An FPGA based SoC Design for Testing Embedded Automotive Communication Systems employing the FlexRay Protocol

Martin Horauer, Florian Rothensteiner, Martin Zauner
University of Applied Sciences Technikum Wien
Höchstädtplatz 5, A-1200 Vienna
{horauer, rothensteiner, zauner}@technikum-wien.at

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

Hannes Friedl, Roman Pallierer
Dependable Computer Systems
DECOMSYS GmbH
Stumpergasse 48/28, A-1060 Vienna
{friedl, pallierer}@decomsys.com

Abstract

This paper presents an implementation of a System-on-Chip solution for monitoring the communication subsystem of FlexRay-based automotive distributed networks. The same principles are used in the reverse direction for a "replay" mechanism that allows re-enacting recorded situations for debugging and fault confinement. Next to an overview of our realization, we present some informal details of a set of rules to provide temporal ordering of multiple packets that are generated in parallel from different monitoring sources. Altogether, the presented FPGA implementation is tailored for an embedded test and fault diagnosis and shall enable an assessment of the reliability and dependability of future distributed automotive networks.

1 Introduction

A very promising candidate for future automotive, distributed electronic control systems termed FlexRay[1] is presently developed by an industrial consortium of several key players in the automotive and electronic market, cf. [6, 5]. Relying on both the time- and event-triggered paradigms this communication protocol combines reliability and fault-tolerance aspects with the bandwidth to serve the needs of a communication backbone and for the coupling of sensor/actuator systems required for future X-by-wire solutions.

For the development, operation, service and maintenance of such a distributed system facilities for test and replay will be required in order to enable monitoring, debugging and fault diagnosis respectively. Furthermore, some means for fault injection will be required to asses the reliability of the system under inspection.

[1] http://www.flexray.com

It is the aim of our STEACS[2] project to address these challenges and take a first step towards a solution. To that end, we present in this paper an FPGA-based SoC design suited for monitoring and replay of FlexRay-based communication networks that provides information and input for a subsequent fault diagnosis stage with the vision to enable an embedded self-test.

Our paper is organized as follows: First, we present the system architecture under consideration in Sec. 2. Next, we describe the architecture of our tester node followed by some interesting details of our test and replay hardware in Sec. 4. Finally we conclude our paper with some directions for future enhancements and developments.

2 System Architecture

Fig. 1 illustrates a small distributed FlexRay system consisting of several nodes connected with each other either via the help of star-couplers and/or by a linear bus system. In general, in FlexRay it is possible to have both static TDMA-based and dynamic arbitrated bus communication within one single communication cycle. In addition, every node is equipped with two channels that can be used in a redundant or non-redundant fashion. Based on such a system, our approach is to add a dedicated tester node that allows monitoring and replay of communication patterns at several levels of abstraction from within the communication controller. Assuming single faults and a properly functional tester, it will be possible to pin-point the source of most faults that are detected within the

[2] 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.

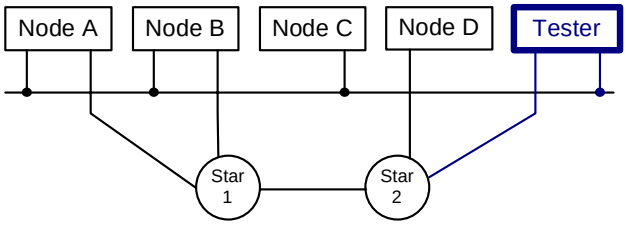

Figure 1: An example for a considered system architecture

recorded data-stream. This works well for the sending path of remote nodes since their is typically a one-to-one mapping of functional abstraction levels in the receiving path of our tester node. However for the receiving path of the remote nodes a stimulus/response mechanism with a loop-back must be facilitated beforehand during system design [1] or with the help of some software agents [4] since the data end-point is usually the application at the remote node.

3 Node Architecture

In order to speed-up the development process of the tester node we adopted a COTS FlexRay node termed <NodeARM> from DECOMSYS[3], see [2]. This device is built using an Altera Excalibur device resembling a hardwired processor stripe including an ARM processor core, a UART, several memories and interfaces, and a large FPGA block. The latter hosts the communications controller that we modified and adopted to our needs as illustrated in Fig. 2.

In particular, we removed the controller to host interface from the standard node and replaced it with a timer, several units for monitoring, replay and fault injection and, two DPRAM blocks and a configuration interface to couple these units with the host processor. The free-running timer provides the time-base for time-stamping of events, e.g., when frames are received or, when pre-defined states are detected (e.g. the beginning of a new communication cycle). With a resolution of $25ns$ even single bits on the bus lines can be over-sampled — FlexRay supports bus speeds up to 10Mb/s — a feature that can become useful when e.g. the operating speed of the bus is unknown.

Monitoring can be activated at different levels of abstraction within the receive path of the communication controller in parallel, e.g. after decoding the received serial bitstream, after the calculation of the checksums or, after the frame processing, etc. To that end, we have developed several encapsulated, stand-alone modules that perform these mechanisms. In this way providing a high degree of flexibility we can easily plug-in or un-plug modules as required.

The idea of the "replay" mechanism is to capture and record a faulty behavior of the system, to further isolate data of the affected node(s) by eventually adapting the relevant recorded data and to re-enact the situation in order to debug and possibly determine the cause of the misbehavior. Depending on the temporal exactness of the replay, however, certain restriction need to be obeyed. E.g., when the tester node re-enacts only the behavior of a minority of nodes the remaining nodes will dominate the distributed clock synchronization and, hence, dictate the temporal instances when replay of recorded frames is invoked. In practice, however, we assume that in frequent cases the tester node will re-enact $n-1$ out of n nodes — meaning that all nodes except the tester and the node under test are removed from the cluster — and thus dictate the temporal events.

The monitoring, replay and fault injection modules transport the data to/from the ARM host processor via two FIFOs implemented with the help of two dual-ported RAM blocks located within the Excalibur processor stripe. All data is encapsulated in dedicated frames consisting of an identifier, a packet size field, a 32-bit wide timestamp and the actual content (i.e. bus-frames, status information, etc.). Apart from the data, the configuration and control information is directly managed with the help of a set of dedicated configuration interface registers.

The embedded software executing on the ARM processor core consists of a set of Linux RTAI tasks that manage the data transfer and, provide triggering and filtering. The data is either stored on a large flash disk or, transferred and stored via a Fast-Ethernet connection to/from a remote host computer. The latter handles the graphical presentation and provides means for post-processing.

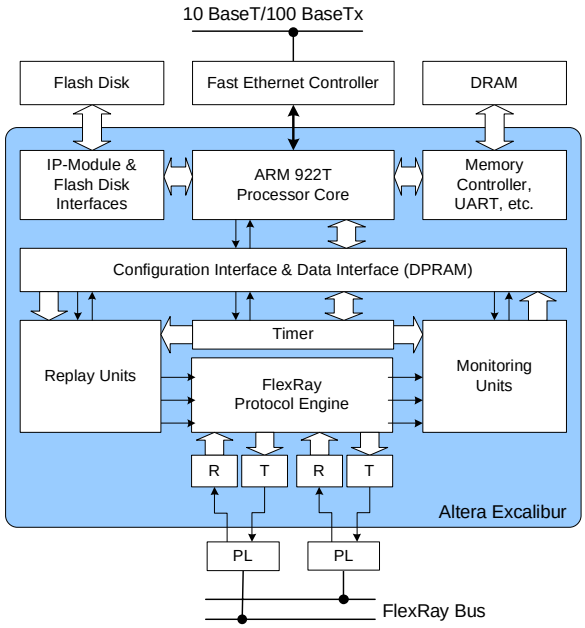

Figure 2: Tester Node Hardware Architecture

[3]http://www.decomsys.com

4 Test and Replay Hardware

Central to our tester hardware are the modules for frame monitoring and replay. These units are each one built using two alternating queues that can hold one maximum sized packet. With the help of a state machine the received packets are re-arranged; i.e., the FlexRay controller uses 16 bits/transfer whereas the host processor operates on 32 bit wide boundaries. Furthermore, a packet identifier and the (re-)calculated size field are added in front of the received data followed by a timestamp that marks the point in time when the information is generated.

Next to units for frame monitoring and replay we have presently implemented modules to record the synchronized cluster time, status information and to allow for an ID-preview using the same architecture. In particular, the synchronized cluster time is usually maintained within the distributed FlexRay controllers with the help of a clock synchronization algorithm that uses the information about the beginning of the static timeslots of the TDMA bus schedule to correct the local clock. Thus, by monitoring the cluster time it is possible to determine the behavior of the system time and to synchronize the timestamps taken from the local free-running timer. Similarly, the status information provides a "health" indicator, whether several mechanisms within the controller provided correct output from the given input data, e.g., if the receive checksum was valid. Finally, an ID-preview packet provides information about the identifiers that were seen on the bus; in that way, the post-processing software gains some knowledge about the nodes that were active without scanning the entire data-log an otherwise although simple but time consuming task.

The principal simplified structure of these modules is outlined in Fig. 3 for the receive path, see also [3]. Every module can be activated by a programmable trigger and, when one entire event is stored in one queue access to the DPRAM is arbitrated. When programmable thresholds within the FIFO are reached after data has been transferred from the queues to the DPRAM an interrupt request to the ARM processor is signaled to indicate that a sector of the FIFO can be read-out. So called "dirty bits" are used to distinguish between regions that contain valid or no data. In case of an overflow — i.e. when the processor is not able to service the interrupt requests in time — the monitoring process is stopped and an overflow interrupt is signalled.

A fundamental requirement for the presentation of the data to the user, the post-processing and a subsequent replay is the correct *temporal order* of packets. This requirement becomes an issue when the timestamps are drawn at the start of packets, since data can be recorded from the two channels in parallel and, data for monitoring is generated and recorded simultaneously at different levels of abstraction. For pure monitoring this requirement could be relaxed when timestamps would be drawn at the end of packets instead; this, however, is not

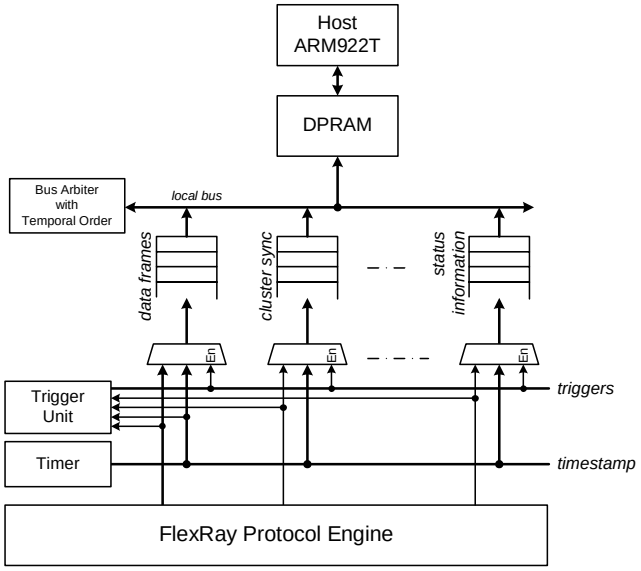

Figure 3: Monitoring Hardware

useful for the replay of recorded packets. In particular, for a timely replay it is evident to draw and insert the timestamp at the start of a packet since the replay hardware compares the timestamp value with the actual time in order to decide when the packet shall be sent. Furthermore, note that usually (*i*) one cannot exactly deduce the start of the packet from a timestamp drawn at the end of a packet due to variable internal processing delays and (*ii*) even if one would relax this precision requirement an additional computational overhead must be considered to re-arrange the packets for the replay.

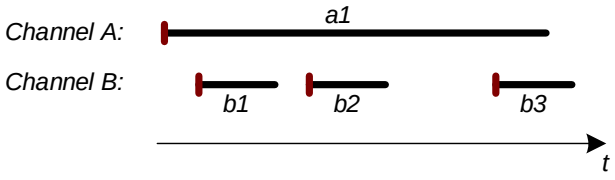

Figure 4: Temporal Order

Fig. 4 illustrates an example where different sized packets are received simultaneously on both channels as it can be the case primarily during the dynamic part of the communication cycles. The horizontal bar represents the length of the various packets while the vertical bar at the start of every packet marks the point in time when the actual timestamp is drawn and inserted. Here packet $a1$ is the first in the temporal order, however, the following packets $b1$ and $b2$ are both entirely received before packet $a1$ is entirely processed. To provide these packets in their temporal order to the application and for subsequent replay the following solutions seem feasible:

1. The packets are copied in the sequence of their completion to the DPRAM, i.e. $b1$, $b2$, $a1$ and $b3$. In this case the temporal order can be established in software while/after recording, however, in these cases a substantial delay and overhead penalty for the additional computation needs to be tolerated.

2. The packets are copied in the sequence of their temporal order to the DPRAM, i.e. $a1$, $b1$, $b2$ and $b3$. Herein two different approaches can be used:

   (a) Shorter packets with a later timestamp and an earlier completion time are queued by the hardware until the first packets are entirely processed.

   (b) The packets are written in their temporal order to the DPRAM. This approach requires knowledge about the temporal order and the size of the packets, e.g. packets $b1$ and $b2$ are written before packet $a1$, however, to address locations following those of $a1$.

Since for our given application the embedded processor was already heavily loaded, we decided to tackle this temporal order requirement in hardware with approach 2(b). Note that approach 2(a) would consume an excessive amount of FPGA internal storage elements due to the possible overlapping of packets for a worst-case scenario.

For an implementation following the temporal order of timestamps and the packet length need to be stored. The bus arbiter simply grants access in the order of requests, however, address generation for the DPRAM write operations needs adjustments to guarantee proper temporal order of packets in memory. The clear advantage of this approach is lower usage of storage elements within the monitoring modules since access to the DPRAM is requested as soon as the packet has been completely received. This advantage, however, comes at the expense of a complex addressing mechanism. For our implementation we used a table approach to manage the temporal order and the address generation for writing to the DPRAM with the following set of rules:

Table Entry E1a: In a first step a global count value that represents the highest registered temporal order value is incremented: $temporal\_order = temporal\_order + 1$

Table Entry E1b: Next, the resulting value is stored as temporal order value to_i with $i \in 1 \ldots n$, where n is the number of monitoring modules and i is the associated row for the particular monitoring module: $to_i = temporal\_order$

Table Entry E2a: When the packet length field becomes available it is written to the column pl_i in the same row.

Table Entry E2b: In addition, the address offset is calculated and written to column off_i using the formula:

$$off_i = (off_{to_i - 1} + pl_{to_i - 1}) \% DPRAM\_Size.$$

In particular this means that the row with a temporal order value by one less than that of the current row is searched. From this row the valid offset and the packet length values are added and stored as address offset value for the current row. The modulo operation simply accounts for the finite size of the DPRAM.

For the boundary conditions it is required to correctly initialize the offset value, i.e. $off_0 = 0$ before the first packet is registered. In addition, the packet length and offset values of the last removed row must be kept for the case when only one further row k is in use; in order to be able to correctly calculate the offset value for row k afterwards.

Whenever a packet is ready to be transferred to the DPRAM it requests the bus from the arbiter and the packet contents are written to the address offset stored in its associated column off_i. Furthermore, the following rules apply:

Table Removal R1a: First, rows holding a temporal order value greater than the one to be removed are decremented by one, i.e.: $to_j = to_j - 1 \quad \forall to_j > to_i$ and $j \in 1 \ldots n$, where n is the number of rows in the table.

Table Removal R1b: Next, the temporal order value of the affected row is cleared, i.e.: $to_i = 0$.

Table Removal R1c: Finally, the global temporal order value must be decremented: $temporal\_order = temporal\_order - 1$.

Using these set of rules the bus arbiter must simply generate addresses ranging from off_i up to $off_i + pl_i$ when the packet associated with row i is transferred. Furthermore, the hardware has to track a $last\_written\_pointer$ in order to avoid generation of interrupt requests before an entire sector of the FIFO has been filled.

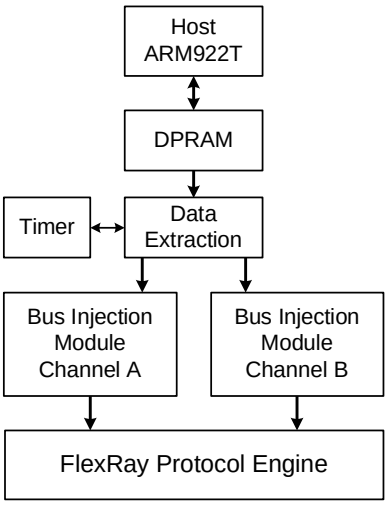

Figure 5: Replay Hardware

Finally, Fig. 5 illustrates the architecture of the replay unit. Central to this unit is a data extraction module that decides whether, where and when to forward data frames from the DPRAM to the ”injection modules”. In asynchronous mode, i.e. when the tester dictates the clock synchronization, a frame is forwarded when the timestamp value within the according data field matches the current value of the timer. The latter is the same as used for data monitoring. In synchronous mode this decision is made, when the timestamp matches with the slot identifier and the cycle count value of the bus schedule. The injection modules in turn resemble a similar architecture as used for the monitoring modules. In particular, they are built from two queues and a state machine that handles the transfer with the FlexRay protocol engine.

5 Conclusion

In this paper we presented an overview of our FPGA-based SoC implementation tailored to support monitoring and replay of FlexRay-based automotive distributed systems. By replacing the controller host interface of the FlexRay communication controller with a large dual ported RAM and some monitoring and replay modules we get access to information otherwise visible only from within the controller. To capture data along with other information we added a time base and a trigger unit with several queues. Timestamps are sampled from the time base and encapsulated into frames with every recorded event. The conditions to initiate the process of monitoring are derived from a trigger that can be programmed via the host. Data is transferred to the host in the correct temporal order by using an address re-write mechanism that was implemented with a set of rules. Similarly to monitoring, a replay mode was implemented in the reverse direction using the same architectural approach to enable fault confinement.

In a next step we want to add some fault injection functionality in order to assess the dependability of the distributed automotive system under inspection.

References

[1] E. Armengaud, A. Steininger, M. Horauer, and R. Pallierer. A Layer Model for the Systematic Test of Time-Triggered Automotive Communication Systems. 5^{th} *IEEE International Workshop on Factory Communication Systems (WFCS'04)*, September 2004. (to appear).

[2] E. Armengaud, A. Steininger, M. Horauer, and R. Pallierer. Design Trade-offs for Systematic Tests of Embedded Communication Systems. *Supplemental Volume of the International Conference on Dependable Systems and Networks (DSN'04)*, pages 118–119, June 2004.

[3] E. Armengaud, A. Steininger, M. Horauer, R. Pallierer, and H. Friedl. A Monitoring Concept for an Automotive Distributed Network - The FlexRay Example. 7^{th} *IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS'04)*, pages 173–178, April 2004.

[4] T. Galla, K. Hummel, and R. Pallierer. Software implemented fault injection for safety-critical distributed systems by means of mobile agents. *Proc. of the Hawaii International Conference on System Sciences (HICSS-37)*, January 2004.

[5] G. Leen and D. Hefferman. In-Vehicle Networks, Expanding Automotive Electronic Systems. *IEEE Transaction on Computers*, pages 88–93, January 2002.

[6] R. Mores, G. Hay, R. Belschner, J. Berwanger, C. Ebner, S. Fluhrer, E. Fuchs, B. Hedenetz, W. Kuffner, A. Krüger, P. Lohrmann, D. Millinger, M. Peller, J. Ruh, A. Schedl, and M. Sprachmann. FlexRay - The Communication System for Advanced Automotive Control Systems. *Society of Automotive Engineers (SAE) 2001 World Congress*, March 2001.