A MONITORING CONCEPT FOR AN AUTOMOTIVE DISTRIBUTED NETWORK - THE FLEXRAY EXAMPLE

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

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

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

Abstract. *This paper discusses the requirements for and presents an implementation of a hardware architecture for monitoring of FlexRay-based automotive distributed networks. The same principles are used in the reverse direction for fault injection and a "replay" mechanism that allows re-enacting recorded situations for debugging and fault confinement. 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

Electronics is the innovation driver in the automotive domain. The use of electronic control units (ECUs) and especially their interoperation allow the establishment of increased and improved functionality in comparison with standalone components (e.g., combining speed with steering information).

For non safety-critical communication (power windows, light control) a number of protocols have become popular, such as LIN, CAN or Byteflight. For safety-critical and more complex applications ("by-wire"-applications and power-train) an industrial consortium is establishing a common communication protocol, called FlexRay[1]. The introduction of such by-wire systems – which essentially means replacing mechanical and hydraulic components by electronic components connected by wires without the provision of a mechanical fallback – will be a substantial innovation step for the automotive industry [1]. However, this will require efficient methods for test and diagnosis to check the functional integrity of all involved components (plus their inter-operation) and hence prevent failures.

[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. Since testing is a prerequisite not only for service and maintenance but during all phases of the design as well, we propose a layer-based approach in order to model, identify and localize faults originating from different sources. To that end, we present in this paper an FPGA design suited for monitoring 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.

2 System model

We consider a system of several ECUs interconnected and operated as specified by the FlexRay protocol. A typical FlexRay Bus-System along with the simplified architecture of a node is illustrated in Fig. 1.

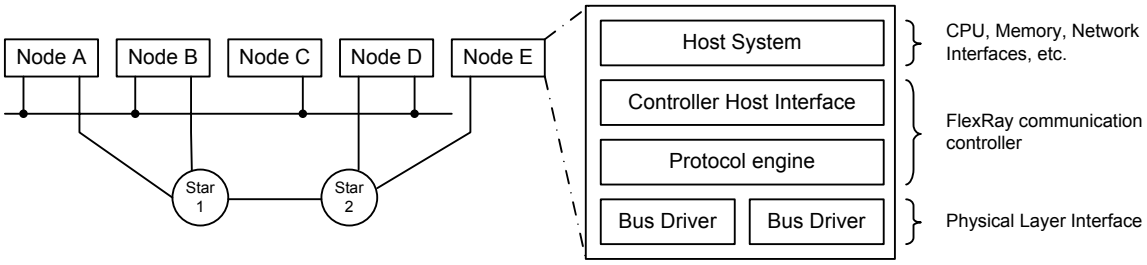

Figure 1: FlexRay Bus-System Example and Node Architecture

In this architecture the controller host interface (CHI) appears as a dual-ported memory block with several additional services (message filtering, network management, error signaling, etc.) and is used as control- and data-interface between the host system and the FlexRay protocol engine next to the transmit and receive channels.

As with all layered architectures, every layer hides particular information from the layers above. E.g. bit-synchronization, CRC-checking, framing, clock ticks, etc., are handled by the FlexRay communications controller and typically only status information is directly accessible by the host processor. Similarly, the physical layer handles and hides the electrical properties from the controller. However, for fault diagnosis this detailed information can become useful especially when the distributed system is operated with settings close to the specified limits. Furthermore, the point in time of message reception is difficult to discern since frames are stored in the CHI for an arbitrarily long time and the frames themselves incorporate no timestamp. Exact timestamps are required when the recorded data shall be retransmitted in the same timely order in a kind of replay fashion in order to re-enact recorded situations; a mechanism strongly argued for by our OEM partners. The idea herein is to strip the data that stems from one/several nodes off the recordings. Next, the network is modified to include the nodes whose data have been removed and the test node. The latter is finally used to re-enact the modified recordings in order to replay previously monitored behavior for debugging and thus emulate a realistic environment for the other nodes.

[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.

3 Requirements Analysis

Monitoring of events in a distributed communication system may be performed at different levels of abstraction. Monitoring at the bit-level, e.g., is useful to probe for the actual bus configuration whereas monitoring at the frame-level is required for debugging of higher level protocols and applications. Without direct monitoring access to different layers diagnosis capabilities are very limited. Therefore one major aim in the design of the diagnosis tool was to allow selective monitoring of different event types from different sources at different layers. For the reconstruction of precedence and causality the recorded events have to be sorted in temporal order. This further allows re-enacting a specific situation.

Another important issue is triggering. In general, brute-force monitoring yields a huge amount of information to be stored and analyzed, while only a small amount is usually needed for diagnosis. In order to avoid time and resource consuming storage and post-processing, on-line trigger mechanisms are required to allow for targeting the observation to the desired interval and for filtering of the relevant information. In addition to specialized data recording and transmission capabilities, fault injection will become useful to assess dependability and reliability properties during system development, requiring several trigger mechanisms as well [3].

The following requirements turned out to be of particular importance for our implementation.

Memory and Data format:

- Identifier: We defined several frame formats to transport all required information between the FlexRay controller and the host processor. To discern between all these frames, and frames monitored from other networks (e.g. CAN, LIN, etc.) a frame identifier shall be used.

- Packet length: A length field at the head of these new/enhanced frames allows for simpler referring to the follow-up packet, an otherwise burdensome task for the host software.

- Data rate: The hard+software should be able to monitor the bus at full speed and at different layers in parallel during test runs that last several hours.

Timestamp:

- Resolution: The granularity of the timestamps shall enable monitoring down to the bit level. In particular, for low level diagnostics we shall be able to attain an over-sampling for bits. With a bit-rate ρ_{chan} on the communication channel and a sampling clock C_s an over-sampling ratio Ω can be attained according to

$$\Omega = \lfloor C_s/\rho_{chan} \rfloor. \tag{1}$$

  The timestamp should allow an identification of every single sample which translates into a timestamp granularity requirement of $1/C_s$.

- Wrap-around: The width of the time base shall be large enough to keep the wrap-arounds rather infrequent since their handling incurs additional processing

overhead. With a granularity of $1/C_s$ and a time base width of b-bits the wrap-around occurs with a period of T_{wa} according to

$$T_{wa} = 2^b/C_s. \tag{2}$$

- Synchronization capabilities: For the "replay" mode or when simultaneously acquired recordings shall be related towards each other regular timestamps and records of the cluster time and/or an external time base (e.g. GPS) are mandatory.

- Chronoscopic time base: In order to guarantee correct temporal ordering and alignment of captured events potential synchronization activities must not lead to discontinuities in the time scale.

- Calibration: Records spanning several hours will exhibit drift deviations when employing ordinary oscillators. A calibration of the (static) frequency error should be possible in any case; a calibration of frequency drift would be desirable. Performing the calibration a posterior in software is computing intensive, whereas a hardware solution as presented in [2] involves considerable hardware efforts.

Trigger:

- Trigger events: To support an analysis on different levels of abstraction different types of events have to be detected by the trigger: Signal edges, a logic state of signals or buses equaling a predefined reference value, data values matching a reference or lying within a given range, time conditions, conditions of the communication channel, error conditions, etc. Furthermore, it is desirable to have the option of formulating combinations of several conditions as the trigger event, like e.g., "*trigger on (CONDITION A* $\wedge$ *CONDITION B)* $\vee$ *(*$\neg$ *CONDITION C)*".

- Trigger sequences: An even higher selectivity can be attained, if sequences of trigger conditions can be applied, like "*trigger on CONDITION A first, and then on CONDITION B next*". Obviously, this allows a more precise positioning within the control flow on the bus. Similarly, an event counter for the trigger is very useful "*trigger on CONDITION A 3 times*".

- Trigger position within the record: For most sequences to be observed some unique event is known such that the trigger event can start the acquisition. For debugging, on the other hand, it will often be the case that the failure must be used as a trigger event where the bus activities and states before this event are of interest. As a consequence, both a post-trigger and a pre-trigger are required.

- Trigger response time: Of course, it is highly desirable to have an immediate trigger response, in particular, recording shall start within the same acquisition clock cycle as the trigger event. Considering the requirement for more complex trigger conditions like combinations or sequences of events this immediate response is not realistic. Therefore, a subset of simpler trigger functions shall be available that guarantee very fast trigger response when necessary while the more complex trigger functions may be helpful in many other cases.

We have performed this requirement analysis with the FlexRay target in mind in front of the actual implementation. The problems we encountered turned out quite generic and, hence, our analysis can be applied to many other embedded distributed networks.

4 Implementation of the Monitoring Device

For the monitoring hard+software our starting point was to use a COTS FlexRay controller and implement the required modifications. This approach relieved us from implementing the standard FlexRay controller hardware which allowed us to concentrate our efforts on the monitoring aspects. Furthermore, in this way we avoided compatibility problems with the standard controllers we use as monitoring target.

The main problem we were facing with this approach was the attainable data rate. The COTS FlexRay controller architecture has been optimized for the data rates occurring during normal bus operation. According to our monitoring requirements the implementation must be able to handle multiple data streams in parallel when used as a monitoring device which means that a much higher data rate had to be handled. This issue turned out as one of the most critical aspects in our architectural considerations.

According to the strategy outlined above we replaced the controller to host interface (CHI) of a COTS FlexRay controller implementation as illustrated in Fig. 1 with a large dual ported RAM interface and added several hardware units – in particular, a time base and a trigger unit – to the existing FlexRay controller. Fig. 2 shows a block-diagram of the monitoring hardware implementation without the interfaces required to configure and program these units, and the host interfaces (e.g. memory, Ethernet interface, etc.).

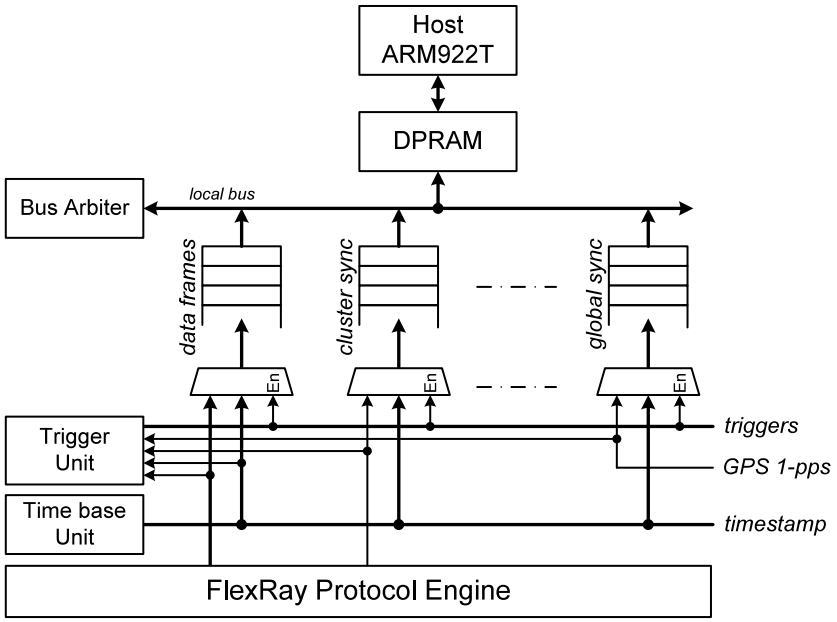

Figure 2: Monitoring hardware block-diagram

In principle, after a trigger every event that is recorded using this architecture is encapsulated in a dedicated frame format optimized for the processing by the host (e.g.

data frames, cluster sync frame, global sync frame, etc.) and stored in a dedicated queue. Dedicated queues are required because the information to be recorded is usually generated in parallel. A bus arbiter manages the local bus and when scheduled, copies the contents of the queues into the dual-ported memory. The dual ported memory is organized as a FIFO; whenever a certain threshold is reached an interrupt request is signaled to the host processor. In the respective interrupt service routine the processor empties the FIFO and stores the relevant information in a large external dynamic memory bank. Higher level software succinctly processes the stored information and copies it either to a large flash-disk or streams it via a Ethernet interface to a remote host for visualization and further processing.

For the replay and fault injection modes a similar architecture is used, although, the data path is in the reversed direction and some additional units are added (e.g. to extract the required information from the frames).

As target for the implementation of the presented monitoring ECU we adapted the `Node<ARM>` an automotive prototyping platform for FlexRay from Decomsys GmbH[3] exploiting Altera's Excalibur architecture. In particular, the EPA4 device offers an embedded ARM922T processor hardcore with several integrated peripherals and a 400k FPGA that is used to implement the FlexRay communications controller and the hardware support for monitoring, replay and fault injection. Designed for 10 Mb/s operation our actual implementation achieves an over-sampling of $\Omega = 8$ and a wrap around period of $T_{wa} = 53sec.$ according to Equ. 1 and 2, respectively.

5 Conclusions

We presented several requirements and an overview of our FPGA implementation tailored to support monitoring of FlexRay-based automotive distributed systems. By replacing the controller host interface of the FlexRay communication controller with a large dual ported RAM 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 in various different ways. Similarly, the trigger is used for fault injection and a "replay" mode in the reverse direction to enable fault confinement and assessment of the dependability of the distributed automotive system under inspection.

References

[1] Leen G. and Hefferman D.: In-Vehicle Networks, Expanding Automotive Electronic Systems. IEEE Transaction on Computers, pp. 88-93, January 2002.

[2] Schossmaier K., Schmid U., Horauer M. and Loy D.: Specification and Implementation of the Universal Time Coordinated Synchronization Unit (UTCSU). Journal of Real-Time Systems, May 1997, No. 3, Vol. 12, pp. 295-327.

[3] Hsueh M.C., Tsai T.K. and Iyer R.K.: Fault Injection Techniques and Tools. IEEE Transactions on Computer, Vol. 30, No. 4, pp. 75-82, 1997.

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