A FLEXIBLE HARDWARE ARCHITECTURE FOR FAST ACCESS TO LARGE NON-VOLATILE MEMORIES

Eric Armengaud, Florian Rothensteiner, Andreas Steininger
Vienna University of Technology
Embedded Computing Systems Group E182-2
Treitlstr. 3, A-1040 Vienna
{armengaud, rothensteiner, steininger}@ecs.tuwien.ac.at

Martin Horauer
University of Applied Sciences
Technikum Wien
Höchstädtplatz 5, A-1200 Vienna
horauer@technikum-wien.at

Abstract. *For distributed systems, the ability to accurately monitor the bus traffic is required in order to enable fault removal and fault forecasting for system verification, enhancement and evaluation. Typically, modern communication systems have high data throughput, and monitoring the bus for hours produces large amounts of data at high rates. In the expected operating environment of the monitoring tool, non-volatile memories are often mandatory as data storage. The required frequent access to these memories, however, results in high processor load. To that aim, this paper presents an approach that illustrates the improvements that can be achieved when several memory access functionalities are transferred from software to hardware. To quantify the results, the different approaches implemented in an FPGA were subjected to an experimental evaluation.*

1 Introduction

An important issue in our STEACS[1] project is to monitor the FlexRay [1] network traffic and store it to non-volatile memories, which is especially useful when the tests cannot be performed on-line. By post-processing the monitored data it is possible to apply systematic tests on the syntax and the content of the messages exchanged between the nodes. This aids in correctness checking, debugging and testing of the distributed system, see [2]. Next to testing, performance enhancement at the system level can be achieved by analyzing and optimizing the data flow between the nodes. Finally, replaying monitored data is useful for debugging purposes and to force the system into defined states in order to observe the reaction and compare it to a known-good reference.

One application of such a monitoring tool is prototyping in the automotive domain; here power supply continuity cannot always be assumed, therefore, using non-volatile memories is mandatory. Another constraint issued by our industry partners is that the monitoring

[1] The STEACS-project received support from the Austrian "FIT-IT[embedded systems" initiative, funded by the Austrian Ministry for Traffic, Innovation and Technology (BMVIT) and managed by the Austrian Industrial Research Promotion Fund (FFF) under grant 807146. See http://embsys.technikum-wien.at/steacs.html for further information.

device should not contain any rotating parts, i.e. no hard-disks. As a result, we decided to rely on compact flash technology. These devices are designed for a sustained speed of up to 9 Mbytes/s [3]. In practice, a maximum effective continuous data rate of 4 Mbytes/s can be reached, while the FlexRay bus uses two channels with 10 Mbits/s and additional data has to be written into the memory. The memory bandwidth turned out to be close to our requirements, thus reducing the flexibility for further improvements. Moreover, first prototypes of our implementation yielded a high processor load for writing the data into the compact flash device. Hence, almost no processor time was left for other tasks.

This paper presents a hardware dedicated to improving the data throughput and to reducing the processor resources needed for memory access. In particular, the presented architecture is inspired from the OSI reference model to decrease complexity and improve flexibility. The idea is to implement one physical layer and one data link layer for each memory bank connected, and a global transport layer on the top of them. The structured architecture improves the flexibility; when a new bus or memory is used, only one layer needs to be changed. Moving the services onto a dedicated hardware and parallelizing memories saves processor resources and improves performance.

The paper is organized as follows: The next section identifies the requirements for this specific problem. The aim of Section 3 is to give an overview of the state of the art and present our approach. After that, the proposed architecture is discussed more in detail in Section 4. In Section 5 an implementation is described and the achieved improvements are discussed. Finally, some conclusions are drawn in Section 6.

2 Requirements

One goal of the STEACS project is to design a monitoring tool for network traffic. This tool shall be used autonomously within an automobile as a single box storing recorded information into onboard memories. This section presents the requirements for this module.

a) *Non-volatile memories, i.e. compact flash devices shall be used*: In the prototyping environment power supply continuity cannot be assumed and in case of a system crash the monitored data shall not be lost. Moreover, the operational environment restricts us from using mechanical parts (hard disks) for that purpose.

b) *The transfer of a large amount of data with high throughput shall be supported*: The monitoring module is expected to record bus traffic up to some hours with a maximum bandwidth of up to 20 Mbits/s (2 channels at 10 Mbit/s). Peak loads of the data generated can reach up to 25 Mbits/s due to additional information such as timestamps or identifiers that are added in the monitoring process. In particular, monitoring one hour of bus traffic might generate up to 11 Gbytes of raw data (i.e. before filtering or compressing).

c) *The data flow is mostly sequential, only a few memory jumps will be needed*: The data monitored is written to the memory bank sequentially. However, a basic file system is used and some ‘super blocks’ have to be periodically updated to describe the data validity within the memory banks. Due to the sequential data flow optimization techniques that make use of data access recurrence to improve memory access time – such as memory caches [4] – can not be applied. However, the use of sector based memory accesses as described in the ATA protocol specification [5] and commonly implemented in hard-disks and compact flash devices is favorable.

d) *A protection of critical data against a system crash is required*: The file system shall be resistant to a system crash; in fact, without protection, the corruption of a super block can lead to a loss of the whole monitored data.

e) *Flexible start-up and configuration of the memory banks:* while these specific operations are not time critical, a high flexibility is required to enable different operating modes or for system upgrades to more powerful memory elements.

f) *The processor load shall be kept as low as possible in order to save processor time:* The host needs resources for other tasks such as e.g. controlling or filtering. Moreover, frequent requests to drive the memory banks shall be avoided in order to reduce the scheduling constraints or the interrupt overhead (task switch).

g) *The architecture shall be flexible* to allow the use of different types of memories and/or different bus interfaces.

3 Improving the state of the art

While higher data throughput can be achieved by accessing memory banks in parallel, reducing processor load is usually done by moving tasks to dedicated hardware. Both methods in isolation have already been largely explored and documented [4]. The aim of this work is to combine both methods and provide a structured architecture with small independent modules and well specified interfaces to achieve flexibility; a single module can be easily exchanged when another bus system or memory type is used, and the global architecture is kept unchanged. Moreover, these small and independent modules are much easier to design, verify and integrate in hardware than a single big one. This approach is largely based on the OSI reference model [6]. Indeed, similarities between serial and parallel communication systems can be exploited for the physical layer (transmitting atomic information – bit or vector – over a communication medium), data link layer (structure raw transmission into frames – or sectors) and transport layer (data organization). Figure 1-A presents the starting architecture without dedicated interface hardware.

The use of several memory banks in parallel is nothing new since it can be found in almost every digital system design, e.g. when two 8-bit wide memory banks are connected to a 16-bit CPU. Another approach using configurable parallel memory architectures has been developed in [7]. Using a dedicated hardware with several memory banks connected in parallel, a new bank can be accessed during ongoing accesses on other banks (‘interleaving’ architecture, Figure 1-B). This solution can be seen as a parallelization of the physical layer in order to pipeline the atomic transfers. While the resulting data throughput from processor view is improved, the alternating access to the memory banks results in frequent interrupts and makes the task scheduling difficult.

Optimization at the physical layer as described above does not care about the internal structure of large memory banks. They are typically organized in heads and sectors and special handling and long pauses are needed when accessing a new sector. It is precisely the aim of the data link layer to organize data transfers in frames or sectors. This layer can be moved into hardware (Figure 1-C) and be parallelized (one for every memory bank). The resulting module is in charge of accessing autonomously an entire sector for a specific memory bank; its autonomous work results in a smaller processor load because the host does not need to explicitly start every atomic operation anymore. Besides, an internal high speed interface can be used between the host and these modules and the data can be internally buffered. To sustain the desired high data rate, the host has to provide the hardware with data (sectors) as soon as needed, which causes strict real time constraints. With this architecture the service of splitting and merging the data between the different memory banks is still done in software.

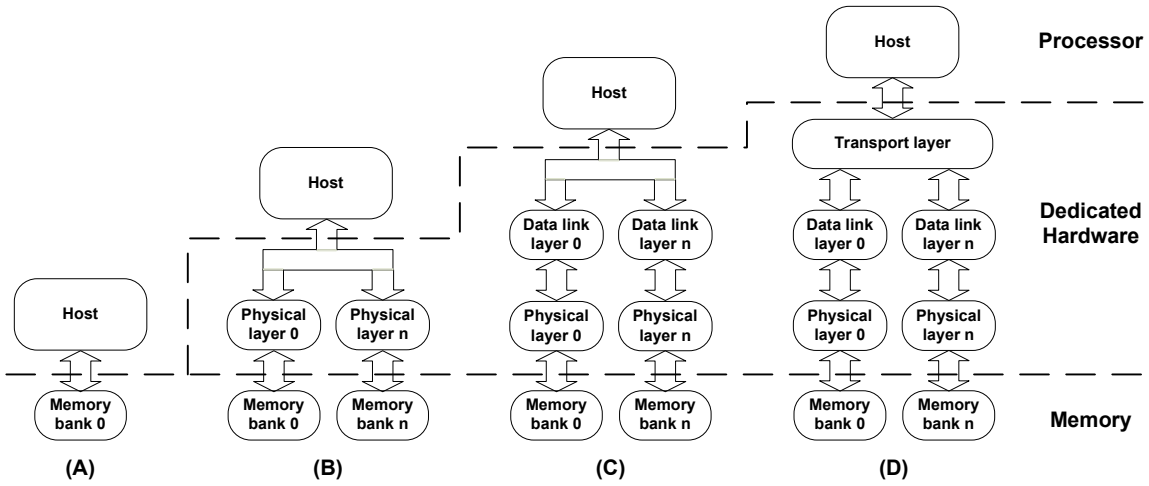


Figure 1: Architecture proposals

The transport layer is situated on top of the data link layer. During the transmission part its function is to split the data flow from the host into sectors, distribute these sectors to the different memory banks and control the memory accesses. During the receive part the data is read in parallel from the memory banks and then re-organized for the host (Figure 1-D).

The idea is to implement one physical layer and data link layer for every memory bank, and an additional transport layer on top of them. This structured architecture eases maintenance; when a new bus or memory bank is added only one layer needs to be changed. Moreover these parallel processing paths enable the use of different memory types (data link layer) and / or different memory interfaces (physical layer), which improves flexibility. Finally, moving the services to a dedicated hardware and parallelizing the memory accesses saves processor time and improves performance. As a result, the non-volatile aspect is kept, the memory speed and size are improved from the host's view, and the number of connected memory banks is abstracted to the host.

4 Proposed architecture

Figure 2 illustrates the proposed solution. The lowest layer (*physical layer*) builds a physical connection with the memory banks (using for example an IDE, IndustryPack or ISA interface), and describes the mechanical, electrical and timing interface. It typically implements a wrapper between a specific internal bus system to an external one, and performs single and atomic memory accesses to specified addresses. Next, the *data link layer* implements the access protocol to a specific memory type (e.g. sector addressing using the ATA protocol [5]). It typically offers read and write functions to access the devices sector-by-sector and provides a high speed internal buffer. These two layers are already well documented in different implementation notes (e.g. [5], [8]) and won't be described any further in this document.

The aim of the *transport layer* is to abstract the physical memory organization to the host and to automatically distribute the data stream to the different connected memory banks. To that aim, this layer comprises a set of control and status registers, a control state machine and a Memory Management Unit (MMU) module. The register set is used to configure and control the layer. The state machine processes the commands of the host, routes the requests to the different memory interfaces and controls them. Finally, the MMU is in charge of mapping the data between the host and the different memory banks.

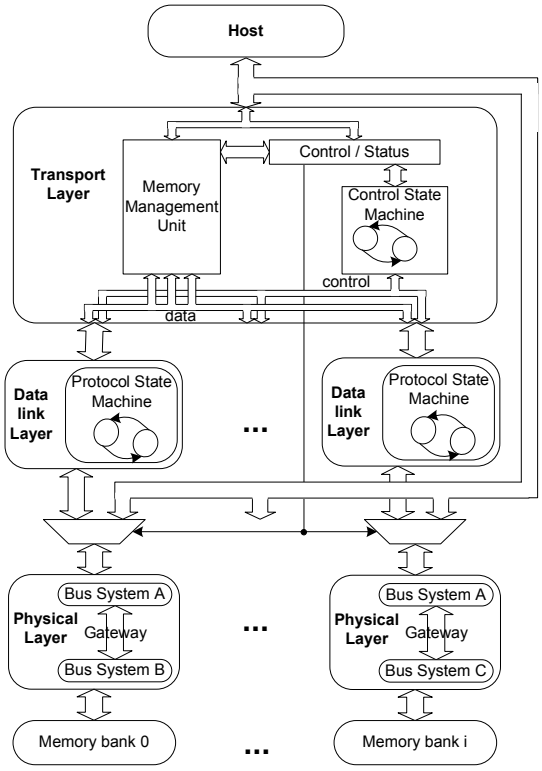


Figure 2: Global overview of the architecture

The Memory Management Unit maintains a FIFO queue for each physical memory bank, which is selected in a cyclic manner using a multiplexer. As a result, the data stream is automatically allocated to the different banks during write operation and automatically sorted during read operation. Additionally, the MMU supports a basic file system where the status of the global memory content (typically beginning and end of the valid data and information about the data content) is summarized. This information has to be periodically saved to the memory banks to refresh the memory status and is thus very sensitive: The entire memory content may be lost when this block is corrupted. To avoid conflicts, a lock system is used to share this block between the host and the different memory banks. That way, the information can be written on different memory banks (value redundancy) and at different points in time (timing redundancy resulting from the lock mechanism).

The host might additionally bypass the dedicated hardware to directly access the memory banks. This operation is needed for initialization (not time-critical) to save resources in reducing the functionalities required in hardware and to improve the configuration flexibility.

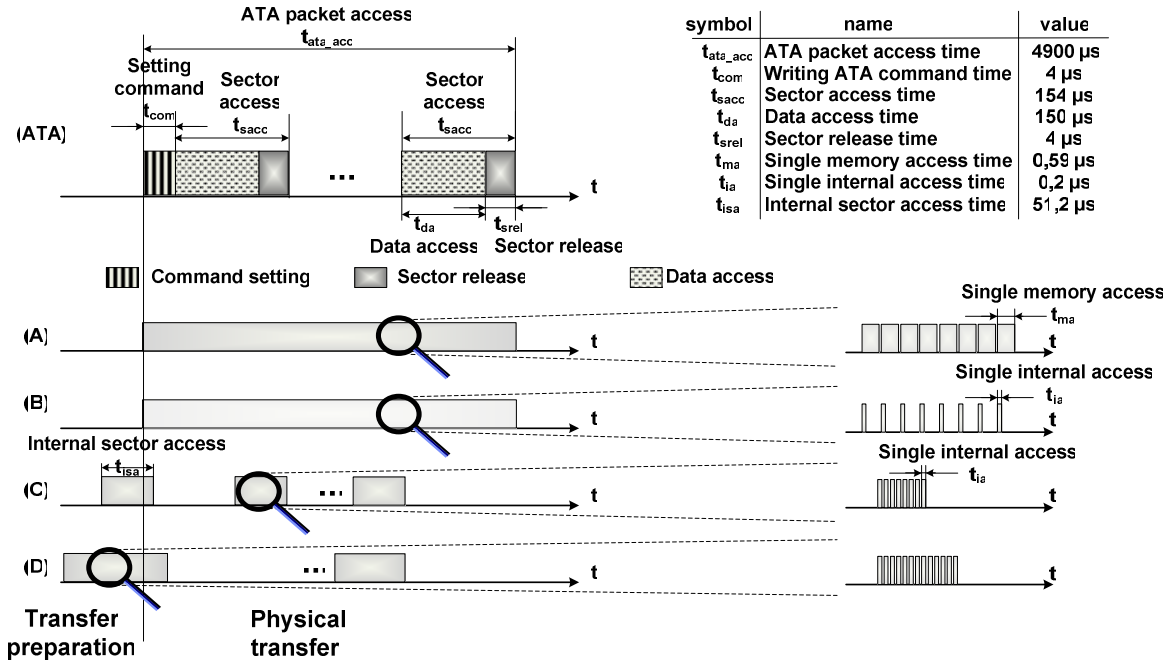
5 Case study

In order to test our concept we made an implementation which enables the connection of two compact flash memory banks in parallel. Both physical layers consist of a wrapper between a standard parallel interface (AMBA, [9]) and the IndustryPack logic interface [8]. The data link layers implement the ATA protocol with the capability to directly address memory sectors. The transport layer MMU provides a FIFO queue with size of two ATA sectors for each compact flash. Our actual implementation has been done using an Altera

Excalibur chip EPXA4FI672-2 [10] that combines a 400kGates FPGA with an ARM processor stripe. Our implementation consumes 2.100 logic cells (13%), 20.480 memory bits (9%) and operates at a maximum frequency of 55 MHz.

5.1 Comparison between the different optimizations

Figure 3 outlines a sector access according to the ATA protocol with respect to timing at the physical pins of the compact flash memory bank (ATA) and the resulting processor load (A – D) for the different architectures described in Figure 1.



symbol	name	value
t_{ata_acc}	ATA packet access time	4900 µs
t_{com}	Writing ATA command time	4 µs
t_{sacc}	Sector access time	154 µs
t_{da}	Data access time	150 µs
t_{srel}	Sector release time	4 µs
t_{ma}	Single memory access time	0,59 µs
t_{ia}	Single internal access time	0,2 µs
t_{isa}	Internal sector access time	51,2 µs

Figure 3: ATA access and processor load for different architectures

An ATA packet access is divided into command setting field, data accesses and sector release. In the first part (command setting), always the same data structure is written (first the base address followed by the access type, the length, and finally waiting for the memory availability). Then a configurable number of sectors are accessed – 32 for our implementation. This task consists of two phases: first accessing a sector (data access) and then waiting for the memory availability (sector release), where a register content is polled until a value is set.

Figure 3-A represents the processor load for the purely software based architecture of Figure 1-A. All memory accesses are explicitly performed by software and the CPU must wait for the availability of the memory bank. Consequently, the processor load is hundred percent during the whole ATA access.

Figure 3-B is using a first optimization at the physical layer according to Figure 1-B. The host initiates the individual memory accesses which are then performed by a dedicated hardware. Consequently the processor is free to perform other tasks during the slow memory access – such as initiating an access to another compact flash device. This solution allows interleaving architectures where several memory banks can be accessed in parallel [11]; however, to allow fully pipelined accesses, these banks must have exactly the same access timing. The limitation of this architecture is the high interrupt frequency.

Figure 3-C presents a further optimization at the data link layer according to Figure 1-C. The whole sector access is automatically performed by the dedicated hardware. This solution presents the advantage of saving explicit control accesses by the software and allows more efficient memory accesses due to the use of a high speed data buffer. Consequently, computation resources are saved and the processor has more time (one sector release t_{srel}) between two successive requests. However, the resulting real time requirements were still too tight for our application and the memory management remained complex.

Figure 3-D presents the processor load for the final architecture employing an optimization at the transport layer (according to Figure 1-D). While the cumulated processor load stayed the same as in the previous optimization, the maximum time between two successive host requests was extended, thus decreasing real time constraints for the other tasks. Additionally, the Memory Management Unit simplified the memory management.

5.2 Results

Table 1 presents the results achieved with a single and a dual memory architecture, respectively. The main advantage of the second architecture is the ability to access both memory blocks simultaneously, which consequently increases the data rate. For a given architecture, the memory values (overall access time and data rate) are constant for all optimization stages since they represent limitations of the physical memory banks. It would be possible to connect more memory banks in parallel to improve the data throughput further; the limitation herein is typically given by the processor throughput (processor load).

Table 1: Experimental comparison of single and dual memory architecture

Memory architecture		Single memory bank architecture				Dual memory bank architecture			
Data size		100 KB	1 MB	10 MB	100 MB	100 KB	1 MB	10 MB	100 MB
Memory banks	overall access time	25,6 ms	268 ms	2,67 s	28s	12,8 ms	134 ms	1,33 s	14s
	data rate (MB/s)	3,9	3,73	3,75	3,57	7,8	7,5	7,5	7,1
Pure software (fig. 3-A)	processor time	25,6 ms	268 ms	2,67 s	28s	Not applicable			
	processor load	100%	100%	100%	100%				
	request period	0 µs (continuous)							
Physical layer optimization (fig. 3-B)	processor time	6,0 ms	60,1 ms	603 ms	6,03 s	6,0 ms	60,1 ms	603 ms	6,03 s
	processor load	23,4%	22,4%	22,5%	21,5%	47%	44,8%	45,3%	43%
	request period	0,59 µs				0,59 µs			
Data link layer optimization (fig. 3-C)	processor time	5,83 ms	58,3 ms	584 ms	5,83 s	5,83 ms	58,3 ms	584 ms	5,83 s
	processor load	22,7%	21,8%	21,9%	20,8%	45,5%	43,5%	43,9%	41,6%
	request period	154 µs				154 µs			
Transport layer optimization (fig. 3-D)	processor time	5,83 ms	58,3 ms	584 ms	5,83 s	5,83 ms	58,3 ms	584 ms	5,83 s
	processor load	22,7%	21,8%	21,9%	20,8%	45,5%	43,5%	43,9%	41,6%
	request period	308 µs				308 µs			

The row ‘request period’ represents the time between two successive software requests to access the memory banks, which, in other words, represents the real-time constraints. Since the processor resources are already completely blocked for driving a single memory bank, the pure software method can not be used with a dual memory architecture. The processor load is decreased to approximately 22% and the request period is relaxed to 0,59 µs with the physical layer optimization. It enables other tasks to run in parallel, for example the alternating control of several memory banks. Next, the request period is relaxed by a factor of 300 with the data link layer optimization since an entire sector can be buffered within the dedicated hardware. Finally, the period is increased by an additional factor of two with the last optimization. This result is implementation specific and is due to the use

of a FIFO queue size of two sectors. Increasing the FIFO size would increase the interrupt period at the cost of hardware resources.

In summary, the optimization at the transport layer with a dual memory architecture resulted (i) in an increased memory bandwidth, which enable high data rates, and (ii) in long request periods, which enable continuous processing of other tasks without being interrupted. Furthermore, this experiment illustrates the independence between the improvements achieved and the data volume transferred. Additionally, the resulting memory size was doubled.

6 Conclusion

The required operations for storing a large amount of data generated by unfiltered monitoring of a distributed FlexRay-based automotive system can heavily load an embedded processor system leaving insufficient resources for other tasks. To that aim, this paper has described how to optimize and improve the accesses to non-volatile memories, i.e. compact flash devices, by transferring several functionalities from software to hardware.

In our FPGA based implementation we used the layering and data encapsulation aspects found in serial networks. Our modular approach proposed a method to improve the processor load and the real time requirements, i.e. the frequency and duration of the required accesses to the memory bank, by transferring different functionalities from software to hardware in a systematic, structured manner.

Following the presentation of our approach an implementation and the achieved results are listed by way of an experimental evaluation. The results show the optimizations due to the hardware implementations of different layers. While the data throughput is improved when increasing the number of memory banks driven in parallel, the most efficient architecture in terms of processor load and real time constraints is reached when the services up to the transport layer are shifted to dedicated hardware.

7 References

[1] FlexRay Consortium, FlexRay Communications System Protocol Specification Version 2.0, www.flexray.com, 2004.

[2] Beth A. Schroeder, On-Line Monitoring: A Tutorial, Computer Practices, Volume 28, pages: 72 -78, June 1995.

[3] SanDisk Ultra® II CompactFlash, http://www.sandisk.com/retail/ultra2-cf.asp.

[4] J. L. Hennessy and D. A. Patterson, Computer Architecture, a Quantitative Approach, second edition, Edition Morgan Kaufmann, 1996.

[5] AT Attachment - 4 with Packet Interface Extension, ANSI NCITS 317-1998.

[6] A.S. Tanenbaum. Computer Networks, fourth edition. Pearson Education, Inc, 2003.

[7] J. Vanne, E. Aho, K. Kuusilinna and T. Hamalainen. Enhanced Configurable Parallel Memory Architecture. Proceedings of the Euromicro Symposium on Digital System Design, pages: 28 – 35, 2002.

[8] IP Modules Draft Standard, VITA 4-1995, Draft 1.0.d.0, April 7, 1995, VITA Standards Organization.

[9] ARM Corporation, AMBA Specification Rev 2.0, 1999, http://www.arm.com/.

[10] Altera Corporation. The Advantages of Hard Subsystems in Embedded Processors PLDs, version 1.0. March 2002, White paper

[11] A. Seznec and J. Lenfant. Interleaved Parallel Schemes. IEEE Transactions on Parallel and Distributed Systems, Vol. 5, Issue 12, pp. 1329 – 1334, Dec. 1994.