Design and Evaluation of a Low-Latency AVB Ethernet Endpoint based on ARM SoC

Christian Herber, Ammar Saeed, Andreas Herkersdorf

Technische Universität München - Institute for Integrated Systems
Munich, Germany
{christian.herber, ammar.saeed, herkersdorf}@tum.de

***Abstract*—Communication requirements in automotive electronics are steadily increasing. To satisfy this demand and enable future automotive embedded architectures, new interconnect technologies are needed. Audio Video Bridging (AVB) Ethernet is a promising candidate to accomplish this as it features time-sensitive and synchronous communication in combination with high bit rates. However, there is a lack of commercial products as well as research regarding AVB-capable system-on-chips (SoCs).**

In this paper, we investigate how and at what cost a legacy Ethernet MAC can be enhanced into an AVB Ethernet controller. Using FPGA prototyping and a real system based on an ARM Cortex-A9 SoC running Linux, we conducted a series of experiments to evaluate important performance metrics and to validate our design decisions. We achieved frame release latencies of less than 6 µs and time-synchronization with an endpoint-induced inaccuracy of up to 8 µs.

***Index Terms*—Audio Video Bridging, Ethernet, Automotive Electronics.**

I. Introduction

Innovation in automotive technology is highly reliant on advancements in electronic functions like infotainment and driver assistance. As more and more functions get introduced, the communication requirements steadily increase as well [1]. This trend will be ongoing and be further intensified with transition into highly autonomous driving and integration of cars into cyber-physical systems. Providing time-sensitive and high bandwidth communication is therefore a major challenge for enabling future innovations.

The growing communication requirements cannot be satisfied by currently employed interconnect technologies [2]. FlexRay and CAN (FD) don not support bandwidth greater than 10 to 20 Mbit/s and MOST has limited real-time support. Ethernet has always been interesting due to its wide adoption and high throughput, but has not been used in automotive electronics due to wiring cost and the need for specialized, real-time capable endpoints and switches.

Audio Video Bridging (AVB) is specified by a set of IEEE standards allowing time-sensitive and synchronous Ethernet communication. It was originally developed for multimedia applications, but its properties also fit the requirements of car manufacturers. Additionally, due to IEEE standardization, it has the potential of wide-spread adoption in low-cost networking devices. Because of its acceptance in the automotive domain, AVB's successor that is under development was renamed to Time-Sensitive Networking (TSN).

AVB combines stream reservation (IEEE 801.1Qat), traffic shaping and flow control (IEEE 801.1Qav) to guarantee time-sensitive communication. AVB packets are transmitted in streams, which are grouped into traffic classes and have a reserved bandwidth. Usually, three traffic classes are proposed (Class A, Class B, and legacy traffic in descending priority). In addition to strict prioritization among classes, all switches and talker endpoints (stream sources) employ a credit based shaping (CBS) for each class based on its reserved bandwidth. Talker endpoints also use the same traffic shaping for each stream. While the standard allows dynamic reservation of streaming resources, automotive implementations are expected to rely on static implementations.

AVB uses the precision time protocol (IEEE 802.1AS) to enable clock synchronization between distributed nodes. A globally synchronized clock improves the accuracy of distributed control applications [3]. In automotive scenarios, it allows e.g. synchronous sensor sampling and compensation for transmission delays within the network to produce equal presentation times within distributed nodes [4].

While AVB found broad acceptance, it has not yet been implemented in commercial SoCs. In consequence, there is also a lack of experimental data regarding AVB endpoints. In this paper, we present a HW/SW implementation of the mentioned AVB protocols targeted for automotive use. Our implementation includes an FPGA prototype of an AVB-capable Ethernet controller and a Linux kernel space driver. The main contribution lies within the experimental evaluation, in which we determine hardware costs, latencies and synchronization errors. The results obtained clearly demonstrate the applicability of our system in low-latency and synchronous networking operations.

II. Background & Related Work

AVB is a relatively young technology with active research, but limited number of commercial products. Most products are either switches or audio-specific endpoints. AVB Ethernet switches are available from e.g. NETGEAR, Pathway, or Extreme Networks. As endpoints, AVB-capable loudspeakers, signal processors, and amplifiers are available. Intel's I210 Ethernet controller also supports AVB, but architectural details are not publicly available. So far, it was used in two mainboards by ASRock and Supermicro, respectively.

No AVB-capable network interface card exists. Xilinx' Tri-Mode Media Access Controller (TEMAC) has AVB features, yet only supports one AVB traffic class and one stream. This architecture is inapplicable, if e.g. one audio and video stream should be sent. To be applicable for highly integrated automotive ECUs where also multiple control streams may be necessary in addition to audio/video data, a flexible solution supporting multiple streams and traffic classes is needed. Very recently (in 02/2015), an integrated ARM SoC with AVB functionalities was announced by Freescale (i.MX6SX). However, architectural details or performance data are also not available.

We implemented our own AVB Ethernet controller integrated with an ARM SoC for three major reasons: First, it allows us to take and evaluate HW/SW partitioning and architectural decisions. Second, we can use this implementation to evaluate HW resource requirements of an AVB controller. Third, it allows us to embed timing measurements directly into the hardware, thus enabling a precise assessment of important performance metrics. We use this advantage to conduct a series of experiments in Section IV.

Due to a lack of suitable hardware, other research activities have been mainly focused on simulation and formal timing analysis. In the field of real-time analysis, steady progress is made, but no universally accepted approach exists. Analyses have been proposed using compositional analysis [5], [6], real-time calculus [7], and network calculus [8], [9]. Bordoloi et al. [10] provide a formal refinement of previous analyses.

The mentioned analyses focus on networking delays and consider an idealized behavior of endpoints and switches. In this work, we used a real system and considered endpoint-related performance metrics rather than networking delays.

Another field of research focuses on the quality of time-synchronization achievable in AVB. Different sources of clock synchronization errors including endpoint implementation, networking interference, and temperature variations have been researched.

Mahmood et al. [11] evaluated software and hardware implementations of timestamping with respect to clock synchronization performance. The authors measured interrupt handling and timestamping delays for software implementations. Based on this data for two Intel platforms, they simulated the synchronization performance. Depending on the synchronization interval and platform, they obtained root mean square (RMS) synchronization errors between 409 ns and 1.78 μs. We also follow a software-based approach, but use experimental evaluation instead of simulation to assess synchronization errors.

Kern et al. [12] conducted experiments to determine time synchronization accuracy in AVB for varying temperature conditions. They found the synchronization errors to be sub-microsecond even for extreme temperature shocks.

Lim et al. [13] simulated an AVB Ethernet-based in-car network to assess synchronization errors. In a daisy-chain topology with seven hops (six switches), errors up to 985 ns were observed. The simulation assumes an ideal implementation of the synchronization protocol in the endpoints. Our experiments specifically focus on errors originating from non-idealities within the endpoint.

III. Design & Implementation

In this section, we present a composable architecture solution, which extends an Ethernet MAC into an AVB controller. We propose a HW/SW partitioning targeted to comply with automotive application requirements while minimizing resource usage.

We implemented the design in VHDL using a SoC with integrated FPGA fabric for prototyping. This method allows time and bit accurate evaluation of all system aspects like memory and interconnect interference, cache behavior, etc. that are hard to fully cover using traditional system level simulation approaches. On this basis, Section IV will evaluate design decisions on system level.

Our design is closely integrated with the processing and memory subsystem of the SoC. We will first introduce the system architecture, because it directly influences the design decisions presented later on. Afterwards, hardware and software components are presented.

A. System

Our implementation is based on Xilinx Zynq 7000 SoC, which consists of a processing system (PS) and programmable logic (PL). This allows us to implement an AVB Ethernet controller tightly integrated into the SoC. We use a ZedBoard as development kit and a Linux (kernel version 3.14) based operating system.

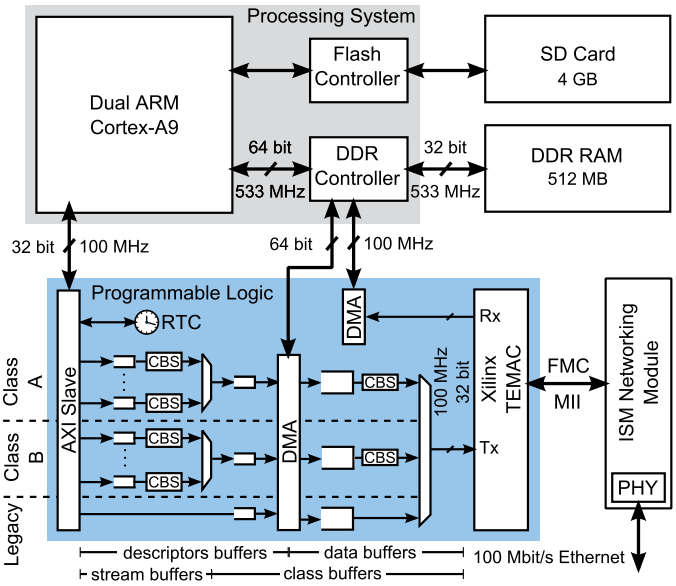

Fig. 1. System overview with detailed presentation of the AVB Ethernet controller implementation

Fig. 1 shows a simplified representation of the system. Zynq has a variety of peripheral modules not depicted here, which are not relevant to the design at stake. Core of the PS is a dual ARM Cortex-A9 running at 533 MHz. Controllers for Flash and DDR RAM integrated in the PS connect off-chip

resources including an SD card storing the boot image and the main memory.

The PL incorporates an FPGA fabric equivalent to a Xilinx Artix-7 FPGA with 85K logic cells. The PS can communicate with the PL using a AXI bus. Additionally the PL can perform direct memory access (DMA) operations through up to four 64 bit AXI buses. They are connected to two distinct physical ports at the DDR controller. We use at most two of these buses to perform DMA operations.

The ZedBoard has an FPGA Mezzanine Card (FMC) connector, which we use to add an external PHY to the system. We use an ISM Networking FMC module that features a 10/100 Ethernet PHY.

B. AVB Ethernet Controller

Subsets of the AVB Ethernet standard like flow control and traffic shaping are highly time-critical and therefore best-suited for hardware implementations. Especially traffic shaping, which has to be carried out accurately and in parallel for multiple streams would not be suitable in software. On the other hand, it is possible to implement time-synchronization in software, but with reduced precision compared to hardware realizations.

The lower half of Fig. 1 shows a schematic of the AVB Ethernet controller. It can be configured and controlled through memory mapped I/O writes towards the AXI slave interface. To minimize the software overhead, all data transactions are performed through DMA. The Xilinx Tri-Mode Media Access Controller (TEMAC) was used. It interfaces the PHY using a Media Independent Interface (MII). For presentation reasons, several configuration and interrupt lines are not shown within the figure.

As shown in [11], synchronization precision within microsecond range can be achieved with a software-based time-synchronization. Because sources of error like asymmetric networking delays have to be added when determining the overall performance, precision much lower than microseconds within our endpoint gives limited benefits. Thus, we choose to implement major parts of the time-synchronization in software. We only provide a real-time clock (RTC) in hardware, which will be used by the software. When used as a master clock, it generates an interrupt in a configurable interval between 125 ms and 1 s (as required by the specification), which initiates synchronization within the driver. In slaves, the RTC is adjusted after each synchronization.

Outgoing traffic has to be policed according to the Credit Based Shaper (CBS) defined in the AVB protocol. CBS maintains a credit score for each traffic class (and talker stream in endpoints), which is reduced during transmission and recovers proportional to its reserved bandwidth. Transmission are only allowed for positive credit values. Therefore, CBS has to update its decision whether a stream or class is eligible for transmission after every transmitted octet, which corresponds to 80 ns for 100 Mbit/s. Because software scheduling granularities are usually multiple orders of magnitude larger (e.g. Linux' minimum scheduling granularity is 75 ms), CBS requires a hardware implementation.

Our implementation of the stream and class buffers along with the CBS modules is illustrated in Fig. 1. The number of stream supported by the AVB Ethernet controller can be configured at design-time. During run-time, these streams can be assigned to one of two traffic classes A & B. According to the standard, each stream is subjected to the CBS. Traffic from each stream is then forwarded to a class buffer, which is then also policed with CBS.

To minimize memory resources within the hardware, messages are buffered in the form of DMA descriptors as long as possible. Descriptors contain the information necessary to fetch the actual packet from the RAM (packet length and memory address). For each class, there is one egress buffer capable of holding a maximum sized data frame. During packet transmission, a DMA operation is initiated just in time to avoid underruns.

Legacy traffic is handled separately and is not subject to any traffic shaping. AVB traffic has priority over legacy traffic, if a transmission is allowed according to the shaping. However, a valid configuration has to ensure at least 25 % of available bandwidth is not allocated to AVB streams.

Within the receive side, no traffic shaping is required. Incoming data is transferred to the RAM via DMA and the processing system is notified through an interrupt. To avoid interference, the Tx and Rx DMA engines are connected to separate buses, which are themselves connected to different ports on the DDR controller. During the evaluation, we will compare this approach to one using only a single bus.

C. Linux Driver

The AVB Ethernet controller is operated by a character device driver implemented in Linux' kernel space. Its tasks include descriptor and data buffer management, interrupt handling and time-synchronization.

A data transmission via AVB starts, when a user application calls the kernel module, passing stream ID, data pointer, and packet length. The data is copied into kernel space and a DMA descriptor is written to the hardware. A separate ring buffer for descriptors is maintained for each stream. After the hardware has copied the data, the processing system is interrupted and deallocates the memory.

The driver is notified when the AVB Ethernet controller has copied a received frame into the memory using an interrupt. Sorting towards streams is not implemented in hardware and has to be done within the driver. Data pointers are sorted towards stream based buffers and therefore, no data copy operations are needed.

We implemented a master and a slave version of 802.1AS time-synchronization, which both use the RTC implemented within the AVB Ethernet controller. The synchronization process is depicted in Fig. 2.

The master endpoint is triggered periodically by an interrupt generated from the RTC. It then sends a *Sync* message containing a current timestamp $T1$. Upon reception, the slave

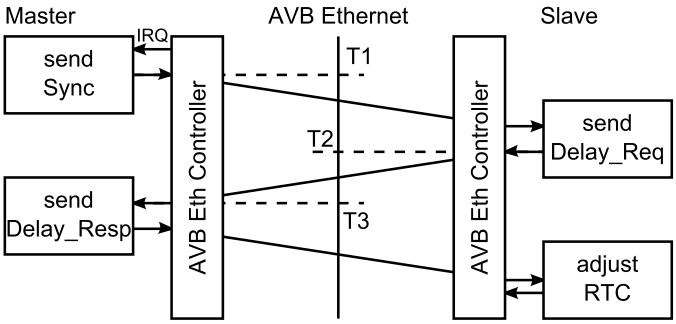

Fig. 2. Time synchronization between two endpoints using 802.1AS

takes one timestamp $T1'$ and another one $T2$ when sending the *Delay_Req* message. A final timestamp $T2'$ is taken by the master and the series of timestamps is sent to the slave using the *Delay_Resp* message. The slave then adjusts the current RTC value offset according to

$$offset = \frac{(T1' - T1) - (T2 - T2')}{2}.$$

This scheme leads to perfect synchronization, if communication delays are equal in master-slave and slave-master direction. Any asymmetry will lead to a synchronization error. While a software implementation adds variance due to interrupt latencies, cache behavior etc., we argue that these are small compared to ones originating from variable networking latencies. In a simple example, we assume that transmission of the *Sync* message is delayed by a single maximum sized Ethernet frame (1530 octets plus 12 octets interpacket gap), while the *Delay_Req* is transmitted without interference. This single interference causes a synchronization error of 61 µs. Of course, in a worst-case multiple such interferences can occur at multiple hops. We will quantify the synchronization errors contributed from our software implementation in the experiments below.

IV. Experiments & Results

We conducted a series of experiments targeted at central performance and cost metrics of the overall system. In order to get isolated measurements of endpoint specific qualities, we try to minimize effects contributed from the AVB network itself. Therefore, we use two endpoints directly interconnected without any switches or other endpoints interfering. Following, we present latencies and synchronization errors induced by the system. Additionally, we discuss overheads in hardware utilization.

A. Endpoint-Induced Latencies

Low end-to-end latencies are the central feature of AVB Ethernet. Real-time analyses neglect latencies within endpoints and only focus on networking delays. However, as we are dealing with an implementation in a real system, we are working with finite processing and data transfer capabilities. Before AVB packets can be sent on the network, they have to be copied and transferred across the system. With the following experiments, we try to assess the latency contribution attributed to the implementation of the endpoint itself. We conduct different experiments to determine latencies within hard- and software.

Experiment 1: To measure hardware latencies, we extended the AVB Ethernet controller with an additional timer (resolution: 10 ns). This timer is started when the driver notifies the hardware of an upcoming transmission. Specifically, it is started in the same clock cycle in which the final register write to signal the transmission occurs at the AXI slave interface. The descriptor proceeds through the stream shaping and triggers a DMA transfer. After completion of the DMA transfer, the packet has to be forwarded to the MAC. The timer stops, when the last data word is written to the MAC. The timer is read 500 µs after the packet is issued, because then, even maximum size packets are already transferred. The experimental setup is depicted in Fig. 3.

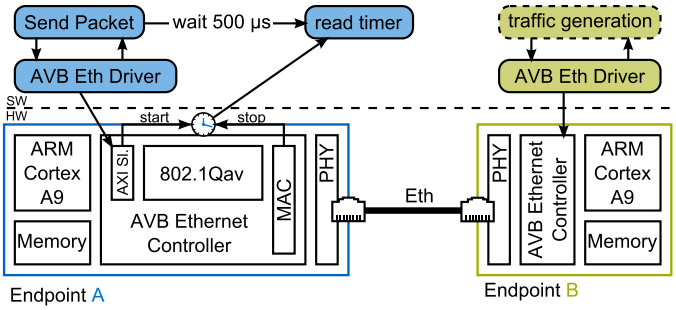

Fig. 3. Measurement setup to determine latencies within the AVB Ethernet controller (Experiment 1)

We conducted the experiment for various packet sizes, separate and shared memory interfaces, and with optional receive side traffic. We repeat the experiment 1000 times for each setup. We assume DMA packet fetch and the copy operation towards the MAC to be most time consuming, because these move the actual packet data and not just descriptors. The duration of these operations scales with packet size. Additionally, the DMA packet fetch is prone to interference, as it uses shared resources (on-chip interconnect, memory). We quantify the effect of this interference by injecting 8000 packets/s with 1500 B (96 Mbit/s) using a second endpoint B. Using maximum sized packets maximizes the duration of blocking from Rx DMA operations. We evaluate the interference for shared and separate memory interfaces among Tx and Rx.

Experiment 2: The second experiment is targeted at determining software delays. Here, we can use a counter within the processing system that is sourced with 200 MHz, thus providing 5 ns resolution. It is based on the same setup as in experiment 1, with two endpoints A and B interconnected.

The measurement flow is illustrated in Fig. 4. After initializing the data to be transmitted, timestamp t_1 is taken. In actual applications, we assume the data (e.g. video frames) to be already within the memory and therefore do not include it in the sending latency. After initialization of the packet header,

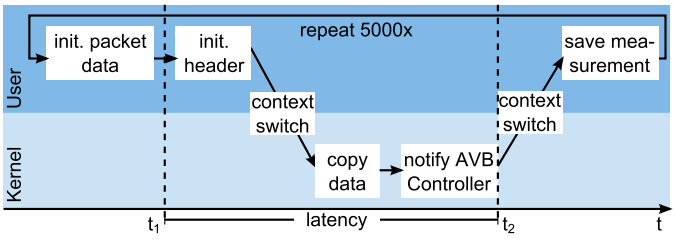

Fig. 4. Measurement flow to determine software latencies (Experiment 2)

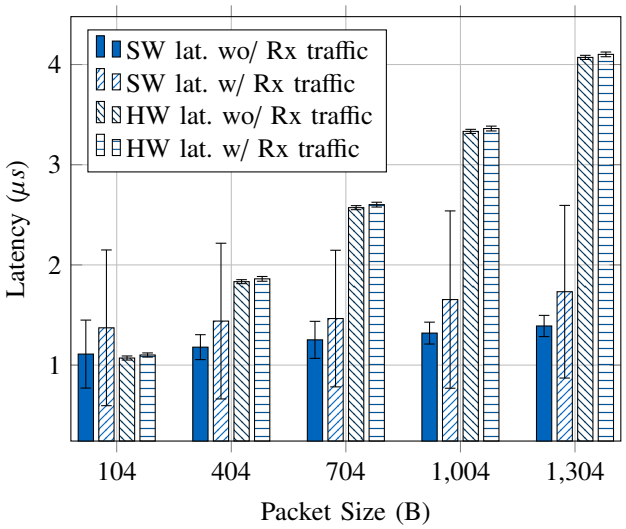

Fig. 5. SW and HW transmit latencies observed in experiment 1 and 2

the kernel module is invoked. Following a context switch to kernel space, the data is copied into the transmit buffer and the AVB Ethernet controller is notified about the enqueued packet. Once this is completed, the AVB Ethernet controller starts taking over and we take another timestamp t_2. The recorded latency equals the difference $t_2 - t_1$.

Again, we vary the packet size and use the second endpoint B to generate receive side traffic. To maximize the impact of the receive traffic, we use minimum size packets. This maximizes the number of interrupts and therefore the number of context switches. For each experiment, we make 5000 measurements of the latency.

Fig. 5 presents latencies measured within HW (Experiment 1) and SW (Experiment 2) components of the system. We observe that HW and SW latencies are similar for short packets, but HW latencies are more strongly dependent on the packet size. The processing system is connected to the DDR controller using a 64 bit AXI3 bus at 533 MHz (c.f. Fig. 1). The FPGA is connected at 100 MHz and has an additional bottleneck, as the port of the MAC is only 32 bit wide. Thus, the processing system achieves a memory throughput 10.6 times that of the FPGA. In the experiments, we measured an average increase in latency of 0.23 ns/B for SW and 2.42 ns/B for HW. Their ratio of 10.52 closely resembles the difference in memory throughput.

Considering additional Rx traffic is important, as it shows the systems behavior when exposed to interference. The presence of Rx traffic means additional memory and on-chip interconnect accesses, as well as Rx processing in SW and interrupts. SW latencies are more sensitive to interference. While HW latencies remain nearly constant, average SW latencies increase by up to 30% when exposed to receive side traffic. For HW latencies, we evaluated the increase in maximum latencies for different architectures connecting Tx and Rx interfaces to the memory through shared (6.57% increase) or separate (2.05% increase) buses. The use of separate buses increases performance isolation between Tx and Rx operations. Such isolation is important in embedded systems, because it allows better prediction of the system behavior.

Latencies were observed in the microsecond range, which is multiple orders smaller than typical end-to-end latency requirements (usually no less than 2.5 ms). However, software-based latencies are subject to large variations. This issue cannot be solved without employing a real-time operating system or a real-time patched Linux kernel.

B. Synchronization Errors

The synchronization algorithm is implemented in software. Several run-time properties of the system like cache behavior or interrupt latencies are hard to predict, and thus cause errors in the synchronization process.

Experiment 3: In this experiment, we quantify the error caused by the implementation of the synchronization protocol within the endpoint. We want to determine this error isolated from other sources like network interference or temperature variations. While we can eliminate network interference by running the synchronization without background traffic, temperature cannot be completely kept constant.

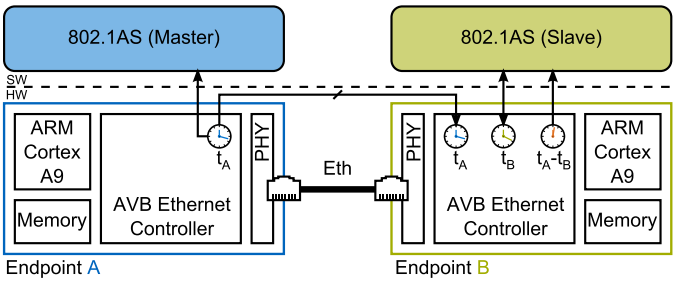

Fig. 6. Measurement setup to determine synchronization errors (Experiment 3)

Fig. 6 describes the measurement setup. Endpoint A is serving as a master while endpoint B must synchronize its time t_B to t_A. For precise comparison between the timers, we use a parallel connection to connect endpoint A's timer to endpoint B. In hardware, we accurately compute the synchronization error $t_A - t_B$. We sample the synchronization error in endpoint B before and after the protocol adjusts the counter. In an ideal system, this error should be zero after every synchronization. We therefore consider the error sampled just after synchronization a measure for the inaccuracies introduced by our system. Additionally, by considering the difference between

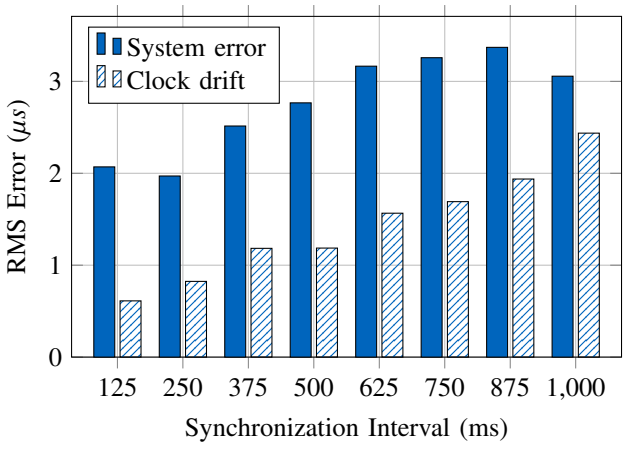

Fig. 7. Synchronization error from system non-idealities as well as clock drift in experiment 3

the error before the current synchronization and after the last synchronization, we assess the error contributed from clock drift.

We conducted the experiments for synchronization intervals of $\{125, 250, ..., 1000\}ms$. For each interval, we measured the synchronization error 1000 times. In Fig. 7, we present the root mean square (RMS) synchronization errors contributed from clock drift and from our system.

The results show a trend of linear increasing clock drift for increasing synchronization intervals, reaching from 0.61 µs for 125 ms up to 2.44 µs for 1 s. Deviations from this trend are due to variations in oscillator frequency, which can occur due to temperature variations between the measurements.

The system error represents inaccuracies caused by non-idealities within the system, which include variable interrupt latencies, memory access times, cache behavior etc. The RMS error ranges between 1.98 µs and 3.37 µs. There is a slight tendency of larger errors for larger synchronization intervals, which we attribute to additional cache pollution.

We obtained increased errors compared to the simulation results from [11], which indicate timestamping errors between 409 ns and 1.78 µs depending on the synchronization interval and CPU type. We attribute this to the fact that we were able to cover a wider range of possible error sources in our experiments, especially cache behavior.

The maximum system error recorded was 8 µs. Together with errors due to background traffic, which were not part of this evaluation, and clock drift, errors smaller 10 µs can be expected in typical scenarios. In worst-case scenarios, the synchronization error induced from the asymmetric networking latencies can be two orders of magnitude larger. Generally, we assume a precision of 10 µs sufficient for automotive application like synchronous sensor sampling. Sensors are usually connected through field buses like Local Interconnect Network, which operate at up to 20 kHz (equivalent to a bit time of 50 µs). Therefore, no benefit would be gained from increased synchronization precision, achievable e.g. with hardware assistance.

C. Hardware Overhead

We implemented the AVB Ethernet controller as an extension to Xilinx' standard Ethernet design. The resource utilization of the simple design including MAC, DMA engine and AXI interface will be used as baseline. Using Vivado 2013.4, we measured the resource consumption for logic as look up tables (LUTs) and flip flops (FFs). The results for a design supporting four streams are shown in Fig. 8.

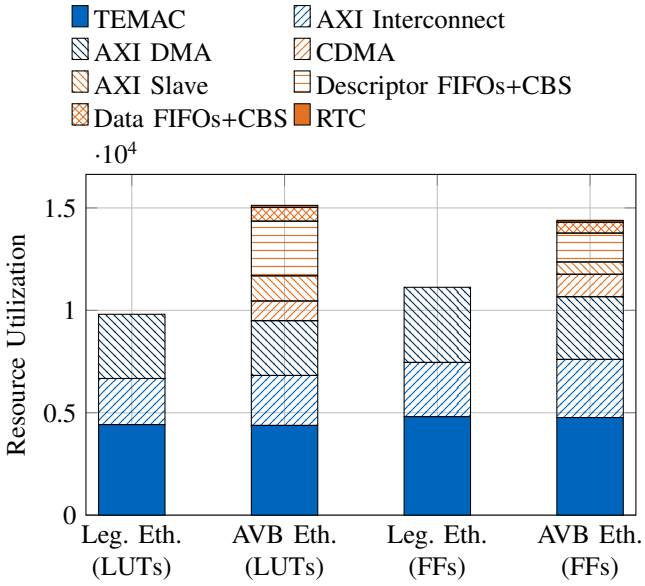

Fig. 8. Hardware resources used by the AVB Ethernet controller compared to a legacy controller

The overall resource overhead of the AVB Ethernet controller compared to the legacy design is increased by 53% in LUTs and 28% in FFs. Most resources are consumed by Descriptor FIFOs and the associated traffic shaping. Together with the data FIFOs, this represents the traffic shaping extensions. The real-time clock (RTC) for time-synchronization uses less than 1% of resources within the AVB Ethernet controller. The AXI slave is used as a register interface to communicate with the Linux driver.

In our design, we used different DMA engines in Tx and Rx path. The TEMAC uses AXI stream interfaces, and therefore the AXI DMA module can be used directly for Rx operations. For Tx operations, the DMA engine is not directly connected to the TEMAC, but has to perform DMA transactions to multiple class buffers. Because AXI stream does not support addressing, we could not use the existing AXI DMA but used a Central DMA (CDMA) instead. The increased resource utilization by DMA engines constitutes 4.8% of the total overhead.

The increase in resources of around 40% is significant, but also adds important functionalities like the credit based traffic shaping, which cannot be implemented in software. Commercial Ethernet controllers are usually more complex

than the baseline implementation used here and incorporate protocol specific accelerators like TCP offload engines. Such extensions could also be used in combination with AVB enablements. The baseline is therefore pessimistic.

V. Conclusion

In this paper, we presented an integrated solution of an ARM SoC with an AVB Ethernet controller. We proposed a HW/SW partitioning and a composable hardware architecture, that extends a conventional MAC to enable AVB capability at the cost of 53% additional logic and 28% additional flip flops.

Currently, little commercial hardware is available and experimental data is lacking as well. Our work is the first to provide experimental data on the temporal behavior of AVB endpoints. Based on a prototypical implementation using an integrated system composed of a dual ARM Cortex-A9 and an FPGA, we conducted a series of experiments, evaluating key performance aspects of the system like packet release latencies and synchronization errors.

Hardware latencies within our design occur in the range from 1 µs to 4 µs depending on the packet size with deviations of less than 25 ns even when exposed to interfering receive packets. While latency within the software components are smaller with less than 1.5 µs, they are subject to standard deviations of up to 1 µs. Based on our measurements, packet release latencies smaller 6 µs can be expected. This is a similar order of magnitude as the transmission time of a minimum sized Ethernet frame at 100 Mbit/s (5.12 µs), yet it is orders of magnitude smaller than automotive end-to-end latency requirements (≥2.5 ms).

In another experiment, we were able to show that the synchronization error caused by the non-idealities of the real system is smaller than 8 µs. The achieved accuracy is sufficient for automotive applications like synchronous sensor sampling. However, if future applications require increased precision, the synchronization protocol could be offloaded into the AVB Ethernet controller.

The proposed architecture and the measured performance are closely connected. While we found the performance sufficient for applications expected in the automotive domain, in other or future use-cases increased performance and therefore additional hardware support might be necessary. Nevertheless, the architecture and results presented can serve as baseline for such improvements. Additionally, our data can be used as input in the planning and design process of distributed applications and embedded architectures.

AVB Ethernet is on the edge of adoption in automotive onboard networks. Additionally, AVB's successor called Time-Sensitive Networking (TSN) is already in development. Building upon the AVB standard, it adds additional features aimed enabling very low latency transmissions. Due to the modularity of the extensions, our AVB controller can be used as a basis to implement extensions proposed in TSN.

Acknowledgments

This work was funded within the project ARAMiS by the German Federal Ministry for Education and Research with the funding IDs 01|S11035. The responsibility for the content remains with the authors.

References

[1] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-vehicle networks: A review," *Intelligent Transportation Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–12, 2014.

[2] L. L. Bello, "Novel trends in automotive networks: A perspective on ethernet and the ieee audio video bridging," in *Emerging Technologies Factory Automation (ETFA), 2014 IEEE 19th Conference on*, Sept 2014, pp. 1–9.

[3] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Synchronize your watches: Part i: General-purpose solutions for distributed real-time control," *Industrial Electronics Magazine, IEEE*, vol. 7, no. 1, pp. 18–29, 2013.

[4] R. Kreifeldt, A. Chang, A. J. Huotari, Y. Kim, K. Lewis, and K. B. Stanton, "Avb for automotive use," *AVnu Alliance White paper*, vol. 20, 2009.

[5] J. Diemer, J. Rox, and R. Ernst, "Modeling of ethernet avb networks for worst-case timing analysis," *MATHMOD, Austria*, 2012.

[6] P. Axer, D. Thiele, R. Ernst, and J. Diemer, "Exploiting shaper context to improve performance bounds of ethernet avb networks," in *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*. ACM, 2014, pp. 1–6.

[7] F. Reimann, S. Graf, F. Streit, M. Glas, and J. Teich, "Timing analysis of ethernet avb-based automotive e/e architectures," in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*. IEEE, 2013, pp. 1–8.

[8] R. Queck, "Analysis of ethernet avb for automotive networks using network calculus," in *Vehicular Electronics and Safety (ICVES), 2012 IEEE International Conference on*. IEEE, 2012, pp. 61–67.

[9] D. Azua, J. A. Ruiz *et al.*, "Complete modelling of avb in network calculus framework," in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*. ACM, 2014, p. 55.

[10] U. D. Bordoloi, A. Aminifar, P. Eles, and Z. Peng, "Schedulability analysis of ethernet avb switches," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*. IEEE, 2014, pp. 1–10.

[11] A. Mahmood, R. Exel, and T. Sauter, "Impact of hard-and software timestamping on clock synchronization performance over ieee 802.11," in *Factory Communication Systems (WFCS), 2014 10th IEEE Workshop on*. IEEE, 2014, pp. 1–8.

[12] A. Kern, H. Zinner, T. Streichert, J. Nöbauer, and J. Teich, "Accuracy of ethernet avb time synchronization under varying temperature conditions for automotive networks," in *Proceedings of the 48th Design Automation Conference*. ACM, 2011, pp. 597–602.

[13] H.-T. Lim, D. Herrscher, L. Volker, and M. J. Waltl, "Ieee 802.1 as time synchronization in a switched ethernet based in-car network," in *Vehicular Networking Conference (VNC), 2011 IEEE*. IEEE, 2011, pp. 147–154.