabled in the model, the version field in %fsr has the value of 3.

68.8 Register file SEU protection

68.8.1 IU SEU protection

The SEU protection for the integer unit register file can be implemented in four different ways, depending on target technology and available RAM blocks. The SEU protection scheme is selected during synthesis, using the *iuft* VHDL generic. Table 979 below shows the implementation characteristics of the four possible SEU protection schemes.

*Table 979.*Integer unit SEU protection schemes

ID	Implementation	Description
0	Hardened flip-flops or TMR	Register file implemented with SEU hardened flip-flops. No error checking.
1	4-bit parity with restart	4-bit checksum per 32-bit word. Detects and corrects 1 bit per byte (4 bits per word). Pipeline restart on correction.
2	8-bit parity without restart	8-bit checksum per 32-bit word. Detects and corrects 1 bit per byte (4 bits per word). Correction on-the-fly without pipeline restart.
3	7-bit BCH with restart	7-bit BCH checksum per 32-bit word. Detects 2 bits and corrects 1 bit per word. Pipeline restart on correction.

The SEU error detection has no impact on behavior, but a correction cycle (scheme 1 and 3) will delay the current instruction with 6 clock cycles. An uncorrectable error in the IU register file will cause trap 0x20 (*register_access_error*).

68.8.2 FPU SEU protection

The FPU register file has similar SEU protection as the IU register file, but with less configuration options. When the GRFPU is selected and the FPU register file protection is enabled, the protection scheme is always 8-bit parity without pipeline restart. For GRFPU-Lite the protection scheme is always 4-bit parity with pipeline restart. An uncorrectable error in the FPU register file will cause an (deferred) FPU exception with %fsr.ftt set to 5 (hardware_error). When FPU register file protection is disabled the FPU register file is implemented using flip-flops.

68.8.3 ASR16 register

ASR register 16 (%asr16) is used to control the IU/FPU register file SEU protection. It is possible to disable the SEU protection by setting the IDI/FDI bits, and to inject errors using the ITE/FTE bits. Corrected errors in the register file are counted, and available in ICNT and FCNT fields. The counters saturate at their maximum value (7), and should be reset by software after read-out.

Table 980. LEON3FT Register protection control register (%asr16)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FPFT		FCNT												FTE	FDI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IUFT		ICNT			RFTB[7:0]								DP	ITE	IDI

31:30	FP FT ID - Defines which SEU protection is implemented in the FPU (see table 979)
29:27	FP RF error counter - Number of detected parity errors in the FP register file.
26:18	Reserved for future implementations
17	FPU RF Test Enable - Enables FPU register file test mode. Parity bits are xored with TB before written to the FPU register file.
16	FP RF protection disable (FDI) - Disables FP RF parity protection when set.
15:14	IU FT ID - Defines which SEU protection is implemented in the IU (see table 979)
13:11	IU RF error counter - Number of detected parity errors in the IU register file.
10:3	RF Test bits (RFTB) - In test mode, these bits are xored with correct parity bits before written to the register file.
2	DP ram select (DP) - Only applicable if the IU or FPU register files consists of two dual-port rams. See table 981 below.
1	IU RF Test Enable - Enables register file test mode. Parity bits are xored with TB before written to the register file.
0	IU RF protection disable (IDI) - Disables IU RF parity protection when set.

*Table 981.*DP ram select usage

ITE/FTE	DP	Function
1	0	Write to IU register (%i, %l, %o, %g) will only write location of %rs2 Write to FPU register (%f) will only write location of %rs2
1	1	Write to IU register (%i, %l, %o, %g) will only write location of %rs1 Write to FPU register (%f) will only write location of %rs1
0	X	IU and FPU registers written nominally

68.8.4 Register file EDAC/parity bits diagnostic read-out

The register file EDAC/parity bits can be read out through the DSU address space at 0x300800, or by the processor using an LDUHA instruction to ASI 0x0F. The ECC bits are read out for both read ports simultaneously as defined in the figure below:

31 16	8	7 0
RESERVED	RF ECC Port 2	RF ECC port 1

Figure 194. Register file ECC read-out layout

When the checkbits are read out using LDUHA, bit 29 (RFT) in the cache control register should be set to 1. The desired register should be used as address, as shown below (%l0):

```
lduha   [%l0 + %l0] 0x0F, %g1
```

Bit 0 (RF EDAC disable) in %asr16 should be set to 1 during diagnostic read-out with LDUHA, to avoid EDAC correction cycles or error traps.

68.8.5 IU/FPU register file error injection

For test purposes, the IU and FPU register file EDAC/parity checkbits can be modified by software. This is done by setting the ITE or FTE bits to ‘1’. In this mode, the EDAC/parity bits are first XORed with the contents of %asr16.FTB before written to the register files.

68.9 Cache memory protection

Each word in the tag or data memories is protected by four check bits. An error during cache access will cause a cache line flush, and a re-execution of the failing instruction. This will ensure that the complete cache line (tags and data) is refilled from external memory. For every detected error, a counter in the cache control register is incremented. The counters saturate at their maximum value (3), and should be reset by software after read-out. The cache memory check bits can be diagnostically read by setting the PS bit in the cache control register and then perform a normal tag or data diagnostic read.

68.9.1 Diagnostic cache access

The context and parity bits for data and instruction caches can be read out via ASI 0xC - 0xF when the PS bit in the cache control register is set. The data will be organized as shown below:

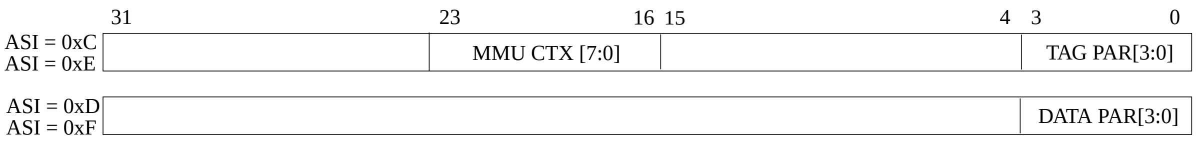


Figure 195. Data cache tag diagnostic access when CCR.PS = ‘1’

68.10 Additional considerations for protection

68.10.1 Data scrubbing

There is generally no need to perform data scrubbing on either IU/FPU register files or the cache memory. During normal operation, the active part of the IU/FPU register files will be flushed to memory on each task switch. This will cause all registers to be checked and corrected if necessary. Since most real-time operating systems performs several task switches per second, the data in the register files will be frequently refreshed.

The similar situation arises for the cache memory. In most applications, the cache memory is significantly smaller than the full application image, and the cache contents is gradually replaced as part of normal operation. For very small programs, the only risk of error build-up is if a part of the application is resident in the cache but not executed for a long period of time. In such cases, executing a cache flush instruction periodically (e.g. once per minute) is sufficient to refresh the cache contents.

68.10.2 Initialization

After power-on, the check bits in the IU and FPU register files are not initialized. This means that access to an un-initialized (un-written) register could cause a register access trap (tt = 0x20). Such behavior is considered as a software error, as the software should not read a register before it has been written. It is recommended that the boot code for the processor writes all registers in the IU and FPU register files before launching the main application.

The check bits in the cache memories do not need to be initialized as this is done automatically during cache line filling.

68.11 LEON3 versions

The primary way to identify the version of a implemented LEON3 processor is to look at the GRLIB build ID, plug&play device identifier and plug&play core revision (part of plug&play information, see GRLIB User's Manual for additional information). This documentation applies to version 3 of the LEON3 processor. Figure 196 shows the relationship between the different earlier LEON3 versions and the current LEON3v3.

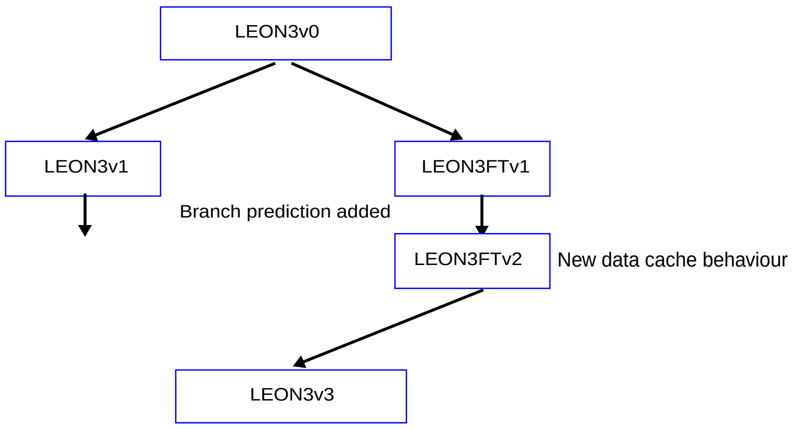


Figure 196. LEON3 processor evolution

Section 68.5.11 describes the difference in data cache behaviour between (LEON3v3, LEON3FTv2) and version 1 of the processor. Other features have been added incrementally to the processor without increasing the processor core version.

68.12 Vendor and device identifiers

The core will have one of two device identifiers depending on if the processor has been implemented with or without fault-tolerance features.

The standard core has vendor identifiers 0x01 (Aeroflex Gaisler) and device identifier 0x003.

If the core has been implemented with fault-tolerance features then the core will be identified with vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x053.

For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

68.13 Implementation

68.13.1 Area and timing

Both area and timing of the LEON3 core depends strongly on the selected configuration, target technology and the used synthesis tool. The table below indicates the typical figures for two baseline configurations.

*Table 982.*Area and timing

	Actel AX2000			**ASIC (0.13 um)**	
Configuration	**Cells**	**RAM64**	**MHz**	**Gates**	**MHz**
LEON3, 8 + 8 Kbyte cache	6,500	40	30	25,000	400
LEON3, 8 + 8 Kbyte cache + DSU3	7,500	40	25	30,000	400

68.13.2 Technology mapping

LEON3 has two technology mapping generics, *fabtech* and *memtech*. The *fabtech* generic controls the implementation of some pipeline features, while *memtech* selects which memory blocks will be used to implement cache memories and the IU/FPU register file. *fabtech* can be set to any of the provided technologies (0 - NTECH) as defined in the GRPIB.TECH package. See the GRLIB Users's Manual for available settings for *memtech*.

68.13.3 RAM usage

The LEON3 core maps all usage of RAM memory on the *syncram*, *syncram_2p* and *syncram_dp* components from the technology mapping library (TECHMAP). The type, configuration and number of RAM blocks is described below.

Register file

The register file is implemented with two *synram_2p* blocks for all technologies where the *regfile_3p_infer* constant in TECHMAP.GENCOMP is set to 0. The organization of the syncram_2p is shown in the following table:

*Table 983.*syncram_2p sizes for LEON3 register file

Register windows	Syncram_2p organization
2 - 3	64x32
4 - 7	128x32
8 - 15	256x32
16-31	512x31
32	1024x32

If *regfile_3p_infer* is set to 1, the synthesis tool will automatically infer the register. On FPGA technologies, it can be in either flip-flops or RAM cells, depending on the tool and technology. On ASIC technologies, it will be flip-flops. The amount of flip-flops inferred is equal to the number of registers:

Number of flip-flops = ((NWINDOWS *16) + 8) * 32

FP register file

If FPU support is enabled, the FP register file is implemented with four *synram_2p* blocks when the *regfile_3p_infer* constant in TECHMAP.GENCOMP is set to 0. The organization of the syncram_2p blocks is 16x32.

If *regfile_3p_infer* is set to 1, the synthesis tool will automatically infer the FP register file. For ASIC technologies the number of inferred flip-flops is equal to number of bits in the FP register file which is 32 * 32 = 1024.

Cache memories

RAM blocks are used to implement the cache tags and data memories. Depending on cache configuration, different types and sizes of RAM blocks are used.

The tag memory is implemented with one *syncram* per cache way when no snooping is enabled. The tag memory depth and width is calculated as follows:

Depth = (cache way size in bytes) / (cache line size in bytes)

Width = 32 - log2(cache way size in bytes) + (cache line size in bytes)/4 + lrr + lock

For a 2 Kbyte cache way with lrr replacement and 32 bytes/line, the tag RAM depth will be (2048/32) = 64. The width will be: 32 - log2(2048) + 32/4 + 1 = 32 - 11 + 8 + 1 = 28. The tag RAM organization will thus be 64x28 for the configuration. If the MMU is enabled, the tag memory width will increase with 8 to store the process context ID, and the above configuration will us a 64x36 RAM.

If snooping is enabled, the tag memories will be implemented using the *syncram_dp* component (dual-port RAM). One port will be used by the processor for cache access/refill, while the other port will be used by the snooping and invalidation logic. The size of the *syncram_dp* block will be the same as when snooping is disabled. If physical snooping is enabled (separate snoop tags), one extra RAM block per data way will be instatiated to hold the physical tags. The width of the RAM block will be the same as the tag address: 32 - log2(way size). A 4 Kbyte data cache way will thus require a 32 - 12 = 20 bit wide RAM block for the physical tags. If fast snooping is enabled, the tag RAM (virtual and physical) will be implemented using *syncram_2p* instead of *syncram_dp*. This can be used to implement snooping on technologies which lack dual-port RAM but have 2-port RAM.

The data part of the caches (storing instructions or data) is always 32 bit wide. The depth is equal to the way size in bytes, divided by 4. A cache way of 2 Kbyte will thus use *syncram* component with and organization of 512x32.

Instruction Trace buffer

The instruction trace buffer will use four identical RAM blocks (*syncram*) to implement the buffer memory. The syncrams will always be 32-bit wide. The depth will depend on the TBUF generic, which indicates the total size of trace buffer in Kbytes. If TBUF = 1 (1 kbyte), then four RAM blocks of 64x32 will be used. If TBUF = 2, then the RAM blocks will be 128x32 and so on.

Scratch pad RAM

If the instruction scratch pad RAM is enabled, a *syncram* block will be instantiated with a 32-bit data width. The depth of the RAM will correspond to the configured scratch pad size. An 8 kbyte scratch pad will use a *syncram* with 2048x32 organization. The RAM block for the data scratch pad will be configured in the same way as the instruction scratch pad.

68.13.4 Double clocking

The LEON3 CPU core be clocked at twice the clock speed of the AMBA AHB bus. When clocked at double AHB clock frequency, all CPU core parts including integer unit and caches will operate at double AHB clock frequency while the AHB bus access is performed at the slower AHB clock frequency. The two clocks have to be synchronous and a multicycle paths between the two clock domains have to be defined at synthesis tool level. Separate components (leon3s2x, leon3x, leon3ft2x) are provided for the double clocked core. Double clocked versions of DSU (dsu3_2x) and MP interrupt controller (irqmp2x) are used in a double clocked LEON3 system. An AHB clock qualifier signal (*clken* input) is used to identify end of AHB cycle. The AHB qualifier signal is generated in CPU clock domain and is high during the last CPU clock cycle under AHB clock low-phase. Sample *leon3-clk2x* design provides a module that generates an AHB clock qualifier signal.

Double-clocked design has two clock domains: AMBA clock domains (HCLK) and CPU clock domain (CPUCLK). LEON3 (leon3s2x component) and DSU3 (dsu3_2x) belong to CPU clock domain (clocked by CPUCLK), while the rest of the system is in AMBA clock domain (clocked by HCLK). Paths between the two clock domains (paths starting in CPUCLK domain and ending in

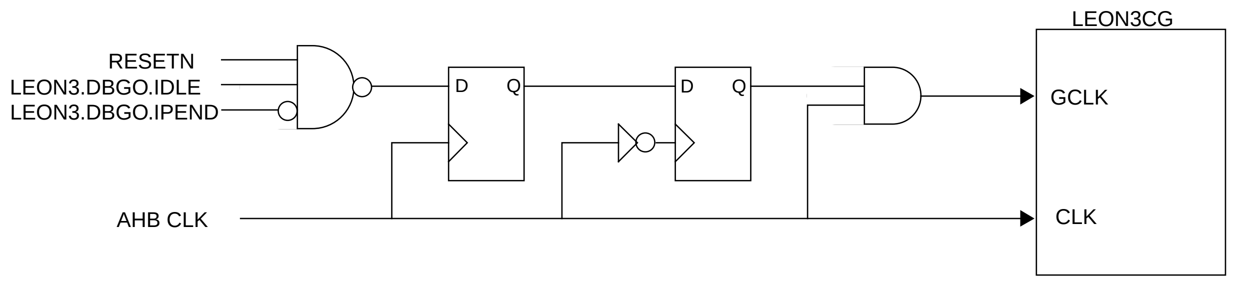
HCLK and paths starting in HCLK domain and ending in CPUCLK domain) are multicycle paths with propagation time of two CPUCLK periods (or one HCLK period) with following exceptions:

Start point	**Through**	**End point**	**Propagation time**
leon3s2x core			
CPUCLK	ahbi	CPUCLK	2 CPUCLK
CPUCLK	ahbsi	CPUCLK	2 CPUCLK
CPUCLK	ahbso	CPUCLK	2 CPUCLK
HCLK	irqi	CPUCLK	1 CPUCLK
CPUCLK	irqo	HCLK	1 CPUCLK
CPUCLK		u0_0/p0/c0/sync0/r[*] (register)	1 CPUCLK

Start point	**Through**	**End point**	**Propagation time**
dsu3_2x core			
CPUCLK	ahbmi	CPUCLK	2 CPUCLK
CPUCLK	ahbsi	CPUCLK	2 CPUCLK
	dsui	CPUCLK	1 CPUCLK
r[*] (register)		rh[*] (register)	1 CPUCLK
irqmp2x core			
r2[*] (register)		r[*] (register)	1 CPUCLK

68.13.5 Clock gating

To further reduce the power consumption of the processor, the clock can be gated-off when the processor has entered power-down state. Since the cache controllers and MMU operate in parallel with the processor, the clock cannot be gated immediately when the processor has entered the power-down state. Instead, a power-down signal (DBGO.idle) is generated when all outstanding AHB accesses have been completed and it is safe to gate the clock. This signal should be clocked though a positive-edge flip-flop followed by a negative-edge flip-flop to guarantee that the clock is gated off during the clock-low phase. To ensure proper start-up state, the clock should not be gated during reset and at least 3 clocks after that reset has been de-asserted.



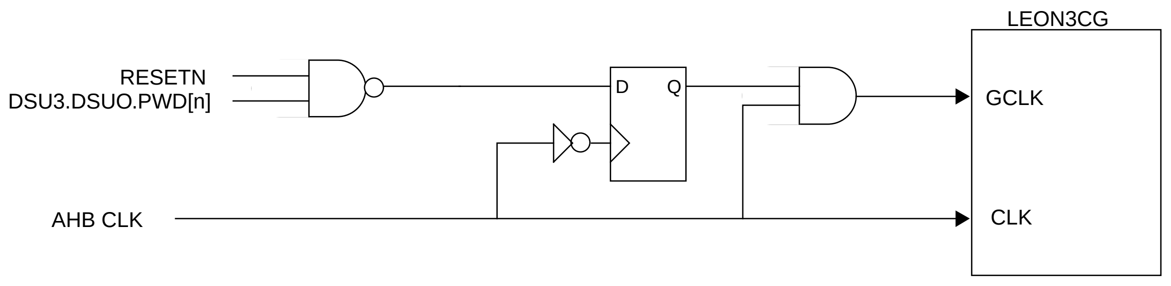


Figure 197. Examples of LEON3 clock gating

The processor should exit the power-down state when an interrupt become pending. The signal DBGO.ipend will then go high when this happen, and should be used to re-enable the clock.

When the debug support unit (DSU3) is used, the DSUO.pwd signal should be used instead of DBGO.idle. This will ensure that the clock also is re-enabled when the processor is switched from power-down to debug state by the DSU. The DSUO.pwd is a vector with one power-down signal per CPU (for SMP systems). DSUO.pwd takes DBGO.ipend into account, and no further gating or latching needs to be done of this signal. If cache snooping has been enabled, the continuous clock will ensure that the snooping logic is activated when necessary and will keep the data cache synchronized even when the processor clock is gated-off. In a multi-processor system, all processor except node 0 will enter power-down after reset and will allow immediate clock-gating without additional software support.

Clock-tree routing must ensure that the continuous clock (CLK) and the gated clock (GCLK) are phase-aligned. The template design *leon3-clock-gate* shows an example of a clock-gated system. The *leon3cg* entity should be used when clock gating is implemented. This entity has one input more (GCLK) which should be driven by the gated clock. Using the double-clocked version of leon3 (leon3s2x), the GCLK2 is the gated double-clock while CLK and CLK2 should be continuous.

68.13.6 Scan support

If the SCANTEST generic is set to 1, support for scan testing is enabled. This will make use of the AHB scan support signals in the following manner: when AHBI.testen and AHBI.scanen are both ‘1’, the select signals to all RAM blocks (cache RAM, register file and DSU trace buffers) are disabled. This means that when the scan chain is shifted, no accidental write or read can occur in the RAM blocks. The scan signal AHBI.testrst is not used as there are no asynchronous resets in the LEON3 core.

68.14 Configuration options

Table 984 shows the configuration options of the core (VHDL generics).

*Table 984.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
fabtech	Target technology	0 - NTECH	0 (inferred)
memtech	Vendor library for regfile and cache RAMs	0 - NTECH	0 (inferred)
nwindows	Number of SPARC register windows. Choose 8 windows to be compatible with Bare-C and RTEMS cross-compilers.	2 - 32	8
dsu	Enable Debug Support Unit interface	0 - 1	0
fpu	Floating-point Unit 0 : no FPU 1 - 7: GRFPU 1 - inferred multiplier, 2 - DW multiplier, 3 - Module Generator multiplier, 4 - Technology specific multiplier 8 - 14: GRFPU-Lite 8 - simple FPC, 9 - data forwarding FPC, 10 - non-blocking FPC 15: Meiko 16 - 31: as above (modulo 16) but use netlist	0 - 31	0
v8	Generate SPARC V8 MUL and DIV instructions This generic is assigned with the value: *mult + 4*struct* Where *mult* selects between the following implementation options for the multiplier and divider: 0 : No multiplier or divider 1 : 16x16 multiplier 2 : 16x16 pipelined multiplier 16#32# : 32x32 pipelined multiplier Where *struct* selects the structure option for the integer multiplier. The following structures can be selected: 0: Inferred by synthesis tool 1: Generated using Module Generators from NTNU 2: Using technology specific netlists (techspec) 3: Using Synopsys DesignWare (DW02_mult and DW_mult_pipe)	0 - 16#3F#	0
cp	Generate co-processor interface	0 -1	0
mac	Generate SPARC V8e SMAC/UMAC instruction	0 - 1	0
pclow	Least significant bit of PC (Program Counter) that is actually generated. PC[1:0] are always zero and are normally not generated. Generating PC[1:0] makes VHDL-debugging easier.	0, 2	2
notag	Disable tagged arithmetic and CASA instructions	0 - 1	0
nwp	Number of watchpoints	0 - 4	0
icen	Enable instruction cache	0 - 1	1
irepl	Instruction cache replacement policy. 0 - least recently used (LRU), 1 - least recently replaced (LRR), 2 - random	0 - 1	0
isets	Number of instruction cache ways. Note: Generic named isets due to historical reasons.	1 - 4	1
ilinesize	Instruction cache line size in number of words	4, 8	4
isetsize	Size of each instruction cache way in kByte	1 - 256	1
isetlock	Enable instruction cache line locking	0 - 1	0
dcen	Data cache enable	0 - 1	1

*Table 984.*Configuration options

Generic	Function	Allowed range	Default
drepl	Data cache replacement policy. 0 - least recently used (LRU), 1 - least recently replaced (LRR), 2 - random	0 - 1	0
dsets	Number of data cache ways Note: Generic named dsets due to historical reasons.	1 - 4	1
dlinesize	Data cache line size in number of words	4, 8	4
dsetsize	Size of each data cache way in kByte	1 - 256	1
dsetlock	Enable data cache line locking	0 - 1	0
dsnoop	Enable data cache snooping Bit 0-1: 0: disable, 1: slow, 2: fast (see text) Bit 2: 0: simple snooping, 1: save extra physical tags (MMU snooping)	0 - 6	0
ilram	Enable local instruction RAM	0 - 1	0
ilramsize	Local instruction RAM size in kB	1 - 512	1
ilramstart	8 MSB bits used to decode local instruction RAM area	0 - 255	16#8E#
dlram	Enable local data RAM (scratch-pad RAM)	0 - 1	0
dlramsize	Local data RAM size in kB	1 - 512	1
dlramstart	8 MSB bits used to decode local data RAM area	0 - 255	16#8F#
mmuen	Enable memory management unit (MMU)	0 - 1	0
itlbnum	Number of instruction TLB entries	2 - 64	8
dtlbnum	Number of data TLB entries	2 - 64	8
tlb_type	0 : separate TLB with slow write 1: shared TLB with slow write 2: separate TLB with fast write	0 - 2	1
tlb_rep	LRU (0) or Random (1) TLB replacement	0 - 1	0
lddel	Load delay. One cycle gives best performance, but might create a critical path on targets with slow (data) cache memories. A 2-cycle delay can improve timing but will reduce performance with about 5%. Note that lddel = 1 is required for CASA.	1 - 2	2
disas	Print instruction disassembly in VHDL simulator console.	0 - 1	0
tbuf	Size of instruction trace buffer in kB (0 - instruction trace disabled)	0 - 64	0
pwd	Power-down. 0 - disabled, 1 - area efficient, 2 - timing efficient.	0 - 2	1
svt	Enable single-vector trapping	0 - 1	0
rstaddr	Default reset start address	0 - (2**^{20}-1)	0
smp	Enable multi-processor support	0 - 15	0
iuft, fpft	Register file SEU protection. (0: no protection; 1 : 4-bit parity, 2 : 8-bit parity; 3 : 7-bit BCH)	0 - 3	0
cft	Enable cache memory SEU protection.	0 - 1	0
iuinj ceinj	Error injection. Used for simulation only.	0 - 3	0
cached	Fixed cacheability mask	0 - 16#FFFF#	0
clk2x	Double-clocking, frequency factor		0
netlist	Use netlist rather then RTL code (currently unused)	0 - 1	0
scantest	Enable scan test support	0 - 1	0
mmupgsz	MMU Page size. 0 = 4K, 1 = 8K, 2 = 16K, 3 = 32K, 4 = programmable.	0 - 4	0

*Table 984.*Configuration options

Generic	Function	Allowed range	Default
bp	Enable branch prediction	0 - 2	1

68.15 Signal descriptions

Table 985 shows the interface signals of the core (VHDL ports). There are several top-level entities available for the LEON3 processor. The *leon3x* entity contains all signals and settings. The other entities are wrappers around *leon3x*. The available entities are:

- leon3cg - Top-level with support for clock gating. Deprecated, do not use for new designs.
- leon3ft2x - Top-level with support for FT, double clocking and clock gating.
- leon3ftsh - Entity with support for FT and shared FPU.
- leon3ft - Entity with support for FT and clock gating, no separate FPU clock.
- leon3s2x - Top-level with support for clock gating and double clocking, no separate FPU clock.
- leon3sh - Top-level with support for shared FPU.
- leon3s - Simplest top-level, no FT, clock gating or shared FPU.
- leon3x - Entity with support for all features (double clocking, FT, clock gating, shared FPU)

*Table 985.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	AMBA and processor clock	-
GCLK2	N/A	Input	Gated processor clock in 2x mode	
GFCLK2	N/A	Input	Gated FPU clock in 2x mode	
CLK2	N/A	Input	Processor clock in 2x mode	
RSTN	N/A	Input	Reset	Low
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
AHBSI	*	Input	AHB slave input signals	-
IRQI	IRL[3:0]	Input	Interrupt level	High
	RST	Input	Reset power-down and error mode	High
	RUN	Input	Start after reset (SMP system only)	High
	RSTVEC[31:12]	Input	Reset start addr. (SMP and dynamic reset addr.)	-
	IACT	Input	Unused	-
	INDEX[3:0]	Input	Unused	-
	HRDRST	Input	Unused	-
IRQO	INTACK	Output	Interrupt acknowledge	High
	IRL[3:0]	Output	Processor interrupt level	High
	PWD	Output	Processor in power-down mode	High
	FPEN	Output	Floating-point unit enabled	High
	IDLE	Output	Always low	High
DBGI	-	Input	Debug inputs from DSU	-
DBGO	-	Output	Debug outputs to DSU	-
	ERROR		Processor in error mode, execution halted	Low
CLKEN		Input	Clock enable/qualifier used in 2x mode	High

* see GRLIB IP Library User’s Manual

68.16 Library dependencies

Table 986 shows the libraries used when instantiating the core (VHDL libraries).

*Table 986.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	LEON3	Component, signals	LEON3 component declaration, interrupt and debug signals declaration

68.17 Component declaration

The core has the following component declaration.

```
entity leon3s is
  generic (
    hindex    : integer               := 0;
    fabtech   : integer range 0 to NTECH  := 0;
    memtech   : integer range 0 to NTECH  := 0;
    nwindows  : integer range 2 to 32 := 8;
    dsu       : integer range 0 to 1  := 0;
    fpu       : integer range 0 to 3  := 0;
    v8        : integer range 0 to 2  := 0;
    cp        : integer range 0 to 1  := 0;
    mac       : integer range 0 to 1  := 0;
    pclow     : integer range 0 to 2  := 2;
    notag     : integer range 0 to 1  := 0;
    nwp       : integer range 0 to 4  := 0;
    icen      : integer range 0 to 1  := 0;
    irepl     : integer range 0 to 2  := 2;
    isets     : integer range 1 to 4  := 1;
    ilinesize : integer range 4 to 8  := 4;
    isetsize  : integer range 1 to 256 := 1;
    isetlock  : integer range 0 to 1  := 0;
    dcen      : integer range 0 to 1  := 0;
    drepl     : integer range 0 to 2  := 2;
    dsets     : integer range 1 to 4  := 1;
    dlinesize : integer range 4 to 8  := 4;
    dsetsize  : integer range 1 to 256 := 1;
    dsetlock  : integer range 0 to 1  := 0;
    dsnoop    : integer range 0 to 6:= 0;
    ilram      : integer range 0 to 1 := 0;
    ilramsize  : integer range 1 to 512 := 1;
    ilramstart : integer range 0 to 255 := 16#8e#;
    dlram      : integer range 0 to 1 := 0;
    dlramsize  : integer range 1 to 512 := 1;
    dlramstart : integer range 0 to 255 := 16#8f#;
    mmuen     : integer range 0 to 1  := 0;
    itlbnum   : integer range 2 to 64 := 8;
    dtlbnum   : integer range 2 to 64 := 8;
    tlb_type  : integer range 0 to 1 := 1;
    tlb_rep   : integer range 0 to 1 := 0;
    lddel     : integer range 1 to 2  := 2;
    disas     : integer range 0 to 1  := 0;
    tbuf      : integer range 0 to 64 := 0;
    pwd       : integer range 0 to 2  := 2;     -- power-down
    svt       : integer range 0 to 1  := 1;     -- single vector trapping
    rstaddr   : integer               := 0;
    smp       : integer range 0 to 15 := 0;    -- support SMP systems
    cached    : integer               := 0;     -- cacheability table
    scantest  : integer               := 0
     mmupgsz   : integer range 0 to 5  := 0;
    bp        : integer               := 1
);
```

```
port (
    clk    : in  std_ulogic;
    rstn   : in  std_ulogic;
    ahbi   : in  ahb_mst_in_type;
    ahbo   : out ahb_mst_out_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : in  ahb_slv_out_vector;
    irqi   : in  l3_irq_in_type;
    irqo   : out l3_irq_out_type;
    dbgi   : in  l3_debug_in_type;
    dbgo   : out l3_debug_out_type
  );
end;
```

69 LEON4 - High-performance SPARC V8 32-bit Processor

69.1 Overview

LEON4 is a 32-bit processor core conforming to the IEEE-1754 (SPARC V8) architecture. It is designed for embedded applications, combining high performance with low complexity and low power consumption.

The LEON4 core has the following main features: 7-stage pipeline with Harvard architecture, separate instruction and data caches, hardware multiplier and divider, on-chip debug support and multi-processor extensions.

The LEON4 processor can be enhanced with fault-tolerance against SEU errors. The fault-tolerance is focused on the protection of on-chip RAM blocks, which are used to implement IU/FPU register files and the cache memory.

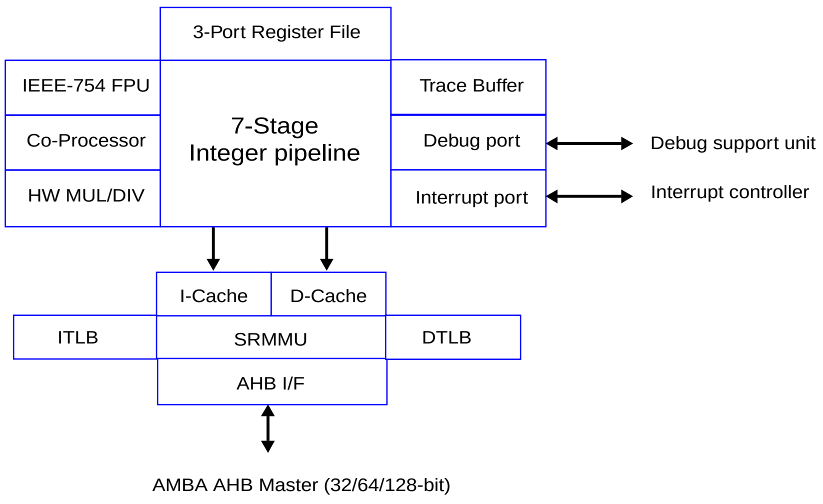


Figure 198. LEON4 processor core block diagram

Note: This manual describes the full functionality of the LEON4 core. Through the use of VHDL generics, parts of the described functionality can be suppressed or modified to generate a smaller or faster implementation.

69.1.1 Integer unit

The LEON4 integer unit implements the full SPARC V8 standard, including hardware multiply and divide instructions. The number of register windows is configurable within the limit of the SPARC standard (2 - 32), with a default setting of 8. The pipeline consists of 7 stages with a separate instruction and data cache interface (Harvard architecture).

69.1.2 Cache sub-system

LEON4 has a highly configurable cache system, consisting of a separate instruction and data cache. Both caches can be configured with 1 - 4 ways, 1 - 256 kbyte/way. The instruction cache contains 16 or 32 bytes/line while the data cache uses 16 byte/line. The data cache uses write-through policy and implements a double-word write-buffer. The data cache can also perform bus-snooping on the AHB bus.

69.1.3 Floating-point unit

The LEON4 integer unit provides an interface for the high-performance GRFPU floating-point unit. The floating-point unit executes in parallel with the integer unit, and does not block the operation unless a data or resource dependency exists.

69.1.4 Memory management unit

A SPARC V8 Reference Memory Management Unit (SRMMU) can optionally be enabled. The SRMMU implements the full SPARC V8 MMU specification, and provides mapping between multiple 32-bit virtual address spaces and physical memory. A three-level hardware table-walk is implemented, and the MMU can be configured to up to 64 fully associative TLB entries per implemented TLB.

69.1.5 On-chip debug support

The LEON4 pipeline includes functionality to allow non-intrusive debugging on target hardware. To aid software debugging, up to four watchpoint registers can be enabled. Each register can cause a breakpoint trap on an arbitrary instruction or data address range. When the (optional) debug support unit is attached, the watchpoints can be used to enter debug mode. Through a debug support interface, full access to all processor registers and caches is provided. The debug interfaces also allows single stepping, instruction tracing and hardware breakpoint/watchpoint control. An internal trace buffer can monitor and store executed instructions, which can later be read out over the debug interface.

69.1.6 Interrupt interface

LEON4 supports the SPARC V8 interrupt model with a total of 15 asynchronous interrupts. The interrupt interface provides functionality to both generate and acknowledge interrupts.

69.1.7 AMBA interface

The cache system implements an AMBA AHB master to load and store data to/from the caches. The interface is compliant with the AMBA-2.0 standard. During line refill, incremental burst are generated to optimise the data transfer. The AMBA interface can be configured to use a 64 or 128-bit bus on cache line fills.

69.1.8 Power-down mode

The LEON4 processor core implements a power-down mode, which halts the pipeline and caches until the next interrupt. This is an efficient way to minimize power-consumption when the application is idle, and does not require tool-specific support in form of clock gating. To implement clock-gating, a suitable clock-enable signal is produced by the processor.

69.1.9 Multi-processor support

LEON4 is designed to be used in multi-processor systems. Each processor has a unique index to allow processor enumeration. The write-through caches and snooping mechanism guarantees memory coherency in shared-memory systems.

69.2 LEON4 integer unit

69.2.1 Overview

The LEON4 integer unit implements the integer part of the SPARC V8 instruction set. The implementation is focused on high performance and low complexity. The LEON4 integer unit has the following main features:

- 7-stage instruction pipeline
- Separate instruction and data cache interface
- Support for 2 - 32 register windows
- Hardware multiplier with optional 16x16 bit MAC and 40-bit accumulator
- Radix-2 divider (non-restoring)
- Static branch prediction
- Single-vector trapping for reduced code size

Figure 199 shows a block diagram of the integer unit.

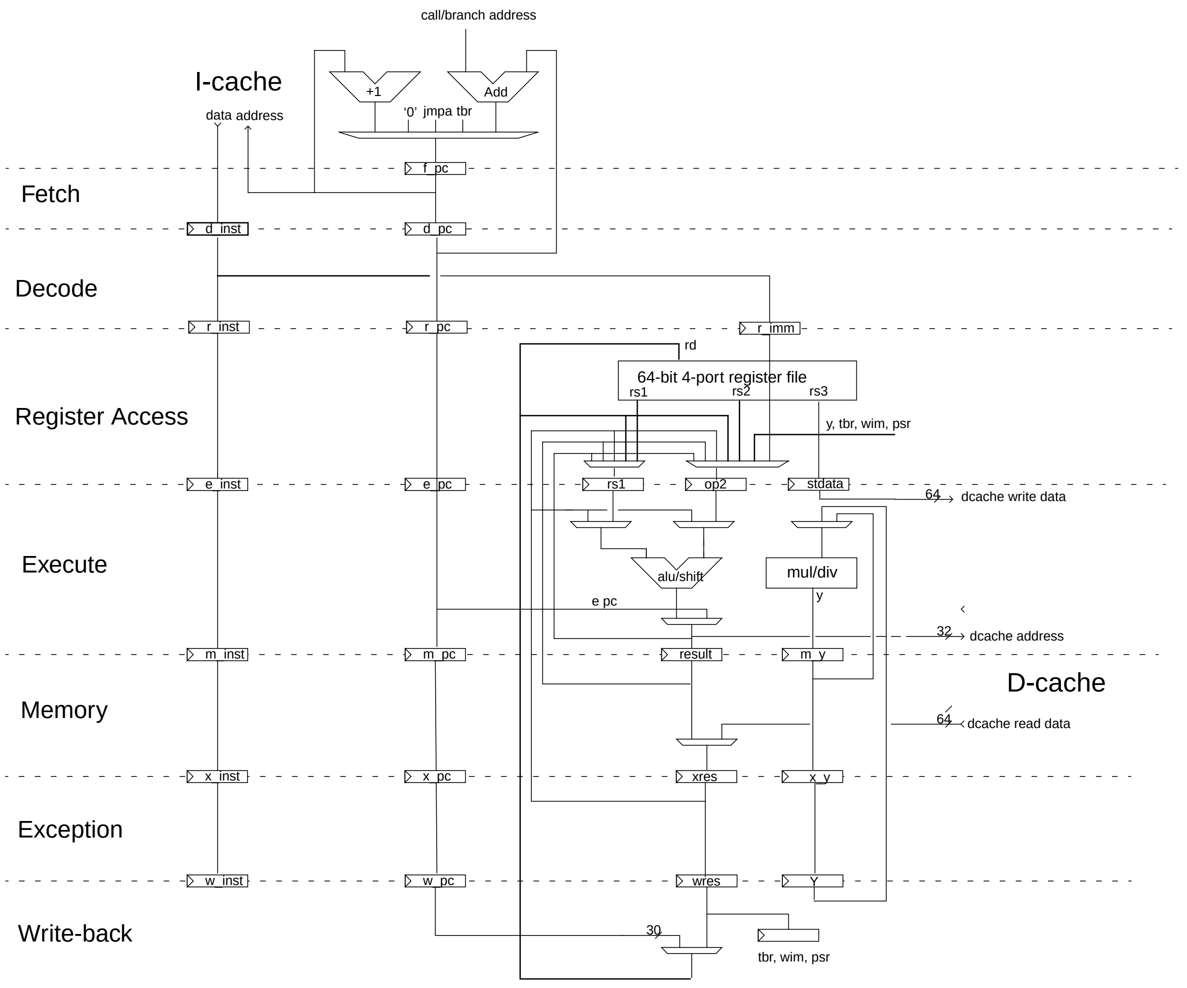


Figure 199. LEON4 integer unit datapath diagram

69.2.2 Instruction pipeline

The LEON4 integer unit uses a single instruction issue pipeline with 7 stages:

1. FE (Instruction Fetch): If the instruction cache is enabled, the instruction is fetched from the instruction cache. Otherwise, the fetch is forwarded to the memory controller. The instruction is valid at the end of this stage and is latched inside the IU.
2. DE (Decode): The instruction is decoded and the CALL/Branch target addresses are generated.
3. RA (Register access): Operands are read from the register file or from internal data bypasses.
4. EX (Execute): ALU, logical, and shift operations are performed. For memory operations (e.g., LD) and for JMPL/RETT, the address is generated.
5. ME (Memory): Data cache is accessed. Store data read out in the execution stage is written to the data cache at this time.
6. XC (Exception) Traps and interrupts are resolved. For cache reads, the data is aligned.
7. WR (Write): The result of ALU and cache operations are written back to the register file.

Table 987 lists the cycles per instruction (assuming cache hit and no icc or load interlock):

*Table 987.*Instruction timing

Instruction	**Cycles (MMU disabled)**
JMPL, RETT	3
SMUL/UMUL	1/4*
SDIV/UDIV	35
Taken Trap	5
Atomic load/store	5
All other instructions	**1**

* Multiplication cycle count is 1 clock (1 clock issue rate, 2 clock data latency), for the 32x32 multiplier and 4 clocks (issue rate, 4/5 clocks data latency for standard/pipelined version) for the 16x16 version.

Additional events that affects instruction timing are listed below:

*Table 988.*Event timing

Event	**Cycles**
Instruction cache miss processing, MMU disabled	3 + mem latency
Instruction cache miss processing, MMU enabled	5 + mem latency
Data cache miss processing, MMU disabled (read), L2 hit	3 + mem latency
Data cache miss processing, MMU disabled (write), write-buffer empty	0
Data cache miss processing, MMU enabled (read)	5 + mem latency
Data cache miss processing, MMU enabled (write), write-buffer empty	0
MMU page table walk	10 + 3 * mem latency
Branch prediction miss, branch follows ICC setting	2
Branch prediction miss, one instruction between branch and ICC setting	1
Pipeline restart due to register file or cache error correction	7

69.2.3 SPARC Implementor’s ID

Aeroflex Gaisler is assigned number 15 (0xF) as SPARC implementor’s identification. This value is hard-coded into bits 31:28 in the %psr register. The version number for LEON4 is 3 (same as for LEON3 due to software compatibility), which is hard-coded in to bits 27:24 of the %psr.

69.2.4 Divide instructions

Full support for SPARC V8 divide instructions is provided (SDIV, UDIV, SDIVCC & UDIVCC). The divide instructions perform a 64-by-32 bit divide and produce a 32-bit result. Rounding and overflow detection is performed as defined in the SPARC V8 standard.

69.2.5 Multiply instructions

The LEON processor supports the SPARC integer multiply instructions UMUL, SMUL UMULCC and SMULCC. These instructions perform a 32x32-bit integer multiply, producing a 64-bit result. SMUL and SMULCC performs signed multiply while UMUL and UMULCC performs unsigned multiply. UMULCC and SMULCC also set the condition codes to reflect the result. The multiply instructions are performed using a 32x32 pipelined hardware multiplier, or a 16x16 hardware multiplier which is iterated four times. To improve the timing, the 16x16 multiplier can optionally be provided with a pipeline stage.

69.2.6 Multiply and accumulate instructions

To accelerate DSP algorithms, two multiply&accumulate instructions are implemented: UMAC and SMAC. The UMAC performs an unsigned 16-bit multiply, producing a 32-bit result, and adds the result to a 40-bit accumulator made up by the 8 lsb bits from the %y register and the %asr18 register. The least significant 32 bits are also written to the destination register. SMAC works similarly but performs signed multiply and accumulate. The MAC instructions execute in one clock but have two clocks latency, meaning that one pipeline stall cycle will be inserted if the following instruction uses the destination register of the MAC as a source operand.

Assembler syntax:

```
    umac  rs1, reg_imm, rd
    smac  rs1, reg_imm, rd
```

Operation:

```
prod[31:0] = rs1[15:0] * reg_imm[15:0]
result[39:0] = (Y[7:0] & %asr18[31:0]) + prod[31:0]
(Y[7:0] & %asr18[31:0]) = result[39:0]
rd = result[31:0]
```

%asr18 can be read and written using the RDASR and WRASR instructions.

69.2.7 Compare and Swap instruction (CASA)

LEON4 implements the SPARC V9 Compare and Swap Alternative (CASA) instruction. The CASA operates as described in the SPARC V9 manual and will be enabled in implementations that have the integer load delay set to 1. The instruction is privileged, except when setting ASI = 0xA (user data).

69.2.8 Branch prediction

LEON4 implements branch-always speculative execution, potentially saving 1 - 2 clocks if the %psr.icc field was updated in the two instructions preceding a conditional branch. On miss-prediction, no extra instruction delay is incurred.

69.2.9 Register file data protection

The integer and FPU register files can optionally be protected against soft errors using triple modular redundancy TMR. Data errors will then be transparently corrected without impact at application level. DMR can correct up to four errors per 32-bit register while TMR can correct any number of errors. The protection scheme is enabled through the FT generic.

69.2.10 Hardware breakpoints

The integer unit can be configured to include up to four hardware breakpoints. Each breakpoint consists of a pair of application-specific registers (%asr24/25, %asr26/27, %asr28/29 and %asr30/31) registers; one with the break address and one with a mask:

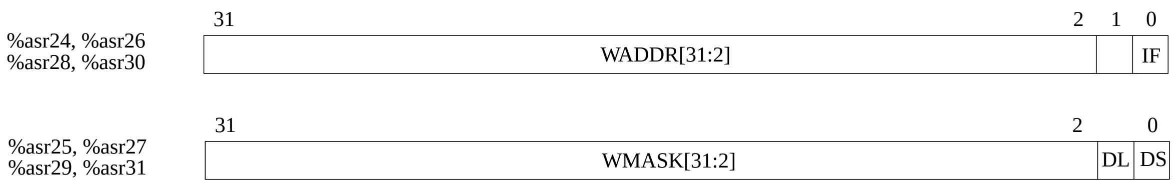


Figure 200. Watch-point registers

Any binary aligned address range can be watched - the range is defined by the WADDR field, masked by the WMASK field (WMASK[x] = 1 enables comparison). On a breakpoint hit, trap 0x0B is generated. By setting the IF, DL and DS bits, a hit can be generated on instruction fetch, data load or data store. Clearing these three bits will effectively disable the breakpoint function.

69.2.11 Instruction trace buffer

The instruction trace buffer consists of a circular buffer that stores executed instructions. The trace buffer operation is controlled through the debug support interface, and does not affect processor operation (see the DSU description). The size of the trace buffer is configurable from 1 to 64 kB through a VHDL generic. The trace buffer is 128 bits wide, and stores the following information:

- Instruction address and opcode
- Instruction result
- Load/store data and address
- Trap information
- 30-bit time tag

The operation and control of the trace buffer is further described in section 26.4. Note that in multiprocessor systems, each processor has its own trace buffer allowing simultaneous tracing of all instruction streams.

69.2.12 Processor configuration register

The application specific register 17 (%asr17) provides information on how various configuration options were set during synthesis. This can be used to enhance the performance of software, or to support enumeration in multi-processor systems. The register can be accessed through the RDASR instruction, and has the following layout:

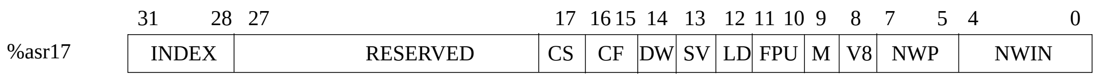


Figure 201. LEON4 configuration register (%asr17)

Field Definitions:

[31:28]:	Processor index. In multi-processor systems, each LEON core gets a unique index to support enumeration. The value in this field is identical to the *hindex* generic parameter in the VHDL model if *smp* = 1, or from the irqi.index signal if *smp* = 16.
[17]:	Clock switching enabled (CS). If set switching between AHB and CPU frequency is available.
[16:15]:	CPU clock frequency (CF). CPU core runs at (CF+1) times AHB frequency.
[14]:	Disable write error trap (DWT). When set, a write error trap (tt = 0x2b) will be ignored. Set to zero after reset.
[13]:	Single-vector trapping (SVT) enable. If set, will enable single-vector trapping. Fixed to zero if SVT is not implemented. Set to zero after reset.
[12]:	Load delay. If set, the pipeline uses a 2-cycle load delay. Otherwise, a 1-cycle load delay i s used. Generated from the *lddel* generic parameter in the VHDL model.
[11:10]:	FPU option. “00” = no FPU; “01” = GRFPU; “11” = GRFPU-Lite
[9]:	If set, the optional multiply-accumulate (MAC) instruction is available
[8]:	If set, the SPARC V8 multiply and divide instructions are available.
[7:5]:	Number of implemented watchpoints (0 - 4)
[4:0]:	Number of implemented registers windows corresponds to NWIN+1.

69.2.13 Exceptions

LEON4 adheres to the general SPARC trap model. The table below shows the implemented traps and their individual priority. When PSR (processor status register) bit ET=0, an exception trap causes the processor to halt execution and enter error mode, and the external error signal will then be asserted.

*Table 989.*Trap allocation and priority

Trap	TT	Pri	Description
reset	0x00	1	Power-on reset
write error	0x2b	2	write buffer error
instruction_access_error	0x01	3	Error during instruction fetch
illegal_instruction	0x02	5	UNIMP or other un-implemented instruction
privileged_instruction	0x03	4	Execution of privileged instruction in user mode
fp_disabled	0x04	6	FP instruction while FPU disabled
cp_disabled	0x24	6	CP instruction while Co-processor disabled
watchpoint_detected	0x0B	7	Hardware breakpoint match
window_overflow	0x05	8	SAVE into invalid window
window_underflow	0x06	8	RESTORE into invalid window
mem_address_not_aligned	0x07	10	Memory access to un-aligned address
fp_exception	0x08	11	FPU exception
cp_exception	0x28	11	Co-processor exception
data_access_exception	0x09	13	Access error during load or store instruction
tag_overflow	0x0A	14	Tagged arithmetic overflow
divide_exception	0x2A	15	Divide by zero
interrupt_level_1	0x11	31	Asynchronous interrupt 1
interrupt_level_2	0x12	30	Asynchronous interrupt 2
interrupt_level_3	0x13	29	Asynchronous interrupt 3
interrupt_level_4	0x14	28	Asynchronous interrupt 4
interrupt_level_5	0x15	27	Asynchronous interrupt 5
interrupt_level_6	0x16	26	Asynchronous interrupt 6
interrupt_level_7	0x17	25	Asynchronous interrupt 7
interrupt_level_8	0x18	24	Asynchronous interrupt 8
interrupt_level_9	0x19	23	Asynchronous interrupt 9
interrupt_level_10	0x1A	22	Asynchronous interrupt 10
interrupt_level_11	0x1B	21	Asynchronous interrupt 11
interrupt_level_12	0x1C	20	Asynchronous interrupt 12
interrupt_level_13	0x1D	19	Asynchronous interrupt 13
interrupt_level_14	0x1E	18	Asynchronous interrupt 14
interrupt_level_15	0x1F	17	Asynchronous interrupt 15
trap_instruction	0x80 - 0xFF	16	Software trap instruction (TA)

69.2.14 Single vector trapping (SVT)

Single-vector trapping (SVT) is an SPARC V8e option to reduce code size for embedded applications. When enabled, any taken trap will always jump to the reset trap handler (%tbr.tba + 0). The trap type will be indicated in %tbr.tt, and must be decoded by the shared trap handler. SVT is enabled by setting bit 13 in %asr17. The model must also be configured with the SVT generic = 1.

69.2.15 Address space identifiers (ASI)

In addition to the address, a SPARC processor also generates an 8-bit address space identifier (ASI), providing up to 256 separate, 32-bit address spaces. During normal operation, the LEON4 processor accesses instructions and data using ASI 0x8 - 0xB as defined in the SPARC standard. Using the LDA/STA instructions, alternative address spaces can be accessed. The table shows the ASI usage for LEON. Only ASI[5:0] are used for the mapping, ASI[7:6] have no influence on operation.

*Table 990.*ASI usage

ASI	Usage
0x01	Forced cache miss
0x02	System control registers (cache control register)
0x08, 0x09, 0x0A, 0x0B	Normal cached access (replace if cacheable)
0x0C	Instruction cache tags (requires that i-cache is disabled)
0x0D	Instruction cache data (requires that i-cache is disabled)
0x0E	Data cache tags
0x0F	Data cache data
0x10	Flush instruction cache (and also data cache when system is implemented with MMU)
0x11	Flush data cache

69.2.16 Power-down

The processor can be configured to include a power-down feature to minimize power consumption during idle periods. The power-down mode is entered by performing a WRASR %asr19 instruction. The data value written to %asr19 must be zero to ensure compatibility with future extensions of the processor. During power-down, the pipeline is halted until the next interrupt occurs. Signals inside the processor pipeline and caches are then static, reducing power consumption from dynamic switching.

69.2.17 Processor reset operation

The processor is reset by asserting the RESET input for at least 4 clock cycles. The following table indicates the reset values of the registers which are affected by the reset. All other registers maintain their value (or are undefined).

*Table 991.*Processor reset values

Register	Reset value
TBR (trap base register)	TBA = 0x0 (set to reset start address)
PC (program counter)	0x0
nPC (next program counter)	0x4
PSR (processor status register)	ET=0, S=1

By default, the execution will start from address 0. This can be overridden by setting the RSTADDR generic in the model to a non-zero value. The reset address is always aligned on a 4 kbyte boundary. If RSTADDR is set to 16#FFFFF#, then the reset address is taken from the signal IRQI.RSTVEC. This allows the reset address to be changed dynamically. The processor will also use the (possibly dynamic) reset address when it is in error mode and the IRQI.RST signal is asserted.

69.2.18 Multi-processor support

The LEON4 processor supports symmetric multi-processing (SMP) configurations, with up to 16 processors attached to the same AHB bus. In multi-processor systems, only the first processor will start. All other processors will remain halted in power-down mode. After the system has been initialized, the remaining processors can be started by writing to the 'MP status register', located in the multi-processor interrupt controller. The halted processors start executing from the reset address (0 or

RSTADDR generic). Enabling SMP is done by setting the *smp* generic to 1 or higher. Cache snooping should always be enabled in SMP systems to maintain data cache coherency between the processors.

69.2.19 Cache sub-system

The LEON4 processor implements a Harvard architecture with separate instruction and data buses, connected to two independent cache controllers. Both instruction and data cache controllers can be separately configured to implement a direct-mapped cache or a multi-way cache with an associativity of 2 - 4. The way size is configurable to 1 - 256 kbyte, divided into cache lines with 16 or 32 bytes of data. In multi-way configurations, one of two replacement policies can be selected: least-recently-used (LRU) or (pseudo-) random. The LRU hardware configuration also supports LRR and random replacement configurable through software.

The LRU algorithm needs extra flip-flops per cache line to store access history. The random replacement algorithm is implemented through modulo-N counter that selects which line to evict on cache miss.

Cachability for both caches is controlled through the AHB plug&play address information. The memory mapping for each AHB slave indicates whether the area is cachable, and this information is used to (statically) determine which accesses that will be treated as cacheable. This approach means that the cachability mapping is always coherent with the current AHB configuration. The AMBA plug&play cachability can be overridden using the CACHED generic. When this generic is not zero, it is treated as a 16-bit field, defining the cachability of each 256 Mbyte address block on the AMBA bus. A value of 16#00F3# will thus define cachable areas in 0 - 0x20000000 and 0x40000000 - 0x80000000.

The type of AMBA accesses used, and supported by the processor, for a memory area depends on the area's cachability and the values of the WBMASK and BUSW VHDL generics, see section 69.5.8 for more information.

69.2.20 AHB bus interface

The LEON4 processor uses one AHB master interface for all data and instruction accesses. Instructions are fetched with incremental bursts.

Data is accessed using byte, half-word, word, double-word and quad-word accesses, depending on the width of the AMBA bus. See section 69.2.20 for a description of cache behaviour. The HPROT signals of the AHB bus are driven to indicate if the accesses is instruction or data, and if it is a user or supervisor access.

69.3 Instruction cache

69.3.1 Operation

The instruction cache can be configured as a direct-mapped cache or as a multi-way cache with associativity of 2 - 4 implementing either LRU or random replacement policy. The way size is configurable to 1 - 256 kbyte and divided into cache lines of 16 or 32 bytes. Each line has a cache tag associated with it consisting of a tag field and valid field with one valid bit for each cache line. To maintain backward compatibility with LEON3, the valid bit is replicated when the tag is read to show one valid bit for each four-byte sub-block of the cache line. On an instruction cache miss to a cachable location, the full cache line is fetched from memory and the corresponding on-chip tag and data memories are updated. In a multi-way configuration, the line to be replaced is chosen according to the replacement policy. During cache line fill, the instructions are simultaneously forwarded to the processor (streaming). If a memory access error occurs during a line fetch, the corresponding valid bit in the cache tag will be cleared and an instruction access error trap (tt=0x1) will be generated.

69.3.2 Instruction cache tag

A instruction cache tag entry consists of several fields as shown in figure 202:

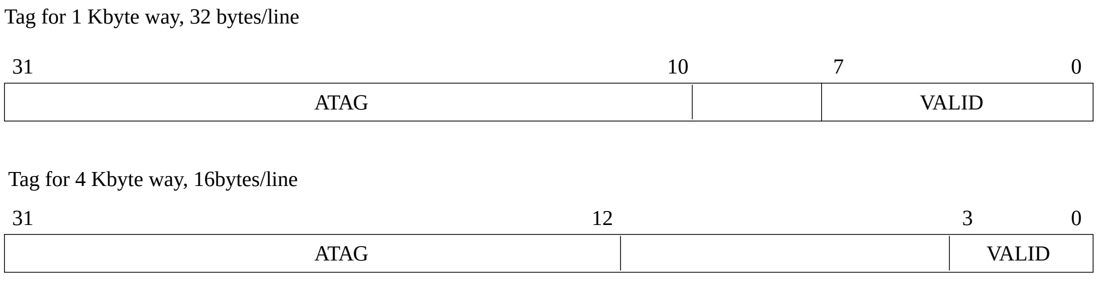

Tag for 1 Kbyte way, 32 bytes/line
31
10
7
0
ATAG
VALID
Tag for 4 Kbyte way, 16bytes/line
31
12
3
0
ATAG
VALID

Figure 202. Instruction cache tag layout examples

Field Definitions:

[31:10]:	Address Tag (ATAG) - Contains the tag address of the cache line.
[7:0]:	Valid (V) - When set, the cache line contains valid data. These bits are set when a line is filled due to a successful cache miss; a cache fill which results in a memory error will leave the valid bit unset. A FLUSH instruction will clear the valid bits.

NOTE: only the necessary bits will be implemented in the cache tag, depending on the cache configuration. As an example, a 4 kbyte cache with 16 bytes per line would only show four valid bits and have 20 tag bits. The cache rams are sized automatically by the ram generators in the model.

69.4 Data cache

69.4.1 Operation

The data cache can be configured as a direct-mapped cache or as a multi-way cache with associativity of 2 - 4 implementing either LRU or (pseudo-) random replacement policy. The way size is configurable to 1 - 64 kbyte and divided into cache lines of 16 or 32 bytes. Each line has a cache tag associated with it consisting of a tag field and a valid field with one valid bit for the entire cache line. To maintain backward compatibility with LEON3, the valid bit is replicated when the tag is read to show one valid bit for each four-byte sub-block of the cache line. On a data cache read-miss to a cachable location, the full cache line is loaded into the cache from main memory. The write policy for stores is write-through with no-allocate on write-miss. In a multi-way configuration, the line to be replaced on read-miss is chosen according to the replacement policy.

Locked AHB transfers are generated for LDSTUB, SWAP and CASA instructions. Locked transfers are always uncacheable.

If a memory access error occurs during a data load, the corresponding valid bit in the cache tag will be cleared, and a data access error trap (tt=0x9) will be generated.

69.4.2 Write buffer

The write buffer (WRB) consists of three 32-bit registers used to temporarily hold store data until it is sent to the destination device. For half-word or byte stores, the stored data replicated into proper byte alignment for writing to a word-addressed device, before being loaded into one of the WRB registers. The WRB is emptied prior to a load-miss cache-fill sequence to avoid any stale data from being read in to the data cache.

Since the processor executes in parallel with the write buffer, a write error will not cause an exception to the store instruction. Depending on memory and cache activity, the write cycle may not occur until several clock cycles after the store instructions has completed. If a write error occurs, the currently executing instruction will take trap 0x2b.

Note: the 0x2b trap handler should flush the data cache, since a write hit would update the cache while the memory would keep the old value due the write error.

69.4.3 Data cache tag

A data cache tag entry consists of several fields as shown in figure 203:

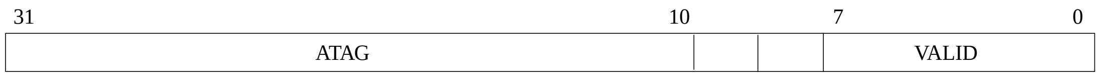


Figure 203. Data cache tag layout

Field Definitions:

[31:10]:	Address Tag (ATAG) - Contains the address of the data held in the cache line.
[7:0]:	Valid (V) - When set, the cache line contains valid data. These bits are set when a cache line is filled due to a successful cache miss; a cache fill which results in a memory error will leave the valid bit unset.

NOTE: only the necessary bits will be implemented in the cache tag, depending on the cache configuration. As an example, a 2 kbyte cache with 16 bytes per line would have four valid bits and 21 tag bits. The cache rams are sized automatically by the ram generators in the model.

69.5 Additional cache functionality

69.5.1 Cache flushing

Both instruction and data cache are flushed by executing the FLUSH instruction. The instruction cache is also flushed by setting the FI bit in the cache control register, while the data cache is also flushed by setting the FD bit in the cache control register. When the processor is implemented with an MMU, both I and D caches can be flushed by writing to any location with ASI=0x10.

Cache flushing takes one cycle per cache line, during which the IU will not be halted, but during which the caches are disabled. When the flush operation is completed, the cache will resume the state (disabled, enabled or frozen) indicated in the cache control register. Diagnostic access to the cache is not possible during a FLUSH operation and will cause a data exception (trap=0x09) if attempted.

69.5.2 Diagnostic cache access

Tags and data in the instruction and data cache can be accessed through ASI address space 0xC, 0xD, 0xE and 0xF by executing LDA and STA instructions. Address bits making up the cache offset will be used to index the tag to be accessed while the least significant bits of the bits making up the address tag will be used to index the cache set.

Diagnostic read of tags is possible by executing an LDA instruction with ASI=0xC for instruction cache tags and ASI=0xE for data cache tags. A cache line and set are indexed by the address bits making up the cache offset and the least significant bits of the address bits making up the address tag. Similarly, the data sub-blocks may be read by executing an LDA instruction with ASI=0xD for instruction cache data and ASI=0xF for data cache data. The sub-block to be read in the indexed cache line and set is selected by A[4:2].

The tags can be directly written by executing a STA instruction with ASI=0xC for the instruction cache tags and ASI=0xE for the data cache tags. The cache line and set are indexed by the address bits making up the cache offset and the least significant bits of the address bits making up the address tag. D[31:10] is written into the ATAG field (see above) and the valid bits are written with the D[7:0] of the write data. Bit D[9] is written into the LRR bit (if enabled) and D[8] is written into the lock bit (if enabled). The data sub-blocks can be directly written by executing a STA instruction with ASI=0xD for the instruction cache data and ASI=0xF for the data cache data. The sub-block to be written in the indexed cache line and set is selected by A[4:2].

In multi-way caches, the address of the tags and data of the ways are concatenated. The address of a tag or data is thus:

ADDRESS = WAY & LINE & DATA & “00”

Examples: the tag for line 2 in way 1 of a 2x4 Kbyte cache with 16 byte line would be:

A[13:12]	= 1	(WAY)
A[11:4]	= 2	(LINE)

=> TAG ADDRESS = 0x1020

The data of this line would be at addresses 0x1020 - 0x102C.

Note that diagnostic accesses to the instruction cache requires that the instruction cache is disabled (ICS field in cache control register “00”).

69.5.3 Data Cache snooping

To keep the data cache synchronized with external memory, cache snooping can be enabled through the *dsnoop* generic. When enabled, the data cache monitors write accesses on the AHB bus to cacheable locations. If an other AHB master writes to a cacheable location which is currently cached in the data cache, the corresponding cache line is marked as invalid. Snooping can be implemented in two different ways:

1. Using dual-port data tag RAMs. A snoop hit will clear the valid bits in the corresponding tag.
2. Using two 2-port RAMs for the data tags, one for virtual addresses and one for physical.

Option 1 is suitable for systems that do not require snooping when the MMU is enabled, and which do not contain AHB slaves capable of zero-waitstates write access. Option 2 allows snooping under all conditions, but requires separate snoop tags.

69.5.4 Cache memory data protection

The cache memories (tags and data) can optionally be protected agains soft errors using byte-parity codes. On a detected parity error, the corresponding cache (I or D) will be flushed and the data will be refetched from external memory. This is done transparently to execution, and incur the same timing penalty as a regular cache miss. Enabling of the data protection is done through the FT generic.

69.5.5 Local instruction and data RAM

Local instruction and data RAM is currently not supported by LEON4 at this time. The VHDL generics to enable these features are present in the LEON4 component declaration, but are unused.

69.5.6 Cache Control Register

The operation of the instruction and data caches is controlled through a common Cache Control Register (CCR) (table 992). Each cache can be in one of three modes: disabled, enabled and frozen. If disabled, no cache operation is performed and load and store requests are passed directly to the memory controller. If enabled, the cache operates as described above. In the frozen state, the cache is accessed and kept in sync with the main memory as if it was enabled, but no new lines are allocated on read misses.

Table 992. LEON4 Cache Control Register (CCR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
			PS	TB				DS	FD	FI	FT			ST	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IP	DP	ITE		IDE		DTE		DDE		DF	IF	DCS		ICS	

31:29	Reserved for future implementations
28	Parity Select (PS) - if set diagnostic read will return 4 check bits in the lsb bits, otherwise tag or data word is returned. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
27:24	Test Bits (TB) - if set, check bits will be xored with test bits TB during diagnostic write. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
23	Data cache snoop enable (DS) - if set, will enable data cache snooping.
22	Flush data cache (FD). If set, will flush the instruction cache. Always reads as zero.
21	Flush Instruction cache (FI). If set, will flush the instruction cache. Always reads as zero.
20:19	FT scheme (FT) - “00” = no FT, “01” = byte-parity checking implemented
18	Reserved for future implementations
17	Separate snoop tags (ST). This read-only bit is set if separate snoop tags are implemented.
16	Reserved
15	Instruction cache flush pending (IP). This bit is set when an instruction cache flush operation is in progress
14	Data cache flush pending (DP). This bit is set when an data cache flush operation is in progress.
13:12	Instruction Tag Errors (ITE) - Number of detected parity errors in the instruction tag cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
11:10	Instruction Data Errors (IDE) - Number of detected parity errors in the instruction data cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
9:8	Data Tag Errors (DTE) - Number of detected parity errors in the data tag cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
7:6	Data Data Errors (DDE) - Number of detected parity errors in the data data cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
5	Data Cache Freeze on Interrupt (DF) - If set, the data cache will automatically be frozen when an asynchronous interrupt is taken.
4	Instruction Cache Freeze on Interrupt (IF) - If set, the instruction cache will automatically be frozen when an asynchronous interrupt is taken.
3:2	Data Cache state (DCS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.
1:0	Instruction Cache state (ICS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.

If the DF or IF bit is set, the corresponding cache will be frozen when an asynchronous interrupt is taken. This can be beneficial in real-time system to allow a more accurate calculation of worst-case execution time for a code segment. The execution of the interrupt handler will not evict any cache lines and when control is returned to the interrupted task, the cache state is identical to what it was before the interrupt. If a cache has been frozen by an interrupt, it can only be enabled again by enabling it in the CCR. This is typically done at the end of the interrupt handler before control is returned to the interrupted task.

69.5.7 Cache configuration registers

The configuration of the two caches if defined in two registers: the instruction and data configuration registers. These registers are read-only, except for the REPL field, and indicate the size and configuration of the caches.

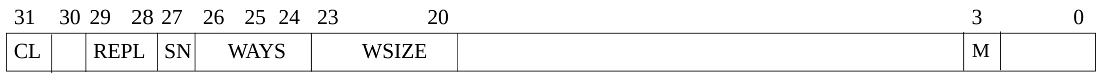


Figure 204. Cache configuration register

[31]:	Cache locking (CL). Set if cache locking is implemented.
[29:28]:	Cache replacement policy (REPL). 00 - reserved, 01 - least recently used (LRU), 10 - least recently replaced (LRR), 11 - random. This field is writable when LRU policy is implemented.
[27]:	Cache snooping (SN). Set if snooping is implemented.
[26:24]:	Cache associativity (WAYS). Number of ways in the cache: 000 - direct mapped, 001 - 2-way associative, 010 - 3-way associative, 011 - 4-way associative
[23:20]:	Way size (WSIZE). Indicates the size (Kbytes) of each cache way. Size = 2^{SIZE}
[18:16]:	Line size (LSIZE). Indicated the size (words) of each cache line. Line size = 2^{LSZ}
[3]:	MMU present. This bit is set to '1' if an MMU is present.

All cache registers are accessed through load/store operations to the alternate address space (LDA/STA), using ASI = 2. The table below shows the register addresses:

*Table 993.*ASI 2 (system registers) address map

Address	**Register**
0x00	Cache control register
0x04	Reserved
0x08	Instruction cache configuration register
0x0C	Data cache configuration register

69.5.8 AMBA AHB interface

The LEON4 processor has one AHB master interface. The types of AMBA accesses supported and performed by the processor depend on the accessed memory area's cachability, the maximum bus width, if the corresponding cache is enabled, and if the accessed memory area has been marked as being on the wide bus.

Cacheable instructions are fetched with a burst of 32-bit accesses, or 64- or 128-bit accesses depending on the cache line size and the AHB bus width.

Cacheable data is fetched in a burst of 64- or 128-bit accesses, depending on the cache line size and AHB bus width. Data access to uncacheable areas may only be done with 8-, 16- and 32-bit accesses, i.e. the LDD and STD instructions may not be used. If an area is marked as cacheable then the data cache will automatically try to use 64- or 128-bit accesses. This means that if a slave does not support 64- or 128-bit accesses and is mapped as cacheable then software should only perform data acceesses with forced cache miss and no 64-bit loads (LDD) when accessing the slave. One example of how to use forced cache miss for loads is given by the following function:

```
static inline int load(int addr)
{
    int tmp;
    asm volatile(" lda [%1]1, %0 "
        : "=r"(tmp)
        : "r"(addr)
        );
    return tmp;
}
```

The area which supports 64- or 128-bit access is indicated in the WBMASK generic. This generic is treated as a 16-bit field, defining the 64/128-bit capability of each 256 Mbyte address block on the AMBA bus. A value of 16#00F3# will thus define areas in 0 - 0x20000000 and 0x40000000 - 0x80000000 to be 64/128-bit capable. The maximum access size to be used in the area(s) marked with WBMASK is determined by the BUSW generic.

Store instructions result in a AMBA access with size corresponding to the executed instruction, 64-bit store instructions (STD) are always translated to 64-bit accesses (never converted into two 32-bit stores as is done for LEON3). The table below indicates the access types used for instruction and data accesses depending on cachability, wide bus mask (wbmask), and cache configuration.

Processor operation	**Accessed memory area is 32-bit only, wbmask(address) = 0**			**Accessed memory area is on wide bus wbmask(address) = 1**		
	Area not cacheable[1]	**Area is cacheable**[1]		**Area not cacheable**[1]	**Area is cacheable**[1]	
		Cache enabled[2]	**Cache disabled**		**Cache enabled**[2]	**Cache disabled**
Instruction fetch	Burst of 32-bit read accesses			Burst of 64- or 128-bit accesses[5]		
Data load <= 32-bit	Read access with size specified by load instruction	**Illegal**[3,6] Burst of 32-bit accesses, software may get incorrect data	Read access with size specified by load instruction	Read access with size specified by load instruction	Burst of 64- or 128-bit accesses[5]	Read access with size specified by load instruction
Data load 64-bit (LDD)	**Illegal**[4] Single 64-bit access will be performed	**Illegal**[3] Burst of 64- or 128-bit accesses[5]	**Illegal**[3] Single 64-bit access will be performed	**Illegal**[4] Single 64-bit access will be performed	Burst of 64- or 128-bit accesses[5]	Single 64-bit read access
Data store <= 32-bit	Store access with size specified by store instruction.					
Data store 64-bit (STD)	**Illegal (64-bit store performed to 32-bit area)** 64-bit store access will be performed.			64-bit store access		

[1] Cachability is determined by CACHED generic, if CACHED is zero then cachability is determined via AMBA PnP.

[2] Bus accesses for reads will only be made on L1 cache miss or on load with forced cache miss.

[3] LEON4 is designed to always make use of wide bus accesses for cacheable data. Cacheable data can only be handled with 64- or 128 bit accesses.

[4] Data accesses to uncachable areas may only be done with 8-, 16- and 32-bit accesses.

[5] 64- or 128-bit accesses depending on BUSW generic.

[6] Loads with forced cache miss can be used to perform single accesses to cachable slaves in 32-bit memory area.

In the event of AMBA ERROR responses the cache line will not be written to the cache. If software accessed the memory address that resulted in an AMBA ERROR response then a trap will be generated

69.5.9 Software consideration

After reset, the caches are disabled and the cache control register (CCR) is 0. Before the caches may be enabled, a flush operation must be performed to initialized (clear) the tags and valid bits. A suitable assembly sequence could be:

```
flush
set 0x81000f, %g1
sta %g1, [%g0] 2
```

69.6 Memory management unit

A memory management unit (MMU) compatible with the SPARC V8 reference MMU can optionally be configured. For details on operation, see the SPARC V8 manual.

69.6.1 ASI mappings

When the MMU is used, the following ASI mappings are added:

*Table 994.*MMU ASI usage

ASI	Usage
0x10	Flush I and D cache
0x14	MMU diagnostic dcache context access
0x15	MMU diagnostic icache context access (requires that i-icache is disabled)
0x18	Flush TLB and I/D cache
0x19	MMU registers
0x1C	MMU bypass
0x1D	MMU diagnostic access
0x1E	MMU snoop tags diagnostic access

69.6.2 MMU/Cache operation

When the MMU is disabled, the caches operate as normal with physical address mapping. When the MMU is enabled, the caches tags store the virtual address and also include an 8-bit context field. If cache snooping is desired, bit 2 of the *dsnoop* generic has to be set. This will store the physical tag in a separate RAM block, which then is used for snooping. In addition, the size of each data cache way has to be smaller or equal to 4 kbyte (MMU page size). This is necessary to avoid aliasing in the cache since the virtual tags are indexed with a virtual offset while the physical tags are indexed with a physical offset. Physical tags and snoop support is needed for SMP systems using the MMU (linux-2.6).

Because the cache is virtually tagged, no extra clock cycles are needed in case of a cache load hit. In case of a cache miss, at least 2 extra clock cycles are used if there is a TLB hit. If there is a TLB miss the page table must be traversed, resulting in up to 4 AMBA read accesses and one possible writeback operation. If a combined TLB is used by the instruction cache, the translation is stalled until the TLB is free. If fast TLB operation is selected (tlb_type = 2), the TLB will be accessed simultaneously with tag access, saving 2 clocks on cache miss. This will increase the area somewhat, and may reduce the timing, but usually results in better overall throughput.

69.6.3 MMU registers

The following MMU registers are implemented:

*Table 995.*MMU registers (ASI = 0x19)

Address	Register
0x000	MMU control register
0x100	Context pointer register
0x200	Context register
0x300	Fault status register
0x400	Fault address register

The MMU control register layout can be seen below, while the definition of the remaining MMU registers can be found in the SPARC V8 manual.

31 28	27 24	23 21	20 18	17 16	15	14	2	1	0
IMPL	VER	ITLB	DTLB	00	TD	ST	RESERVED	NF	E

Figure 205. MMU control register

[31:28]: MMU Implementation ID. Hardcoded to “0000”.
[27:24]: MMU Version ID. Hardcoded to “0000”.
[23:21]: Number of ITLB entries. The number of ITLB entries is calculated as 2^{ITLB}. If the TLB is shared between instructions and data, this field indicates to total number of TLBs.
[20:18]: Number of DTLB entries. The number of DTLB entries is calculated as 2^{DTLB}. If the TLB is shared between instructions and data, this field is zero.
[15]: TLB disable. When set to 1, the TLB will be disabled and each data access will generate an MMU page table walk.
[14]: Separate TLB. This bit is set to 1 if separate instruction and data TLBs are implemented.
[1]: No Fault. When NF= 0, any fault detected by the MMU causes FSR and FAR to be updated and causes a fault to be generated to the processor. When NF= 1, a fault on an access to ASI 9 is handled as when NF= 0; a fault on an access to any other ASI causes FSR and FAR to be updated but no fault is generated to the processor.
[0]: Enable MMU. 0 = MMU disabled, 1 = MMU enabled.

69.6.4 Translation look-aside buffer (TLB)

The MMU can be configured to use a shared TLB, or separate TLB for instructions and data. The number of TLB entries (for each implemented TLB) can be set to 2 - 64 in the configuration record. The organisation of the TLB and number of entries is not visible to the software and does thus not require any modification to the operating system.

69.6.5 Snoop tag diagnostic access

If the MMU has been configured to use separate snoop tags, they can be accessed via ASI 0x1E. This is primarily useful for RAM testing, and should not be performed during normal operation. The figure below shows the layout of the snoop tag for a 1 Kbyte data cache:

31 10	9 2	1	0
ATAG	“0000”	PAR	HIT

Figure 206. Snoop cache tag layout

[31:10] Address tag. The physical address tag of the cache line.
[1]: Parity. The odd parity over the data tag.
[0]: Snoop hit. When set, the cache line is not valid and will cause a cache miss if accessed by the processor.

69.7 Floating-point unit and custom co-processor interface

The SPARC V8 architecture defines two (optional) co-processors: one floating-point unit (FPU) and one user-defined co-processor. The LEON4 pipeline provides an interface port for Aeroflex Gaisler’s GRFPU and GRFPU-Lite. The characteristics of the two FPU’s are described in the next sections.

69.7.1 Aeroflex Gaisler’s floating-point unit (GRFPU)

The high-performance GRFPU operates on single- and double-precision operands, and implements all SPARC V8 FPU instructions. The FPU is interfaced to the LEON4 pipeline using a LEON4-specific FPU controller (GRFPC) that allows FPU instructions to be executed simultaneously with integer instructions. Only in case of a data or resource dependency is the integer pipeline held. The GRFPU is fully pipelined and allows the start of one instruction each clock cycle, with the exception is FDIV and FSQRT which can only be executed one at a time. The FDIV and FSQRT are however executed in a separate divide unit and do not block the FPU from performing all other operations in parallel.

All instructions except FDIV and FSQRT has a latency of three cycles, but to improve timing, the LEON4 FPU controller inserts an extra pipeline stage in the result forwarding path. This results in a latency of four clock cycles at instruction level. The table below shows the GRFPU instruction timing when used together with GRFPC:

*Table 996.*GRFPU instruction timing with GRFPC

Instruction	**Throughput**	**Latency**
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FSMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	1	4
FDIVS	14	16
FDIVD	15	17
FSQRTS	22	24
FSQRTD	23	25

The GRFPC controller implements the SPARC deferred trap model, and the FPU trap queue (FQ) can contain up to 7 queued instructions when an FPU exception is taken. When the GRFPU is enabled in the model, the version field in %fsr has the value of 2. When the GRFPU/FPC are enabled, the processor pipeline is effectively extended to 8 stages. This however not visible to software and does not impact integer operations.

69.7.2 GRFPU-Lite

GRFPU-Lite is a smaller version of GRFPU, suitable for FPGA implementations with limited logic resources. The GRFPU-Lite is not pipelined and executes thus only one instruction at a time. To improve performance, the FPU controller (GRLFPC) allows GRFPU-Lite to execute in parallel with the processor pipeline as long as no new FPU instructions are pending. Below is a table of worst-case throughput of the GRFPU-Lite:

*Table 997.*GRFPU-Lite worst-case instruction timing with GRLFPC

Instruction	**Throughput**	**Latency**
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FSMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	8	8
FDIVS	31	31
FDIVD	57	57
FSQRTS	46	46
FSQRTD	65	65

When the GRFPU-Lite is enabled in the model, the version field in %fsr has the value of 3.

69.8 Vendor and device identifiers

The core has vendor identifiers 0x01 (Aeroflex Gaisler) and device identifiers 0x048. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

69.9 Implementation

69.9.1 Technology mapping

LEON4 has two technology mapping generics, *fabtech* and *memtech*. The *fabtech* generic controls the implementation of some pipeline features, while *memtech* selects which memory blocks will be used to implement cache memories and the IU/FPU register file. *Fabtech* can be set to any of the provided technologies (0 - NTECH) as defined in the GRPIB.TECH package. See the GRLIB Users's Manual for available settings for *memtech*.

69.9.2 RAM usage

The LEON4 core maps all usage of RAM memory on the *syncram*, *syncram_2p* and *syncram_dp* components from the technology mapping library (TECHMAP). The type, configuration and number of RAM blocks is described below.

Register file

The register file is implemented with six *synram_2p* blocks for all technologies where the *regfile_3p_infer* constant in TECHMAP.GENCOMP is set to 0. The organization of the syncram_2p is shown in the following table:

*Table 998.*syncram_2p sizes for LEON4 register file

Register windows	Syncram_2p organization
2 - 3	32x32
4 - 7	64x32
8 - 15	128x32
16-31	256x31
32	512x32

If *regfile_3p_infer* is set to 1, the synthesis tool will automatically infer the register. On FPGA technologies, it can be in either flip-flops or RAM cells, depending on the tool and technology. On ASIC technologies, it will be flip-flops. The amount of flip-flops inferred is equal to the number of registers:

Number of flip-flops = ((NWINDOWS *16) + 8) * 32

FP register file

If FPU support is enabled, the FP register file is implemented with four *synram_2p* blocks when the *regfile_3p_infer* constant in TECHMAP.GENCOMP is set to 0. The organization of the syncram_2p blocks is 16x32.

If *regfile_3p_infer* is set to 1, the synthesis tool will automatically infer the FP register file. For ASIC technologies the number of inferred flip-flops is equal to number of bits in the FP register file which is 32 * 32 = 1024.

Cache memories

RAM blocks are used to implement the cache tags and data memories. Depending on cache configuration, different types and sizes of RAM blocks are used.

The tag memory is implemented with one *syncram* per cache way when no snooping is enabled. The tag memory depth and width is calculated as follows:

Depth = (cache way size in bytes) / (cache line size in bytes)

Width = 32 - log2(cache way size in bytes) + (cache line size in bytes)/4 + lrr + lock

For a 2 Kbyte cache way with 32 bytes/line, the tag RAM depth will be (2048/32) = 64. The width will be: 32 - log2(2048) + 1 = 32 - 11 + 1 = 22. The tag RAM organization will thus be 64x22 for the configuration. If the MMU is enabled, the tag memory width will increase with 8 to store the process context ID, and the above configuration will us a 64x30 RAM.

If snooping is enabled, the tag memories will be implemented using the *syncram_dp* component (dual-port RAM). One port will be used by the processor for cache access/refill, while the other port will be used by the snooping and invalidation logic. The size of the *syncram_dp* block will be the same as when snooping is disabled. If physical snooping is enabled (separate snoop tags), one extra RAM block per data way will be instatiated to hold the physical tags. The width of the RAM block will be the same as the tag address: 32 - log2(way size). A 4 Kbyte data cache way will thus require a 32 - 12 = 20 bit wide RAM block for the physical tags. If fast snooping is enabled, the tag RAM (virtual and physical) will be implemented using *syncram_2p* instead of *syncram_dp*. This can be used to implement snooping on technologies which lack dual-port RAM but have 2-port RAM.

The data part of the caches (storing instructions or data) is either 64 or 128 bit wide, depending on the setting of the BUSW generic. The depth is equal to the way size in bytes, divided by 8 (BUSW=64) or 16 (BUSW=128). A 64-bit cache way of 4 Kbyte will use two *syncram* components with and organization of 512x32. If the 128-bit AHB bus option is used, the data RAM will be divided on four 32-bit RAM blocks to allow loading of a 16-bit cache line in one clock. A 4 Kbyte data cache will then use four 256x32 RAM blocks.

Instruction Trace buffer

The instruction trace buffer will use four identical RAM blocks (*syncram*) to implement the buffer memory. The syncrams will always be 32-bit wide. The depth will depend on the TBUF generic, which indicates the total size of trace buffer in Kbytes. If TBUF = 1 (1 Kbyte), then four RAM blocks of 64x32 will be used. If TBUF = 2, then the RAM blocks will be 128x32 and so on.

69.9.3 Double clocking

The LEON4 CPU core be clocked at twice the clock speed of the AMBA AHB bus. When clocked at double AHB clock frequency, all CPU core parts including integer unit and caches will operate at double AHB clock frequency while the AHB bus access is performed at the slower AHB clock frequency. The two clocks have to be synchronous and a multicycle paths between the two clock domains have to be defined at synthesis tool level. A separate component (LEON4s2x) is provided for the double clocked core. Double clocked versions of DSU (dsu4_2x) and MP interrupt controller (irqmp2x) are used in a double clocked LEON4 system. An AHB clock qualifier signal (*clken* input) is used to identify end of AHB cycle. The AHB qualifier signal is generated in CPU clock domain and is high during the last CPU clock cycle under AHB clock low-phase. Sample *LEON4-clk2x* design provides a module that generates an AHB clock qualifier signal.

Double-clocked design has two clock domains: AMBA clock domains (HCLK) and CPU clock domain (CPUCLK). LEON4 (LEON4s2x component) and DSU4 (dsu4_2x) belong to CPU clock domain (clocked by CPUCLK), while the rest of the system is in AMBA clock domain (clocked by HCLK). Paths between the two clock domains (paths starting in CPUCLK domain and ending in

HCLK and paths starting in HCLK domain and ending in CPUCLK domain) are multicycle paths with propagation time of two CPUCLK periods (or one HCLK period) with following exceptions:

Start point	**Through**	**End point**	**Propagation time**
LEON4s2x core			
CPUCLK	ahbi	CPUCLK	2 CPUCLK
CPUCLK	ahbsi	CPUCLK	2 CPUCLK
CPUCLK	ahbso	CPUCLK	2 CPUCLK
HCLK	irqi	CPUCLK	1 CPUCLK
CPUCLK	irqo	HCLK	1 CPUCLK
CPUCLK		u0_0/p0/c0/sync0/r[*] (register)	1 CPUCLK

Start point	**Through**	**End point**	**Propagation time**
dsu4_2x core			
CPUCLK	ahbmi	CPUCLK	2 CPUCLK
CPUCLK	ahbsi	CPUCLK	2 CPUCLK
	dsui	CPUCLK	1 CPUCLK
r[*] (register)		rh[*] (register)	1 CPUCLK
irqmp2x core			
r2[*] (register)		r[*] (register)	1 CPUCLK

69.9.4 Clock gating

To further reduce the power consumption of the processor, the clock can be gated-off when the processor has entered power-down state. Since the cache controllers and MMU operate in parallel with the processor, the clock cannot be gated immediately when the processor has entered the power-down state. Instead, a power-down signal (DBGO.idle) is generated when all outstanding AHB accesses have been completed and it is safe to gate the clock. This signal should be clocked though a positive-edge flip-flop followed by a negative-edge flip-flop to guarantee that the clock is gated off during the clock-low phase. To insure proper start-up state, the clock should not be gated during reset.

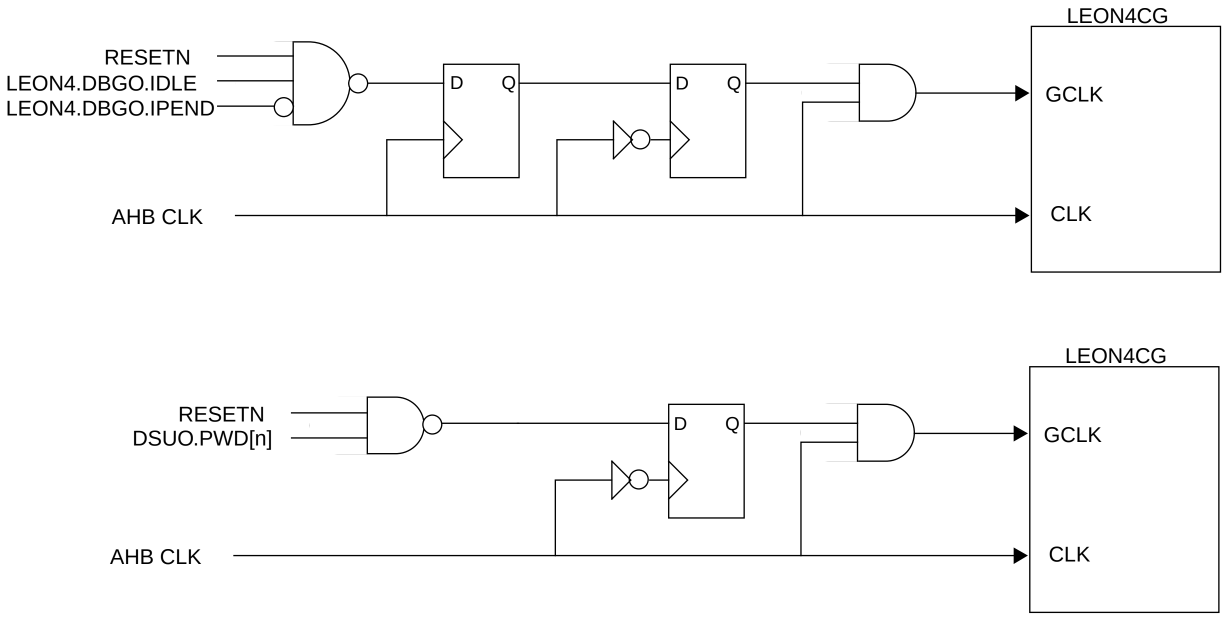


Figure 207. Examples of LEON4 clock gating

The processor should exit the power-down state when an interrupt become pending. The signal DBGO.ipend will then go high when this happen, and should be used to re-enable the clock.

When the debug support unit (DSU4) is used, the DSUO.pwd signal should be used instead of DBGO.idle. This will insure that the clock also is re-enabled when the processor is switched from power-down to debug state by the DSU. The DSUO.pwd is a vector with one power-down signal per CPU (for SMP systems). DSUO.pwd takes DBGO.ipend into account, and no further gating or latching needs to be done of this signal. If cache snooping has been enabled, the continuous clock will insure that the snooping logic is activated when necessary and will keep the data cache synchronized even when the processor clock is gated-off. In a multi-processor system, all processor except node 0 will enter power-down after reset and will allow immediate clock-gating without additional software support.

Clock-tree routing must insure that the continuous clock (CLK) and the gated clock (GCLK) are phase-aligned. The template design *leon4-clock-gating* shows an example of a clock-gated system. The *leon4cg* entity should be used when clock gating is implemented. This entity has one input more (GCLK) which should be driven by the gated clock. Using the double-clocked version of LEON4 (leon4s2x), the GCLK2 is the gated double-clock while CLK and CLK2 should be continuous.

69.9.5 Scan support

If the SCANTEST generic is set to 1, support for scan testing is enabled. This will make use of the AHB scan support signals in the following manner: when AHBI.testen and AHBI.scanen are both ‘1’, the select signals to all RAM blocks (cache RAM, register file and DSU trace buffers) are disabled. This means that when the scan chain is shifted, no accidental write or read can occur in the RAM blocks. The scan signal AHBI.testrst is not used as there are no asynchronous resets in the LEON4 core.

69.10 Configuration options

Table 999 shows the configuration options of the core (VHDL generics).

*Table 999.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
fabtech	Target technology	0 - NTECH	0 (inferred)
memtech	Vendor library for regfile and cache RAMs	0 - NTECH	0 (inferred)
nwindows	Number of SPARC register windows. Choose 8 windows to be compatible with Bare-C and RTEMS cross-compilers.	2 - 32	8
dsu	Enable Debug Support Unit interface	0 - 1	0
fpu	Floating-point Unit 0 : no FPU 1 - 7: GRFPU 1 - inferred multiplier, 2 - DW multiplier, 3 - Module Generator multiplier 8 - 14: GRFPU-Lite 8 - simple FPC, 9 - data forwarding FPC, 10 - non-blocking FPC 16 - 31: as above (modulo 16) but use netlist 32 - 63: as above (modulo 32) but uses shared GRFPU interface	0 - 63	0
v8	Generate SPARC V8 MUL and DIV instructions This generic is assigned with the value: *mult + 4*struct* Where mult selects between the following implementation options for the multiplier and divider: 0 : No multiplier or divider 1 : 16x16 multiplier 2 : 16x16 pipelined multiplier 16#32# : 32x32 pipelined multiplier Where *struct* selects the structure option for the integer multiplier. The following structures can be selected: 0: Inferred by synthesis tool 1: Generated using Module Generators from NTNU 2: Using technology specific netlists (techspec) 3: Using Synopsys DesignWare (DW02_mult and DW_mult_pipe)	0 - 16#3F#	0
cp	Generate co-processor interface	0 -1	0
mac	Generate SPARC V8e SMAC/UMAC instruction	0 - 1	0
pclow	Least significant bit of PC (Program Counter) that is actually generated. PC[1:0] are always zero and are normally not generated. Generating PC[1:0] makes VHDL-debugging easier.	0, 2	2
notag	Disable tagged instructions.	0 - 1	0
nwp	Number of watchpoints	0 - 4	0
icen	Enable instruction cache	0 - 1	1
irepl	Instruction cache replacement policy. 0 - least recently used (LRU)/LRR/random, 2 - random only	0, 2	0
isets	Number of instruction cache ways	1 - 4	1
ilinesize	Instruction cache line size in number of words	4, 8	4
isetsize	Size of each instruction cache way in kByte	1 - 256	1
isetlock	Unused	0 - 1	0
dcen	Data cache enable	0 - 1	1
drepl	Data cache replacement policy. 0 - least recently used (LRU)/LRR/random, 2 - random only	0, 2	0

*Table 999.*Configuration options

Generic	Function	Allowed range	Default
dsets	Number of data cache ways	1 - 4	1
dlinesize	Data cache line size in number of words	4, 8	4
dsetsize	Size of each data cache way in kByte	1 - 256	1
dsetlock	Unused	0 - 1	0
dsnoop	Enable data cache snooping Bit 0-1: 0: disable, 1: slow, 2: fast (see text) Bit 2: 0: simple snooping, 1: separate tags	0 - 6	0
ilram	Enable local instruction RAM (not used at this point)	0 - 1	0
ilramsize	Local instruction RAM size in kB (not used at this point)	1 - 512	1
ilramstart	8 MSB bits used to decode local instruction RAM area (not used at this point)	0 - 255	16#8E#
dlram	Enable local data RAM (scratch-pad RAM) (not used at this point)	0 - 1	0
dlramsize	Local data RAM size in kB (not used at this point)	1 - 512	1
dlramstart	8 MSB bits used to decode local data RAM area (not used at this point)	0 - 255	16#8F#
mmuen	Enable memory management unit (MMU)	0 - 1	0
itlbnum	Number of instruction TLB entries	2 - 64	8
dtlbnum	Number of data TLB entries	2 - 64	8
tlb_type	0 : separate TLB with slow write 1: shared TLB with slow write 2: separate TLB with fast write	0 - 2	1
tlb_rep	LRU (0) or Random (1) TLB replacement	0 - 1	0
lddel	Load delay. One cycle gives best performance, but might create a critical path on targets with slow (data) cache memories. A 2-cycle delay can improve timing but will reduce performance with about 5%. Note that lddel = 1 is required for CASA.	1 - 2	2
disas	Print instruction disassembly in VHDL simulator console.	0 - 1	0
tbuf	Size of instruction trace buffer in kB (0 - instruction trace disabled)	0 - 64	0
pwd	Power-down. 0 - disabled, 1 - area efficient, 2 - timing efficient.	0 - 2	1
svt	Enable single-vector trapping	0 - 1	0
rstaddr	Default reset start address. If this generic is set to 16#fffff# the processor will read its start address from the interrupt controller interface signal IRQI.RSTVEC (dynamic reset start address).	0 - (2**^{20}-1)	0
smp	Enable multi-processor support 0: SMP support disabled 1- 15: SMP enabled, cpu index is taken from hindex generic 16-31: SMP enabled, cpu index is taken from irqi.index signal	0 - 31	0
cached	Fixed cacheability mask. See sections 69.2.19 and 69.5.8 for more information.	0 - 16#FFFF#	0
clk2x	Enables double-clocking. See section 69.9.3 and the LEON/GRLIB Design and Configuration Guide. Not present on all top-level entites.	0 - 15	0
scantest	Enable scan test support	0 - 1	0

*Table 999.*Configuration options

Generic	Function	Allowed range	Default
wbmask	Wide-bus mask. Indicates which address ranges are 64/128 bit capable. Treated as a 16-bit vector with LSB bit (right-most) indicating address 0 - 0x10000000. See section 69.5.8 for more information.	0 - 16#FFFF#	0
busw	Bus width of the wide bus area (64 or 128). See section 69.5.8 for more information.	64, 128	64
netlist	Use technology specific netlist	0 - 1	0
ft	Register file and cache memory protection, 4-bit bitfield Bit3: enable cache memory parity protection Bit2: enable register file TMR Bit1,0: unused Note	0 - 15	0

69.11 Signal descriptions

Table 1000 shows the interface signals of the core (VHDL ports).

*Table 1000.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	AMBA and processor clock.	-
CLK2		Input	Processor clock in 2x mode (LEON4sx2)	
GCLK2		Input	Gated processor clock in 2x mode (LEON4sx2)	
RSTN	N/A	Input	Reset	Low
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
AHBSI	*	Input	AHB slave input signals	-
IRQI	IRL[3:0]	Input	Interrupt level	High
	RST	Input	Reset power-down and error mode	High
	RUN	Input	Start after reset (SMP system only)	High
	RSTVEC[31:12]	Input	Reset start addr. (SMP and dynamic reset addr.)	-
	IACT	Input	Unused	-
	INDEX[3:0]	Input	CPU index when SMP = 2	-
	HRDRST	Input	Resets processor	High
IRQO	INTACK	Output	Interrupt acknowledge	High
	IRL[3:0]	Output	Processor interrupt level	High
	PWD	Output	Processor in power-down mode	High
	FPEN	Output	Floating-point unit enabled	High
	IDLE	Output	Processor idle	High
DBGI	-	Input	Debug inputs from DSU	-
DBGO	-	Output	Debug outputs to DSU	-
	ERROR		Processor in error mode, execution halted	Low
GCLK		Input	Gated processor clock for LEON4cg	

* see GRLIB IP Library User’s Manual

69.12 Library dependencies

Table 1001 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1001.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	LEON3, LEON4	Component, signals	LEON4 component declaration, interrupt and debug signals declaration

69.13 Component declaration

The LEON4 core has the following component declaration. There are also LEON4 top-levels that support clock gating (leon4cg), double-clocking (leon4s2x), shared FPU (leon4sh) and all available interfaces (leon4x). See the GRLIB template designs for instantiation examples.

```
entity leon4s is
  generic (
    hindex    : integer               := 0;
    fabtech   : integer range 0 to NTECH  := DEFFABTECH;
    memtech   : integer range 0 to NTECH  := DEFMEMTECH;
    nwindows  : integer range 2 to 32 := 8;
    dsu       : integer range 0 to 1  := 0;
    fpu       : integer range 0 to 31 := 0;
    v8        : integer range 0 to 63 := 0;
    cp        : integer range 0 to 1  := 0;
    mac       : integer range 0 to 1  := 0;
    pclow     : integer range 0 to 2  := 2;
    notag     : integer range 0 to 1  := 0;
    nwp       : integer range 0 to 4  := 0;
    icen      : integer range 0 to 1  := 0;
    irepl     : integer range 0 to 2  := 2;
    isets     : integer range 1 to 4  := 1;
    ilinesize : integer range 4 to 8  := 4;
    isetsize  : integer range 1 to 256 := 1;
    isetlock  : integer range 0 to 1  := 0;
    dcen      : integer range 0 to 1  := 0;
    drepl     : integer range 0 to 2  := 2;
    dsets     : integer range 1 to 4  := 1;
    dlinesize : integer range 4 to 8  := 4;
    dsetsize  : integer range 1 to 256 := 1;
    dsetlock  : integer range 0 to 1  := 0;
    dsnoop    : integer range 0 to 6  := 0;
    ilram      : integer range 0 to 1 := 0;
    ilramsize  : integer range 1 to 512 := 1;
    ilramstart : integer range 0 to 255 := 16#8e#;
    dlram      : integer range 0 to 1 := 0;
    dlramsize  : integer range 1 to 512 := 1;
    dlramstart : integer range 0 to 255 := 16#8f#;
    mmuen     : integer range 0 to 1  := 0;
    itlbnum   : integer range 2 to 64 := 8;
    dtlbnum   : integer range 2 to 64 := 8;
    tlb_type  : integer range 0 to 3  := 1;
    tlb_rep   : integer range 0 to 1  := 0;
    lddel     : integer range 1 to 2  := 2;
    disas     : integer range 0 to 2  := 0;
    tbuf      : integer range 0 to 64 := 0;
    pwd       : integer range 0 to 2  := 2;     -- power-down
    svt       : integer range 0 to 1  := 1;     -- single vector trapping
    rstaddr   : integer               := 0;
    smp       : integer range 0 to 15 := 0;     -- support SMP systems
    cached    : integer               := 0;     -- cacheability table
    scantest  : integer               := 0;
    wbmask    : integer               := 0;     -- Wide-bus mask
    busw      : integer               := 64;    -- AHB/Cache data width (64/128)
    ft        : integer               := 0
);
  port (
    clk    : in  std_ulogic;
    rstn   : in  std_ulogic;
    ahbi   : in  ahb_mst_in_type;
    ahbo   : out ahb_mst_out_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : in  ahb_slv_out_vector;
    irqi   : in  l3_irq_in_type;
    irqo   : out l3_irq_out_type;
    dbgi   : in  l3_debug_in_type;
    dbgo   : out l3_debug_out_type
  );
end;
```

70 LOGAN - On-chip Logic Analyzer

70.1 Introduction

The LOGAN core implements an on-chip logic analyzer for tracing and displaying of on-chip signals. LOGAN consists of a circular trace buffer and a triggering module. When armed, the logic analyzers stores the traced signals in the circular buffer until a trigger condition occurs. A trigger condition will freeze the buffer, and the traced data can then be read out via an APB interface.

The depth and width of the trace buffer is configurable through VHDL generics, as well as the number of trigger levels.

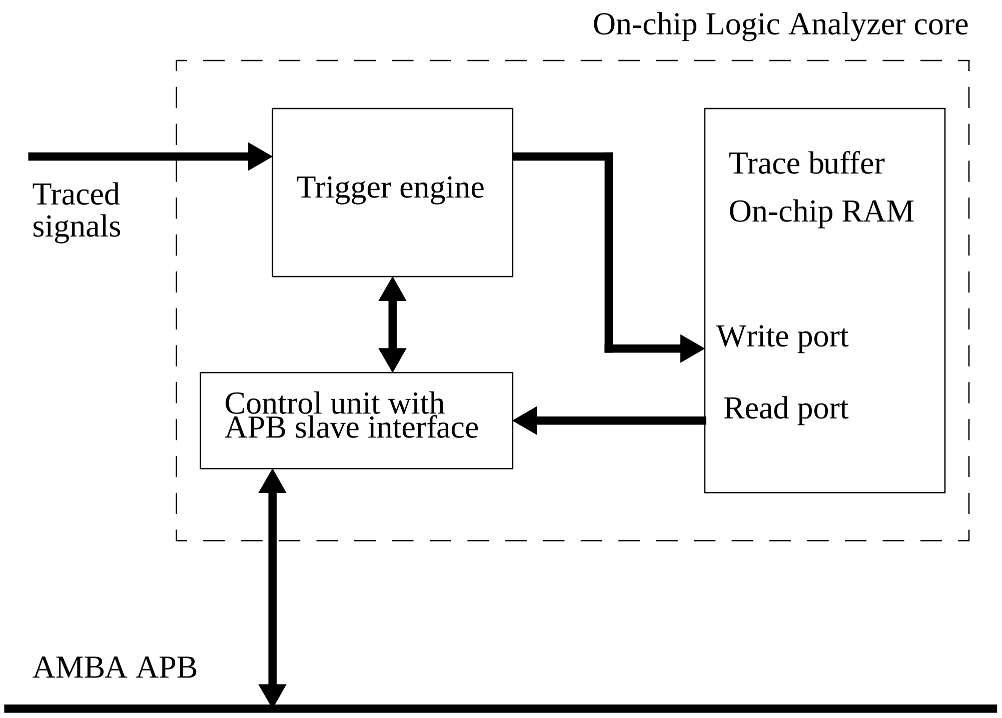


Figure 208. On-chip Logic Analyzer block diagram

70.2 Operation

70.2.1 Trace buffer

When the logic analyzer is armed, the traced signals are sampled and stored to the trace buffer on the rising edge of the sample clock (TCLK). The trace buffer consists of a circular buffer with an index register pointing to the next address in the buffer to be written. The index register is automatically incremented after each store operation to the buffer.

70.2.2 Clocking

LOGAN uses two clocks: TCLK and the APB clock. The trace signals are sampled on the rising edge of the sample clock (TCLK), while the control unit and the APB interface use the APB clock. TCLK and the APB clock does not need to be synchronized or have the same frequency.

70.2.3 Triggering

The logic analyzer contains a configurable number of trig levels. Each trig level is associated with a pattern and a mask. The traced signals are compared with the pattern, only comparing the bits set in the mask. This allows for triggering on any specific value or range. Furthermore each level has a match counter and a boolean equality flag. The equality flag specifies whether a match means that the pattern should equal the traced signals or that it should not be equal. It is possible to configure the trigger engine to stay at a certain level while the traced signals have a certain value using this flag. The match counter is a 6 bit counter which can be used to specify how many times a level should match before proceeding to the next. This is all run-time configurable through registers described in the register section.

To specify post-, center- or pre-triggering mode, the user can set a counter register that controls when the sampling stops relative to the triggering event. It can be set to any value in the range 0 to *depth*-1 thus giving total control of the trace buffer content.

To support the tracing of slowly changing signals, the logic analyzer has a 16-bit sample frequency divider register that controls how often the signals are sampled. The default divider value of 1 will sample the signals every clock cycle.

The *usequal* configuration option has a similar purpose as the sample frequency divider. The user can define one of the traced signals as a qualifier bit that has to have a specified value for the current signals to be stored in the trace buffer. This makes sampling of larger time periods possible if only some easily distinguished samples are interesting. This option has to be enabled with the *usequal* generic and the qualifier bit and value are written to a register.

70.2.4 Arming

To start operation, the logic analyzer needs to be armed. This is done by writing to the status register with bit 0 set to 1. A reset can be performed anytime by writing zero to the status register. After the final triggering event, the trigged flag will be raised and can be read out from the status register. The logic analyzer remains armed and trigged until the trigger counter reaches zero. When this happens the index of the oldest sample can be read from the trace buffer index register.

70.3 Registers

Both trace data and all registers are accessed through an APB interface. The LOGAN core will allocate a 64 kbyte block in the APB address space.

*Table 1002.*APB address mapping

APB address offset	Registers
0x0000	Status register
0x0004	Trace buffer index
0x0008	Page register
0x000C	Trig counter
0x0010	Sample freq. divider
0x0014	Storage qualifier setting
0x2000 - 0x20FF	Trig control settings
0x6000 - 0x6FFF	Pattern/mask configuration
0x8000 - 0xFFFF	Trace data

70.3.1 Status register

31	30	29	28	27 20	19 6	5 0
usereg	qualifier	armed	trigged	dbits	depth	trig levels

Figure 209. Status register

[31:28] These bits indicate whether an input register and/or storage qualifier is used and if the Logic Analyzer is armed and/or trigged.

[27:20] Number of traced signals.

[19:6] Last index of trace buffer. Depth-1.

[5:0] Number of trig levels.

70.3.2 Trace buffer index

31 abits	abits-1 0
“000...0”	the index of the oldest sample

Figure 210. Trace buffer index register

[31:*abits*] - Reserved.

[*abits*-1:0] - The index of the oldest sample in the buffer. *abits* is the number of bits needed to represent the configured depth. Note that this register is written by the trigger engine clock domain and thus needs to be known stable when read out. Only when the ‘armed’ bit in the status register is zero is the content of this register reliable.

70.3.3 Page register

31 4	3 0
“000...0”	current page

Figure 211. Page register

[31:4] - Reserved.

[3:0] - This register selects what page that will be used when reading from the trace buffer. The trace buffer is organized into pages of 1024 samples. Each sample can be between 1 and 256 bits. If the depth of the buffer is more than 1024 the page register has to be used to access the other pages. To access the i:th page the register should be set i (where i=0..15).

70.3.4 Trig counter

31 abits	abits-1 0
“000...0”	trig counter value

Figure 212. Trig counter register

[31:abits] - Reserved.

[nbits-1:0] - Trig counter value. A counter is incremented by one for each stored sample after the final triggering event and when it reaches the value stored in this register the sampling stops. 0 means posttrig and *depth-1* is pretrig. Any value in between can be used.

70.3.5 Sample frequency divider

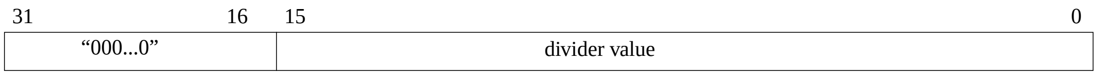


Figure 213. Sample freq. divider register

[31:16] - Reserved.

[15:0] - A sample is stored on every i:th clock cycle where i is specified through this register. This resets to 1 thus sampling occurs every cycle if not changed.

70.3.6 Storage qualifier

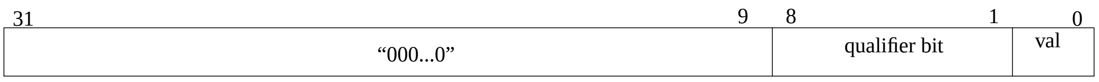


Figure 214. Storage qualifier register

[31:9] - Reserved.

[8:1] - Which bit to use as qualifier.

[0] - Qualify storage if bit is 1/0.

70.3.7 Trig control registers

This memory area contains the registers that control when the trigger engine shall proceed to the next level, i.e the match counter and a one bit field that specifies if it should trig on equality or inequality. There are *trigl* words where each word is used like in the figure below.

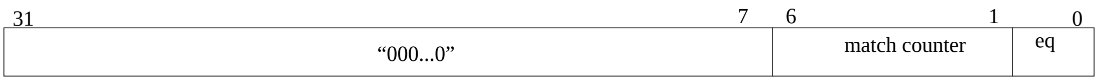


Figure 215. Trigger control register

[31:7] - Reserved.

[6:1] - Match counter. A counter is increased with one on each match on the current level and when it reaches the value stored in this register the trigger engine proceeds to the next level or if it is the last level it raises the trigged flag and starts the count of the trigger counter.

[0] - Specifies if a match is that the pattern/mask combination is equal or inequal compared to the traced signals.

70.3.8 Pattern/mask configuration

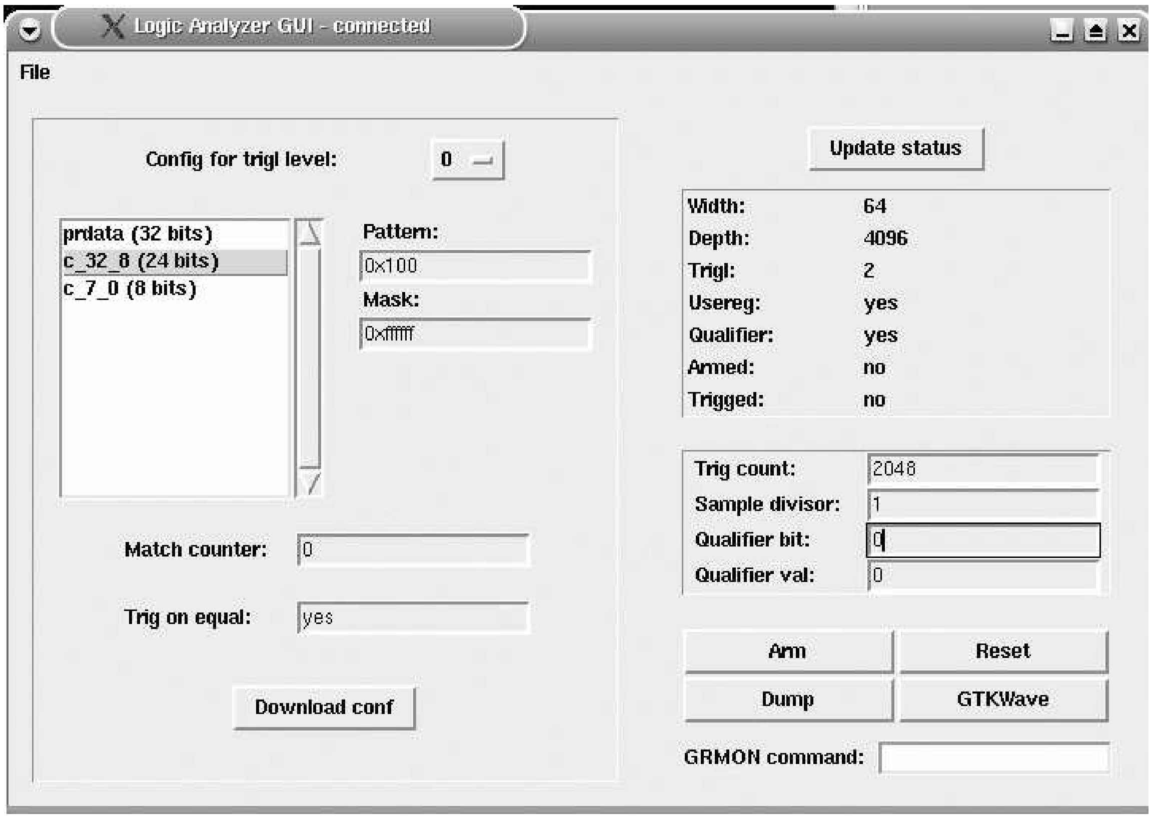
In these registers the pattern and mask for each trig level is configured. The pattern and mask can contain up to 8 words (256 bits) each so a number of writes can be necessary to specify just one pattern. They are stored with the LSB at the lowest address. The pattern of the first trig level is at 0x6000 and the mask is located 8 words later at 0x6020. Then the next trig levels starts at address 0x6040 and so on.

70.3.9 Trace data

It is placed in the upper half of the allocated APB address range. If the configuration needs more than the allocated 32 kB of the APB range the page register is used to page into the trace buffer. Each stored word is *dbits* wide but 8 words of the memory range is always allocated so the entries in the trace buffer are found at multiples of 0x20, i.e. 0x8000, 0x8020 and so on.

70.4 Graphical interface

The logic analyzer is normally controlled by the LOGAN debug driver in GRMON. It is also possible to control the LOGAN operation using a graphical user interface (GUI) written in Tcl/Tk. The GUI is provided with GRMON, refer to the GRMON manual for more details.



Config for trigl level: 0

prdata (32 bits)
c_32_8 (24 bits)
c_7_0 (8 bits)

Pattern: 0x100
Mask: 0xffffff

Match counter: 0
Trig on equal: yes

Width:	64
Depth:	4096
Trigl:	2
Usereg:	yes
Qualifier:	yes
Armed:	no
Trigged:	no

Trig count:	2048
Sample divisor:	1
Qualifier bit:	0
Qualifier val:	0

GRMON command:

70.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x062. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

70.6 Configuration options

Table 1003 shows the configuration options of the core (VHDL generics).

*Table 1003.*Configuration options

Generic	Function	Allowed range	Default
dbits	Number of traced signals	1 - 255	32
depth	Number of stored samples	256 - 16384	1024
trigl	Number of trigger levels	1 - 63	1
usereg	Use input register	0 - 1	1
usequal	Use storage qualifier	0 - 1	0
pindex	APB slave index	0 - NAPBSLV - 1	0
paddr	The 12-bit MSB APB address	0 -16#FFF#	0
pmask	The APB address mask	16#000 - 16#F00#	F00
memtech	Memory technology	0 - NTECH	0

The usereg VHDL generic specifies whether to use an input register to synchronize the traced signals and to minimize their fan out. If usereg=1 then all signals will be clocked into a register on the positive edge of the supplied clock signal, otherwise they are sent directly to the RAM.

70.7 Signal descriptions

Table 1004 shows the interface signals of the core (VHDL ports).

*Table 1004.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	System clock	-
TCLK	N/A	Input	Sample clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SIGNALS	N/A	Input	Vector of traced signals	-

* See GRLIB IP Library users manual

70.8 Library dependencies

Table 1005 shows libraries used when instantiating the core (VHDL libraries).

*Table 1005.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

70.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

entity logan_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ... -- other signals
    );
end;

architecture rtl of logan_ex is

  -- AMBA signals
signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);
signal signals : std_logic_vector(63 downto 0);

begin

-- Logic analyzer core
```

```
  logan0 : logan
  generic map (dbits=>64,depth=>4096,trigl=>2,usereg=>1,usequal=>0,
            pindex => 3, paddr => 3, pmask => 16#F00#, memtech => memtech)
  port map (rstn, clk, clk, apbi, apbo(3), signals);

end;
```

71 MCTRL - Combined PROM/IO/SRAM/SDRAM Memory Controller

71.1 Overview

The memory controller handles a memory bus hosting PROM, memory mapped I/O devices, asynchronous static ram (SRAM) and synchronous dynamic ram (SDRAM). The controller acts as a slave on the AHB bus. The function of the memory controller is programmed through memory configuration registers 1, 2 & 3 (MCFG1, MCFG2 & MCFG3) through the APB bus. The memory bus supports four types of devices: prom, sram, sdram and local I/O. The memory bus can also be configured in 8- or 16-bit mode for applications with low memory and performance demands.

Chip-select decoding is done for two PROM banks, one I/O bank, five SRAM banks and two SDRAM banks.

The controller decodes three address spaces (PROM, I/O and RAM) whose mapping is determined through VHDL-generics.

Figure 216 shows how the connection to the different device types is made.

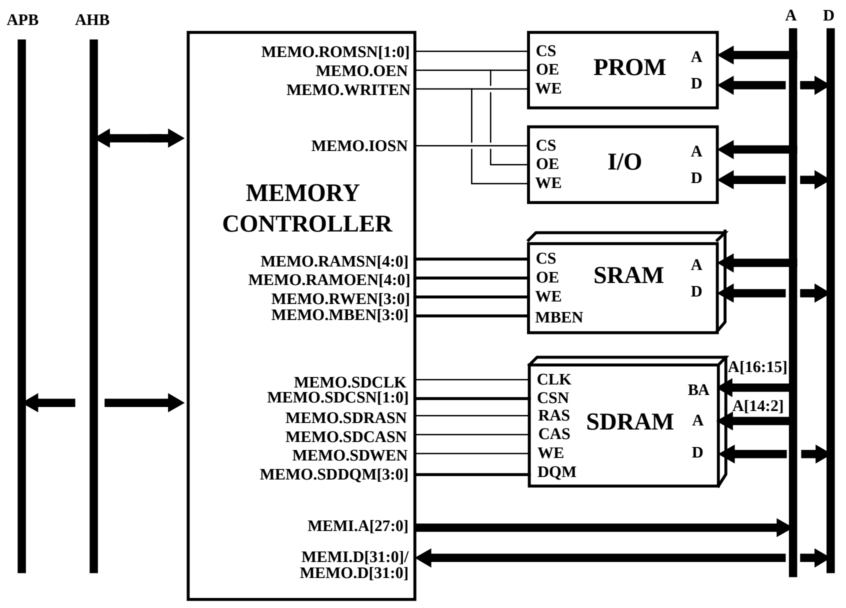


Figure 216. **Memory controller conected to AMBA bus and different types of memory devices**

71.2 PROM access

Accesses to prom have the same timing as RAM accesses, the differences being that PROM cycles can have up to 15 waitstates.

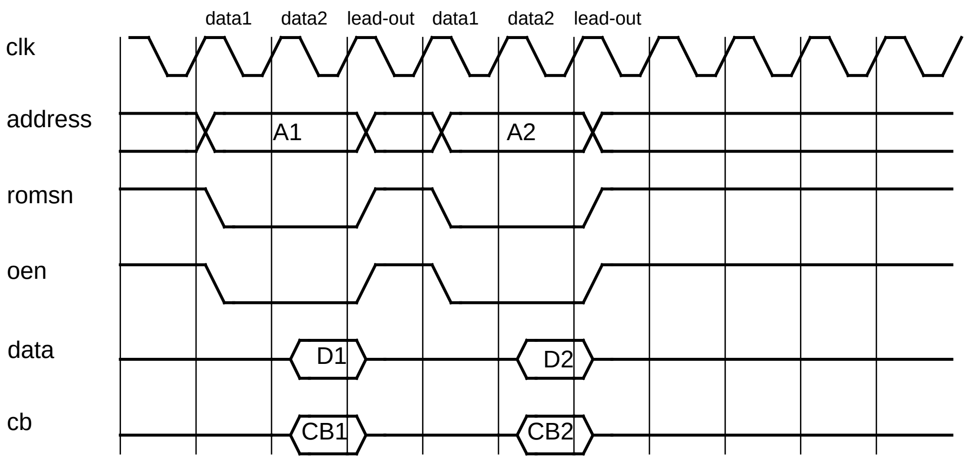


Figure 217. **Prom non-consecutive read cyclecs.**

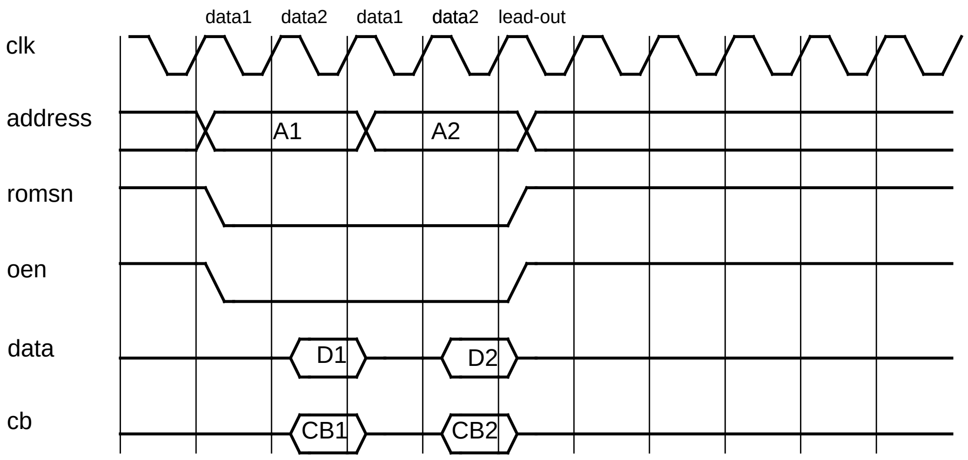


Figure 218. **Prom consecutive read cyclecs.**

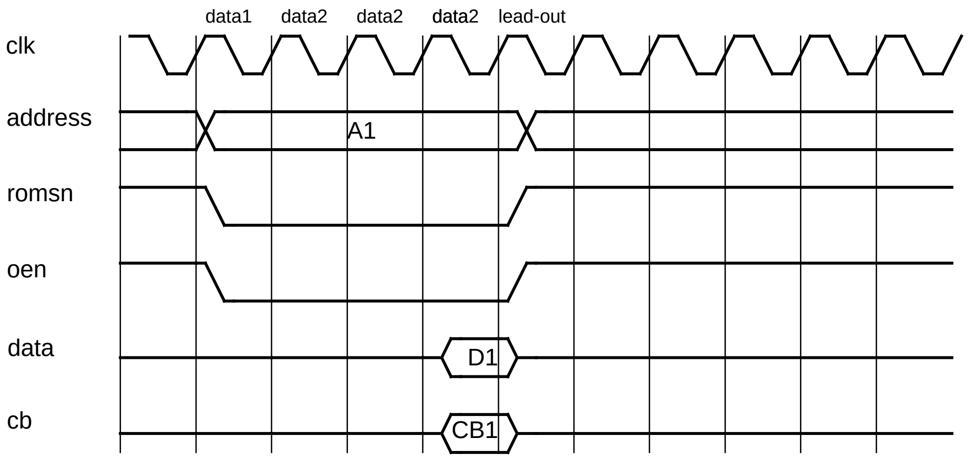


Figure 219. **Prom read access with two waitstates.**

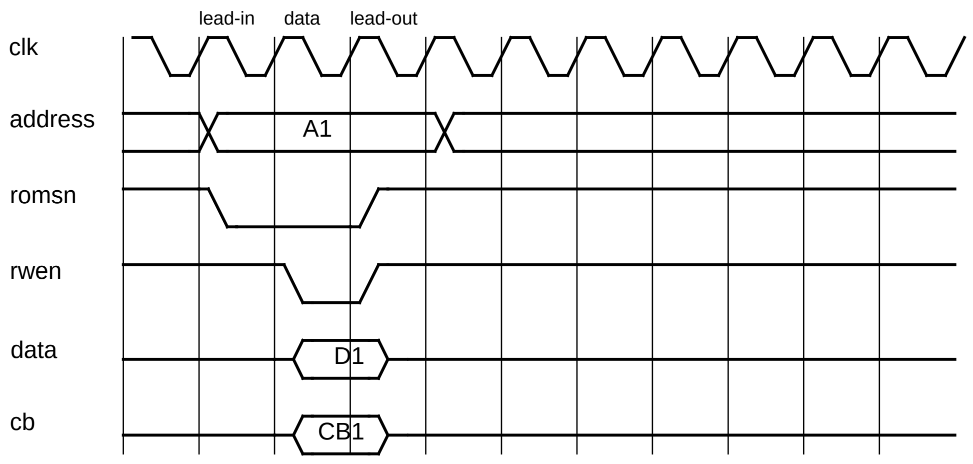


Figure 220. **Prom write cycle (0-waitstates)**

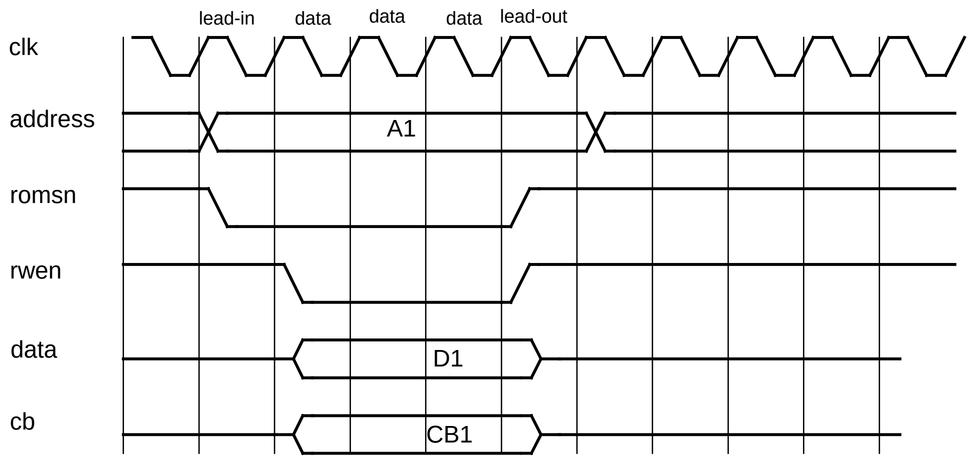


Figure 221. **Prom write cycle (2-waitstates)**

Two PROM chip-select signals are provided, MEMO.ROMSN[1:0]. MEMO.ROMSN[0] is asserted when the lower half of the PROM area as addressed while MEMO.ROMSN[1] is asserted for the upper half. When the VHDL model is configured to boot from internal prom, MEMO.ROMSN[0] is never asserted and all accesses to the lower half of the PROM area are mapped on the internal prom.

71.3 Memory mapped I/O

Accesses to I/O have similar timing to ROM/RAM accesses, the differences being that a additional waitstates can be inserted by de-asserting the MEMI.BRDYN signal. The I/O select signal (MEMO.IOSN) is delayed one clock to provide stable address before MEMO.IOSN is asserted.

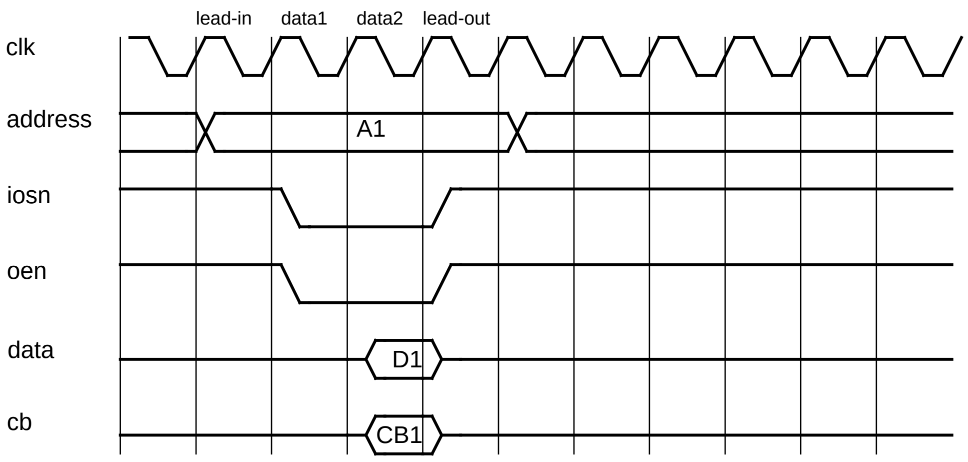


Figure 222. **I/O read cycle (0-waitstates)**

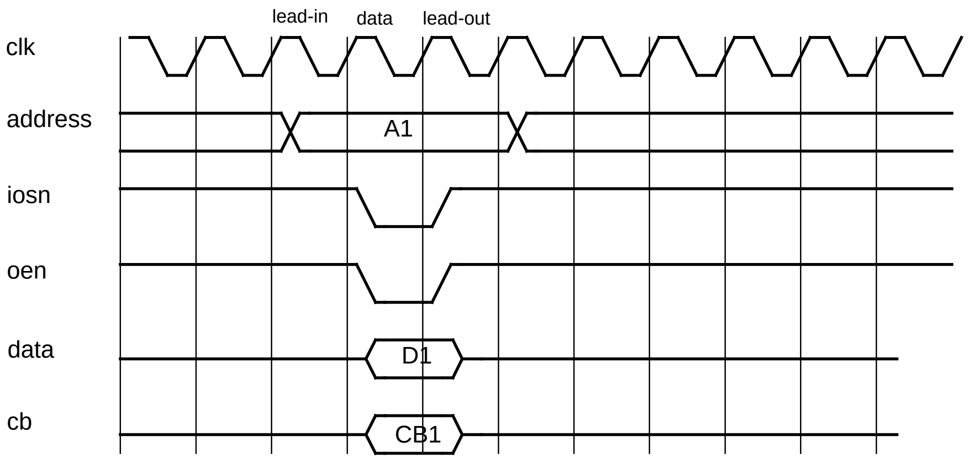


Figure 223. **I/O write cycle (0-waitstates)**

71.4 SRAM access

The SRAM area can be up to 1 Gbyte, divided on up to five RAM banks. The size of banks 1-4 (MEMO.RAMSN[3:0]is programmed in the RAM bank-size field (MCFG2[12:9]) and can be set in binary steps from 8 Kbyte to 256 Mbyte. The fifth bank (RAMSN[4]) decodes the upper 512 Mbyte (controlled by means of the *sdrasel* VHDL generic) and cannot be used simultaneously with SDRAM memory. A read access to SRAM consists of two data cycles and between zero and three waitstates. Accesses to MEMO.RAMSN[4] can further be stretched by de-asserting MEMI.BRDYN until the data is available. On non-consecutive accesses, a lead-out cycle is added after a read cycle to prevent bus contention due to slow turn-off time of memories or I/O devices. Figure 224 shows the basic read cycle waveform (zero waitstate).

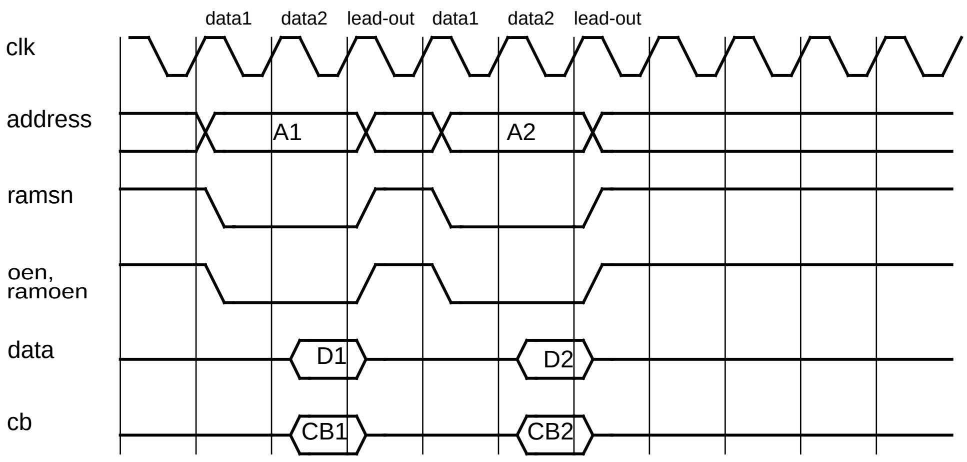


Figure 224. **SRAM non-consecutive read cyclecs.**

For read accesses to MEMO.RAMSN[4:0], a separate output enable signal (MEMO.RAMOEN[n]) is provided for each RAM bank and only asserted when that bank is selected. A write access is similar to the read access but takes a minimum of three cycles:

Through an (optional) feed-back loop from the write strobes, the data bus is guaranteed to be driven until the write strobes are de-asserted. Each byte lane has an individual write strobe to allow efficient byte and half-word writes. If the memory uses a common write strobe for the full 16- or 32-bit data, the read-modify-write bit in the MCFG2 register should be set to enable read-modify-write cycles for sub-word writes.

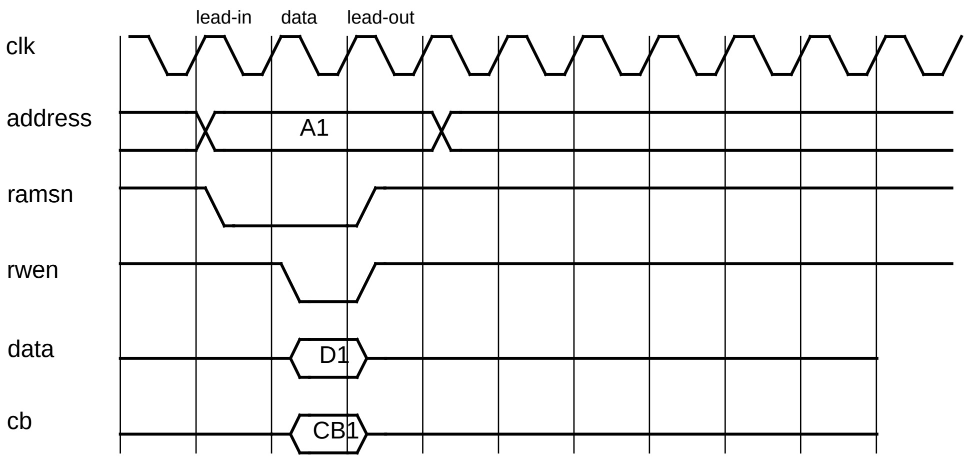


Figure 225. **Sram write cycle (0-waitstates)**

A drive signal vector for the data I/O-pads is provided which has one drive signal for each data bit. It can be used if the synthesis tool does not generate separate registers automatically for the current technology. This can remove timing problems with output delay.

71.5 8-bit and 16-bit PROM and SRAM access

To support applications with low memory and performance requirements efficiently, it is not necessary to always have full 32-bit memory banks. The SRAM and PROM areas can be individually configured for 8- or 16-bit operation by programming the ROM and RAM size fields in the memory configuration registers. Since read access to memory is always done on 32-bit word basis, read access to 8-bit memory will be transformed in a burst of four read cycles while access to 16-bit memory will generate a burst of two 16-bits reads. During writes, only the necessary bytes will be writen. Figure 226 shows an interface example with 8-bit PROM and 8-bit SRAM. Figure 227 shows an example of a 16-bit memory interface.

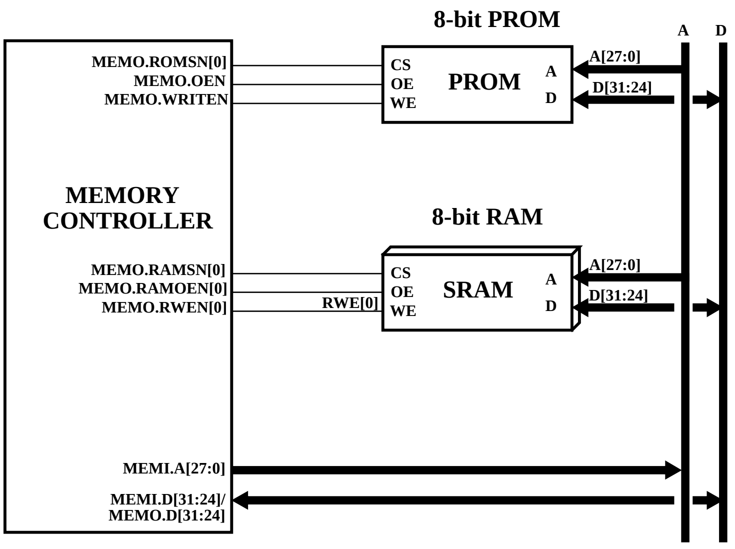


Figure 226. **8-bit memory interface example**

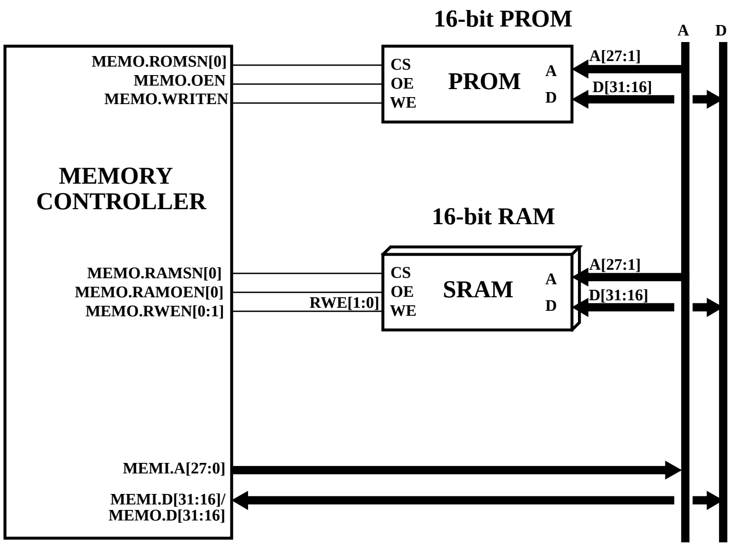


Figure 227. **16-bit memory interface example**

71.6 Burst cycles

To improve the bandwidth of the memory bus, accesses to consecutive addresses can be performed in burst mode. Burst transfers will be generated when the memory controller is accessed using an AHB burst request. These includes instruction cache-line fills, double loads and double stores. The timing of a burst cycle is identical to the programmed basic cycle with the exception that during read cycles, the lead-out cycle will only occurs after the last transfer. Burst cycles will not be generated to the IO area.

Only word (HSIZE = "010") bursts of incremental type (HBURST=INCR, INCR4, INCR8 or INCR16) are supported.

71.7 8- and 16-bit I/O access

Similar to the PROM/RAM areas, the I/O area can also be configured to 8- or 16-bit mode. However, the I/O device will NOT be accessed by multiple 8/16 bit accesses as the memory areas, but only with one single access just as in 32-bit mode. To access an I/O device on a 16-bit bus, LDUH/STH instructions should be used while LDUB/STB should be used with an 8-bit bus.

71.8 SDRAM access

71.8.1 General

Synchronous dynamic RAM (SDRAM) access is supported to two banks of PC100/PC133 compatible devices. This is implemented by a special version of the SDCTRL SDRAM controller core from Aeroflex Gaisler, which is optionally instantiated as a sub-block. The SDRAM controller supports 64M, 256M and 512M devices with 8 - 12 column-address bits, and up to 13 row-address bits. The size of the two banks can be programmed in binary steps between 4 Mbyte and 512 Mbyte. The operation of the SDRAM controller is controlled through MCFG2 and MCFG3 (see below). Both 32- and 64-bit data bus width is supported, allowing the interface of 64-bit DIMM modules. The memory controller can be configured to use either a shared or separate bus connecting the controller and SDRAM devices. When the VHDL generic **mobile** is set to a value not equal to 0, the controller supports mobile SDRAM.

71.8.2 Address mapping

The two SDRAM chip-select signals are decoded. SDRAM area is mapped into the upper half of the RAM area defined by BAR2 register. When the SDRAM enable bit is set in MCFG2, the controller is enabled and mapped into upper half of the RAM area as long as the SRAM disable bit is not set. If the SRAM disable bit is set, all access to SRAM is disabled and the SDRAM banks are mapped into the lower half of the RAM area.

71.8.3 Initialisation

When the SDRAM controller is enabled, it automatically performs the SDRAM initialisation sequence of PRECHARGE, 2x AUTO-REFRESH and LOAD-MODE-REG on both banks simultaneously. When mobile SDRAM functionality is enabled the initialization sequence is appended by a LOAD-EXTMODE-REG command. The controller programs the SDRAM to use page burst on read and single location access on write.

71.8.4 Configurable SDRAM timing parameters

To provide optimum access cycles for different SDRAM devices (and at different frequencies), some SDRAM parameters can be programmed through memory configuration register 2 (MCFG2) The programmable SDRAM parameters can be seen in tabel 1006.

Table 1006.**SDRAM programmable timing parameters**

Function	Parameter	Range	Unit
CAS latency, RAS/CAS delay	t_{CAS}, t_{RCD}	2 - 3	clocks
Precharge to activate	t_{RP}	2 - 3	clocks
Auto-refresh command period	t_{RFC}	3 - 11	clocks
Auto-refresh interval		10 - 32768	clocks

Remaining SDRAM timing parameters are according the PC100/PC133 specification.

When mobile SDRAM support is enabled, one additional timing parameter (TXSR) can be programmed though the Power-Saving configuration register.

*Table 1007.*Mobile SDRAM programmable minimum timing parameters

SDRAM timing parameter	Minimum timing (clocks)
Exit Self Refresh mode to first valid command (t_{XSR})	tXSR

71.9 Refresh

The SDRAM controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the MCFG3 register. Depending on SDRAM type, the required period is typically 7.8 or 15.6 µs (corresponding to 780 or 1560 clocks at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by setting bit 31 in MCFG2.

71.9.1 Self Refresh

The self refresh mode can be used to retain data in the SDRAM even when the rest of the system is powered down. When in the self refresh mode, the SDRAM retains data without external clocking and refresh are handled internally. The memory array that is refreshed during the self refresh operation is defined in the extended mode register. These settings can be changed by setting the PASR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the PASR bits are changed. The supported “Partial Array Self Refresh” modes are: Full, Half, Quarter, Eighth, and Sixteenth array. “Partial Array Self Refresh” is only supported when mobile SDRAM functionality is enabled. To enable the self refresh mode, set the PMODE bits in the Power-Saving configuration register to “010” (Self Refresh). The controller will enter self refresh mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. When exiting this mode the controller introduce a delay defined by tXSR in the Power-Saving configuration register and a AUTO REFRESH command before any other memory access is allowed. The minimum duration of this mode is defined by tRAS. This mode is only available then the VHDL generic **mobile** >=1.

71.9.2 Power-Down

When entering the power-down mode all input and output buffers, excluding SDCKE, are deactivated. All data in the SDRAM is retained during this operation. To enable the power-down mode, set the PMODE bits in the Power-Saving configuration register to “001” (Power-Down). The controller will enter power-down mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits is cleared. The REFRESH command will still be issued by the controller in this mode. When exiting this mode a delay of one clock cycles are added before issue any command to the memory. This mode is only available then the VHDL generic **mobile** >=1.

71.9.3 Deep Power-Down

The deep power-down operating mode is used to achieve maximum power reduction by eliminating the power of the memory array. Data will not be retained after the device enters deep power-down mode. To enable the deep power-down mode, set the PMODE bits in the Power-Saving configuration register to “101” (Deep Power-Down). To exit the deep power-down mode the PMODE bits in the Power-Saving configuration register must be cleared. The controller will respond with an AMBA ERROR response to an AMBA access, that will result in a memory access, during Deep Power-Down mode. This mode is only available then the VHDL generic **mobile** >=1 and mobile SDRAM functionality is enabled.

71.9.4 Temperature-Compensated Self Refresh

The settings for the temperature-compensation of the Self Refresh rate can be controlled by setting the TCSR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the TCSR bits are changed. Note that some vendors implements a Internal Temperature-Compensated Self Refresh feature, which makes the memory to ignore the TCSR bits. This functionality is only available then the VHDL generic **mobile** >=1 and mobile SDRAM functionality is enabled.

71.9.5 Drive Strength

The drive strength of the output buffers can be controlled by setting the DS bits in the Power-Saving configuration register. The extended mode register is automatically updated when the DS bits are changed. The available options are: full, three-quarter, one-half, and one-quarter drive strengths. This functionality is only available then the VHDL generic **mobile** >=1 and mobile SDRAM functionality is enabled.

71.9.6 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in MCFG2: PRE-CHARGE, AUTO-REFRESH, LOAD-MODE-REG (LMR) and LOAD-EXTMODE-REG (EMR). If the LMR command is issued, the CAS delay as programmed in MCFG2 will be used, remaining fields are fixed: page read burst, single location write, sequential burst. If the EMR command is issued, the DS, TCSR and PASR as programmed in Power-Saving configuration register will be used. To issue the EMR command, the EMR bit in the MCFG4 register has to be set. The command field will be cleared after a command has been executed. Note that when changing the value of the CAS delay, a LOAD-MODE-REGISTER command should be generated at the same time.

71.9.7 Read cycles

A read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command after the programmed CAS delay. A read burst is performed if a burst access has been requested on the AHB bus. The read cycle is terminated with a PRE-CHARGE command, no banks are left open between two accesses.

71.9.8 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. A write burst on the AHB bus will generate a burst of write commands without idle cycles in-between.

71.9.9 Address bus connection

The memory controller can be configured to either share the address and data buses with the SRAM, or to use separate address and data buses. When the buses are shared, the address bus of the SDRAMs should be connected to A[14:2], the bank address to A[16:15]. The MSB part of A[14:2] can be left unconnected if not used. When separate buses are used, the SDRAM address bus should be connected to SA[12:0] and the bank address to SA[14:13].

71.9.10 Data bus

SDRAM can be connected to the memory controller through the common or separate data bus. If the separate bus is used the width is configurable to 32 or 64 bits. 64-bit data bus allows the 64-bit SDRAM devices to be connected using the full data capacity of the devices. 64-bit SDRAM devices can be connected to 32-bit data bus if 64-bit data bus is not available but in this case only half the full data capacity will be used. There is a drive signal vector and separate data vector available for SDRAM. The drive vector has one drive signal for each data bit. These signals can be used to remove timing problems with the output delay when a separate SDRAM bus is used. SDRAM bus signals are described in section 71.13, for configuration options refer to section 71.15.

71.9.11 Clocking

The SDRAM clock typically requires special synchronisation at layout level. For Xilinx and Altera device, the GR Clock Generator can be configured to produce a properly synchronised SDRAM clock. For other FPGA targets, the GR Clock Generator can produce an inverted clock.

71.10 Using bus ready signalling

The MEMI.BRDYN signal can be used to stretch access cycles to the I/O area and the ram area decoded by MEMO.RAMSN[4]. The accesses will always have at least the pre-programmed number of waitstates as defined in memory configuration registers 1 & 2, but will be further stretched until MEMI.BRDYN is asserted. MEMI.BRDYN should be asserted in the cycle preceding the last one. The use of MEMI.BRDYN can be enabled separately for the I/O and RAM areas.

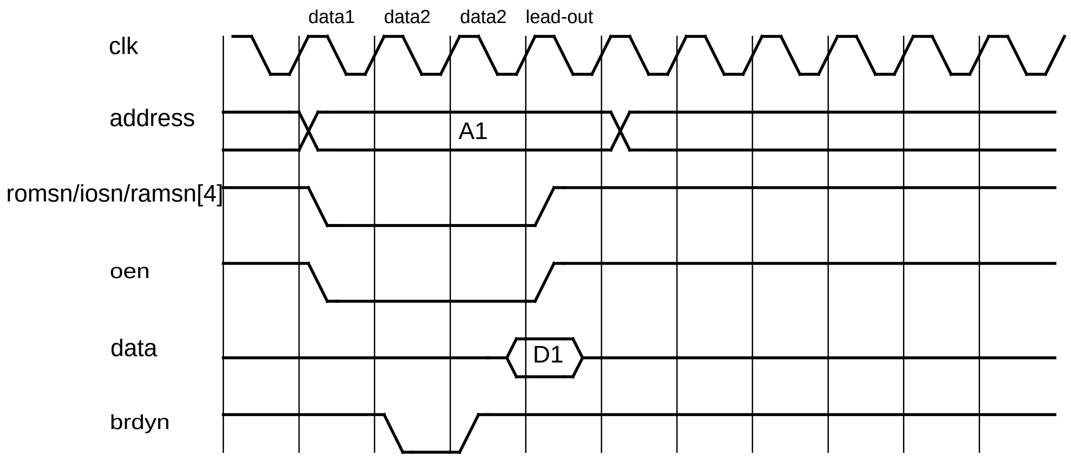


Figure 228. **READ cycle with one extra data2 cycle added with BRDYN (synchronous sampling). Lead-out cycle is only applicable for I/O accesses.**

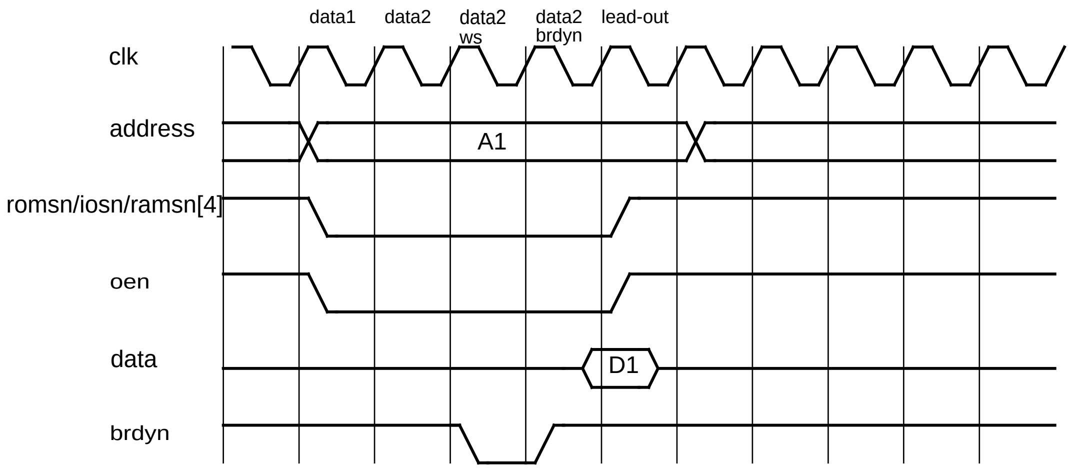


Figure 229. **Read cycle with one waitstate (configured) and one BRDYN generated waitstate (synchronous sampling).**

71.11 Access errors

An access error can be signalled by asserting the MEMI.BEXCN signal, which is sampled together with the data. If the usage of MEMI.BEXCN is enabled in memory configuration register 1, an error response will be generated on the internal AMBA bus. MEMI.BEXCN can be enabled or disabled through memory configuration register 1, and is active for all areas (PROM, I/O an RAM).

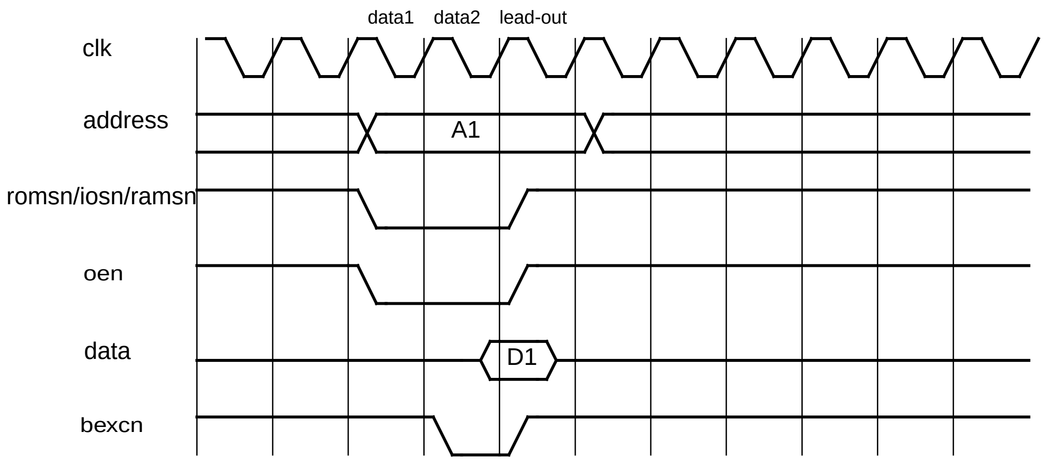


Figure 230. **Read cycle with BEXCN.**

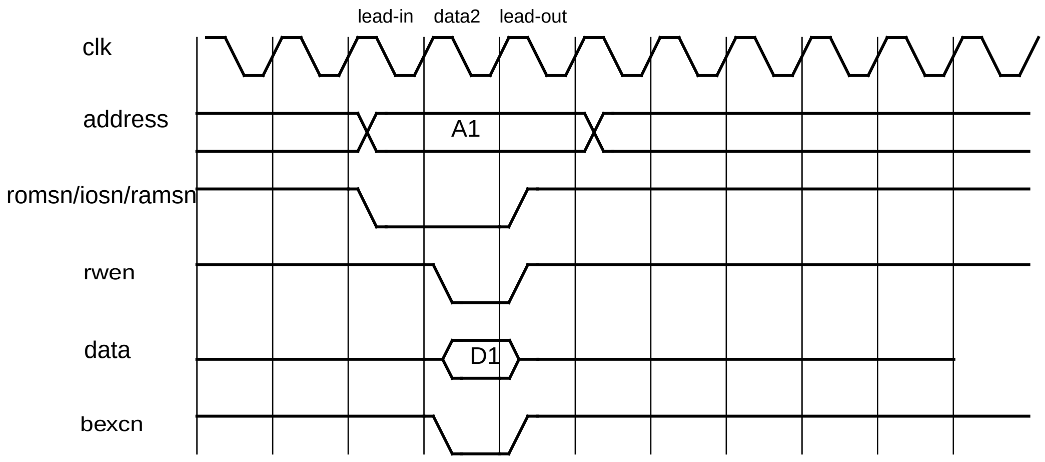


Figure 231. **Write cycle with BEXCN. Chip-select (iosn) is not asserted in lead-in cycle for io-accesses.**

71.12 Attaching an external DRAM controller

To attach an external DRAM controller, MEMO.RAMSN[4] should be used since it allows the cycle time to vary through the use of MEMI.BRDYN. In this way, delays can be inserted as required for opening of banks and refresh.

71.13 Registers

The memory controller is programmed through registers mapped into APB address space.

Table 1008.**Memory controller registers**

APB address offset	Register
0x0	MCFG1
0x4	MCFG2
0x8	MCFG3
0xC	MCFG4 (Power-Saving configuration register)

71.13.1 Memory configuration register 1 (MCFG1)

Memory configuration register 1 is used to program the timing of rom and local I/O accesses.

Table 1009. Memory configuration register 1.

31 29	28 27	26	25	24	23 20	19	18
RESERVED	IOBUSW	IBRDY	BEXCN		IO WAITSTATES	IOEN	

12	11	10	9 8	7 4	3 0
RESERVED	PWEN		PROM WIDTH	PROM WRITE WS	PROM READ WS

31 : 29	RESERVED
28 : 27	I/O bus width (IOBUSW) - Sets the data width of the I/O area (“00”=8, “01”=16, “10” =32).
26	I/O bus ready enable (IBRDY) - Enables bus ready (BRDYN) signalling for the I/O area. Reset to ‘0’.
25	Bus error enable (BEXCN) - Enables bus error signalling. Reset to ‘0’.
24	RESERVED
23 : 20	I/O waitstates (IO WAITSTATES) - Sets the number of waitstates during I/O accesses (“0000”=0, “0001”=1, “0010”=2,..., “1111”=15).
19	I/O enable (IOEN) - Enables accesses to the memory bus I/O area.
18:12	RESERVED

Table 1009. Memory configuration register 1.

11	PROM write enable (PWEN) - Enables write cycles to the PROM area.
10	RESERVED
9 : 8	PROM width (PROM WIDTH) - Sets the data width of the PROM area (“00”=8, “01”=16, “10”=32).
7 : 4	PROM write waitstates (PROM WRITE WS) - Sets the number of wait states for PROM write cycles (“0000”=0, “0001”=1, “0010”=2,..., “1111”=15).
3 : 0	PROM read waitstates (PROM READ WS) - Sets the number of wait states for PROM read cycles (“0000”=0, “0001”=1, “0010”=2,...,”1111”=15). Reset to “1111”.

During power-up, the prom width (bits [9:8]) are set with value on MEMI.BWIDTH inputs. The prom waitstates fields are set to 15 (maximum). External bus error and bus ready are disabled. All other fields are undefined.

71.13.2 Memory configuration register 2 (MCFG2)

Memory configuration register 2 is used to control the timing of the SRAM and SDRAM.

Table 1010. Memory configuration register 2.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SDRF	TRP	SDRAM TRFC			TCAS	SDRAM BANKSZ			SDRAM COLSZ		SDRAM CMD		D64	RES	MS
15	14	13	12			9	8	7	6	5	4	3	2	1	0
RES	SE	SI	RAM BANK SIZE					RBRDY	RMW	RAM WIDTH		RAM WRITE WS		RAM READ WS	

31	SDRAM refresh (SDRF) - Enables SDRAM refresh.
30	SDRAM TRP parameter (TRP) - t_{RP} will be equal to 2 or 3 system clocks (0/1).
29 : 27	SDRAM TRFC parameter (SDRAM TRFC) - t_{RFC} will be equal to 3+field-value system clocks.
26	SDRAM TCAS parameter (TCAS) - Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-COMMAND-REGISTER command must be issued at the same time. Also sets RAS/CAS delay ($t_{RCD).}$
25 : 23	SDRAM bank size (SDRAM BANKSZ) - Sets the bank size for SDRAM chip selects (“000”=4 Mbyte, “001”=8 Mbyte, “010”=16 Mbyte.... “111”=512 Mbyte).
22 : 21	SDRAM column size (SDRAM COLSZ) - “00”=256, “01”=512, “10”=1024, “11”=4096 when bit 25:23=”111” 2048 otherwise.
20 : 19	SDRAM command (SDRAM CMD) - Writing a non-zero value will generate a SDRAM command. “01”=PRECHARGE, “10”=AUTO-REFRESH, “11”=LOAD-COMMAND-REGISTER. The field is reset after the command has been executed.
18	64-bit SDRAM data bus (D64) - Reads ‘1’ if the memory controller is configured for 64-bit SDRAM data bus width, ‘0’ otherwise. Read-only.
17	RESERVED
16	Mobile SDR support enabled. ‘1’ = Enabled, ‘0’ = Disabled (read-only)
15	RESERVED
14	SDRAM enable (SE) - Enables the SDRAM controller.
13	SRAM disable (SI) - Disables accesses RAM if bit 14 (SE) is set to ‘1’.
12 : 9	RAM bank size (RAM BANK SIZE) - Sets the size of each RAM bank (“0000”=8 kbyte, “0001”=16 kbyte, ..., “1111”=256 Mbyte).
8	RESERVED
7	RAM bus ready enable (RBRDY) - Enables bus ready signalling for the RAM area.
6	Read-modify-write enable (RMW) - Enables read-modify-write cycles for sub-word writes to 16- bit 32-bit areas with common write strobe (no byte write strobe).
5 : 4	RAM width (RAM WIDTH) - Sets the data width of the RAM area (“00”=8, “01”=16, “1X”=32).
3 : 2	RAM write waitstates (RAM WRITE WS) - Sets the number of wait states for RAM write cycles (“00”=0, “01”=1, “10”=2, “11”=3).
1 : 0	RAM read waitstates (RAM READ WS) - Sets the number of wait states for RAM read cycles (“00”=0, “01”=1, “10”=2, “11”=3).

71.13.3 Memory configuration register 3 (MCFG3)

MCFG3 is contains the reload value for the SDRAM refresh counter.

31 30 29 28 27	26 25 24 23 22 21 20 19 18 17 16 15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0
RESERVED	SDRAM REFRESH RELOAD VALUE	RESERVED

31: 27	RESERVED
26: 12	SDRAM refresh counter reload value (SDRAM REFRESH RELOAD VALUE)
11: 0	RESERVED

The period between each AUTO-REFRESH command is calculated as follows:

$t_{REFRESH}$ = ((reload value) + 1) / SYSCLK

Table 1011. MCFG4 Power-Saving configuration register

31	30	29	28 24	23 20	19	18 16	15 7	6 5	4 3	2 0
ME	CE	EM	Reserved	tXSR	res	PMODE	Reserved	DS	TCSR	PASR

31	Mobile SDRAM functionality enabled. ‘1’ = Enabled (support for Mobile SDRAM), ‘0’ = disabled (support for standard SDRAM)
30	Clock enable (CE). This value is driven on the CKE inputs of the SDRAM. Should be set to ‘1’ for correct operation. This register bit is read only when Power-Saving mode is other then none.
29	EMR. When set, the LOAD-COMMAND-REGISTER command issued by the SDRAM command filed in MCFG2 will be interpret as a LOAD-EXTENDED-COMMAND-REGISTER command.
28: 24	Reserved
23: 20	SDRAM tXSR timing. tXSR will be equal to field-value system clocks. (Read only when Mobile SDR support is disabled).
19	Reserved
18: 16	Power-Saving mode (Read only when Mobile SDR support is disabled). “000”: none “001”: Power-Down (PD) “010”: Self-Refresh (SR) “101”: Deep Power-Down (DPD)
15: 7	Reserved
6: 5	Selectable output drive strength (Read only when Mobile SDR support is disabled). “00”: Full “01”: One-half “10”: One-quarter “11”: Three-quarter
4: 3	Reserved for Temperature-Compensated Self Refresh (Read only when Mobile SDR support is disabled). “00”: 70ªC “01”: 45ªC “10”: 15ªC “11”: 85ªC
2: 0	Partial Array Self Refresh (Read only when Mobile SDR support is disabled). “000”: Full array (Banks 0, 1, 2 and 3) “001”: Half array (Banks 0 and 1) “010”: Quarter array (Bank 0) “101”: One-eighth array (Bank 0 with row MSB = 0) “110”: One-sixteenth array (Bank 0 with row MSB = 00)

71.14 Vendor and device identifiers

The core has vendor identifier 0x04 (ESA) and device identifier 0x00F. For description of vendor and device identifier see GRLIB IP Library User's Manual.

71.15 Configuration options

Table 1012 shows the configuration options of the core (VHDL generics).

Table 1012.**Configuration options**

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0x1FFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#E00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space. Default I/O area is 0x20000000 - 0x2FFFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#E00#
ramaddr	ADDR field of the AHB BAR2 defining RAM address space. Default RAM area is 0x40000000-0x7FFFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR2 defining RAM address space.	0 -16#FFF#	16#C00#
paddr	ADDR field of the APB BAR configuration registers address space.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR configuration registers address space.	0 - 16#FFF#	16#FFF#
wprot	RAM write protection.	0 - 1	0
invclk	Inverted clock is used for the SDRAM.	0 - 1	0
fast	Enable fast SDRAM address decoding.	0 - 1	0
romasel	log2(PROM address space size) - 1. E.g. if size of the PROM area is 0x20000000 romasel is log2(2^29)-1 = 28.	0 - 31	28
sdrasel	log2(RAM address space size) - 1. E.g if size of the RAM address space is 0x40000000 sdrasel is log2(2^30)-1= 29.	0 - 31	29
srbanks	Number of SRAM banks.	0 - 5	4
ram8	Enable 8-bit PROM and SRAM access.	0 - 1	0
ram16	Enable 16-bit PROM and SRAM access.	0 - 1	0
sden	Enable SDRAM controller.	0 - 1	0
sepbus	SDRAM is located on separate bus.	0 - 1	1
sdbits	32 or 64 -bit SDRAM data bus.	32, 64	32
oepol	Select polarity of drive signals for data pads. 0 = active low, 1 = active high.	0 - 1	0
mobile	Enable Mobile SDRAM support 0: Mobile SDRAM support disabled 1: Mobile SDRAM support enabled but not default 2: Mobile SDRAM support enabled by default 3: Mobile SDRAM support only (no regular SDR support)	0 - 3	0

71.16 Signal descriptions

Table 1013 shows the interface signals of the core (VHDL ports).

Table 1013.**Signal descriptions**

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
MEMI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Bus ready strobe	Low
	BEXCN	Input	Bus exception	Low
	WRN[3:0]	Input	SRAM write enable feedback signal	Low
	BWIDTH[1:0]	Input	Sets the reset value of the PROM data bus width field in the MCFG1 register	High
	SD[31:0]	Input	SDRAM separate data bus	High
MEMO	ADDRESS[31:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data	-
	SDDATA[63:0]	Output	Sdram memory data	-
	RAMSN[4:0]	Output	SRAM chip-select	Low
	RAMOEN[4:0]	Output	SRAM output enable	Low
	IOSN	Output	Local I/O select	Low
	ROMSN[1:0]	Output	PROM chip-select	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0].	Low
	MBEN[3:0]	Output	Byte enable: MBEN[0] corresponds to DATA[31:24], MBEN[1] corresponds to DATA[23:16], MBEN[2] corresponds to DATA[15:8], MBEN[3] corresponds to DATA[7:0].	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus.Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0].	Low/High
	VBDRIVE[31:0]	Output	Vectored I/O-pad drive signals.	Low/High
	SVBDRIVE[63:0]	Output	Vectored I/O-pad drive signals for separate sdram bus.	Low/High
	READ	Output	Read strobe	High
	SA[14:0]	Output	SDRAM separate address bus	High
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

Table 1013.**Signal descriptions**

Signal name	Field	Type	Function	Active
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
WPROT	WPROTHIT	Input	Unused	-
SDO	SDCASN	Output	SDRAM column address strobe	Low
	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDDQM[7:0]	Output	SDRAM data mask: DQM[7] corresponds to DATA[63:56], DQM[6] corresponds to DATA[55:48], DQM[5] corresponds to DATA[47:40], DQM[4] corresponds to DATA[39:32], DQM[3] corresponds to DATA[31:24], DQM[2] corresponds to DATA[23:16], DQM[1] corresponds to DATA[15:8], DQM[0] corresponds to DATA[7:0].	Low
	SDRASN	Output	SDRAM row address strobe	Low
	SDWEN	Output	SDRAM write enable	Low

* see GRLIB IP Library User’s Manual

71.17 Library dependencies

Table 1014 shows libraries used when instantiating the core (VHDL libraries).

Table 1014.**Library dependencies**

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals Components	Memory bus signals definitions SDMCTRL component
ESA	MEMORYCTRL	Component	Memory controller component declaration

71.18 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined on the example designs port map and connected to the memory controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

Memory controller decodes default memory areas: PROM area is 0x0 - 0x1FFFFFFF, I/O-area is 0x20000000-0x3FFFFFFF and RAM area is 0x40000000 - 0x7FFFFFFF. SDRAM controller is enabled. SDRAM clock is synchronized with system clock by clock generator.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
```

```
library esa;
use esa.memoryctrl.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;

    -- memory bus
    address  : out   std_logic_vector(27 downto 0); -- memory bus
    data     : inout std_logic_vector(31 downto 0);
    ramsn    : out   std_logic_vector(4 downto 0);
    ramoen   : out   std_logic_vector(4 downto 0);
    rwen     : inout std_logic_vector(3 downto 0);
    romsn    : out   std_logic_vector(1 downto 0);
    iosn     : out   std_logic;
    oen      : out   std_logic;
    read     : out   std_logic;
    writen   : inout std_logic;
    brdyn    : in    std_logic;
    bexcn    : in    std_logic;
-- sdram i/f
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo : sdram_out_type;

  signal wprot : wprot_out_type;  -- dummy signal, not used
  signal clkm, rstn : std_ulogic; -- system clock and reset

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;

  signal gnd : std_ulogic;

begin

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                              tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  -- Memory controller
  mctrl0 : mctrl generic map (srbanks => 1, sden => 1)
```

```
      port map (rstn, clkm, memi, memo, ahbsi, ahbso(0), apbi, apbo(0), wprot, sdo);

    -- memory controller inputs not used in this configuration
    memi.brdyn <= ’1’; memi.bexcn <= ’1’; memi.wrn <= "1111";
    memi.sd <= sd;

    -- prom width at reset
    memi.bwidth <= "10";

    -- I/O pads driving data memory bus data signals
    datapads : for i in 0 to 3 generate
        data_pad : iopadv generic map (width => 8)
        port map (pad => data(31-i*8 downto 24-i*8),
                  o => memi.data(31-i*8 downto 24-i*8),
                  en => memo.bdrive(i),
                  i => memo.data(31-i*8 downto 24-i*8));
    end generate;

    -- connect memory controller outputs to entity output signals
    address <= memo.address; ramsn <= memo.ramsn; romsn <= memo.romsn;
    oen <= memo.oen; rwen <= memo.wrn; ramoen <= "1111" & memo.ramoen(0);
    sa <= memo.sa;
    writen <= memo.writen; read <= memo.read; iosn <= memo.iosn;
    sdcke <= sdo.sdcke; sdwen <= sdo.sdwen; sdcsn <= sdo.sdcsn;
    sdrasn <= sdo.rasn; sdcasn <= sdo.casn; sddqm <= sdo.dqm;
  end;
```

72 MEMSCRUB - AHB Memory Scrubber and Status Register

72.1 Overview

The memory scrubber monitors an AMBA AHB bus for accesses triggering an error response, and for correctable errors signaled from fault tolerant slaves on the bus. The core can be programmed to scrub a memory area by reading through the memory and writing back the contents using a locked read-write cycle whenever a correctable error is detected. It can also be programmed to initialize a memory area to known values.

The memory scrubber core is largely backwards compatible with the AHBSTAT core, and can replace it in many cases. Unlike AHBSTAT, the scrubber's registers are accessed through the AMBA AHB bus.

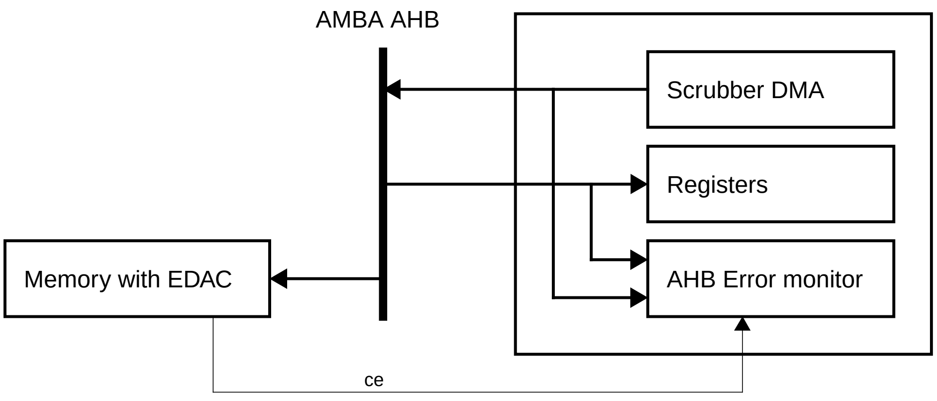


Figure 232. Memory scrubber block diagram

72.2 Operation

72.2.1 Errors

All AMBA AHB bus transactions are monitored and current HADDR, HWRITE, HMASTER and HSIZE values are stored internally. When an error response (HRESP = “01”) is detected, an internal counter is increased. When the counter exceeds a user-selected threshold, the status and address register contents are frozen and the New Error (NE) bit is set to one. At the same time an interrupt is generated, as described hereunder.

The default threshold is zero and enabled on reset so the first error on the bus will generate an interrupt.

Note that many of the fault tolerant units containing EDAC signal an un-correctable error as an AMBA error response, so that it can be detected by the processor as described above.

72.2.2 Correctable errors

Not only error responses on the AHB bus can be detected. Many of the fault tolerant units containing EDAC have a correctable error signal which is asserted each time a correctable error is detected. When such an error is detected, the effect will be the same as for an AHB error response. The only difference is that the Correctable Error (CE) bit in the status register is set to one when a correctable error is detected. Correctable and uncorrectable errors use separate counters and threshold values.

When the CE bit is set, the interrupt routine can acquire the address containing the correctable error from the failing address register and correct it. When it is finished it resets the CE bit and the monitoring becomes active again. Interrupt handling is described in detail hereunder.

The correctable error signals from the fault tolerant units should be connected to the *scrubi.cerror* input signal vector of the AHB status register core, which is or-ed internally and if the resulting signal is asserted, it will have the same effect as an AHB error response.

72.2.3 Scrubbing

The memory scrubber can be commanded to scrub a certain memory area, by writing a start and end address to the scrubbers start/end registers, followed by writing “00” to the scrub mode field and ‘1’ to the scrub enable bit in the scrubber control register.

After starting, the core will proceed to read the memory region in bursts. The burst size is fixed (set by the *burstlen* generic) and typically tuned to match the cache-line size or native block size of the slave. When a correctable error is detected, the scrubber performs a locked read-write cycle to correct the error, and then resumes the scrub operation.

If the correctable error detected is in the middle of a burst, the following read in the burst is completed before the read-write cycle begins. The core can handle the special case where that access also had a correctable error within the same locked scrub cycle.

If an uncorrectable error is detected, that location is left untouched.

Note that the status register functionality is running in parallel with the scrubber, so correctable and uncorrectable errors will be logged as usual. To prevent double logging, the core masks out the (expected) correctable error arising during the locked correction cycle.

To allow normal access to the bus, the core sleeps for a number of cycles between each burst. The number of cycles can be adjusted in the config register.

If the ID bit is set in the config register, the core will interrupt when the complete scrub is done.

72.2.4 Scrubber error counters

The core keeps track of the number of correctable errors detected during the current scrub run and the number of errors detected during processing of the current “count block”. The size of the count block is a fixed power of two equal or larger than the burst length (set by the *countlen* generic).

The core can be set up to interrupt when the counters exceed given thresholds. When this happens, the NE bit, plus one of the SEC/SBC bits, is set in the status register.

72.2.5 External start and clear

If the ES bit is set in the config register, the scrub enable bit is set automatically when the start input signal goes high. This can be used to set up periodic scrubbing.

The external input signal clrerr can be used to clear the global error counters. If this is connected to a timer, it is possible to count errors that have occurred within a specific unit of time. This signal can be disabled through the EC bit in the config register.

72.2.6 Memory regeneration

The regeneration mode performs the same basic function as the scrub mode, but is optimised for the case where many (or all) locations have correctable errors.

In this mode, the whole memory area selected is scrubbed using locked read/write bursts.

If an uncorrectable error is encountered during the read burst, that burst block is processed once again using the regular scrub routine, and the regeneration mode resumes on the following block. This avoids overwriting uncorrectable error locations.

72.2.7 Initialization

The scrubber can be used to write a pre-defined pattern to a block of memory. This is often necessary on EDAC memory before it can be used.

Before running the initialization, the pattern to be written to memory should be written into the scrubber initialization data register. The pattern has the same size as the burst length, so the corresponding number of writes to the initialization data register must be made.

72.2.8 Interrupts

The interrupt is generated on the line selected by the *hirq* VHDL generic. The interrupt is connected to the interrupt controller to inform the processor of the event.

After an interrupt is generated, either the NE bit or the DONE bit in the status register is set, to indicate which type of event caused the interrupt.

The normal procedure is that an interrupt routine handles the error with the aid of the information in the status registers. When it is finished it resets the NE bit in the AHB status register or the DONE bit in the scrubber status register, and the monitoring becomes active again. Error interrupts can be generated for both AMBA error responses and correctable errors as described above.

72.2.9 Mode switching

Switching between scrubbing and regeneration modes can be done on the fly during a scrub by modifying the MODE field in the scrubber configuration register. The mode change will take effect on the following scrub burst.

If the address range needs to be changed, then the core should be stopped before updating the registers. This is done by clearing the SCEN bit, and waiting for the ACTIVE bit in the status register to go low. An exception is when making the range larger (i.e. increasing the end address or decreasing the start address), as this can be done on the fly.

72.2.10 Dual range support

The scrubber can work over two non-overlapping memory ranges. This feature is enabled by writing the start/end addresses of the second range into the scrubber's second range start/end registers and setting the SERA bit in the configuration register. The two address ranges should not overlap.

72.3 Registers

The core is programmed through registers mapped into an I/O region in the AHB address space. Only 32-bit accesses are supported.

*Table 1015.*Memory scrubber registers

AHB address offset	**Registers**
0x00	AHB Status register
0x04	AHB Failing address register
0x08	AHB Error configuration register
0x0C	Reserved
0x10	Scrubber status register
0x14	Scrubber configuration register
0x18	Scrubber range low address register
0x1C	Scrubber range high address register
0x20	Scrubber position register
0x24	Scrubber error threshold register
0x28	Scrubber initialization data register
0x2C	Scrubber second range start address register
0x30	Scrubber second range end address register

Table 1016. AHB Status register

31 22	21 14	13	12	11	10	9	8	7	6 3	2 0
CECNT	UECNT	DONE	RES	SEC	SBC	CE	NE	HWRITE	HMASTER	HSIZE

31: 22	CECNT: Global correctable error count
21: 14	UECNT: Global uncorrectable error count
13	DONE: Scrubber run completed. (read-only) This is a read-only copy of the DONE bit in the scrubber status register.
12	RESERVED
11	SEC: Scrubber error counter threshold exceeded. Asserted together with NE.
10	SBC: Scrubber block error counter threshold exceeded. Asserted together with NE.
9	CE: Correctable Error. Set if the detected error was caused by a correctable error and zero otherwise.
8	NE: New Error. Deasserted at start-up and after reset. Asserted when an error is detected. Reset by writing a zero to it.
7	The HWRITE signal of the AHB transaction that caused the error.
6: 3	The HMASTER signal of the AHB transaction that caused the error.
2: 0	The HSIZE signal of the AHB transaction that caused the error

Table 1017. AHB Failing address register

31 0
AHB FAILING ADDRESS

31: 0	The HADDR signal of the AHB transaction that caused the error.

Table 1018. AHB Error configuration register

31 22	21 14	13 2	1	0
CORRECTABLE ERROR COUNT THRESHOLD	UNCORR. ERROR COUNT THRESH.	RESERVED	CECTE	UECTE

31: 22	Interrupt threshold value for global correctable error count
21: 14	Interrupt threshold value for global uncorrectable error count
13: 2	RESERVED
1	CECTE: Correctable error count threshold enable
0	UECTE: Uncorrectable error count threshold enable

Table 1019. Scrubber status register

31 22	21 14	13	12 5	4 1	0
SCRUB RUN ERROR COUNT	BLOCK ERROR COUNT	DONE	RESERVED	BURSTLEN	ACTIVE

31: 22	Number of correctable errors in current scrub run (read-only(
21: 14	Number of correctable errors in current block (read-only)
13	DONE: Scrubber run completed.
	Needs to be cleared (by writing zero) before a new scrubber done interrupt can occur.
12: 5	RESERVED
4: 1	Burst length in 2-log of AHB bus cycles; “0000”=1, “0001”=2, “0010”=4, “0011”=8, ...
0	Current scrubber state: 0=Idle, 1=Running (read-only)

Table 1020. Scrubber configuration register

31 16	15 8	7	6	5	4	3 2	1	0
RESERVED	DELAY	IRQD	EC	SERA	LOOP	MODE	ES	SCEN

31: 16	RESERVED
15: 8	Delay time between processed blocks, in cycles
7	Interrupt when scrubber has finished
6	External clear counter enable
5	Second memory range enable
4	Loop mode, restart scrubber when run finishes
3: 2	Scrubber mode (00=Scrub, 01=Regenerate, 10=Initialize, 11=Undefined)
1	External start enable
0	Scrubber enable

Table 1021. Scrubber range low address register

31 0
SCRUBBER RANGE LOW ADDRESS

31: 0	The lowest address in the range to be scrubbed
	The address bits below the burst size alignment are constant ‘0’

Table 1022. Scrubber range high address register

31 0
SCRUBBER RANGE HIGH ADDRESS

31: 0	The highest address in the range to be scrubbed
	The address bits below the burst size alignment are constant ‘1’

Table 1023. Scrubber position register

31 … 0
SCRUBBER POSITION

31: 0	The current position of the scrubber while active, otherwise zero. The address bits below the burst size alignment are constant ‘0’

Table 1024. Scrubber error threshold register

31 … 22	21 … 14	13 … 2	1	0
SCRUB RUN ERROR COUNT THRESHOLD	BLOCK ERROR COUNT THRESH.	RESERVED	RECTE	BECTE

31: 22	Interrupt threshold value for current scrub run correctable error count
21: 14	Interrupt threshold value for current scrub block correctable error count
13: 2	RESERVED
1	RECTE: Scrub run correctable error count threshold enable
0	BECTE: Scrub block uncorrectable error count threshold enable

Table 1025. Scrubber initialization data register (write-only)

31 … 0
SCRUBBER INITIALIZATION DATA

31: 0	Part of data pattern to be written in initialization mode.

Table 1026. Scrubber second range low address register

31 … 0
SCRUBBER RANGE LOW ADDRESS

31: 0	The lowest address in the second range to be scrubbed (if SERA=1) The address bits below the burst size alignment are constant ‘0’

Table 1027. Scrubber second range high address register

31 … 0
SCRUBBER RANGE HIGH ADDRESS

31: 0	The highest address in the second range to be scrubbed (if SERA=1) The address bits below the burst size alignment are constant ‘1’

72.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x057. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

72.5 Configuration options

Table 1028 shows the configuration options of the core (VHDL generics).

*Table 1028.*Configuration options

Generic	Function	Allowed range	Default
hmindex	AHB master index	0 - NAHBMST-1	0
hsindex	AHB slave index	0 - NAHBSLV-1	0
ioaddr	AHB slave register area address	0 - 16#FFF#	0
iomask	AHB slave register area address mask	0 - 16#FFF#	16#FFF#
hirq	Interrupt line driven by the core	0 - 16#FFF#	0
nftslv	Number of FT slaves connected to the cerror vector	1 - NAHBSLV-1	3
memwidth	Width of accesses to scrubbed memory slave in bits	32, 64, ..., 1024	AHBDW
burstlen	Length of burst accesses to scrubbed memory slave	2, 4, 8, 16, ...	2
countlen	Length of blocks used for block error count	burstlen x (1,2,4,8 ...)	8

72.6 Signal descriptions

Table 1029 shows the interface signals of the core (VHDL ports).

*Table 1029.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signal	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SCRUBI	CERROR	Input	Correctable Error Signals	High
	CLRCOUNT	Input	Clear global error counters	High
	START	Input	External start signal	High

* see GRLIB IP Library User’s Manual

72.7 Library dependencies

Table 1030 shows libraries used when instantiating the core (VHDL libraries).

*Table 1030.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MISC	Component	Component declaration

72.8 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory scrubber. There are three Fault Tolerant units with EDAC connected to the scrubber’s *cerror* vector. The connection of the different memory controllers to external memory is not shown.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;
    --other signals
    ....
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo, sdo2: sdctrl_out_type;

  signal sdi : sdctrl_in_type;

  signal aramo : ahbram_out_type;

  -- correctable error vector
  signal scrubi : memscrub_in_type;

begin

  -- AMBA Components are defined here ...

  -- AHB Memory Scrubber and Status Register
  mscrub0 : memscrub
    generic map(hmindex => 4, hsindex => 5, ioaddr => 16#C00#,
                hirq => 11, nftslv => 3);
    port map(rstn, clkm, ahbmi, ahbmo(4), ahbsi, ahbso(5), scrubi);

  scrubi.start <= ’0’; scrubi.clrcount <= ’0’;
  scrubi.cerror(3 to NAHBSLV-1) <= (others => ’0’);

  --FT AHB RAM
  a0 : ftahbram
    generic map(hindex => 1, haddr => 1, tech => inferred,  kbytes => 64,
                pindex => 4, paddr => 4, edacen => 1, autoscrub => 0,
                errcnt => 1, cntbits => 4)
    port map(rst, clk, ahbsi, ahbso(1), apbi, apbo(4), aramo);

  scrubi.cerror(0) <= aramo.ce;

  -- SDRAM controller
  sdc : ftsdctrl
    generic map (hindex => 3, haddr => 16#600#, hmask => 16#F00#, ioaddr => 1,
                 fast => 0, pwron => 1, invclk => 0, edacen => 1, errcnt => 1,
                 cntbits => 4)
    port map (rstn, clk, ahbsi, ahbso(3), sdi, sdo);
  stati.cerror(1) <= sdo.ce;

  -- Memory controller
  mctrl0 : ftsrctrl
    generic map (rmw => 1, pindex => 10, paddr => 10, edacen => 1, errcnt => 1,
                 cntbits => 4)
    port map (rstn, clk, ahbsi, ahbso(0), apbi, apbo(10), memi, memo, sdo2);

  scrubi.cerror(2) <= memo.ce;

end;
```

73 MUL32 - Signed/unsigned 32x32 multiplier module

73.1 Overview

The multiplier module is highly configurable module implementing 32x32 bit multiplier. Multiplier takes two signed or unsigned numbers as input and produces 64-bit result. Multiplication latency and hardware complexity depend on multiplier configuration. Variety of configuration option makes it possible to configure the multiplier to meet wide range of requirements on complexity and performance.

For DSP applications the module can be configured to perform multiply & accumulate (MAC) operation. In this configuration 16x16 multiplication is performed and the 32-bit result is added to 40-bit value accumulator.

73.2 Operation

The multiplication is started when ‘1’ is samples on MULI.START on positive clock edge. Operands are latched externally and provided on inputs MULI.OP1 and MULI.OP2 during the whole operation. The result appears on the outputs during the clock cycle following the clock cycle when MULO.READY is asserted if multiplier if 16x16, 32x8 or 32x16 configuration is used. For 32x32 configuration result appears on the output during the second clock cycle after the MULI.START was asserted.

Signal MULI.MAC shall be asserted to start multiply & accumulate (MAC) operation. This signal is latched on positive clock edge. Multiplication is performed between two 16-bit values on inputs MULI.OP1[15:0] and MULI.OP2[15:0]. The 32-bit result of the multiplication is added to the 40-bit accumulator value on signal MULI.ACC to form a 40-bit value on output MULO.RESULT[39:0]. The result of MAC operation appears during the second clock cycle after the MULI.MAC was asserted.

73.3 Synthesis

Table 1031 shows hardware complexity in ASIC gates and latency for different multiplier configurations.

*Table 1031.*Multiplier latencies and hardware complexity

Multiplier size (multype)	**Pipelined (pipe)**	**Latency (clocks)**	**Approximate area (gates)**
16x16	1	5	6 500
16x16	0	4	6 000
32x8	-	4	5 000
32x16	-	2	9 000
32x32	-	1	15 000

73.4 Configuration options

Table 1032 shows the configuration options of the core (VHDL generics).

*Table 1032.*Configuration options

Generic	Function	Allowed range	Default
tech	Multiplier technolofy selection. If set to 0 the multipliers will be inferred by the synthesis tool. Use this option if your synthesis tool i capable of inferring efficient multiplier implementation.	0 to NTECH-1	0
multype	Size of the multiplier that is actually implemented. All configuration produce 64-bit result with different latencies. 0 - 16x16 bit multiplier 1 - 32x8 bit multiplier 2 - 32x16 bit multiplier 3 - 32x32 bit multiplier	0 to 3	0
pipe	Used in 16x16 bit multiplier configuration. Adds a pipeline register stage to the multiplier. This option gives better timing but adds one clock cycle to latency.	0 to 1	0
mac	Enable multiply & accumulate operation. Use only with 16x16 multiplier option with no pipelining (*pipe* = 0)	0 to 1	0
arch	Multiplier structure 0: Inferred by synthesis tool 1: Generated using Module Generators from NTNU 2: Using technology specific netlists (techspec) 3: Using Synopsys DesignWare (DW02_mult and DW_mult_pipe)	0 to 3	0

73.5 Signal descriptions

Table 1033 shows the interface signals of the core (VHDL ports).

*Table 1033.*Signal declarations

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
HOLDN	N/A	Input	Hold	Low
MULI	OP1[32:0]	Input	Operand 1 OP1[32] - Sign bit. OP1[31:0] - Operand 1 in 2’s complement format	High
	OP2[32:0]		Operand 2 OP2[32] - Sign bit. OP2[31:0] - Operand 2in 2’s complement format	High
	FLUSH		Flush current operation	High
	SIGNED		Signed multiplication	High
	START		Start multiplication	High
	MAC		Multiply & accumulate	High
	ACC[39:0]		Accumulator. Accumulator value is held externally.	High
MULO	READY	Output	Result is ready during the next clock cycle for 16x16, 32x8 and 32x16 configurations. Not used for 32x32 configuration or MAC operation.	High
	NREADY		Not used	-
	ICC[3:0]		Condition codes ICC[3] - Negative result (not used in 32x32 conf) ICC[1] - Zero result (not used in 32x32 conf) ICC[1:0] - Not used	High
	RESULT[63:0]		Result. Available at the end of the clock cycle if MULO.READY was asserted in previous clock cycle. For 32x32 configuration the result is available during second clock cycle after the MULI.START was asserted.	High

73.6 Library dependencies

Table 1034 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1034.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	ARITH	Signals, component	Signals, component declaration

73.7 Component declaration

The core has the following component declaration.

```
component mul32
generic (
    infer   : integer := 1;
    multype : integer := 0;
    pipe    : integer := 0;
    mac     : integer := 0
);
port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    holdn   : in  std_ulogic;
    muli    : in  mul32_in_type;
    mulo    : out mul32_out_type
);
end component;
```

73.8 Instantiation

This example shows how the core can be instantiated.

The module is configured to implement 16x16 pipelined multiplier with support for MAC operations.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use gaisler.arith.all;

.
.
.

signal muli  : mul32_in_type;
signal mulo  : mul32_out_type;

begin

mul0 : mul32 generic map (infer => 1, multype => 0, pipe => 1, mac => 1)
    port map (rst, clk, holdn, muli, mulo);


end;
```

74 MULTLIB - High-performance multipliers

74.1 Overview

The GRLIB.MULTLIB VHDL-library contains a collection of high-performance multipliers from the Arithmetic Module Generator at Norwegian University of Science and Technology. 32x32, 32x8, 32x16, 16x16 unsigned/signed multipliers are included. 16x16-bit multiplier can be configured to include a pipeline stage. This option improves timing but increases latency with one clock cycle.

74.2 Configuration options

Table 1035 shows the configuration options of the core (VHDL generics).

*Table 1035.*Configuration options

Generic	Function	Allowed range	Default
mulpipe	Include a pipeline stage (0 -pipelining disabled, 1 - pipelining enabled)	0 - 1	0

74.3 Signal descriptions

Table 1036 shows the interface signals of the core (VHDL ports).

*Table 1036.*Signal descriptions

Signal name	Type	Function	Active
CLK (16x16 multiplier only)	Input	Clock	-
HOLDN (16x16 multiplier only)	Input	Hold. When active, the pipeline register is not updates	Low
X[16:0] (16x16 mult) X[32:0] (32x8 mult) X[32:0] (32x16 mult) X[32:0] (32x32 mult)	Input	Operand 1. MBS bit is sign bit.	High
Y[16:0] (16x16 mult) Y[8:0] (32x8 mult) Y[16:0] (32x16 mult) Y[32:0] (32x32 mult)	Input	Operand 2. MSB bit is sign bit.	High
P[33:0] (16x16 mult) P[41:0] (32x8 mult) P[49:0] (32x16 mult) P[65:0] (32x32 mult)		Result. Two MSB bits are sign bits.	High

74.4 Library dependencies

Table 1037 shows libraries used when instantiating the core (VHDL libraries).

*Table 1037.*Library dependencies

Library	Package	Imported unit	Description
GRLIB	MULTLIB	Component	Multiplier component declarations

74.5 Component declaration

The core has the following component declaration.

```
component mul_33_33
  port (
    x    : in  std_logic_vector(32 downto 0);
    y    : in  std_logic_vector(32 downto 0);
    p    : out std_logic_vector(65 downto 0)
  );
end component;
```

74.6 Instantiation

This example shows how the core can be instantiated.

The core is configured to implement 16x16 pipelined multiplier with support for MAC operations.

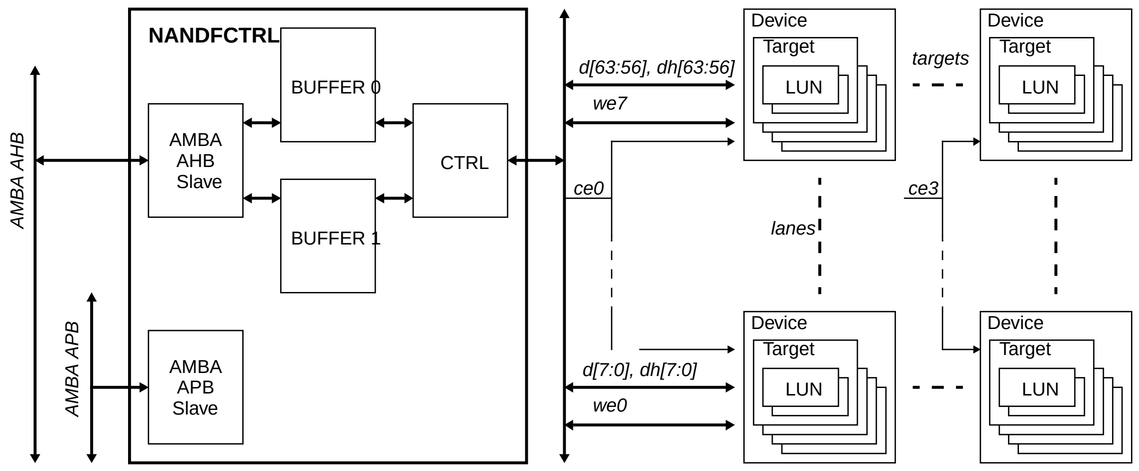
```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.multlib.all;

.
.

signal op1, op2 : std_logic_vector(32 downto 0);
signal prod : std_logic_vector(65 downto 0);

begin

m0 : mul_33_33
         port map (op1, op2, prod);

end;
```

75 NANDFCTRL - NAND Flash Memory Controller

75.1 Overview

The NAND Flash Memory Controller (NANDFCTRL) core provides a bridge between external NAND flash memory and the AMBA bus. The memory controller is an Open NAND Flash Interface (ONFI) 2.2 command compliant core (see exceptions below) and it can communicate with multiple parallel flash memory devices simultaneously, where each device in turn can consist of up to four individually addressable targets, one target addressed at a time. The core is configured through a set of AMBA APB registers, described in section 75.5, and data is written to / read from the flash memory by accessing internal buffers mapped over AMBA AHB.

This document mainly describes the NANDFCTRL core’s functionality. For details about the actual flash memory interface, flash memory architecture and ONFI 2.2 command set please refer to the *Open NAND Flash Interface specification, revision 2.2,* hereafter called the ONFI 2.2 specification.



Note: One device might have more than one target, using one chip enable signal for each target.
This will reduce the number devices that can be placed horizontally in the figure.
All devices (and the internal targets) placed vertically in the figure belong to the same chip enable signal,
with individual write enable signals controlling each 8-bit/16-bit data lane.

Figure 233. Block diagram

75.2 Operation

75.2.1 System overview

A block diagram of the core can be seen in figure 233. Features and limitations of the core are listed below:

- All commands defined in the ONFI 2.2 standard are supported, except Synchronous Reset.
- The core does not implement support for the source synchronous data interface, only asynchronous data interface.
- The core does not place any other limitation on the device architecture other than those specified in the ONFI 2.2 standard. For example, the core does not need to know how many LUNs, blocks, or pages a connected flash memory device has. (See the ONFI 2.2 specification for information about LUNs, blocks, and pages.)
- Multiple parallel data lanes are supported, which gives the possibility to read / write several flash memory devices at the same time.
- To support interleaving of flash memory accesses and AMBA accesses and give greater throughput, two buffers for reading / writing flash memory data are implemented.

- The data interface timing can either be fixed (set at implementation time) or programmable through AMBA APB registers. With fixed data interface timing support for ONFI timing modes 0 - 5 can be implemented and then switched between during run-time. Programmable timing interface allows for clock frequencies unknown at implementation time, as well as custom timing.
- The core does not implement any wear-leveling or bad block management.
- The core does not implement any direct access to flash memory devices from the AMBA AHB bus. Instead accesses are performed through two temporary buffers by initiating command and data transfers through control registers via the AMBA APB bus.
- The temporary buffers are mapped into AMBA AHB memory space. Note that the buffer addresses are not directly mapped to the flash memory, but are mapped specific addressing registers. This allows large amount of flash memory to be address, exceeding the 4 Gbyte address space of the AMBA bus.

75.2.2 Internal buffer structure

The number of buffers implemented in the core depends on whether or not the core is implemented with separate buffers for each parallel 8-bit/16-bit data lane or not. This is indicated by the *sepb* field in the *Capability register.* How many data lanes the core implements can be found by reading the *nlane* field of the *Capability register* and adding one. If separate buffers are used then for each data lane there are four different buffers implemented. Two buffers are used for data that are read from / written to any page area in the flash memory device (hereafter called *page buffer*), and the other two are used for data read from / written to any page's spare area (hereafter called *spare buffer*). The size of each page buffer (in bytes) is $2^{\wedge(\text{pbits}+1)}$, where pbits is the value of the *pbits* field in the *Capability register.* The size of each spare buffer (in bytes) is $2^{\wedge(\text{sbits}+1)}$, where sbits is the value of the *sbits* field in the *Capability register.* If separate buffers aren't used, then the core implements one set of the above mentioned four buffers and uses them for all data lanes.

One page buffer and one spare buffer for each data lane (or all lanes if separate buffers aren't used) are grouped together into what in this document is called *buffer 0.* The other set of page buffers and spare buffers are grouped into *buffer 1.* For example, if the core has support for eight data lanes with separate buffers, and the page buffers are 4096 bytes, and the spare buffers are 256 bytes, then buffer 0 and buffer 1 will each be 32768 + 2048 bytes large. Buffer 0 and buffer 1 are associated with their own set of control registers, described in section 75.5.

Note that support for buffer 1 is optional. Whether or not support for buffer 1 is implemented is indicated by the *b1en* bit in the *Capability register.*

All buffers are mapped into AMBA AHB address space, and the core supports two different mapping schemes, with their own designated AMBA AHB address space:

- For the first address map, called *Consecutive address map,* the first part of the assigned AHB memory area is mapped to the page buffer corresponding to the first 8-bit/16-bit data lane, followed by the page buffer corresponding to the second 8-bit/16-bit data lane etc. After all the page buffers the spare buffers follow in the same manner. See table 1038 for an example with 4096 bytes page buffer and 256 bytes spare buffer.
- For the second address map, called *By page address map,* the first part of the assigned AHB memory area is mapped to the page buffer corresponding to the first 8-bit/16-bit data lane, followed by the spare buffer corresponding to the same data lane. After that follows the page buffer and spare buffer pairs for all other data lanes. Note that since the spare buffers normally are much smaller than the page buffers there is normally a gap between the last address of a spare buffer and the first address of the next page buffer. The size of the gap is page buffer size - spare buffer size. See table 1039 for an example with 4096 bytes page buffer and 256 bytes spare buffer.

Note that the page and spare buffer areas might be larger than the size of the page and spare memory implemented in the actual flash memory device. Thus, there might be gaps in the addressing schemes.

If the EDAC is implemented (indicated by the *edac* bit in the *Capability register*) then the buffers have an additional AHB address map. Read and write accesses to the additional AHB address space will trigger EDAC operation. This additional address map is also shown in table 1038 and table 1039.

Table 1038. Buffer memory map, consecutive addressing. Example with 4096 B page size, 256 B spare size, separate buffers, both buffer 0 and buffer 1 implemented, and with EDAC.

AMBA AHB address offset		**Contents**	**AMBA AHB address offset**		**Contents**
Normal address space (no EDAC operation)			**Normal address space (no EDAC operation)**		
0x00000	0x00FFF	Buffer 0, 4096 byte page data, lane 0	0x10000	0x10FFF	Buffer 1, 4096 byte page data, lane 0
0x01000	0x01FFF	Buffer 0, 4096 byte page data, lane 1	0x11000	0x11FFF	Buffer 1, 4096 byte page data, lane 1
0x02000	0x02FFF	Buffer 0, 4096 byte page data, lane 2	0x12000	0x12FFF	Buffer 1, 4096 byte page data, lane 2
0x03000	0x03FFF	Buffer 0, 4096 byte page data, lane 3	0x13000	0x13FFF	Buffer 1, 4096 byte page data, lane 3
0x04000	0x04FFF	Buffer 0, 4096 byte page data, lane 4	0x14000	0x14FFF	Buffer 1, 4096 byte page data, lane 4
0x05000	0x05FFF	Buffer 0, 4096 byte page data, lane 5	0x15000	0x15FFF	Buffer 1, 4096 byte page data, lane 5
0x06000	0x06FFF	Buffer 0, 4096 byte page data, lane 6	0x16000	0x16FFF	Buffer 1, 4096 byte page data, lane 6
0x07000	0x07FFF	Buffer 0, 4096 byte page data, lane 7	0x17000	0x17FFF	Buffer 1, 4096 byte page data, lane 7
0x08000	0x080FF	Buffer 0, 256 byte spare data, lane 0	0x18000	0x180FF	Buffer 1, 256 byte spare data, lane 0
0x08100	0x081FF	Buffer 0, 256 byte spare data, lane 1	0x18100	0x181FF	Buffer 1, 256 byte spare data, lane 1
0x08200	0x082FF	Buffer 0, 256 byte spare data, lane 2	0x18200	0x182FF	Buffer 1, 256 byte spare data, lane 2
0x08300	0x083FF	Buffer 0, 256 byte spare data, lane 3	0x18300	0x183FF	Buffer 1, 256 byte spare data, lane 3
0x08400	0x084FF	Buffer 0, 256 byte spare data, lane 4	0x18400	0x184FF	Buffer 1, 256 byte spare data, lane 4
0x08500	0x085FF	Buffer 0, 256 byte spare data, lane 5	0x18500	0x185FF	Buffer 1, 256 byte spare data, lane 5
0x08600	0x086FF	Buffer 0, 256 byte spare data, lane 6	0x18600	0x186FF	Buffer 1, 256 byte spare data, lane 6
0x08700	0x087FF	Buffer 0, 256 byte spare data, lane 7	0x18700	0x187FF	Buffer 1, 256 byte spare data, lane 7
EDAC address space (EDAC operation on page buffers)			**EDAC address space (EDAC operation on page buffers)**		
0x20000	0x20FFF	Buffer 0, 4096 byte page data, lane 0	0x30000	0x30FFF	Buffer 1, 4096 byte page data, lane 0
0x21000	0x21FFF	Buffer 0, 4096 byte page data, lane 1	0x31000	0x31FFF	Buffer 1, 4096 byte page data, lane 1
0x22000	0x22FFF	Buffer 0, 4096 byte page data, lane 2	0x32000	0x32FFF	Buffer 1, 4096 byte page data, lane 2
0x23000	0x23FFF	Buffer 0, 4096 byte page data, lane 3	0x33000	0x33FFF	Buffer 1, 4096 byte page data, lane 3
0x24000	0x24FFF	Buffer 0, 4096 byte page data, lane 4	0x34000	0x34FFF	Buffer 1, 4096 byte page data, lane 4
0x25000	0x25FFF	Buffer 0, 4096 byte page data, lane 5	0x35000	0x35FFF	Buffer 1, 4096 byte page data, lane 5
0x26000	0x26FFF	Buffer 0, 4096 byte page data, lane 6	0x36000	0x36FFF	Buffer 1, 4096 byte page data, lane 6
0x27000	0x27FFF	Buffer 0, 4096 byte page data, lane 7	0x37000	0x37FFF	Buffer 1, 4096 byte page data, lane 7
0x28000	0x280FF	Buffer 0, 256 byte spare data, lane 0	0x38000	0x380FF	Buffer 1, 256 byte spare data, lane 0
0x28100	0x281FF	Buffer 0, 256 byte spare data, lane 1	0x38100	0x381FF	Buffer 1, 256 byte spare data, lane 1
0x28200	0x282FF	Buffer 0, 256 byte spare data, lane 2	0x38200	0x382FF	Buffer 1, 256 byte spare data, lane 2
0x28300	0x283FF	Buffer 0, 256 byte spare data, lane 3	0x38300	0x383FF	Buffer 1, 256 byte spare data, lane 3
0x28400	0x284FF	Buffer 0, 256 byte spare data, lane 4	0x38400	0x384FF	Buffer 1, 256 byte spare data, lane 4
0x28500	0x285FF	Buffer 0, 256 byte spare data, lane 5	0x38500	0x385FF	Buffer 1, 256 byte spare data, lane 5
0x28600	0x286FF	Buffer 0, 256 byte spare data, lane 6	0x38600	0x386FF	Buffer 1, 256 byte spare data, lane 6
0x28700	0x287FF	Buffer 0, 256 byte spare data, lane 7	0x38700	0x387FF	Buffer 1, 256 byte spare data, lane 7

Table 1039. Buffer memory map, by page addressing. Example with 4096 B page size, 256 B spare size, separate buffers, both buffer 0 and buffer 1 implemented, and with EDAC.

AMBA AHB address offset		**Contents**	**AMBA AHB address offset**		**Contents**
Normal address space (no EDAC operation)			**Normal address space (no EDAC operation)**		
0x00000	0x00FFF	Buffer 0, 4096 byte page data, lane 0	0x10000	0x10FFF	Buffer 1, 4096 byte page data, lane 0
0x01000	0x010FF	Buffer 0, 256 byte spare data, lane 0	0x11000	0x110FF	Buffer 1, 256 byte spare data, lane 0
0x02000	0x02FFF	Buffer 0, 4096 byte page data, lane 1	0x12000	0x12FFF	Buffer 1, 4096 byte page data, lane 1
0x03000	0x030FF	Buffer 0, 256 byte spare data, lane 1	0x13000	0x130FF	Buffer 1, 256 byte spare data, lane 1
0x04000	0x04FFF	Buffer 0, 4096 byte page data, lane 2	0x14000	0x14FFF	Buffer 1, 4096 byte page data, lane 2
0x05000	0x050FF	Buffer 0, 256 byte spare data, lane 2	0x15000	0x150FF	Buffer 1, 256 byte spare data, lane 2
0x06000	0x06FFF	Buffer 0, 4096 byte page data, lane 3	0x16000	0x16FFF	Buffer 1, 4096 byte page data, lane 3
0x07000	0x070FF	Buffer 0, 256 byte spare data, lane 3	0x17000	0x170FF	Buffer 1, 256 byte spare data, lane 3
0x08000	0x08FFF	Buffer 0, 4096 byte page data, lane 4	0x18000	0x18FFF	Buffer 1, 4096 byte page data, lane 4
0x09000	0x090FF	Buffer 0, 256 byte spare data, lane 4	0x19000	0x190FF	Buffer 1, 256 byte spare data, lane 4
0x0A000	0x0AFFF	Buffer 0, 4096 byte page data, lane 5	0x1A000	0x1AFFF	Buffer 1, 4096 byte page data, lane 5
0x0B000	0x0B0FF	Buffer 0, 256 byte spare data, lane 5	0x1B000	0x1B0FF	Buffer 1, 256 byte spare data, lane 5
0x0C000	0x0CFFF	Buffer 0, 4096 byte page data, lane 6	0x1C000	0x1CFFF	Buffer 1, 4096 byte page data, lane 6
0x0D000	0x0D0FF	Buffer 0, 256 byte spare data, lane 6	0x1D000	0x1D0FF	Buffer 1, 256 byte spare data, lane 6
0x0E000	0x0EFFF	Buffer 0, 4096 byte page data, lane 7	0x1E000	0x1EFFF	Buffer 1, 4096 byte page data, lane 7
0x0F000	0x0F0FF	Buffer 0, 256 byte spare data, lane 7	0x1F000	0x1F0FF	Buffer 1, 256 byte spare data, lane 7
EDAC address space (EDAC operation on page buffer)			**EDAC address space (EDAC operation on page buffer)**		
0x20000	0x20FFF	Buffer 0, 4096 byte page data, lane 0	0x30000	0x30FFF	Buffer 1, 4096 byte page data, lane 0
0x21000	0x210FF	Buffer 0, 256 byte spare data, lane 0	0x31000	0x310FF	Buffer 1, 256 byte spare data, lane 0
0x22000	0x22FFF	Buffer 0, 4096 byte page data, lane 1	0x32000	0x32FFF	Buffer 1, 4096 byte page data, lane 1
0x23000	0x230FF	Buffer 0, 256 byte spare data, lane 1	0x33000	0x330FF	Buffer 1, 256 byte spare data, lane 1
0x24000	0x24FFF	Buffer 0, 4096 byte page data, lane 2	0x34000	0x34FFF	Buffer 1, 4096 byte page data, lane 2
0x25000	0x250FF	Buffer 0, 256 byte spare data, lane 2	0x35000	0x350FF	Buffer 1, 256 byte spare data, lane 2
0x26000	0x26FFF	Buffer 0, 4096 byte page data, lane 3	0x36000	0x36FFF	Buffer 1, 4096 byte page data, lane 3
0x27000	0x270FF	Buffer 0, 256 byte spare data, lane 3	0x37000	0x370FF	Buffer 1, 256 byte spare data, lane 3
0x28000	0x28FFF	Buffer 0, 4096 byte page data, lane 4	0x38000	0x38FFF	Buffer 1, 4096 byte page data, lane 4
0x29000	0x290FF	Buffer 0, 256 byte spare data, lane 4	0x39000	0x390FF	Buffer 1, 256 byte spare data, lane 4
0x2A000	0x2AFFF	Buffer 0, 4096 byte page data, lane 5	0x3A000	0x3AFFF	Buffer 1, 4096 byte page data, lane 5
0x2B000	0x2B0FF	Buffer 0, 256 byte spare data, lane 5	0x3B000	0x3B0FF	Buffer 1, 256 byte spare data, lane 5
0x2C000	0x2CFFF	Buffer 0, 4096 byte page data, lane 6	0x3C000	0x3CFFF	Buffer 1, 4096 byte page data, lane 6
0x2D000	0x2D0FF	Buffer 0, 256 byte spare data, lane 6	0x3D000	0x3D0FF	Buffer 1, 256 byte spare data, lane 6
0x2E000	0x2EFFF	Buffer 0, 4096 byte page data, lane 7	0x3E000	0x3EFFF	Buffer 1, 4096 byte page data, lane 7
0x2F000	0x2F0FF	Buffer 0, 256 byte spare data, lane 7	0x3F000	0x3F0FF	Buffer 1, 256 byte spare data, lane 7

75.2.3 Data interface timing

The ONFI timing parameters that the core explicitly handles are:

- tCCS - Change Column setup time
- tADL - ALE to data loading time
- tCS - CE setup time
- tRP - RE pulse width
- tRR - Ready to RE low (data only)
- tWP - WE pulse width
- tWB - WE high to SR[6] low
- tRHW - RE high to we WE low
- tWH - WE high hold time
- tWHR - WE high to RE low
- tWW - WP transition to WE low

All other timing requirements are either fulfilled through design, or are handled by the flash memory devices. See the ONFI specification for details about the different timing parameters.

The data interface timing can be either fixed (set at implementation time) or programmable to allow both system clock frequencies unknown at implementation time as well as custom timing parameters.

When the timing is fixed all timing parameters are calculated on implementation time based on the specified system clock frequency. When programmable timing is used the timing parameters are programmed through AMBA APB registers, described in section 75.5. The registers power-up / reset values are set at implementation time.

Note that the timing parameter tCCS is always programmable, even if fixed timing is used for all other parameters. This is because the ONFI specification says that after initialization is complete, the value for tCCS specified in the flash memory's parameter page should be used.

75.2.4 Accessing the NAND flash memory devices

The steps that need to be taken to access (i.e. send an ONFI 2.2 command to) the flash memory devices are:

1. Make sure that the chosen buffer is not busy by checking the *run* and *bsy* bits in the *Buffer control / status* register. If both the *run* bit and *bsy* bits are set then the core is currently executing a command associated with that buffer, and the buffer can not be used. If only the *bsy* bit is set then the core is done executing a command but the corresponding data buffer and control bits are still write protected. Software then needs to clear the *bsy* bit by writing '1' to it.

2. If the command requires a row address to be sent, write it to the *Buffer row address register.* Otherwise this step can be skipped.

3. If the command requires a column address to be sent, write it to the *coladdr* field of the *Buffer column address register.* Bits 7:0 of the *coladdr* field also need to be written with the one byte address used for SET FEATURES, GET FEATURES, READ ID, READ UNIQUE ID, and READ PARAMETER PAGE commands. Otherwise this step can be skipped.

4. Write the command value to the *Buffer command register,* and possibly set the control bits *sel*, *cd*, or *sc2* if needed. See table 1047 in section 75.5 for a description of these bits.

5. If the command should include a data phase then set the size of the data by writing to the *size* field of the *Buffer column address register.* This step is not necessary for the SET FEATURES, GET FEATURES, READ STATUS, and READ STATUS ENHANCED commands since they always have a fixed size data phase.

6. If data should be written to the flash memory devices then write this data to the corresponding buffers. Note that the core uses the *coladdr* field in the *Buffer column address register* to index into the buffers, which means that data that should be written with an offset into a flash memory page (i.e. column address is not zero) need to be written with the same offset into the buffers. The exception is the commands mentioned in step 3. They always read from the beginning of the buffers. This step can be skipped if no data are to be written.

7. Select which data lanes and targets the command should be sent to, if an interrupt should be generated when the command is finished, and start execution by writing to the *lanesel*, *targsel*, *irqmsk*, and *exe* bits in the *Buffer control / status register.* See table 1048 in section 75.5 for a description of these bits.

Once command execution has been started software can monitor the *run* bit in the *Buffer control / status register* (or wait for an interrupt if the core was configured to generate one) to learn when the command is finished. If data was read from the flash memory then it can be found in the buffers. Note that the core uses the *coladdr* field in the *Buffer column address register* to index into the buffers, which means that data that was read from a flash memory page with an offset (i.e. column address is not zero) was written into the buffers with the same offset. The exception are the commands mentioned in step 3. They always place their data in the beginning of the buffers.

75.3 EDAC

75.3.1 EDAC operation

The optional NAND Flash Memory controller EDAC automatically encode and decode data being stored in the NAND Flash memory with an error correcting code. The *edac* bit in the *Capability* register shows if the EDAC is implemented or not. If implemented, the EDAC is enabled by setting the *edacen* bit in the *Core control* register. Optional AHB error responses can be generated upon an uncorrectable error by setting the *ahberren* bit in the same register. See section 75.5 for more information.

The checksum of the code is calculated and stored for each word that is written, via the AMBA interface, into the page data part of the buffer, and automatically stored in parallel in the spare data part of the buffer. For each word of data written, one byte is stored. Note that data must be written into the EDAC part of the AHB address space. Data written to the non EDAC part of the address space does not trigger EDAC operation. See section 75.2.2 for information about the buffer memory map. Halfword and byte writes are not supported when the EDAC is enabled.

Since the number of spare bytes normally is less than what is sufficient to protect the full page data, the page data size is limited by the programmable *lpaddr* field in the *Core control* register. The *lpaddr* field defines how much user data (i.e. page data) is to be used, with the remaining part of the page data and spare data being used for checksums. Note that the spare data buffer portion is therefore made bigger than the actual spare data in the NAND Flash memory. When the NAND Flash memory is written, the data will be fetched first from the page data buffer up to *lpaddr*, and then the rest will be fetched from the spare data buffer. Errors can be detected in the underlying buffer memory when reading them as part of the write operation to the NAND Flash memory.

The reverse applies to when data is fetched from the NAND Flash memory during read. When the buffer contents are read out, via the AMBA interface, each read page data word is automatically corrected using the corresponding spare buffer checksum byte. Two levels of errors can occur, errors stemming from the underlying buffer memory protection, or errors stemming from the EDAC protecting the NAND Flash memory contents. All error flags are described in the *Core status* register.

It is possible to write or read additional dummy words to the page data buffer past the LPADDR address. This can be used to handle any surplus spare data bytes.

The memory contents are protected by means of a Bose Chaudhuri Hocquenghem (BCH) type of code. It is a Quad Error Correction/Quad Error Detection (QEC/QED) code.

The data symbols are 4-bit wide, represented as GF(2^4). The has the capability to detect and correct a single symbol error anywhere in the codeword.

75.3.2 Code

The code has the following definition:

- there are 4 bits per symbol; most significant has index 3 (to the left), least significant has index 0
- there are 17 symbols per codeword, of which 2 symbols represent the checksum;
- the code is systematic;
- the code can correct one symbol error per codeword;
- the field polynomial is

$$f(x) = x^4 + x + 1$$

- all multiplications are performed as Galois Field multiplications over the above field polynomial
- all additions/subtractions are performed as Galois Field additions (i.e. bitwise exclusive-or)

Note that only 4 of the 17 symbols are used for data, 2 symbols are used for the checksum, and the reset are not used, the code is thus shortened by 11 symbols.

75.3.3 Encoding

- a codeword is defined as 17 symbols:

  [c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, $c_{14,}$ $c_{15,}$ c_{16}]

  where c_0 to c_{14} represent information symbols and c_{15} to c_{16} represent check symbols.
- c_{15} is calculated as follows

$$c_{15} = \sum_{0}^{14}(k_i \times c_i)$$

- c_{16} is calculated as follows

$$c_{16} = \sum_{0}^{14} c_i$$

- where the constant vector k is defined as:

  k_0=0xF, k_1=0xE, .., k_{14}=0x1 (one can assume k_{15}=0x1 and k_{16}=0x1 for correction purposes)
- bit 1 of c_{15}, and bits 3 and 1 of c_{16} are inverted after encoded

75.3.4 Decoding

- the corrupt codeword is defined as 17 symbols:

  [r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, $r_{14,}$ $r_{15,}$ r_{16}]
- the corrupt codeword can also be defined as 17 uncorrupt symbols and an error:

  [c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, $c_{14,}$ $c_{15,}$ c_{16}] + [e_x]

  where the error is defined as e_x, e being the unknown magnitude and
  x being the unknown index position in the codeword
- recalculated checksum rc_0 is calculated as follows (k_i is as defined above, x being the unknown index)

$$rc_0 = \sum_{0}^{14}(k_i \times r_i) = \sum_{0}^{14}(k_i \times c_i) + (k_x \times e_x)$$

- recalculated rc_1 is calculated as follows

$$rc_1 = \sum_{0}^{14} r_i = \sum_{0}^{14} c_i + e_x$$

- syndrome s_0 is calculated as follows

$$s_0 = rc_0 + r_{15} = \sum_{0}^{14}(k_i \times r_i) + \sum_{0}^{14}(k_i \times c_i) = k_x \times e_x$$

- syndrome s_1 is calculated as follows, which gives the magnitude (not applicable to c_{15} and c_{16})

$$s_1 = rc_1 + r_{16} = \sum_{0}^{14} r_i + \sum_{0}^{14} c_i = e_x$$

- in case s_0 and s_1 are both non-zero, to located the error in range c_0 to c_{14}, multiply error magnitude e_x with each element of the constant vector defined above:

$$k_i e_x = k_i \times s_1 = k_i \times e_x \qquad i = [0,14]$$

- search the resulting vector to find the element matching syndrome s_0, the resulting index i points to the error location (applicable only to i in [0, 14])

$$k_i e_x \Leftrightarrow k_i \times e_x$$

- finally perform the correction (applicable only to i in [0, 14])

$$c_i = r_i - s_1 = r_i - e_x = (c_i - e_x) \times e_x = c_i - e_x + e_x = c_i$$

- when s_0 is zero and s_1 is non-zero, the error is located in checksum r_{15}, no correction is necessary
- when s_1 is zero and s_0 is non-zero, the error is located in checksum r_{16}, no correction is necessary
- when s_0 and s_1 are both zero, no error has been detected, no correction is necessary
- bit 1 of r_{15}, and bits 3 and 1 of r_{16} are inverted before decoded

75.4 Scan test support

The VHDL generic *scantest* enables scan test support. If the core has been implemented with scan test support and the *testen* input signal is high, the core will:

- disable the internal RAM blocks when the *scanen* signal is asserted.
- use the *testoen* signal as output enable signal.
- use the *testrst* signal as the reset signal for those registers that are asynchronously reseted.

The *testen*, *scanen*, *testrst*, and *testoen* signals are routed via the AHB slave interface.

75.5 Registers

The core is programmed through registers mapped into APB address space.

*Table 1040.*NANDFCTRL registers

APB address offset	**Register**
0x00	Core control register
0x04	Core status register
0x08	Interrupt pending register
0x0C	Capability register
0x10	Buffer 0 row address register
0x14	Buffer 0 column address register
0x18	Buffer 0 command register
0x1C	Buffer 0 control / status register
0x20*	Buffer 1 row address register
0x24*	Buffer 1 column address register
0x28*	Buffer 1 command register
0x2C*	Buffer 1 control / status register
0x30	Programmable timing register 0
0x34**	Programmable timing register 1
0x38**	Programmable timing register 2

* Only present if buffer 1 registers are implemented. Indicated by *b1en* bit in *Capability register.*

** Only present if programmable timing is implemented. Indicated by *prgt* bit in *Capability register.*

Note: The Buffer 0 and Buffer 1 registers are identical, and therefore only one set of tables describing the registers are presented below.

Table 1041. Core control register

15:13	12	11	10	9	8	7:6	5	4:2	1	0
R	DW	CMDO	R	AHBERREN	EDACEN	R	ABORT	TMODE	WP	RST

31:16
LPADDR

31:16	Last page address (LPADDR) - This field should be set to the last addressable byte in a flash memory page, not including the spare area. The core uses these bits to know when to switch to the internal buffers for the page's spare area. For example: If the flash memory devices have a page size of 4096 bytes (and an arbitrary sized spare area for each page) this field should be set to 0xFFF (= 4095). The actual number of bits used for this field depends on the size of the implemented buffers. The number of bits can be found by reading the *pbits* field of the *Capability register* and adding one. Reset value: 0xF..F
15:13	Reserved (R) - Always reads zero.
12	Data width (DW) - Sets the default data lane width. 0 = Core uses 8-bit data lanes. 1 = Core uses 16-bit data lanes. This can be overridden for individual commands by setting the *dwo* bit in the *Buffer command* register. This bit is only available if the *dw16* bit in the *Capability register* is 1. Reset value 0.
11	Command bit order (CMDO) - When this bit is set to 0 the ONFI command bytes are mapped to the core's data lane(s) as follows: Cmd bit 0 -> Data lane bit 7, Cmd bit 1 -> Data lane bit 6., and so on. When this bit is set to 1 the commands are mapped as follows: Cmd bit 0 -> Data lane bit 0, Cmd bit 1 -> Data lane bit 1, and so on. Reset value equals the value of the *cmdo* bit in the *Capability register.*
10	Reserved (R) - Always reads zero.
9	AHB error response generation (AHBERREN) - If this bit is set then the core will generate an AMBA error response if the EDAC detects an uncorrectable error, or if the fault tolerance logic for the internal buffers report and uncorrectable error, upon and AHB read. If this bit is not set when an uncorrectable error is detected, the core's output signal *error* is set high for one clock cycle instead. This bit is only present when the EDAC is implemented or when the internal buffers are implemented with either byte parity and DMR or only byte parity. The *ft* bits in the *Capability register* shows which fault tolerance that is implemented, and the *edac* bit in the *Capability register* shows if EDAC is implemented. Reset value 0.
8	EDAC enable (EDACEN) - When this bit is set the EDAC is enabled. This bit is only present when the EDAC is implemented. The *edac* bit in the *Capability register* shows if EDAC is implemented or not. Reset value 0.
7	Reserved (R) - Always reads zero.
6	EDO data output (EDO) - If programmable timing is implemented (indicated by the *prgt* field in the *Capability register*) then this bit should be set if EDO data output cycles should be used. See ONFI 2.2 Specification for more information. If programmable timing is not implemented then this bit is not present. Reset value 0.
5	Command abort (ABORT) - This bit can be set to 1 to abort a command that for some reason has put the core in a dead lock waiting for the *rb* input signal to go high. This could happen for example if a program or erase command was issued while the memory was in write protect mode. This bit is automatically cleared by the core. Reset value 0. Only available if the *rev* field in the *Capability register* > 0, otherwise always 0.
4:2	Timing mode (TMODE) - If programmable timing is not implemented (indicated by the *prgt* field in the *Capability register*) then writing this field changes the core's internal timing mode. See ONFI 2.2 Specification for more information on the different timing modes. Note that in order to change timing mode on the flash memory devices a SET FEATURES command needs to be issued. This is not done automatically when writing these bits. Timing mode 0 i always supported. Additional supported timing modes is indicated by the *tm[5:1]* bits in the *Capability register.* Should not be written while a command is in progress. Note that if this field is written with a value matching a timing mode that is not supported, then the core will operate in timing mode 0 (even though this field still changes to the invalid value). If programmable timing is implemented then this field is not present. Reset value: 0b000
1	Write protect (WP) - When this bit is set to 1 the core puts the flash memory devices in write protect mode by asserting the *wp* signal. In write protect mode, the memories won't respond to PROGRAM or ERASE commands. If the core is active when software writes this bit there is a delay before the actual write protect signal goes low. Software can use the *wp* field in the *Core status register* to see when the signal has changed value. Reset value: 0

Table 1041. Core control register

0	Software reset (RST) - If software writes this bit to 1 the core is reset, and a RESET (0xFF) command is issued to all targets on all attached flash memory devices. The only difference between a software reset and a hardware reset / power up is that the core does not reset it's *tmode* field (described above) nor the *Programmable timing registers* during software reset. The reason for this is that the flash memory devices does not change timing mode after receiving a RESET command. This bit is cleared automatically. Reset value: 0

Table 1042. Core status register (read only)

15 12	11 9	8 4	3 2	1	0
EERRFLAGS	R	STATE	R	WP	RDY

31 24	23 22	21 20	19 18	17 16
R	FTNERRFLAGS	R	FTAERRFLAGS	R

31:24	Reserved (R) - Always reads zero.
23:22	Fault tolerance error flags on NAND side (FTNERRFLAGS) - The bits in this field indicates the following errors: Bit 22: Uncorrectable error in buffer 0. Bit 23: Uncorrectable error in buffer 1. If the fault tolerance logic for the internal buffers indicates an error when the buffers are read during a NAND flash write operation the corresponding error flag in this register is set. These bits can be cleared by writing a 1 to them. The error flag for each buffer is also automatically cleared when an AHB write access occurs to that buffer (independent of address). These bits are only present when the internal buffers are implemented with either byte parity and DMR or only byte parity. The *ft* bits in the *Capability register* shows which fault tolerance that is implemented. Reset value 0.
21:20	Reserved (R) - Always reads zero.
19:18	Fault tolerance error flags on AHB side (FTAERRFLAGS) - The bits in this field indicates the following errors: Bit 18: Uncorrectable error in buffer 0. Bit 19: Uncorrectable error in buffer 1. If the fault tolerance logic for the internal buffers indicates an error when the buffers are read over AHB the corresponding error flag in this register is set. These bits can be cleared by writing a 1 to them. The error flag for each buffer is also automatically cleared when an AHB read access occurs to that buffer with an address which is at offset 0 in the page data buffer for any data lane. For example, if the core is configured with four data lanes and 4096 byte large page buffers, then an AHB read access with offsets 0x0000, 0x1000, 0x2000, and 0x3000 (with consecutive address scheme) would reset the error flag for buffer 0. These bits are only present when the internal buffers are implemented with either byte parity and DMR or only byte parity. The *ft* bits in the *Capability register* shows which fault tolerance that is implemented. When these bits get set then the core's output signal *err* is also set high for one clock cycle, if the *ahberren* bit in the *Core control* register is not set. Reset value 0.
17:16	Reserved (R) - Always reads zero.

Table 1042. Core status register (read only)

15:12	EDAC error flags (EERRFLAGS) - The different bits indicate the following errors: Bit 12: Correctable error in buffer 0. Bit 13: Correctable error in buffer 1. Bit14: Uncorrectable error in buffer 0. Bit 15: Uncorrectable error in buffer 1. If the EDAC detects an error while an internal buffer is being read over AHB the corresponding error flag in this register is set. These bits can be cleared by writing a 1 to them. The error flags for each buffer are also automatically cleared when an AHB read access occurs to that buffer with an address which is at offset 0 in the page data buffer for any data lane. For example, if the core is configured with four data lanes and 4096 byte large page buffers, then an AHB read access with offsets 0x0000, 0x1000, 0x2000, and 0x3000 (with the consecutive address scheme) would reset the error flag for buffer 0. These bits are only present when the EDAC is implemented. The *edac* bit in the *Capability register* shows if EDAC is implemented or not. When the correctable error status bits get set then the core's output signal *err* is also set high for one clock cycle. The *err* output is also set for one cycle if the error flags for uncorrectable errors get set and the *ahberren* bit in the *Core control* register is not set. Reset value 0x0.
11:9	Reserved (R) - Always reads zero.
8:4	Core state - Shows the core's internal state. Implemented for debugging purposes. 0 = reset, 1-2 = idle, 3-8 = command state, 9-11 = address state, 12-15 = data in state, 16-18 = data out state, 19-31 = unused.
3:2	Reserved (R) - Always reads zero.
1	Write protect (WP) - Shows if the flash memory devices are in write protect mode or not. 0 = Not in write protect mode. 1 = In write protect mode. Reset value: 1
0	Core ready (RDY) - After a power up / reset this bit will be cleared. Once the core is done with it's reset procedure (waiting for ready signal from all flash memory devices and issuing a RESET command) this bit is set to 1. Reset value: 0

Table 1043. Interrupt pending register

31 … 2	1	0
R	B1IRQ	B0IRQ

31:2	Reserved (R) - Always reads zero.
1	Buffer 1 interrupt (B1IRQ) - This bit is set to one when an interrupt linked to buffer 1 is generated. Software can clear this bit by writing 1 to it. Only present if the *b1en* bit in the *Capability register* indicates that buffer 1 is implemented. Reset value: 0
0	Buffer 0 interrupt (B0IRQ) - This bit is set to one when an interrupt linked to buffer 0 is generated. Software can clear this bit by writing 1 to it. Reset value: 0

Table 1044. Capability register (read only)

15	14	13	10	9	8	7	6	5	4	2	1	0
SBITS		PBITS		TM5	TM4	TM3	TM2	TM1	NLANES		NTARGS	

31	28	27	26	25	24	23	22	21	20	19	18	17	16
REV		R	PRGT	SEPB	CMDO	FT		R	B1EN	EDAC	DW16	SBITS	

Table 1044. Capability register (read only)

31:28	Revision (REV) - Indicates the revision of the core.
27	Reserved (R) - Always reads zero.
26	Programmable timing (PRGT) - 0 = Data interface timing is set at implementation time and not programmable. 1 = Data interface timing is programmable (only reset values are set at implementation time). Reflects value of VHDL generic *progtime*.
25	Separate buffers (SEPB) - Indicates if the buffers for the data lanes are shared or separate. 0 = All data lanes share buffers. 1 = All data lanes have their own buffers. When 0 a command with a data in phase should only be issued to one target and lane. A command with a data out phase can still be issued to several lanes but with the limitation that the same data will be sent to all lanes. When set to 1 several lanes can be read at the same time, and lanes can be written with individual data simultaneously. Reflects value of VHDL generic *sepbufs*.
24	Command bit order (CMDO) - Indicates the reset value for the *cmdo* bit in the *Core control register*. Reflects value of VHDL generic *cmdorder*.
23:22	Fault tolerant buffers (FT) - These bits indicate if the internal buffers in the core is implemented with fault tolerance. 0b00 = no fault tolerance, 0b01 = Byte parity DMR, 0b10 = TMR, 0b11 = Byte parity, no DMR. Reflects value of VHDL generic *ft*.
21	Reserved (R) - Always reads zero.
20	Buffer 1 enabled (B1EN) - If this bit is 1 then the core implements both buffer 0 and buffer 1, otherwise only buffer 0 is implemented. See section 75.2.2 for more information about the buffer structure. Reflects value of VHDL generic *b1en*.
19	EDAC support (EDAC) - If this bit is 1 then the core implements error detection and correction on data read from the NAND flash memory. See section 75.3 for details. Reflects value of VHDL generic *edac*.
18	16-bit memory support (DW16) - If this bit is 0 the core only support 8-bit memories. If this bit is 1 the core support both 8-bit and 16-bit memories. Only available if the *rev* field > 0, otherwise always 0. Reflects value of VHDL generic *dwidth16*.
17:14	Spare area buffer address bits (SBITS) - This field indicates how many address bits that are implemented for the buffers for the pages spare area. Add one to the value of this field to get the number of bits. The size of the buffers are $2^{\wedge(SBITS+1)}$ Reflects value of VHDL generic *sbufsize*.
13:10	Page buffer address bits (PBITS) - This field indicated how many address bits that are implemented for the page buffers. Add one to the value of this field to get the number of bits. The size (in bytes) of the buffers are $2^{\wedge(PBITS+1)}$ Reflects value of VHDL generic *pbufsize*.
9	Timing mode 5 support (TM5) - If the core does not support programmable timing (indicated by *prgt* bit described above) then this bit is 1 if the core supports timing mode 5. If programmable timing is implemented, or if the core does not support timing mode 5 then this bit is 0. Reflects value of VHDL generic *tm5*.
8	Timing mode 4 support (TM4) - If the core does not support programmable timing (indicated by *prgt* bit described above) then this bit is 1 if the core supports timing mode 4. If programmable timing is implemented, or if the core does not support timing mode 4then this bit is 0. Reflects value of VHDL generic *tm4*.
7	Timing mode 3 support (TM3) - If the core does not support programmable timing (indicated by *prgt* bit described above) then this bit is 1 if the core supports timing mode 3. If programmable timing is implemented, or if the core does not support timing mode 3then this bit is 0. Reflects value of VHDL generic *tm3*.
6	Timing mode 2 support (TM2) - If the core does not support programmable timing (indicated by *prgt* bit described above) then this bit is 1 if the core supports timing mode 2. If programmable timing is implemented, or if the core does not support timing mode 2then this bit is 0. Reflects value of VHDL generic *tm2*.
5	Timing mode 1 support (TM1) - If the core does not support programmable timing (indicated by *prgt* bit described above) then this bit is 1 if the core supports timing mode 1. If programmable timing is implemented, or if the core does not support timing mode 1 then this bit is 0. Reflects value of VHDL generic *tm1*.
4:2	Number of flash memory devices (NLANES) - This field indicates how many 8-bit/16-bit data lanes (minus one) the core can access, i.e. number of write enable signals. A write enable signal can be connected to one or more targets (i.e. one or more flash memory devices). Reflects value of VHDL generic *nlanes*.

Table 1044. Capability register (read only)

1:0	Number of targets per device (NTARGS) - This field indicates how many individual targets (minus one) the core can access, i.e. number of chip enable signals. A flash memory device can have one or more targets, each with an individual chip enable signal. Reflects value of VHDL generic *ntargets*.

Table 1045. Buffer row address register

31 24	23 0
R	ROWADDR

31:24	Reserved (R) - Always reads zero.
23:0	Row address (ROWADDR) - This field sets the three byte row address, which is used to address LUNs, blocks and pages. As described in the ONFI 2.2 specification the least significant part of the row address is the page address, the middle part block address, and the most significant part is the LUN address. Exactly how many bits that are used for each part of the address depends on the architecture of the flash memory. Software needs to write this field prior to issuing any command that has an address phase that includes the row address. The core ignores this field if the command doesn't use the row address. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.

Table 1046. Buffer column address register

31 16	15 0
SIZE	COLADDR

31:16	Command data size (SIZE) - If a command has a data out or data in phase then software needs to set this field to the size of the data that should be read / written. Software does not need to set this field for the commands SET FEATURES, GET FEATURES, READ STATUS, or READ STATUS ENHANCED their data phases are always the same size. The core also ignores this field if the command issued doesn't have a data phase, as for example BLOCK ERASE. The actual number of bits used for this field depends on the size of the implemented buffers. The number of bits can be found by reading the *pbits* field of the *Capability register* and adding one. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.
15:0	Column address (COLADDR) - This field sets the two byte column address, which is used to address into a flash memory page. See the ONFI 2.2 specification for more information about column address. Software needs to write this field for those commands that have an address phase that includes the column address, as well as for those special commands that only have a one byte address phase (SET FEATURES; GET FEATURES; READ ID, READ UNIQUE ID, and READ PARAMETER PAGE). The core uses this field as an offset into the buffers when reading / writing data. The exception is the one byte address commands mentioned above, which always store their data in the beginning of the buffers. This field is ignored by the core if the command only uses the row address. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.

Table 1047. Buffer command register

31						20	19	18	17	16	15 8	7 0
R						DWO	R	SEL	SC2	CD	CMD2	CMD1

31:21	Reserved (R) - Always reads zero.
20	Data width override (DWO) - Set this bit to 1 to override the *dw* bit in the *Core control register* for the current command. If this bit is 0, then the *dw* bit in the *Core control register* decides whether to use an 8-bit or 16-bit data lane for the current command. This bit is only available if the *dw16* bit in the *Capability register* is 1. Reset value 0.
19	Reserved (R) - Always reads zero.
18	Command select (SEL) - This bit is used to select between commands that have the same opcode in the first command cycle. This applies to the CHANGE WRITE COLUMN and COPYBACK PROGRAM commands, which both have a first command byte with the value of 0x85. If a COPYBACK PROGRAM is to be executed, this bit should be set to 0. If a CHANGE WRITE COLUMM command is to be executed, this bit should be set to 1. If the *rev* field in the *Capability register* > 0 then this bit is also used to select between a PAGE PROGRAM and a SMALL DATA MOVE (with opcode 0x80). If the SMALL DATA MOVE should be executed then this bit should be set to 1, otherwise it should be set to 0. The core ignores this bit for all other commands. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.
17	Skip second command phase (SC2) - If this bit is set and a program command (PAGE PROGRAM, COPYBACK PROGRAM, or CHANGE WRITE COLUMN) is being executed the core skips the second command phase for that command and jumps back to idle state once all data has been written. This is done in order to support the CHANGE WRITE COLUMN command, which needs to be executed in between the first and second phase of the program command. This bit is ignored by the core for all other commands. See the ONFI 2.2 specification for details on the CHANGE WRITE COLUMN command. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.
16	Common data (CD) - Sometimes, for example for the SET FEATURES command, it is desirable to send the same data on all data lanes. If this is the case, software can write the data to send in the buffer corresponding to the first data lane and then set to this bit to 1. When the core executes the command it will then send the same data on all lanes without the need for software to fill all the corresponding buffers. Needs to be set to 0 if individual data should be send to the devices. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.
15:8	Second command phase (CMD2) - If the command to execute is a two byte (two phase) command then software should write the second byte of the command to this field. The core ignores this field for commands that only have one command phase. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.
7:0	First command phase (CMD1) - Software should write this field with the first byte of the command to execute. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.

Table 1048. Buffer control / status register

31:27	26	25	24	23:16	15:8	7:4	3:2	1	0
R	INV	BSY	RUN	R	LANESEL	TARGSEL	R	IRQM	EXE

31:27	Reserved (R) - Always reads zero.
26	Invalid command (INV) - This bit is set to one if an invalid command is written to the *Buffer command register* when the *exe* bit is written. This bit is cleared automatically once a new command is started. Reset value: 0
25	Buffer busy (BSY) - Core sets this bit to 1 when a command is being executed. Once the command is done (*run* bit is cleared) software can clear this bit by writing 1 to it. While this bit is set it prevents software from writing to the buffer. This bit is write clear, but only when the *run* bit is zero. Reset value: 0
24	Command running (RUN) - Core sets this bit to 1 when a command is being executed, and clears it again automatically once the command is done. While this bit is set it prevents software from accessing the buffer. Reset value: 0
23:16	Reserved (R) - Always reads zero.
15:8	Data lane select (LANESEL) - The core uses this field to select which of the connected 8-bit/16-bit data lanes to send the command to (which write enable (WE) signals to assert). (A write enable signal can be connected to one or more flash memory devices but only one of the targets is selected at a time.) The least significant bit in this field corresponds to the first connected data lane (WE(0)) etc. The actual number of bits implemented equals the *nlanes* field in the *Capability register* plus one. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.
7:4	Target select (TARGSEL) - The core uses this field to select which targets to send the command to (which chip enable (CE) signals to assert). The least significant bit in this field corresponds to the first target (CE(0)) etc. (A chip enable signal can be connected to one or more flash memory devices, all on different 8-bit/16-bit memory lanes. One or more chip enable signals can be connected to a flash memory device, depending on how many targets the device implements.) The actual number of bits implemented equals the *ntargs* field in the *Capability register* plus one. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.
3:2	Reserved (R) - Always reads zero.
1	Interrupt mask (IRQM) - If this bit is set to 1 an interrupt will be generated when the command linked to the corresponding buffer has been executed. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero. Reset value: 1
0	Execute command (EXE) - When software writes this bit to 1 the core sends the command programmed into the corresponding *Buffer command register* to the lanes and targets selected by the *lanesel* and *targsel* field in this register. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero. Reset value: 0

Table 1049. Programmable timing register 0

15:9	8:0
tCS	tCCS

31:30	29:24	23:16
R	tRP	tRHW

31:30	Reserved (R) - Always reads zero.
29:24	RE pulse width (tRP) - Length of tRP in clock cycles, minus one. See ONFI 2.2 specification for more information. Field only present if programmable timing is implemented. Indicated by *prgt* bit in *Capability* register. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.
23:16	RE high to WE low (tRHW) - Length of tRHW in clock cycles, minus one. See ONFI 2.2 specification for more information. Field only present if programmable timing is implemented. Indicated by *prgt* bit in *Capability* register. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

Table 1049. Programmable timing register 0

15:9	CE setup time (tCS) - Length of tCS in clock cycles, minus one. See ONFI 2.2 specification for more information. Field only present if programmable timing is implemented. Indicated by *prgt* bit in *Capability* register. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.
8:0	Change Column setup time (tCCS) - Length of tCCS in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

Table 1050. Programmable timing register 1 (only present if programmable timing is implemented, which is indicated by *prgt* bit in *Capability* register)

15	14 8	7 5	4 0
R	tWHR	R	tWH

31 30 29 24	23 22	21 16
tWB	R	tWP

31:24	WE high to SR[6] low (tWB) - Length of tWB in clock cycles, minus one.ee ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.
23:22	Reserved (R) - Always reads zero.
21:16	WE pulse width (tWP) - Length of tWP in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.
15	Reserved (R) - Always reads zero.
14:8	WE high to RE low (tWHR) - Length of tWHR in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.
7:5	Reserved (R) - Always reads zero.
4:0	WE high hold time (tWH) - Length of tWH in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

Table 1051. Programmable timing register 2 (only present if programmable timing is implemented, which is indicated by *prgt* bit in *Capability* register)

15	14 8	7 6	5 0
R	tWW	R	tRR

31 24	23 16
R	tADL

31:24	Reserved (R) - Always reads zero.
23:16	ALE to data loading time (tADL) - Length of tADL in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.
15	Reserved (R) - Always reads zero.
14:8	WP transition to WE low (tWW) - Length of tWW in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.
7:6	Reserved (R) - Always reads zero.
5:0	Ready to RE low (tRR) - Length of tRR in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

75.6 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x059. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

75.7 Core instantiation

The core maps all usage of RAM on the *syncram* (or *syncramft* if *ft* generic is not set to 0) component from the technology mapping library (TECHMAP). The size of the instantiated RAM is determined by the *pbufsize*, *sbufsize*, *nlanes*, and *sepbufs* generics. Fault tolerance - byte parity DMR or TMR - can be added to the RAM by setting the *ft* generic to 1 or 2.

Note that both the *ft* and *edac* generics need to be set to 0 unless the core is used together with the fault tolerant GRLIB.

The core implements one interrupt, mapped by means of the *pirq* VHDL generic.

75.8 Configuration options

Table 1052 shows the configuration options of the core (VHDL generics).

*Table 1052.*Configuration options

Generic name	Function	Allowed range	Default
hsindex	AHB slave index	0 - NAHBSLV-1	0
haddr0	AHB slave address for BAR 0. See section 75.2.2 for explanation of address scheme.	0 - 16#FFF#	16#000#
haddr1	AHB slave address for BAR 1. See section 75.2.2 for explanation of address scheme.	0 - 16#FFF#	16#001#
hmask0	AHB slave address mask for BAR 0. If set to zero, BAR 0 is disabled. See section 75.2.2 for explanation of address scheme.	0 - 16#FFF#	16#FFF#
hmask1	AHB slave address mask for BAR 1. If set to zero, BAR 1 is disabled. See section 75.2.2 for explanation of address scheme.	0 - 16#FFF#	16#FFF#
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	APB irq number.	0 - NAHBIRQ-1	0
memtech	Memory technology used for the buffers.	0 - NTECH	inferred
sysfreq	System clock frequency in kHz.	1 - 1 000 000	50000
ntargets	Number of targets = Number of chip select signals connected to the core. A flash memory device can have one or more targets.	1 - 4	2
nlanes	Number of 8-bit/16-bit data lanes = Number of write enable signals. A write enable signal can be connected to one or more targets (i.e. one or more flash memory devices).	1 - 8	8
dwidth16	0 = Core implements 8-bit data lanes. 1 = Core implements 16-bit data lanes.	0 - 1	0
pbufsize	Size of each page buffer (in bytes). One or two page buffers are implemented for each connected flash memory device (depends on value of generic *buf1en*.). Generic needs to be set to a value equal to or greater than the flash memory device’s page area. Value also needs to be a power of two.	8, 16, 32 ... 32768	4096
sbufsize	Size of each spare area buffer (in bytes). One or two spare buffers are implemented for each connected flash memory device (depends on value of generic *buf1en*.). Generic needs to be set to a value equal to or greater than the flash memory device’s page spare area. Value also needs to be a power of two.	8, 16, 32 ... 32768	256

*Table 1052.*Configuration options

Generic name	Function	Allowed range	Default
tm1	Enable support for timing mode 1 in the case with fixed data interface timing (*progtime* generic set to 0). This generic has no effect if the *progtime* generic is set to 1.	0 - 1	0
tm2	Enable support for timing mode 2 in the case with fixed data interface timing (*progtime* generic set to 0). This generic has no effect if the *progtime* generic is set to 1.	0 - 1	0
tm3	Enable support for timing mode 3 in the case with fixed data interface timing (*progtime* generic set to 0). This generic has no effect if the *progtime* generic is set to 1.	0 - 1	0
tm4	Enable support for timing mode 4 in the case with fixed data interface timing (*progtime* generic set to 0). This generic has no effect if the *progtime* generic is set to 1.	0 - 1	0
tm5	Enable support for timing mode 5 in the case with fixed data interface timing (*progtime* generic set to 0). This generic has no effect if the *progtime* generic is set to 1.	0 - 1	0
nsync	Number of synchronization registers on R/B input. (Data input is always synchronized through one set of registers.)	0 - 3	2
ft	This generic determines if fault tolerance should be added to the buffers. 0 = no fault tolerance, 1 = Byte parity DMR, 2 = TMR. 3 = Byte parity, no DMR. Note that this generic needs to be set to 0 if the core is used together with the GPL version of GRLIB, since that version does not include any fault tolerance.	0 - 2	0
oepol	Polarity of pad output enable signal.	0 - 1	0
scantest	Enable scan test support.	0 - 1	0
edac	Enable EDAC support. See section 75.3.1 for EDAC information. Note that this generic needs to be set to 0 if the core is used together with the GPL version of GRLIB, since that version does not include any fault tolerance.	0 - 1	0
cmdorder	Sets the default way the ONFI command bytes are mapped to the core’s data lane(s). When set to 0 the commands are mapped as follows: Cmd bit 0 -> Data lane bit 7, Cmd bit 1 -> Data lane bit 6., and so on. When this bit is set to 1 the commands are mapped as follows: Cmd bit 0 -> Data lane bit 0, Cmd bit 1 -> Data lane bit 1, and so on.	0 - 1	1
sepbufs	When set to 0 all data lanes share the same internal memory buffers. When set to 1 all data lanes have their own internal memory buffers. When set to 0 a command with a data in phase should only be issued to one target and lane at the time. A command with a data out phase can still be issued to several lanes and targets but with the limitation that the same data will be sent on all lanes. When set to 1 several lanes can be read at the same time, and lanes can be written with individual data simultaneously.	0 - 1	1
progtime	When set to 0 the data interface timing for the different timing modes are set at implementation time from the *sysfreq* generic. When set to 1 the data interface timing is programmable through APB registers (reset values are set at implementation time from the *sysfreq* generic). Maximum system frequency when this generic is set to 1 is 1 GHz.	0 - 1	0

*Table 1052.*Configuration options

Generic name	Function	Allowed range	Default
buf1en	When this generic is set to 1 then both buffer 0 and buffer 1 are implemented. If set to 0, only buffer 0 is implemented. See section 75.2.2 for more information about the buffer structure.	0 - 1	1

75.9 Signal descriptions

Table 1053 shows the interface signals of the core (VHDL ports).

*Table 1053.*Signal descriptions

Signal name	Field	Type	Function	Active
rst	N/A	Input	Reset	Logical 0
clk	N/A	Input	Clock	-
apbi	*	Input	APB slave input signals	-
apbo	*	Output	APB slave output signals	-
ahbsi	*	Input	AHB slave input signals	-
ahbso	*	Output	AHB slave output signals	-
nandfi	rb	Input	Ready/Busy signal	-
	di(63:0) [2]	Input	Data input (used both for 8-bit and 16-bit lanes)	-
	dih(63:0) [2]	Input	Data input (upper byte for 16-bit lanes)	-
nandfo	ce(3:0) [3]	Output	Chip enable	Logical 0
	we(7:0) [4]	Output	Write enable	Logical 0
	do(63:0) [2]	Output	Data output (used both for 8-bit and 16-bit lanes)	-
	doh(63:0)[2]	Output	Data output (upper byte for 16-bit memories)	-
	cle	Output	Command latch enable	Logical 1
	ale	Output	Address latch enable	Logical 1
	re	Output	Read enable	Logical 0
	wp	Output	Write protect	Logical 0
	err	Output	EDAC / Buffer error on AHB access	Logical 1
	oe	Output	Output enable	[5]

* see GRLIB IP Library User's Manual

[2] The actual number of data input/output signals used depends on core configuration. Eight bits are used for each lane.

[3] The core drives one chip select signal for each target, i.e. on or more attached flash memory devices.

[4] The core drives one write enable signal for each 8-bit/16-bit data lane, i.e. one or more attached flash memory devices.

[5] The polarity of the output enable signal is implementation dependent.

75.10 Library dependencies

Table 1054 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1054.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MEMCTRL	Signals, component	Component declaration

75.11 Instantiation

This example shows how the core can be instantiated. The instantiated core has all its generics, except *hsindex*, *pindex*, *paddr*, and *pirq* at their default values. The impact of the generics can be seen in table 1052.

```
library ieee, grlib, gaisler;
use ieee.std_logic_1164.all;
use grlib.amba.all;
use gaisler.nandpkg.all;

entity nandfctrl_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;
    nandf_d : inout std_logic_vector(63 downto 0);
    nandf_dh : inout std_logic_vector(63 downto 0);
    nandf_rb : in std_ulogic;
    nandf_ce : out std_logic_vector(1 downto 0);
    nandf_we : out std_logic_vector(7 downto 0);
    nandf_re : out std_ulogic;
    nandf_cle : out std_ulogic;
    nandf_ale : out std_ulogic;
    nandf_wp : out std_ulogic;
    nandf_err : out std_ulogic
    );
end;

architecture rtl of nandfctrl_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);


  -- NANDFCTRL signals
  signal nandfo : nandfctrl_out_type;
  signal nandfi : nandfctrl_in_type;

begin

  -- AMBA Components are instantiated here
  ...

  -- NANDFCTRL core
  nand0 : nandfctrl
    generic map (hsindex => 1, pindex => 10, paddr => 10, pirq => 10)
    port map (rstn, clk, apbi, apbo(10), ahbsi, ahbso(1), nandfi, nandfo);

  -- Pads for NANDFCTRL core
  nandf_d : iopadv generic map (tech => padtech, width => 64)
                   port map (nandf_d, nandfo.do, nandfo.oe, nandfi.di);
  nandf_dh : iopadv generic map (tech => padtech, width => 64)
                   port map (nandf_dh, nandfo.doh, nandfo.oe, nandfi.dih);
  nandf_rb : inpad generic map (tech => padtech)
                   port map (nandf_rb, nandfi.rb);
  nandf_ce : outpadv generic map (tech => padtech, width => 2)
                   port map (nandf_ce, nandfo.ce);
  nandf_we : outpadv generic map (tech => padtech, width => 8)
                   port map (nandf_we, nandfo.we);
  nandf_re : outpad generic map (tech => padtech)
                   port map (nandf_re, nandfo.re);
  nandf_cle : outpad generic map (tech => padtech)
                   port map (nandf_cle, nandfo.cle);
  nandf_ale : outpad generic map (tech => padtech)
                   port map (nandf_ale, nandfo.ale);
  nandf_wp : outpad generic map (tech => padtech)
```

```
                    port map (nandf_wp, nandfo.wp);
  nandf_err : outpad generic map (tech => padtech)
                    port map (nandf_err, nandfo.err);

end;
```

76 GRPCI - 32-bit PCI Master/Target with configurable FIFOs and AHB back end

76.1 Overview

The PCI Master/Target is a bridge between the PCI bus and the AMBA AHB bus. The core is connected to the PCI bus through two interfaces, a target and master. The PCI master interface is optional and can be disabled through a VHDL generic. The AHB side of the core uses one slave interface and one master interface. Configuration registers are available through the AMBA APB bus.

The PCI and AMBA interfaces belong to two different clock domains. Synchronization is performed inside the core through FIFOs with configurable depth.

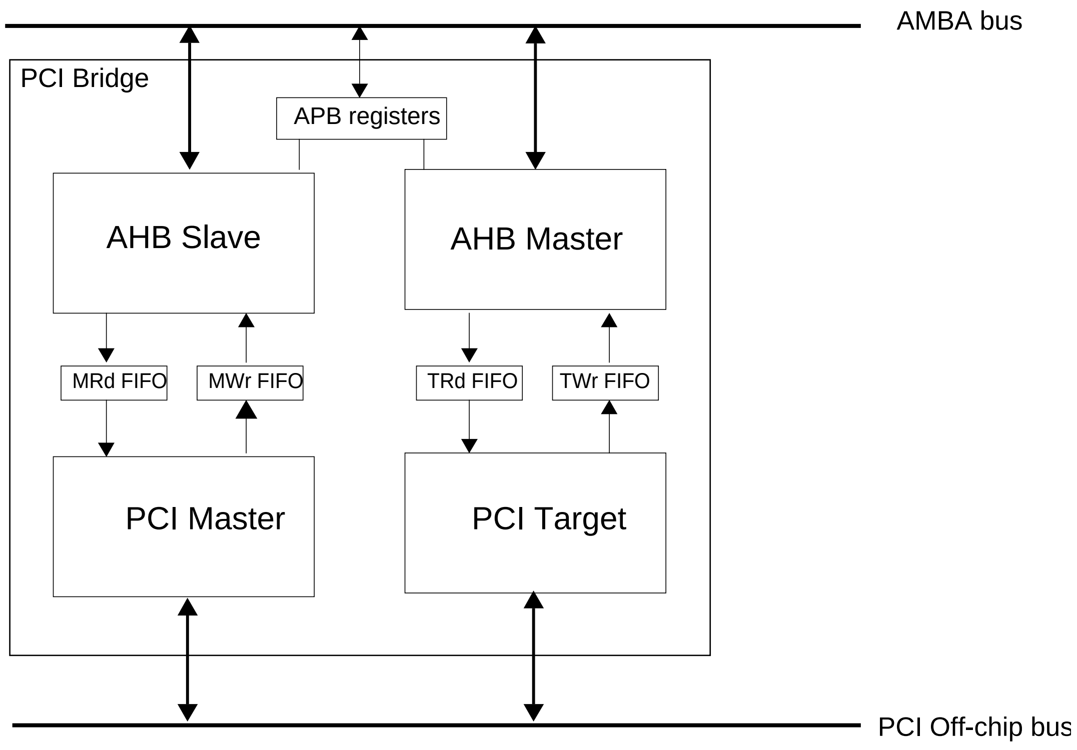


Figure 234. GRPCI master/target

A summary of the GRPCI key features:

- 32-bit PCI interface
- PCI bus master and target
- AMBA AHB/APB 2.0 back end interface
- Configurable FIFOs for both master and target operation
- Supports incremental bursts and single accesses
- Bus master capabilities:

  o Memory read, memory write

  o Memory read multiple

  o Memory read line

  o I/O read, I/O write

  o Type 0 and 1 configuration read and write

  - o Host bridging
- Target capabilities:
  - o Type 0 configuration space header
  - o Configuration read and write
  - o Parity generation (PAR), Parity error detection (PERR, SERR)
  - o 2 Memory BARs
  - o Memory read, memory write
  - o Memory read multiple
  - o Memory read line
  - o Memory write and invalidate
- Optional DMA engine add on (see PCIDMA IP core)

76.2 Operation

PCI transactions are initiated by accessing the GRPCI AHB slave. The AHB slave has one memory bank of configurable size (128 MB - 2 GB) and one 128 KB IO bank. Accesses to the memory bank are translated into PCI memory cycles while accesses to the lower half of the IO bank are translated into IO cycles. Configuration cycles are generated through accesses to the upper half of the IO bank. The AHB slave supports 8/16/32-bit single accesses and 32-bit bursts. The address translation from AHB to PCI is determined by the memory map register and the IO map register.

A connection from the PCI bus to the AHB bus is provided by the core's PCI target interface and AHB master. The PCI target is capable of handling configuration cycles, 8/16/32-bit single access memory cycles and burst memory cycles of any alignment on the PCI bus. Configuration cycles are used to access the PCI targets configuration space while the memory cycles are translated to AHB accesses. The PCI target provides two memory space Base Address Registers (BARs) of configurable size and can thus occupy two areas in the PCI memory space. Each BAR is associated with a PAGE register which determines the address translation from PCI to AHB.

For burst transactions the data is buffered internally in FIFOs with configurable size. For more information about the flow of data through the FIFOs and how it affects the operation see section 76.8.

Since PCI is little endian and LEON3 big endian GRPCI defaults to performing byte twisting on all accesses to preserve the byte ordering. See 76.7 for more information.

76.3 PCI target interface

The PCI target interface occupies two memory areas in the PCI address space and the memory mapping is determined by the BAR0 and BAR1 registers in the targets configurations space. The size of the PCI memory areas is determined by number of bits actually implemented by the BAR registers (configurable through *abits* and *dmaabits* VHDL-generics).

76.3.1 PCI commands

The GRPCI target interface handles the following PCI commands:

- **Configuration Read/Write:** Single access to the configuration space. No AHB access is performed.
- **Memory Read:** If prefetching is enabled through the *readpref* generic (it is disabled per default), the units AHB master interface fetches a cache line, otherwise a single AHB access is performed.

- **Memory Read Line:** The unit prefetches data according to the value of the cache line size register.
- **Memory Read Multiple:** The unit performs maximum prefetching. This can cause long response time, depending of the user defined FIFO depth.
- **Memory Write, Memory Write and Invalidate:** Handled similarly.

76.3.2 PCI responses

The target interface can terminate a PCI transaction with one of the following responses:

- **Retry:** This response indicates that the master should perform the same request later as the target is temporarily busy. This response is always given at least one time for read accesses, but can also occur for write accesses.
- **Disconnect with data:** Terminate the transaction and transfer data in the current data phase. This occurs if a master tries to transfer more data that fits in the FIFO or if prefetching is disabled and a Memory Read command is performed.
- **Disconnect without data:** Terminate the transaction without transfering data in the current data phase. This can occur during a PCI read burst if the PCI target is forced to insert more than 8 wait states while waiting for the AHB master to complete.
- **Target-Abort:** Indicates that the current access caused an internal error, and the target never will be able to finish it.

An AHB transaction with ‘retry’ responses is repeated by the AHB master until an ‘ok’ or ‘error’ response is received. The ‘error’ response on AHB bus will result in ‘target abort’ response for the PCI memory read cycle. In case of PCI memory write cycle, AHB access will not be finished with error response since write data is posted to the destination unit. Instead the write error (WE) bit will be set in the APB configuration/status register.

76.3.3 Bursts and byte enables

The target is capable of handling burst transactions. A PCI burst crossing 1 kB address boundary will be performed as multiple AHB bursts by the AHB master interface. The AHB master interface will insert an idle-cycle before requesting a new AHB burst to allow re-arbitration on the AHB.

A PCI master is allowed to change the byte enables on any data phase during a transaction. The GRPCI core handles this but each non 32-bit access in a PCI write burst will be translated into an AHB single access of the corresponding size by the AHB master. For PCI reads the byte enables are ignored by the PCI target and all byte lanes are driven with valid data.

76.3.4 Configuration Space

The following registers are implemented in the GRPCI configuration space. Please refer to the PCI specification for more information than is given here.

*Table 1055.*Configuration Space Header registers

Address offset	**Register**
0x00	Device ID, Vendor ID
0x04	Status, Command
0x08	Class Code & Revision ID
0x0C	BIST, Header Type, Latency Timer, Cache Line Size
0x10	BAR0
0x14	BAR1
0x3C	Max_lat, min_gnt, interrupt pin, interrupt line.

31 16	15 0
DEVICE_ID	VENDOR_ID

Figure 235. Device ID & Vendor ID register

[31:16]: Device ID (read-only). Returns value of *device_id* VHDL-generic.
[15:0]: Vendor ID (read-only). Returns value of *vendor_id* VHDL-generic.

31	30	29	28	27	26 25	24	23	22	21	20 16	15 10	9	8	7	6	5	4	3	2	1	0
DPE	‘0’	RMA	RTA	STA	‘10’	DPD	‘0’	‘0’	‘0’	RESERVED	RESERVED	‘0’	‘0’	‘0’	PER	‘0’	MIE	‘0’	BM	MS	‘0’

Figure 236. Status & Command register

[31:16]: Status Register - Writing one to a bit 31 - 16 clears the bit. Writes can not set a bit.
[31]: Detected parity Error (DPE).
[30]: Signalled System Error (SSE) - Not implemented. Always reads 0.
[29]: Received Master Abort (RMA) - Set by the PCI Master interface when its transaction is terminated with Master-Abort.
[28]: Received Target Abort (RTA) - Set by the PCI Master interface when its transaction is terminated with Target-Abort.
[27]: Signalled Target Abort (STA) - Set by the PCI Target Interface when the target terminates transaction with Target-Abort.
[26:25]: DEVSEL timing (DST) -Always reads “10” - medium DEVSEL timing.
[24]: Data Parity Error Detected (DPD).
[23]: Fast Back-to-Back Capable - The Target interface is not capable of fast back-to-back transactions. Always reads ‘0’.
[22]: UDF Supported - Not supported. Always reads ‘0’,
[21]: 66 Mhz Capable - Not supported. Always reads ‘0’.
[20:16]: Reserved. Always reads ‘00..0’.
[15:0]: Command Register - Writing one to an implemented bit sets the bit. Writing zero clears the bit.
[15:10]: Reserved - Always reads as ‘00..0’.
[9]: Fast back-to-Back Enable - Not implemented. Always reads ‘0’.
[8]: SERR# enable - Not implemented. Always reads ‘0’.
[7]: Wait cycle control - Not implemented. Always reads ‘0’.
[6]: Parity Error Response (PER) - Controls units response on parity error.
[5]: VGA Palette Snoop - Not implemented. Always reads ‘0’.

[4]: Memory Write and Invalidate Enable (MIE) - Enables the PCI Master interface to generate Memory Write and Invalidate Command.
[3]: Special Cycles - Not implemented. Always reads ‘0’.
[2]: Bus Master (BM) - Enables the Master Interface to generate PCI cycles.
[1]: Memory Space (MS) - Allows the unit to respond to Memory space accesses.
[0]: I/O Space (IOS) - The unit never responds to I/O cycles. Always reads as ‘0’.

31 8	7 0
CLASS CODE	REVISION ID

Figure 237. Class Code & revision ID

[31:8]: Class Code - Processor device class code. Set with *class_code* generic (read-only).
[7:0]: Revision ID - Set with *rev* generic (read-only).

31 24	23 16	15 8	7 0
BIST	HEADER	LTIM	CLS

Figure 238. BIST, Header Type, Latency Timer and Cache Line Size register

[31:24]: BIST - Not supported. Reads always as ‘00..0’.
[23:16]: Header Type (HEADER)- Header Type 0. Reads always as ‘00..0’.
[15:8]: Latency Timer (LTIM) - Maximum number of PCI clock cycles that Master can own the bus.
[7:0]: Cache Line Size (CLS) - System cache line size. Defines the prefetch length for ‘Memory Read’ and ‘Memory Read Line’ commands.

31 *abits*	*abits*-1 4	3	2 1	0
BASE ADDRESS	‘00..0’	‘0’	‘00’	‘0’

Figure 239. BAR0 register

[31:*abits*]: PCI Base Address - PCI Targets interface Base Address 0. The number of implemented bits depend on the VHDL-generic *abits*. Memory area of size 2^*abits* bytes at Base Address is occupied through this Base Address register. Register PAGE0 is accessed through upper half of this area. PCI memory accesses to the lower half of this area is translated to AHB accesses using PAGE0 map register.
[*abits*-1:4]: These bits are read-only and always read as ‘00..0’. This field can be used to determine devices memory requirement by writing value of all ones to this register and reading the value back. The device will return zeroes in unimplemented bits positions effectively defining memory area requested.
[3]: Prefetchable: Not supported. Always reads ‘0’.
[2:1]: Base Address Type - Mapping can be done anywhere in the 32-bit memory space. Reads always as ‘00’.
[0]: Memory Space Indicator - Register maps into Memory space. Read always as ‘0’.

PAGE0 register is mapped into upper half of the PCI address space defined by BAR0 register.

31 *dmaabits*	*dmaabits*-1 4	3	2 1	0
BASE ADDRESS	‘00..0’	‘0’	‘00’	‘0’

Figure 240. BAR1 register

[31:*dmaabits*]: PCI Base Address - PCI Targets interface Base Address 1. The number of implemented bits depends on the VHDL-generic *dmaabits*. Memory area of size 2^*dmaabits* bytes at Base Address is occupied through this Base Address register. PCI memory accesses to this memory space are translated to AHB accesses using PAGE1 map register.

[*dmaabits*-1:4]: These bits are read-only and always read as ‘00..0’. This field can be used to determine devices memory requirement by writing value of all ones to this register and reading the value back. The device will return zeroes in unimplemented bits positions effectively defining memory area requested.

[3]: Prefetchable: Not supported. Always reads as ‘0’.

[2:1]: Base Address Type - Mapping can be done anywhere in the 32-bit memory space. Reads always as ‘00’.

[0]: Memory Space Indicator - Register maps in Memory space. Read always as ‘0’.

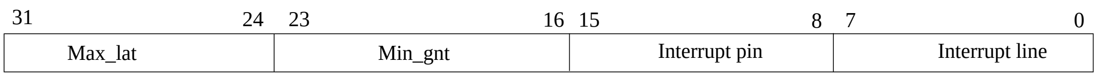


Figure 241. Max_lat, min_gnt and interrupt settings

[31-24]: Max_lat. Reads 0.

[23-16]: Min_gnt. Reads 2^(*fifodepth-3).*

[15:8]: Interrupt pin. Always reads 1, indicating that the device uses PCI interrupt pin A. If used the interrupt must be driven from outside the GRPCI core.

[7-0]: Interrupt line. Write able register used by the operating system to store information about interrupt routing.

76.3.5 The PAGE0/1 map registers

The PAGE0 and PAGE1 registers are used to translate PCI addresses to AHB addresses for BAR0 and BAR1 respectively. PAGE0 is mapped into the PCI address space defined by BAR0, while PAGE1 is an APB register.

*Table 1056.*PCI target map registers

Register	**Address**	**Address space**
PAGE0	Upper half of PCI address space defined by BAR0 register	PCI
PAGE1	APB base address + 0x10	APB

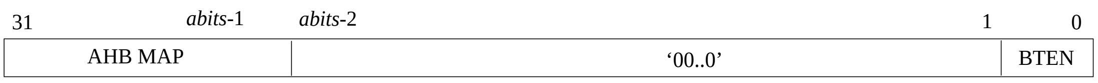


Figure 242. PAGE0 register

[31:*abits*-1]: AHB Map Address - Maps PCI accesses to PCI BAR0 address space to AHB address space. AHB address is formed by concatenating AHB MAP with LSB of the PCI address.

[*abits*-2:1]: Reserved. Reads always as ‘00..0’.

[0]: BTEN - Byte twisting enabled if ‘1’. Reset value ‘1’. May only be altered when bus mastering is disabled.

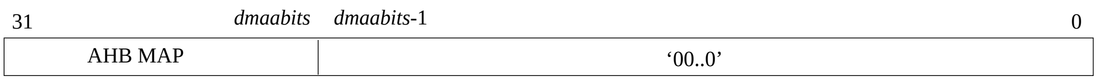


Figure 243. PAGE1 register

[31:*dmaabits*]: AHB Map Address (AHB MAP) - Maps PCI accesses to PCI BAR1 address space to AHB address space. AHB address is formed by concatenating AHB MAP with LSB of the PCI address.

[*dmaabits*-1:0]: Reserved. Reads always as ‘00..0’.

Note that it is possible to set the PAGE1 register from PCI by configuring PAGE0 to point to the APB base address and then writing to BAR0 + GRPCI_APB_OFFSET + 0x10.

76.3.6 Calculating the address of PAGE0

Since the size of BAR0 is configurable the address of the PAGE0 register is not constant between designs. It is possible to calculate the address of PAGE0 in a simple manner in order to write code that is portable between devices with differently sized BAR0 registers. This is shown below using C syntax.

```
/* Save original BAR0 (address 0x10 in the conf. space) */
pci_read_config_dword(bus,slot,function,0x10, &tmp);

/* Write 0xffffffff to BAR0 */
pci_write_config_dword(bus, slot, function, 0x10, 0xffffffff);

/* Read it back */
pci_read_config_dword(bus, slot, function, 0x10, &addr);

/* Restore to original */
pci_write_config_dword(bus, slot, function, 0x10, tmp);

/* Calculate PAGE0 offset from BAR0 (upper half of BAR0) */
addr = (~addr+1)>>1;

/* Set PAGE0 to point to start of APB */
page0 = tmp + addr;
*page0 = (unsigned int *) 0x80000000;
```

76.4 PCI master Interface

The PCI Master interface occupies 128 MB to 2 GB of AHB memory address space and 128 kB of AHB I/O address space. It handles AHB accesses to its AHB slave interface and translates them to PCI configuration (host only), memory or I/O cycles.

The mapping of the AHB slave into AHB address space is configurable through VHDL generics (see the GRLIB User’s Manual for a detailed description of the AHB address mapping). The PCI cycles performed on the PCI bus depends on the AHB access and the values in the configuration/status register.

If the PCI host signal is asserted (active low) during reset, the PCI master function will be enabled after reset. Otherwise the PCI host must enable the device to act as a master through the Bus Master bit in the command register in PCI configuration space.

The PCI master interface is capable of performing the following PCI cycles:

76.4.1 Configuration cycles

PCI configuration cycles are performed by accessing the upper 64 kB of the AHB I/O address space allocated by the AHB slave. If the bus number field in the AHB configuration register is set to 0 then type 0 cycles are generated with the mapping shown in the figure below.

31 … 16	15 … 11	10 … 8	7 … 2	1 … 0
AHB ADDRESS MSB	IDSEL	FUNC	REGISTER	‘00’

Figure 244. Mapping of AHB I/O addresses to PCI address for PCI Configuration cycle, type 0

[31:16]:	AHB Address MSB - Not used for configuration cycle address mapping.
[15:11]:	IDSEL - This field is decoded to drive PCI AD[IDSEL+10]. AD[31:11] signal lines are supposed to drive IDSEL lines during configuration cycles.
[10:8]:	Function Number (FUNC) - Selects function on multi-function device.
[7:2]:	Register Number (REGISTER) - Used to index a PCI DWORD in configuration space.
[1:0]:	Always driven to ‘00’ to generate Type 0 configuration cycle.

If the bus number field in the AHB configuration register is set to a value between 1 and 15 then type 1 cycles are generated from the AHB address as shown in the figure below. The bus number is inserted in PCI AD(20 : 16).

31 … 16	15 … 11	10 … 8	7 … 2	1 … 0
AHB ADDRESS MSB	DEVICE	FUNC	REGISTER	‘01’

Figure 245. Mapping of AHB I/O addresses to PCI address for PCI Configuration cycle, type 0

[31:16]:	AHB Address MSB - Not used for configuration cycle address mapping.
[15:11]:	Device number (DEVICE) - Which device to select.
[10:8]:	Function Number (FUNC) - Selects function on multi-function device.
[7:2]:	Register Number (REGISTER) - Used to index a PCI DWORD in configuration space.
[1:0]:	Always driven to ‘01’ to generate Type 1 configuration cycle.

When a configuration cycle does not get a response the configuration timeout (CTO) bit is set in the APB configuration/status register. This bit should be checked when scanning the bus for devices in order to detect the slots which are used.

76.4.2 I/O cycles

Single access PCI I/O cycles are supported. Accesses to the lower 64 kB of the GRPCI AHB I/O address space are translated into PCI I/O cycles. The address translation is determined by the value in the I/O map register.

76.4.3 Memory cycles

PCI memory cycles are performed by accessing the AHB memory slave. Mapping and PCI command generation are determined by the values in the AMBA configuration/status register. Burst operation is supported for PCI memory cycles.

The PCI commands generated by the master are directly dependant of the AMBA access and the value of the configuration/status register. The configuration/status register can be programmed to issue Memory Read, Memory Read Line, Memory Read Multiple, Memory Write or Memory Write and Invalidate.

If a burst AHB access is made to PCI master’s AHB slave it is translated into burst PCI memory cycles. When the PCI master interface is busy performing the transaction on the PCI bus, its AHB slave interface will not be able to accept new requests. A ‘retry’ response will be given to all accesses to its AHB slave interface. A requesting AHB master should repeat its request until ‘ok’ or ‘error’ response is given by the PCI master’s AHB slave interface. This means that all masters on the bus accessing the AHB slave interface must be round-robin arbitrated without prioritization to avoid deadlock situations.

Note that ‘retry’ responses on the PCI bus will automatically be retried by the PCI master interface until the transfer is either finished or aborted.

For burst accesses, only linear-incremental mode is supported and is directly translated from the AMBA commands.

76.4.4 PCI byte enable generation

The byte-enables on the PCI bus are translated from the AHB HSIZE control signal and the AHB address according to the table below. Note that only word, half-word and byte values of HSIZE are valid.

*Table 1057.*Byte enable generation (in PCI little endian configuration)

HSIZE	Address[1:0]	CBE[3:0]
00 (8 bit)	00	1110
00 (8 bit)	01	1101
00 (8 bit)	10	1011
00 (8 bit)	11	0111
01 (16 bit)	00	1100
01 (16 bit)	10	0011
10 (32 bit)	00	0000

76.5 PCI host operation

The GRPCI core provides a host input signal that must be asserted (active low) for PCI host operation. If this signal is asserted the bus master interface is automatically enabled (BM bit set in PCI configuration space command register).

An asserted PCI host signal also makes the PCI target respond to configuration cycles when no IDSEL signals are asserted (none of AD[31:11] are asserted). This is done for the master to be able to configure its own target.

For designs intended to be hosts or peripherals only the pci_host signal can be tied low or high internally in the design. For multi-purpose designs it should be connected to a pin. The PCI Industrial Computers Manufacturers Group (PICMG) cPCI specification uses pin C2 on connector P2 for this purpose. The pin should have pull-up resistors since peripheral slots leave it unconnected.

It is possible to enable the GRPCI core to drive the PCI reset signal if in the host slot. Normally the PCI reset is used as an input but if the *hostrst* generic is enabled it will drive the reset signal when located in the host slot. The GRPCI reset output signal should then be connected to an open drain pad with a pull up on the output.

PCI interrupts are supported as inputs for PCI hosts. See section 76.6.

76.6 PCI interrupt support

When acting as a PCI host the GRPCI core can take the four PCI interrupt lines as inputs and use them to forward an interrupt to the interrupt controller.

If any of the PCI interrupt lines are asserted and the interrupts are enabled the PCI core will drive the internal irq line specified through the *irq* generic.

There is no built in support in the PCI core to generate PCI interrupts. These should be generated by the respective IP core and drive an open-drain pad connected to the correct PCI interrupt line. Note that all single function PCI devices should drive PCI interrupt A.

76.7 Byte twisting

To maintain the correct byte order on transfers going from AHB to PCI and vice versa the GRPCI core defaults to byte twisting on all such accesses. This means that all the byte lanes are swapped as shown in the figure below.

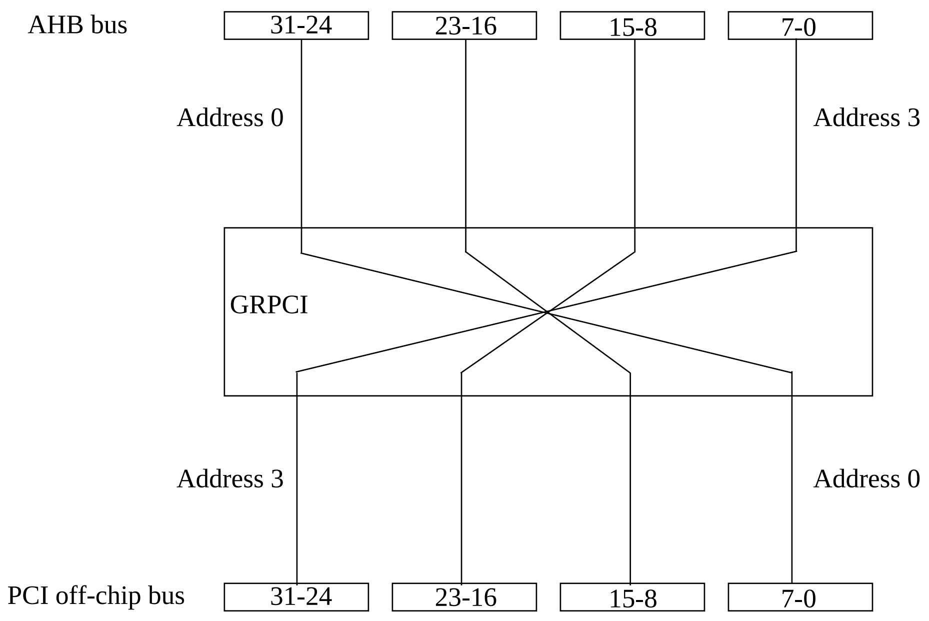


Figure 246. GRPCI byte twisting

The byte twisting can be disabled by writing 0 to bit 0 in the PAGE0 register. This should be done if the AHB bus is little endian or if no twisting is wanted for another reason. It it also possible to configure the PCI bus to be big endian through the *endian* generic (0, meaning little endian, is default).

NOTE: Only accesses that go from AHB to PCI and vice versa are twisted, i.e not accesses to configuration space or the PAGE0 register as they are little endian.

76.7.1 Byte twisting for hosts

Byte twisting should be enabled for big endian PCI hosts. Otherwise DMA transfers from PCI peripherals into the host memory will not have the correct byte ordering.

When the byte twisting is enabled byte sized PIO accesses work as expected but 16 bit and larger PIO accesses need to be twisted before being sent to the PCI core. I.e. if the value 0x12345678 is supposed to be written to a 32-bit register in a PCI peripheral the CPU will need to twist this into 0x78563412 before doing the access. Then the hardware will twist this value back and the correct 32-bit value of 0x12345678 is presented on the PCI bus. Non 8-bit descriptors must also be twisted.

76.7.2 Byte twisting for peripherals

Byte twisting must be enabled if the GRPCI core is used in a peripheral that does DMA to a little endian (or byte twisting big endian) PCI host and the correct byte order is of importance. In this case the data will keep the original byte order but non 8-bit descriptors and PIO accesses must be pre-twisted in software.

If the host is a LEON3 with GRPCI and all peripherals are big endian systems then the PCI bus could be defined big endian and byte twisting disabled for all devices (including the host).

76.8 FIFO operation

Asynchronous FIFOs of configurable size are used for all burst transactions. They are implemented with two port RAM blocks as described in 76.13.2. The PCI master and target interface have two FIFOs each, one used for read transactions and one for write transactions. Each FIFO is divided into two ping pong buffers. How the FIFOs operate during the possible transactions are described below.

76.8.1 PCI target write

When the GRPCI core is the target of a PCI write burst the PCI target begins to fill the write FIFO. After the first ping pong buffer has been filled by the PCI target it continues with filling the second buffer and the AHB master initiates an AHB burst and reads the first ping pong buffer. When the PCI target has finished filling the second buffer it is also emptied by the AHB master. If the PCI write burst is larger than the size of the FIFO the PCI target terminates the PCI transaction with a disconnect with data.

The PCI target does not accept any new transactions until the AHB master has finished emptying the complete FIFO. Any incoming access will receive a retry response.

76.8.2 PCI target read

How PCI target reads are treated depend on the PCI command used. If prefetching is disabled (*read-pref* generic = 0) then for Memory Read (0x6) commands data is fetched one word at a time. After each word the target terminates using disconnect with data. If prefetching is enabled one cache line (as defined by the cache line register) is prefetched. For bursts the Memory Read Multiple (0xC) command should be used. In this case the PCI target requests the AHB master to start filling the FIFO. The PCI targets gives retry responses until the AHB master has filled the first buffer. When the first buffer is full the PCI target responds with the data while the AHB master fills the second buffer. When the PCI target has read out the complete FIFO it terminates the read transaction with disconnect with data.

If the PCI target is forced to insert more than 8 wait states while waiting for the AHB master to to fill the second buffer it will generate a disconnect without data response.

During the period before the target responds it will give retry responses to masters trying to read an address different from the address of the already initiated read transaction.

76.8.3 PCI master write

A PCI write transaction is initiated by an access to the AHB slave of GRPCI. The AHB slave interface fills the FIFO with data. When the first buffer is full the PCI master begins the PCI write transaction while the AHB slave continues filling the second buffer. Accesses to the AHB slave that occurs when the PCI master is emptying the FIFO receives retry responses.

76.8.4 PCI master read

When the AHB slave receives a read access it requests the PCI master to issue a read transaction on the PCI bus and to fill the FIFO with the result. After the first buffer has been filled the AHB slave reads the buffer and begins to empty it to the AHB bus. When the second buffer is full it is also emptied by the AHB slave. While the AHB slave is waiting for the PCI master to fill the FIFO it gives retry responses.

76.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 1058.*AMBA registers

Address offset	Register	Note
0x00	Configuration/Status register	-
0x04	BAR0 register	Read-only access from AMBA, write/read access from PCI (see section 76.3.4).
0x08	PAGE0 register	Read-only access from AMBA, write/read access from PCI (see section 76.3.5).
0x0C	BAR1 register	Read-only access from AMBA, write/read access from PCI (see section 76.3.4).
0x10	PAGE1 register	-
0x14	IO Map register	-
0x18	Status & Command register (PCI Configuration Space Header)	Read-only access from AMBA, write/read access from PCI (see section 76.3.4).
0x1C	Interrupt enable & pending	-

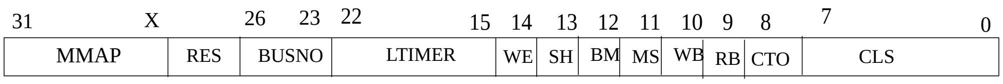


Figure 247. Configuration/Status register

[31:X]: Memory Space Map register - Defines mapping between PCI Master's AHB memory address space and PCI address space when performing PCI memory cycles. Value of this field is used as the MSB of the PCI address. LSB bits are taken from the AHB address (X = 32 - number of bits not masked with the *hmask* generic).
[X:27]: Reserved.
[26:23]: Bus number (BUSNO) - Which bus to access when generating configuration cycles.
[22:15]: Latency Timer (LTIMER) - Value of Latency Timer Register in Configuration Space Header. (Read-only)
[14]: Write Error (WE) - Target Write Error. Write access to units target interface resulted in error. (Read-only)
[13]: System Host (SH) - Set if the unit is system host. (Read-only)
[12]: Bus Master (BM) - Value of BM field in Command register in Configuration Space Header. (Read-only)
[11]: Memory Space (MS) - Value of MS field in Command register in Configuration Space Header. (Read-only)
[10]: Write Burst Command (WB) - Defines PCI command used for PCI write bursts.
'0' - 'Memory Write'
'1' - 'Memory Write and Invalidate'
[9]: Read Burst Command (RB) - Defines PCI command used for PCI read bursts.
'0' - Memory Read Multiple'
'1' - Memory Read Line'
[8]: Configuration Timeout (CTO) - Received timeout when performing Configuration cycle. (Read-only)
[7:0]: Cache Line Size (CLS) - Value of Cache Line Size register in Configuration Space Header. (Read-only)

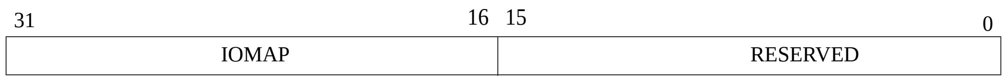


Figure 248. I/O Map register

[31:16]: I/O Map (IOMAP) - Most significant bits of PCI address when performing PCI I/O cycle. Concatenated with low bits of AHB address to from PCI address.
[15:0]: Reserved.

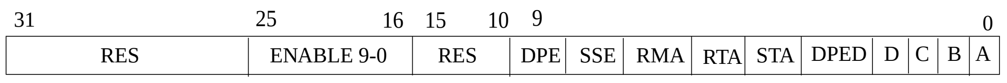


Figure 249. Interrupt enable and pending register

[31:26]:	Reserved
[25:16]:	Interrupt enable for bits 9 : 0
[15:10]:	Reserved
[9]:	DPE - Detected Parity Error Interrupt
[8]:	SSE - Signaled System Error Interrupt
[7]:	RMA - Received Master Abort Interrupt
[6]:	RTA - Received Target Abort Interrupt
[5]:	STA - Signaled Target Abort Interrupt
[4]:	DPED - Data Parity Error Detected Interrupt
[3:0]:	A:D - PCI IRQ A-D Interrupt (host only)

The error interrupts are signaled when the corresponding bit is set in the PCI configuration space status register. This bit must be cleared to clear the interrupt.

76.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x014. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

76.11 Scan support

When the SCANEN generic is 1, scan support is enabled. All asynchronous reset are then connected to AHBMI.testrst when AHBMI.testen = ‘1’. Note that the PCI clock is not multiplexed, and should be driven with the same clock as the AHB clk when AHBMI.testen = ‘1’.

76.12 Configuration options

Table 1059 shows the configuration options of the core (VHDL generics).

*Table 1059.*Configuration options

Generic	Function	Allowed range	Default
memtech	The memory technology used for the FIFO instantiation	-	0
mstndx	The AMBA master index for the target backend AHB master interface.	0 - NAHBMST-1	0
dmamst	The AMBA master index for the DMA controller, if present. This value is used by the PCI core to detect when the DMA controller accesses the AHB slave interface.	0 - NAHBMS	NAHBMST (= disabled)
readpref	Prefetch data for the ‘memory read’ command. If set, the target prefetches a cache line, otherwise the target will give a single word response.	0 -1	0
abits	Least significant implemented bit of BAR0 and PAGE0 registers. Defines PCI address space size.	16 - 28	21
dmaabits	Least significant implemented bit of BAR1 and PAGE1 registers. Defines PCI address space size.	16 - 28	26
fifodepth	Size of each FIFO is 2^fifodepth 32-bit words.	>= 3	5
device_id	PCI device ID number	0 -16#FFE#	0
vendor_id	PCI vendor ID number	0 - 16#FFF#	0
master	Disables/enables PCI master interface.	0 - 1	0
slvndx	The AHB index of the master backend AHB slave interface.	0 - NAHBSLV-1	0
apbndx	The AMBA APB index for the configuration/status APB interface	0 - NAPBMAX-1	0
paddr	APB interface base address	0 - 16#FFF#	0
pmask	APB interface address mask	0 - 16#FFF#	16#FFF#
haddr	AHB slave base address	0 - 16#FFF#	16#F00#
hmask	AHB address mask. 128 MB - 2 GB.	16#800# - 16#F80#	16#F00#
ioaddr	AHB I/O area base address	0 - 16#FFF#	0
irq	IRQ line driven by the PCI core	0 - NAHBIRQ-1	0
irqmask	Specifies which PCI interrupt lines that can cause an interrupt	0 - F	0
nsync	The number of clock registers used by each signal that crosses the clock regions.	1 - 2	2
oepol	Polarity of pad output enable signals. 0=active low, 1=active high	0 - 1	0
endian	Endianess of the PCI bus. 0 is little and 1 big.	0 - 1	0
class_code	Class code. Defaults to base class 0x0B (processor), sub class 0x40 (co-processor).	See PCI spec.	16#0B4000#
rev	Revision	0 - 16#FF#	0
scanen	Enable scan support	0 - 1	0
hostrst	Enable driving of pci_rst when host	0 - 1	0

76.13 Implementation

76.13.1 Technology mapping

GRPCI has one technology mapping generic, *memtech*, which controls how the memory cells used will be implemented. See the GRLIB Users's Manual for available settings.

76.13.2 RAM usage

The FIFOs in GRPCI are implemented with the *syncram_2p* (with separate clocks for each port) component from the technology mapping library (TECHMAP). The number of FIFOs used depends on whether the core is configured to be master/target or target only (as selected with the *master* generic). The master and target interface both use two 32 bits wide FIFOs. The depth of all FIFOs is the same and is controlled by the *fifodepth* generic.

*Table 1060.*RAM usage

Configuration	**Number of *fifodepth* x 32 bit RAM blocks**
Master/target	4
Target only	2

76.13.3 Area

The GRPCI is portable and can be implemented on most FPGA and ASIC technologies. The table below shows the approximate area usage.

*Table 1061.*Approximate area requirements

FIFO size	**VirtexII LUTs**	**StratixII LUTs**	**ASIC gates**
8	1500	1200	8000
32	1900	1300	10000

76.13.4 Timing

In order for the PCI core to function properly in a PCI system it is necessary to meet the PCI timing constraints. The PCI clock to out should not exceed 11 ns and the setup time must be below 7 ns. If you experience excessive clock to out make sure that the synthesizer has not removed the output registers. This can happen with too aggressive pipelining/retiming.

76.13.5 Pull-ups

For PCI hosts we recommend that the following signals are provided with pull-ups (if these are not available on the motherboard/rack).

*Table 1062.*PCI host pull-ups

pci_devsel
pci_frame
pci_irdy
pci_lock
pci_perr
pci_serr
pci_stop
pci_trdy
pci_gnt
pci_par
pci_rst
pci_arb_req(x)

76.14 Signal description

Table 1063 shows the interface signals of the core (VHDL ports).

*Table 1063.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RST	N/A	Input	Reset	Low
CLK	N/A	Input	AMBA system clock	-
PCICLK	N/A	Input	PCI clock	-
PCII	*1	Input	PCI input signals	-
PCIO	*1	Output	PCI output signals	-
APBI	*2	Input	APB slave input signals	-
APBO	*2	Output	APB slave output signals	-
AHBMI	*2	Input	AHB master input signals	-
AHBMO	*2	Output	AHB master output signals	-
AHBSI	*2	Input	AHB slave input signals	-
AHBSO	*2	Output	AHB slave output signals	-

*1) see PCI specification
*2) see GRLIB IP Library User’s Manual

The PCIO record contains an additional output enable signal VADEN. It is has the same value as aden at each index but they are all driven from separate registers. A directive is placed on this vector so that the registers will not be removed during synthesis. This output enable vector can be used instead of aden if output delay is an issue in the design.

For a system host, the (active low) PCII.host signal should to be connected to the PCI SYSEN signal. For a device that is not a system host, this signal should have a pull-up connection.

76.15 Library dependencies

Table 1064 shows libraries used when instantiating the core (VHDL libraries).

*Table 1064.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	PCI	Signals, component	PCI signals and component declaration

76.16 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
library techmap;
use techmap.gencomp.all;
library gaisler;
use gaisler.pci.all;

.
.
signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi : ahb_mst_in_type;
signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);


signal pcii : pci_in_type;
signal pcio : pci_out_type;

begin

pci0 : pci_mtf generic map (memtech => memtech,
hmstndx => 1,
fifodepth => log2(CFG_PCIDEPTH), device_id => CFG_PCIDID, vendor_id => CFG_PCIVID,
hslvndx => 4, pindex => 4, paddr => 4, haddr => 16#E00#,
ioaddr => 16#400#, nsync => 2)
port map (rstn, clkm, pciclk, pcii, pcio, apbi, apbo(4), ahbmi,
ahbmo(1), ahbsi, ahbso(4));

pcipads0 : pcipads generic map (padtech => padtech)-- PCI pads
    port map ( pci_rst, pci_gnt, pci_idsel, pci_lock, pci_ad, pci_cbe,
               pci_frame, pci_irdy, pci_trdy, pci_devsel, pci_stop, pci_perr,
               pci_par, pci_req, pci_serr, pci_host, pci_66, pcii, pcio );
```

76.17 Software support

Support for LEON3 systems acting as PCI hosts using the GRPCI is available in Linux 2.6, RTEMS and VxWorks. In the GRLIB IP library there is a simple Bare C (BCC) example of how configure and use PCI in GRLIB/software/leon3/pcitest.c. See also 76.18, Appendix A for BCC source code examples.

The debug monitor GRMON supports PCI bus scanning and configuration for GRPCI hosts.

76.18 Appendix A - Software examples

Examples of PCI configurations functions.

pci_read_config_dword - generate a configuration read (type 0) cycle

pci_write_config_dword - generate a configuration write (type 0) cycle

pci_mem_enable - enable the memory space of a device

pci_master_enable - enable bus master for a device

```
/* Upper half of IO area */
#define PCI_CONF 0xfff10000

typedef struct {
volatile unsigned int cfg_stat;
volatile unsigned int bar0;
volatile unsigned int page0;
volatile unsigned int bar1;
volatile unsigned int page1;
volatile unsigned int iomap;
volatile unsigned int stat_cmd;
} LEON3_GRPCI_Regs_Map;

int
pci_read_config_dword(unsigned char bus, unsigned char slot, unsigned char function, unsigned
char offset, unsigned int *val) {

    volatile unsigned int *pci_conf;

    if (offset & 3) return -1;

    if (slot >= 21) {
        *val = 0xffffffff;
        return 0;
    }

    pci_conf = PCI_CONF + ((slot<<11) | (function<<8) | offset);

    *val =  *pci_conf;

    if (pcic->cfg_stat & 0x100) {
        *val = 0xffffffff;
    }

    return 0;
}

int
pci_write_config_dword(unsigned char bus, unsigned char slot, unsigned char function,
unsigned char offset, unsigned int val) {

    volatile unsigned int *pci_conf;

    if (offset & 3) return -1;

    pci_conf = PCI_CONF + ((slot<<11) | (function<<8) | (offset & ~3));

    *pci_conf = val;

    return 0;
}

void pci_mem_enable(unsigned char bus, unsigned char slot, unsigned char function) {
    unsigned int data;

    pci_read_config_dword(0, slot, function, 0x04, &data);
    pci_write_config_dword(0, slot, function, 0x04, data | 0x2);
}

void pci_master_enable(unsigned char bus, unsigned char slot, unsigned char function) {
    unsigned int data;

    pci_read_config_dword(0, slot, function, 0x04, &data);
    pci_write_config_dword(0, slot, function, 0x04, data | 0x4);

}
```

76.19 Appendix B - Troubleshooting

76.19.1 GRPCI does not operate correctly.

Make sure that the PCI timing constraints are met (see 76.13.4) and that all necessary pull-ups are available. The generic nsync should be set to 2 for reliable operation.

76.19.2 It is impossible to meet the PCI clock to out constraint due to too many levels of logic.

The PCI output registers have been removed. This can happen when the tools use pipelining and retiming to aggressively.

76.19.3 I write 0x12345678 but get 0x78563412, what is the matter?

Please read about byte twisting in section 76.7.

76.19.4 The PCI target responds by asserting stop when I try to read data.

The PCI target gives a ‘retry’ response (stop but not trdy asserted on initial data phase). This is normal since the PCI target needs to request the data from the AHB master before it can deliver it on the PCI bus.

If a Memory Read Multiple command is issued the PCI target will give retry responses until half the FIFO has been prefetched. Thus the initial latency is dependant on the FIFO size and the PCI command. See section 76.8.

76.19.5 Configuration space accesses do not work.

In order to initiate any access the device must have the Bus Master bit set in the PCI configuration space command register. This bit is automatically set if the GRPCI host input signal is asserted (low). See section 76.5 (PCI host operation).

77 GRPCI2 - 32-bit PCI(Initiator/Target) / AHB(Master/Slave) bridge

77.1 Overview

The GRPCI2 core is a bridge between the PCI bus and the AMBA AHB bus. The core is capable of connecting to the PCI bus via both a target and a initiator/master interface. The connection to the AMBA bus is an AHB master interface for the PCI target functionality and an AHB slave interface for the PCI initiator functionality. The core also contains a DMA controller. For the DMA functionality, the core uses the PCI initiator to connect to the PCI bus and an AHB master to connect to the AMBA bus. Configuration registers in the core are accessible via a AMBA APB slave interface.

The PCI and AMBA interfaces belong to two different clock domains. Synchronization is performed inside the core through FIFOs with configurable depth.

The PCI interface is compliant with the 2.3 PCI Local Bus Specification.

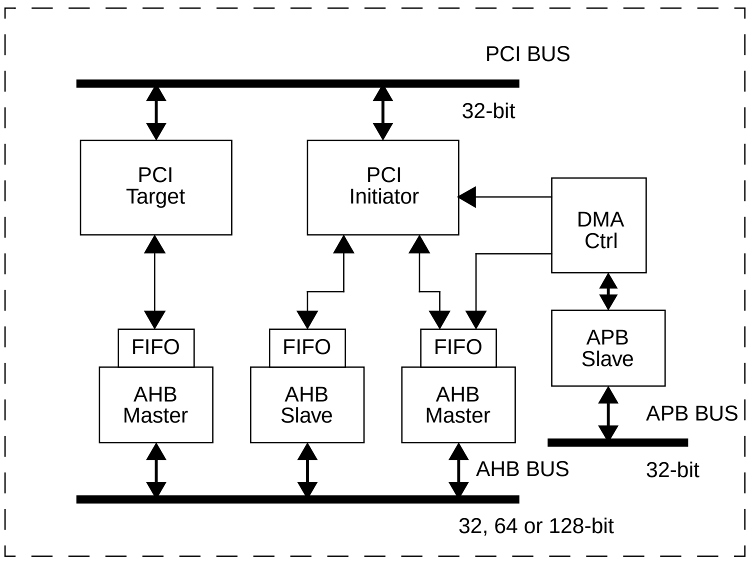


Figure 250. Block diagram

77.2 Configuration

The core has configuration registers located both in PCI Configuration Space (Compliant with the 2.3 PCI Local Bus Specification) and via an AMBA APB slave interface (for core function control and DMA control). This section defines which configuration options that are implemented in the PCI configuration space together with a list of capabilities implemented in the core. For a more detailed description of the core registers and DMA controller registers, see section Registers.

77.2.1 Configuration & Capabilities

Which of the core capabilities that are implemented is configured through VHDL generics at core instantiation. The implemented configuration can be determined by reading the Status & Capability register accessible via the APB slave interface.

- The PCI vendor and device ID is set with the VHDL generic *vendorid* and *deciceid*.
- The PCI class code and revision ID is set with the VHDL generic *classcode* and *revisionid*.
- 32-bit PCI initiator interface is implemented when the VHDL generic *master* is enabled.
- 32-bit PCI target interface is implemented when the VHDL generic *target* is enabled.

- DMA controller is implemented when the VHDL generic *dma* is enabled.
- The depth and number of FIFOs is configured with the VHDL generic *fifo_depth* and *fifo_count*.
- PCI BARs. The default size and number of BARs implemented is configured with the VHDL generic *bar0* to *bar5*.
- User defined register in Extended PCI Configuration Space can be enabled with the VHDL generic *ext_cap_pointer.*
- Device interrupt generation is enabled with the VHDL generic *deviceirq*.
- PCI interrupt sampling and forwarding is enabled with the VHDL generic *hostirq*.
- Support for two PCI functions is enabled with the VHDL generic multifunc.

77.2.2 PCI Configuration Space

The core implements the following registers in the PCI Configuration Space Header. For more detailed information regarding each field in these registers please refer to the PCI Local Bus Specification.

*Table 1065.*GRPCI2: Implemented register in the PCI Configuration Space Header

PCI address offset	**Register**
0x00	Device ID, Vendor ID
0x04	Status, Command
0x08	Class Code, Revision ID
0x0C	BIST, Header Type, Latency Timer, Cache Line Size
0x10 - 0x24	Base Address Registers
0x34	Capabilities Pointer
0x3C	Max_Lat, Min_Gnt, Interrupt Pin, Interrupt Line

Table 1066. GRPCI2 Device ID and Vendor ID register (address offset 0x00)

31 16	15 0
Device ID	Vendor ID

31 : 16	Device ID, Set by the *deviceid* VHDL generic.
15 : 0	Vendor ID, Set by the *vendorid* VHDL generic.

Table 1067. GRPCI2 Status and Command register (address offset 0x04)

31						24	23	22	21	20	19	18 11	10	9	8	7	6	5	4	3	2	1	0
DPE	SSE	RMA	RTA	STA	DEV SEL timing	MDPE	FBBC	RES	66 MHz	CL	IS	RESERVED	ID	Not Imp	SE	RES	PER	Not Imp	MWI	Not Imp	BM	MS	Not Imp

31	Detected Parity Error
30	Signaled System Error
29	Received Master Abort
28	Received Target Abort
27	Signaled Target Abort
26: 25	DEVSEL timing, Returns “01“ indicating medium
24	Master Data Parity Error

Table 1067. GRPCI2 Status and Command register (address offset 0x04)

23	Fast Back-to-Back Capable, Returns zero. (Read only)
22	RESERVED
21	66 MHz Capable (Read only) NOTE: In this core this bit has been defined as the status of the M66EN signal rather than the capability of the core. For a 33 MHz design, this signal should be connected to ground and this status bit will have the correct value of ‘0’. For a 66 MHz design, this signal is pulled-up by the backplane and this status bit will have the correct value of ‘1’. For a 66 MHz capable design inserted in a 33 MHz system, this bit will then unfortunate only indicate a 33 MHz capable device.
20	Capabilities List, Returns one (Read only)
19	Interrupt Status (Read only)
18: 11	RESERVED
10	Interrupt Disable
9	NOT IMPLEMENTED, Returns zero.
8	SERR# Enable
7	NOT IMPLEMENTED, Returns zero.
6	Parity Error Response
5	NOT IMPLEMENTED, Returns zero.
4	Memory Write and Invalidate Enable
3	NOT IMPLEMENTED, Returns zero.
2	Bus Master
1	Memory Space
0	NOT IMPLEMENTED, Returns zero.

Table 1068. GRPCI2 Class Code and Revision ID register (address offset 0x08)

31 8	7 0
Class Code	Revision ID

31 : 8	Class Code, Set by the *classcode* VHDL generic.
7 : 0	Revision ID, Set by the *revisionid* VHDL generic.

Table 1069. GRPCI2 BIST, Header Type, Latency Timer, and Cache Line Size register (address offset 0x0C)

31 24	23 16	15 8	7 0
BIST	Header Type	Latency Timer	Cache Line Size

31 : 24	NOT IMPLEMENTED, Returns zeros
23 : 16	Header Type, Returns 00
15 : 8	Latency Timer, All bits are writable.
7 : 0	NOT IMPLEMENTED, Returns zero.

Table 1070. GRPCI2 Base Address Registers (address offset 0x10 - 0x24)

31 4	3	2 1	0
Base Address	PF	Type	MS

31 : 4	Base Address. The size of the BAR is determine by how many of the bits (starting from bit 31) are implemented. Bits not implemented returns zero.
3	Prefetchable, Returns zero indicating non-prefetchable.

Table 1070. GRPCI2 Base Address Registers (address offset 0x10 - 0x24)

2 : 1	Type, Returns zero.
0	Memory Space Indicator

Table 1071. GRPCI2 Capabilities Pointer Register (address offset 0x34)

31 - 8	7 - 0
RESERVED	Capabilities Pointer

31 : 8	RESERVED
7 : 0	Capabilities Pointer. Indicates the first item in the list of capabilities of the Extended PCI Configuration Space. This offset is set with the VHDL generic *cap_pointer.*

Table 1072. GRPCI2 Max_Lat, Min_Gnt, Interrupt Pin,and Interrupt Line register (address offset 0x3C)

31 - 24	23 - 16	15 - 8	7 - 0
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line

31 : 24	NOT IMPLEMENTED, Returns zero
23 : 16	NOT IMPLEMENTED, Returns zero
15 : 8	Interrupt Pin, Indicates INTA# when VHDL generic *deviceirq* is 1, otherwise zero is returned (Read only)
7 : 0	Interrupt Line

77.2.3 Extended PCI Configuration Space

This section describes the first item in the list of capabilities implemented in the Extended PCI Configuration Space. This capability is core specific and contains the PCI to AMBA address mapping and the option to change endianess of the PCI bus.

When user defined capability list items are implemented, the next pointer defines the offset of this list item. The AMBA address mapping for these registers can be accessed in the core specific item (first list item). The registers implemented in this AMBA address range must be compliant to the capability list items defined in the 2.3 PCI Local Bus Specification.

*Table 1073.*GRPCI2: Internal capabilities of the Extended PCI Configuration Space

PCI address offset (with the Capabilities pointer as base)	**Register**
0x00	Length, Next Pointer, ID
0x04 - 0x18	PCI BAR to AHB address mapping
0x1C	Extended PCI Configuration Space to AHB address mapping
0x20	AHB IO base address and PCI bus config (endianess switch)
0x24 - 0x38	PCI BAR size and prefetch
0x3C	AHB master prefetch burst limit

Table 1074. GRPCI2 Length, Next pointer and ID (address offset 0x00)

31 : 24	23 : 16	15 : 8	7 : 0
RESERVED	Length	Next Pointer	Capability ID

31 : 24	RESERVED.
23 : 16	Length, Returns 0x40. (Read only)
15 : 8	Pointer to the next item in the list of capabilites.This offset is set with the VHDL generic *ext_cap_pointer.* (Read only)
7 : 0	Capability ID, Returns 0x09 indicating Vendor Specific. (Read only)

Table 1075. GRPCI2 PCI BAR to AHB address mapping register (address offset 0x04 - 0x18)

31 : 0
PCI BAR to AHB address mapping

31 : 0	32-bit mapping register for each PCI BAR. Translate an access to a PCI BAR to a AHB base address. The size of the BAR determine how many bits (starting form bit 31) are implemented. Bits non implemented returns zero.

Table 1076. GRPCI2 Extended PCI Configuration Space to AHB address mapping register (address offset 0x1C)

31 : 8	7 : 0
Extended PCI Configuration Space to AHB address mapping	RESERVED

31 : 8	Translates an access to the Extended PCI Configuration Space (excluding the address range for the internal register located in this configuration space) to a AHB address.
7 : 0	RESERVED

Table 1077. GRPCI2 AHB IO base address and PCI bus config (endianess register) (address offset 0x20)

31 : 20	19 : 2	1	0
AHB IO base address	RESERVED	DISEN	Endian

31 : 8	Base address of the AHB IO area. (Read only, not replicated for each PCI function)
19 : 2	RESERVED
1	Target access discard time out enable. When set to ‘1’, the target will discard a pending access if no retry of the access is detected during 2**15 PCI clock cycles. (Not replicated for each PCI function)
0	PCI bus endianess switch. 1: defines the PCI bus to be little-endian, 0: defines the PCI bus to be big-endian. Reset value is set be the conv_endian VHDL generic. (Not replicated for each PCI function)

Table 1078. GRPCI2 PCI BAR size and prefetch register (address offset 0x24 - 0x38)

31 : 4	3	2 : 1	0
PCI BAR size mask	Pre	RESERVED	Type

31 : 4	A size mask register for each PCI BAR. When bit[n] is set to ‘1’ bit[n] in the PCI BAR register is implemented and can return a non-zero value. All bits from the lowest bit set to ‘1’ up to bit 31 need to be set to ‘1’. When bit 31 is ‘0’, this PCI BAR is disabled. The number of implemented bits in this field depends in the VHDL generic barminsize. The minimal size of the BAR is not allowed to be smaller than the internal FIFO.

Table 1078. GRPCI2 PCI BAR size and prefetch register (address offset 0x24 - 0x38)

3	Prefetch bit in PCI BAR register
2 : 1	RESERVED
0	BAR type. 0 = Memory BAR, 1 = IO BAR

Table 1079. GRPCI2 AHB master prefetch burst limit (address offset 0x3C)

31	30 - 16	15 - 0
SRF	RESERVED	Burst length

31	Store Read FIFO. When set to 1, the prefetched FIFO will be stored until the next PCI access when the PCI target terminates the access with disconnect without data.
30 : 16	RESERVED
15 : 0	Maximum number of beats - 1 in the burst. (Maximin value is 0xFFFF => 0x10000 beats => 65kB address)

77.2.4 Multi-Function

The core supports up to two PCI functions starting from function 0. Each function has its own PCI configuration space located at offset 0x0 for function 0 and offset 0x100 for function 1. Some registers in the Extended PCI configuration space is shared between all functions. All functions also share the same Vendor ID.

77.3 Operation

77.3.1 Access support

The core supports both single and burst accesses on the AMBA AHB bus and on the PCI bus. For more information on which PCI commands that are supported, see the PCI target section and for burst limitations see the Burst section.

77.3.2 FIFOs

The core has separate FIFOs for each data path: PCI target read, PCI target write, PCI master read, PCI master write, DMA AHB-to-PCI, and DMA PCI-to-AHB. The number and depth of the FIFOs for each data path is configurable by VHDL generics.

77.3.3 Byte enables and byte twisting (endianess)

The core has the capability of converting endianess between the two busses. This means that all byte lanes can be swapped by the core as shown in figure below.

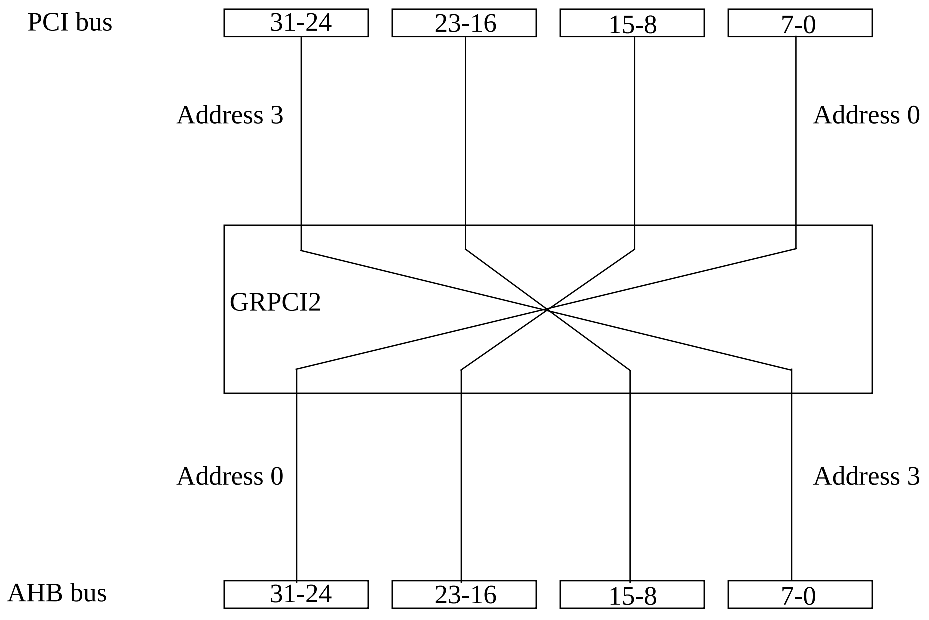


Figure 251. GRPCI2 byte twisting

Table 1080 defines the supported AHB address/size and PCI byte enable combinations.

*Table 1080.*AHB address/size <=> PCI byte enable combinations.

AHB HSIZE	**AHB ADDRESS[1:0]**	**Little-endian CBE[3:0]**	**Big-endian CBE[3:0]**
00 (8-bit)	00	1110	0111
00 (8-bit)	01	1101	1011
00 (8-bit)	10	1011	1101
00 (8-bit)	11	0111	1110
01 (16-bit)	00	1100	0011
01 (16-bit)	10	0011	1100
10 (32-bit)	00	0000	0000

As the AHB bus in GRLIB is defined as big-endian, the core is able to define the PCI bus as little-endian (as defined by the PCI Local Bus Specification) with endianess conversion or define the PCI bus as big-endian without endianess conversion.

The endianess of the PCI bus is configured via the core specific Extended PCI Configuration Space. The default value is set by a VHDL generic *conv_endian*.

77.3.4 PCI configuration cycles

Accesses to PCI Configuration Space are not altered by the endianess settings. The PCI Configuration Space is always defined as little-endian (as specified in the PCI Local Bus Specification). This means that the PCI target does not change the byte order even if the endianess conversion is enabled and the PCI master always converts PCI Configuration Space accesses to little-endian.

Data stored in a register in the PCI Configuration Space as 0x12345678 (bit[31:0]) is transferred to the AHB bus as 0x78563412 (bit[31:0]). This means that non-8-bit accesses to the PCI Configuration Space must be converted in software to get the correct byte order.

77.3.5 Memory and I/O accesses

Memory and I/O accesses are always affected by the endianess conversion setting. The core should define the PCI bus as little-endian in the following scenarios: When the core is the PCI host and little-endian peripherals issues DMA transfers to host memory. When the core is a peripheral device and issues DMA transfers to a little-endian PCI host.

77.3.6 Bursts

PCI bus: The PCI target terminates a burst when no FIFO is available (the AMBA AHB master is not able to fill or empty the FIFO fast enough) or for reads when the burst reached the length specified by the “AHB master prefetch burst limit” register. This register defines a boundary which a burst can not cross i.e. when set to 0x400 beats (address boundary at 4kB) the core only prefetch data up to this boundary and then terminates the burst with a disconnect.

The PCI master stops the burst when the latency timer times out (see the PCI Local Bus Specification for information on the latency timer) or for reads when the burst reaches the limit defined by “PCI master prefetch burst limit” register (if AHB master performing the access is unmasked). If the master is masked in this register, the limit is set to 1kB. The PCI master do not prefetch data across this address boundary.

AHB bus: As long as FIFOs are available for writes and data in a FIFO is available for read, the AHB slave do not limit the burst length. The burst length for the AHB master is limited by the FIFO depth. The AHB master only burst up to the FIFO boundary. Only linear-incremental burst mode is supported.

DMA: DMA accesses are not affected by the “AHB master prefetch burst limit“ register or the “PCI master prefetch burst limit“ register.

All FIFOs are filled starting at the same word offset as the bus access (i.e. with a FIFO of depth 8 words and the start address of a burst is 0x4, the first data word is stored in the second FIFO entry and only 7 words can be stored in this FIFO).

77.3.7 Host operation

The core provides a system host input signal that must be asserted (active low) for PCI system host operations. The status of this signal is available in the Status & Capability register accessible via the APB slave interface. The device is only allowed to generate PCI configuration cycles when this signal is asserted (device is the system host).

For designs intended to be host or peripherals only the PCI system host signal can be tied low or high internally in the design. For multi-purpose designs it should be connected to a pin. The PCI Industrial Computer Manufacturers Group (PCIMG) cPCI specification uses pin C2 on connector P2 for this purpose. The pin should have a pull-up resistor since peripheral slots leave it unconnected.

An asserted PCI system host signal makes the PCI target respond to configuration cycles when no IDSEL signal is asserted (none of AD[31:11] are asserted). This is done for the PCI master to be able to configure its own PCI target.

77.4 PCI Initiator interface

The PCI master interface is accessible via the AMBA AHB slave interface. The AHB slave interface occupies 1MB to 2GB of the AHB memory address space and 128kB to 256kB of AHB I/O address space. An access to the AHB memory address area is translated to a PCI memory cycle. An access to the first 64kB of the AHB IO area is translated to a PCI I/O cycle. The next 64kB are translated to PCI configuration cycles. When the PCI trace buffer is implemented, it is accessible via the last 128kB of the AHB I/O area.

77.4.1 Memory cycles

A single read access to the AHB memory area is translated into a PCI memory read access, while a burst read translates into a PCI memory read multiple access. A write to this memory area is translated into a PCI write access.

The address translation is determined by AHB master to PCI address mapping registers accessible via the APB slave interface. Each AHB master on the AMBA AHB bus has its own mapping register. These registers contain the MSBs of the PCI address.

When the PCI master is busy performing a transaction on the PCI bus and not able to accept new requests, the AHB slave interface will respond with an AMBA RETRY response. This occurs on reads when the PCI master is fetching the requested data to fill the read FIFO or on writes when no write FIFO is available. This means that all masters on the AMBA bus accessing the AHB slave interface must be round-robin arbitrated without prioritization to avoid deadlock situations.

77.4.2 I/O cycles

Accesses to the lowest 64kB of the AHB I/O address area are translated into PCI I/O cycles. The address translation is determined by the "AHB to PCI mapping register for PCI I/O". This register sets the 16 MSB of the PCI address. The "AHB to PCI mapping register for PCI I/O" is accessible via the APB slave interface. When the "IB" (PCI IO burst) bit in the Control register (accessible via the APB slave interface) is cleared, the PCI master does not perform burst I/O accesses.

77.4.3 Configuration cycles

Accesses to the second 64kB address block (address offset range 64kB to 128kB) of the AHB I/O address area is translated into PCI configuration cycles. The AHB address is translated into PCI configuration address different for type 0 and type 1 PCI configuration cycles. When the "bus number" field in the control register (accessible via the APB slave interface) is zero, type 0 PCI configuration cycles is issued. When the "bus number" field is non-zero, type 1 PCI configuration cycles are issued to the PCI bus determine by this field. The AHB I/O address mapping to PCI configuration address for type 0 and type 1 PCI configuration cycles is defined in table 1081 and table 1082.

Only the system host is allowed to generate PCI configuration cycles. The core provides a system host input signal that must be asserted (active low) for PCI system host operations. The status of this signal is available in the Status & Capability register accessible via the APB slave interface.When the "CB" (PCI Configuration burst) bit in the Control register (accessible via the APB slave interface) is cleared, the PCI master does not perform burst configuration accesses.

Table 1081. GRPCI2 Mapping of AHB I/O address to PCI configuration cycle, type 0

31 16	15 11	10 8	7 2	1 0
AHB ADDRESS MSB	IDSEL	FUNC	REGISTER	BYTE

31: 16	AHB address MSBs: Not used for PCI configuration cycle address mapping.
15: 11	IDSEL: This field is decoded to drive PCI AD[IDSEL+10]. Each of the signals AD[31:11] are suppose to be connected (by the PCI back plane) to one corresponding IDSEL line.
10: 8	FUNC: Selects function on a multi-function device.
7: 2	REGISTER: Used to index a PCI DWORD in configuration space.
1: 0	BYTE: Used to set the CBE correctly for non PCI DWORD accesses.

Table 1082. GRPCI2 Mapping of AHB I/O address to PCI configuration cycle, type 1

31 16	15 11	10 8	7 2	1 0
AHB ADDRESS MSB	DEVICE	FUNC	REGISTER	BYTE

31: 16	AHB address MSBs: Not used for PCI configuration cycle address mapping.
15: 11	DEVICE: Selects which device on the bus to access.
10: 8	FUNC: Selects function on a multi-function device.
7: 2	REGISTER: Used to index a PCI DWORD in configuration space.
1: 0	BYTE: Used to set the CBE correctly for non PCI DWORD accesses.

77.4.4 Error handling

When a read access issued by the PCI master is terminated with target-abort or master-abort, the AHB slave generates an AMBA ERROR response when the “ER” bit in the control register is set. When the “EI“ bit in the control register is set, an AMBA interrupt is generated for the error. The interrupt status field in the control register indicates the cause of the error.

77.5 PCI Target interface

The PCI Target occupies memory areas in the PCI address space corresponding to the BAR registers in the PCI Configuration Space. Each BAR register (BAR0 to BAR5) defines the address allocation in the PCI address space. The size of each BAR is set by the “BAR size and prefetch” registers accessible via the core specific Extended PCI Configuration Space. The size of a BAR can be determined by checking the number of implemented bits in the BAR register. Non-implemented bits returns zero and are read only. The size of the BAR is not allowed to be smaller then the size of the internal FIFO.

77.5.1 Supported PCI commands

These are the PCI commands that are supported by the PCI target.

- **PCI Configuration Read/Write:** Burst and single access to the PCI Configuration Space. These accesses are not transferred to the AMBA AHB bus except for the access of the user defined capability list item in the Extended PCI Configuration Space.
- **Memory Read:** A read command to the PCI memory BAR is transferred to a single read access on the AMBA AHB bus.
- **Memory Read Multiple, Memory Read Line:** A read multiple command to the PCI memory BAR is transferred to a burst access on the AMBA AHB bus. This burst access prefetch data to fill the maximum amount of data that can be stored in the FIFO.
- **Memory Write, Memory Write and Invalidate:** These command are handled similarly and are transferred to the AMBA AHB bus as a single or burst access depending on the length of the PCI access (a single or burst access).
- **IO Read:** A read command to the PCI IO BAR is transferred to a single read access on the AMBA AHB bus.
- **IO Write:** A write command to the PCI IO BAR is transferred to the AMBA AHB bus as a single access.

77.5.2 Implemented PCI responses

The PCI target can terminate a PCI access with the following responses.

- **Retry:** This response indicates the PCI target is busy by either fetching data for the AMBA AHB bus on a PCI read or emptying the write FIFO for a PCI write. A new PCI read access will always be terminated with a retry at least one time before the PCI target is ready to deliver data.
- **Disconnect with data:** Terminate the transaction and transfer data in the current data phase. This occurs when the PCI master request more data and the next FIFO is not yet available or for a PCI burst access with the Memory Read command.
- **Disconnect without data:** Terminate the transaction without transferring data in the current data phase. This occurs if the CBE change within a PCI burst write.
- **Target Abort:** Indicates that the current access caused an internal error and the target is unable to finish the access. This occurs when the core receives a AMBA AHB error during a read operation.

77.5.3 PCI to AHB translation

Each PCI BAR has translation register (mapping register) to translate the PCI access to a AMBA AHB address area. These mapping registers are accessible via the core specific Extended PCI Configuration Space. The number of implemented bits in these registers correspond to the size of (and number of implemented bits in) the BARs registers.

77.5.4 PCI system host signal

When the PCI system host signal is asserted the PCI target responds to configuration cycles when no IDSEL signal is asserted (none of AD[31:11] are asserted). This is done for the PCI master, in a system host position, to be able to configure its own PCI target.

77.5.5 Error handling

The PCI target terminates the access with target-abort when the PCI target requests data from the AHB bus which results in an error response on the AHB bus. Because the writes to the PCI target is posted, no error is reported on write AHB errors.

When a PCI master is terminated with a retry response it is mandatory for that master to retry this access until the access is completed or terminated with target-abort. If the master never retries the access, the PCI target interface would be locked on this access and never accept any new access. To recover from this situation, the PCI target has a option to discard an access if it is not retried within 2**15 clock cycles. This discard time out can be enabled via the “AHB IO base address and PCI bus config” located in the core specific Extended PCI Configuration Space.

77.6 DMA Controller

The DMA engine is descriptor base and uses two levels of descriptors.

77.6.1 DMA channel

The first level is a linked list of DMA channel descriptors. Each descriptor has a pointer to its data descriptor list and a pointer to the next DMA channel. The last DMA channel descriptor should always points to the first DMA channel for the list to be a closed loop. The descriptor needs to be aligned to 4 words (0x10) in memory and have the following structure.

*Table 1083.*GRPCI2: DMA channel descriptor structure

Descriptor address offset	**Descriptor word**
0x00	DMA channel control
0x04	Next DMA channel (32-bit address to next DMA channel descrip-tor).
0x08	Next data descriptor in this DMA channel (32-bit address to next data descriptor).
0x0C	RESERVED

Table 1084. GRPCI2 DMA channel control

31	30 25	24 22	21 20	19 16	15 0
EN	RESERVED	CID	Type	RESERVED	Data descriptor count

31	Channel descriptor enable.
30: 25	RESERVED
24: 22	Channel ID. Each DMA channel needs a ID to determine the source of a DMA interrupt.
21: 20	Descriptor type. 01 = DMA channel descriptor.
19: 16	RESERVED
15: 0	Maximum number of data destructors to be executed before moving to the next DMA channel. 0 indicates that all data descriptors should be executed before moving to the next DMA channel.

The number of enabled DMA channels must be stored in the "Number of DMA channels" field in the DMA control register accessible via the APB slave interface.

77.6.2 Data descriptor

The second descriptor level is a linked list of data transfers. The last descriptor in this list needs to be a disabled descriptor. To add a new data transfer, this disabled descriptor is updated to reflect the data transfer and to point to a new disabled descriptor. The control word in the descriptor should be updated last to enable the valid descriptor. To make sure the DMA engine reads this new descriptor, the enable bit in the DMA control register should be updated. The descriptor needs to be aligned to 4 words (0x10) in memory and have the following structure.

*Table 1085.*GRPCI2: DMA data descriptor structure

Descriptor address offset	**Descriptor word**
0x00	DMA data control
0x04	32-bit PCI start address
0x08	32-bit AHB start address
0x0C	Next data descriptor in this DMA channel (32-bit address to next data descriptor).

Table 1086. GRPCI2 DMA data control

31	30	29	28	22	21 20	19	18 16	15 0
EN	IE	DR	BE	RESERVED	Type	ER	RESERVED	LEN

31	Data descriptor enable.

Table 1086. GRPCI2 DMA data control

30	Interrupt generation enable.
29	Transfer direction. 0: PCI to AMBA, 1: AMBA to PCI.
28	PCI bus endianess switch. 1: defines the PCI bus to be little-endian for this transfer, 0: defines the PCI bus to be big-endian for this transfer.
27: 22	RESERVED (Must be set to zero)
21: 20	Descriptor type. 00 = DMA data descriptor.
19	Error status
18: 16	RESERVED
15: 0	Transfer length. The number of word of the transfer is (this field)+1.

77.6.3 Data transfer

The DMA engine starts by reading the descriptor for the first DMA channel. If the DMA channel is enabled the first data descriptor in this channel is read and executed. When the transfer is done the data descriptor is disabled and status is written to the control word. If no error occurred during the transfer, the error bit is not set and the transfer length field is unchanged. If the transfer was terminated because of an error, the error bit is set in the control word and the length field indicates where in the transfer the error occurred. If no error has occurred, the next data descriptor is read and executed. When a disabled data descriptor is read or the maximum number of data descriptors has been executed, the DMA channel descriptor is updated to point to the next data descriptor and the DMA engine moves on to the next DMA channel.

The DMA engine will stop when an error is detected or when no enabled data descriptors is found. The error type is indicated by bit 7 to bit 11 in the DMA control register. The error type bits must be cleared (by writing ‘1’) before the DMA can be reenabled.

77.6.4 Interrupt

The DMA controller has an interrupt enable bit in the DMA control register (accessible via the APB slave interface) which enables interrupt generation.

Each data descriptor has an interrupt enable bit which determine if the core should generate a interrupt when the descriptor has been executed.

The VHDL generic *irqmode* determines if the DMA engine assert the same interrupt as the PCI core or the DMA uses the irq signal following the PCI core interrupt, see the IRQ mode field in the Status and Capability register for irq routing information.

77.7 PCI trace buffer

77.7.1 Trace data

The data from the trace buffer is accessible in the last 128 kB block of the AHB I/O address area. Each 32-bit word in the first 64kB of this block represents a sample of the AD PCI signal. The second 64kB of the block is the corresponding PCI control signal. Each 32-bit word is defined in table 1087.

Table 1087. GRPCI2 PCI control signal trace (32-bit word)

31 - 20	19 - 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2 - 0
RESERVED	CBE[3:0]	FRAME	IRDY	TRDY	STOP	DEVSEL	PAR	PERR	SERR	IDSEL	REQ	GNT	LOCK	RST	RES

Table 1087. GRPCI2 PCI control signal trace (32-bit word)

31: 20	RESERVED
19: 3	The state of the PCI control signals.
2: 0	RESERVED

77.7.2 Triggering function

The core can be programmed to trigger on any combination of the PCI AD and PCI Control signals by setting up the desired pattern and mask in the PCI trace buffer registers accessible via the APB slave interface. Each bit the PCI AD signal and any PCI control signal can be masked (mask bit equal to zero) to always match the triggering condition.

The “Trig count” field in the “PCI trace buffer: counter & mode” register defines how many times the trigger condition should occur before the trace buffer disarms and eventually stops sampling. The number of samples stored after the triggering condition occurs defines by the “Delayed stop“ + 2.

To start sampling, the trace buffer needs to be armed by writing one to the start bit in the “PCI trace buffer: Control“ register. The state of the trace buffer can be determine by reading the Armed and Enable/Running bit in the this control register. When the Armed bit is set, the triggering condition has not occurred. The Enable/Running bit indicates that the trace buffer still is storing new samples. When the delayed stop is field is set to a non zero value, the Enabled bit is not cleared until all samples are stored in the buffer). The trace buffer can also be disarmed by writing the “stop” bit in the “PCI trace buffer: control” register.

When the trace buffer has been disarmed, the “trig index” in the “PCI trace buffer: control” register is updated with index of trace entry which match the triggering condition. The address offset of this entry is the value of the “trig index“ field times 4.

77.7.3 Trace Buffer APB interface

A separate APB register can optionally be enabled for access of the PCI trace buffer. The register layout is the same as the core APB interface but only registers related to the PCI trace buffer is accessable. The trace buffer data is located at offset 0x20000 for PCI AD and offset 0x30000 for PCI control signals.

77.8 Interrupts

The core is capable of sampling the PCI INTA-D signals and forwarding the interrupt to the APB bus. The PCI INTA-D signals can be connected to one APB irq signal or to 4 different irq signals. This is configured by the VHDL generic *irqmode*. The “host INT mask” field in the control register is used only for sampling the valid PCI INT signal.

The core supports PCI interrupt generation. For single function configuration the dirq signal is sampled and forwarded to the PCI INTA signal. For a multi function (and multi interrupt) configured device, each bit of the dirq signal is connected to one of the PCI INTA..D signal (dirq[0] => INTA, dirq[1] => INTB, ...). The core has a mask bit (the “device INT mask“ field in the control register) for each bit in the dirq vector. The core also has a PCI interrupt force bit in the control register to be able to force the PCI INT asserted. For a multi interrupt configuration the PCI interrupt force bit is masked by the “device INT mask” to be able to assert all PCI INT signals separately.

When the system error PCI signal (SERR) is sampled asserted the core sets the system error bit in the “core interrupt status” field in the Status & Capability register. If the system interrupts is enabled the core will also generate a interrupt on the APB bus.

77.9 Reset

The deassertion of the PCI reset is synchronized to the PCI clock and delayed 3 clock cycles.

The core can be configured to drive the AHB reset on the PCI reset signal. This option is used when the backplane does not have logic to drive the PCI reset.

The PCI reset signal can optionally be forwarded to the AHB reset via the ptarst signal. This functionality can be used then the AMBA clock domain needs to be reset when the PCI reset is asserted.

77.10 Registers

The core is configured via registers mapped into the APB memory address space.

*Table 1088.*GRPCI2: APB registers

APB address offset	**Register**
0x00	Control
0x04	Status & Capability (Read only)
0x08	PCI master prefetch burst limit
0x0C	AHB to PCI mapping for PCI IO
0x10	DMA Control & Status
0x14	DMA descriptor base
0x18	DMA channel active (read only)
0x1C	RESERVED
0x20 - 0x34	PCI BAR to AHB address mapping (Read only)
0x38	RESERVED
0x3C	RESERVED
0x40 - 0x7C	AHB master to PCI memory address mapping
0x80	PCI trace buffer: control & status
0x84	PCI trace buffer: counter & mode
0x88	PCI trace buffer: AD pattern
0x8C	PCI trace buffer: AD mask
0x90	PCI trace buffer: Ctrl signal pattern
0x94	PCI trace buffer: Ctrl signal mask
0x98	PCI trace buffer: AD state
0x9C	PCI trace buffer: Ctrl signal state

Table 1089. GRPCI2 Control register (address offset 0x00)

31	30	29	28	27	26	25	24	23 16	15 11	10	9	8	7 4	3 0
RE	MR	TR	R	SI	PE	ER	EI	Bus Number	RESERVED	IB	CB	DIF	Device INT mask	Host INT mask

31	PCI reset. When set, the PCI reset signal is asserted. Needs to be cleared to deassert PCI reset.
30	PCI master reset. Set to reset the cores PCI master. This bit is self clearing.
29	PCI target reset. Set to reset the cores PCI target. This bit is self clearing.
28	RESERVED
27	When set, Interrupt is enabled for System error (SERR)
26	When set, AHB error response is enabled for Parity error
25	When set, AHB error response is enabled for Master and Target abort.
24	When set, Interrupt is enabled for Master and Target abort and Parity error.

Table 1089. GRPCI2 Control register (address offset 0x00)

23: 16	When not zero, type 1 configuration cycles is generated.This field is also used as the Bus Number in type 1 configuration cycles.
15: 11	RESERVED
10	When set, burst accesses may be generated by the PCI master for PCI IO cycles
9	When set, burst accesses may be generated by the PCI master for PCI configuration cycles.
8	Device interrupt force. When set, a PCI interrupt is forced.
7: 4	Device interrupt mask. When bit[n] is set dirq[n] is unmasked

3: 0	Host interrupt mask
	bit[3] = 1: unmask INTD.
	bit[2] = 1: unmask INTC.
	bit[1] = 1: unmask INTB.
	bit[0] = 1: unmask INTA.

Table 1090. GRPCI2 Status and Capability register (address offset 0x04)

31	30	29	28	27	26	25 24	23	22	21	20	19	18 12	11 8	7 5	4 2	1 0
Host	MST	TAR	DMA	DI	HI	IRQ mode	Trace	RES	FH	CFGDO	CFGER	Core interrupt status	Host interrupt status	RES	FDEPTH	FNUM

31	When zero, the core is inserted in the System slot and is allowed to act as System Host.
30	Master implemented
29	Target implemented
28	DMA implemented
27	Device drives PCI INTA
26	Device samples PCI INTA..D (for host operations)
25: 24	APB IRQ mode 00: PCI INTA..D, Error interrupt and DMA interrupt on the same IRQ signal 01: PCI INTA..D and Error interrupt on the same IRQ signal. DMA interrupt on IRQ+1 10: PCI INTA..D on IRQ..IRQ+3. Error interrupt and DMA interrupt on IRQ. 11: PCI INTA..D on IRQ..IRQ+3. Error interrupt on IRQ. DMA interrupt on IRQ+4
23	PCI trace buffer implemented
22	RESERVED
21	Fake device in system slot (Host). This bit should always be written with ‘0’. Only for debugging.
20	PCI configuration access done, PCI configuration error status valid.
19	Error during PCI configuration access
18: 12	Interrupt status: bit[6]: PCI target access discarded due to time out (access not retried for 2**15 PCI clock cycles) bit[5]: System error bit[4]: DMA interrupt bit[3]: DMA error bit[2]: Master abort. bit[1]: Target abort. bit[0]: Parity error.

11: 8	Host interrupt status
	bit[3] = 0: indicates that INTD is asserted.
	bit[2] = 0: indicates that INTC is asserted.
	bit[1] = 0: indicates that INTB is asserted.
	bit[0] = 0: indicates that INTA is asserted.

7: 5	RESERVED
4: 2	Words in each FIFO = 2**(FIFO depth)
1: 0	Number of FIFOs

Table 1091. GRPCI2 PCI master prefetch burst limit (address offset 0x08)

31 : 16	15 : 8	7 : 0
AHB master unmask	RESERVED	Burst length

31 : 16	When bit[n] is set, the prefetch burst of AHB master n is limited by the “Burst length” field.
15 : 8	RESERVED
7 : 0	Maximum number of beats - 1 in the burst. (Maximin value is 0xFF => 0x100 beats => 1kB address)

Table 1092. GRPCI2 AHB to PCI mapping for PCI IO (address offset 0x0C)

31 : 16	15 : 0
AHB to PCI IO	RESERVED

31 : 16	Used as the MSBs of the base address for a PCI IO access.
15 : 0	RESERVED

Table 1093. GRPCI2 DMA control and status register (address offset 0x10)

31	30 - 20	19 : 12	11	10	9	8	7	6 : 4	3	2	1	0
SAFE	RES	CHIRQ	MA	TA	PE	AE	DE	Number of DMA channels	ACTIVE	DIS	IE	EN

31	Safety guard for update of control fields. Needs to be set to ‘1’ for the control fields to be updated.
30 : 20	RESERVED
19 : 12	Channel IRQ status. Set to ‘1’ when a descriptor is configured to signal interrupt. bit[0] corresponds to the channel with ID 0, bit[1] corresponds to the channel with ID 1, ... Clear by writing ‘1’.
11	Master abort during PCI access. Clear by writing ‘1’
10	Target abort during PCI access. Clear by writing ‘1’
9	Parity error during PCI access. Clear by writing ‘1’
8	Error during AHB data access. Clear by writing ‘1’
7	Error during descriptor access. Clear by writing ‘1’.
6 : 4	Number of DMA channels (Guarded by bit[31], safety guard)
3	DMA is active (read only)
2	DMA disable/stop. Writing ‘1’ to this bit disables the DMA.
1	Interrupt enable (Guarded by bit[31], safety guard).
0	DMA enable/start. Writing ‘1’ to this bit enables the DMA.

Table 1094. GRPCI2 DMA descriptor base address register (address offset 0x14)

31 : 0
DMA descriptor base address

31 : 0	Base address of the DMA descriptor table. When running, this register points to the active descriptor.

Table 1095. GRPCI2 DMA channel active register (address offset 0x18)

31 : 0
DMA descriptor base address

Table 1095. GRPCI2 DMA channel active register (address offset 0x18)

31 : 0	Base address of the active DMA channel.

Table 1096. GRPCI2 PCI BAR to AHB address mapping register (address offset 0x20 - 0x34)

31 0
PCI BAR to AHB address mapping

31 : 0	32-bit mapping register for each PCI BAR. Translate an access to a PCI BAR to a AHB base address.

Table 1097. GRPCI2 AHB master to PCI memory address mapping register (address offset 0x40 - 0x7C)

31 0
AHB master to PCI memory address mapping

31 : 0	32-bit mapping register for each AHB master. Translate an access from a specific AHB master to a PCI base address. The size of the AHB slave address area determine how many bits (starting from bit 31) are implemented. Bits not implemented returns zero. The mapping register for AHB master 0 is located at offset 0x40, AHB master 1 at offset 0x44, and so on up to AHB master 15 at offset 0x7C. Mapping registers are only implemented for existing AHB masters.

Table 1098. GRPCI2 PCI trace Control and Status register (address offset 0x80)

31 16	15	14	13 12	11 4	3 2	1	0
TRIG INDEX	AR	EN	RES	DEPTH	RES	SO	SA

31: 16	Index of the first entry of the trace.
15	Set when trace buffer is armed (started but the trig condition has not occurred).
14	Set when trace buffer is running
13: 12	RESERVED
11: 4	Number of buffer entries = 2**DEPTH
3: 2	RESERVED
1	Stop tracing. (Write only)
0	Start tracing. (Write only)

Table 1099. GRPCI2 PCI trace counter and mode register (address offset 0x84)

31 28	27 24	23 16	15 0
RES	Trace mode	Trig count	Delayed stop

31: 28	RESERVED
27: 24	Tracing mode 00: Continuos sampling 01: RESERVED 10: RESERVED 11: RESERVED
23: 16	The number the trig condition should occur before the trace is disarmed.
15: 0	The number of entries stored after the trace buffer has been disarmed. (Should not be lager then number of buffer entries - 2).

Table 1100. GRPCI2 PCI trace AD pattern register (address offset 0x88)

31 … 0
PCI AD pattern

31: 0	AD pattern to trig on

Table 1101. GRPCI2 PCI trace AD mask register (address offset 0x8C)

31 … 0
PCI AD mask

31: 0	Mask for the AD patter. When mask bit[n] = 0 pattern bit[n] will always be a match.

Table 1102. GRPCI2 PCI trace Ctrl signal pattern register (address offset 0x90)

31 … 20	19 … 16	15	14	13	12	11	10	9	8	7	6	5	4	3	… 0
RESERVED	CBE[3:0]	FRAME	IRDY	TRDY	STOP	DEVSEL	PAR	PERR	SERR	IDSEL	REQ	GNT	LOCK	RST	RES

31: 20	RESERVED
19: 3	PCI Ctrl signal pattern to trig on
2: 0	RESERVED

Table 1103. GRPCI2 PCI trace Ctrl signal mask register (address offset 0x94)

31 … 20	19 … 16	15	14	13	12	11	10	9	8	7	6	5	4	3	… 0
RESERVED	CBE[3:0]	FRAME	IRDY	TRDY	STOP	DEVSEL	PAR	PERR	SERR	IDSEL	REQ	GNT	LOCK	RST	RES

31: 20	RESERVED
19: 3	Mask for the Ctrl signal patter. When mask bit[n] = 0 pattern bit[n] will always be a match.
2: 0	RESERVED

Table 1104. GRPCI2 PCI trace PCI AD state register (address offset 0x98)

31 … 0
Sampled PCI AD signal

31: 0	The state of the PCI AD signal.

Table 1105. GRPCI2 PCI trace PCI Ctrl signal state register (address offset 0x9C)

31 20	19 16	15	14	13	12	11	10	9	8	7	6	5	4	3	0
RESERVED	CBE[3:0]	FRAME	IRDY	TRDY	STOP	DEVSEL	PAR	PERR	SERR	IDSEL	REQ	GNT	LOCK	RST	RES

31: 20	RESERVED
19: 3	The state of the PCI Ctrl signals.
2: 0	RESERVED

77.11 Configuration options

Table 1106 shows the configuration options of the core (VHDL generics).

*Table 1106.*Configuration options

Generic name	Function	Allowed range	Default
memtech	The memory technology used for the internal FIFOs.	0 - NTECH	0
tbmemtech	The memory technology used for trace buffers	0 - NTECH	0
oepol	Polarity of the pad output enable signal. 0 = active low, 1 = active high.	0 - 1	0
hmindex	AHB master index.	0 - NAHBMST-1	0
hdmindex	DMA AHB master index.	0 - NAHBMST-1	0
hsindex	AHB slave index.	0 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR (for PCI memory access).	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR.	0 - 16#FFF#	16#000#
ioaddr	ADDR field of the AHB IO BAR (for PCI configuration and PCI IO access).	0 - 16#FFF#	16#000#
pindex	APB slave index	0 - APBMAX-1	0
paddr	APB interface base address	0 - 16#FFF#	0
pmask	APB interface address mask	0 - 16#FFF#	16#FFF#
irq	Interrupt line used by the core.	0 - NAHBIRQ-1	0
irqmode	IRQ routing option: 00: PCI INTA..D, Error interrupt and DMA interrupt on the same IRQ signal 01: PCI INTA..D and Error interrupt on the same IRQ signal. DMA interrupt on IRQ+1 10: PCI INTA..D on IRQ..IRQ+3. Error interrupt and DMA interrupt on IRQ. 11: PCI INTA..D on IRQ..IRQ+3. Error interrupt on IRQ. DMA interrupt on IRQ+4	0 - 3	0
master	Enable the PCI master	0 - 1	1
target	Enable the PCI target	0 - 1	1
dma	Enable the PCI dma	0 - 1	1
tracebuffer	Enable and number of entries of the PCI trace buffer, Allowed values is 0, 32, 64, 128, ..., 16384.	0 - 16384	0
confspace	Enable the PCI Configuration Space when PCI target is disabled	0 - 1	1
vendorid	PCI vendor ID	0 - 16#FFFF#	0
deviceid	PCI device ID	0 - 16#FFFF#	0
classcode	PCI class code	0 - 16#FFFFFF#	0
revisionid	PCI revision ID	0 - 16#FF#	0
cap_pointer	Enabled and sets the offset of the first item in the Extended PCI Configuration Space	0 - 16#C0#	0
ext_cap_pointer	Offset of the first user defined item in the capability list	0 - 16#FC#	0
iobase	AHB base address of the AHB I/O area	0 - 16#FFF#	16#FFF#
extcfg	Default value of the user defined Extended PCI Configu-ration Space to AHB address mapping.	0 - 16#FFFFFFF#	0
bar0	Sets the default size of BAR0 in address bits.	0 - 31	0
bar1	Sets the default size of BAR1 in address bits.	0 - 31	0
bar2	Sets the default size of BAR2 in address bits.	0 - 31	0
bar3	Sets the default size of BAR3 in address bits.	0 - 31	0

*Table 1106.*Configuration options

Generic name	Function	Allowed range	Default
bar4	Sets the default size of BAR4 in address bits.	0 - 31	0
bar5	Sets the default size of BAR5 in address bits.	0 - 31	0
bar0_map	Set the default PCI BAR to AHB address mapping for BAR0	0 - 16#FFFFFF#	0
bar1_map	Set the default PCI BAR to AHB address mapping for BAR1	0 - 16#FFFFFF#	0
bar2_map	Set the default PCI BAR to AHB address mapping for BAR2	0 - 16#FFFFFF#	0
bar3_map	Set the default PCI BAR to AHB address mapping for BAR3	0 - 16#FFFFFF#	0
bar4_map	Set the default PCI BAR to AHB address mapping for BAR4	0 - 16#FFFFFF#	0
bar5_map	Set the default PCI BAR to AHB address mapping for BAR5	0 - 16#FFFFFF#	0
bartype	Bit[5:0] set the reset value of the prefetch bit for the BAR. bit[n] corresponds to BARn . Bit[13:8] set the reset value of the BAR type bit for the BAR. bit[n + 8] corresponds to BARn.	0 - 16#FFFF#	0
barminsize	Sets the minimal supported BAR size in address bits. The minimal BAR size is not allowed to be smaller then the internal FIFO (or barminsize >= 2 + fifo_depth).	5 - 31	12
fifo_depth	Depth of each of the FIFOs in the data path. Depth = 2**fifo_depth.The minimal BAR size is not allowed to be smaller then the internal FIFO (or barminsize >= 2 + fifo_depth).	3 - 7	3
fifo_count	Number of FIFOs in the data path	2 - 4	2
conv_endian	Default value of the endianess conversion setting	0 - 1	1
deviceirq	Enable the device to drive the PCI INTA signal	0 - 1	1
deviceirqmask	Default value of the irq mask for the dirq input	0 - 16#F#	16#0#
hostirq	Enable the core to sample the PCI INTA-D signals to drive a AHB irq.	0 - 1	1
hostirqmask	Default value for the PCI INTA-D signals.	0 - 16#F#	16#0#
nsync	Number of synchronization registers between the two clock domains.	0 - 2	2
hostrst	Mode of the reset signal. 0: PCI reset is input only 1: The AHB reset is driven on the PCI reset when PCII.HOST is asserted 2: The AHB reset is driven on the PCI reset.	0 - 2	0
bypass	When 1, logic is implemented to bypass the pad on signals driven by the core.	0 - 1	1
ft	Enable fault-tolerance against SEU errors	0 - 1	0

*Table 1106.*Configuration options

Generic name	Function	Allowed range	Default
scantest	Enable support for scan test	0 - 1	0
debug	Enables debug output signals	0 - 1	0
tbapben	Enables a separate APB interface for access of the Trace-Buffer.	0 - 1	0
tbpindex	Trace-Buffer APB slave index	0 - APBMAX-1	0
tbpaddr	Trace-Buffer APB interface base address	0 - 16#FFF#	0
tbmask	Trace-Buffer APB interface address mask	0 - 16#FFF#	16#FFF#
netlist	Enables a netlist implementation of the logic controlled by the PCI bus signals (GRPCI2_PHY).	0 - 1	0
masters	Controls which AHB masters belongs to PCI function0	0 - 16#FFFF#	16#FFFF#
multifunc	Enables Multi-Function support	0 - 1	0
multiint	Enables support to drive all PCI interrupt signals-INTA...D	0 - 1	0
mf1_deviceid	PCI device ID (PCI function1)	0 - 16#FFFF#	0
mf1_classcode	PCI class code (PCI function1)	0 - 16#FFFFFF#	0
mf1_revisionid	PCI revision ID (PCI function1)	0 - 16#FF#	0
mf1_bar0	Sets the default size of BAR0 in address bits. (PCI function1)	0 - 31	0
mf1_bar1	Sets the default size of BAR1 in address bits. (PCI function1)	0 - 31	0
mf1_bar2	Sets the default size of BAR2 in address bits. (PCI function1)	0 - 31	0
mf1_bar3	Sets the default size of BAR3 in address bits. (PCI function1)	0 - 31	0
mf1_bar4	Sets the default size of BAR4 in address bits. (PCI function1)	0 - 31	0
mf1_bar5	Sets the default size of BAR5 in address bits. (PCI function1)	0 - 31	0
mf1_bartype	Bit[5:0] set the reset value of the prefetch bit for the BAR. bit[n] corresponds to BARn . Bit[13:8] set the reset value of the BAR type bit for the BAR. bit[n + 8] corresponds to BARn.	0 - 16#FFFF#	0
mf1_bar0_map	Set the default PCI BAR to AHB address mapping for BAR0 (PCI function1)	0 - 16#FFFFFF#	0
mf1_bar1_map	Set the default PCI BAR to AHB address mapping for BAR1 (PCI function1)	0 - 16#FFFFFF#	0
mf1_bar2_map	Set the default PCI BAR to AHB address mapping for BAR2 (PCI function1)	0 - 16#FFFFFF#	0
mf1_bar3_map	Set the default PCI BAR to AHB address mapping for BAR3 (PCI function1)	0 - 16#FFFFFF#	0
mf1_bar4_map	Set the default PCI BAR to AHB address mapping for BAR4 (PCI function1)	0 - 16#FFFFFF#	0
mf1_bar5_map	Set the default PCI BAR to AHB address mapping for BAR5 (PCI function1)	0 - 16#FFFFFF#	0
mf1_cap_pointer	Enabled and sets the offset of the first item in the Extended PCI Configuration Space (PCI function1)	0 - 16#C0#	0
mf1_ext_cap_pointer	Offset of the first user defined item in the capability list (PCI function1)	0 - 16#FC#	0

*Table 1106.*Configuration options

Generic name	Function	Allowed range	Default
mf1_extcfg	Default value of the user defined Extended PCI Configuration Space to AHB address mapping. (PCI function1)	0 - 16#FFFFFFF#	0
mf1_masters	Controls which AHB masters belongs to PCI function1	0 - 16#FFFF#	16#0000#

77.12 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x07C. The DMA engine has device identifier 0x07D. The separate APB interface for the PCI Trace-Buffer has device identifier 0x07E. For description of vendor and device identifier see GRLIB IP Library User's Manual

77.13 Implementation

77.13.1 Technology mapping

The core has a technology mapping VHDL generic, *memtech*, which controls how the memory cell used will be implemented. See the GRLIB Users's Manual for available settings.

77.13.2 RAM usage

The FIFOs in the core is implemented with the *syncram_2pft* (with separate clocks for each port) component from the technology mapping library (TECHMAP). Each data path implements its FIFOs in a separate 32-bit wide syncram_2pft component. The depth of each of these RAMs is the FIFO depth * number of FIFOs.

77.13.3 Pull-ups

Please refer to the PCI Local Bus Specification on which of the PCI signals needs to have pull-ups for correct operations.

77.13.4 PHY

All logic and registers directly controlled by the PCI bus signals has be placed in a separate entity. This makes it easier to control the setup-, hold- and clock-to-out timing for the PCI bus signals. This logic can also be implemented as a netlist which can be manually placed before running place-and-route for the entire design. A netlist is provided for Axcelerator and RTAX targets.

77.14 Signal descriptions

Table 1107 shows the interface signals of the core (VHDL ports).

*Table 1107.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
PCICLK	N/A	Input	PCI Clock	-
AHBSI	*1	Input	AHB slave input signals	-
AHBSO	*1	Output	AHB slave output signals	-
AHBMI	*1	Input	AHB master input signals	-
AHBMO	*1	Output	AHB master output signals	-
AHBDMO	*1	Output	DMA AHB master output signals	-
APBI	*1	Input	APB slave input signals	-
APBO	*1	Output	APB slave output signals	-
PCII	*2	Input	PCI input signals	-
PCIO	*2	Output	PCI output signals	-
DIRQ		Input	Interrupt signals	High
TBAPBI	*1,*3	Input	Trace-Buffer APB slave input signals	-
TBAPBO	*1, *3	Output	Trace-Buffer APB slave output signals	-
PTARST	N/A, *3	Output	PCI reset to AMBA reset output signal	Low
DEBUG	N/A, *3	Output	Debug signals	-

*1) see GRLIB IP Library User’s Manual.
*2) see PCI Local Bus Specification
*3) Can be left unconnected, if not used.

The PCII.HOST signal selects of the core should operate as a system host or peripheral device.

77.15 Library dependencies

Table 1108 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1108.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	PCI	Component	Component declaration

77.16 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
use grlib.tech.all;
library gaisler;
use gaisler.pci.all;

.
.
signal apbi : apb_slv_in_type;
signal apbo : apb_slv_out_type;
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector;
signal ahbmi : ahb_mst_in_type;
signal ahbmo : ahb_mst_out_vector;

signal pcii : pci_in_type;
signal pcio : pci_out_type;

begin

pci0 : grpci2
generic map (
 memtech => memtech,
 oepol => OEPOL,
 hmindex => CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG,
 hdmindex => CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG+1,
 hsindex => 4,
 haddr => 16#c00#,
 hmask => 16#f00#,
 ioaddr => 16#000#,
 pindex => 4,
 paddr => 4,
 irq => 0,
 irqmode => 0,
 master => CFG_GRPCI2_MASTER,
 target => CFG_GRPCI2_TARGET,
 dma => CFG_GRPCI2_DMA,
 tracebuffer => CFG_GRPCI2_TRACE,
 vendorid => CFG_GRPCI2_VID,
 deviceid => CFG_GRPCI2_DID,
 classcode => CFG_GRPCI2_CLASS,
 revisionid => CFG_GRPCI2_RID,
 cap_pointer => CFG_GRPCI2_CAP,
 ext_cap_pointer => CFG_GRPCI2_NCAP,
 iobase => CFG_AHBIO,
 extcfg => CFG_GRPCI2_EXTCFG,
 bar0 => CFG_GRPCI2_BAR0,
 bar1 => CFG_GRPCI2_BAR1,
 bar2 => CFG_GRPCI2_BAR2,
 bar3 => CFG_GRPCI2_BAR3,
 bar4 => CFG_GRPCI2_BAR4,
 bar5 => CFG_GRPCI2_BAR5,
 fifo_depth => log2(CFG_GRPCI2_FDEPTH),
 fifo_count => CFG_GRPCI2_FCOUNT,
 conv_endian => CFG_GRPCI2_ENDIAN,
 deviceirq => CFG_GRPCI2_DEVINT,
 deviceirqmask => CFG_GRPCI2_DEVINTMSK,
 hostirq => CFG_GRPCI2_HOSTINT,
 hostirqmask => CFG_GRPCI2_HOSTINTMSK,
 nsync => 2,
 hostrst => 2,
 bypass => CFG_GRPCI2_BYPASS)
port map (
 rstn,
 clkm,
 pciclk,
 gnd(3 downto 0),
 pcii,
 pcio,
 apbi,
 apbo(4),
 ahbsi,
 ahbso(4),
 ahbmi,
 ahbmo(CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG),
 ahbmo(CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG+1);

pcipads0 : pcipads generic map (padtech => padtech, host => 1, oepol => OEPOL,
                                    noreset => 0, drivereset => 1)  -- PCI pads
    port map ( pci_rst, pci_gnt, pci_idsel, pci_lock, pci_ad, pci_cbe,
      pci_frame, pci_irdy, pci_trdy, pci_devsel, pci_stop, pci_perr,
      pci_par, pci_req, pci_serr, pci_host, pci_66, pcii, pcio );
;
```

78 PCIDMA - DMA Controller for the GRPCI interface

78.1 Introduction

The DMA controller is an add-on interface to the GRPCI interface. This controller perform bursts to or from PCI bus using the master interface of GR PCI Master/target unit.

Figure 1 below illustrates how the DMA controller is attached between the AHB bus and the PCI master interface.

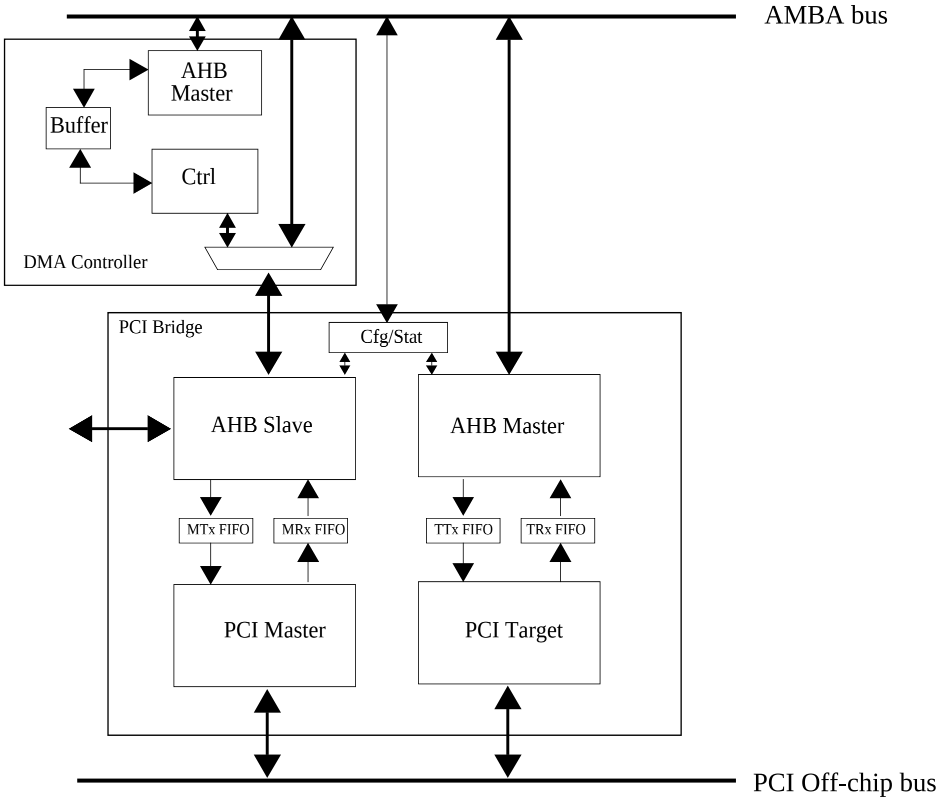


Figure 252. DMA Controller unit

78.2 Operation

The DMA controller is set up by defining the location of memory areas between which the DMA will take place in both PCI and AHB address space as well as direction, length and type of the transfer. Only 32-bit word transfer are supported.

The DMA transfer is automatically aborted when any kind of error is detected during a transfer. The DMA controller does not detect deadlocks in its communication channels. If the system concludes that a deadlock has occurred, it can manually abort the DMA transfer. It is allowed to perform burst over a 1 Kbyte boundary of the AHB bus. When this happens, an AHB idle cycle will be automatically inserted to break up the burst over the boundary. The core can be configured to generate a interrupt when the transfer is completed.

When the DMA is not active the AHB slave interface of PCI Master/Target unit will be directly connected to AMBA AHB bus.

78.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 1109.*DMA Controller registers

Address offset	**Register**
0x00	Command/status register
0x04	AMBA Target Address
0x08	PCI Target Address
0x0C	Burst length

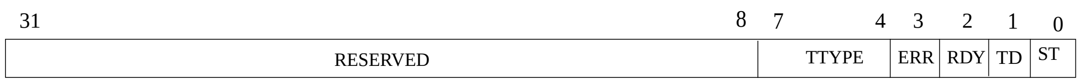


Figure 253. Status/Command register

[31:8]: Reserved.

[7:4]: Transfer Type (TTYPE) - Perform either PCI Memory or I/O cycles. “1000” - memory cycles, “0100” - I/O cycles. This value drives directly HMBSEL signals on PCI Master/Targets units AHB Slave interface.

[3]: Error (ERR) - Last transfer was abnormally terminated. If set by the DMA Controller this bit will remain zero until cleared by writing ‘1’ to it.

[2]: Ready (RDY) - Current transfer is completed. When set by the DMA Controller this bit will remain zero until cleared by writing ‘1’ to it.

[1]: Transfer Direction (TD) - ‘1’ - write to PCI, ‘0’ - read form PCI.

[0]: Start (ST) - Start DMA transfer. Writing ‘1’ will start the DMA transfer. All other registers have to be set up before setting this bit. Set by the PCI Master interface when its transaction is terminated with Target-Abort. Writing ‘1’

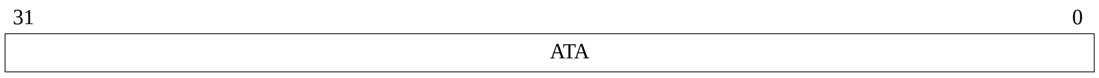


Figure 254. AMBA Target Address

[31:0]: AMAB Target Address (ATA) - AHB start address for the data on AMBA bus. In case of error, it indicated failing address.

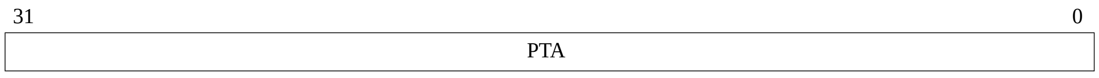


Figure 255. PCI Target Address

[31:0]: PCI Target Address (PTA) - PCI start address on PCI bus. This is a complete 32-bit PCI address and is not further mapped by the PCI Master/Target unit. In case of error, it indicated failing address.

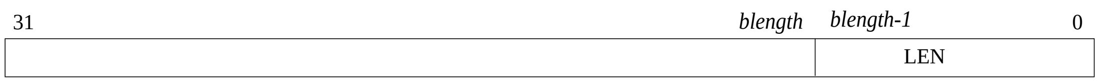


Figure 256. Length register

[*blentgh-1*:0]: DMA Transfer Length (LEN) - Number of 32-bit words to be transferred.

78.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x016. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

78.5 Configuration options

Table 1110 shows the configuration options of the core (VHDL generics).

*Table 1110.*Configuration options

Generic	Function	Allowed range	Default
mstndx	DMA Controllers AHB Master interface index	0 - NAHBMST-1	0
apbndx	The AMBA APB index for the configuration/status APB interface	0 - NAPBMAX-1	0
apbaddr	APB interface base address	0 - 16#FFF#	0
apbmask	APB interface address mask	0 - 16#FFF#	16#FFF#
apbirq	APB interrupt line	0 to MAXIRQ-1	0
blength	Number of bits in the Burst length register	-	16

78.6 Signal description

Table 1111 shows the interface signals of the core (VHDL ports).

*Table 1111.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	AMBA system clock	-
PCICLK	N/A	Input	PCI clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
AHBSI0	*	Input	AHB slave input signals, main AHB bus	-
AHBSO0	*	Output	AHB slave output signals, main AHB bus	-
AHBSI1	*	Input	AHB slave input signals, connected to PCI Target/Master unit	-
AHBSO1	*	Output	AHB slave output signals, connected to PCI Target/Master unit	-

* see GRLIB IP Library User's Manual

78.7 Library dependencies

Table 1112 shows libraries used when instantiating the core (VHDL libraries).

*Table 1112.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	PCI	Component	Component declaration

78.8 Instantiation

This example shows how the core can be instantiated

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.pci.all;
use gaisler.pads.all;

signal pcii : pci_in_type;
signal pcio : pci_out_type;

dma : pcidma generic map (memtech => memtech, dmstndx => 1,
  dapbndx => 5, dapbaddr => 5, blength => blength, mstndx => 0,
  fifodepth => log2(fifodepth), device_id => CFG_PCIDID, vendor_id => CFG_PCIVID,
  slvndx => 4, apbndx => 4, apbaddr => 4, haddr => 16#E00#, ioaddr => 16#800#,
  nsync => 1)
port map (rstn, clkm, pciclk, pcii, pcio, apbo(5), ahbmo(1),
apbi, apbo(4), ahbmi, ahbmo(0), ahbsi, ahbso(4));


pcipads0 : pcipads generic map (padtech => padtech)
port map ( pci_rst, pci_gnt, pci_idsel, pci_lock, pci_ad, pci_cbe,
      pci_frame, pci_irdy, pci_trdy, pci_devsel, pci_stop, pci_perr,
      pci_par, pci_req, pci_serr, pci_host, pci_66, pcii, pcio );
```

79 PCITB_MASTER_SCRIPT - Scriptable PCI testbench master

79.1 Overview

The PCITB_MASTER_SCRIPT IP core provides a simulation model for a PCI master with system host capabilities. The user specifies the requested PCI commands and check their result in a script file which has an easy to use syntax.

Features:

- Support for PCI memory, i/o and configuration cycles
- Easy to use scripting syntax
- Write/read PCI data to/from files
- Flexible burst length (1-65535 words)
- User can specify byte enables for each data phase

79.2 Operation

As soon as the PCI reset is released the PCI master begins to parse the specified script file. It will display the operations performed in the simulation console.

The following commands are supported. Comments are supported in script files and such lines must begin with ‘#’. Note that address and data must be specified in hexadecimal width 8 digits and that length must be hexadecimal with 4 digits. Each command should be ended with a semicolon.

79.2.1 wait <cycles>;

The master waits the specified number of clock cycles.

Example:

```
wait 5;
```

79.2.2 comp <file1> <file2>;

Compare file1 to file1 and display the result.

Example:

```
comp test1.log test2.log;
```

79.2.3 stop;

End PCI master operation.

79.2.4 halt;

Halt simulation.

79.2.5 print <string>;

Prints everything between ‘print’ and ‘;’ to the simulation console.

Example:

```
print PCI testbench print example;
```

79.2.6 estop <0|1>;

Turn on (1) or off (0) stop on error. If on, the PCI test master will stop the testbench with a failure if any error is detected.

Example:

```
estop 1;
```

79.2.7 rcfg <addr> <data | *>;

Perform a configuration read cycle from address ‘addr’ and compare to ‘data’. If a ‘*’ is specified instead of a data word then no checking is done.

Examples:

```
rcfg 10000000 12345678;
rcfg 10000000 *;
```

79.2.8 wcfg <addr> <data.cbe>;

Perform a configuration write cycle with the specified data to address ‘addr’. Specifying the byte enables is optional.

Examples:

```
wcfg 10000000 12345678;
wcfg 10000000 12345678.C;
```

79.2.9 rmem <cmd> <addr> <length> <data | filename | *>;

Perform a PCI read transaction using command ‘cmd’ from address ‘addr’ and compare to the specified data. If a ‘*’ is specified then no checking is done. If a filename is specified then the read data is stored in that file. Note that data must be specfied between braces. If byte enables are specified for a word then only the enabled bytes will be compared.

Examples:

```
# Burst read 4 words and compare against specified data
rmem C 10000000 0004 {
12345678
87654321
AA55AA55
11223344
};
# Read a single word and compare using CBE=3
rmem 6 10000000 0001 {
12345678.3
};
# Read single word and ignore result
rmem 6 10000000 0001 *;

# Burst read 64 words and store in file read.log
rmem C 10000000 0040 read.log;
```

79.2.10 wmem <cmd> <addr> <length> <data | filename>;

Perform a PCI write transaction using command ‘cmd’ to address ‘addr’. Note that data must be specfied between braces. If a filename is specified then the data is read from that file.

Examples:

```
# Burst write 4 words
wmem 7 10000000 0004 {
12345678
87654321
AA55AA55
11223344
};

# Write a single word using CBE=3
wmem 7 10000000 0001 {
12345678.3
};

# Write burst 64 words from file write.txt (write.txt should have
# one word per line)
wmem 7 10000000 0040 write.txt;
```

79.2.11 Example script file

```
#########################
# PCI TEST SCRIPT
#########################

# Wait 10 clock cycles
wait 10;

# Read vendor/device id and compare to 0xBACCFEED (of device with idsel <= ad31)
rcfg 80000000 BACCFEED;

# Enable memory space
wcfg 80000004 00000002;

# Set BAR0 to 0x10000000
wcfg 80000010 10000000;

# Set BAR1 to 0x20000000
wcfg 80000014 20000000;

# Write single word to PAGE0 (BAR0 + 0x8000).
# BAR0 -> APB, byte twisting enabled
wmem 7 10008000 0001 {
80000001
};

# Read single word from BAR0 and compare to 0x80000000
rmem 6 10008000 0001 {
80000001
};

# Set GRPCI PAGE1 to 0x40000000 (RAM) through BAR0 (BAR0 + 0x410)
wmem 7 10000410 0001 {
00000040.0
};

# Read back PAGE1 and  compare
rmem 6 10000410 0001 {
00000040
};

# Burst write 8 words to BAR1 (memory)
wmem 7 20000000 0008 {
11223344.3
22334411.0
33441122.0
44112233.0
12345678.0
87654321.0
a5a5a5a5.0
00001111.c
};

# Burst read 8 words and check against specified data
rmem C 20000000 0008 {
11223344.3
22334411
33441122
44112233
12345678
87654321
a5a5a5a5
00001111.c
};

# Burst write 64 words from file wf1.txt
wmem 7 20000000 0040 wf1.txt;

# Burst read 64 words and store result in rf1.txt
rmem C 20000000 0040 rf1.txt;

# Compare wf1.txt with rf1.txt
comp wf1.txt rf1.txt;

# End of Simulation
stop;
```

79.3 Configuration options

Table 1113 shows the configuration options of the core (VHDL generics).

*Table 1113.*Configuration options

Generic	Function	Allowed range	Default
slot	PCI slot used by master. Determines which req/gnt pair to use	0-20	0
tval	Output delay for signals that are driven by this unit	0-7 ns	7 ns
dbglevel	Debug level. Higher value means more debug information	0-2	1
maxburst	Maximum burst length supported	1-65535	1024
filename	PCI command script file	-	pci.cmd

79.4 Signal descriptions

Table 1114 shows the interface signals of the core (VHDL ports).

*Table 1114.*Signals descriptions

Signal name	Field	Type	Function	Active
PCIIN	*	Input	PCI input signals	-
PCIOUT	*	Output	PCI output signals	-

*1) PCI_TYPE (declared in pcitb.vhd). See PCI specification for more info on PCI signals

79.5 Library dependencies

Table 1115 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1115.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GAISLER	PCITB	Signals, component	PCI TB signals and component declaration

80 PCITARGET - Simple 32-bit PCI target with AHB interface

80.1 Overview

This core implements PCI interface with a simple target-only interface. The interface is developed primarily to support DSU communication over the PCI bus. Focus has been put on small area and robust operation, rather than performance. The interface has no FIFOs, limiting the transfer rate to about 5 Mbyte/s. This is however fully sufficient to allow fast download and debugging using the DSU.

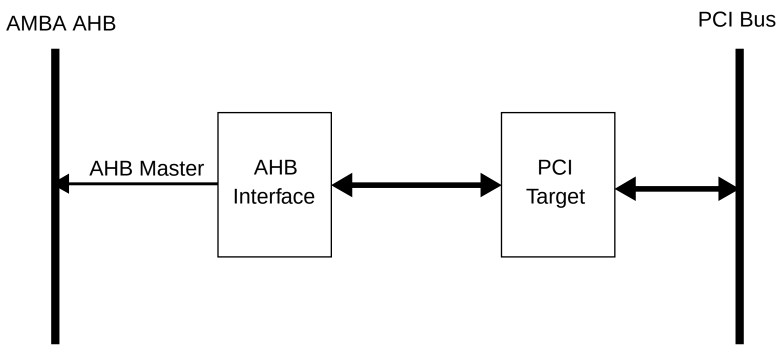


Figure 257. Target-only PCI interface

80.2 Registers

The core implements one PCI memory BAR.

31 — *abits*-1	*abits*-2 — 0
AHB address [31:*abits*-1]	UNUSED

Figure 258. AHB address register (BAR0, 0x100000)

The interface consist of one PCI memory BAR occupying (2^*abits*) bytes (default: 2 Mbyte) of the PCI address space, and an AHB address register. Any access to the lower half of the address space (def.: 0 - 0xFFFFF) will be forwarded to the internal AHB bus. The AHB address will be formed by concatenating the AHB address field of AHB address register with the LSB bits of the PCI address. An access to the upper half of the address space (default: 1 Mbyte on 0x100000 - 0x1FFFFF) of the BAR will read or write the AHB address register.

80.3 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x012. For description of vendor and device identifies see GRLIB IP Library User’s Manual.

80.4 Configuration options

Table 1116 shows the configuration options of the core (VHDL generics).

*Table 1116.*Configuration options

Generic	Function	Allowed range	Default
hindex	Selects which AHB select signal (HSEL) will be used to access the PCI target core	0 to NAHBMAX-1	0
abits	Number of bits implemented for PCI memory BAR	0 to 31	21
device_id	PCI device id	0 to 65535	0
vendor_id	PCI vendor id	0 to 65535	0
nsync	One or two synchronization registers between clock regions	1 - 2	1
oepol	Polarity of output enable signals. 0=active low, 1=active high	0 - 1	0

80.5 Signal descriptions

Table 1117 shows the interface signals of the core (VHDL ports).

*Table 1117.*Signals descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	AHB system clock	-
PCICLK	N/A	Input	PCI clock	-
PCII	*1	Input	PCI input signals	-
PCIO	*1	Output	PCI output signals	-
APBI	*2	Input	APB slave input signals	-
APBO	*2	Output	APB slave output signals	-

*1) see PCI specification
*2) see GRLIB IP Library User's Manual

The PCIO record contains an additional output enable signal vaden. It is has the same value as aden at each index but they are all driven from separate registers. A directive is placed on this vector so that the registers will not be removed during synthesis. This output enable vector can be used instead of aden if output delay is an issue in the design.

80.6 Library dependencies

Table 1118 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1118.*Library dependencies

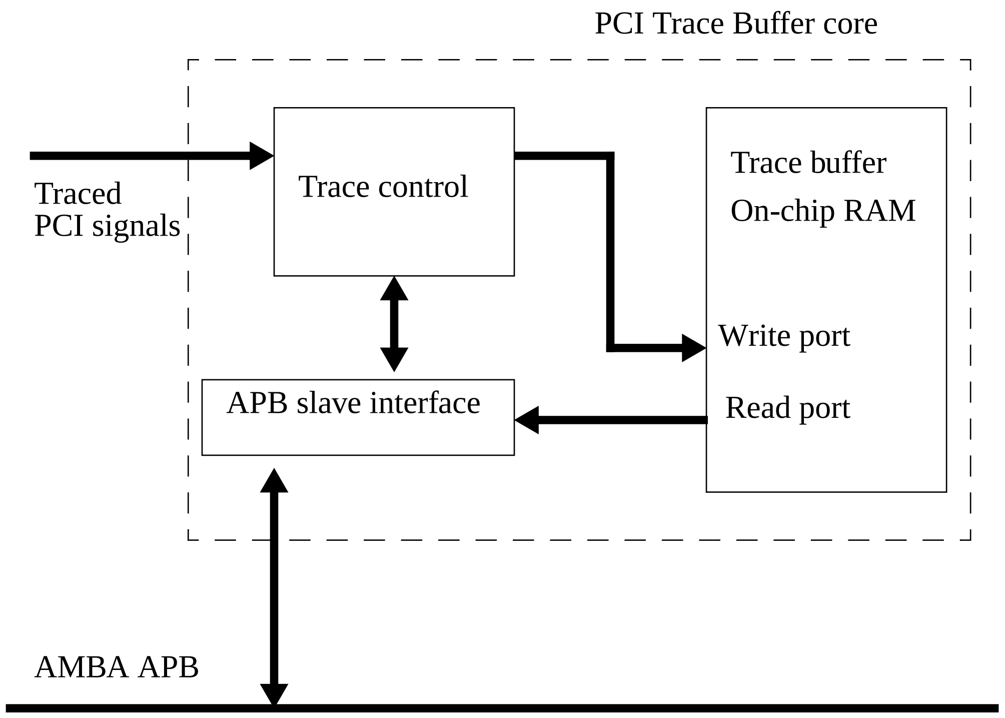
Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	PCI	Signals, component	PCI signals and component declaration

81 PCITRACE - PCI Trace Buffer

81.1 Overview

The PCI Trace Buffer core consists of a circular trace buffer and a control module. When armed, the core stores the traced PCI signals in the circular buffer until a trigger condition occurs. A trigger condition will freeze the buffer, and the traced data can then be read out via an APB interface.

The depth of the trace buffer is configurable through VHDL generics.



81.2 Operation

81.2.1 Clocking

The core uses two clocks: PCI clock and the AMBA clock (system clock). The traced signals are sampled with the PCI clock, while the control unit and the APB interface use the system clock. The PCI clock and system clock does not need to be synchronized or have the same frequency.

81.2.2 Traced PCI signals

The core samples the 32-bit PCI address as well as the PCI control signals listed below. The number given in parentheses is the bit number in the *PCI Control Signals Pattern*, *PCI Control Signals Mask*, and *Traced PCI Control Signals* APB registers that represent the corresponding signal.

C/BE#(3:0), PAR(4), SERR#(5), PERR#(6), LOCK#(7), STOP#(8), GNT#(9), DEVSEL#(10), IRDY#(11), TRDY#(12), FRAME#(13), IDSEL(14), RST#(15)

81.2.3 Triggering

The core can be programmed to trigger on any combination of the PCI input signals by setting up the desired pattern in the *PCI Address Pattern* and *PCI Control Signals Pattern* registers. Certain signals can be programmed to be ignored when comparing against the pattern. This is done by clearing the corresponding bits in the *PCI Address Mask* and *PCI Control Signals Mask* registers.

The core also offers the possibility to program how many times the trigger condition need to occur as well as how many samples the core should take after the trigger condition occurred (for the final time). The number of times the trigger condition need to occur is programmed by writing to the *TMCNTCOMP* field in the *Trigger Match Counter* register. The number of samples that should be taken is programmed by writing to the *Trigger Stop Counter* register.

To start sampling, the core needs to be armed. This is done by writing to the *Arm/Busy* register. The core remains armed until the trigger match counter and trigger stop counter reach zero, or until a system reset occurs.

The index (internal trace buffer address) of the last sample can be found be reading the *TADDR* field of the *Trigger Match Counter* register and subtracting one. The *TMCNT* field of the same register contains the actual value of the trigger stop counter, which equals the number of more times the trigger condition need to occur.

The traced PCI signals can be read from the *Traced PCI Address* register(s) and the *Traced PCI Control Signals* register(s).

81.3 Registers

The core is controlled through registers mapped in APB address space. The traced signals are accessible through APB registers as well. A 64 K address space is needed to fit both the control registers and traced signals.

*Table 1119.*APB address mapping

APB address offset	Registers
0x0000	PCI Address Mask register
0x0004	PCI Control Signals Mask register
0x0008	PCI Address Pattern register
0x000C	PCI Control Signals Pattern register
0x0010	Trigger Stop Counter register
0x0014	Trigger Arm/Busy register
0x0018	Capability register
0x001C	Trigger Match Counter register
0x8000 - 0xBFFC*	Traced PCI Address register(s)
0xC000 - 0xFFFC*	Traced PCI Control Signals register(s)

* The number of implemented registers depend on the TDEPTH field in the Capability Register. Number of registers = $2^{\wedge TDEPTH}$

Table 1120. PCI Address Mask register

31 0
ADMASK
0x00000000

31: 0	PCI address mask (ADMASK) - Mask to select which bits of the PCI address that are used when comparing against the trigger pattern. If a bit is set to ‘1’ then the corresponding bit in the PCI address must match the corresponding bit in the trigger pattern. If a bit is set to ‘0’ then the value of that bit in the PCI address does not matter.	rw

Table 1121. PCI Control Signals Mask register

31	15 0
RESERVED	SIGMASK

Table 1121. PCI Control Signals Mask register

N/A	0x0000

31: 16	RESERVED	r
15: 0	PCI signals mask (SIGMASK) - Mask to select which signals of the PCI control signal inputs that are used when comparing against the trigger pattern. If a bit is set to ‘1’ then the corresponding input must match the corresponding bit in the trigger pattern. If a bit is set to ‘0’ then the value of the corresponding signal does not matter. Which PCI control signal input that corresponds to which bit is explained above.	rw

Table 1122. PCI Address Pattern register

31 0
ADPATTERN
0x00000000

31: 0	PCI address pattern (ADPATTERN) - Trigger pattern for the PCI address. Used together with the PCI address mask to compare against the PCI address input when deciding whether or not the trigger condition for the PCI address is fulfilled.	rw

Table 1123. PCI Control Signals Pattern register

31 16	15 0
RESERVED	SIGPATTERN
N/A	0x0000

31: 16	RESERVED	r
15: 0	PCI control signals pattern (SIGPATTERN) - Trigger pattern for the PCI control signals. Used together with the PCI control signals mask to compare against the PCI control signal inputs when deciding whether or not the trigger condition is fulfilled. Which PCI control signal input that corresponds to which bit is explained above.	rw

Table 1124. Trigger Stop Counter register

31 X+1	X-1 0
RESERVED	TSCNT
N/A	0x0..0

31:X+1	RESERVED	r
X-1:0	Trigger stop counter value (TSCNT) - Used to set the number of samples the core should take after the trigger condition is fulfilled before sampling is stopped. Depending on the TMCNT field in the *Trigger Match Counter* register, the trigger condition might need to occur several times before TSCNT is used.	rw

X = The value of the TDEPTH field in the *Capability* register.

Table 1125. Trigger Arm/Busy register

31 0
ARM/BUSY
0x00000000

31: 0	Trigger arm/busy (ARM/BUSY) - When this register is written, no matter what value is written, the trigger is armed. This register is always read either 0x00000000 or 0x00000001. All zeroes means that the trigger is not armed and the core is not sampling the PCI inputs. 0x00000001 means that the trigger is armed and the core is sampling the PCI inputs.	rw

Table 1126. Capability register

31 4	3 0
RESERVED	TDEPTH
N/A	N/A

Table 1126. Capability register

31: 4	RESERVED	r
15: 0	Trace buffer depth (TDEPTH) - Shows the number of address bits used for the core's internal trace buffer. The number of possible samples = 2^^{TDEPTH}. TDEPTH = value of VHDL generic *depth*.	r

Table 1127. Trigger Match Counter register

31 … X+16	X+15 … 16	15 … 8	7 … 0
RESERVED	TADDR	TMCNT	TMCNTCOMP
N/A	0x0..0	0x00	0x00

31:X+16	RESERVED	r
X+15:16	Trace buffer address (TADDR) - Shows current trace buffer address, which is the same as the address where the next sample will be stored.	r
15: 8	Trigger match counter value (TMCNT) - Actual value of the trigger match counter. Shows how many more times the trigger condition need to occur before sampling is stopped.	r
7: 0	Trigger match counter compare value (TMCNTCOMP) - Used to set the number of times the trigger condition need to be fulfilled before the sampling is stopped. Depending on the value of the TSCNT field in the Trigger Stop Counter Register, the sampling might not be stopped immediately after the trigger condition is fulfilled the final time.	rw

X = The value of the TDEPTH field in the *Capability* register.

Table 1128. Traced PCI Address register(s)

31 … 0
TADDR
N/A

31: 0	.Traced PCI address (TADDR) - APB address bits TDEPTH+1:2 are used to address the trace buffer, and the traced PCI address saved at that address in the trace buffer can be read from the corresponding APB register. The number of implemented registers depend on the TDEPTH field in the *Capability* register. Number of registers = 2$^{\wedge TDEPTH}$	rw

Table 1129. Traced PCI Control Signals register(s)

31 … 16	15 … 0
RESERVED	TSIGNALS
N/A	N/A

15: 0	Traced PCI control signals (TSIGNALS) - APB address bits TDEPTH+1:2 are used to address the trace buffer, and the traced PCI control signals saved at that address can be read from the corresponding APB register. The number of implemented registers depend on the TDEPTH field in the *Capability* register. Number of registers = 2$^{\wedge TDEPTH}$	r

81.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x015. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

81.5 Configuration options

Table 1130 shows the configuration options of the core (VHDL generics).

*Table 1130.*Configuration options

Generic	Function	Allowed range	Default
depth	Used to set the number of samples stored in the internal buffers. Number of samples = $2^{\wedge depth}$	6 - 12	8
iregs	Used to add registers on PCI input signals before comparing and sampling. 0 = No registers. 1 = Registers.	0 - 1	1
memtech	Memory technology	0 - NTECH	0
pindex	APB slave index	0 - NAPBSLV - 1	0
paddr	ADDR field of the APB BAR.	0 - 0xFFF	0
pmask	MASK field of the APB BAR.	0 - 0xF00	0xF00

81.6 Signal descriptions

Table 1131 shows the interface signals of the core (VHDL ports).

*Table 1131.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	System clock	-
PCICLK	N/A	Input	PCI clock	-
PCII	N/A	Input	PCI input signals *	
APBI	**	Input	APB slave input signals	-
APBO	**	Output	APB slave output signals	-

* See PCI specification

** See GRLIB IP Library users manual

81.7 Library dependencies

Table 1132 shows libraries used when instantiating the core (VHDL libraries).

*Table 1132.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	PCI	Signals, component	Component declaration, PCI signals

81.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee, grlib, gaisler;
use ieee.std_logic_1164.all;
use grlib.amba.all;
use gaisler.pci.all;

entity pcitrace_ex is
  generic (
    memtech => memtech
    ... -- other generics
  );
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;
    pciclk : in std_ulogic;
    pcii : in pci_in_type;

    ... -- other signals
  );
end;

architecture rtl of pcitrace_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

begin

  -- PCI Trace Buffer core
  logan0 : logan
    generic map (
      depth => 8,
      iregs => 1,
      memtech => memtech,
      pindex => 3,
      paddr => 3,
      pmask => 16#F00#
    )
    port map (
      rst => rstn,
      clk => clk,
      pciclk => pciclk,
      pcii => pcii,
      apbi => apbi,
      apbo => apbo(3)
    );

  -- Other cores
  ...

end;
```

82 PHY - Ethernet PHY simulation model

82.1 Overview

The PHY is a simulation model of an IEEE 802.3 compliant Ethernet PHY. It provides a complete MII and GMII interface with the basic, extended status and extended capability registers accessible through the management interface (MDIO). Not all of the functionality is implemented and many of the register bits are therefore only writable and readable but do not have any effect. Currently only the loopback is supported.

82.2 Operation

The PHY simulation model was designed to make it possible to perform simple simulations on the GRETH and GRETH_GBIT cores in GRLIB. It provides the complete set of basic, extended capability and extended status registers through the MII management interface (MDIO) and a loopback mode for data transfers. Figure 1 shows a block diagram of a typical connection.

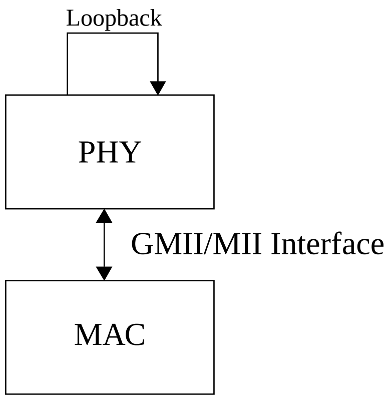


Figure 259. Block diagram of the PHY simulation model connected to a MAC.

The PHY model provides the complete GMII and MII interface as defined by the IEEE 802.3 standard. The model can be used in any of the following modes: 10 Mbit half- or full duplex, 100 Mbit half- or full-duplex and 1000 Mbit half- or full-duplex. This support refers only to the configuration settings available through the MDIO registers. Since the datapath implementation is loopback no collisions will ever be seen on the network and operation will essentially be full-duplex all the time. In loopback mode, rx_clk and tx_clk are identical in both frequency and phase and are driven by the PHY when not in gigabit mode. In gigabit mode the gtx_clk input is used as the transmitter clock and it also drives rx_clk.

When not configured to loopback mode the PHY just sits idle and ignores transmitted packet and does not insert any activity on the receive interface. Clocks are still generated but in this case rx_clk and tx_clk does have the same frequency but not the same phase when not in gigabit mode.

A simple auto-negotiation function is provided and the supported and advertised modes are set through vhdl generics. The generic values will be directly reflected in the reset values and read-only values of all corresponding MII management registers.

82.3 Configuration options

Table 1133 shows the configuration options of the model (VHDL generics).

*Table 1133.*Configuration options

Generic	Function	Allowed range	Default
address	Address of the PHY on the MII management interface	0 - 31	0
extended_regs	Include extended register capability	0 - 1	1
aneg	Enable auto-negotiation functionality	0 - 1	1
base100_t4	Enable support for 100Base-T4	0 - 1	0
base100_x_fd	Enable support for 100Base-X full-duplex	0 - 1	1
base100_x_hd	Enable support for 100Base-X half-duplex	0 - 1	1
fd_10	Enable support for 10Base-T full-duplex	0 - 1	1
hd_10	Enable support for 10Base-T half-duplex	0 - 1	1
base100_t2_fd	Enable support for 100Base-T2 full-duplex	0 - 1	1
base100_t2_hd	Enable support for 100Base-T2 half-duplex	0 - 1	1
base1000_x_fd	Enable support for 1000Base-X full-duplex	0 - 1	0
base1000_x_hd	Enable support for 1000Base-X half-duplex	0 - 1	0
base1000_t_fd	Enable support for 1000Base-T full-duplex	0 - 1	1
base1000_t_hd	Enable support for 1000Base-T half-duplex	0 - 1	1
rmii	Set PHY in RMII mode	0 - 1	0

82.4 Signal descriptions

Table 1134 shows the interface signals of the model (VHDL ports).

*Table 1134.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	-	Input	Reset	Low
MDIO	-	Input/ Output	Data signal for the management interface (Currently not used)	-
TX_CLK	-	Output	Transmitter clock	-
RX_CLK	-	Output	Receiver clock	-
RXD	-	Output	Receiver data	-
RX_DV	-	Output	Receiver data valid	High
RX_ER	-	Output	Receiver error	High
RX_COL	-	Output	Collision	High
RX_CRS	-	Output	Carrier sense	High
TXD	-	Input	Transmitter data	-
TX_EN	-	Input	Transmitter enable	High
TX_ER	-	Input	Transmitter error	High
MDC	-	Input	Management interface clock (Currently not used)	-
GTX_CLK	-	Input	Gigabit transmitter clock	-

see the IEEE 802.3 standard for a description of how the signals are used.

82.5 Library dependencies

Table 1135 shows the libraries used when instantiating the model (VHDL libraries).

*Table 1135.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	SIM	Component	Component declaration

82.6 Instantiation

This example shows how the model can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library gaisler;
use gaisler.sim.all;

entity phy_ex is
  port (
rst : std_ulogic;
clk : std_ulogic;
    );
end;

architecture rtl of phy_ex is

  -- Signals

 signal etx_clk   : std_logic;
 signal gtx_clk   : std_logic;
 signal erx_clk   : std_logic;
 signal erxd      : std_logic_vector(7 downto 0);
 signal erx_dv    : std_logic;
 signal erx_er    : std_logic;
 signal erx_col   : std_logic;
 signal erx_crs   : std_logic;
 signal etxd      : std_logic_vector(7 downto 0);
 signal etx_en    : std_logic;
 signal etx_er    : std_logic;
 signal emdc      : std_logic;

begin

  -- Other components are instantiated here
  ...


  -- PHY model
  phy0 : phy
  generic map (address => 1)
  port map(resetn => rst, mdio => open, tx_clk => etx_clk, rx_clk => erx_clk, rxd => erxd,
  rx_dv => erx_dv, rx_er => erx_er,
  rx_col => erx_col, rx_crs => erx_crs, txd => etxd, tx_en => etx_en,
  tx_er => etx_er, mdc => emdc, gtx_clk => gtx_clk);
end;
```

83 REGFILE_3P 3-port RAM generator (2 read, 1 write)

83.1 Overview

The 3-port register file has two read ports and one write port. Each port has a separate address and data bus. All inputs are latched on the rising edge of clk. The read data appears on dataout directly after the clk rising edge. Note: on most technologies, the register file is implemented with two 2-port RAMs with combined write ports. Address width, data width and target technology is parametrizable through generics.

Write-through is supported if the function *syncram_2p_write_through(tech)* returns 1 for the target technology.

83.2 Configuration options

Table 1136 shows the configuration options of the core (VHDL generics).

*Table 1136.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table 1137	-
dbits	Data width	see table 1137l	-
wrfst	Write-first (write-through). Only applicable to inferred technol-ogy	0 - 1	0
numregs	Not used		

Table 1137 shows the supported technologies for the core.

*Table 1137.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
axcel / axdsp	Actel AX/RTAX & RTAX-DSP	RAM64K36	2 - 12	unlimited
altera	All Altera devices	altsyncram	unlimited	unlimited
ihp25	IHP 0.25	flip-flops	unlimited	unlimited
inferred	Behavioural description	synthesis tool dependent		
rhumc	Rad-hard UMC 0.18	flip-flops	unlimited	unlimited
virtex	Xilinx Virtex, Virtex-E, Spartan-2	RAMB4_Sn	2 - 10	unlimited
virtex2	Xilinx Virtex2, Spartan3, Virtex4	RAMB16_Sn	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	dp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss2_64x32cm4sw0 hdss2_128x32cm4sw0 hdss2_256x32cm4sw0 hdss2_512x32cm4sw0	6 - 9	32
eclipse	Aeroflex/Quicklogic FPGA	RAM128x18_25um RAM256X9_25um RAM512X4_25um RAM1024X2_25um	2 - 10	unlimited
easic90	eASIC 90 nm Nextreme	eram	2 - 12	unlimited

83.3 Signal descriptions

Table 1138 shows the interface signals of the core (VHDL ports).

*Table 1138.*Signal descriptions

Signal name	Field	Type	Function	Active
WCLK	N/A	Input	Write port clock	
WADDR	N/A	Input	Write address	
WDATA	N/A	Input	Write data	
WE	N/A	Input	Write enable	High
RCLK	N/A	Input	Read ports clock	-
RADDR1	N/A	Input	Read port1 address	-
RE1	N/A	Input	Read port1 enable	High
RDATA1	N/A	Output	Read port1 data	-
RADDR2	N/A	Input	Read port2 address	-
RE2	N/A	Input	Read port2 enable	High
RDATA2	N/A	Output	Read port2 data	-

83.4 Library dependencies

Table 1139 shows libraries used when instantiating the core (VHDL libraries).

*Table 1139.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology contants

83.5 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

component regfile_3p
  generic (tech : integer := 0; abits : integer := 6; dbits : integer := 8;
           wrfst : integer := 0; numregs : integer := 64);
  port (
    wclk   : in  std_ulogic;
    waddr  : in  std_logic_vector((abits -1) downto 0);
    wdata  : in  std_logic_vector((dbits -1) downto 0);
    we     : in  std_ulogic;
    rclk   : in  std_ulogic;
    raddr1 : in  std_logic_vector((abits -1) downto 0);
    re1    : in  std_ulogic;
    rdata1 : out std_logic_vector((dbits -1) downto 0);
    raddr2 : in  std_logic_vector((abits -1) downto 0);
    re2    : in  std_ulogic;
    rdata2 : out std_logic_vector((dbits -1) downto 0)
  );
  end component;
```

84 RSTGEN - Reset generation

84.1 Overview

The RSTGEN reset generator implements input reset signal synchronization with glitch filtering and generates the internal reset signal. The input reset signal can be asynchronous.

84.2 Operation

The reset generator latches the value of the clock lock signal on each rising edge of the clock. The lock signal serves as input to a five-bit shift register. The three most significant bits of this shift register are clocked into the reset output register. The reset signal to the system is high when both the reset output register and the reset input signal are high. Since the output register depends on the system clock the active low reset output from the core will go high synchronously to the system clock. The raw reset output does not depend on the system clock or clock lock signal and is polarity adjusted to be active low.

The VHDL generic *syncrst* determines how the core resets its shift register and the reset output register. When *syncrst* is set to 1 the core's shift register will have an synchronous reset and no reset signal will be connected to the output reset register, see figure 260. Note that the core's reset output signal will always go low when the input reset signal is activated.

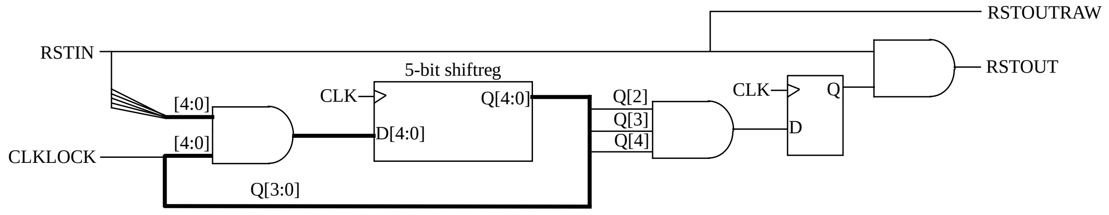


Figure 260. Reset generator with VHDL generic syncrst set to 1

When *syncrst* is 0 the shift register will be reset asynchronously together with the reset output register. Figure 261 shows the reset generator when scan test support is disabled. The shift register reset will be connected to the core's normal reset input and the test reset input will be unused. When scan test support is enabled, the core's test reset input can be connected to the reset input on both registers. The reset signal to use for the registers is selected with the test enable input, see figure 262.

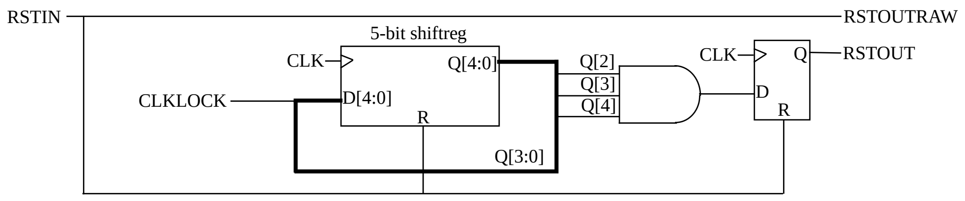


Figure 261. Reset generator with VHDL generic syncrst set to 0 and scan test disabled

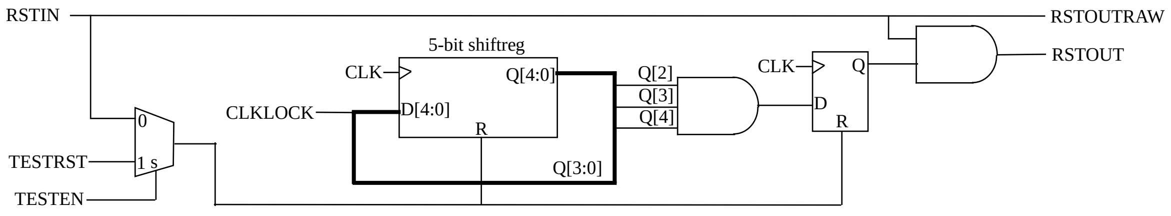


Figure 262. Reset generator with VHDL generic syncrst set to 0 and scan test enabled

84.3 Configuration options

Table 1140 shows the configuration options of the core (VHDL generics).

*Table 1140.*Configuration options

Generic name	Function	Allowed range	Default
acthigh	Set to 1 if reset input is active high. The core outputs an active low reset.		0
syncrst	When this generic is set to 1 the reset signal will use a synchronous reset to reset the filter registers. When this generic is set to 1 the TESTRST and TESTEN inputs will not be used.		0
scanen	Setting this generic to 1 enables scan test support. This connects the TESTRST input via a multiplexer so that the TESTRST and TESTEN signals can be used to asynchronously reset the core's registers. This also requires that the generic syncrst is set to 1.		0

84.4 Signal descriptions

Table 1141 shows the interface signals of the core (VHDL ports).

*Table 1141.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTIN	N/A	Input	Reset	-
CLK	N/A	Input	Clock	-
CLKLOCK	N/A	Input	Clock lock	High
RSTOUT	N/A	Output	Filtered reset	Low
RSTOUTRAW	N/A	Output	Raw reset	Low
TESTRST	N/A	Input	Test reset	-
TESTEN	N/A	Input	Test enable	High

84.5 Library dependencies

Table 1142 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1142.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	MISC	Component	Component definition

84.6 Instantiation

This example shows how the core can be instantiated together with the GRLIB clock generator.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;
library gaisler;
use gaisler.misc.all;

entity rstgen_ex is
  port (
    resetn  : in  std_ulogic;
    clk     : in  std_ulogic; -- 50 MHz main clock
    pllref  : in  std_ulogic;
    testrst : in std_ulogic;
    testen  : in std_ulogic
  );
end;

architecture example of rstgen_ex is

signal lclk, clkm, rstn, rstraw, sdclkl, clk50: std_ulogic;
signal cgi   : clkgen_in_type;
signal cgo   : clkgen_out_type;

begin
  cgi.pllctrl <= "00"; cgi.pllrst <= rstraw;
  pllref_pad : clkpad generic map (tech => padtech) port map (pllref, cgi.pllref);
  clk_pad : clkpad generic map (tech => padtech) port map (clk, lclk);
  clkgen0 : clkgen  -- clock generator
    generic map (clktech, CFG_CLKMUL, CFG_CLKDIV, CFG_MCTRL_SDEN,
                 CFG_CLK_NOFB, 0, 0, 0, BOARD_FREQ)
    port map (lclk, lclk, clkm, open, open, sdclkl, open, cgi, cgo, open, clk50);
  sdclk_pad : outpad generic map (tech => padtech, slew => 1, strength => 24)
    port map (sdclk, sdclkl);

  resetn_pad : inpad generic map (tech => padtech) port map (resetn, rst);

  rst0 : rstgen  -- reset generator
    generic map (acthigh => 0, syncrst => 0, scanen => 1)
    port map (rst, clkm, cgo.clklock, rstn, rstraw, testrst, testen);
end;
```

85 GR(2^4)(68, 60, 8, T=1) - QEC/QED error correction code encoder/decoder

85.1 Overview

The gf4_e1 VHDL package provides functions for encoding and decoding a Bose Chaudhuri Hocquenghem (BCH) type of code. It is a Quad Error Correction/Quad Error Detection (QEC/QED) code.

The data symbols are 4-bit wide, represented as GF(2^4). The has the capability to detect and correct a single symbol error anywhere in the codeword. The data is represented as 60 bits and the checksum is represented as 8 bits, and the code can correct up to four bit errors when located in the same nibble.

85.2 Code

The code has the following definition:

- there are 4 bits per symbol;
- there are 17 symbols per codeword, of which 2 symbols represent the checksum;
- the code is systematic;
- the code can correct one symbol error per codeword;
- the field polynomial is

$$f(x) = x^4 + x + 1$$

- all multiplications are performed as Galois Field multiplications over the above field polynomial
- all additions/subtractions are performed as Galois Field additions (i.e. bitwise exclusive-or)

85.3 Encoding

- a codeword is defined as 17 symbols:

  $[c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}, c_{16}]$

  where c_0 to c_{14} represent information symbols and c_{15} to c_{16} represent check symbols.
- c_{15} is calculated as follows

$$c_{15} = \sum_{0}^{14} (k_i \times c_i)$$

- c_{16} is calculated as follows

$$c_{16} = \sum_{0}^{14} c_i$$

- where the constant vector k is defined as:

  k_0=0xF, k_1=0xE, ..., k_{14}=0x1 (one can assume k_{15}=0x1 and k_{16}=0x1 for correction purposes)

85.4 Decoding

- the corrupt codeword is defined as 17 symbols:

  $[r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}]$
- the corrupt codeword can also be defined as 17 uncorrupt symbols and an error:

  $[c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}, c_{16}] + [e_x]$

  where the error is defined as e_x, e being the unknown magnitude and
  x being the unknown index position in the codeword

- recalculated checksum rc_0 is calculated as follows (k_i is as defined above, x being the unknown index)

$$rc_0 = \sum_0^{14}(k_i \times r_i) = \sum_0^{14}(k_i \times c_i) + (k_x \times e_x)$$

- recalculated rc_1 is calculated as follows

$$rc_1 = \sum_0^{14} r_i = \sum_0^{14} c_i + e_x$$

- syndrome s_0 is calculated as follows

$$s_0 = rc_0 + r_{15} = \sum_0^{14}(k_i \times r_i) + \sum_0^{14}(k_i \times c_i) = k_x \times e_x$$

- syndrome s_1 is calculated as follows, which gives the magnitude (not applicable to c_{15} and c_{16})

$$s_1 = rc_1 + r_{16} = \sum_0^{14} r_i + \sum_0^{14} c_i = e_x$$

- in case s_0 and s_1 are both non-zero, to located the error in range c_0 to c_{14}, multiply error magnitude e_x with each element of the constant vector defined above:

$$k_ie_x = k_i \times s_1 = k_i \times e_x \qquad i = [0,14]$$

- search the resulting vector to find the element matching syndrome s_0, the resulting index i points to the error location (applicable only to i in [0, 14])

$$k_ie_x \Leftrightarrow k_i \times e_x$$

- finally perform the correction (applicable only to i in [0, 14])

$$c_i = r_i - s_1 = r_i - e_x = (c_i - e_x) \times e_x = c_i - e_x + e_x = c_i$$

- when s_0 is zero and s_1 is non-zero, the error is located in checksum r_{15}, no correction is necessary
- when s_1 is zero and s_0 is non-zero, the error is located in checksum r_{16}, no correction is necessary
- when s_0 and s_1 are both zero, no error has been detected, no correction is necessary

85.5 Capability

The decoder has the following capabilities. It detects and corrects up to four bit errors in the same nibble. The described errors can be located anywhere in the codeword.

85.6 Operation

85.6.1 Encoder

The encoder is defined by the gf4_60_8_encode function. The function is called with the 60-bit wide data that should be encoded, and returns a 68-bit wide codeword of which bits 67 downto 8 represent the data and bits 7 downto 0 represent the checksum.

85.6.2 Decoder

The decoder is defined by the gf4_60_8_decode function.

The gf4_60_8_decode function calculates the syndromes, calculates the error magnitude and the error location, and returns a bit indicating whether an error has been detected and corrected, and the corrected data.

The function is called with a 68-bit wide codeword of which bits 67 downto 8 represent the data and bits 7 downto 0 represent the checksum. It returns the record type gf4_60_8_type, containing the 60-bit wide corrected data and an indication if an error was detected and corrected over the complete codeword.

85.7 Type descriptions

Table 1143 shows the type declarations used by the functions in the package (VHDL types).

*Table 1143.*Type declarations

Name	Field	Type	Function	Active
gf4_60_8_type	cerr	Std_Logic	error corrected	
	data	Std_Logic_Vector(59 downto 0)	data	

85.8 Library dependencies

Table 1144 shows the libraries used when instantiating the functions in the package (VHDL libraries).

*Table 1144.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	StdLib	All	Common VHDL functions

85.9 Instantiation

This example shows how the functions in the package can be instantiated. Note that all input and outputs are synchronized to remove any timing constraints for pads in an example design. Timing analysis can then be made purely for the register-to-register paths.

```
library  IEEE;
use      IEEE.Std_Logic_1164.all;

entity gf4_60_8_encode_sync is
   port(
      clk:        in       std_ulogic;
      data:       in       std_logic_vector(59 downto 0);
      codeword:   out      std_logic_vector(67 downto 0));
end entity gf4_60_8_encode_sync;

library  grlib;
use      grlib.gf4_e1.all;

architecture rtl of gf4_60_8_encode_sync is
   signal   int_data:      std_logic_vector(59 downto 0);
   signal   int_codeword:  std_logic_vector(67 downto 0);
begin
   process(clk)
   begin
      if rising_edge(clk) then
          codeword      <= int_codeword;
          int_codeword  <= gf4_60_8_encode(int_data);
          int_data      <= data;
      end if;
   end process;
end architecture;
```

```
library  IEEE;
use      IEEE.Std_Logic_1164.all;

entity gf4_60_8_decode_sync is
   port(
         clk:        in    std_ulogic;
         codeword:   in    std_logic_vector(67 downto 0);
         cerr:       out   std_ulogic;
         data:       out   std_logic_vector(59 downto 0));
end entity gf4_60_8_decode_sync;

library  grlib;
use      grlib.gf4_e1.all;

architecture rtl of gf4_60_8_decode_sync is
   signal   int_codeword:  std_logic_vector(67 downto 0);
   signal   int_result:    gf4_60_8_type;
begin
   process(clk)
   begin
      if rising_edge(clk) then
         cerr           <= int_result.cerr;
         data           <= int_result.data;
         int_result     <= gf4_60_8_decode(int_codeword);
         int_codeword   <= codeword;
      end if;
   end process;
end architecture;
```

86 RS(24, 16, 8, E=1) - Reed-Solomon encoder/decoder

86.1 Overview

The rs_gf4_e1 VHDL package provides functions for encoding and decoding data with a Reed-Solomon code. It also provides a data type storing intermediate results from the functions.

The Reed-Solomon data symbols are 4-bit wide, represented as GF(2^4). The Reed-Solomon code is a shortened RS(15, 13, 2) code, represented as RS(6, 4, 2). It has the capability to detect and correct a single symbol error anywhere in the codeword. The data is represented as 16 bits and the checksum is represented as 8 bits, and the code can correct 4-bit errors when located in the same nibble.

86.2 Capability

The Reed-Solomon decoder has the following capabilities. The described errors can be located anywhere in the codeword.

It detects and corrects any single bit error.

It detects 63% of all double bit errors and reports them as multiple bit errors.

It detects 27% of all double bit errors and reports them (incorrectly) as single bit errors.

It detects 63,5% of all triple bit errors and reports them as multiple bit errors.

It detects 36% of all triple bit errors and reports them (incorrectly) as single bit errors.

It does not detect 0,5% of all triple bit errors and reports them (incorrectly) as without errors.

It detects 63,5% of all quadruple bit errors and reports them as multiple bit errors.

It detects 36% of all quadruple bit errors and reports them (incorrectly) as single bit errors.

It does not detect 0,5% of all quadruple bit errors and reports them (incorrectly) as without error.

It detects and corrects up to four bit errors in the same nibble.

86.3 Operation

86.3.1 Encoder

The encoder is defined by the rs_16_8_encode function. The function is called with the 16-bit wide data that should be encoded, and returns 24-bit wide codeword of which bits 0 to 15 represent the data and bits 16 to 23 represent the checksum.

86.3.2 Decoder

The decoder is defined by the rs_16_8_check, rs_16_8_precorrect and rs_16_8_correct functions. The decoder has been split in three functions to facilitate pipelining, with each function being fairly balanced with respect to the depth of the resulting combinatorial logic.

The rs_16_8_check function calculates the syndrome and returns a bit indicating whether an error has been detected. Note that it can be any type of error: correctable or uncorrectable. The function is called with a 24-bit wide codeword of which bits 0 to 15 represent the data and bits 16 to 23 represent the checksum. It returns the record type rs_16_8_type, containing the 16-bit wide data to be corrected, the syndrome and an indication if an error was detected over the complete codeword.

The rs_16_8_precorrect function is called with the intermediate result from the rs_16_8_check function. The input is the record type rs_16_8_type. It returns the record type rs_16_8_type, containing the 16-bit wide data to be corrected, the syndrome, intermediate data and an indication if an error was detected over the complete codeword.

The rs_16_8_correct function is called with the intermediate result from the rs_16_8_precorrect function. The input is the record type rs_16_8_type. It returns the record type rs_16_8_type, containing the corrected 16-bit wide data, an indication if the error was correctable or non-correctable over the complete codeword, and the union of the two.

To pipeline the decoder, the rs_16_8_check function should be called in the first stage and the intermediated result should be stored. Note that the intermediate result contains the input data required for the correction in the next stage. The rs_16_8_precorrect function should be called in the second stage. The rs_16_8_correct function should be called in the third stage.

86.4 Type descriptions

Table 1145 shows the type declarations used by the functions in the package (VHDL types).

*Table 1145.*Type declarations

Name	Field	Type	Function	Active
rs_16_8_type	err	Std_Logic	error detected	High
	cerr	Std_Logic	error corrected	
	merr	Std_Logic	errors uncorrected	
	data	Std_Logic_Vector(0 to 15)	data	
	s_1	Std_Logic_Vector(0 to 3)	-	-
	s_2	Std_Logic_Vector(0 to 3)	-	-
	elp_3_1	Std_Logic_Vector(0 to 3)	-	-

86.5 Library dependencies

Table 1146 shows the libraries used when instantiating the functions in the package (VHDL libraries).

*Table 1146.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	StdLib	All	Common VHDL functions

86.6 Instantiation

This example shows how the functions in the package can be instantiated.

The example design shows a codec for which the decoder is pipelined, with the error flag output one clock cycle earlier than the corrected data. Note that all input and outputs are synchronised to remove any timing constraints for pads in an example design. Timing analysis can be made purely for the register-to-register paths.

```
library  IEEE;
use      IEEE.Std_Logic_1164.all;

entity rs_gf4_16_8_codec is
   port(
      clk:        in    Std_Logic;
      din:        in    Std_Logic_Vector(0 to 15);       -- encoder input
      cout:       out   Std_Logic_Vector(0 to 23);       -- encoder output

      cin:        in    Std_Logic_Vector(0 to 23);       -- decoder input
      terr:       out   Std_Logic;                       -- intermediate error

      dout:       out   Std_Logic_Vector(0 to 15);       -- decoder output
      err:        out   Std_Logic;                       -- error detected
      cerr:       out   Std_Logic;                       -- error corrected
      merr:       out   Std_Logic);                      -- errors uncorrected
end entity;

library  grlib;
use      grlib.rs_gf4_e1.all;

architecture rtl of rs_gf4_16_8_codec is
   signal   s_din:      Std_Logic_Vector(0 to 15);
   signal   s_cout:     Std_Logic_Vector(0 to 23);

   signal   s_cin:      Std_Logic_Vector(0 to 23);
   signal   s_dout:     Std_Logic_Vector(0 to 15);
   signal   s_err:      Std_Logic;
   signal   s_cerr:     Std_Logic;
   signal   s_merr:     Std_Logic;

   signal   check:      rs_16_8_type;                    -- intermediate
   signal   precorr:    rs_16_8_type;
   signal   corr:       rs_16_8_type;
begin
   SyncronizeInput: process(clk)
   begin
      if Rising_Edge(clk) then
         s_din       <= din;
         s_cin       <= cin;
      end if;
   end process;

   SyncronizeOutput: process(clk)
   begin
      if Rising_Edge(clk) then
         cout        <= s_cout;
         err         <= corr.err;
         cerr        <= corr.cerr;
         merr        <= corr.merr;
         dout        <= corr.data;
         terr        <= check.err;
      end if;
   end process;

   encoder: process(clk)
   begin
      if Rising_Edge(clk) then
         s_cout      <= rs_16_8_encode(s_din);
      end if;
   end process;

   decoder: process(clk)
   begin
      if Rising_Edge(clk) then
         corr        <= rs_16_8_correct(precorr);        -- third phase
         precorr     <= rs_16_8_precorrect(check);       -- second phase
         check       <= rs_16_8_check(s_cin);            -- first phase
      end if;
   end process;
end architecture rtl;
```

87 RS(48, 32, 16, E=1+1) - Reed-Solomon encoder/decoder - interleaved

87.1 Overview

The rs_gf4_e1 VHDL package provides functions for encoding and decoding data with a Reed-Solomon code. It also provides a data type storing intermediate results from the functions.

The Reed-Solomon data symbols are 4-bit wide, represented as GF(2^4). The Reed-Solomon code is a shortened RS(15, 13, 2) code, represented as RS(6, 4, 2). It has the capability to detect and correct a single symbol error anywhere in the codeword. The data is represented as 16 bits and the checksum is represented as 8 bits, and the code can correct 4-bit errors when located in the same nibble.

The gf4_32_16 functions provide an interleaved RS(6, 4, 2) where the data is represented as 32 bits and the checksum is represented as 16 bits, and the code can correct two 4-bit errors when each error is located in a nibble and not in the same original RS(6, 4, 2) codeword. The codewords are interleaved nibble-wise.

87.2 Capability

The Reed-Solomon decoder has the same capabilities as the original RS(6, 4, 2) code, but distributed per original RS(6, 4, 2) codeword.

87.3 Operation

87.3.1 Encoder

The encoder is defined by the rs_32_16_encode function. The function is called with the 32-bit wide data that should be encoded, and returns 48-bit wide codeword of which bits 0 to 31 represent the data and bits 32 to 47 represent the checksum.

87.3.2 Decoder

The decoder is defined by the rs_32_16_check, rs_32_16_precorrect and rs_32_16_correct functions. The decoder has been split in three functions to facilitate pipelining, with each function being fairly balanced with respect to the depth of the resulting combinatorial logic.

The rs_32_16_check function calculates the syndrome and returns a bit indicating whether an error has been detected. Note that it can be any type of error: correctable or uncorrectable. The function is called with a 48-bit wide codeword of which bits 0 to 31 represent the data and bits 32 to 47 represent the checksum. It returns the record type rs_32_16_type, containing the 32-bit wide data to be corrected, the syndrome and an indication if an error was detected over the two complete codewords.

The rs_32_16_precorrect function is called with the intermediate result from the rs_32_16_check function. The input is the record type rs_32_16_type. It returns the record type rs_32_16_type, containing the 32-bit wide data to be corrected, the syndrome, intermediate data and an indication if an error was detected over the two complete codewords.

The rs_32_16_correct function is called with the intermediate result from the rs_32_16_precorrect function. The input is the record type rs_32_16_type. It returns the record type rs_32_16_type, containing the corrected 32-bit wide data, an indication if the error was correctable or non-correctable over the complete two codewords, and the union of the two.

To pipeline the decoder, the rs_32_16_check function should be called in the first stage and the intermediated result should be stored. Note that the intermediate result contains the input data required for the correction in the next stage. The rs_32_16_precorrect function should be called in the second stage. The rs_32_16_correct function should be called in the third stage.

87.4 Type descriptions

Table 1147 shows the type declarations used by the functions in the package (VHDL types).

*Table 1147.*Type declarations

Name	Field	Type	Function	Active
rs_32_16_type	err	Std_Logic	error detected	High
	cerr	Std_Logic	error corrected	
	merr	Std_Logic	errors uncorrected	
	data	Std_Logic_Vector(0 to 31)	data	
	e_0	Std_Logic	-	-
	s_1_0	Std_Logic_Vector(0 to 3)	-	-
	s_2_0	Std_Logic_Vector(0 to 3)	-	-
	elp_3_1_0	Std_Logic_Vector(0 to 3)	-	-
	e_1	Std_Logic	-	-
	s_1_1	Std_Logic_Vector(0 to 3)	-	-
	s_2_1	Std_Logic_Vector(0 to 3)	-	-
	elp_3_1_1	Std_Logic_Vector(0 to 3)	-	-

87.5 Library dependencies

Table 1148 shows the libraries used when instantiating the functions in the package (VHDL libraries).

*Table 1148.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	StdLib	All	Common VHDL functions

88 RS(40, 32, 8, E=1) - Reed-Solomon encoder/decoder

88.1 Overview

The rs_gf4_e1 VHDL package provides functions for encoding and decoding data with a Reed-Solomon code. It also provides a data type storing intermediate results from the functions.

The Reed-Solomon data symbols are 4-bit wide, represented as GF(2^4). The Reed-Solomon code is a shortened RS(15, 13, 2) code, represented as RS(10, 8, 2). It has the capability to detect and correct a single symbol error anywhere in the codeword. The data is represented as 32 bits and the checksum is represented as 8 bits, and the code can correct 4-bit errors when located in the same nibble.

88.2 Operation

88.2.1 Encoder

The encoder is defined by the rs_32_8_encode function. The function is called with the 32-bit wide data that should be encoded, and returns 40-bit wide codeword of which bits 0 to 31 represent the data and bits 32 to 39 represent the checksum.

88.2.2 Decoder

The decoder is defined by the rs_32_8_check, rs_32_8_precorrect and rs_32_8_correct functions. The decoder has been split in three functions to facilitate pipelining, with each function being fairly balanced with respect to the depth of the resulting combinatorial logic.

The rs_32_8_check function calculates the syndrome and returns a bit indicating whether an error has been detected. Note that it can be any type of error: correctable or uncorrectable. The function is called with a 24-bit wide codeword of which bits 0 to 31 representsthe data and bits 32 to 39 represent the checksum. It returns the record type rs_32_8_type, containing the 32-bit wide data to be corrected, the syndrome and an indication if an error was detected over the complete codeword.

The rs_32_8_precorrect function is called with the intermediate result from the rs_32_8_check function. The input is the record type rs_32_8_type. It returns the record type rs_32_8_type, containing the 32-bit wide data to be corrected, the syndrome, intermediate data and an indication if an error was detected over the complete codeword.

The rs_32_8_correct function is called with the intermediate result from the rs_32_8_precorrect function. The input is the record type rs_32_8_type. It returns the record type rs_32_8_type, containing the corrected 32-bit wide data, an indication if the error was correctable or non-correctable over the complete codeword, and the union of the two.

To pipeline the decoder, the rs_32_8_check function should be called in the first stage and the intermediated result should be stored. Note that the intermediate result contains the input data required for the correction in the next stage. The rs_32_8_precorrect function should be called in the second stage. The rs_32_8_correct function should be called in the third stage.

88.3 Type descriptions

Table 1149 shows the type declarations used by the functions in the package (VHDL types).

*Table 1149.*Type declarations

Name	Field	Type	Function	Active
rs_32_8_type	err	Std_Logic	error detected	High
	cerr	Std_Logic	error corrected	
	merr	Std_Logic	errors uncorrected	
	data	Std_Logic_Vector(0 to 31)	data	
	s_1	Std_Logic_Vector(0 to 3)	-	-
	s_2	Std_Logic_Vector(0 to 3)	-	-
	elp_3_1	Std_Logic_Vector(0 to 3)	-	-

88.4 Library dependencies

Table 1150 shows the libraries used when instantiating the functions in the package (VHDL libraries).

*Table 1150.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	StdLib	All	Common VHDL functions

88.5 Instantiation

This example shows how the functions in the package can be instantiated.

The example design shows a codec for which the decoder is pipelined, with the error flag output one clock cycle earlier than the corrected data. Note that all input and outputs are synchronised to remove any timing constraints for pads in an example design. Timing analysis can be made purely for the register-to-register paths.

```
library  IEEE;
use      IEEE.Std_Logic_1164.all;

entity rs_gf4_32_8_codec is
   port(
      clk:        in    Std_Logic;

      din:        in    Std_Logic_Vector(0 to 31);       -- encoder input
      cout:       out   Std_Logic_Vector(0 to 39);       -- encoder output

      cin:        in    Std_Logic_Vector(0 to 39);       -- decoder input
      terr:       out   Std_Logic;                       -- intermediate error

      dout:       out   Std_Logic_Vector(0 to 31);       -- decoder output
      err:        out   Std_Logic;                       -- error detected
      cerr:       out   Std_Logic;                       -- error corrected
      merr:       out   Std_Logic);                      -- errors uncorrected
end entity;

library  grlib;
use      grlib.rs_gf4_e1.all;

architecture rtl of rs_gf4_32_8_codec is
   signal   s_din:      Std_Logic_Vector(0 to 31);
   signal   s_cout:     Std_Logic_Vector(0 to 39);

   signal   s_cin:      Std_Logic_Vector(0 to 39);
   signal   s_dout:     Std_Logic_Vector(0 to 31);
   signal   s_err:      Std_Logic;
   signal   s_cerr:     Std_Logic;
   signal   s_merr:     Std_Logic;

   signal   check:      rs_32_8_type;                    -- intermediate
   signal   precorr:    rs_32_8_type;
   signal   corr:       rs_32_8_type;
begin
   SyncronizeInput: process(clk)
   begin
      if Rising_Edge(clk) then
         s_din       <= din;
         s_cin       <= cin;
      end if;
   end process;

   SyncronizeOutput: process(clk)
   begin
      if Rising_Edge(clk) then
         cout        <= s_cout;
         err         <= corr.err;
         cerr        <= corr.cerr;
         merr        <= corr.merr;
         dout        <= corr.data;
         terr        <= check.err;
      end if;
   end process;

   encoder: process(clk)
   begin
      if Rising_Edge(clk) then
         s_cout      <= rs_32_8_encode(s_din);
      end if;
   end process;

   decoder: process(clk)
   begin
      if Rising_Edge(clk) then
         corr        <= rs_32_8_correct(precorr);        -- third phase
         precorr     <= rs_32_8_precorrect(check);       -- second phase
         check       <= rs_32_8_check(s_cin);            -- first phase
      end if;
   end process;
end architecture rtl;
```

89 RS(48, 32, 16, E=2) - Reed-Solomon encoder/decoder

89.1 Overview

The rs_gf4_e2 VHDL package provides functions for encoding and decoding data with a Reed-Solomon code. It also provides a data type storing intermediate results from the functions.

The Reed-Solomon data symbols are 4-bit wide, represented as GF(2^4). The Reed-Solomon code is a shortened RS(15, 11, 4) code, represented as RS(12, 8, 4). It has the capability to detect and correct two symbol errors anywhere in the codeword. The data is represented as 32 bits and the checksum is represented as 16 bits, and the code can correct up to two 4-bit errors when located within nibble boundaries.

89.2 Operation

89.2.1 Encoder

The encoder is defined by the rs_32_16_2_encode function. The function is called with the 32-bit wide data that should be encoded, and returns 48-bit wide codeword of which bits 0 to 31 represent the data and bits 32 to 47 represent the checksum.

89.2.2 Decoder

The decoder is defined by the rs_32_16_2_check, rs_32_16_2_precorrect and rs_32_16_2_correct functions. The decoder has been split in three functions to facilitate pipelining, with each function being fairly balanced with respect to the depth of the resulting combinatorial logic.

The rs_32_16_2_check function calculates the syndrome and returns a bit indicating whether an error has been detected. Note that it can be any type of error: correctable or uncorrectable. The function is called with a 48-bit wide codeword of which bits 0 to 31 represent the data and bits 32 to 47 represent the checksum. It returns the record type rs_32_16_2_type, containing the 32-bit wide data to be corrected, the syndrome and an indication if an error was detected.

The rs_32_16_2_precorrect function is called with the intermediate result from the rs_32_16_2_check function. The input is the record type rs_32_16_2_type. It returns the record type rs_32_16_2_type, containing the 32-bit wide data to be corrected, the syndrome, intermediate data and an indication if an error was detected.

The rs_32_16_2_correct function is called with the intermediate result from the rs_32_16_2_precorrect function. The input is the record type rs_32_16_2_type. It returns the record type rs_32_16_2_type, containing the corrected 32-bit wide data, an indication if the error was correctable or non-correctable, and the union of the two.

To pipeline the decoder, the rs_32_16_2_check function should be called in the first stage and the intermediated result should be stored. Note that the intermediate result contains the input data required for the correction in the next stage. The rs_32_16_2_precorrect function should be called in the second stage. The rs_32_16_2_correct function should be called in the third stage.

89.3 Type descriptions

Table 1151 shows the type declarations used by the functions in the package (VHDL types).

*Table 1151.*Type declarations

Name	Field	Type	Function	Active
rs_32_16_2_type	err	Std_Logic	errors detected	High
	cerr	Std_Logic	errors corrected	High
	merr	Std_Logic	errors uncorrected	High
	data	Std_Logic_Vector(0 to 31)	data	
	l_u	Std_Logic_Vector(0 to 1)		
	s_1	Std_Logic_Vector(0 to 3)		
	s_2	Std_Logic_Vector(0 to 3)		
	s_3	Std_Logic_Vector(0 to 3)		
	s_4	Std_Logic_Vector(0 to 3)		
	elp_5_1	Std_Logic_Vector(0 to 3)		
	elp_5_2	Std_Logic_Vector(0 to 3)		

89.4 Library dependencies

Table 1152 shows the libraries used when instantiating the functions in the package (VHDL libraries).

*Table 1152.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	StdLib	All	Common VHDL functions

89.5 Instantiation

This example shows how the functions in the package can be instantiated.

The example design shows a codec for which the decoder is pipelined, with the error flag output one clock cycle earlier than the corrected data. Note that all input and outputs are synchronised to remove any timing constraints for pads in an example design. Timing analysis can be made purely for the register-to-register paths.

```
library  IEEE;
use      IEEE.Std_Logic_1164.all;

entity rs_gf4_32_16_2_codec is
   port(
      clk:        in    Std_Logic;

      din:        in    Std_Logic_Vector(0 to 31);       -- encoder input
      cout:       out   Std_Logic_Vector(0 to 47);       -- encoder output

      cin:        in    Std_Logic_Vector(0 to 47);       -- decoder input
      terr:       out   Std_Logic;                       -- intermediate error

      dout:       out   Std_Logic_Vector(0 to 31);       -- decoder output
      err:        out   Std_Logic;                       -- error detected
      cerr:       out   Std_Logic;                       -- error corrected
      merr:       out   Std_Logic);                      -- errors uncorrected
end entity;

library  grlib;
use      grlib.rs_gf4_e2.all;
```

```
architecture rtl of rs_gf4_32_16_2_codec is
   signal   s_din:      Std_Logic_Vector(0 to 31);
   signal   s_cout:     Std_Logic_Vector(0 to 47);
   signal   s_cin:      Std_Logic_Vector(0 to 47);
   signal   s_dout:     Std_Logic_Vector(0 to 31);
   signal   s_err:      Std_Logic;
   signal   s_cerr:     Std_Logic;
   signal   s_merr:     Std_Logic;
   signal   check:      rs_32_16_2_type;                 -- intermediate
   signal   precorr:    rs_32_16_2_type;
   signal   corr:       rs_32_16_2_type;
begin
   SyncronizeInput: process(clk)
   begin
      if Rising_Edge(clk) then
         s_din       <= din;
         s_cin       <= cin;
      end if;
   end process;

   SyncronizeOutput: process(clk)
   begin
      if Rising_Edge(clk) then
         cout        <= s_cout;
         err         <= corr.err;
         cerr        <= corr.cerr;
         merr        <= corr.merr;
         dout        <= corr.data;
         terr        <= check.err;
      end if;
   end process;

   encoder: process(clk)
   begin
      if Rising_Edge(clk) then
         s_cout      <= rs_32_16_2_encode(s_din);
      end if;
   end process;

   decoder: process(clk)
   begin
      if Rising_Edge(clk) then
         corr        <= rs_32_16_2_correct(precorr);     -- third phase
         precorr     <= rs_32_16_2_precorrect(check);    -- second phase
         check       <= rs_32_16_2_check(s_cin);         -- first phase
      end if;
   end process;
end architecture rtl;
```

90 SDCTRL - 32/64-bit PC133 SDRAM Controller

90.1 Overview

The SDRAM controller handles PC133 SDRAM compatible memory devices attached to a 32 or 64 bit wide data bus. The controller acts as a slave on the AHB bus where it occupies a configurable amount of address space for SDRAM access. The SDRAM controller function is programmed by writing to a configuration register mapped into AHB I/O address space.

Chip-select decoding is provided for two SDRAM banks.

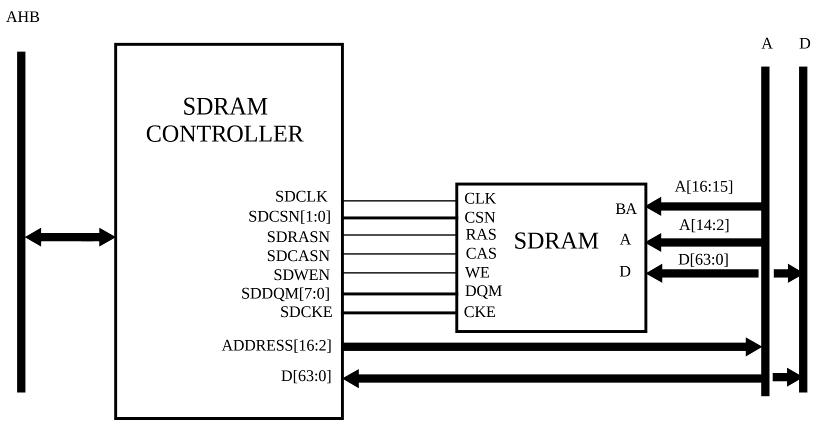


Figure 263. SDRAM Memory controller connected to AMBA bus and SDRAM

90.2 Operation

90.2.1 General

Synchronous dynamic RAM (SDRAM) access is supported to two banks of PC100/PC133 compatible devices. The controller supports 64M, 256M and 512M devices with 8 - 12 column-address bits, up to 13 row-address bits, and 4 banks. The size of each of the two banks can be programmed in binary steps between 4 Mbyte and 512 Mbyte. The operation of the SDRAM controller is controlled through the configuration register SDCFG (see section 90.3). The SDRAM bank's data bus width is configurable between 32 and 64 bits. When the VHDL generic *mobile* is set to a value not equal to 0, the controller supports mobile SDRAM.

90.2.2 Initialization

When the SDRAM controller is enabled, it automatically performs the SDRAM initialization sequence of PRECHARGE, 8x AUTO-REFRESH and LOAD-MODE-REG on both banks simultaneously. When mobile SDRAM functionality is enabled the initialization sequence is appended with a LOAD-EXTMODE-REG command. The controller programs the SDRAM to use page burst on read accesses and single location access on write accesses. If the *pwron* VHDL generic is 1, the initialization sequence is also sent automatically when reset is released. Note that some SDRAM devices require a stable clock of 100 us before any commands might be sent. When using on-chip PLL, this might not always be the case and the *pwron* VHDL generic should be set to 0 in such cases.

90.2.3 Configurable SDRAM timing parameters

To provide optimum access cycles for different SDRAM devices (and at different frequencies), three SDRAM parameters can be programmed through memory configuration register 2 (MCFG2): TCAS, TRP and TRFCD. The value of these fields affect the SDRAM timing as described in table 1153.

*Table 1153.*SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
CAS latency, RAS/CAS delay (t_{CAS}, t_{RCD})	TCAS + 2
Precharge to activate (t_{RP})	TRP + 2
Auto-refresh command period (t_{RFC})	TRFC + 3
Activate to precharge (t_{RAS})	TRFC + 1
Activate to Activate (t_{RC})	TRP + TRFC + 4

If the TCAS, TRP and TRFC are programmed such that the PC100/133 specifications are fulfilled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 100 and 133 MHz operation and the resulting SDRAM timing (in ns):

*Table 1154.*SDRAM example programming

SDRAM settings	t_{CAS}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
100 MHz, CL=2; TRP=0, TCAS=0, TRFC=4	20	80	20	70	50
100 MHz, CL=3; TRP=0, TCAS=1, TRFC=4	30	80	20	70	50
133 MHz, CL=2; TRP=1, TCAS=0, TRFC=6	15	82	22	67	52
133 MHz, CL=3; TRP=1, TCAS=1, TRFC=6	22	82	22	67	52

When mobile SDRAM support is enabled, one additional timing parameter (TXSR) can be programmed though the Power-Saving configuration register.

*Table 1155.*Mobile SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
Exit Self Refresh mode to first valid command (t_{XSR})	tXSR

90.2.4 Refresh

The SDRAM controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the SDCFG register. Depending on SDRAM type, the required period is typically 7.8 or 15.6 µs (corresponding to 780 or 1560 clocks at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by setting bit 31 in SDCFG register.

90.2.5 Self Refresh

The self refresh mode can be used to retain data in the SDRAM even when the rest of the system is powered down. When in the self refresh mode, the SDRAM retains data without external clocking and refresh are handled internally. The memory array that is refreshed during the self refresh operation is defined in the extended mode register. These settings can be changed by setting the PASR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the

PASR bits are changed. The supported “Partial Array Self Refresh” modes are: Full, Half, Quarter, Eighth, and Sixteenth array. “Partial Array Self Refresh” is only supported when mobile SDRAM functionality is enabled. To enable the self refresh mode, set the PMODE bits in the Power-Saving configuration register to “010” (Self Refresh). The controller will enter self refresh mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. When exiting this mode the controller introduce a delay defined by tXSR in the Power-Saving configuration register and a AUTO REFRESH command before any other memory access is allowed. The minimum duration of this mode is defined by tRAS. This mode is only available when the VHDL generic *mobile* is >= 1.

90.2.6 Power-Down

When entering the power-down mode all input and output buffers, excluding SDCKE, are deactivated. All data in the SDRAM is retained during this operation. To enable the power-down mode, set the PMODE bits in the Power-Saving configuration register to “001” (Power-Down). The controller will enter power-down mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits is cleared. The REFRESH command will still be issued by the controller in this mode. When exiting this mode a delay of one clock cycles are added before issue any command to the memory. This mode is only available when the VHDL generic *mobile* is >= 1.

90.2.7 Deep Power-Down

The deep power-down operating mode is used to achieve maximum power reduction by eliminating the power of the memory array. Data will not be retained after the device enters deep power-down mode. To enable the deep power-down mode, set the PMODE bits in the Power-Saving configuration register to “101” (Deep Power-Down). To exit the deep power-down mode the PMODE bits in the Power-Saving configuration register must be cleared. The controller will respond with an AMBA ERROR response to an AMBA access, that will result in a memory access, during Deep Power-Down mode. This mode is only available when the VHDL generic *mobile* is >= 1 and mobile SDRAM functionality is enabled.

90.2.8 Temperature-Compensated Self Refresh

The settings for the temperature-compensation of the Self Refresh rate can be controlled by setting the TCSR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the TCSR bits are changed. Note that some vendors implements a Internal Temperature-Compensated Self Refresh feature, which makes the memory ignore the TCSR bits. This functionality is only available when the VHDL generic *mobile* is >= 1 and mobile SDRAM functionality is enabled.

90.2.9 Drive Strength

The drive strength of the output buffers can be controlled by setting the DS bits in the Power-Saving configuration register. The extended mode register is automatically updated when the DS bits are changed. The available options are: full, three-quarter, one-half, and one-quarter drive strengths. This functionality is only available when the VHDL generic *mobile* is >= 1 and mobile SDRAM functionality is enabled.

90.2.10 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in the SDRAM Configuration register: PRE-CHARGE, AUTO-REFRESH, LOAD-MODE-REG (LMR) and LOAD-EXTMODE-REG (EMR). If the LMR command is issued, the CAS delay as programmed in SDCFG will be used, remaining fields are fixed: page read burst, single location write, sequential burst. If the EMR command is issued, the DS, TCSR and PASR as programmed in Power-Saving configuration register will be used. The command field will be cleared after a command has been executed. Note that when changing the value of the CAS delay, a LOAD-MODE-REGISTER command should be generated at the same time.

90.2.11 Read cycles

A read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command with data read after the programmed CAS delay. A read burst is performed if a burst access has been requested on the AHB bus. The read cycle is terminated with a PRE-CHARGE command, no banks are left open between two accesses. Note that only word bursts are supported by the SDRAM controller. The AHB bus supports bursts of different sizes such as bytes and half-words but they cannot be used.

90.2.12 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. A write burst on the AHB bus will generate a burst of write commands without idle cycles in-between. As in the read case, only word bursts are supported.

90.2.13 Address bus connection

The SDRAM address bus should be connected to SA[12:0], the bank address to SA[14:13], and the data bus to SD[31:0] or SD[63:0] if a 64-bit SDRAM data bus is used.

90.2.14 Data bus

The external SDRAM data bus is configurable to either 32 or 64 bits width, using the *sdbits* VHDL generic. A 64-bit data bus allows 64-bit (SO)DIMMs to be connected using the full data capacity of the devices. The polarity of the output enable signal to the data pads can be selected with the oepol generic. Sometimes it is difficult to fulfil the output delay requirements of the output enable signal. In this case, the vbdrive signal can be used instead of bdrive. Each index in this vector is driven by a separate register and a directive is placed on them so that they will not be removed by the synthesis tool.

90.2.15 Clocking

The SDRAM controller is designed for an external SDRAM clock that is in phase or slightly earlier than the internal AHB clock. This provides the maximum margin for setup and hold on the external signals, and allows highest possible frequency. For Xilinx and Altera devices, the GRLIB Clock Generator (CLKGEN) can be configured to produce a properly synchronized SDRAM clock. For other FPGA targets, the custom clock synchronization must be designed, or the inverted clock option can be used (see below). For ASIC targets, the SDRAM clock can be derived from the AHB clock with proper delay adjustments during place&route.

If the VHDL generic INVCLK is set, then all outputs from the SDRAM controller are delayed for 1/2 clock. This is done by clocking all output registers on the falling clock edge. This option can be used on FPGA targets where proper SDRAM clock synchronization cannot be achieved. The SDRAM clock can be the internal AHB clock without further phase adjustments. Since the SDRAM signals will only have 1/2 clock period to propagate, this option typically limits the maximum SDRAM frequency to 40 - 50 MHz.

90.3 Registers

The memory controller is programmed through register(s) mapped into the AHB I/O space defined by the controllers AHB BAR1.

*Table 1156.*SDRAM controller registers

AHB address offset	**Register**
0x0	SDRAM Configuration register
0x4	SDRAM Power-Saving configuration register

Table 1157. SDRAM configuration register

31	30	29 27	26	25 23	22 21	20 18	17	16	15	14 0
Refresh	tRP	tRFC	tCD	SDRAM bank size	SDRAM col. size	SDRAM command	Page-Burst	MS	D64	SDRAM refresh load value

31	SDRAM refresh. If set, the SDRAM refresh will be enabled.
30	SDRAM tRP timing. tRP will be equal to 2 or 3 system clocks (0/1). When mobile SDRAM support is enabled, this bit also represent the MSB in the tRFC timing.
29: 27	SDRAM tRFC timing. tRFC will be equal to 3 + field-value system clocks. When mobile SDRAM support is enabled, this field is extended with the bit 30.
26	SDRAM CAS delay. Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-COMMAND-REGISTER command must be issued at the same time. Also sets RAS/CAS delay (tRCD).
25: 23	SDRAM banks size. Defines the decoded memory size for each SDRAM chip select: “000”= 4 Mbyte, “001”= 8 Mbyte, “010”= 16 Mbyte .... “111”= 512 Mbyte.
22: 21	SDRAM column size. “00”=256, “01”=512, “10”=1024, “11”=4096 when bit[25:23]= “111”, 2048 otherwise.
20: 18	SDRAM command. Writing a non-zero value will generate an SDRAM command: “010”=PRE-CHARGE, “100”=AUTO-REFRESH, “110”=LOAD-COMMAND-REGISTER, “111”=LOAD-EXTENDED-COMMAND-REGISTER. The field is reset after command has been executed.
17	1 = pageburst is used for read operations, 0 = line burst of length 8 is used for read operations. (Only available when VHDL generic pageburst i set to 2)
16	Mobile SDR support enabled. ‘1’ = Enabled, ‘0’ = Disabled (read-only)
15	64-bit data bus (D64) - Reads ‘1’ if memory controller is configured for 64-bit data bus, otherwise ‘0’. Read-only.
14: 0	The period between each AUTO-REFRESH command - Calculated as follows: tREFRESH = ((reload value) + 1) / SYSCLK

Table 1158. SDRAM Power-Saving configuration register

31	30	29 24	23 20	19	18 16	15 7	6 5	4 3	2 0
ME	CE	Reserved	tXSR	res	PMODE	Reserved	DS	TCSR	PASR

31	Mobile SDRAM functionality enabled. ‘1’ = Enabled (support for Mobile SDRAM), ‘0’ = disabled (support for standard SDRAM)
30	Clock enable (CE). This value is driven on the CKE inputs of the SDRAM. Should be set to ‘1’ for correct operation. This register bit is read only when Power-Saving mode is other then none.
29: 24	Reserved
23: 20	SDRAM tXSR timing. tXSR will be equal to field-value system clocks. (Read only when Mobile SDR support is disabled).
19	Reserved

Table 1158. SDRAM Power-Saving configuration register

18: 16	Power-Saving mode (Read only when Mobile SDR support is disabled). “000”: none “001”: Power-Down (PD) “010”: Self-Refresh (SR) “101”: Deep Power-Down (DPD)
15: 7	Reserved
6: 5	Selectable output drive strength (Read only when Mobile SDR support is disabled). “00”: Full “01”: One-half “10”: One-quarter “11”: Three-quarter
4: 3	Reserved for Temperature-Compensated Self Refresh (Read only when Mobile SDR support is disabled). “00”: 70ªC “01”: 45ªC “10”: 15ªC “11”: 85ªC
2: 0	Partial Array Self Refresh (Read only when Mobile SDR support is disabled). “000”: Full array (Banks 0, 1, 2 and 3) “001”: Half array (Banks 0 and 1) “010”: Quarter array (Bank 0) “101”: One-eighth array (Bank 0 with row MSB = 0) “110”: One-sixteenth array (Bank 0 with row MSB = 00)

90.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x009. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

90.5 Configuration options

Table 1159 shows the configuration options of the core (VHDL generics).

*Table 1159.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where SDCFG register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#FFF#
wprot	Write protection.	0 - 1	0
invclk	Inverted clock is used for the SDRAM.	0 - 1	0
pwron	Enable SDRAM at power-on initialization	0 - 1	0
sdbits	32 or 64-bit data bus width.	32, 64	32
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
pageburst	Enable SDRAM page burst operation. 0: Controller uses line burst of length 8 for read operations. 1: Controller uses pageburst for read operations. 2: Controller uses pageburst/line burst depending on PageBurst bit in SDRAM configuration register.	0 - 2	0
mobile	Enable Mobile SDRAM support 0: Mobile SDRAM support disabled 1: Mobile SDRAM support enabled but not default 2: Mobile SDRAM support enabled by default 3: Mobile SDRAM support only (no regular SDR support)	0 - 3	0

90.6 Signal descriptions

Table 1160 shows the interface signals of the core (VHDL ports).

*Table 1160.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
AHBSI	1)	Input	AHB slave input signals	-
AHBSO	1)	Output	AHB slave output signals	-
SDI	WPROT	Input	Not used	-
	DATA[63:0]	Input	Data	High
SDO	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDWEN	Output	SDRAM write enable	Low
	RASN	Output	SDRAM row address strobe	Low
	CASN	Output	SDRAM column address strobe	Low
	DQM[7:0]	Output	SDRAM data mask: DQM[7] corresponds to DATA[63:56], DQM[6] corresponds to DATA[55:48], DQM[5] corresponds to DATA[47:40], DQM[4] corresponds to DATA[39:32], DQM[3] corresponds to DATA[31:24], DQM[2] corresponds to DATA[23:16], DQM[1] corresponds to DATA[15:8], DQM[0] corresponds to DATA[7:0].	Low
	BDRIVE	Output	Drive SDRAM data bus	Low/High[2]
	VBDRIVE[31:0]	Output	Identical to BDRIVE but has one signal for each data bit. Every index is driven by its own register. This can be used to reduce the output delay.	Low/High[2]
	ADDRESS[16:2]	Output	SDRAM address	Low
	DATA[31:0]	Output	SDRAM data	Low

1) see GRLIB IP Library User's Manual

2) Polarity selected with the oepol generic

90.7 Library dependencies

Table 1161 shows libraries used when instantiating the core (VHDL libraries).

*Table 1161.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

90.8 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the SDRAM controller. The external SDRAM bus is defined on the example designs port map and connected to the SDRAM controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

SDRAM controller decodes SDRAM area:0x60000000 - 0x6FFFFFFF. SDRAM Configuration register is mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;
sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

signal sdi   : sdctrl_in_type;
  signal sdo   : sdctrl_out_type;

  signal clkm, rstn : std_ulogic;
signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;
  signal gnd : std_ulogic;

begin

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                              tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);

  -- SDRAM controller
  sdc : sdctrl generic map (hindex => 3, haddr => 16#600#, hmask => 16#F00#,
    ioaddr => 1, pwron => 0, invclk => 0)
```

```
      port map (rstn, clkm, ahbsi, ahbso(3), sdi, sdo);

    -- input signals
    sdi.data(31 downto 0) <= sd(31 downto 0);

    -- connect SDRAM controller outputs to entity output signals
    sa <= sdo.address; sdcke <= sdo.sdcke; sdwen <= sdo.sdwen;
    sdcsn <= sdo.sdcsn; sdrasn <= sdo.rasn; sdcasn <= sdo.casn;
    sddqm <= sdo.dqm;

  --Data pad instantiation with scalar bdrive
  sd_pad : iopadv generic map (width => 32)
  port map (sd(31 downto 0), sdo.data, sdo.bdrive, sdi.data(31 downto 0));
  end;

  --Alternative data pad instantiation with vectored bdrive
  sd_pad : iopadvv generic map (width => 32)
  port map (sd(31 downto 0), sdo.data, sdo.vbdrive, sdi.data(31 downto 0));
  end;
```

91 SPI2AHB - SPI to AHB bridge

91.1 Overview

The SPI to AHB bridge is an SPI slave that provides a link between a SPI bus (that consists of two data signals, one clock signal and one select signal) and AMBA AHB. On the SPI bus the slave acts as an SPI memory device where accesses to the slave are translated to AMBA accesses. The core can translate SPI accesses to AMBA byte, half-word or word accesses. The access size to use is configurable via the SPI bus.

The core synchronizes the incoming clock and can operate in systems where other SPI devices are driven by asynchronous clocks.

GRLIB also contains a SPI master/slave controller core, without an AHB interface, where the transfer of each individual byte is controlled by software via an APB interface, see the SPICTRL core documentation for more information.

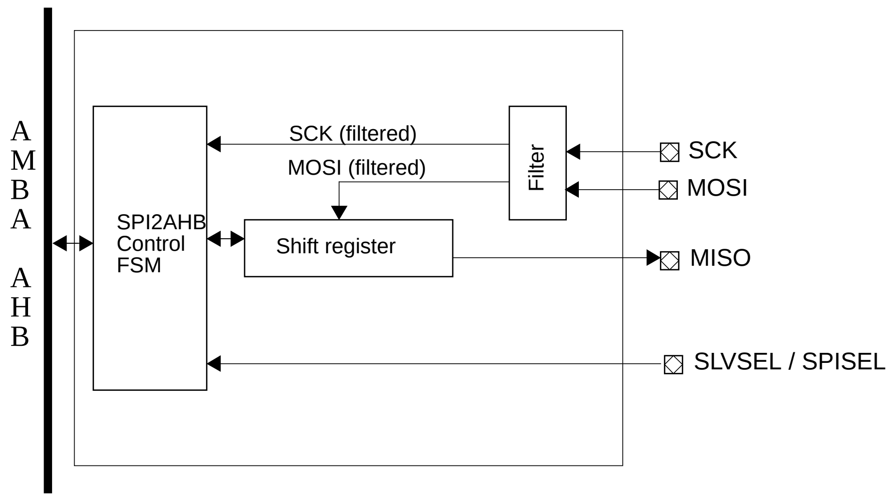


Figure 264. Block diagram, optional APB interface not shown

91.2 Transmission protocol

The SPI bus is a full-duplex synchronous serial bus. Transmission starts when a master selects a slave through the slave’s Slave Select (SLVSEL) signal and the clock line SCK transitions from its idle state. Data is transferred from the master through the Master-Output-Slave-Input (MOSI) signal and from the slave through the Master-Input-Slave-Output (MISO) signal. In some systems with only one master and one slave, the Slave Select input of the slave may be always active and the master does not need to have a slave select output. This does not apply to this SPI to AHB bridge, the slave select signal must be used to mark the start and end of an operation.

During a transmission on the SPI bus data is either changed or read at a transition of SCK. If data has been read at edge n, data is changed at edge n+1. If data is read at the first transition of SCK the bus is said to have clock phase 0, and if data is changed at the first transition of SCK the bus has clock phase 1. The idle state of SCK may be either high or low. If the idle state of SCK is low, the bus has clock polarity 0 and if the idle state is high the clock polarity is 1. The combined values of clock polarity (CPOL) and clock phase (CPHA) determine the mode of the SPI bus. Figure 265 shows one byte (0x55) being transferred MSb first over the SPI bus under the four different modes. Note that the idle state of the MOSI line is ‘1’ and that CPHA = 0 means that the devices must have data ready before the first transition of SCK. The figure does not include the MISO signal, the behavior of this line is the same as for the MOSI signal. However, due to synchronization the MISO signal will be delayed for a period of time that depends on the system clock frequency.

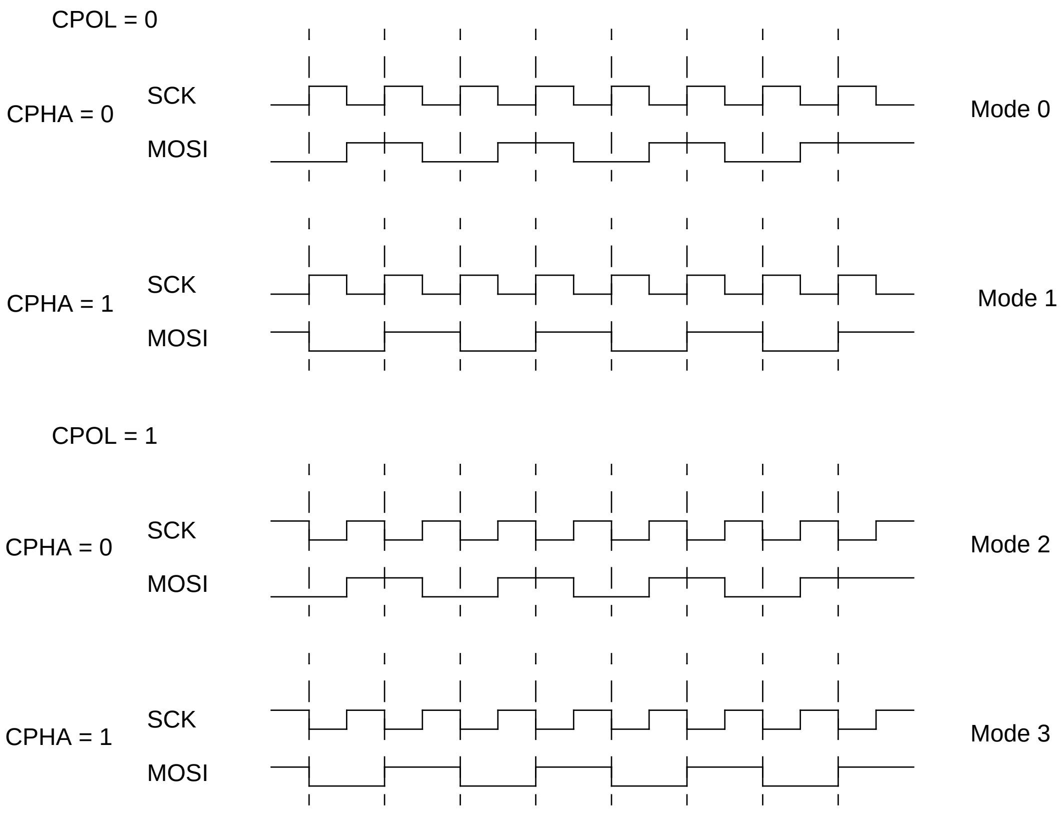


Figure 265. SPI transfer of byte 0x55 in all modes

The SPI to AHB bridge makes use of a protocol commonly used by SPI Flash memory devices. A master first selects the slave via the slave select signal and then issues a one-byte instruction. The instruction is then followed by additional bytes that contain address or data values. All instructions, addresses and data are transmitted with the most significant bit first. All AMBA accesses are done in big endian format. The first byte sent to or from the slave is the most significant byte.

91.3 System clock requirements and sampling

The core samples the incoming SPI SCK clock and does not introduce any additional clock domains into the system. Both the SCK and MOSI lines first pass through two stage synchronizers and are then filtered with a low pass filter.

The synchronizers and filters constrain the minimum system frequency. The core requires the SCK signal to be stable for at least two system clock cycles before the core accepts the SCK value as the new clock value. The core's reaction to transitions will be additionally delayed since both lines are taken through two-stage synchronizers before they are filtered. In order for the slave to be able to output data on the SCK 'change' transition and for this data to reach the master before the next edge the SCK frequency should not be higher than one tenth of the system frequency of core (with the standard VHDL generic *filter* setting of 2).

The slave select input should be asserted at least two system clock cycles before the SCK line starts transitioning.

91.4 SPI instructions

91.4.1 Overview

The core is controlled from the SPI bus by sending SPI instructions. Some commands require additional bytes in the form of address or data. The core makes use of the same instructions as commonly available SPI Flash devices. Table 1162 summarizes the available instructions.

*Table 1162.*SPI instructions

Instruction	Description	Instruction code	Additional bytes
RDSR	Read status/control register	0x05	Core responds with register value
WRSR	Write status/control register	0x01	New register value
READ	AHB read access	0x03	Four address bytes, after which core responds with data.
READD	AHB read access with dummy byte	0x0B	Four address butes and one dummy byte, after which core responds with data
WRITE	AHB write access	0x02	Four address bytes followed by data to be written

All instructions, addresses and data are transmitted with the most significant bit first. All AMBA accesses are done in big endian format. The first byte sent to or from the slave is the most significant byte.

91.4.2 SPI status/control register accesses (RDSR/WRSR)

The RDSR and WRSR instructions access the core’s SPI status/control register. The register is accessed by issuing the wanted instruction followed by the data byte to be written (WRSR) or any value on the byte in order to shift out the current value of the status/control register (RDSR). The fields available in the SPI status/control register are shown in table 1163.

Table 1163. SPI2AHB SPI status/control register

7	6	5	4	3	2	1	0
Reserved		PROT	MEXC	DMAACT	NACK	HSIZE	

7	Reserved, always zero (read only)
6	Read ahead (RAHEAD) - When this bit is set the core will make a new access to fetch data as soon as the last current data bit has been moved. Otherwise the core will not attempt the new access until the ‘change’ transition on SCK. Setting this bit to ‘1’ allows higher SCK frequencies to be used but will also result in a data fetch as soon as the current data has been read out. This means that RAHEAD may not be suitable when accessing FIFO interfaces. (read/write)
5	Memory protection triggered (PROT) - ‘1’ if last AHB access was outside the allowed memory area. Updated after each AMBA access (read only). Note that since this bit is updated after each access the RAHEAD = ‘1’ setting may hide errors.
4	Memory exception (MEXC) - ‘1’ if core receives AMBA ERROR response. Updated after each AMBA access (read only). Note that since this bit is updated after each access the RAHEAD = ‘1’ setting may hide errors.
3	DMA active (DMAACT) - ‘1’ if core is currently performing a DMA operation.
2	Malfunction (MALF): This bit is set to one by the core is DMA is not finished when a new byte starts getting shifted. If this bit is set to ‘1’ then the last AHB access was not successful.
1:0	AMBA access size (HSIZE) - Controls the access size that the core will use for AMBA accesses. 0: byte, 1: half-word, 2: word. HSIZE = “11” is illegal.

Reset value: 0x42

91.4.3 Read and write instructions (WRITE and READ/READD)

The READD is the same as the READ instruction with an additional dummy byte inserted after the four address bytes. To perform a read operation on AHB via the SPI bus the following sequence should be performed:

1. Assert slave select
2. Send READ instruction
3. Send four byte AMBA address, the most significant byte is transferred first

3a. Send dummy byte (if READD is used)

4. Read the wanted number of data bytes
5. De-assert slave select

To perform a write access on AHB via the SPI bus, use the following sequence:

1. Assert slave select
2. Send WRITE instruction
3. Send four byte AMBA address, the most significant byte is transferred first
4. Send the wanted number of data bytes
5. De-assert slave select

During consecutive read or write operations, the core will automatically increment the address. The access size (byte, halfword or word) used on AHB is set via the HSIZE field in the SPI status/control register.

The core always respects the access size specified via the HSIZE field. If a write operation writes fewer bytes than what is required to do an access of the specified HSIZE then the write data will be dropped, no access will be made on AHB. If a read operation reads fewer bytes than what is specified by HSIZE then the remaining read data will be dropped when slave select is de-asserted.

The core will not mask any address bits. Therefore it is important that the SPI master respects AMBA rules when performing half-word and word accesses. A half-word access must be aligned on a two byte address boundary (least significant bit of address must be zero) and a word access must be aligned on a four byte boundary (two least significant address bits must be zero).

The core can be configured to generate interrupt requests when an AHB access is performed if the core is implemented with the APB register interface, see the APB register documentation for details.

91.4.4 Memory protection

The core is configured at implementation time to only allow accesses to a specified AHB address range (which can be the full 4 GiB AMBA address range). If the core has been implemented with the optional APB register interface then the address range is soft configurable and the reset value is specified with VHDL generics.

The VHDL generics *ahbaddrh* and *ahbaddrl* define the base address for the allowed area. The VHDL generics *ahbmaskh* and *ahbmaskl* define the size of the area. The generics are used to assign the memory protection area's address and mask in the following way:

Protection address, bits 31:16 (*protaddr[31:16]*): ahbaddrh
Protection address, bits 15:0 (*protaddr[15:0]*): ahbaddrl
Protection address, bits 31:16 (*protmask[31:16]*): ahbmaskh
Protection address, bits 15:0 (*protmask[15:0]*): ahbmaskh

Before the core performs an AMBA access it will perform the check:

(((incoming address) xor (protaddr)) and protmask) /= 0x00000000

If the above expression is true (one or several bits in the incoming address differ from the protection address, and the corresponding mask bits are set to '1') then the access is inhibited. As an example, assume that *protaddr* is 0xA0000000 and *protmask* is 0xF0000000. Since *protmask* only has ones in the most significant nibble, the check above can only be triggered for these bits. The address range of allowed accessed will thus be 0xA0000000 - 0xAFFFFFFF..

The core will set the configuration register bit PROT if an access is attempted outside the allowed address range. This bit is updated on each AHB access and will be cleared by an access inside the allowed range. Note that the (optional) APB status register has a PROT field with a slightly different behavior.

91.5 Registers

The core can optionally be implemented with an APB interface that provides registers mapped into APB address space.

*Table 1164.*APB registers

APB address offset	**Register**
0x00	Control register
0x04	Status register
0x08	Protection address register
0x0C	Protection mask register

Table 1165. Control register

31	2	1	0
RESERVED		IRQEN	EN

31 : 2	RESERVED
1	Interrupt enable (IRQEN) - When this bit is set to ‘1’ the core will generate an interrupt each time the DMA field in the status register transitions from ‘0’ to ‘1’.
0	Core enable (EN) - When this bit is set to ‘1’ the core is enabled and will respond to SPI accesses. Otherwise the core will not react to SPI traffic.

Reset value: Implementation dependent

Table 1166. Status register

31	3	2	1	0
RESERVED		PROT	WR	DMA

31 : 3	RESERVED
2	Protection triggered (PROT) - Set to ‘1’ if an access has triggered the memory protection. This bit will remain set until cleared by writing ‘1’ to this position. Note that the other fields in this register will be updated on each AHB access while the PROT bit will remain at ‘1’ once set.
1	Write access (WR) - Last AHB access performed was a write access. This bit is read only.
0	Direct Memory Access (DMA) - This bit gets set to ‘1’ each time the core attempts to perform an AHB access. By setting the IRQEN field in the control register this condition can generate an interrupt. This bit can be cleared by software by writing ‘1’ to this position.

Reset value: 0x00000000

Table 1167. Protection address register

31	0
PROTADDR	

Table 1167. Protection address register

31 : 0	Protection address (PROTADDR) - Defines the base address for the memory area where the core is allowed to make accesses.

Reset value: Implementation dependent

Table 1168. Protection mask register

31 ... 0
PROTMASK

31 : 0	Protection mask (PROTMASK) - Selects which bits in the Protection address register that are used to define the protected memory area.

Reset value: Implementation dependent

91.6 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x05C. For a description of vendor and device identifiers see the GRLIB IP Library User’s Manual.

91.7 Configuration options

Table 1169 shows the configuration options of the core (VHDL generics). Two different top level entities for the core is available. One with the optional APB interface (spi2ahb_apb) and one without the APB interface (spi2ahb). The entity without the APB interface has fewer generics as indicated in the table below.

*Table 1169.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST	0
ahbaddrh	Defines bits 31:16 of the address used for the memory protection area	0 - 16#FFFF#	0
ahbaddrl	Defines bits 15:0 of the address used for the memory protection area	0 - 16#FFFF#	0
ahbmaskh	Defines bits 31:16 of the mask used for the memory protection area	0 - 16#FFFF#	0
ahbmaskl	Defines bits 15:0 of the mask used for the memory protection area	0 - 16#FFFF#	0
resen	Reset value for core enable bit (only available on the spi2ahb_apb entity).	0 - 1	0
pindex	APB slave index (only available on the spi2ahb_apb entity).	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR (only available on the spi2ahb_apb entity).	0 - 16#FFF#	0
pmask	MASK field of the APB BAR (only available on the spi2ahb_apb entity).	0 - 16#FFF#	16#FFF#
pirq	Interrupt line driven by APB interface (only available on the spi2ahb_apb entity).	0 - NAHBIRQ-1	0
oepol	Output enable polarity	0 - 1	0
filter	Low-pass filter length. This generic should specify, in number of system clock cycles plus one, the time of the shortest pulse on the SCK clock line to be registered as a valid value.	2 - 512	2
cpol	Clock polarity of SPI clock (SCK)	0 - 1	0

*Table 1169.*Configuration options

Generic name	Function	Allowed range	Default
cpha	Clock phase of SPI communication	0 - 1	0

91.8 Signal descriptions

Table 1170 shows the interface signals of the core (VHDL ports).

*Table 1170.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SPII	SCK	Input	SPI clock line input	-
	MOSI	Input	SPI data line input	-
	SPISEL	Input	SPI slave select input	
	Other fields	Input	Unused	
SPIO	MISO	Output	SPI data line output	-
	MISOOEN	Output	SPI data line output enable	Low**
	Other fields	Output	Unused	-

* see GRLIB IP Library User’s Manual
** depends on value of OEPOL VHDL generic.

91.9 Library dependencies

Table 1171 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1171.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPI	Component, signals	Component declaration, SPI signal definitions

91.10 Instantiation

The example below shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;
library gaisler;
use gaisler.misc.all;

entity spi2ahb_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- SPI signals
    miso : inout std_logic;
    mosi : in    std_logic;
    sck  : in    std_logic;
    sel  : in    std_logic;
    );
end;

architecture rtl of spi2ahb_ex is
  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector;
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector;
  -- SPI signals
  signal spislvi : spi_in_type;
  signal spislvo : spi_out_type;
begin

  -- AMBA Components are instantiated here
  ...
  -- SPI to AHB bridge
  spibridge : if CFG_SPI2AHB /= 0 generate
    withapb : if CFG_SPI2AHB_APB /= 0 generate
      spi2ahb0 : spi2ahb_apb
        generic map(hindex => 10,
          ahbaddrh => CFG_SPI2AHB_ADDRH, ahbaddrl => CFG_SPI2AHB_ADDRL,
          ahbmaskh => CFG_SPI2AHB_MASKH, ahbmaskl => CFG_SPI2AHB_MASKL,
          resen => CFG_SPI2AHB_RESEN, pindex => 11, paddr => 11, pmask => 16#fff#,
          pirq => 11, filter => CFG_SPI2AHB_FILTER, cpol => CFG_SPI2AHB_CPOL,
          cpha => CFG_SPI2AHB_CPHA)
        port map (rstn, clkm, ahbmi, ahbmo(10),
                  apbi, apbo(11), spislvi, spislvo);
    end generate;
    woapb : if CFG_SPI2AHB_APB = 0 generate
      spi2ahb0 : spi2ahb
        generic map(hindex => 10,
          ahbaddrh => CFG_SPI2AHB_ADDRH, ahbaddrl => CFG_SPI2AHB_ADDRL,
          ahbmaskh => CFG_SPI2AHB_MASKH, ahbmaskl => CFG_SPI2AHB_MASKL,
          filter => CFG_SPI2AHB_FILTER,
          cpol => CFG_SPI2AHB_CPOL, cpha => CFG_SPI2AHB_CPHA)
        port map (rstn, clkm, ahbmi, ahbmo(10),
                  spislvi, spislvo);
    end generate;
    spislv_miso_pad : iopad generic map (tech => padtech)
      port map (miso, spislvo.miso, spislvo.misooen, spislvi.miso);
    spislvl_mosi_pad : inpad generic map (tech => padtech)
      port map (miso, spislvi.mosi);
    spislv_sck_pad  : inpad generic map (tech => padtech)
      port map (sck, spislvi.sck);
    spislv_slvsel_pad : iopad generic map (tech => padtech)
      port map (sel, spislvi.spisel);
  end generate;
  nospibridge : if CFG_SPI2AHB = 0 or CFG_SPI2AHB_APB = 0 generate
    apbo(11) <= apb_none;
  end generate;
end;
```

92 SPICTRL - SPI Controller

92.1 Overview

The core provides a link between the AMBA APB bus and the Serial Peripheral Interface (SPI) bus and can be dynamically configured to function either as a SPI master or a slave. The SPI bus parameters are highly configurable via registers. Core features also include configurable word length, bit ordering, clock gap insertion, automatic slave select and automatic periodic transfers of a specified length. All SPI modes are supported and also a 3-wire mode where one bidirectional data line is used. In slave mode the core synchronizes the incoming clock and can operate in systems where other SPI devices are driven by asynchronous clocks.

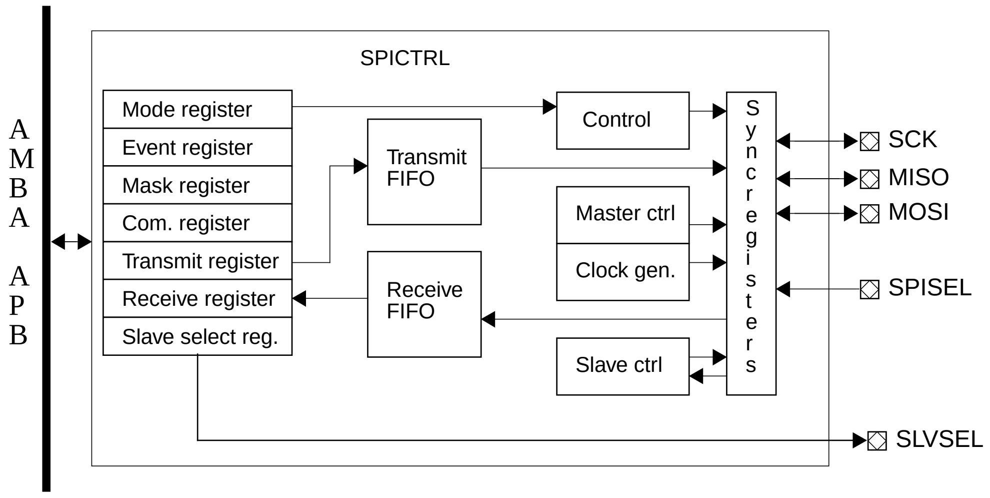


Figure 266. Block diagram

92.2 Operation

92.2.1 SPI transmission protocol

The SPI bus is a full-duplex synchronous serial bus. Transmission starts when a master selects a slave through the slave’s Slave Select (SLVSEL) signal and the clock line SCK transitions from its idle state. Data is transferred from the master through the Master-Output-Slave-Input (MOSI) signal and from the slave through the Master-Input-Slave-Output (MISO) signal. In a system with only one master and one slave, the Slave Select input of the slave may be always active and the master does not need to have a slave select output. If the core is configured as a master it will monitor the SPISEL signal to detect collisions with other masters, if SPISEL is activated the master will be disabled.

During a transmission on the SPI bus data is either changed or read at a transition of SCK. If data has been read at edge n, data is changed at edge n+1. If data is read at the first transition of SCK the bus is said to have clock phase 0, and if data is changed at the first transition of SCK the bus has clock phase 1. The idle state of SCK may be either high or low. If the idle state of SCK is low, the bus has clock polarity 0 and if the idle state is high the clock polarity is 1. The combined values of clock polarity (CPOL) and clock phase (CPHA) determine the mode of the SPI bus. Figure 267 shows one byte (0x55) being transferred MSb first over the SPI bus under the four different modes. Note that the idle state of the MOSI line is ‘1’ and that CPHA = 0 means that the devices must have data ready before the first transition of SCK. The figure does not include the MISO signal, the behavior of this line is the same as for the MOSI signal. However, due to synchronization issues the MISO signal will be delayed when the core is operating in slave mode, please see section 92.2.5 for details.

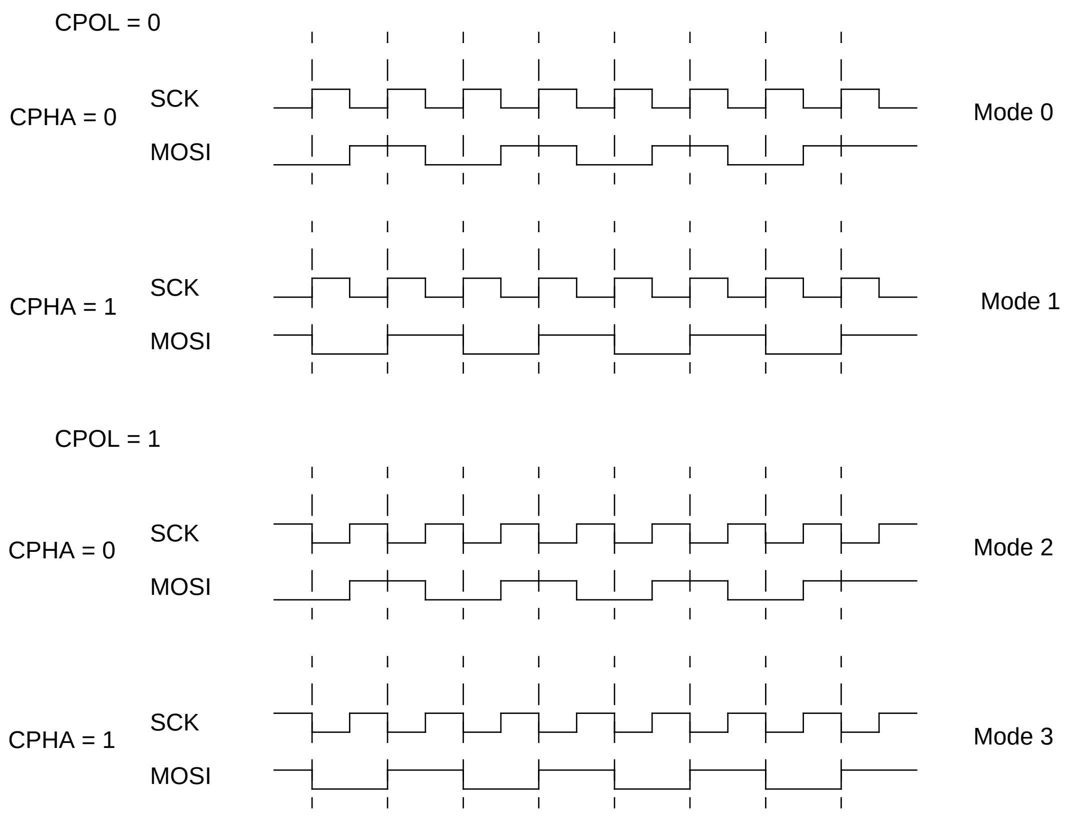


Figure 267. SPI transfer of byte 0x55 in all modes

92.2.2 3-wire transmission protocol

The core can be configured to operate in 3-wire mode, if the TWEN field in the core's Capability register is set to '1', where the controller uses a bidirectional dataline instead of separate data lines for input and output data. In 3-wire mode the bus is thus a half-duplex synchronous serial bus. Transmission starts when a master selects a slave through the slave's Slave Select (SLVSEL) signal and the clock line SCK transitions from its idle state. Only the Master-Output-Slave-Input (MOSI) signal is used for data transfer in 3-wire mode. The MISO signal is not used.

The direction of the first data transfer is determined by the value of the 3-wire Transfer Order (TTO) field in the core's Mode register. If TTO is '0', data is first transferred from the master (through the MOSI signal). After a word has been transferred, the slave uses the same data line to transfer a word back to the master. If TTO is '1' data is first transferred from the slave to the master. After a word has been transferred, the master uses the MOSI line to transfer a word back to the slave.

The data line transitions depending on the clock polarity and clock phase in the same manner as in SPI mode. The aforementioned slave delay of the MISO signal in SPI mode will affect the MOSI signal in 3-wire mode, when the core operates as a slave.

92.2.3 Receive and transmit queues

The core's transmit queue consists of the transmit register and the transmit FIFO. The receive queue consists of the receive register and the receive FIFO. The total number of words that can exist in each queue is thus the FIFO depth plus one. When the core has one or more free slots in the transmit queue it will assert the Not full (NF) bit in the event register. Software may only write to the transmit register when this bit is asserted. When the core has received a word, as defined by word length (LEN) in the Mode register, it will place the data in the receive queue. When the receive queue has one or more elements stored the Event register bit Not empty (NE) will be asserted. The receive register will only contain valid data if the Not empty bit is asserted and software should not access the receive register unless this bit is set. If the receive queue is full and the core receives a new word, an overrun condition will occur. The received data will be discarded and the Overrun (OV) bit in the Event register will be set.

The core will also detect underrun conditions. An underrun condition occurs when the core is selected, via SPISEL, and the SCK clock transitions while the transmit queue is empty. In this scenario the core will respond with all bits set to ‘1’ and set the Underrun (UN) bit in the Event register. An underrun condition will never occur in master mode. When the master has an empty transmit queue the bus will go into an idle state.

92.2.4 Clock generation

The core only generates the clock in master mode, the generated frequency depends on the system clock frequency and the Mode register fields DIV16, FACT, and PM. Without DIV16 the SCK frequency is:

$$SCKFrequency = \frac{AMBAclockfrequency}{(4-(2\cdot FACT))\cdot(PM+1)}$$

With DIV16 enabled the frequency of SCK is derived through:

$$SCKFrequency = \frac{AMBAclockfrequency}{16\cdot(4-(2\cdot FACT))\cdot(PM+1)}$$

Note that the fields of the Mode register, which includes DIV16, FACT and PM, should not be changed when the core is enabled. If the FACT field is set to 0 the core’s register interface is compatible with the register interface found in MPC83xx SoCs. If the FACT field is set to 1, the core can generate an SCK clock with higher frequency.

92.2.5 Slave operation

When the core is configured for slave operation it does not drive any SPI signal until the core is selected, via the SPISEL input, by a master. If the core operates in SPI mode when SPISEL goes low the core configures MISO as an output and drives the value of the first bit scheduled for transfer. If the core is configured into 3-wire mode the core will first listen to the MOSI line and when a word has been transferred drive the response on the MOSI line. If the core is selected when the transmit queue is empty it will transfer a word with all bits set to ‘1’ and the core will report an underflow.

Since the core synchronizes the incoming clock it will not react to transitions on SCK until two system clock cycles have passed. This leads to a delay of three system clock cycles when the data output line should change as the result of a SCK transition. This constrains the maximum input SCK frequency of the slave to (system clock) / 8 or less. The controlling master must also allow the decreased setup time on the slave data out line.

The core can also filter the SCK input. The value of the PM field in the Mode register defines for how many system clock cycles the SCK input must be stable before the core accepts the new value. If the PM field is set to zero, then the maximum SCK frequency of the slave is, as stated above, (system clock) / 8 or less. For each increment of the PM field the clock period of SCK must be prolonged by two times the system clock period as the core will require longer time discover and respond to SCK transitions.

92.2.6 Master operation

When the core is configured for master operation it will transmit a word when there is data available in the transmit queue. When the transmit queue is empty the core will drive SCK to its idle state. If the

SPISEL input goes low during master operation the core will abort any active transmission and the Multiple-master error (MME) bit will be asserted in the Event register. If a Multiple-master error occurs the core will be disabled. Note that the core will react to changes on SPISEL even if the core is operating in loop mode and that the core can be configured to ignore SPISEL by setting the IGSEL field in the Mode register.

92.2.7 Automated periodic transfers

The core supports automated periodic transfers if the AMODE field in the core’s Capability register is ‘1’. In this mode the core will perform transfers with a specified period and length. The steps below outline how to set up automated transfers:

1. Configure the core’s Mode register as a master and set the AMEN field (bit 31) to ‘1’. Possibly also configure the automatic slave select settings.
2. Write to the AM Mask registers to configure which parts of the AM transmit queue that will be used. The number of bits in the AM Mask registers that are set to one together with the word length (set in the Mode register) defines how long the transfer should be.
3. Write data to the AM transmit queue (AM Transmit registers). Only those registers that correspond to a bit that is set to one in the AM Mask registers need to be written.
4. Set the transfer period in the AM Period register.
5. Set the options for the automated transfers in the AM Configuration register
6. Set the ACT or EACT field in the AM Configuration register.
7. Wait for the Not Empty field to be set in the Event register
8. Read out the AM Receive queue (AM Receive registers). If lock bit (LOCK) in AM Configuration register is set then all registers which have a bit in the AM Mask registers set must be read. If the lock bit is not set software does not need to read out any data, the core can write new data to the AM Receive registers anyway.
9. Go back to step 7.

When an automated transfer is performed, data is not immediately placed in AM receive queue. Instead the data is placed in a temporary queue to ensure that a full transfer can be read out atomically without interference from incoming data.

The AM receive queue is filled with the data from the temporary queue if the AM receive queue is empty, or if it is full and Sequential transfers (SEQ) is disabled in the AM Configuration register. It is possible to configure the core not to place new data in the AM receive queue while software is reading out data from the queue. This is done by setting the lock bit (LOCK) in the AM Configuration register.

If the AM Configuration register’s SEQ bit is set the core will not move data from the temporary queue until the AM receive queue has been cleared. Demanding Sequential transfers means that the AM receive queue’s data will never be overwritten. However, data may still be lost, depending on the settings that determine how the temporary queue handles overflow conditions.

The controller will attempt to place data into the temporary receive queue when the automated transfer period counter reaches zero. If the temporary queue is filled, which can occur if the controller is prevented from moving the data to the receive queue, the core’s behavior will depend on the setting of the Strict Period (STRICT) field in the AM Configuration register:

If the value of STRICT is ‘0’ the core will delay the transfer and wait until the temporary queue has been cleared.

If the value of STRICT is ‘1’, and the contents of the temporary queue can not be moved to the AM receive queue, there will be an overflow condition in the temporary queue. The core’s behavior on a temporary queue overflow is defined by the AM Configuration register fields Overflow Transfer Behavior (OVTB) and Overflow Data Behavior (OVDB). If there is a temporary queue overflow and

OVTB is set, the transfer will be skipped and the core's internal period counter will be reloaded. If the OVTB bit is not set the transfer will be performed. If the transfer is performed and the OVDB bit is set the data will be disregarded. If the OVDB bit is not set the data will be placed in the temporary receive queue and the previous data will be overwritten.

A series of automated transfers can be started by an external event. If the AM Configuration register field EACT is set, the core will activate Automated transfers when its internal ASTART input signal goes high. When the core detects that EACT and ASTART are both set, it will set the AM Configuration register ACT bit and reset the EACT bit. Note that subsequent automated transfers will be started when the period counter reaches zero, if ERPT field of the AM Configuration register is set to zero. If the ERPT field is set to one then the ASTART input is used to start subsequent transfers instead.

When automated transfers are enabled by setting the AM Configuration register ACT bit, the core will send a pulse on its internal ASTART output signal. This means that several cores can be connected together and have their start event synchronized. To synchronize a start event, set the EACT bit in all cores, except in the last core which is activated by setting the AM Configuration register ACT field. The last core will then pulse its ASTART output and trigger the start event in all the other connected cores. When this has been done the cores' transfers will be synchronized. However this synchronization may be lost if a core's receive queues are filled and STRICT transfers are disabled, since this will lead to a delay in the start of the core's next transfer.

When the core operates in AM mode, the Receive and Transmit registers should not be accessed. Nor should the AM transmit registers be updated when automatic transfers are enabled.

92.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 1172.*SPI controller registers

APB address offset	**Register**
0x00	Capability register
0x04-0x1C	Reserved
0x20	Mode register
0x24	Event register
0x28	Mask register
0x2C	Command register
0x30	Transmit register
0x34	Receive register
0x38	Slave Select register (optional)
0x3C	Automatic slave select register*
0x40	AM Configuration register**
0x44	AM Period register**
0x48-0x4C	Reserved
0x50-0x5C	AM Mask register(s)***
0x200-0x3FC	AM Transmit register(s)****
0x400-0x5FC	AM Receive register(s)****

*Only available if ASEL (bit 17) in the SPI controller Capability register is set.
**Only available if AMODE (bit 18) in the SPI controller Capability register is set.

***Only available if AMODE (bit 18) in the SPI controller Capability register is set. Number of implemented registers depend on FDEPTH (bits 15:8) in the SPI controller Capability register in the following way: Number of registers = (FDEPTH-1)/32 + 1.

****Only available if AMODE (bit 18) in the SPI controller Capability register is set. Number of implemented registers equals FDEPTH (bits 15:8) in the SPI controller Capability register.

Table 1173. SPI controller Capability register

31 24	23 20	19	18	17	16
SSSZ	MAXWLEN	TWEN	AMODE	ASELA	SSEN

15 8	7	6 5	4 0
FDEPTH	SR	FT	REV

31 : 24	Slave Select register size (SSSZ) - If the core has been configured with slave select signals this field contains the number of available signals. This field is only valid is the SSEN bit (bit 16) is ‘1
23 : 20	Maximum word Length (MAXWLEN) - The maximum word length supported by the core: 0b0000 - 4-16, and 32-bit word length 0b0011-0b1111 - Word length is MAXWLEN+1, allows words of length 4-16 bits. The core must not be configured to use a word length greater than what is defined by this register.
19	Three-wire mode Enable (TWEN) - If this bit is ‘1’ the core supports three-wire mode.
18	Auto mode (AMODE) - If this bit is ‘1’ the core supports Automated transfers.
17	Automatic slave select available (ASELA) - If this bit is set, the core has support for setting slave select signals automatically.
16	Slave Select Enable (SSEN) - If the core has a slave select register, and corresponding slave select lines, the value of this field is one. Otherwise the value of this field is zero.
15 : 8	FIFO depth (FDEPTH) - This field contains the depth of the core’s internal FIFOs. The number of words the core can store in each queue is FDEPTH+1, since the transmit and receive registers can contain one word each.
7	SYNCRAM (SR) - If this field is ‘1’ the core has buffers implemented with SYNCRAM components.
6 : 5	Fault-tolerance (FT) - This field signals if the core has any fault-tolerant capabilities. “00” - No fault-tolerance. “01” - Parity DMR, “10” - TMR.
4 : 0	Core revision (REV) - This manual applies to core revision 5.

Table 1174. SPI controller Mode register

31	30	29	28	27	26	25	24	23 20	19 16
AMEN	LOOP	CPOL	CPHA	DIV16	REV	MS	EN	LEN	PM

15	14	13	12	11 7	6 5	4	3	2	1	0
TWEN	ASEL	FACT	OD	CG	ASELDEL	TAC	TTO	IGSEL	CITE	R

31	Auto mode enable (AMEN) - When this bit is set to ‘1’ the core will be able to perform automated periodic transfers. See the AM registers below. The core supports this mode if the AMODE field in the capability register is set to ‘1’. Otherwise writes to this field has no effect. When this bit is set to ‘1’ the core can only perform automated transfers. Software is allowed to initialize the transmit queue and to read out the receive queue but no transfers except the automated periodic transfers may be performed. The core must be configured to act as a master (MS field set to ‘1’) when performing automated transfers.
30	Loop mode (LOOP) - When this bit is set, and the core is enabled, the core’s transmitter and receiver are interconnected and the core will operate in loopback mode. The core will still detect, and will be disabled, on Multiple-master errors.
29	Clock polarity (CPOL) - Determines the polarity (idle state) of the SCK clock.
28	Clock phase (CPHA) - When CPHA is ‘0’ data will be read on the first transition of SCK. When CPHA is ‘1’ data will be read on the second transition of SCK.
27	Divide by 16 (DIV16) - Divide system clock by 16, see description of PM field below and see section 92.2.4 on clock generation. This bit has no significance in slave mode.
26	Reverse data (REV) - When this bit is ‘0’ data is transmitted LSB first, when this bit is ‘1’ data is transmitted MSB first. This bit affects the layout of the transmit and receive registers.
25	Master/Slave (MS) - When this bit is set to ‘1’ the core will act as a master, when this bit is set to ‘0’ the core will operate in slave mode.
24	Enable core (EN) - When this bit is set to ‘1’ the core is enabled. No fields in the mode register should be changed while the core is enabled. This can bit can be set to ‘0’ by software, or by the core if a multiple-master error occurs.

Table 1174. SPI controller Mode register

23 : 20	Word length (LEN) - The value of this field determines the length in bits of a transfer on the SPI bus. Values are interpreted as: 0b0000 - 32-bit word length 0b0001-0b0010 - Illegal values 0b0011-0b1111 - Word length is LEN+1, allows words of length 4-16 bits. The value of this field must never specify a word length that is greater than the maximum allowed word length specified by the MAXWLEN field in the Capability register.
19 : 16	Prescale modulus (PM) - This value is used in master mode to divide the system clock and generate the SPI SCK clock. The value in this field depends on the value of the FACT bit. If bit 13 (FACT) is ‘0’:The system clock is divided by 4*(PM+1) if the DIV16 field is ‘0’ and 16*4*(PM+1) if the DIV16 field is set to ‘1’. The highest SCK frequency is attained when PM is set to 0b0000 and DIV16 to ‘0’, this configuration will give a SCK frequency that is (system clock)/4. With this setting the core is compatible with the SPI register interface found in MPC83xx SoCs. If bit 13 (FACT) is ‘1’: The system clock is divided by 2*(PM+1) if the DIV16 field is ‘0’ and 16*2*(PM+1) if the DIV16 field is set to ‘1’. The highest SCK frequency is attained when PM is set to 0b0000 and DIV16 to ‘0’, this configuration will give a SCK frequency that is (system clock)/2. In slave mode the value of this field defines the number of system clock cycles that the SCK input must be stable for the core to accept the state of the signal. See section 92.2.5.
15	Three-wire mode (TW) - If this bit is set to ‘1’ the core will operate in 3-wire mode. This bit can only be set if the TWEN field of the Capability register is set to ‘1’.
14	Automatic slave select (ASEL) - If this bit is set to ‘1’ the core will swap the contents in the Slave select register with the contents of the Automatic slave select register when a transfer is started and the core is in master mode. When the transmit queue is empty, the slave select register will be swapped back. Note that if the core is disabled (by writing to the core enable bit or due to a multiple-master-error (MME)) when a transfer is in progress, the registers may still be swapped when the core goes idle. This bit can only be set if the ASELA field of the Capability register is set to ‘1’. Also see the ASELDEL field which can be set to insert a delay between the slave select register swap and the start of a transfer.
13	PM factor (FACT) - If this bit is 1 the core’s register interface is no longer compatible with the MPC83xx register interface. The value of this bit affects how the PM field is utilized to scale the SPI clock. See the description of the PM field.
12	Open drain mode (OD) - If this bit is set to ‘0’, all pins are configured for operation in normal mode. If this bit is set to ‘1’ all pins are set to open drain mode. The implementation of the core may or may not support open drain mode. If this bit can be set to ‘1’ by writing to this location, the core supports open drain mode. The pins driven from the slave select register are not affected by the value of this bit.
11 : 7	Clock gap (CG) - The value of this field is only significant in master mode. The core will insert CG SCK clock cycles between each consecutive word. This only applies when the transmit queue is kept non-empty. After the last word of the transmit queue has been sent the core will go into an idle state and will continue to transmit data as soon as a new word is written to the transmit register, regardless of the value in CG. A value of 0b00000 in this field enables back-to-back transfers.
6 : 5	Automatic Slave Select Delay (ASELDEL) - If the core is configured to use automatic slave select (ASEL field set to ‘1’) the core will insert a delay corresponding to ASELDEL*(SPI SCK cycle time)/2 between the swap of the slave select registers and the first toggle of the SCK clock. As an example, if this field is set to “10” the core will insert a delay corresponding to one SCK cycle between assigning the Automatic slave select register to the Slave select register and toggling SCK for the first time in the transfer. This field can only be set if the ASELA field of the Capability register is set to ‘1’.
4	Toggle Automatic slave select during Clock Gap (TAC) - If this bit is set, and the ASEL field is set, the core will perform the swap of the slave select registers at the start and end of each clock gap. The clock gap is defined by the CG field and must be set to a value >= 2 if this field is set. This field can only be set if the ASELA field of the Capability register is set to ‘1’.
3	3-wire Transfer Order (TTO) - This bit controls if the master or slave transmits a word first in 3-wire mode.If this bit is ‘0’, data is first transferred from the master to the slave. If this bit is ‘1’, data is first transferred from the slave to the master. This bit can only be set if the TWEN field of the Capability register is set to ‘1’.
2	Ignore SPISEL input (IGSEL) - If this bit is set to ‘1’ then the core will ignore the value of the SPISEL input.

Table 1174. SPI controller Mode register

1	Require Clock Idle for Transfer End (CITE) - If this bit is ‘0’ the core will regard the transfer of a word as completed when the last bit has been sampled. If this bit is set to ‘1’ the core will wait until it has set the SCK clock to its idle level (see CI field) before regarding a transfer as completed. This setting only affects the behavior of the TIP status bit, and automatic slave select toggling at the end of a transfer, when the clock phase (CP field) is ‘0’.
0	RESERVED (R) - Read as zero and should be written as zero to ensure forward compatibility.

Table 1175. SPI controller Event register

31	30 15	14	13	12	11	10	9	8	7 0
TIP	R	LT	R	OV	UN	MME	NE	NF	R

31	Transfer in progress (TIP) - This bit is ‘1’ when the core has a transfer in progress. Writes have no effect. This bit is set when the core starts a transfer and is reset to ‘0’ once the core considers the transfer to be finished. Behavior affected by setting of CITE field in Mode register.
30 : 15	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
14	Last character (LT) - This bit is set when a transfer completes if the transmit queue is empty and the LST bit in the Command register has been written. This bit is cleared by writing ‘1’, writes of ‘0’ have no effect.
13	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
12	Overrun (OV) - This bit gets set when the receive queue is full and the core receives new data. The core continues communicating over the SPI bus but discards the new data. This bit is cleared by writing ‘1’, writes of ‘0’ have no effect.
11	Underrun (UN) - This bit is only set when the core is operating in slave mode. The bit is set if the core’s transmit queue is empty when a master initiates a transfer. When this happens the core will respond with a word where all bits are set to ‘1’. This bit is cleared by writing ‘1’, writes of ‘0’ have no effect.
10	Multiple-master error (MME) - This bit is set when the core is operating in master mode and the SPISEL input goes active. In addition to setting this bit the core will be disabled. This bit is cleared by writing ‘1’, writes of ‘0’ have no effect.
9	Not empty (NE) - This bit is set when the receive queue contains one or more elements. It is cleared automatically by the core, writes have no effect.
8	Not full (NF) - This bit is set when the transmit queue has room for one or more words. It is cleared automatically by the core when the queue is full, writes have no effect.
7 : 0	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

Table 1176. SPI controller Mask register

31	30 15	14	13	12	11	10	9	8	7 0
TIPE	R	LTE	R	OVE	UNE	MMEE	NEE	NFE	R

31	Transfer in progress enable (TIPE) - When this bit is set the core will generate an interrupt when the TIP bit in the Event register transitions from ‘0’ to ‘1’.
30 : 15	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
14	Last character enable (LTE) - When this bit is set the core will generate an interrupt when the LT bit in the Event register transitions from ‘0’ to ‘1’.
13	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
12	Overrun enable (OVE) - When this bit is set the core will generate an interrupt when the OV bit in the Event register transitions from ‘0’ to ‘1’.
11	Underrun enable (UNE) - When this bit is set the core will generate an interrupt when the UN bit in the Event register transitions from ‘0’ to ‘1’.
10	Multiple-master error enable (MMEE) - When this bit is set the core will generate an interrupt when the MME bit in the Event register transitions from ‘0’ to ‘1’.
9	Not empty enable (NEE) - When this bit is set the core will generate an interrupt when the NE bit in the Event register transitions from ‘0’ to ‘1’.
8	Not full enable (NFE) - When this bit is set the core will generate an interrupt when the NF bit in the Event register transitions from ‘0’ to ‘1’.
7 : 0	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

Table 1177. SPI controller Command register

31 23	22	21 0
R	LST	R

31 : 23	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
22	Last (LST) - After this bit has been written to ‘1’ the core will set the Event register bit LT when a character has been transmitted and the transmit queue is empty. If the core is operating in 3-wire mode the Event register bit is set when the whole transfer has completed. This bit is automatically cleared when the Event register bit has been set and is always read as zero.
21 : 0	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

Table 1178. SPI controller Transmit register

31 0
TDATA

31 : 0	Transmit data (TDATA) - Writing a word into this register places the word in the transmit queue. This register will only react to writes if the Not full (NF) bit in the Event register is set. The layout of this register depends on the value of the REV field in the Mode register: Rev = ‘0’: The word to transmit should be written with its least significant bit at bit 0. Rev = ‘1’: The word to transmit should be written with its most significant bit at bit 31.

Table 1179. SPI controller Receive register

31 0
RDATA

31 : 0	Receive data (RDATA) - This register contains valid receive data when the Not empty (NE) bit of the Event register is set. The placement of the received word depends on the Mode register fields LEN and REV: For LEN = 0b0000 - The data is placed with its MSb in bit 31 and its LSb in bit 0. For other lengths and REV = ‘0’ - The data is placed with its MSB in bit 15. For other lengths and REV = ‘1’ - The data is placed with its LSB in bit 16. To illustrate this, a transfer of a word with eight bits (LEN = 7) that are all set to one will have the following placement: REV = ‘0’ - 0x0000FF00 REV = ‘1’ - 0x00FF0000

Table 1180. SPI Slave select register (optional)

31 SSSZ	SSSZ-1 0
R	SLVSEL

31 : SSSZ	RESERVED (R) - The lower bound of this register is determined by the Capability register field SSSZ if the SSEN field is set to 1. If SSEN is zero bits 31:0 are reserved.
(SSSZ-1) : 0	Slave select (SLVSEL) - If SSEN in the Capability register is 1 the core’s slave select signals are mapped to this register on bits (SSSZ-1):0. Software is solely responsible for activating the correct slave select signals, the core does not assert or deassert any slave select signal automatically.

Table 1181. SPI controller Automatic slave select register

31 SSSZ	SSSZ-1 0
R	ASLVSEL

31 : SSSZ	RESERVED (R) - The lower bound of this register is determined by the Capability register field SSSZ if the SSEN field is set to 1. If SSEN is zero bits 31:0 are reserved.

Table 1181. SPI controller Automatic slave select register

(SSSZ-1) : 0	Automatic Slave select (ASLVSEL) - If SSEN and ASELA in the Capability register are both ‘1’ the core’s slave select signals are assigned from this register when the core is about to perform a transfer and the ASEL field in the Mode register is set to ‘1’. After a transfer has been completed the core’s slave select signals are assigned the original value in the slave select register.

Note: This register is only available if ASELA (bit 17) in the SPI controller Capability register is set

Table 1182. SPI controller AM configuration register

31	30									16
RESERVED										
15	9	8	7	6	5	4	3	2	1	0
RESERVED		ECGC	LOCK	ERPT	SEQ	STRICT	OVTB	OVDB	ACT	EACT

31 : 9	RESERVED - This field is reserved for future use and should always be written as zero.
8	External clock gap control (ECGC) - If software sets this bit to ‘1’ then the clock gap between individual transfers in a set of automated transfers is controller by the core’s CSTART input instead of the CG field in the Mode registers. Note that the requirement that the CG field must be set to a value >= 2 if the TAC bit is set still applies even if this bit is set. Reset value ‘0’.
7	Lock bit (LOCK) - If software sets this bit to ‘1’ then the core will not place new data in the AM Receive registers while software is reading out new data.
6	External repeat (ERPT) - When this bit is set the core will use the input signal astart to start a new periodic transfer. If this bit is cleared, the period counter will be used instead.
5	Sequential transfers (SEQ) - When this bit is set the core will not update the receive queue unless the queue has been emptied by reading out its contents. Note that the contents in the temporary FIFO may still be overwritten with incoming data, depending on the setting of the other fields in this register.
4	Strict period (STRICT) - When this bit is set the core will always try to perform a transfer when the period counter reaches zero, if this bit is not set the core will wait until the receive FIFO is empty before it tries to perform a new transfer.
3	Overflow Transfer Behavior (OVTB) - If this bit is set to ‘1’ the core will skip transfers that would result in data being overwritten in the temporary receive queue. Note that this bit only decides if the transfer is performed. If this bit is set to ‘0’ a transfer will be performed and the setting of the Overflow Data Behavior bit (OVDB) will decide if data is actually overwritten.
2	Overflow Data Behavior (OVDB) - If this bit is set to ‘1’ the core will skip incoming data that would overwrite data in the receive queues. If this bit is ‘0’ the core will overwrite data in the temporary queue.
1	Activate automated transfers (ACT) - When this bit is set to ‘1’ the core will start to decrement the AM period register and perform automated transfers. The system clock cycle after this bit has been written to ‘1’ there will be a pulse on the core’s ASTART output. Automated transfers can be deactivated by writing this bit to ‘0’. The core will wait until any ongoing transfer has finished before deactivating automated transfers. Software should not perform any operation on the core before this bit has been read back as ‘0’. The data in the last transfer(s) will be lost if there is a transfer in progress when this bit is written to ‘0’. All words present in the transmit queue will also be dropped.
0	External activation of automated transfers (EACT) - When this bit is set to ‘1’ the core will activate automated transfers when the core’s ASTART input goes HIGH. When the core has been activated by the external signal this bit will be reset and the ACT field (bit 1 will be set).

Note: This register is only available if AMODE (bit 18) in the SPI controller Capability register is set

Table 1183. SPI controller AM period register

31	0
AMPER	

Table 1183. SPI controller AM period register

31 : 0	AM Period (AMPER) - This field contains the period, in system clock cycles, of the automated transfers. The core has an internal counter that is decremented each system clock cycle. When the counter reaches zero the core will begin to transmit all data in the transmit queue and reload the internal counter, which will immediately begin to start count down again. If the core has a transfer in progress when the counter reaches zero, the core will stall and not start a new transfer, or reload the internal counter, before the ongoing transfer has completed. The number of bits in this register is implementation dependent. Software should write this register with 0xFFFFFFFF and read back the value to see how many bits that are available.

Note: This register is only available if AMODE (bit 18) in the SPI controller Capability register is set

Table 1184. SPI controller AM Mask register(s)

31 … 0
AM MASK

31 : 0	AM Mask - This field is used as a bit mask to determine which words in the AM Transmit / Receive queues to read from / write to. Bit 0 of the first mask register corresponds to the first position in the queues, bit 1 of the first mask register to the second position, bit 0 of the second mask register corresponds to the 33:d position, etc. The total number of bits implemented equals FDEPTH (bit 15:8) in the SPI controller Capability register. If a bit is set to one then the core will read / write the corresponding position in the queue, otherwise it will be skipped. Software can write these registers at all times. However if a automated transfer is in progress when the write occurs, then the core will save the new value in a temporary register until the transfer is complete. The reset value is all ones.

Note: This register is only available if AMODE (bit 18) in the SPI controller Capability register is set

Table 1185. SPI controller AM Transmit register(s)

31 … 0
TDATA

31 : 0	Transmit data (TDATA) - Writing a word into these register places the word in the AM Transmit queue. The address of the register determines the position in the queue. Address offset 0x200 corresponds to the first position, offset 0x204 to the second position etc. The layout of the registers during write depends on the value of the REV field in the Mode register: Rev = ‘0’: The word to transmit should be written with its least significant bit at bit 0. Rev = ‘1’: The word to transmit should be written with its most significant bit at bit 31. The layout of the registers during read is fixed, the word is read with its least significant bit at bit 0.

Note: This register is only available if AMODE (bit 18) in the SPI controller Capability register is set

Table 1186. SPI controller AM Receive register(s)

31 … 0
RDATA

31 : 0	Receive data (RDATA) - The address of the register determines the position in the queue. Address offset 0x200 corresponds to the first position, offset 0x204 to the second position etc. The placement of the received word depends on the Mode register fields LEN and REV. For LEN = 0b0000 - The data is placed with its MSb in bit 31 and its LSb in bit 0. For other lengths and REV = ‘0’ - The data is placed with its MSB in bit 15. For other lengths and REV = ‘1’ - The data is placed with its LSB in bit 16. To illustrate this, a transfer of a word with eight bits (LEN = 7) that are all set to one will have the following placement: REV = ‘0’ - 0x0000FF00 REV = ‘1’ - 0x00FF0000

Note: This register is only available if AMODE (bit 18) in the SPI controller Capability register is set

92.4 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x02D. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

92.5 Configuration options

Table 1187 shows the configuration options of the core (VHDL generics).

*Table 1187.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by SPI controller	0 - NAHBIRQ-1	0
fdepth	FIFO depth. The FIFO depth in the core is 2^{fdepth}. Note that the depth of the transmit and receive queues is FIFO depth + 1 since the Transmit and Receive registers can hold one word. The number of AM Transmit / Receive registers are however 2^{fdepth}.	1 - 7	1
slvselen	Enable Slave Select register. When this value is set to 1 the core will include a slave select register that controls slvselsz slave select signals.	0 - 1	0
slvselsz	Number of Slave Select (slvsel) signals that the core will generate. These signals can be controlled via the Slave select register if the generic slvselen has been set to 1, otherwise they are driven to '1'.	1 - 32	1
oepol	Selects output enable polarity	0 - 1	0
odmode	Open drain mode. If this generic is set to 1, the OD bit in the mode register can be set to 1 and the core must be connected to I/O or OD pads.	0 - 1	0
automode	Enable automated transfers. If this generic is set to 1 the core will include support to automatically perform periodic transfers. The core's receive and transmit queues must not contain more than 128 words if automode is enabled.	0 - 1	0
acntbits	Selects the number of bits used in the AM period counter. This generic is only of importance if the automode generic is set to 1.	1 - 32	32
aslvsel	Enable automatic slave select. If this generic is set to 1 the core will include support for automatically setting the slave select register from the automatic slave select register before a transfer, or queue of transfers, starts. This generic is only significant of the slvselen generic is set to 1.	0 - 1	0
twen	Enable three-wire mode. If this generic is set to 1 the core will include support for three-wire mode.	0 - 1	1

*Table 1187.*Configuration options

Generic name	Function	Allowed range	Default
maxwlen	Determines the maximum supported word length. Values are defined as: 0 - Core will support lengths up to 32-bit words 0-2 - Illegal values 3-15 - Maximum word length is maxlen+1, allows words of length 4-16 bits. This generic sets the size of the slots in the transmit and receive queues. If the core will be used in an application that will never need to perform transfers with words as long as 32-bits, this setting can be used to save area.	0 - 15	0
netlist	If this generic is set to 0 (default) then the RTL version of the core will be used. If this generic is non-zero, the netlist version of the core will be used (if available) and the value of *netlist* will specify the target technology.	0 - NTECH	0
syncram	When this generic is set to 1 the core will instantiate SYNCRAM_2P components for the receive and transmit queues. The use of SYNCRAM_2P components can reduce area requirements, particularly when automode is enabled.	0 - 1	1
memtech	Selects memory technology for SYNCRAM_2P components.	0 - NTECH	0 (inferred)
ft	Enables fault tolerance for receive and transmit queues. 0 - No fault tolerance, 1 - Parity DMR, 2 - TMR. This generic only has effect if generic syncram is non-zero.	0 - 2	0
scantest	Enable scan test support. Only applicable if generic syncram is /= 0.	0 - 1	0
syncrst	Use only synchronous reset. If this generic is 0 then the spio.sckoen, spio.misooen, spio.mosioen and slvsel output will have asynchronous reset. Otherwise all registers within the core will have synchronous reset.	0 - 1	0
ignore	Enable AIGNORE/CIGNORE inputs (experimental)	0 - 1	0

92.6 Signal descriptions

Table 1188 shows the interface signals of the core (VHDL ports).

*Table 1188.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SPII	MISO	Input	Master-Input-Slave-Output data line, not used in 3-wire mode.	-
	MOSI	Input	Master-Output-Slave-Input data line	-
	SCK	Input	Serial Clock. If the core is instantiated in a system where it will work only as a master then drive this signal constant Low to save some area.	-
	SPISEL	Input	Slave select input. This signal should be driven High if it is unused in the design.	Low
	ASTART	Input	Automated transfer start. The core can be programmed to use this signal to start a set of automated transfers. This signal should be driven low if it is unused in the design. This signal must be synchronous to the CLK input.	High
	CSTART	Input	Automated clock start. This signal can be used to control when an individual transfer in a set of automated transfers should start. This signal doesn’t affect the start of the first transfer in the set. Also the core needs to be programmed to use the signal. This signal should be driven low if it is unused in the design. This signal must be synchronous to the CLK input.	High
	AIGNORE	Input	Ignore RX fifo adddress increment, ignore first TX fifo address increment	High
	CIGNORE	Input	Ignore TX fifo address increment	High
SPIO	MISO	Output	Master-Input-Slave-Output data line, not used in 3-wire mode.	-
	MISOOEN	Output	Master-Input-Slave-Output output enable, not used in 3-wire mode.	Low
	MOSI	Output	Master-Output-Slave-Input	-
	MOSIOEN	Output	Master-Output-Slave-Input output enable	Low
	SCK	Output	Serial Clock	-
	SCKOEN	Output	Serial Clock output enable	Low
	SSN	Output	Not used	-
	ASTART	Output	Automated transfer start indicator.	High
	AREADY	Output	Automated transfer ready indicator. Set each time an individual transfer in a set of automated transfers is completed.	High
SLVSEL [SSSZ-1:0]	N/A	Output	Slave select output(s). Used if the *slvselen* VHDL generic is set to 1. The range of the vector is (slvselsz-1):0	-

* see GRLIB IP Library User’s Manual

92.7 Library dependencies

Table 1189 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1189.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPI	Component, signals	SPI component and signal definitions.
TECHMAP	GENCOMP	Constant values	Technology constants
TECHMAP	NETCOMP	Component	Netlist component

92.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity spi_ex is
  port (
    clk    : in std_ulogic;
    rstn   : in std_ulogic;

    -- SPI signals
    sck    : inout std_ulogic;
    miso   : inout std_ulogic;
    mosi   : inout std_ulogic;
    spisel : in std_ulogic
    );
end;

architecture rtl of spi_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- SPIsignals
  signal spii : spi_in_type;
  signal spio : spi_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- SPI controller with FIFO depth 2 and no slave select register
  spictrl0 : spictrl generic map (pindex => 10, paddr => 10, pirq => 10,
                                  fdepth => 1, slvselen => 0, slvselsz => 1)
      port map (rstn, clkm, apbi, apbo(10), spii, spio, open);

    misopad : iopad generic map (tech => padtech)
      port map (miso, spio.miso, spio.misooen, spii.miso);
    mosipad : iopad generic map (tech => padtech)
      port map (mosi, spio.mosi, spio.mosioen, spii.mosi);
    sckpad : iopad generic map (tech => padtech)
      port map (sck, spio.sck, spio.sckoen, spii.sck);
    spiselpad : inpad generic map (tech => padtech)
      port map (spisel, spii.spisel);
end;
```

93 SPIMCTRL - SPI Memory Controller

93.1 Overview

The core maps a memory device connected via the Serial Peripheral Interface (SPI) into AMBA address space. Read accesses are performed by performing normal AMBA read operations in the mapped memory area. Other operations, such as writes, are performed by directly sending SPI commands to the memory device via the core's register interface. The core is highly configurable and supports most SPI Flash memory devices. The core also has limited experimental, SPI-mode, SD card support.

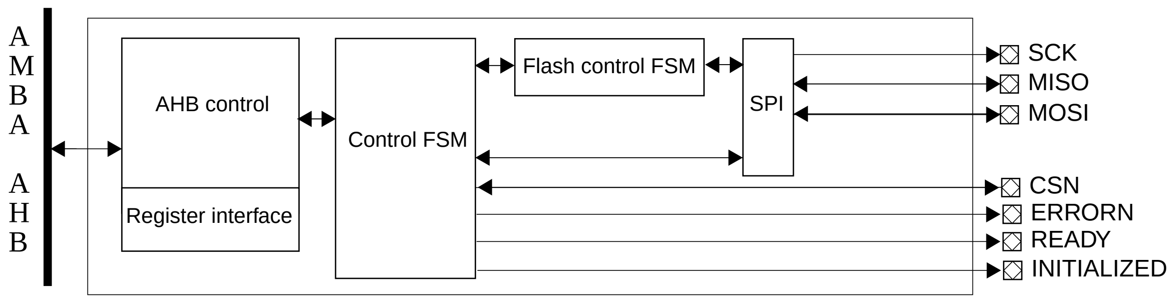


Figure 268. Block diagram

93.2 Operation

93.2.1 Operational model

The core has two memory areas that can be accessed via the AMBA bus; the I/O area and the ROM area. The ROM area maps the memory device into AMBA address space and the I/O area is utilized for status reporting and to issue user commands to the memory device.

When transmitting SPI commands directly to the device the ROM area should be left untouched. The core will issue an AMBA ERROR response if the ROM area is accessed when the core is busy performing an operation initiated via I/O registers.

Depending on the type of device attached the core may need to perform an initialization sequence. Accesses to the ROM area during the initialization sequence receive AMBA error responses. The core has successfully performed all necessary initialization when the Initialized bit in the core's status register is set, the value of this bit is also propagated to the core's output signal *spio.initialized*.

93.2.2 I/O area

The I/O area contains registers that are used when issuing commands directly to the memory device. By default, the core operates in System mode where it will perform read operations on the memory device when the core's ROM area is accessed. Before attempting to issue commands directly to the memory device, the core must be put into User mode. This is done by setting the User Control (USRC) bit in the core's Control register. Care should be taken to not enter User mode while the core is busy, as indicated by the bits in the Status register. The core should also have performed a successful initialization sequence before User mode accesses (INIT bit in the Status register should be set).

Note that a memory device may need to be clocked when there has been a change in the state of the chip select signal. It is recommended that software transmits a byte with the memory device deselected after entering and before leaving User mode.

The following steps are performed to issue a command to the memory device after the core has been put into User mode:

1. Check Status register and verify that the BUSY and DONE bits are cleared. Also verify that the core is initialized and not in error mode.
2. Optionally enable DONE interrupt by setting the Control register bit IEN.
3. Write command to Transmit register.
4. Wait for interrupt (if enabled) or poll DONE bit in Status register.
5. When the DONE bit is set the core has transferred the command and will have new data available in the Receive register.
6. Clear the Status register's DONE bit by writing one to its position.

The core should not be brought out of User mode until the transfer completes. Accesses to ROM address space will receive an AMBA ERROR response when the core is in User mode and when an operation initiated under User mode is active.

93.2.3 ROM area

The ROM area only supports AMBA read operations. Write operations will receive AMBA ERROR responses. When a read access is made to the ROM area the core will perform a read operation on the memory device. If the system has support for AMBA SPLIT responses the core will SPLIT the master until the read operation on the memory device has finished, unless the read operation is a locked access. A locked access never receives a SPLIT response and the core inserts wait states instead. If the system lacks AMBA SPLIT support the core will always insert wait states until the read operation on the memory device has finished. The core uses the value of the VHDL generic *spliten* to determine if the system has AMBA SPLIT support.

When the core is configured to work with a SD card, two types of timeouts are taken into account. The SD card is expected to respond to a command within 100 bytes and the core will wait for a data token following a read command for 312500 bytes. If the SD card does not respond within these limits the core will issue an AMBA error response to the access.

If an error occurs during an access to the ROM area the core will respond with an AMBA ERROR response. It will also set one or both of the Status register bits Error (ERR) and Timeout (TO). If the Error (ERR) bit remains set, subsequent accesses to the ROM area will also receive AMBA ERROR responses. The core can only detect failures when configured for use with a SD card.

The ROM area is marked as cacheable and prefetchable. This must be taken into account if the data in the ROM area is modified via the I/O area.

93.2.4 SPI memory device address offset

An offset can be specified at implementation via the core's *offset* VHDL generic. This offset will be added to all accesses to the ROM area before the address is propagated to the SPI memory device. Specifying an offset can be useful when the SPI memory device contains, as an example, FPGA configuration data at the lower addresses. By specifying an offset, the top of the SPI memory device can be used to hold user data. The AMBA system is unaware of the offset being added. An access to address *n* in the ROM area will be automatically translated to an access to address *offset* + *n* on the SPI memory device.

The offset must be accounted for when accessing the SPI memory device via the core's register interface. If data is programmed to the SPI memory device then the data must be written starting at the offset specified by the VHDL generic *offset*.

93.2.5 Supported memory devices

The core supports a wide range of memory devices due to its configuration options of read instruction, dummy byte insertion and dual output. Table 1190 below lists configurations for some memory devices.

*Table 1190.*Configurations for some memory devices

		VHDL generic*			
Manufacturer	**Memory device**	**sdcard**	**readcmd****	**dummybyte**	**dualoutput**
SD card	SD card	1	-	-	-
Spansion	S25FL-series	0	0x0B	1	0
Winbond	W25X-series	0	0x0B	1	0
	W25X-series with dual output read.	0	0x3B	1	1
* '-' means don't care ** Available in the core's Configuration register.					

When the core is configured for use with a SD card it does not use the VHDL generics *readcmd*, *dummybyte* nor *dualoutput*. All SD cards are initialized to use a block length of four bytes and accesses to the ROM area lead to a READ_SINGLE_BLOCK command being issued to the card.

When the VHDL generic *sdcard* is set to 0 the core is configured to issue the instruction defined by the VHDL generic *readcmd* to obtain data from the device. After an access to the ROM area the core will issue the read instruction followed by 24 address bits. If the VHDL generic *dummybyte* is set to 1 the core will issue a dummy byte following the address. After the possible dummy byte the core expects to receive data from the device. If the VHDL generic *dualoutput* is set to 1 the core will read data on both the MISO and MOSI data line. Otherwise the core will only use the MISO line for input data.

Many memory devices support both a READ and a FAST_READ instruction. The FAST_READ instruction can typically be issued with higher device clock frequency compared to the READ instruction, but requires a dummy byte to be present after the address. The most suitable choice of read instruction depends on the system frequency and on the memory device's characteristics.

93.2.6 Clock generation and power-up timing

The core generates the device clock by scaling the system clock. The VHDL generic *scaler* selects the divisor to use for the device clock that is used when issuing read instructions. The VHDL generic *altscaler* defines an alternate divisor that is used to generate the clock during power-up. This alternate divisor is used during initialization of SD cards and should be selected to produce a clock of 400 kHz or less when the core is configured for use with an SD card.

The alternate clock can be used for all communication by setting the Enable Alternate Scaler (EAS) bit in the Control register. When configuring the core for communication with a non-SD device in a system where the target frequency may change it is recommended to set the VHDL generic *scaler* to a conservative value and configure the alternate scaler to produce a faster clock. A boot loader can then set the Enable Alternate Scaler (EAS) bit early in the boot process when it has been determined that the system can use the memory device at a higher frequency.

When the core is configured for a non-SD card (VHDL generic *sdcard* set to 0), the VHDL generic *pwrupcnt* specifies how many system clock cycles after reset the core should be idle before issuing the first command.

93.3 Registers

The core is programmed through registers mapped into AHB address space.

*Table 1191.*SPIMCTRL registers

AHB address offset	**Register**
0x00	Configuration register
0x04	Control register
0x08	Status register
0x0C	Receive register
0x10	Transmit register

Table 1192. SPIMCTRL Configuration register

31 8	7 0
RESERVED	READCMD

31 :8	RESERVED
7:0	Read instruction (READCMD) - Read instruction that the core will use for reading from the memory device. When the core is configured to interface with a SD card this field will be set to 0.

Reset value: Implementation dependent

Table 1193. SPIMCTRL Control register

31 5	4	3	2	1	0
RESERVED	RST	CSN	EAS	IEN	USRC

31 :5	RESERVED
4	Reset core (RST) - By writing ‘1’ to this bit the user can reset the core. This bit is automatically cleared when the core has been reset. Reset core should be used with care. Writing this bit has the same effect as system reset. Any ongoing transactions, both on AMBA and to the SPI device will be aborted.
3	Chip select (CSN) - Controls core chip select signal. This field always shows the level of the core’s internal chip select signal. This bit is always automatically set to ‘1’ when leaving User mode by writing USRC to ‘0’.
2	Enable Alternate Scaler (EAS) - When this bit is set the SPI clock is divided by using the alternate scaler.
1	Interrupt Enable (IEN) - When this bit is set the core will generate an interrupt when a User mode transfer completes.
0	User control (USRC) - When this bit is set to ‘1’ the core will accept SPI data via the transmit register. Accesses to the memory mapped device area will return AMBA ERROR responses.

Reset value: 0x00000008

Table 1194. SPIMCTRL Status register

31 6	5	4	3	2	1	0
RESERVED	CD	TO	ERR	INIT	BUSY	DONE

31:6	RESERVED

Table 1194. SPIMCTRL Status register

5	Card Detect (CD) - This bit shows the value of the core's CD input signal if the core has been configured to work with an SD card. When using the core with a device that is hot-pluggable it may be necessary to monitor this bit and reset the core if a device has been disconnected and reconnected. This bit is only valid if the core has been configured for use with SD cards, otherwise it is always '0'.
4	Timeout (TO) - This bit is set to '1' when the core times out while waiting for a response from a SD card. This bit is only used when the core is configured for use with a SD card. The state is refreshed at every read operation that is performed as a result of an access to the ROM memory area. User mode accesses can never trigger a timeout. This bit is read only.
3	Error (ERR) - This bit is set to '1' when the core has entered error state. When the core is in this state all accesses to the ROM memory area will receive AMBA ERROR responses. The error state can be cleared by writing '1' to this bit. If the core entered error state during a read operation the ROM area will be available for read accesses again. This bit is follows the negated errorn output signal. The core will only detect errors when configured for use with an SD card. User mode accesses can never trigger an error.
2	Initialized (INIT) - This read only bit is set to '1' when the SPI memory device has been initialized. Accesses to the ROM area should only be performed when this bit is set to '1'.
1	Core busy (BUSY) - This bit is set to '1' when the core is performing an SPI operation.
0	Operation done (DONE) - This bit is set to '1' when the core has transferred an SPI command in user mode.

Reset value: 0x00000000

Table 1195. SPIMCTRL Receive register

31 … 8	7 … 0
RESERVED	RDATA

31 :8	RESERVED
7:0	Receive data (RDATA) : Contains received data byte

Reset value: 0x000000UU, where U is undefined

Table 1196. SPIMCTRL Transmit register

31 … 8	7 … 0
	TDATA

31 :8	RESERVED
7:0	Transmit data (TDATA) - Data byte to transmit

Reset value: 0x00000000

93.4 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x045. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

93.5 Implementation

93.5.1 Technology mapping

The core does not instantiate any technology specific primitives.

93.5.2 RAM usage

The core does not use any RAM components.

93.6 Configuration options

Table 1197 shows the configuration options of the core (VHDL generics).

*Table 1197.*Configuration options

Generic name	**Function**	**Allowed range**	**Default**
hindex	AHB slave index	0 - (NAHBSLV-1)	0
hirq	Interrupt line	0 - (NAHBIRQ-1)	0
faddr	ADDR field of the AHB BAR1 defining ROM address space.	0 - 16#FFF#	16#000#
fmask	MASK field of the AHB BAR1 defining ROM address space.	0 - 16#FFF#	16#FFF#
ioaddr	ADDR field of the AHB BAR0 defining register address space.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR0 defining register space.	0 - 16#FFF#	16#FFF#
spliten	If this generic is set to 1 the core will issue AMBA SPLIT responses when it is busy performing an operation on the memory device. Otherwise the core will insert wait states until the operation completes.	0 - 1	0
oepol	Select polarity of output enable signals. 0 = active low.	0 - 1	0
sdcard	Enable support for SD card	0 - 1	0
readcmd	Read instruction of memory device	0 - 16#FF#	16#0B#
dummybyte	Output dummy byte after address	0 - 1	0
dualoutput	Use dual output when reading data from device	0 - 1	0
scaler	Clock divisor used when generating device clock is 2^{scaler}	1 - 512	1
altscaler	Clock divisor used when generating alternate device clock is $2^{altscaler}$	1 - 512	1
pwrupcnt	Number of clock cycles to wait before issuing first command to memory device	N/A	0
maxahbaccsz	Maximum supported AHB access size. The core will support accesses ranging from 8-bit (BYTE) to the size set by maxahbaccsz. The maximum access size is 256 bits (8WORD). For SD card communication, this generic must be set to 32.	32, 64, 128, 256	AHBDW
offset	Specifies offset that will be added to incoming AMBA address before address is propagated to SPI flash device. An access to memory position *n* in the core’s ROM area will be translated to an access to SPI memory device address *n* + *offset*. Note that this only applies to accesses to the ROM area, accesses via the core’s register interface are unaffected.	-	0

93.7 Signal descriptions

Table 1198 shows the interface signals of the core (VHDL ports).

*Table 1198.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SPII	MISO	Input	Master-input slave-output data line SD card connection: DAT0	-
	MOSI	Input	Master-output slave-input data line SD card connection: None	-
	CD	Input	Card detection. Used in SD card mode to detect if a card is present. Must be pulled high if this functionality is not used. SD card connection: CD/DAT3	High
SPIO	MOSI	Output	Master-output slave-input data line SD card connection: CMD	-
	MOSIOEN	Output	Master-output slave-input output enable	-
	SCK	Output	SPI clock SD card connection: CLK	-
	CSN	Output	Chip select SD card connection: CD/DAT3	Low
	CDCSNOEN	Output	Chip select output enable. If the core is configured to work with an SD card this signal should be connected to the I/O pad that determines if CSN should drive the CD/DAT3 line. For other SPI memory devices this signal can be left unconnected and CSN should be connected to an output pad.	-
	ERRORN	Output	Error signal. Negated version of Error bit in the core's Status register.	Low
	READY	Output	When this signal is low the core is busy performing an operation.	High
	INITIALIZED	Output	This bit goes high when the SPI memory device has been initialized and can accept read accesses. This signal has the same value as the Initialized (INIT) bit in the core's Status register.	High

* see GRLIB IP Library User's Manual

93.8 Library dependencies

Table 1199 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1199.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GAISLER	SPI	Component, signals	Component and signal definitions
GRLIB	AMBA	Signals	AMBA signal definitions

93.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.memctrl.all;

entity spimctrl_ex is
  port (
    clk       : in  std_ulogic;
    rstn      : in  std_ulogic;
    -- SPIMCTRL signals
    -- For SD Card
    sd_dat    : in  std_ulogic;
    sd_cmd    : out std_ulogic;
    sd_sck    : out std_ulogic;
    sd_dat3   : inout std_ulogic;
    -- For SPI Flash
    spi_c     : out std_ulogic;
    spi_d     : out std_ulogic;
    spi_q     : in  std_ulogic;
    spi_sn    : out std_ulogic
    );
end;

architecture rtl of spimctrl_ex is
  -- AMBA signals
  signal ahbsi  : ahb_slv_in_type;
  signal ahbso  : ahb_slv_out_vector := (others => ahbs_none);
  ...
  -- SPIMCTRL signals
  signal spmi0, spmi1 : spimctrl_in_type;
  signal spmo0, spmo1 : spimctrl_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- Two cores are instantiated below. One configured for use with an SD card and one
  -- for use with a generic SPI memory device. Usage of the errorn, ready and
  -- initialized signals is not shown.

  -- SPMCTRL core, configured for use with SD card
  spimctrl0 : spimctrl
      generic map (hindex => 3, hirq => 3, faddr => 16#a00#, fmask  => 16#ff0#,
                   ioaddr => 16#100#, iomask => 16#fff#, spliten => CFG_SPLIT,
                   sdcard => 1, scaler => 1, altscaler => 7)
      port map (rstn, clk, ahbsi, ahbso(3), spmi0, spmo0);

    sd_miso_pad : inpad generic map (tech => padtech)
      port map (sd_dat, spmi0.miso);
    sd_mosi_pad : outpad generic map (tech => padtech)
      port map (sd_cmd, spmo0.mosi);
    sd_sck_pad  : outpad generic map (tech => padtech)
      port map (sd_clk, spmo0.sck);
    sd_slvsel0_pad : iopad generic map (tech => padtech)
      port map (sd_dat3, spmo0.csn, spmo0.cdcsnoen, spmi0.cd);
    -- Alternative use of cd/dat3 if connection detect is not wanted or available:
    -- sd_slvsel0_pad : outpad generic map (tech => padtech)
    --   port map (sd_dat3, spmo0.csn);
    --  spmi0.cd <= ‘1’; -- Must be set if cd/dat3 is not bi-directional

  -- SPMCTRL core, configured for use with generic SPI Flash memory with read
  -- command 0x0B and a dummy byte following the address.
  spimctrl1 : spimctrl
      generic map (hindex => 4, hirq => 4, faddr => 16#b00#, fmask  => 16#fff#,
                   ioaddr => 16#200#, iomask => 16#fff#, spliten => CFG_SPLIT,
                   sdcard => 0, readcmd => 16#0B#, dummybyte => 1, dualoutput => 0,
                   scaler => 1, altscaler => 1)
      port map (rstn, clk, ahbsi, ahbso(4), spmi1, spmo1);

    spi_miso_pad : inpad generic map (tech => padtech)
      port map (spi_q, spmi1.miso);
    spi_mosi_pad : outpad generic map (tech => padtech)
      port map (spi_d, spmo1.mosi);
    spi_sck_pad  : outpad generic map (tech => padtech)
      port map (spi_c, spmo1.sck);
    spi_slvsel0_pad : outpad generic map (tech => padtech)
      port map (spi_sn, spmo1.csn);
end;
```

94 SRCTRL- 8/32-bit PROM/SRAM Controller

94.1 Overview

SRCTRL is an 8/32-bit PROM/SRAM/IO controller that interfaces external asynchronous SRAM, PROM and I/O to the AMBA AHB bus. The controller can handle 32-bit wide SRAM and I/O, and either 8- or 32-bit PROM.

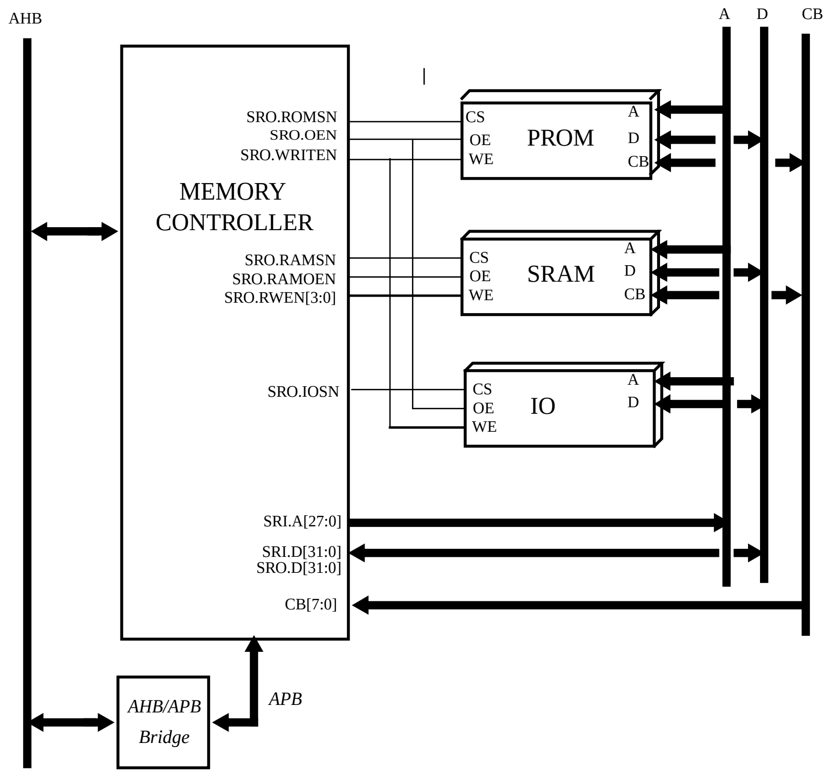


Figure 269. 8/32-bit PROM/SRAM/IO controller

The controller is configured through VHDL-generics to decode three address ranges: PROM, SRAM and I/O area. By default PROM area is mapped into address range 0x0 - 0x00FFFFFF, the SRAM area is mapped into address range 0x40000000 - 0x40FFFFFF, and the I/O area is mapped to 0x20000000 - 0x20FFFFFF.

One chip select is decoded for the I/O area, while SRAM and PROM can have up to four and two select signals respectively. The controller generates both a common write-enable signal (WRITEN) as well as four byte-write enable signals (WREN). If the SRAM uses a common write enable signal the controller can be configured to perform read-modify-write cycles for byte and half-word write accesses. Number of waitstates is separately configurable for the three address ranges.

A single write-enable signal is generated for the PROM area (WRITEN), while four byte-write enable signals (RWEN[3:0]) are provided for the SRAM area. If the external SRAM uses common write enable signal, the controller can be configured to perform read-modify-write cycles for byte and half-word write accesses.

Number of waitstates is configurable through VHDL generics for both PROM and SRAM areas.

A signal (BDRIVE) is provided for enabling the bidirectional pads to which the data signals are connected. The oepol generic is used for selecting the polarity of these enable signals. If output delay is an issue, a vectored output enable signal (VBDRIVE) can be used instead. In this case, each pad has its own enable signal driven by a separate register. A directive is placed on these registers so that they will not be removed during synthesis (if the output they drive is used in the design).

94.2 8-bit PROM access

The SRCTRL controller can be configured to access a 8-bit wide PROM. The data bus of external PROM should be connected to the upper byte of the 32-bit data bus, i.e. D[31:24]. The 8-bit mode is enabled with the prom8en VHDL generic. When enabled, read accesses to the PROM area will be done in four-byte bursts. The whole 32-bit word is then presented on the AHB data bus. Writes should be done one byte at a time and the byte should always be driven on bit 31-24 on the AHB data bus independent of the byte address.

It is possible to dynamically switch between 8- and 32-bit PROM mode using the BWIDTH[1:0] input signal. When BWIDTH is “00” then 8-bit mode is selected. If BWIDTH is “10” then 32-bit mode is selected. Other BWIDTH values are reserved for future use.

SRAM access is not affected by the 8-bit PROM mode.

94.3 PROM/SRAM waveform

Read accesses to 32-bit PROM and SRAM has the same timing, see figure below.

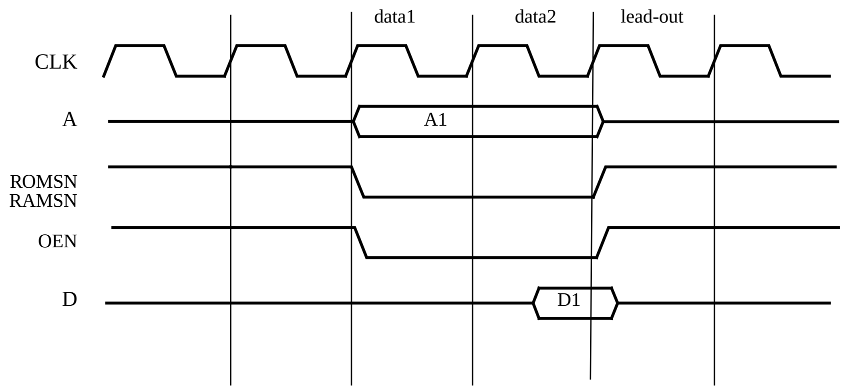


Figure 270. 32-bit PROM/SRAM/IO read cycle

The write access for 32-bit PROM and SRAM can be seen below.

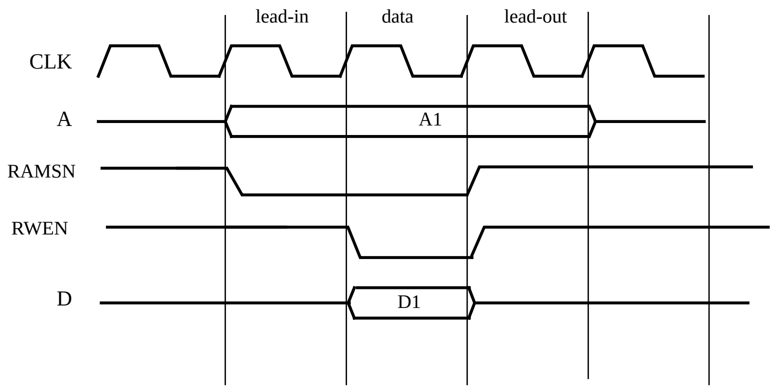


Figure 271. 32-bit PROM/SRAM/IO write cycle

If waitstates are configured through the VHDL generics, one extra data cycle will be inserted for each waitstate in both read and write cycles.

94.4 Burst cycles

To improve the bandwidth of the memory bus, accesses to consecutive addresses can be performed in burst mode. Burst transfers will be generated when the memory controller is accessed using an AHB burst request. These includes instruction cache-line fills and burst from DMA masters. The timing of a burst cycle is identical to the programmed basic cycle with the exception that during read cycles, the lead-out cycle will only occurs after the last transfer.

94.5 Registers

The core does not implement any user programmable registers.

All configuration is done through the VHDL generics.

94.6 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x008. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

94.7 Configuration options

Table 1201 shows the configuration options of the core (VHDL generics).

*Table 1200.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0xFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#FF0#
ramaddr	ADDR field of the AHB BAR1 defining SRAM address space. Default SRAM area is 0x40000000-0x40FFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR1 defining SRAM address space.	0 -16#FFF#	16#FF0#
ioaddr	ADDR field of the AHB BAR2 defining IO address space. Default IO area is 0x20000000-0x20FFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR2 defining IO address space.	0 -16#FFF#	16#FF0#
ramws	Number of waitstates during access to SRAM area	0 - 15	0
romws	Number of waitstates during access to PROM area	0 - 15	2
iows	Number of waitstates during access to IO area	0 - 15	2
rmw	Enable read-modify-write cycles.	0 - 1	0
prom8en	Enable 8 - bit PROM accesses	0 - 1	0
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
srbanks	Set the number of SRAM banks	1 - 5	1
banksz	Set the size of bank 1 - 4. 0 = 8 Kbyte, 1 = 16 Kbyte, ... , 13 = 64Mbyte.	0 - 13	13
romasel	address bit used for PROM chip select.	0 - 27	19

94.8 Signal description

Table 1200 shows the interface signals of the core (VHDL ports).

*Table 1201.*Signal descriptions

Signal name	Field	Type	Function	Polarity
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
SRI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Not used	-
	BEXCN	Input	Not used	-
	WRN[3:0]	Input	Not used	-

	BWIDTH[1:0]	Input	BWIDTH=”00” => 8-bit PROM mode
			BWIDTH=”10” => 32-bit PROM mode
				-

	SD[31:0]	Input	Not used	-
SRO	ADDRESS[27:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data	High
	RAMSN[4:0]	Output	SRAM chip-select	Low
	RAMOEN[4:0]	Output	SRAM output enable	Low
	IOSN	Output	Not used. Driven to ‘1’ (inactive)	Low
	ROMSN[1:0]	Output	PROM chip-select	Low
	RAMN	Output	Common SRAM chip-select. Asserted when one of the RAMSN[4:0] signals is asserted.	Low
	ROMN	Output	Common PROM chip-select. Asserted when one of the ROMSN[1:0] signals is asserted.	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0].	Low
	MBEN[3:0]	Output	Byte enable: MBEN[0] corresponds to DATA[31:24], MBEN[1] corresponds to DATA[23:16], MBEN[2] corresponds to DATA[15:8], MBEN[3] corresponds to DATA[7:0].	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus. Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0].	Low/High[2]
	VBDRIVE[31:0]	Output	Identical to BDRIVE but has one signal for each data bit. Every index is driven by its own register. This can be used to reduce the output delay.	Low/High[2]
	READ	Output	Read strobe	High
	SA[14:0]	Output	Not used	High

*Table 1201.*Signal descriptions

Signal name	Field	Type	Function	Polarity
AHBSI	1)	Input	AHB slave input signals	-
AHBSO	1)	Output	AHB slave output signals	-
SDO	SDCASN	Output	Not used. All signals are driven to inactive state.	Low

1) See GRLIB IP Library User's Manual

2) Polarity is selected with the oepol generic

94.9 Library dependencies

Table 1202 shows libraries used when instantiating the core (VHDL libraries).

*Table 1202.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

94.10 Component declaration

The core has the following component declaration.

```
component srctrl
  generic (
    hindex  : integer := 0;
    romaddr : integer := 0;
    rommask : integer := 16#ff0#;
    ramaddr : integer := 16#400#;
    rammask : integer := 16#ff0#;;
    ioaddr  : integer := 16#200#;
    iomask  : integer := 16#ff0#;
    ramws   : integer := 0;
    romws   : integer := 2;
    iows    : integer := 2;
    rmw     : integer := 0;-- read-modify-write enable
    prom8en : integer := 0;
    oepol   : integer := 0;
    srbanks : integer range 1 to 5 := 1;
    banksz  : integer range 0 to 13:= 13;
    romasel : integer range 0 to 27:= 19
  );
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;
    sri     : in  memory_in_type;
    sro     : out memory_out_type;
    sdo     : out sdctrl_out_type
  );
end component;
```

94.11 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined on the example designs port map and connected to the memory controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

Memory controller decodes default memory areas: PROM area is 0x0 - 0xFFFFFF and SRAM area is 0x40000000 - 0x40FFFFF. The 8-bit PROM mode is disabled. Two SRAM banks of size 64 Mbyte are used and the fifth chip select is disabled.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;
library esa;
use esa.memoryctrl.all;

entity srctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;

    -- memory bus
    address  : out   std_logic_vector(27 downto 0); -- memory bus
    data     : inout std_logic_vector(31 downto 0);
    ramsn    : out   std_logic_vector(4 downto 0);
    ramoen   : out   std_logic_vector(4 downto 0);
    rwen     : inout std_logic_vector(3 downto 0);
    romsn    : out   std_logic_vector(1 downto 0);
    iosn     : out   std_logic;
    oen      : out   std_logic;
    read     : out   std_logic;
    writen   : inout std_logic;
    brdyn    : in    std_logic;
    bexcn    : in    std_logic;
    modesel  : in    std_logic; --PROM width select
-- sdram i/f
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
       );
end;

architecture rtl of srctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo : sdctrl_out_type;

  signal wprot : wprot_out_type;  -- dummy signal, not used
  signal clkm, rstn : std_ulogic; -- system clock and reset

-- signals used by clock and reset generators
```

```
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;

  signal gnd : std_ulogic;

begin

  -- AMBA Components are defined here ...


  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                              tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);


  -- Memory controller
srctrl0 : srctrl generic map (rmw => 1, prom8en => 0, srbanks => 2,
  banksz => 13, ramsel5 => 0)
    port map (rstn, clkm, ahbsi, ahbso(0), memi, memo, sdo);


  -- I/O pads driving data memory bus data signals
  datapads : for i in 0 to 3 generate
      data_pad : iopadv generic map (width => 8)
      port map (pad => data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(i),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  -- Alternative I/O pad instantiation with vectored enable instead
  datapads : for i in 0 to 3 generate
      data_pad : iopadvv generic map (width => 8)
      port map (pad => data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(31-i*8 downto 24-i*8),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  -- connect memory controller outputs to entity output signals
  address <= memo.address; ramsn <= memo.ramsn; romsn <= memo.romsn;
  oen <= memo.oen; rwen <= memo.wrn; ramoen <= memo.ramoen;
  writen <= memo.writen; read <= memo.read; iosn <= memo.iosn;
  sdcke <= sdo.sdcke; sdwen <= sdo.sdwen; sdcsn <= sdo.sdcsn;
  sdrasn <= sdo.rasn; sdcasn <= sdo.casn; sddqm <= sdo.dqm;

end;
```

95 SSRCTRL- 32-bit SSRAM/PROM Controller

95.1 Overview

The memory controller (SSRCTRL) is an 32-bit SSRAM/PROM/IO controller that interfaces external Synchronous pipelined SRAM, PROM, and I/O to the AMBA AHB bus. The controller acts as a slave on the AHB bus and has a configuration register accessible through an APB slave interface. Figure 272 illustrates the connection between the different devices.

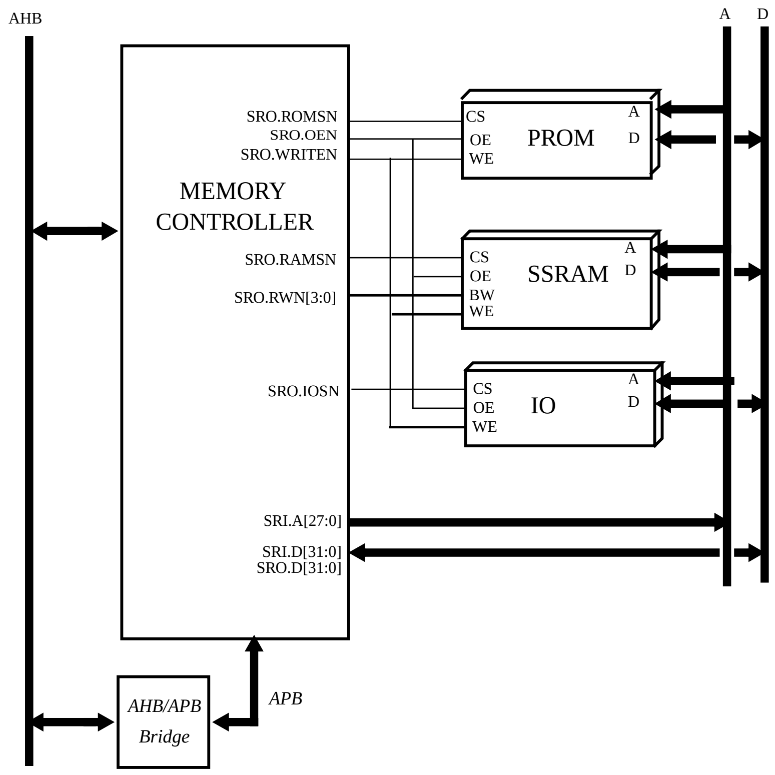


Figure 272. 32-bit SSRAM/PROM/IO controller

The controller is configured by VHDL-generics to decode three address ranges: PROM, SSRAM and I/O area. By default PROM area is mapped into address range 0x0 - 0x00FFFFFF; the SSRAM area is mapped into address range 0x40000000 - 0x40FFFFFF; and the I/O area is mapped to 0x20000000 - 0x20FFFFFF.

One chip select is generated for each of the address areas. The controller generates both a common write-enable signal (WRITEN) as well as four byte-write enable signals (WRN). The byte-write enable signal enables byte and half-word write access to the SSRAM.

A signal (BDRIVE) is provided for enabling the bidirectional pads to which the data signals are connected. The oepol generic is used to select the polarity of these enable signals. If output delay is an issue, a vectored output enable signal (VBDRIVE) can be used instead. In this case, each pad has its own enable signal driven by a separate register. A directive is placed on these registers so that they will not be removed during synthesis (in case the output they drive is used in the design).

The SSRCTRL conteoller can optionally support 16-bit PROM/IO devices. This is enabled through the BUS16 generic. A 32-bit access to the PROM or IO area will be translated into two 16-bit accesses with incrementing address.

95.2 SSRAM/PROM waveform

Because the SSRAM (Synchronous pipelined SRAM) has a pipelined structure, the data output has a latency of three clock cycles. The pipelined structure enables a new memory operation to be issued each clock cycle. Figure 272 and figure 273 show timing diagrams for the SSRAM read and write accesses.

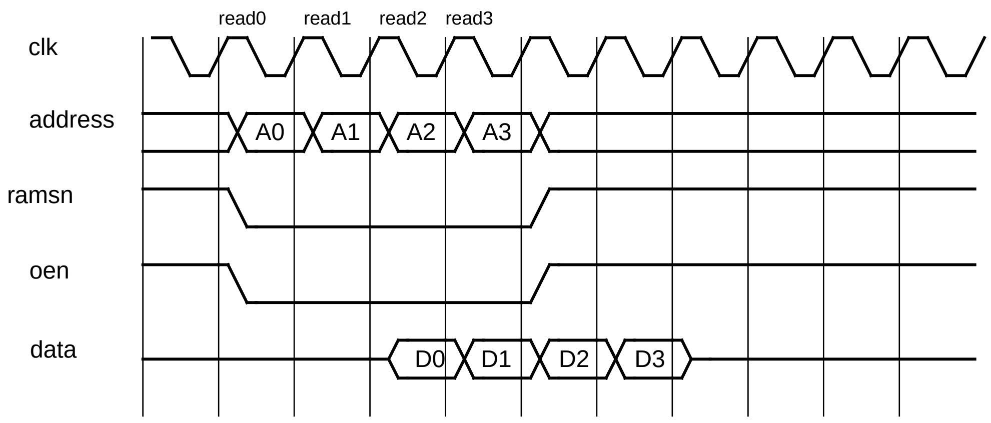


Figure 273. 32-bit SSRAM read cycle

As shown in the figure above, the controller always perform a burst read access to the memory. This eliminates all data output latency except for the first word when a burst read operation is executed.

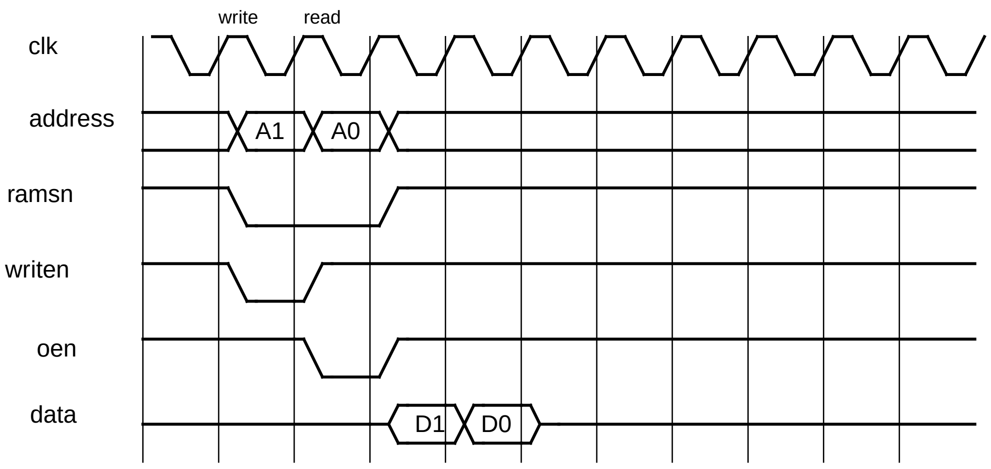


Figure 274. 32-bit SSRAM write cycle

A write operation takes three clock cycles. On the rising edge of the first clock cycle, the address and control signals are latched into the memory. On the next rising edge, the memory puts the data bus in high-impedance mode. On the third rising edge the data on the bus is latched into the memory and the write is complete. The controller can start a new memory (read or write) operation in the second clock cycle. In figure 274 this is illustrated by a read operation following the write operation.

Due to the memory automatically putting the data bus in high-impedance mode when a write operation is performed, the output-enable signal (OEN) is held active low during all SSRAM accesses (including write operations).

95.2.1 PROM and IO access

For the PROM and I/O operations, a number of waitstates can be inserted to increase the read and write cycle. The number of waitstates can be configured separately for the I/O and PROM address ranges, through a programmable register mapped into the APB address space. After a reset the waitstates for PROM area is set to its maximum (15). Figure 275 and figure 276 show timing diagrams for the PROM read and write accesses.

Read accesses to 32-bit PROM and I/O has the same timing, see figure 275

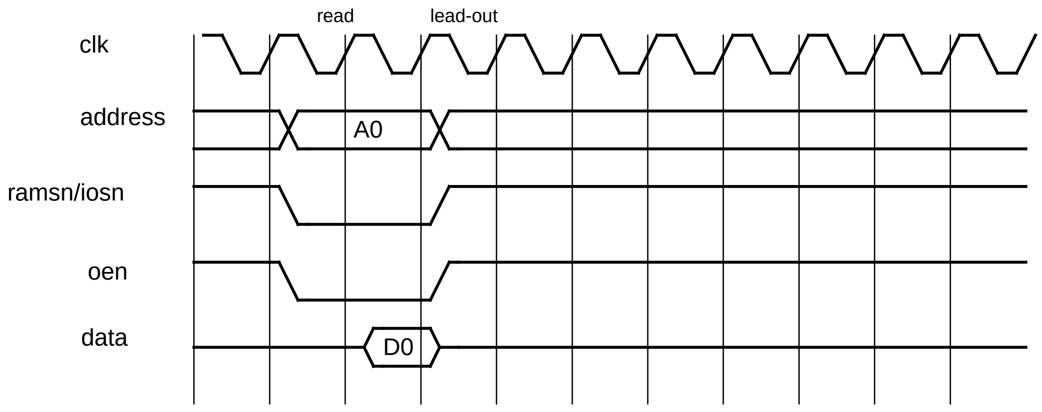


Figure 275. 32-bit PROM/IO read cycle

The write access for 32-bit PROM and I/O can be seen in figure 276

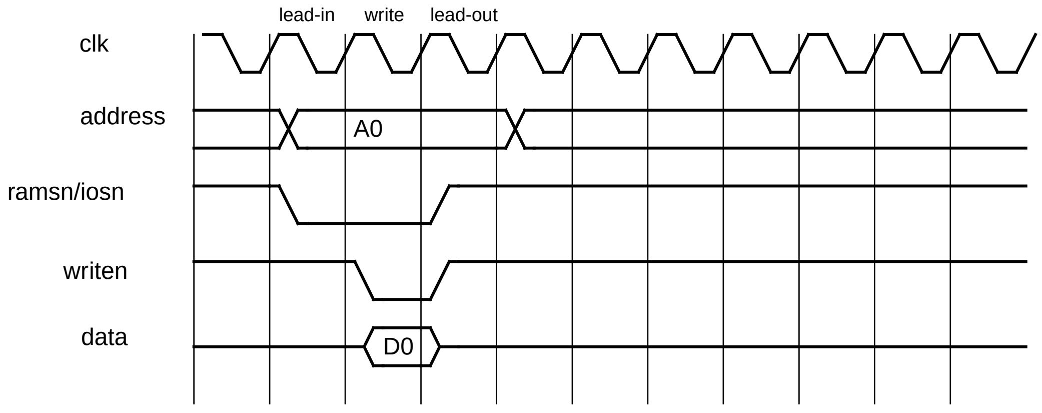


Figure 276. 32-bit PROM/IO write cycle

The SSRCTRL conteoller can optionally support 16-bit PROM/IO devices. This is enabled through the BUS16 generic. A 32-bit access to the PROM or IO area will be translated into two 16-bit accesses with incrementing address. A 16-bit access will result in one bus access only. 8-bit accesses are not allowed.

16-bit PROM/IO operation is enabled by writing “01” to the romwidth field in SSRAM control register. At reset, the romwidth field is set by the MEMI.BWIDTH input signal.

Read accesses to 16-bit PROM and I/O has the same timing, see figure 277

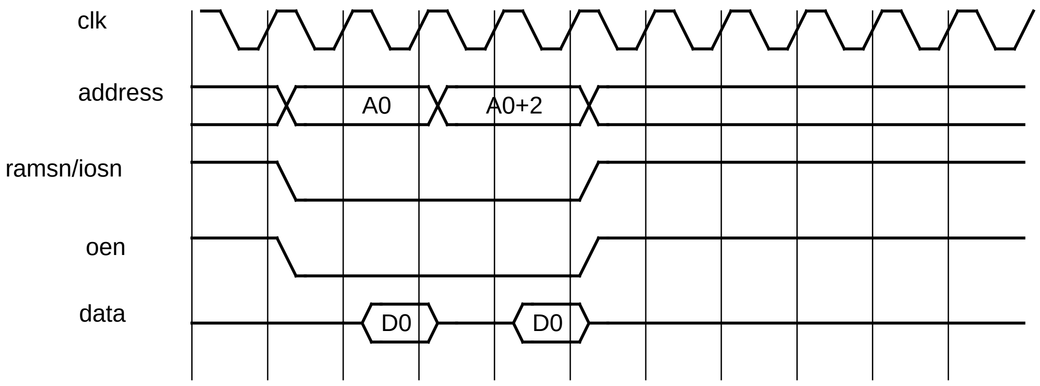


Figure 277. 32-bit PROM/IO read cycle in 16-bit mode

The write access for 32-bit PROM and I/O can be seen in figure 278

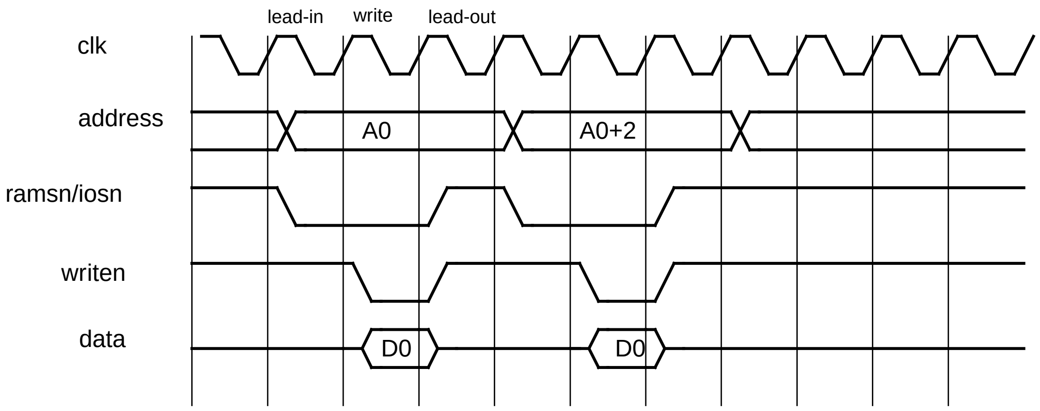


Figure 278. 32-bit PROM/IO write cycle in 16-bit mode

95.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 1203.*SSRAM controller registers

APB address offset	**Register**
0x00	Memory configuration register

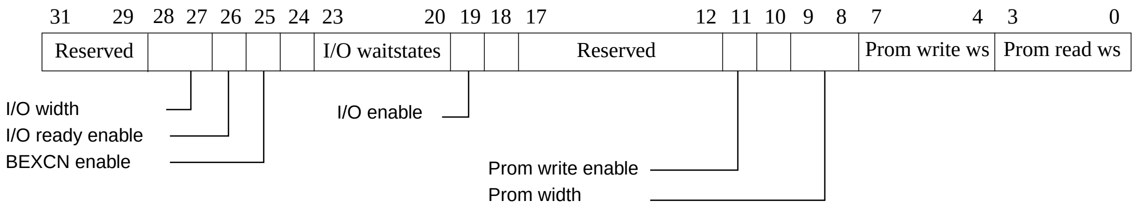


Figure 279. Memory configuration register

[3:0]: Prom read waitstates. Defines the number of waitstates during prom read cycles (“0000”=0, “0001”=1,... “1111”=15).

[7:4]:	Prom write waitstates. Defines the number of waitstates during prom write cycles (“0000”=0, “0001”=1,... “1111”=15). [9:8]: Prom width. Defines the data with of the prom area (“01”=16, “10”=32).
[10]:	Reserved
[11]:	Prom write enable. If set, enables write cycles to the prom area. NOT USED.
[17:12]:	Reserved
[19]:	I/O enable. If set, the access to the memory bus I/O area are enabled. NOT USED.
[23:20]:	I/O waitstates. Defines the number of waitstates during I/O accesses (“0000”=0, “0001”=1, “0010”=2,..., “1111”=15).
[25]:	Bus error (BEXCN) enable. NOT USED.
[26]:	Bus ready (BRDYN) enable. NOT USED.
[28:27]:	I/O bus width. Defines the data with of the I/O area (“01”=16, “10”=32).

During power-up (reset), the PROM waitstates fields are set to 15 (maximum) and the PROM bus width is set to the value of MEMI.BWIDTH. All other fields are initialized to zero.

95.4 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x00A. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

95.5 Configuration options

Table 1204 shows the configuration options of the core (VHDL generics).

*Table 1204.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0xFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#FF0#
ramaddr	ADDR field of the AHB BAR1 defining RAM address space. Default RAM area is 0x40000000-0x40FFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR1 defining RAM address space.	0 -16#FFF#	16#FF0#
ioaddr	ADDR field of the AHB BAR2 defining IO address space. Default IO area is 0x20000000-0x20FFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR2 defining IO address space.	0 -16#FFF#	16#FF0#
paddr	ADDR field of the APB BAR configuration registers address space.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR configuration registers address space.	0 - 16#FFF#	16#FFF#
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
bus16	Enable support for 16-bit PROM/IO accesses	0 - 1	0

95.6 Signal descriptions

Table 1205 shows the interface signals of the core (VHDL ports).

*Table 1205.*Signal descriptions

Signal name	Field	Type	Function	Polarity
CLK	N/A	Input	Clock	-

*Table 1205.*Signal descriptions

Signal name	Field	Type	Function	Polarity
RST	N/A	Input	Reset	Low
SRI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Not used	-
	BEXCN	Input	Not used	-
	WRN[3:0]	Input	Not used	-
	BWIDTH[1:0]	Input	PROM bus width at reset	-
	SD[63:0]	Input	Not used	-
	CB[7:0]	Input	Not used	-
	SCB[7:0]	Input	Not used	-
	EDAC	Input	Not used	-
SRO	ADDRESS[27:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data	High
	SDDATA[63:0]	Output	Not used	-
	RAMSN[7:0]	Output	SSRAM chip-select, only bit 0 is used	Low
	RAMOEN[7:0]	Output	Same as OEN	Low
	IOSN	Output	I/O chip-select	Low
	ROMSN[7:0]	Output	PROM chip-select, only bit 0 is used	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SSRAM byte write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0].	Low
	MBEN[3:0]	Output	Not used	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus. Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0]. Any BDRIVE[ ] signal can be used for CB[ ].	Low/High^2
	VBDRIVE[31:0]	Output	Identical to BDRIVE but has one signal for each data bit. Every index is driven by its own register. This can be used to reduce the output delay.	Low/High^2
	SVBDRIVE	Output	Not used	-
	READ	Output	Not used	-
	SA[14:0]	Output	Not used	-
	CB[7:0]	Output	Not used	-
	SCB[7:0]	Output	Not used	-
	VCDRIVE[7:0]	Output	Not used	-
	SVCDRIVE[7:0]	Output	Not used	-
	CE	Output	Not used	-
AHBSI	1)	Input	AHB slave input signals	-

*Table 1205.*Signal descriptions

Signal name	Field	Type	Function	Polarity
AHBSO	1)	Output	AHB slave output signals	-
APBI	1)	Input	APB slave input signals	-
APBO	1)	Output	APB slave output signals	-

1) See GRLIB IP Library User’s Manual

2) Polarity is selected with the oepol generic

95.7 Library dependencies

Table 1206 shows libraries used when instantiating the core (VHDL libraries).

*Table 1206.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

95.8 Component declaration

The core has the following component declaration.

```
component ssrctrl
  generic (
    hindex  : integer := 0;
    pindex  : integer := 0;
    romaddr : integer := 0;
    rommask : integer := 16#ff0#;
    ramaddr : integer := 16#400#;
    rammask : integer := 16#ff0#;
    ioaddr  : integer := 16#200#;
    iomask  : integer := 16#ff0#;
    paddr   : integer := 0;
    pmask   : integer := 16#fff#;
     oepol   : integer := 0;
     bus16 : integer := 0
  );
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;
    apbi    : in  apb_slv_in_type;
    apbo    : out apb_slv_out_type;
    sri     : in  memory_in_type;
    sro     : out memory_out_type

  );
end component;
```

95.9 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it, including the memory controller. The external memory bus is defined in the example designs port map and connected to the memory controller. System clock and reset are generated by the Clkgen_ml401 Clock Generator and GR Reset Generator.

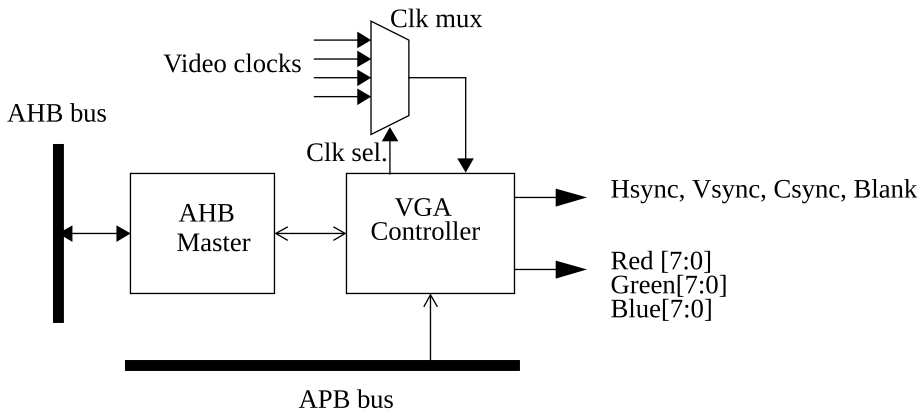
The memory controller decodes default memory areas: PROM area is 0x0 - 0x00FFFFFF, I/O-area is 0x20000000-0x20FFFFFF and RAM area is 0x40000000 - 0x40FFFFFF.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib, techmap;
use grlib.amba.all;
use grlib.stdlib.all;
use techmap.gencomp.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.misc.all;

entity ssrctrl_ex is
 port (
    sys_rst_in: in  std_ulogic;
    sys_clk: in  std_ulogic; -- 100 MHz main clock
    sram_flash_addr : out std_logic_vector(22 downto 0);
    sram_flash_data : inout std_logic_vector(31 downto 0);
    sram_cen  : out std_logic;
    sram_bw   : out std_logic_vector (0 to 3);
    sram_flash_oe_n : out std_ulogic;
    sram_flash_we_n : out std_ulogic;
    flash_ce  : out std_logic;
    sram_clk  : out std_ulogic;
    sram_clk_fb: in  std_ulogic;
    sram_mode : out std_ulogic;
    sram_adv_ld_n : out std_ulogic;
    sram_zz : out std_ulogic;
    iosn    : out std_ulogic;
);
end;

architecture rtl of ssrctrl_ex is

-- Clock generator component
component clkgen_ml401
  generic (
    clk_mul  : integer := 1;
    clk_div  : integer := 1;
    freq     : integer := 100000);-- clock frequency in KHz
  port (
    clkin   : in  std_logic;
    clk     : out std_logic;-- main clock
    ddrclk  : out std_logic;-- DDR clock
    ddrclkfb: in  std_logic;-- DDR clock feedback
    ddrclk90  : out std_logic;-- DDR 90 clock
    ddrclk180 : out std_logic;-- 180 clock
    ddrclk270 : out std_logic;-- DDR clock
    ssrclk  : out std_logic;-- SSRAM clock
    ssrclkfb: in  std_logic;-- SSRAM clock feedback
    cgi     : in clkgen_in_type;
    cgo     : out clkgen_out_type);
end component;


-- signals used to connect memory controller and memory bus
signal memi  : memory_in_type;
signal memo  : memory_out_type;

-- AMBA bus (AHB and APB)
signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi : ahb_mst_in_type;
signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

-- Signals used by clock and reset generators
signal clkm, rstn, rstraw, srclkl : std_ulogic;
signal cgi   : clkgen_in_type;
signal cgo   : clkgen_out_type;
signal ddrclkfb, ssrclkfb, ddr_clkl, ddr_clknl : std_ulogic;


begin

  clkgen0 : clkgen_ml401  -- clock generator
  port map (sys_clk, clkm, ddr_clkl, ddrclkfb, open, ddr_clknl, open, sram_clk,
      sram_clk_fb, cgi, cgo);

  rst0 : rstgen-- reset generator
  port map (sys_rst_in, clkm, cgo.clklock, rstn, rstraw);

  -- AMBA Components are defined here ...

  -- Memory controller
  mctrl0 : ssrctrl generic map (hindex => 0, pindex => 0)
  port map (rstn, clkm, ahbsi, ahbso(0), apbi, apbo(0), memi, memo);

  -- connect memory controller outputs to entity output signals
  sram_adv_ld_n <=  ’0’; sram_mode <=  ’0’; sram_zz <= ’0’;
  sram_flash_addr <= memo.address(24 downto 2); sram_cen <= memo.ramsn(0);
  flash_ce <= memo.romsn(0); sram_flash_oe_n <= memo.oen; iosn <= memo.iosn;
  sram_bw <= memo.wrn; sram_flash_we_n <= memo.writen;

  -- I/O pad instantiation with vectored enable instead
  bdr : for i in 0 to 31 generate
      data_pad : iopad generic map (tech => padtech)
      port map (sram_flash_data(i), memo.data(i),
             memo.vbdrive(i), memi.data(i));
  end generate;

end;
```

96 SVGACTRL - VGA Controller Core

96.1 Overview

The core is a pixel based video controller (frame buffer), capable of displaying standard and custom resolutions with variable bit depth and refresh rates. The video controller consists of a synchronization unit, main control unit, FIFO unit and an AHB master as shown in the figure below.



96.2 Operation

The core uses external frame buffer memory located in the AHB address space. A frame on the display is created by fetching the pixel data from memory and sending it to the screen through an external DAC using three 8-bit color vectors. To hide the AHB bus latency, the pixel data is buffered in a FIFO inside the core. The start address of the frame buffer is specified in the Frame buffer Memory Position register, and can be anywhere in the AHB address space. In addition to the color vectors the video controller also generates HSYNC, VSYNC, CSYNC and BLANK signals control signals.

The video timing is programmable through the Video Length, Front Porch, Sync Length and Line Length registers. The bit depth selection and enabling of the controller is done through the status register. These values make it possible to display a wide range of resolutions and refresh rates.

The pixel clock can be either static or dynamic multiplexed. The frequency of the pixel clock is calculated as *Horizontal Line Length * Vertical Line Length * refresh rate*. When using a dynamically multiplexed clock, bits [5:4] in the status register are used to control the clock selector. The dynamic pixel clocks should be defined in the core’s VHDL generics to allow software to read out the available pixel clock frequencies.

The core can use bit depths of 8, 16 and 32 bits. When using 32 bits, bits[23:0] are used, when 16 bits a [5,6,5] color scheme is used and when using 8 bits a color lookup table “CLUT” is used. The CLUT has 256 positions, each 24 bits wide, and the 8 bit values read from memory are used to index the CLUT to obtain the actual color.

96.3 DVI support

In order to initialize a DVI transmitter, an additional core such as the I^2C master is normally required. Additional glue logic may also be required since the interfaces of DVI transmitters differ between manufacturers and product lines. Examples on how to interface the core to a DVI transmitter are available in the GRLIB IP Library’s template designs.

96.4 Registers

The core is programmed through registers mapped into APB address space.

*Table 1207.*VGA controller registers

APB address offset	Register
0x00	Status register
0x04	Video length register
0x08	Front Porch register
0x0C	Sync Length register
0x10	Line Length register
0x14	Framebuffer Memory Position register
0x18	Dynamic Clock 0 register
0x1C	Dynamic Clock 1 register
0x20	Dynamic Clock 2 register
0x24	Dynamic Clock 3 register
0x28	CLUT Access register

Table 1208. VGA controller Status register

31 10	9	8	7 6	5 4	3	2	1	0
RESERVED	VPOL	HPOL	CLKSEL	BDSEL	VR	RES	RST	EN

31:10	RESERVED
9	V polarity (VPOL)- Sets the polarity for the vertical sync pulse.
8	H polarity (HPOL) - Sets the polarity for the horizontal sync pulse.
7:6	Clock Select (CLKSEL) Clock selector when using dynamic pixelclock
5:4	Bit depth selector (BDSEL) - “01” = 8-bit mode; “10” = 16-bit mode; “11” = 32-bit mode
3	Vertical refresh (VR) - High during vertical refresh
2	RESERVED
1	Reset (RST) - Resets the core
0	Enable (EN) - Enables the core

Table 1209. VGA controller Video Length register

31 16	15 0
VRES	HRES

31:16	Vertical screen resolution (VRES) - Vertical screen resolution in pixels -1
15:0	Horisontal screen resolution (HRES) - Horizontal screen resolution in pixels -1.

Table 1210. VGA controller Front porch register

31 16	15 0
VPORCH	HPORCH

31:16	Vertical front porch (VPORCH) - Vertical front porch in pixels.
15:0	Horisontal front porch (HPORCH) - Horizontal front porch in pixels.

Table 1211. VGA controller Sync pulse register

31 16	15 0
VPLEN	HPLEN

31:16	Vertical sync pulse length (VPLEN) - Vertical sync pulse length in pixels.
15:0	Horisontal sync pulse length (HPLEN) - Horizontal sync pulse length in pixels.

Table 1212. VGA controller Line Length register

31 16	15 0
VLLEN	HLLEN

31:16	Vertical line length (VLLEN) - The length of the total line with front and back porch, sync pulse length and vertical screen resolution.
15:0	Horisontal line length (HLLEN) - The length of the total line with front and back porch, sync pulse length and horizontal screen resolution,

Table 1213. VGA controller Framebuffer Memory Position register

31 0
FMEM

31:0	Framebuffer memory position (FMEM) - Holds the memory position of the framebuffer, must be aligned on a 1 Kbyte boundary.

Table 1214. VGA controller Dynamic clock 0 register

31 0
CLK0

31:0	Dynamic pixel clock 0 (CLK0) - Dynamic pixel clock defined in ps.

Table 1215. VGA controller Dynamic clock 1 register

31 0
CLK1

31:0	Dynamic pixel clock 1 (CLK1) - Dynamic pixel clock defined in ps.

Table 1216. VGA controller Dynamic clock 2 register

31 0
CLK2

31:0	Dynamic pixel clock 2 (CLK2) - Dynamic pixel clock defined in ps.

Table 1217. VGA controller Dynamic clock 3 register

31 ... 0
CLK3

31:0	Dynamic pixel clock 3 (CLK3) - Dynamic pixel clock defined in ps.

Table 1218. VGA controller CLUT Access register

31 ... 24	23 ... 16	15 ... 6	7 ... 0
CREG	RED	GREEN	BLUE

31:24	Color lookup table register (CREG) - Color lookup table register to set.
23:16	Red color data (RED) - Red color data to set in the specified register.
15:8	Green color data (GREEN) - Green color data to set in the specified register.
7:0	Blue color data (BLUE) - Blue color data to set in the specified register.

96.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x063. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

96.6 Configuration options

Table 1219 shows the configuration options of the core (VHDL generics).

*Table 1219.*Configuration options

Generic name	**Function**	**Allowed range**	**Default**
length	Size of the pixel FIFO	3 - 1008	384
part	Pixel FIFO part length	1 - 336	128
memtech	Memory technology	0 - NTECH	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	12-bit MSB APB address	0 - 16#FFF#	0
pmask	APB address mask	0 - 16#FFF#	16#FFF#
hindex	AHB master index	0 - NAHBMST-1	0
hirq	Interrupt line	0 - NAHBIRQ-1	0
clk0	Period of dynamic clock 0 in ps	0- 16#FFFFFFFF#	40000
clk1	Period of dynamic clock 1 in ps	0- 16#FFFFFFFF#	20000
clk2	Period of dynamic clock 2 in ps	0- 16#FFFFFFFF#	15385
clk3	Period of dynamic clock 3 in ps	0- 16#FFFFFFFF#	0
burstlen	AHB burst length. The core will burst $2^{burstlen}$ words.	2 - 8	8
ahbaccsz	Determines the size of the AMBA accesses that the core will use when fetching data from memory.	32 - AHBDW	32
ayncrst	Use asynchronous reset for the VGA clock domain. If this generic is set to 1 the core will use the *arst* input to reset part of the registers in the VGA domain. Asynchronous reset should be used if the VGA clock is not available during system reset. If this generic is 0 the *arst* input is not used.	0 - 1	0

96.7 Signal descriptions

Table 1220 shows the interface signals of the core (VHDL ports).

*Table 1220.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	System clock	-
VGACLK	N/A	Input	Pixel clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
VGAO	HSYNC	Output	Horizontal sync	-
	VSYNC	Output	Vertical sync	-
	COMP_SYNC	Output	Composite sync	-
	BLANK	Output	Blanking	-
	VIDEO_OUT_R[7:0]	Output	Video out, red.	-
	VIDEO_OUT_G[7:0]	Output	Video out, green.	-
	VIDEO_OUT_B[7:0]	Output	Video out, blue.	-
	BITDEPTH[1:0]	Output	Value of Status register’s BDSEL field	-
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
CLK_SEL[1:0]	N/A	Output	2-bit clock selector	-
ARST	N/A	Input	Asynchronous reset input	Low

* see GRLIB IP Library User’s Manual

96.8 Library dependencies

Table 1221 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1221.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component, signals	Component and signal definitions.

96.9 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
library Gaisler;
use gaiser.misc.all;
.
architecture rtl of test is
signal apbi : apb_slv_in_type;
signal apbo : apb_slv_out;
signal vgao : apbvga_out_type;
signal ahbi : ahb_mst_in_type;
signal ahbo : ahb_mst_out_type;
signal clk_sel :std_logic_vector(1 downto 0));
signal clkmvga : std_logic;
begin
.
.
```

```
-- VGA Controller
  vga0 : svgactrl
  generic map(memtech => memtech, pindex => 6, paddr => 6, hindex => 6,
    clk0 => 40000, clk1 => 20000, clk2 => 15385, clk3 => 0)
  port map(rstn,clkm,clkmvga, apbi, apbo(6), vgao,ahbmi,ahbmo(6),clk_sel);
end;
```

96.10 Linux 2.6 driver

A video driver for the core is provided Snapgear Linux (-p27 and later). The proper kernel command line options must be used for the driver to detect the core. Please see the SnapGear Linux for LEON manual for further information.

97 SYNCRAM - Single-port RAM generator

97.1 Overview

SYNCRAM is a single port RAM that maps on technology-specific RAM blocks. The core has a common address bus, and separate data-in and data-out buses. All inputs are latched on the on the rising edge of clk. The read data appears on dataout directly after the clk rising edge.

97.2 Configuration options

Table 1222 shows the configuration options of the core (VHDL generics).

*Table 1222.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table below	-
dbits	Data width	see table below	-
testen	Enable bypass logic for scan testing	0 - 1	0

97.3 Scan test support

Scan test support will be enabled if the TESTEN generic is set to 1. This option will generate a register (flip-flops) connected between the DATAIN and DATAOUT of the syncram module. In test mode, DATAOUT is driven from the register rather then from the RAM outputs. This will allow both input and output paths around the syncram to be testable by scan. The address bus and control signals are xored with the DATAIN signal to also increase test coverage of those. Test mode is enaled by driving the TESTIN(3) signals to 1. This signal should typically be connected to the global test enable signals of the design.

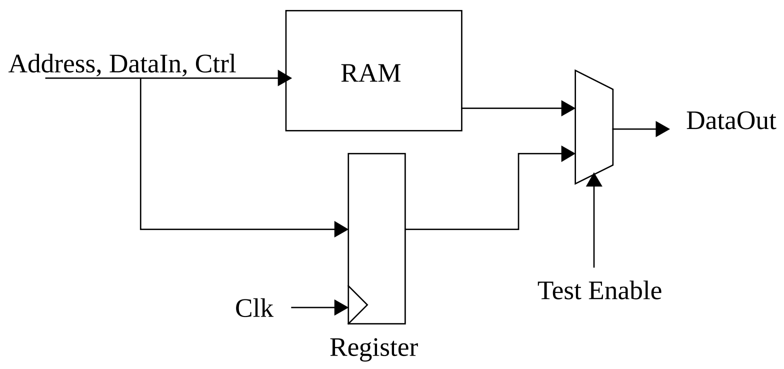


Figure 280. Scan test support

Table 1223 shows the supported technologies for the core.

*Table 1223.*Supported technologies

Tech name	**Technology**	**RAM cell**	**abit range**	**dbit range**
altera	All Altera devices	altsyncram	unlimited	unlimited
ihp15	IHP 0.25	sram2k (512x32)	2 - 9	unlimited
inferred	Behavioral description	Tool dependent	unlimited	unlimited
virtex	Xilinx Virtex, VirtexE, Spartan2	RAMB4_Sn	unlimited	unlimited
virtex2	Xilinx Virtex2/4/5/6, Spartan3/3a/3e/6	RAMB16_Sn	unlimited	unlimited
axcel / axdsp	Actel AX, RTAX and RTAX-DSP	RAM64K36	2 - 12	unlimited
proasic	Actel Proasic	RAM256x9SST	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9, ram512x18	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	sp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss1_128x32cm4sw0 hdss1_256x32cm4sw0 hdss1_512x32cm4sw0 hdss1_1024x32cm8sw0	7 - 11	32
memartisan	Artisan ASIC RAM	sp_256x32m32 sp_512x32m32 sp_1kx32m32 sp_2kx32m32 sp_4kx32m32 sp_8kx32m32 sp_16kx32m32	8 - 14	32
memvirage90	Virage 90 nm ASIC RAM	SPRAM_HS_32x30 SPRAM_HS_128x32 SPRAM_HS_256x32 SPRAM_HS_1024x32	2 - 10	128
eclipse	Aeroflex/Quicklogic FPGA	RAM128x18_25um RAM256X9_25um RAM512X4_25um RAM1024X2_25um	2 - 10	unlimited
easic90	eASIC 90 nm Nextreme	eram, bram	2 - 15	unlimited
easic45	eASIC 45 nm Nextreme2	bRAM, rFile	unlimited	unlimited

97.4 Signal descriptions

Table 1224 shows the interface signals of the core (VHDL ports).

*Table 1224.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
CLK	N/A	Input	Clock. All input signals are latched on the rising edge of the clock.	-
ADDRESS	N/A	Input	Address bus. Used for both read and write access.	-
DATAIN	N/A	Input	Data inputs for write data	-
DATAOUT	N/A	Output	Data outputs for read data	-
ENABLE	N/A	Input	Chip select	High
WRITE	N/A	Input	Write enable	High
TESTIN		Input	Test inputs (see text)	High

97.5 Library dependencies

Table 1225 shows libraries used when instantiating the core (VHDL libraries).

*Table 1225.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology contants

97.6 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

  component syncram
  generic (tech : integer := 0; abits : integer := 6; dbits : integer := 8);
  port (
    clk      : in std_ulogic;
    address  : in std_logic_vector((abits -1) downto 0);
    datain   : in std_logic_vector((dbits -1) downto 0);
    dataout  : out std_logic_vector((dbits -1) downto 0);
    enable   : in std_ulogic;
    write    : in std_ulogic;
    testin   : in std_logic_vector(3 downto 0) := "0000");
  end component;
```

97.7 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;
.

clk      : std_ulogic;
address  : std_logic_vector((abits -1) downto 0);
datain   : std_logic_vector((dbits -1) downto 0);
dataout  : std_logic_vector((dbits -1) downto 0);
enable   : std_ulogic;
write    : std_ulogic);

ram0 : syncram generic map ( tech => tech, abits => addrbits, dbits => dbits)
      port map ( clk, addr, datain, dataout, enable, write);
```

98 SYNCRAMBW - Single-port RAM generator with byte enables

98.1 Overview

SYNCRAMBW implements a single port RAM with byte enables, using the GRLIB technology wrapping for different target technologies. The core operates identically to SYNCRAM, with the addition that each byte has a separate chip select (ENABLE) and write select (WRITE). The core is provided in a generic configuration and also in configurations of 128, 156 and 256 bits, and the corresponding entities are named SYNCRAMBW, SYNCRAM128BW, SYNCRAM156BW and SYNCRAM256BW. In the simplest case, the IP cores just instantiate several eight bit wide SYNCRAM components. SYNCRAM128BW, SYNCRAM156BW and SYNCRAM256BW, used in GRLIB's Level-2 cache core, contain specialized maps for several technologies to more efficiently utilize device resources.

Note that some SYNCRAM components may be missing from the library depending on the type of GRLIB distribution.

98.2 Configuration options

Table 1226 shows the configuration options of the core (VHDL generics).

*Table 1226.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table below	-
testen	Enable bypass logic for scan testing	0 - 1	0

98.3 Scan test support

Scan test support will be enabled if the TESTEN generic is set to 1. This option will generate a register (flip-flops) connected between the DATAIN and DATAOUT of the syncram module. In test mode, DATAOUT is driven from the register rather then from the RAM outputs. This will allow both input and output paths around the syncram to be testable by scan. The address bus and control signals are xored with the DATAIN signal to also increase test coverage of those. Test mode is enaled by driving the TESTIN(3) signals to 1. This signal should typically be connected to the global test enable signals of the design.

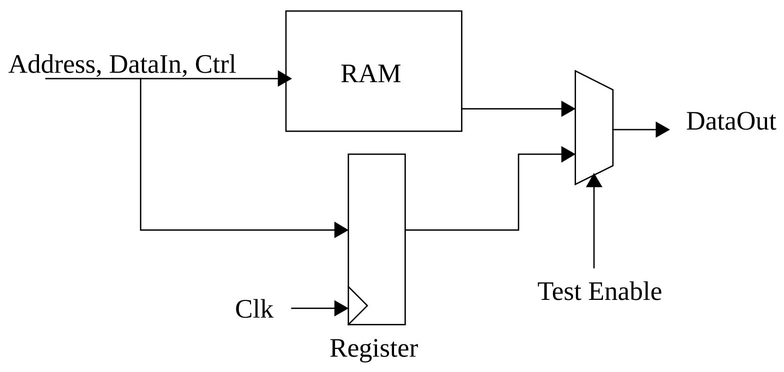


Figure 281. Scan test support

98.4 Technology support

Table 1227 shows the supported technologies for the core.

*Table 1227.*Supported technologies

Tech name	**Technology**	**RAM cell**	**abit range**	**dbit range**
altera	All Altera devices	altsyncram	unlimited	unlimited
inferred	Behavioral description	Tool dependent	unlimited	unlimited
virtex2/4/5/6 Spartan3/3a/3e/6	Xilinx Virtex2/4/5/6, Spartan3/3a/3e/6	RAMB16_Sn	unlimited	unlimited
all others	-	syncram core with dwidth=8	tech depend.	tech depend.

To add support for a new technology, the following steps should be taken:

- Add technology-specific version for the RAM core in lib/techmap/TECH
- Instantiate the technology-specific RAM core in lib/techmap/maps/syncram256bw.vhd, and set the has_sram256bw() constant to 1 for the specific technology:

```
constant has_sram256bw : tech_ability_type := (
virtex2 => 1, virtex4 => 1, virtex5 => 1, spartan3 => 1,
spartan3e => 1, spartan6 => 1, virtex6 => 1,
altera => 1, cyclone3 => 1, stratix2 => 1, stratix3 => 1,
tm65gpl => 0, others => 0);
```

See also syncrambw.vhd, syncram128bw.vhd and syncram156bw.vhd under lib/techmap/maps/ for the corresponding SYNCRAM BW IP cores.

98.5 Signal descriptions

Table 1228 shows the interface signals of the core (VHDL ports).

*Table 1228.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
CLK	N/A	Input	Clock. All input signals are latched on the rising edge of the clock.	-
ADDRESS	N/A	Input	Address bus. Used for both read and write access.	-
DATAIN	N/A	Input	Data inputs for write data	-
DATAOUT	N/A	Output	Data outputs for read data	-
ENABLE	N/A	Input	Byte Chip select	High
WRITE	N/A	Input	Byte Write enable	High
TESTIN		Input	Test inputs (see text)	High

98.6 Library dependencies

Table 1229 shows libraries used when instantiating the core (VHDL libraries).

*Table 1229.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
TECHMAP	GENCOMP	Constants	Technology contants

98.7 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

  component syncram_bw128
  generic (tech : integer := 0; abits : integer := 6);
  port (
    clk      : in std_ulogic;
    address  : in std_logic_vector((abits -1) downto 0);
    datain   : in std_logic_vector(127 downto 0);
    dataout  : out std_logic_vector(127 downto 0);
    enable   : in std_logic_vector(15 downto 0);
    write    : in std_logic_vector(15 downto 0);
    testin   : in std_logic_vector(3 downto 0) := "0000");
  end component;

  component syncram_bw256
  generic (tech : integer := 0; abits : integer := 6);
  port (
    clk      : in std_ulogic;
    address  : in std_logic_vector((abits -1) downto 0);
    datain   : in std_logic_vector(255 downto 0);
    dataout  : out std_logic_vector(255 downto 0);
    enable   : in std_logic_vector(31 downto 0);
    write    : in std_logic_vector(31 downto 0);
    testin   : in std_logic_vector(3 downto 0) := "0000");
  end component;
```

98.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
```

```
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;
.

clk      : std_ulogic;
address  : std_logic_vector(9 downto 0);
datain   : std_logic_vector(255 downto 0);
dataout  : std_logic_vector(255 downto 0);
enable   : std_logic_vector(31 downto 0);
write    : std_logic_vector(31 downto 0);

ram0 : syncram generic map ( tech => tech, abits => 10)
      port map ( clk, addr, datain, dataout, enable, write);
```

99 SYNCRAM_2P - Two-port RAM generator

99.1 Overview

The two-port RAM generator has a one read port and one write port. Each port has a separate address and data bus. All inputs are registered on the rising edge of clk. The read data appears on dataout directly after the clk rising edge. Address width, data width and target technology is parametrizable through generics.

99.2 Write-through operation

Write-through is supported if the function *syncram_2p_write_through(tech)* returns 1 for the target technology, or if the *wrfst* generic is set to 1. If *wrfst* = 1, additional logic will be generated to detect simultaneous read/write to the same memory location, and in that case bypass the written data to the data outputs.

99.3 Conflicts

Some technologies will produce unpredictable results when a read and write operation occurs simultaneously to the same memory location. The function *syncram_2p_dest_rw_collision(tech)* returns 1 for technologies that has this characteristic. If SYNCRAM_2P is implemented with *sepclk* = 0 then logic will be included that disables the read enable signal, if needed, when a collision is detected. If the core is implemented with *sepclk* = 1 (and *syncram_2p_dest_rw_collision(tech)* returns 1) then collision avoidance must be handled by external logic.

99.4 Scan test support

Scan test support will be enabled if the TESTEN generic is set to 1. This option will generate a register (flip-flops) connected between the DATAIN and DATAOUT of the syncram module. In test mode, DATAOUT is driven from the register rather then from the RAM outputs. This will allow both input and output paths around the syncram to be testable by scan. The address bus and control signals are xored with the DATAIN signal to also increase test coverage of those. Test mode is enaled by driving the TESTIN(3) signals to 1. This signal should typically be connected to the global test enable signals of the design.

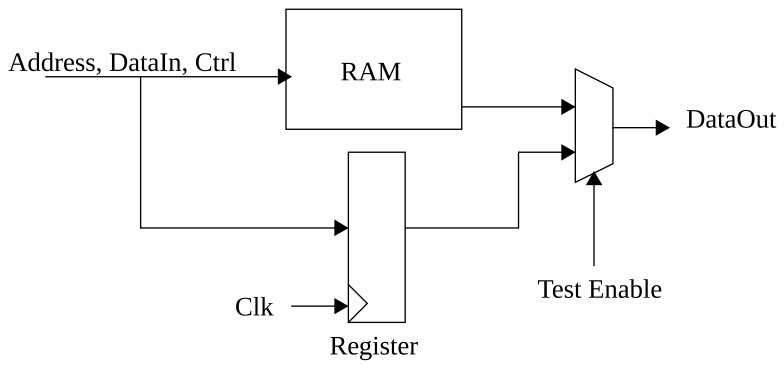


Figure 282. Scan test support

99.5 Configuration options

Table 1230 shows the configuration options of the core (VHDL generics).

*Table 1230.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table below	-
dbits	Data width	see table below	-
sepclk	If 1, separate clocks (rclk/wclk) are used for the two ports. If 0, rclk is used for both ports.	0 - 1	0
wrfst	Enable bypass logic for write-through operation. Can only be enabled for sepclk = 0.	0 - 1	0
testen	Enable bypass logic for scan testing	0 - 1	0

Table 1231 shows the supported technologies for the core.

*Table 1231.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
Inferred	Behavioural description	Tool dependent	unlimited	unlimited
altera	All Altera devices	altsyncram	umlimited	unlimited
virtex	Xilinx Virtex, Virtex-E, Spartan-2	RAMB4_Sn	2 - 10	unlimited
virtex2	Xilinx Virtex2/4/5/6 Spartan3/3a/3e/6	RAMB16_Sn	2 - 14	unlimited
axcel / axdsp	Actel AX, RTAX and RTAX-DSP	RAM64K36	2 - 12	unlimited
proasic	Actel Proasic	RAM256x9SST	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9, ram512x18	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	dp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss2_64x32cm4sw0 hdss2_128x32cm4sw0 hdss2_256x32cm4sw0 hdss2_512x32cm4sw0	6 - 9	32
memartisan	Artisan ASIC RAM	rf2_256x32m4 rf2_512x32m4	8 - 9	32
eclipse	Aeroflex/Quicklogic FPGA	RAM128x18_25um RAM256X9_25um RAM512X4_25um RAM1024X2_25um	2 - 10	unlimited
easic90	eASIC 90 nm Nextreme	eram	2 - 12	unlimited
easic45	eASIC 45 nm Nextreme2	bRAM, rFile	unlimited	unlimited

99.6 Signal descriptions

Table 1232 shows the interface signals of the core (VHDL ports).

*Table 1232.*Signal descriptions

Signal name	Type	Function	Active
RCLK	Input	Read port clock	-
RENABLE	Input	Read enable	High
RADDRESS	Input	Read address bus	-
DATAOUT	Output	Data outputs for read data	-
WCLK	Input	Write port clock	-
WRITE	Input	Write enable	High
WADDRESS	Input	Write address	-
DATAIN	Input	Write data	-
TESTEN	Input	Test inputs (see text)	High

99.7 Library dependencies

Table 1233 shows libraries used when instantiating the core (VHDL libraries).

*Table 1233.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology contants

99.8 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

component syncram_2p
  generic (tech : integer := 0; abits : integer := 6; dbits : integer := 8; sepclk : integer
:= 0);
  port (
    rclk     : in std_ulogic;
    renable  : in std_ulogic;
    raddress : in std_logic_vector((abits -1) downto 0);
    dataout  : out std_logic_vector((dbits -1) downto 0);
    wclk     : in std_ulogic;
    write    : in std_ulogic;
    waddress : in std_logic_vector((abits -1) downto 0);
    datain   : in std_logic_vector((dbits -1) downto 0);
    testin   : in std_logic_vector(3 downto 0) := "0000");
  end component;
```

99.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;

rclk     : in std_ulogic;
renable  : in std_ulogic;
raddress : in std_logic_vector((abits -1) downto 0);
```

```
dataout  : out std_logic_vector((dbits -1) downto 0);
wclk     : in std_ulogic;
write    : in std_ulogic;
waddress : in std_logic_vector((abits -1) downto 0);
datain   : in std_logic_vector((dbits -1) downto 0));

ram0 : syncram_2p generic map ( tech => tech, abits => addrbits, dbits => dbits)
      port map ( rclk, renable, raddress, dataout, wclk, write, waddress, datain, enable,
write);
```

100 SYNCRAM_DP - Dual-port RAM generator

100.1 Overview

The dual-port RAM generator has two independent read/write ports. Each port has a separate address and data bus. All inputs are latched on the on the rising edge of clk. The read data appears on dataout directly after the clk rising edge. Address width, data width and target technology is parametrizable through generics. Simultaneous write to the same address is technology dependent, and generally not allowed.

100.2 Configuration options

Table 1234 shows the configuration options of the core (VHDL generics).

*Table 1234.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table below	-
dbits	Data width	see table below	-

Table 1235 shows the supported technologies for the core.

*Table 1235.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
altera	All altera devices	altsyncram	unlimited	unlimited
virtex	Xilinx Virtex, Virtex-E, Spartan-2	RAMB4_Sn	2 - 10	unlimited
virtex2	Xilinx Virtex2/4/5/6 Spartan3/3a/3e/6	RAMB16_Sn	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	dp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss2_64x32cm4sw0 hdss2_128x32cm4sw0 hdss2_256x32cm4sw0 hdss2_512x32cm4sw0	6 - 9	32
memartisan	Artisan ASIC RAM	dp_256x32m4 dp_512x32m4 dp_1kx32m4	8 - 10	32
memvirage90	Virage 90 nm ASIC RAM	DPRAM_HS_256x20 DPRAM_HS_256x32	2 - 8	128
easic45	eASIC 45 nm Nextreme2	bRAM	unlimited	unlimited

100.3 Signal descriptions

Table 1236 shows the interface signals of the core (VHDL ports).

*Table 1236.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
CLK1	N/A	Input	Port1 clock	-
ADDRESS1	N/A	Input	Port1 address	-
DATAIN1	N/A	Input	Port1 write data	-
DATAOUT1	N/A	Output	Port1 read data	-
ENABLE1	N/A	Input	Port1 chip select	High
WRITE1	N/A	Input	Port 1 write enable	High
CLK2	N/A	Input	Port2 clock	-
ADDRESS2	N/A	Input	Port2 address	-
DATAIN2	N/A	Input	Port2 write data	-
DATAOUT2	N/A	Output	Port2 read data	-
ENABLE2	N/A	Input	Port2 chip select	High
WRITE2	N/A	Input	Port 2 write enable	High

100.4 Library dependencies

Table 1237 shows libraries used when instantiating the core (VHDL libraries).

*Table 1237.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
TECHMAP	GENCOMP	Constants	Technology contants

100.5 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

component syncram_dp
  generic (tech : integer := 0; abits : integer := 6; dbits : integer := 8);
  port (
    clk1     : in std_ulogic;
    address1 : in std_logic_vector((abits -1) downto 0);
    datain1  : in std_logic_vector((dbits -1) downto 0);
    dataout1 : out std_logic_vector((dbits -1) downto 0);
    enable1  : in std_ulogic;
    write1   : in std_ulogic;
    clk2     : in std_ulogic;
    address2 : in std_logic_vector((abits -1) downto 0);
    datain2  : in std_logic_vector((dbits -1) downto 0);
    dataout2 : out std_logic_vector((dbits -1) downto 0);
    enable2  : in std_ulogic;
    write2   : in std_ulogic);
  end component;
```

100.6 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
```

```
use techmap.gencomp.all;

clk1     : in std_ulogic;
address1 : in std_logic_vector((abits -1) downto 0);
datain1  : in std_logic_vector((dbits -1) downto 0);
dataout1 : out std_logic_vector((dbits -1) downto 0);
enable1  : in std_ulogic;
write1   : in std_ulogic;
clk2     : in std_ulogic;
address2 : in std_logic_vector((abits -1) downto 0);
datain2  : in std_logic_vector((dbits -1) downto 0);
dataout2 : out std_logic_vector((dbits -1) downto 0);
enable2  : in std_ulogic;
write2   : in std_ulogic);

ram0 : syncram_dp generic map ( tech => tech, abits => addrbits, dbits => dbits)
      port map ( clk1, address1, datain1, dataout1, enable1, write1, clk2, address2, datain2,
dataout2, enable2, write2);
```

101 TAP - JTAG TAP Controller

101.1 Overview

JTAG TAP Controller provides an Test Access Port according to IEEE-1149 (JTAG) Standard. The core implements the Test Access Port signals, the synchronous TAP state-machine, a number of JTAG data registers (depending on the target technology) and an interface to user-defined JTAG data registers.

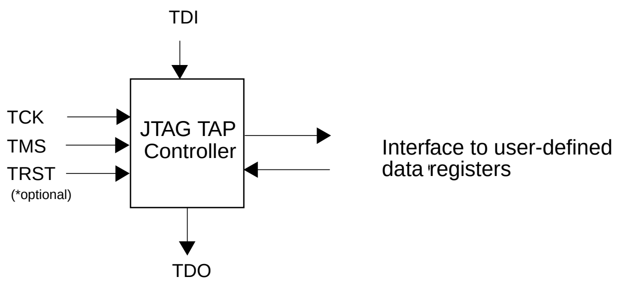


Figure 283. TAP controller block diagram

101.2 Operation

101.2.1 Generic TAP Controller

The generic TAP Controller implements JTAG Test Access Point interface with signals TCK, TMS, TDI and TDO, a synchronous state-machine compliant to the IEEE-1149 standard, JTAG instruction register and two JTAG data registers: bypass and device identification code register. The core is capable of shifting and updating the JTAG instruction register, putting the device into bypass mode (BYPASS instruction) and shifting out the devices identification number (IDCODE instruction). User-defined JTAG test registers are accessed through user-defined data register interface.

The access to the user-define test data registers is provided through the user-defined data register interface. The instruction in the TAP controller instruction register appears on the interface as well as shift-in data and signals indicating that the TAP controller is in Capture-Data-Register, Shift-Data-Register or Update-Data-Register state. Logic controlling user-defined data registers should observe value in the instruction register and TAP controller state signals in order to capture data, shift data or update data-registers.

JTAG test registers such as boundary-scan register can be interfaced to the TAP controller through the user data register interface.

101.3 Technology specific TAP controllers

The core instantiates technology specific TAP controller for Altera and Xilinx devices.

101.4 Registers

The core implements three JTAG registers: instruction, bypass and device identification code register.

101.5 Vendor and device identifiers

The core does not have vendor and device identifiers since it does not have AMBA interfaces.

101.6 Configuration options

Table 1238 shows the configuration options of the core (VHDL generics).

*Table 1238.*Configuration options

Generic	Function	Allowed range	Default
tech	Target technology	0 - NTECH	0
irlen	Instruction register length (generic tech only)	2 - 8	4
idcode	JTAG IDCODE instruction code(generic tech only)	0 - 255	9
manf	Manufacturer id. Appears as bits 11-1 in TAP controllers device identification register. Used only for generic technology. Default is Aeroflex Gaisler manufacturer id.	0 - 2047	804
part	Part number (generic tech only). Bits 27-12 in device id. reg.	0 - 65535	0
ver	Version number (generic tech only). Bits 31-28 in device id. reg.	0-15	0
trsten	Support optional TRST signal (generic tech only)	0 - 1	1
scantest	Enable scan test support	0 - 1	0
oepol	Polarity for TDOEN signal	0 - 1	1
tcknen	Support externally inverted TCK signal (generic tech only)	0 - 1	0

101.7 Signal descriptions

Table 1239 shows the interface signals of the core (VHDL ports).

*Table 1239.*Signal declarations

Signal name	**Field**	**Type**	**Function**	**Active**
TRST	N/A	Input	JTAG TRST signal*	Low
TCK	N/A	Input	JTAG clock*	-
TMS	N/A	Input	JTAG TMS signal*	High
TDI	N/A	Input	JTAG TDI signal*	High
TDO	N/A	Output	JTAG TDO signal*	High
User-defined data register interface				
TAPO_TCK	N/A	Output	TCK signal	High
TAPO_TDI	N/A	Output	TDI signal	High
TAPO_INST[7:0]	N/A	Output	Instruction in the TAP Ctrl instruction register	High
TAPO_RST	N/A	Output	TAP Controller in Test-Logic_Reset state	High
TAPO_CAPT	N/A	Output	TAP Controller in Capture-DR state	High
TAPO_SHFT	N/A	Output	TAP Controller in Shift-DR state	High
TAPO_UPD	N/A	Output	TAP Controller in Update-DR state	High
TAPO_XSEL1	N/A	Output	Xilinx User-defined Data Register 1 selected (Xilinx tech only)	High
TAPO_XSEL2	N/A	Output	Xilinx User-defined Data Register 2 selected (Xilinx tech only)	High
TAPI_EN1	N/A	Input	Enable shift-out data port 1 (TAPI_TDO1), when disabled data on port 2 is used	High
TAPI_TDO1	N/A	Input	Shift-out data from user-defined register port 1	High
TAPI_TDO2	N/A	Input	Shift-out data from user-defined register port 2	High
TAPO_NINST	N/A	Output	Instruction to be written into TAP Ctrl instruction register, only valid when TAPO_IUPD is high. (Generic tech only)	High
TAPO_IUPD	N/A	Output	TAP Controller in Update-IR state (Generic tech only)	High
TAPO_TCKN	N/A	Outpu	Inverted TCK signal	High
Additional signals				
TESTEN	N/A	Input	Test mode enable signal	High
TESTRST	N/A	Input	Test mode reset signal	Low
TESTOEN	N/A	Input	Test mode output-enable control	see oepol
TDOEN	N/A	Output	JTAG TDO enable signal*	see oepol
TCKN	N/A	Input	Inverted clock in (if tcknen generic is set)	

*) If the target technology is Xilinx or Altera the cores JTAG signals TCK, TCKN, TMS, TDI and TDO are not used. Instead the dedicated FPGA JTAG pins are used. These pins are implicitly made visible to the core through technology-specific TAP macro instantiation.

101.8 Library dependencies

Table 1240 shows libraries used when instantiating the core (VHDL libraries).

*Table 1240.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Component	TAP Controller component declaration

101.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library techmap;
use gaisler.gencomp.all;

entity tap_ex is
  port (
    clk : in std_ulogic;
    rst : in std_ulogic;

    -- JTAG signals
    tck  : in std_ulogic;
    tms  : in std_ulogic;
    tdi  : in std_ulogic;
    tdo  : out std_ulogic
);
end;

architecture rtl of tap_ex is

signal gnd : std_ulogic;

signal tapo_tck, tapo_tdi, tapo_rst, tapo_capt : std_ulogic;
signal tapo_shft, tapo_upd : std_ulogic;
signal tapi_en1, tapi_tdo : std_ulogic;
signal tapo_inst : std_logic_vector(7 downto 0);


begin

 gnd <= ‘0’;
 tckn <= not tck;


-- TAP Controller

  tap0 : tap (tech => 0)
    port map (rst, tck, tckn, tms, tdi, tdo, open, tapo_tck, tapo_tdi, tapo_inst,
      tapo_rst, tapo_capt, tapo_shft, tapo_upd, open, open,
       tapi_en1, tapi_tdo, gnd);


-- User-defined JTAG data registers

  ...


end;
```

102 GRUSB_DCL - USB Debug Communication Link

102.1 Overview

The Universal Serial Bus Debug Communication Link (GRUSB_DCL) provides an interface between a USB 2.0 bus and an AMBA-AHB bus. The core must be connected to the USB through an UTMI, UTMI+, or ULPI compliant PHY. Both full-speed and high-speed mode are supported. The GRUSB_DCL rely on the GRUSBDC core for handling the USB communication and communication with the PHY. The GRUSB_DCL implements the minimum required set of USB requests to be Version 2.0 compliant and a simple protocol for performing read and write accesses on the AHB bus. Figure 284 show how the GRUSB_DCL can be connected to a PHY. For more information on the GRUSBDC and the connection to the USB PHY please refer to the GRLIB IP Core User’s Manual.

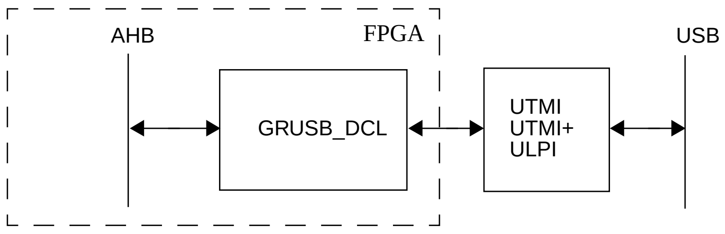


Figure 284. USBDCL connected to an external UTM.

102.2 Operation

102.2.1 System overview

The internal structure of the GRUSB_DCL can be seen in figure 285. The GRUSB_DCL is constructed with two internal AHB busses for communication with the GRUSBDC and one external AHB master interface for reading and writing the external AHB bus. Since the GRUSBDC is connected with point-to-point links there is no need for a conventional AHB arbiter on the internal bus.

The GRUSBDC is configured with two bidirectional endpoints with endpoint zero (EP0) being the default USB control endpoint and endpoint one (EP1) the communication endpoint for the DCL protocol. The GRUSBDC is configured to use DMA and its descriptors as well as the DMA buffers are stored in a local memory with separate read and write ports (SYNCRAM_2P). The two ports makes it possible for the GRUSBDC and the internal workings of the GRUSB_DCL to access the memory in parallel. Arbitration for the read and write port is implemented as two separate procedures. The main functionality of the GRUSB_DCL is implemented in the main FSM procedure. The FSM can be seen in figure 286.

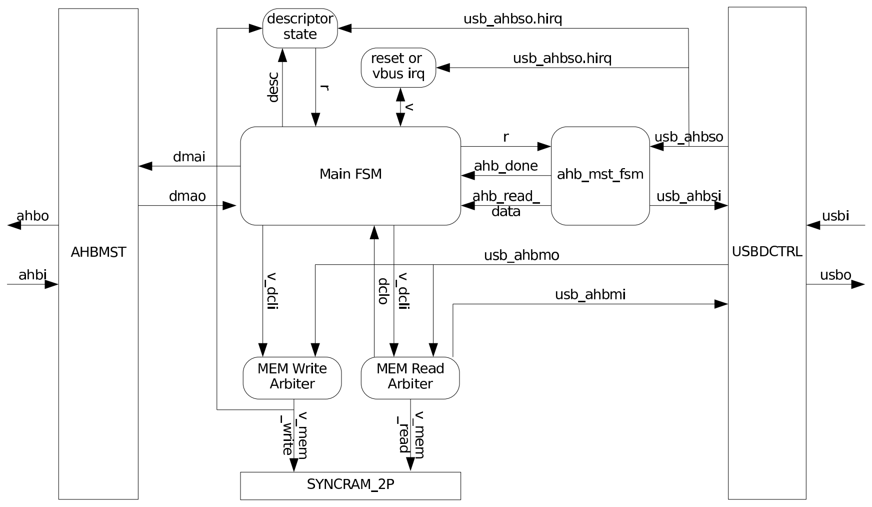


Figure 285. Block diagram of the internal structure of the GRUSBDCL. Blocks with rounded corners are implemented as VHDL procdures while squares represent VHDL entities.

Upon reset the FSM begins with setting up the DMA descriptors in the local memory and then configures the GRUSBDC such that it becomes active. The FSM then waits for incoming requests on either EP0 or EP1. For EP0 each request is validated and then appropriate action is taken according to the USB Version 2.0 standard. For undefined requests the GRUSB_DCL returns an error by stalling EP0. For EP1 the DCL request is fetched and either data is written to the AHB buss from the local memory or data is read from the AHB and stored in the local memory. In the case of the AHB is being read the data is then sent on EP1 IN. To keep track of the state of the DMA descriptors the FSM has help from the descriptor-state procedure. The descriptor-state procedure uses the IRQ from the GRUSBDC to identify incoming packets on either EP0 or EP1. By storing the last accessed address by the GRUSBDC to the local memory the descriptor-state procedure can tell which EP that has been updated. The FSM in turn signals the procedure telling it when a DMA descriptor/buffer has been read/write. A small FSM (ahb_mst_fsm) is also used when the main FSM wants to update the state of the GRUSBDC through the AHB slave interface. Finally, the reset-or-vbus-irq procedure listens on irqs from the USBDC that informs the core that a USB reset or that a change on the VBUS has occured. In that case the core stops what ever it was doing and moves into the USB default state (no addresse set and not configured).

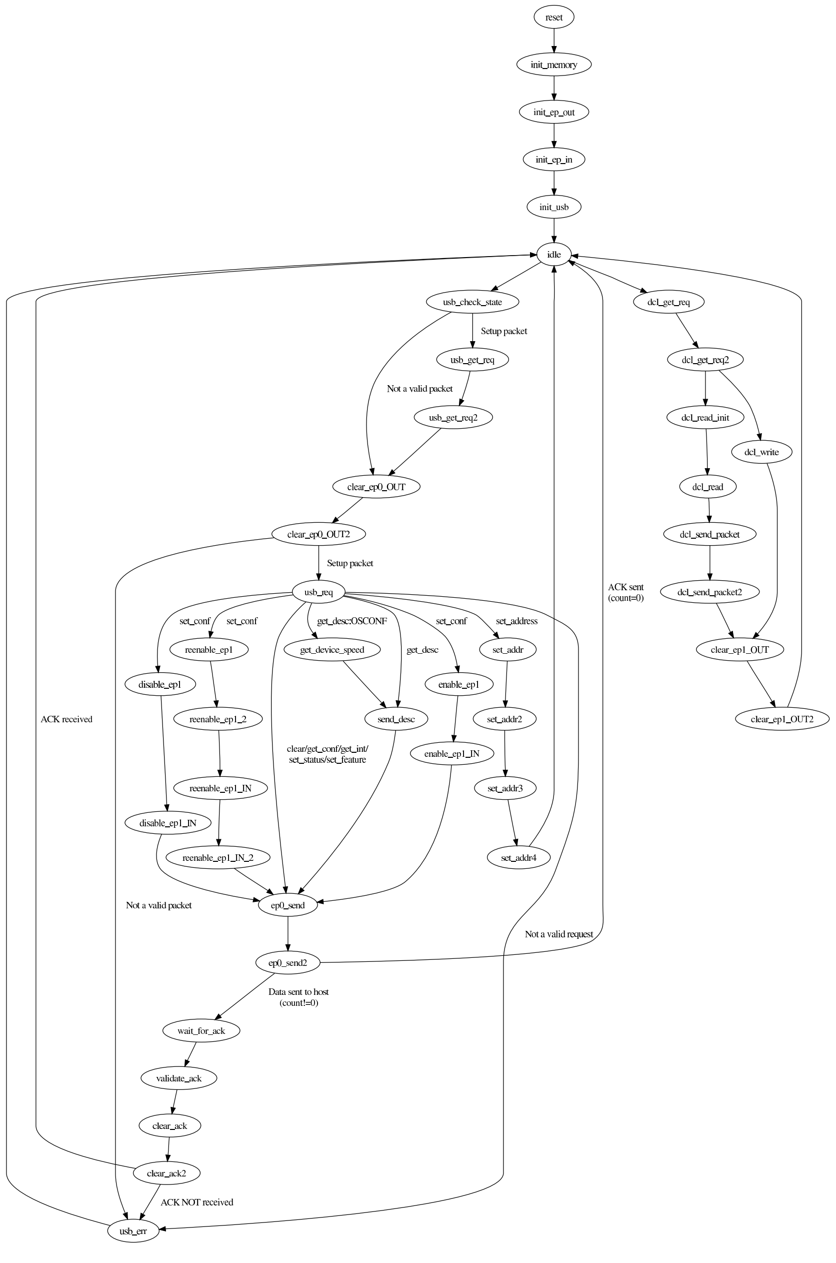


Figure 286. The main FSM of the GRUSBDCL. The FSM can be divided into three major parts i) the initialization (reset to idle), ii) handling of USB requests (seen to the left of idle), and iii) handling of DCL requests (seen to the right of idle).

102.2.2 Protocol

The protocol used for the AHB commands is very simple and consists of two 32-bit control words. The first word consists of the 32-bit AHB address and the second consists of a read/write bit at bit 31 and the number of words to be written at bits 16 downto 2. All other bits in the second word are reserved for future use and must be set to 0. The read/write bit must be set to 1 for writes.

Figure 287 shows the layout of a write command. The command should be sent as the data cargo of an OUT transaction to endpoint 1. The data for a command must be included in the same packet. The maximum payload is 512 B when running in high-speed mode and 64 B in full-speed mode. Since the control information takes 8 B the maximum number of bytes per command is 504 B and 56 B respectively. Subword writes are not supported so the number of bytes must be a multiple of four between 0 and 504.

The words should be sent with the one to be written at the start address first. Individual bytes should be transmitted msb first, i.e. the one at bits 31-24.

There is no reply sent for writes since the USB handshake mechanism for bulk writes guarantees that the packet has been correctly received by the target.

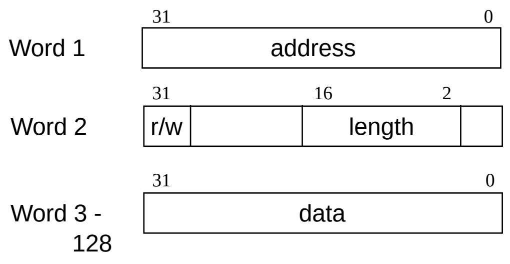


Figure 287. Layout of USBDCL write commands.

Figure 288 shows the layout of read commands and replies. In this case the command only consists of two words containing the same control information as the two first words for write commands. However, for reads the r/w bit must be set to 0.

When the read is performed data is read to the buffer belonging to IN endpoint 1. The reply packet is sent when the next IN token arrives after all data has been stored to the buffer. The reply packets only contains the read data (no control information is needed) with the word read from the start address transmitted first. Individual bytes are sent with most significant byte first, i.e. the byte at bit 31 downto 24.

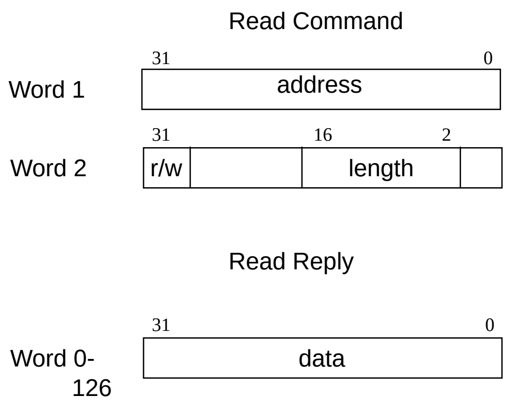


Figure 288. Layout of USBDCL read commands and replies.

102.2.3 AHB operations

All AHB operations are performed as incremental bursts of unspecified length. Only word size accesses are done.

102.2.4 Scan test support

The VHDL generic *scantest* enables scan test support for both the GRUSB_DCL and GRUSBDC. When the scanen and testen signals in the AHB master input record are high the GRUSB_DCL will disable the internal RAM blocks.

See GRUSBDC section of GRLIB IP Core User's Manual for details on the scan support for GRUSBDC.

102.3 Registers

The core does not contain any user accessible registers.

102.4 Vendor and device identifier

The core has vendor identifier 0x01 (Aeroflex Gaisler) and device identifier 0x022. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

The USB vendor identifier is 0x1781 and product identifier is 0x0AA0.

102.5 Configuration options

Table 1241 shows the configuration options of the core (VHDL generics).

*Table 1241.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
memtech	Memory technology used for blockrams (endpoint buffers).	0 - NTECH	0
uiface	Please see GRUSBDC section in the GRLIB IP Core User's Manual.		
dwidth	Please see GRUSBDC section in the GRLIB IP Core User's Manual.		
oepol	Please see GRUSBDC section in the GRLIB IP Core User's Manual.		
syncprst	Please see GRUSBDC section in the GRLIB IP Core User's Manual.		
prsttime	Please see GRUSBDC section in the GRLIB IP Core User's Manual.		
sysfreq	Please see GRUSBDC section in the GRLIB IP Core User's Manual.		
keepclk	Please see GRUSBDC section in the GRLIB IP Core User's Manual.		
functesten	Please see GRUSBDC section in the GRLIB IP Core User's Manual. If this generic is non-zero, the core will sample the value of its functesten input signal during reset. This value will then be used when assigning the Functional Testmode field in the GRUSBDC control register. The functesten input can be useful during netlist simulation as functional test mode reduces simulation time. If this generic is set to zero, the value of the functesten input will be disregarded and the Functional Testmode field will always be written with '0'.		
burstlength	Sets the maximum burst length in 32-bit words. The core will not burst over a burstlength word boundary.	8	1 - 512
scantest	Set this generic to 1 if scan test support should be implemented.	0 - 1	0

102.6 Signal descriptions

Table 1242 shows the interface signals of the core (VHDL ports).

*Table 1242.*Signal descriptions

Signal name	Field	Type	Function	Active
UCLK	N/A	Input	USB UTMI Clock	-
USBI	*	Input	USB Input signals	-
	functesten	Input	Functional test enable. If the core has been implemented with support for functional test mode (VHDL generic *functesten*), this signal will be sampled during core reset. Its value will then be used to set the functional testmode enable bit in the GRUSBDC core's control register.	High
USBO	*	Output	USB Output signals	-
HCLK		Input	AMBA Clock	-
HRST		Input	AMBA Reset	Low
AHBMI	**	Input	AHB master input signals	-
AHBMO	**	Output	AHB master output signals	-

* see GRUSBDC section og GRLIB IP Core User's Manual

** see GRLIB IP Library User's Manual

102.7 Library dependencies

Table 1243 shows libraries used when instantiating the core (VHDL libraries).

*Table 1243.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	GRUSB	Signals, components	GRUSB_DCL and GRUSBDC component declarations, USB signals

102.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.grusb.all;

entity usbdcl_ex is
  port (
    clk : in std_ulogic; --AHB Clock
    rstn : in std_ulogic;

    -- usb signals
    usb_clkout  : in std_ulogic;
    usb_d       : inout std_logic_vector(7 downto 0);
    usb_nxt     : in  std_ulogic;
    usb_stp     : out std_ulogic;
    usb_dir     : in  std_ulogic;
    usb_resetn  : out std_ulogic
```

```
end;

architecture rtl of usbdcl_ex is
  constant padtech : integer := inferred;
  constant memtech : integer := inferred;

  -- AMBA signals
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
  -- USB signals
  signal usbi : grusb_in_type;
  signal usbo : grusb_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- GRUSB_DCL
  usb_d_pad: iopadv
    generic map(tech => padtech, width => 8)
    port map (usb_d, usbo.dataout, usbo.oen, usbi.datain);
  usb_nxt_pad : inpad generic map (tech => padtech)
    port map (usb_nxt, usbi.nxt);
  usb_dir_pad : inpad generic map (tech => padtech)
    port map (usb_dir, usbi.dir);
  usb_resetn_pad : outpad generic map (tech => padtech)
    port map (usb_resetn, usbo.reset);
  usb_stp_pad : outpad generic map (tech => padtech)
    port map (usb_stp, usbo.stp);

  usb_clkout_pad : clkpad
    generic map (tech => padtech)
    port map (usb_clkout, uclk);

  usbi.urstdrive <= ‘0’;

  usbdcl0: grusb_dcl
    generic map (
        hindex  => 0,
        memtech => memtech,
        uiface  => 1,
        dwidth  => 8,
        oepol   => 0)
    port map (
        uclk => uclk,
        usbi => usbi,
        usbo => usbo,
        hclk => clk,
        hrst => rstn,
        ahbi => ahbmi,
        ahbo => ahbmo(0));
end;
```

Aeroflex Gaisler AB	tel +46 31 7758650
Kungsgatan 12	fax +46 31 421407
411 19 Göteborg	sales@gaisler.com
Sweden	www.aeroflex.com/gaisler