A Hardware/Software Co-Design Approach for Ethernet Controllers to Support Time-triggered Traffic in the Upcoming IEEE TSN Standards

Friedrich Groß, Till Steinbach, Franz Korf, Thomas C. Schmidt, Bernd Schwarz
Department of Computer Science
Hamburg University of Applied Sciences, Germany
{friedrich.gross, till.steinbach, korf, schmidt, schwarz}@informatik.haw-hamburg.de

***Abstract*—Due to the increasing bandwidth and timing requirements, next generation communication backbones in cars will most likely base on real-time Ethernet variants that satisfy the demands of the new automotive applications. The upcoming IEEE 802.1Qbv standard shows communication approaches based on coordinated time devision multiple access (TDMA) to be good candidates for providing communication with determinism and highly precise timing. Implementing time-triggered architectures in software requires significant development effort and computational power.**

This paper shows a scalable HW/SW co-design approach for new real-time Ethernet controllers based on the partitioning into communication and application components. The tasks required for communication are divided: Time-critical and computationally intensive parts are realised in dedicated hardware modules allowing the attached CPU to fulfil the timing requirements of the automotive application without interference. The evaluation using a Field Programmable Gate Array (FPGA) based prototype implementation shows that the precision for the time-triggered transmission and the performance of the proposed implementation of the required synchronisation protocols satisfies the requirements of applications in the automotive domain.

I. Introduction

Time-triggered Ethernet variants, such as Profinet [1], TTEthernet (AS6802) [2], or the upcoming IEEE 802.1Qbv [3] extension defined by the Time-Sensitive Networking (TSN) Task Group, that operate using a coordinated time division multiple access (TDMA) policy, become increasingly popular for time-critical applications. Due to their deterministic behaviour, low latency and jitter, they open new application domains for Ethernet in the automotive domain. In particular for advanced driver assistance systems (ADAS) that require end-to-end latency down to 100 µs, together with low jitter.

Due to various mixed critical applications, standard best-effort (BE) and real-time Ethernet traffic share the same physical infrastructure within these networks. With network speeds of currently up to 1 Gbit/s the traffic shaping at the distributed nodes as well as the interconnecting switches, requires scheduling with highest precision that is hard to achieve using a software-only communication stack. Further, software implementations utilise significant computation time, for example for the classification of incoming packets.

In the automotive domain, applications typically operate asynchronously on the electronic control units (ECU) and cannot provide the precision required. Today's automotive architectures, such as present AUTOSAR [4] versions, do not provide sufficient scheduling mechanisms to synchronise time-triggered transmission and reception with the application tasks. Thus, controllers for legacys fieldbusses using the time-triggered paradigm, e.g. FlexRay [5], provide a co-processor or hardware support for the precise timing. For real-time Ethernet such advanced Ethernet controllers are currently unavailable.

Cost and energy efficiency are major target goals for the development of electronic components in the automotive domain. Thus, the design must flexibly adapt to different use-cases and applications with different levels of timing requirements. This degree of freedom can be well achieved with hardware/software co-design, where the partitioning between hardware modules and features implemented in software can be variably chosen and deployed. In addition the system can be specifically configured for the applications use-case and fine tuned to comply with its requirements

This paper contributes a scalable approach for the hardware/software co-design of an Ethernet controller that supports time-triggered communication and synchronisation while using a legacy Ethernet MAC. Different hardware/software partitioning designs are analysed and discussed concerning hardware resources and timing advantages. It is shown, how the careful integration of hardware support reduces load of the attached microcontroller.

The evaluation of the prototype implementation using a Xilinx Virtex-6 FPGA shows the achievable time-triggered precision, even when using a legacy (non-real-time) Ethernet MAC. Due to the hardware support for the synchronous transmission, ultra low jitter can be realised. In comparison with a software only implementation running on an ARM-9 microcontroller, the processors utilisation can be significantly reduced. This saved computation time can be used by the developer for more complex applications and advanced features.

The paper is organised as follows: In Section II, we introduce the concepts of time-triggered Ethernet and present previous and related work. Section III presents the concept and architecture. In Section IV, details from the implementation as well as the evaluation results are shown and discussed. Finally, Section V concludes the work and gives an outlook.

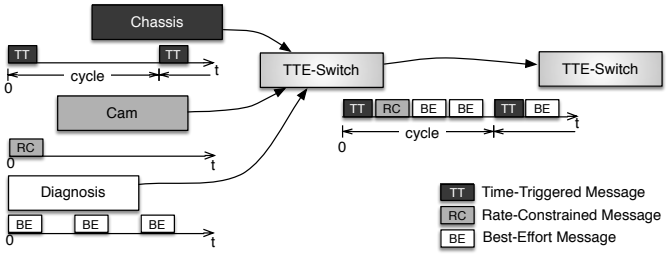

Fig. 1. Prioritising and time-triggered media access in time-triggered Ethernet

II. Background & Related Work

Time-triggered real-time Ethernet extensions such as TTEthernet (AS6802) [2] or IEEE 802.1Qbv [3] operate according to a *coordinated time division multiple access (TDMA)* scheme. Messages are transferred at dedicated offline configured time slots in a globally synchronised cycle. This TDMA based media access strategy allows for a completely deterministic transmission, without congestion at outgoing line-cards and thus with predictable latency and jitter. For synchronous operation the local time of all network participants is adjusted using a synchronisation protocol.

Besides the time-triggered message class, two event-triggered message classes are defined in TTEthernet: *Rate-constrained* (RC) traffic is used for the transmission of messages with less rigid timing requirements. It limits the streams bandwidth and prioritises according to the strategy of the *ARINC-664 (AFDX)* protocol [6].

The third class is *best-effort* (BE) traffic, that conforms to standard Ethernet frames and is transmitted with the lowest priority. It allows the integration of hosts that do not implement the real-time protocol and remain unsynchronised. These nodes communicate using only best-effort frames. In Figure 1 a typical example of the media access and prioritisation for messages of the three different traffic classes is shown.

All time-triggered Ethernet protocols share their requirement for high precision clock synchronisation and scheduled frame transmission. It is challenging to achieve this level of precision without dedicated hardware support, as an operating system with support for strict priority scheduling and very fast context switches is required [7]. Time-triggered fieldbusses, such as FlexRay [5], have similar requirements regarding the bus access. To overcome the lack of real-time scheduling support in the application processor, typically a co-processor manages the time-critical bus access. In this architecture, the host controller is only responsible for filling buffers, which define the asynchronous interface to the co-processor, all time-critical bus access related tasks are executed by the dedicated FlexRay controller. As off-the-shelve Ethernet controllers are not designed for scheduled time-triggered communication, up to now a similar functionality for real-time Ethernet is unavailable.

Due to the early state of IEEE 802.1Qbv the prototype implementation uses the time-triggered traffic class of AS6802. Nevertheless, the nature of time-triggered traffic will allow to transfer the results to an upcoming IEEE standard with scheduled traffic.

Previous & Related Work

Steinhammer and Ademaj [8] provide a reference design for a time-triggered Ethernet (TTEthernet) controller in hardware. In their implementation timestamping is realised in hardware. In contrast to this work, the synchronisation is realised on an attached CPU. Weibel [9] shows hardware timestamping with an additional PHY while using legacy hardware at the receiver. The downside of this approach is the unguaranteed path symmetry, resulting in lower precision.

In previous work, the challenges of a pure software stack for time-triggered communication on microcontrollers was shown [7]. The implementation based on a system-on-chip (SoC) design with hardware timestamping uses a fixed-point timer with a resolution of $10\,\text{ns}$ for the scheduling. Due to the software based approach a very high CPU utilisation (up to $90\,\%$) was measured in worst-case on the $200\,\text{MHz}$ test system. This high CPU utilisation shows the demand for a real-time Ethernet co-processor to hand over computationally intensive parts of the software stack.

III. Concept & Architecture

The concept shown describes a hardware extension for time-triggered Ethernet, that allows to run software applications completely independent from the time-triggered protocol. The hardware/software co-design approach allows certain parts, such as the synchronisation protocol, to scale with the timing requirements by selecting tasks to run completely in hardware, completely in software, or any combination of both. The concept does not require the development of a special Ethernet MAC. A legacy MAC that offers a 1-to-1 connection to receive and send Ethernet-Frames can be attached. Such Ethernet MACs are offered e.g. by Xilinx and Altera. The architecture is optimised for the Xilinx *XPS LL TEMAC* MAC [10].

Figure 2 shows an overview over the architecture. It contains an application CPU, RX- and TX-buffers, a timestamping unit, a fixedpoint timer and the legacy MAC. A switch module controls the classification of incoming packets and a guard controls the outgoing traffic shaping. All modules are interconnected using a CPU bus. For the physical media access an Ethernet PHY (e.g. for automotive Ethernet [11]) is attached.

A. Time-Triggered Transmission

In the architecture time-triggered transmission is done using hardware. In contrast to software, a hardware module can implement tasks nearly jitter free.

In the *RX-buffer* module, a send buffer can be individually created for each traffic class and critical traffic (CT) ID. This allows to individually choose the buffer (memory size, double- or queued buffer) according to the application requirements. The send-buffers are connected through a *guard* module with the 1-to-1 interface of the Ethernet MAC.

The major challenge of this design is, that the MAC has its own transmission buffer. If the MAC would be allowed to

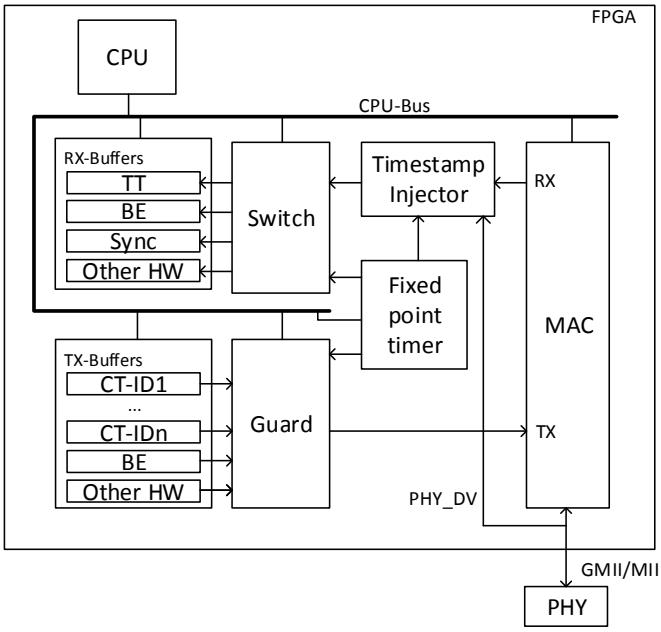

Fig. 2. Overview over the architecture of the FPGA implementation

queue up frames it could be in busy state when a time-triggered message has to be sent and would delay the message. The simplest way to solve this problem is to prohibit media access to messages for all event-triggered send buffers right before a time-triggered message is sent. The prohibit time must be as long as the serialisation time of the transmission buffer inside the MAC. With this solution a huge amount of bandwidth would be lost.

To overcome this problem, a so called *guard* module is placed between the send-buffers and the MAC. The *guard* knows the schedule of all time-triggered messages and makes sure that TT messages will never be delayed. If an event-triggered messages (e.g. best-effort (BE) message) is pending, the BE buffer module reports the size of the message to the guard. Based on the size, the bandwidth on the media and the scheduling, the guard calculates whether the BE message would collide with a time-triggered transmission or could finish its transmission in time.

Example: The Ethernet speed is 100 Mbit/s. At the time 1000 ns [t_now] the BE-buffer asks the guard for permission to send a message with a size of 100 B. The next time-triggered message is scheduled at 10 000 ns [t_next_message]. The serialisation time of 100 B at 100 Mbit/s is 8000 ns [t_serialize]. The inter-frame-gap at a 100 Mbit/s Ethernet is 960 ns [t_ifg].

If condition 1 is true, the packet can be sent:

$$\begin{aligned} t\_now + t\_serialize + t\_ifg &< t\_next\_message \\ 1000ns + 8000ns + 960ns &< 10000ns \\ 9960ns &< 10000ns \end{aligned} \quad (1)$$

In this example the condition is true.

The guard introduces a further challenge: The bus speed between the hardware buffers and the Ethernet linespeed are usually different. In our case the bus speed is 3.2 Gbit/s. Though, the MACs transmission buffer can be filled much faster as it is emptied. Therefore, a low priority frame could be sent fast to the MAC and if there is another frame to be sent, the condition 1 would still be true, although the transmission buffer inside the MAC is still filled. The result could be a delayed time-triggered message.

To solve this problem the guard has to stop giving sending permission to all buffers during the serialisation time of the last frame forwarded to the MAC. This compensates the bandwidth differences of the physical media and the bus. The advantage of the guard concept is, that no Ethernet bandwidth is lost, while the CPU can asynchronously put Ethernet frames in the buffers. The hardware handles time-triggered transmission. All operations of the guard are line-operations (no multiplier or divider is required). Though, the hardware consumption is low.

B. Packet Reception & Timestamping

The precision of the time synchronisation depends on the accuracy of the reception timestamp. For highest precision, the timestamping must be realised between PHY and MAC. In this architecture the PHY data valid (PHY_DV) signal of the MII/GMII interface is used to record the reception time. When the frame leaves the MAC to the internal hardware modules, this timestamp is injected into the frame header. A check mechanism that uses a *receive complete* interrupt from the MAC is implemented to observe frames dropped due to a bad checksum or an overloaded buffer in the MAC. A packet switch is connected at the timestamp injector. It classifies and forwards frames to the different hardware modules based on the timestamp, frame header and type field and is implemented as a cut-through switch to minimise the delay. The switch component significantly reduces the CPU load in comparison with a software only stack: During a traffic burst, low priority frames may be dropped, e.g. when the buffers are full. At the same time, time-triggered frames must be reliably stored. Without the switch module, each frame has to be analysed in software after reception to prevent packet loss, resulting in an enormous number of context switches in the CPU.

C. Clock Synchronisation

Several design approaches with variable scalability exist for implementing the clock synchronisation protocol [2]. A first configuration implements the whole protocol in hardware, including the rate correction. The approach allows to run the CPU asynchronously with the rest of the system and requires no computation time of the microprocessor for the synchronisation. Further it allows advanced powersaving by sending the CPU to sleep without loosing synchronisation. This approach requires most hardware resources.

The second design implements most of the synchronisation protocol in software, utilising only hardware timestamping. The accuracy is comparable with the hardware implementation while the hardware consumption is low. In this design, the CPU utilisation is high.

TABLE I
HARDWARE CONSUMPTION OF IMPORTANT MODULES

	LUT		Register	
Hardware module	**[#]**	**[%]**	**[#]**	**[%]**
Switch	100	0.21	179	0.19
Timestamp injector	109	0.23	72	0.08
Fixed point timer (55 Bit)	55	0.12	55	0.06
Synchronisation client	1100	2.36	736	0.79
Guard	407	0.87	155	0.17
Queued Buffer	207	0.44	360	0.39

The third example design is a trade-off between both concepts: A hardware module receives the synchronisation frames and extracts the parts required for the synchronisation process. In the AS6802 protocol it is necessary to count bits set to 1 in a 32 bit wide vector field inside the synchronisation frame. This field is called membership_new_vector. Such calculations can be efficiently done in hardware to reduce the CPU utilisation.

As shown in table I the Hardware implementation of the synchronisation protocol requires most of the hardware resources, though a trade-off solution is worth considering. The co-design approach allows to flexibly choose the partitioning in hardware and software modules.

For all partitioning schemes rate-correction is implemented using a fixedpoint timer with 32 bit before and 24 bit after the fixedpoint. On startup 1.0 is added every clock tick. After the offset-correction, a new addup is calculated, allowing to adjust the speed of the timer to the synchronised clock.

IV. IMPLEMENTATION & EVALUATION RESULTS

The prototype implementation used for the architecture evaluation is based on the ML605 board from Xilinx with a Virtex-6 FPGA. Table I gives an overview over the hardware consumption of important modules.

A. Time-Triggered Transmission

The *guard* module was implemented as a multiplexer that decides based on current time and the configured scheduling of messages. Our evaluation shows that the presented architecture is able to schedule time-triggered frames very close to each other. The implementation requires only the inter-frame-gap between the frames, no additional management overhead is required. The evaluation shows that due to the guard module no time-triggered frame is delayed by an event-triggered frame with lower priority.

Figure 3 shows the jitter of the scheduled time-triggered messages. To avoid measurement inaccuracy due to rate-correction the jitter was measured with the clock synchronisation disabled. The graph plotting the jitter of the interarrival time over a period of 80 cycles shows a maximum jitter of 80 ns, with accumulations from 70 ns to 80 ns and 0 ns to 10 ns marked in gray. We traced the jitter back to the legacy Ethernet MAC, consequential the result can be further improved using a tailored MAC module. Compared with the precision of the software implementation (approx. 1 µs jitter) implemented in previous work [7], the timing accuracy could be improved by 10 times with the hardware co-processor.

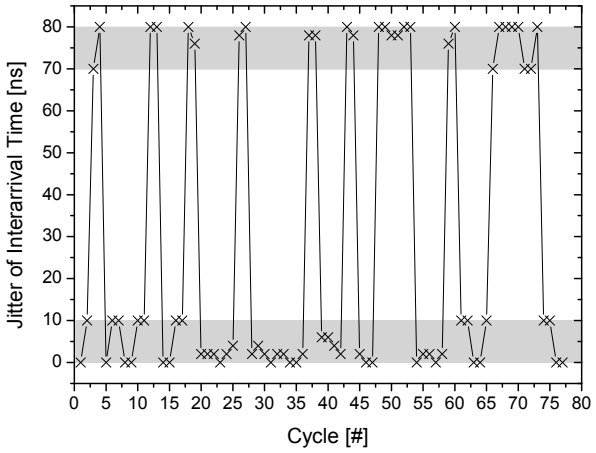

Fig. 3. Jitter of time-triggered messages (measured using interarrival time)

B. Packet Reception & Timestamping

The *PHY_DV* (PHY data valid) signal from the MII interface triggers the *Timestamp Injector* module to timestamp the receipted frame. This signal jitters by approximately 10 ns and is observed 390 ns prior to the deserialisation of first bit of the destination MAC address. The timestamps are stored in a first-in-first-out (FIFO) buffer, parallel to the MAC. To keep the timestamp FIFO synchronous with the frame FIFO, the *receive complete* interrupt from the MAC is observed to drop timestamps of frames that are invalidated in the MAC (e.g. due to the checksum). The *Timestamp Injector* (see Figure 2) works on the fly, delaying incoming frames only for 4 clock ticks (40 ns).

The *Switch module* is implemented as a cut-through switch and causes a delay of 6 additional clock ticks (60 ns). The forwarding decisions are freely programmable based on the MAC addresses, Ethernet-Type field, reception timestamp, current time, configured time-triggered scheduling and utilisation of the RX-buffer. With this set of conditions real-time messages that are received out of schedule can be discarded to prevent interrupts of corrupted frames at the host CPU. Further, unused traffic classes, such as best-effort traffic, can be completely disabled to only allow interrupts of real-time traffic in the CPU. In a software-only stack, packet classification and reception causes high CPU utilisation on heavily saturated links (up to 90 % on a ARM-9 [7]), because every received frame has to be immediately analysed in software. The hardware architecture presented in this paper provides interrupts for each traffic class. Based on these configurable interrupts, the CPU can now prioritise the processing of incoming mixed-critical frames to work most efficiently.

C. Clock Synchronisation

The synchronisation [2] is implemented completely in hardware. In addition a rate-correction module is implemented. The rate correction is triggered directly after the offset correction of the AS6802 algorithm.

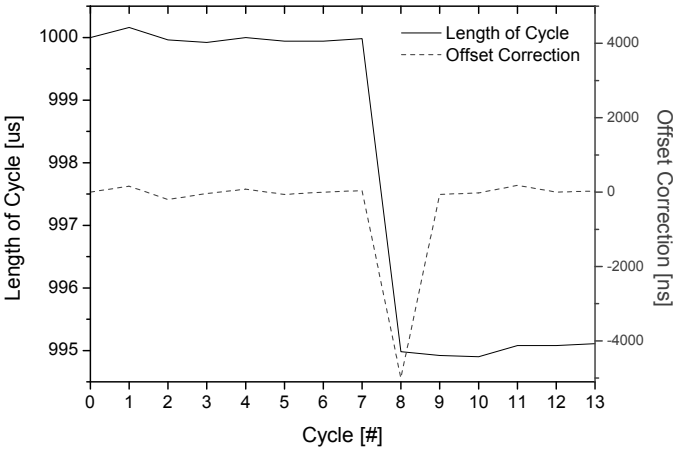

Fig. 4. Behaviour of the Hardware Implementation of the Synchronisation Protocol

The measurement setup is configured with a synchronisation master that sends synchronisation frames with a cycle of 1 ms and a jitter of approximately 400 ns. To show the fast reaction of the synchronisation, after 8 cycles a sudden clock change (5000 ppm) at the master was emulated. Figure 4 shows the behaviour of the synchronisation client. The client is able to adapt to the speed of the master clock within one cycle.

Based on the offset correction value, the new addup value for the fixedpoint timer is calculated:

Example: The cycle duration is 3 ms (3 000 000 ns) [t_cycle_duration]. The offset correction is 30 µs (−30 000 ns) [t_offset_correction]. The current addup in the fixpedpoint timer is 1.0 [addup].

$$addup = addup + (\frac{t\_cycle\_duration}{t\_offset\_correction}) \qquad (2)$$

For a precise result, the sum in equation 2 must be replaced by a multiplication. As the multiplication requires significant hardware resources the equation was approximated using the sum. As the addup is usually very close to 1.0 and the rate correction is usually lower than one percent of the cycle duration, the arithmetical error between multiplication and sum is very low.

The rate-correction allows the synchronisation client to adjust its clock very fast. Consequently, the offset correction is very low. A high offset correction indicates that at the end of the cycle the clocks have a high divergence. With a high divergence, large time windows for time-triggered messages are required, wasting the available bandwidth. Hence the performance of the clock rate-correction can save a significant amount of bandwidth.

V. Conclusion & Outlook

Specialised hardware extensions for standard Ethernet controllers can significantly improve the performance of real-time Ethernet protocols. Especially mixed-critical applications with synchronous time-triggered and asynchronous event-triggered traffic can profit from reduced CPU load and less software complexity. The presented architecture shows a hardware/software co-design concept for a time-triggered-ready Ethernet controller that is attachable to standard legacy Ethernet MACs. The architecture is configurable to scale with the desired application. The evaluation of the prototype implementation using a Virtex-6 FPGA shows a significant performance gain compared with software-only communication stacks. The precision of time-triggered frames can be improved by over 10 times. The rate-correction allows to adapt to leaps in the clock speed in only 1 cycle.

To reduce development effort, the implementation uses FPGA internal memory for all buffers. This is only possible on expensive FPGAs with significant hardware resources. In future work we plan on using external memory, to reduce hardware costs without significant conceptual changes. Further we will implement other traffic classes such as the credit based shaping (CBS) from the IEEE 802.3Qav standard to allow for mixed-critical applications with different traffic classes and define interfaces to support the synchronisation of applications with the global schedule to further reduce jitter and improve the real-time performance of the implementation.

Acknowledgements

This work is funded by the Federal Ministry of Education and Research of Germany (BMBF) within the RECBAR project.

References

[1] PROFIBUS & PROFINET International, “Profinet,” Karlsruhe. [Online]. Available: http://www.profibus.com/technology/profinet

[2] Society of Automotive Engineers - AS-2D Time Triggered Systems and Architecture Committee, “Time-Triggered Ethernet AS6802,” SAE Aerospace, Nov. 2011. [Online]. Available: http://standards.sae.org/as6802/

[3] Institute of Electrical and Electronics Engineers, “802.1Qbv - Bridges and Bridged Networks - Amendment: Enhancements for Scheduled Traffic,” IEEE, Draft Standard P802.1Qbv/D1.0, Dec. 2013.

[4] AUTOSAR Development Cooperation, “AUTomotive Open System ARchitecture.” [Online]. Available: http://www.autosar.org

[5] FlexRay Consortium, “Protocol Specification,” FlexRay Consortium, Stuttgart, Specification 3.0.1, Oct. 2010.

[6] Aeronautical Radio Incorporated, “Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network,” ARINC, Standard ARINC Report 664P7-1, 2009.

[7] K. Müller, T. Steinbach, F. Korf, and T. C. Schmidt, “A Real-time Ethernet Prototype Platform for Automotive Applications,” in *2011 IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. Piscataway, New Jersey: IEEE Press, Sep. 2011, pp. 221–225.

[8] K. Steinhammer and A. Ademaj, “Hardware Implementation of the Time-Triggered Ethernet Controller,” in *IFIP Advances in Information and Communication Technology*, 2007, pp. 325–338.

[9] H. Weibel, “High Precision Clock Synchronization according to IEEE 1588 Implementation and Performance Issues,” in *Embedded World Conference 2005*, Design & Elektronik, WEKA-Fachzeitschriften-Verlag, Feb. 2005, pp. 981–989.

[10] Xilinx Inc., “LogiCORE IP XPS LL TEMAC,” Xilinx Inc., Dec. 2010. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/xps_ll_temac.pdf

[11] Broadcom Corporation, “Product Brief BCM89810, BroadR-Reach Single-Port Automotive Ethernet Transceiver,” Broadcom Corporation, Tech. Rep., Oct. 2011. [Online]. Available: http://www.broadcom.com/collateral/pb/89810-PB00-R.pdf