# A Time-predictable TTEthernet Node

Maja Lund, Luca Pezzarossa, Jens Sparsø, and Martin Schoeberl
Department of Applied Mathematics and Computer Science
Technical University of Denmark
Email: maja_lala@hotmail.com, {lpez, jspa, masca}@dtu.dk

***Abstract*—Distributed real-time systems need time-predictable computation and communication to facilitate static analysis of timing requirements and deadlines. This paper presents the implementation of a deterministic network protocol, TTEthernet, on the time-predictable Patmos processor. The implementation uses the existing Ethernet controller on the processor and we tested it with a TTEthernet system provided by TTTech Inc. Further testing showed that the controller could send time-triggered messages with bounded latency and a small jitter of approximately 4.5 us. We also provide worst-case execution time analysis of the network code, which demonstrates a time-predictable end-to-end solution. This work enables Patmos to communicate with other nodes in a deterministic way. Thus, extending the possible uses of Patmos.**

## I. Introduction

Distributed real-time systems consist of a set of independent real-time computing systems communicating through some kind of networking fabric. Typical applications can be found in industrial process control, automotive, and aerospace. This type of systems needs time-predictable communication and computation to be able to statically analyze that all end-to-end deadlines are met. Deterministic networks are typically used to provide guaranteed communication and are an integral part of many distributed real-time systems.

A widely used solution is the CAN bus. This is a serial bus able to send small prioritized messages with bounded latency [3]. TTCAN is a time-triggered extension of CAN, restricting nodes to only transmit messages in certain time-slots, thus increasing the determinism of the bus [12]. FlexRay was designed to replace the CAN and TTCAN bus. It operates with a predefined communication cycle consisting of a static segment for deterministic traffic and a dynamic segment for other traffic [19]. CAN and FlexRay offer a maximum bandwidth of 1 and 10 Mbit/s respectively, and are characterized by limited cable length.

As the demand for higher bandwidth and cable length increases, the industry has started looking towards Ethernet-based real-time protocols, such as EtherCAT, Ethernet Powerlink, and TTEthernet [5]. A new solution still in the process of standardization is TSN, which allows for time-sensitive data transmission over Ethernet networks [9]. In TSN, determinism is guaranteed by transmitting high priority messages in predefined time-slots, yet maintaining backward compatibility with standard Ethernet solutions.

In this paper, we target the TTEthernet standard [10]. TTEthernet is an extension of Ethernet that guarantees an upper bound on the end-to-end latency and a small and bounded jitter.

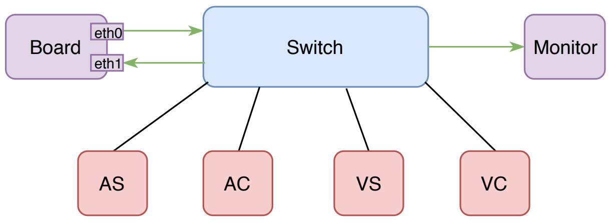

Fig. 1. The test system consisting of our TTE node (Board) and equipment provided by TTTech Inc. Our TTE-node uses two Ethernet controllers; one for sending and one for receiving.

It operates at the data link layer of the OSI model and allows communication between a set of nodes through one or more proprietary switches. Latency and jitter through a TTEthernet switch have been measured in [2], where results of 10 $\mu$s for frames smaller than 128 bytes and 30 $\mu$s for larger frames are presented.

A software-based TTEthernet end-system has been developed for AUTOSAR, which is a standardized software architecture for control units in cars [7]. For this solution, the precise end-to-end latency of the system is unclear due to a non-deterministic receive function and jitter of 32 $\mu$s was observed. Compared to the AUTOSAR solution, we are able to limit the observed jitter to 3.5 $\mu$s

TTEthernet has its own clock synchronization protocol. However, it is also possible to use the IEEE 1588 clock synchronization standard on top of TTEthernet [1].

This paper presents a time-predictable TTEthernet node based on the Patmos time-predictable processor [17] and an open-source Ethernet controller [13], [14]. Thus, this work enables the Patmos processor to communicate through the TTEthernet deterministic network protocol. This is achieved by extending the open-source Ethernet controller in software to allow for TTEthernet compatibility. The software is time-predictable and its worst-case execution time (WCET) can be statically analyzed. In contrast to the existing solutions presented above, our solution is fully time-predictable and does not require expensive proprietary hardware (i.e., custom Ethernet controllers).

The developed solution is evaluated in a use-case setup, shown in Figure 1. Our Patmos based TTE node is implemented on an FPGA board and connected to a test setup provided by TTTech Inc. consisting of a TTethernet Chronos switch, four Linux PCs (AS, AC, VS, and VC) with network cards from TTTech, and a Windows PC for monitoring. Our TTE node interacts with the system as a synchronization client. For this setup, we proved correct functionally, measured latency and jitter, and performed WCET analysis of all network components and software functions.

In summary, the main contribution of this work is a fully time-predictable TTEthernet node based on an open-source Ethernet controller and supported by a WCET-analyzable software stack. To the best of our knowledge, this is the first WCET analyzable TTEthernet node that combines time-predictable communication over Ethernet with time-predictable execution of tasks. The results show that time-triggered messages can be sent with bounded latency and a small jitter of approximately 4.5 $\mu$s.

This paper is organized in 5 sections: Section II provides background on TTEthernet. Section III describes the design and implementation of our time-predictable TTEthernet node. Section IV evaluates the design with measurements and static WCET analysis. Section V concludes the paper.

## II. TTEthernet

TTEthernet is a deterministic extension of Ethernet, which operates at the data link layer of the OSI model [10]. A TTEthernet system generally consists of a set of nodes communicating through one or more switches. It allows regular Ethernet traffic (best-effort traffic) and introduces two classes of critical traffic: (1) rate constraint traffic and (2) time-triggered (TT) traffic. As TT traffic uses reserved time slots, it has the highest priority implicitly. As we are interested in hard-real systems, we only consider TT traffic in the rest of the paper.

TTEthernet uses the concept of a virtual link (VL). A VL is a relation between a sender and one or more receivers and is identified by a unique ID. Switches need to know the VL definitions, and nodes need to know the definitions of the VL they can send on. For TT traffic, a VL definition includes a receive window in which the switch will accept messages from the sender, and send windows where the switch will pass them on to the receivers. Packages received outside of the defined window will be ignored by the switch (and by a receiver). The switch plays a central role in a TTEthernet system to protect TTEthernet nodes from other nodes that might violate the TT schedule.

All nodes and switches need a global notion of time to send packages at the correct points in time. Periodic clock synchronization ensures a common notion of time between the nodes. The time between each synchronization is called an integration cycle and has a period which is typically in the range of 1 to 10 milliseconds. The schedule for TT traffic is periodic as well and repeats every cluster cycle, which is defined as an integer multiple of integration cycles. An example of some TT traffic in a system with an integration period of 10 ms and two integration cycles per cluster cycle can be seen in Figure 2. TT traffic is often defined by its period and the offset from the start of the cluster cycle. For example, TTE3 in the figure has a period of 20 ms and an offset of 10ms.

Every node in a distributed system has its local clock generated by an oscillator. These clocks are not perfect and might have small variations in frequency. Even if all clocks in a distributed system have been perfectly synchronized they would eventually drift apart [4]. The difference in value between two clocks at a certain point in time is called clock skew.

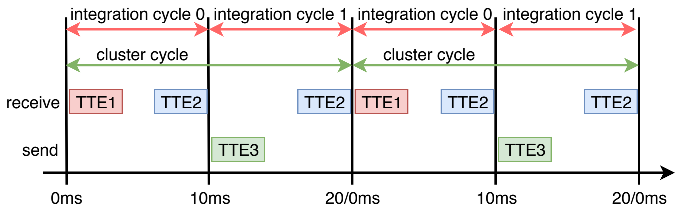

Fig. 2. Example of integration and cluster cycles

Clock synchronization is achieved through the exchange of protocol control frames (PCF). Three roles are involved in the exchange of PCFs: synchronization masters, synchronization clients, and compression masters. Typically, the switches act as compression masters, and nodes are either synchronization masters or clients.

Each node in the synchronization domain keeps track of when they believe the integration cycle has started, which is when synchronization masters send out integration frames. An acceptance window around the expected point in time when the PCF shall arrive defines whether or not the node should accept the PCF as correct. The acceptance window has a width of twice the expected precision of the system, defined as the maximum difference between two correct local clocks. If the PCF is accepted, the difference between the expected and actual time is used to correct the clock.

## III. Design and Implementation

In the following, we present the design and implementation of our TTE node. At first, we describe the platform. Then, we explain the functionality of the developed software stack.

### A. Platform

The proposed TTEthernet node is based on the Patmos time-predictable processor [17] used in the T-CREST platform [6], [16] and on an open-source Ethernet controller. The controller is based on the EthMac block from OpenCore [13], which was previously ported for Patmos [14].

The communication between the controller and the processor is buffer-based using buffer descriptors. These are stored in the controller itself and contain addresses and status of associated RX/TX buffers. The controller is able to receive a frame when there is at least one available receive buffer. Otherwise, the frame is discarded. This implies that in best-effort traffic, frames can be lost. After receiving a frame, the receive status is written to the associated buffer descriptor, and the controller generates an interrupt (if enabled). The buffer will stay unavailable until program marks the buffer descriptor empty again. The controller transmits frames from the buffer as soon as the program marks the associated buffer descriptor as ready.

We implemented two different versions of the proposed solution. One where received messages are discovered through polling, and another where the Ethernet controller triggers an interrupt whenever a frame is received. Using interrupts for time stamping of an arriving Ethernet message is not the best solution since the start of the execution of the interrupt routine will have some jitter (e.g., due to cache hits and misses). Therefore, the produced timestamp is contaminated by the jitter. The polling solution solves this problem by using a periodic task that is scheduled to be released just before the next synchronization message arrives. This includes enough time to contain the worst-case preemption delay and the possible jitter of the PCF itself. In this case, the processor is ready to listen to the Ethernet port in a tight loop in to get a better timestamp in software. Therefore, in the polling solution, the actual polling runs only for a short time, without wasting processor time. As future work, we plan to change the Ethernet controller to include hardware support for time-stamping [11].

For implementation and testing, we used the star network configuration shown in Figure 1. We implemented the presented TTE node on an Altera DE2-115 FPGA board. AS, AC, and VC act as synchronization masters and VS and our TTE node are synchronization clients. The monitor is used to monitor all VL with Wireshark.

We used the TTE tools from TTTech for configuring the TTEthernet system [18]. This consist of the generation of the schedule for the entire network based on the provided information about senders, receivers, physical links, virtual links, synchronization domain, and other network information.

## B. Functionality

To limit complexity, our node only acts as a synchronization client and only connects to a single switch. The developed software stack offers three main functionalities: initialization, receiving, and sending, which are described in the following.

Figure 3 shows the intended flow of programs using the developed system. At first, the program initializes the controller with static information regarding the system, information on VLs, and the schedule. After initialization, the processor continuously polls the Ethernet controller to see if any new messages have arrived. It is necessary to ensure that the receive time of integration frames is recorded as precisely as possible to enable correct clock synchronization. The received message is then passed through the `tte_receive` function, which will synchronize the local clock in case of an integration frame, or otherwise return a value indicating the message type. The rest of the body depends on the purpose of the program itself. Outgoing TT messages can be scheduled anywhere in the program body and will be sent according to the system schedule through timer interrupts.

*1) Initialization:* The system needs initialization before the program loop can start. The initialization consists of initializing the integration cycle, the cluster cycle, how many VLs the node can send on, maximum transmission delay, the compression delay of the switch, and the system precision. Furthermore, the permitted send time for each VL needs to be known, so each VL gets initialized with an ID, an offset, and a period. The TTEthernet switch will ensure that only relevant and correctly timed TT messages are passed on to the network.

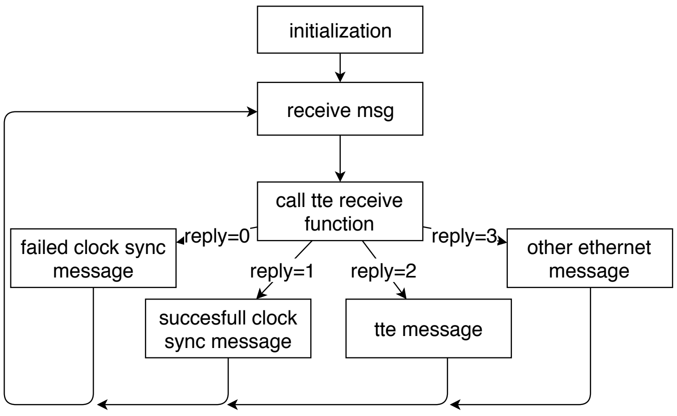

Fig. 3. The intended flow of user programs. The controller is first initialized with all constants of the system. Remaining functionality happens in a loop where the program continually waits until a frame is received, calls the `tte_receive` function, and then reacts to the reply

Our TTE node receives messages like standard Ethernet ones. Initialization also includes the generation of the send schedule and the registration of the timer interrupt function. The timer is started when the first PCF is received.

*2) Receiving and Clock Synchronization:* Message reception is performed by continuously polling the interrupt source register until the bit signifying that a frame has been received is set. This is done by the function `tte_wait_for_message`, which is also responsible for recording the receive time by reading the current clock cycle count. After a message has been received and the receive time is stored, the next buffer should be marked as empty, and the interrupt source register cleared. This can be done by calling `tte_clear_free_rx_buffer`. Afterward, the `tte_receive` function is called with the address of the received frame and the recorded receive time. The `tte_receive` function initially checks the type of the message. If it is a PCF type, the integration frame is used to synchronize the local clock to the master clock. Thus, the local clock is immediately adjusted to the new value. If the received frame is not a PCF type, the function returns and the received frame can be used according to the program.

*3) Sending:* Time-triggered messages must be sent according to a predefined schedule. Thus, the sending requires some queuing mechanism, as the program should not be expected to calculate the exact send times itself. A send FIFO queue is created for each VL during initialization and is allowed to hold the maximum amount of frames that the VL can send in one cluster cycle, calculated as $\frac{clustercycle}{VLperiod}$. Therefore, the maximum delay of a scheduled frame is one cluster cycle. Frames are scheduled through a specific function, `tte_schedule_send`, which takes the address and size of the frame and which VL it should be scheduled for. The function then checks the queue of the VL and returns zero if it is full, and one if the frame was successfully scheduled. It is up to the program to ensure that it does overwrite the buffer before the frame has been sent.

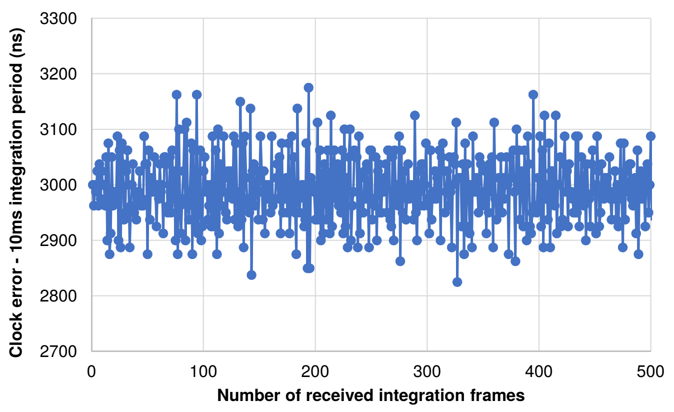

Fig. 4. Clock error, regular implementation, no additional sending or receiving

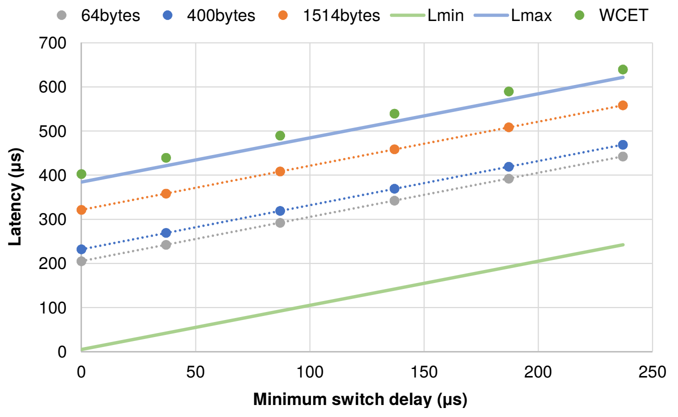

Fig. 5. Latency for various frame sizes as a function of minimum switch delay. The correlation is strong and well within expected minimum and maximum values

## IV. Evaluation

The evaluation of the proposed solution uses the setup illustrated in Figure 1 and already described in the previous sections. For this setup, we evaluate clock synchronization, latency, and jitter. In addition, we performed the WCET analysis of the developed software functions. We tested the network at different integration periods and cluster cycles, although mostly with an integration period of 10 ms and a cluster cycle of 20 ms.

### A. Clock Synchronization

The clock error was measured as the difference between the scheduled receive point in time and the actual point in time at each received integration frame. Figure 4 shows 500 of the 2000 captured data points obtained using an integration period of 10 ms and no additional sending or receiving of frames other than the integration frames. The clock error ranges between 2787 ns and 3225 ns, and averages at 2990 ns. In terms of clock drift, this corresponds to approximately 300 ppm. Typical values are between 10 and 100 ppm, so the error is higher than expected.

### B. Latency and Jitter

With reference to Figure 1, our node uses two Ethernet ports and sets up a schedule with a VL from one port to the other. Both ports are considered to be independent devices by the schedule and are both synchronization clients. A single VL with a period of 10 ms and an offset of 8.2 ms was used. This simplifies the test program since a single message can be sent and received each integration cycle.

The test program follows the form described in Figure 3 with the first controller receiving messages in the overall loop. After a successful synchronization message, a TT message is scheduled and the program waits until the message is received by the second controller before proceeding. This allows to collect both the schedule and receive points of messages as the current clock cycle count. Both receive and send window in the switch are 263 $\mu$s wide.

Figure 5 shows the latency measured as the average difference in send and receive time over 2000 measurements for 3 different frame sizes. The figure also includes the expected minimum and maximum latencies $L_{min}$ and $L_{max}$. It is possible to observe that all measured values are within these values.

The expected transmission times for frames of the measured sizes are 5.12 $\mu$s, 32 $\mu$s, and 121.12 $\mu$s respectively. Judging by the trend-lines, the actual switch delay for these experiments must be approximately 200 $\mu$s higher than the minimum. This indicates that the switch receives the messages approximately 63 $\mu$s into the receive window in these tests. The jitter, measured as the smallest latency subtracted from the highest, stayed between 4.5 $\mu$s and 4.6 $\mu$s throughout all experiments.

### C. Worst-Case Execution Time

WCET analysis has been performed on significant functions of the software stack using the platin WCET tool [8]. The results are presented in Table I, where all functions are part of the implemented library, except for the final two, which are part of the tested demo program. To analyze the timer interrupt function, this had to be explicitly called.

The functions `tte_clear_free_rx_buffer` and `tte_receive` are used for reception. `tte_receive_log` is the receive function with logging enabled. `handle_integration_frame` and `handle_integration_frame_log` are called by the `tte_receive` function if the message is an integration frame. `tte_prepare_test_data` creates a TT message where the data part repeats a specified byte until the message has a certain length. `tte_schedule_send` is the sending function. `tte_clock_tick` and `tte_clock_tick_log` are the timer interrupt functions with and without logging and call `tte_send_data` when actually sending a message. `tte_code_int` is executed after each successfully received integration frame, and `tte_code_tt` is executed after each received TT message.

The example schedule executed by our TTE node has a maximum of 3 incoming TT messages in a single integration cycle. One of the VLs can send a maximum of 3 outgoing TT messages, and the other a maximum of 5. An integration period is 10 ms, which is equivalent to 800,000 clock cycles.

TABLE I
WORST CASE EXECUTION TIME IN CLOCK CYCLES OF DIFFERENT FUNCTIONS.

| Function | WCET (in clock cycles) |
|---|---|
| tte_clear_free_rx_buffer | 10 |
| tte_receive | 3028 |
| tte_receive_log | 3216 |
| handle_integration_frame | 2573 |
| handle_integration_frame_log | 2732 |
| tte_prepare_test_data | 39138 |
| tte_schedule_send | 244 |
| tte_send_data | 289 |
| tte_clock_tick | 1641 |
| tte_clock_tick_log | 1824 |
| tte_code_int | 392419 |
| tte_code_tt | 40156 |

Based on: (a) information about the schedule and the integration period, (b) the delays in the network and the switch, and (c) the WCET figures listed in Table I, it is now possible to verify that everything fits within the 800,000 clock cycle integration cycle. The details of the analysis are beyond the scope of this short paper. Here we limit to observing that the result of the analysis is 527,479 clock cycles, which is well below the available 800,000 clock cycles.

Being able to provide WCET bounds for basic Ethernet functions is an important step towards a time-predictable node for a distributed real-time system. We plan to use this TTEthernet solution with our time-predictable TCP/IP stack tpIP [15].

### D. Source Access

The TTEthernet controller and the relevant software are in open source and can be found at https://github.com/t-crest/patmos/tree/master/c/apps/tte-node.

## V. CONCLUSION

This paper presented a time-predictable TTEthernet node. The TTEthernet node is built on top of the time-predictable Patmos processor for which we have written a fully-analyzable software stack. The solution was evaluated in an environment consisting of one TTEthernet switch and six TTEhernet nodes. We found that the end-to-end latency of transmitted time-triggered messages is predictable and well within the expected maximum latency. The measured jitter was small (around 4.5 $\mu$s) and did not vary significantly in the tested scenarios. WCET analysis was carried out for all the main functions of the developed software. Overall, this project provides a solution for deterministic communication with TTEthernet and WCET-analyzable tasks for the Patmos processor. To the best of our knowledge, this is the first TTEthernet node with a WCET-analyzable network stack.

## ACKNOWLEDGEMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 764785.

## REFERENCES

[1] Astrit Ademaj and Hermann Kopetz. Time-triggered ethernet and ieee 1588 clock synchronization. In *2007 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 41–43, Oct 2007.

[2] Florian Bartols, Till Steinbach, Franz Korf, and Thomas C. Schmidt. Performance analysis of time-triggered ether-networks using off-the-shelf-components. In *Proc. IEEE Intl. Symposium on Object/component/service-oriented Real-time Distributed Computing Workshops (ISORCW)*, pages 49–56. IEEE, 2011.

[3] William Buchanan. *Computer Busses*. Butterworth-Heinemann, 2000. Chapter 21: CAN Bus.

[4] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition, 2011.

[5] Peter Danielis, Jan Skodzik, Vlado Altmann, Eike Bjoern Schweissguth, Frank Golatowski, Dirk Timmermann, and Joerg Schacht. Survey on real-time communication via ethernet in industrial automation environments. In *Proc. IEEE Intl. Conference on Emerging Technologies and Factory Automation (EFTA)*. IEEE, 2014.

[6] Martin Schoeberl et al. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61(9):449–471, 2015.

[7] Thomas Fruhwirth, Wilfried Steiner, and Bernhard Stangl. TTEthernet SW-based end system for AUTOSAR. In *Proc. IEEE Intl. Symposium on Industrial Embedded Systems (SIES)*, pages 21–28. IEEE, 2015.

[8] Stefan Hepp, Benedikt Huber, Jens Knoop, Daniel Prokesch, and Peter P. Puschner. The platin tool kit – The T-CREST approach for compiler and WCET integration. In *Proc. Kolloquium Programmiersprachen und Grundlagen der Programmierung (KPS)*, 2015.

[9] Time-Sensitive Networking Task Group . http://www.ieee802.org/1/pages/tsn.html (Accessed 2018-07-18).

[10] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. The time-triggered ethernet (TTE) design. In *Proc. IEEE Intl. Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, pages 22–33, 2005.

[11] Eleftherios Kyriakakis, Jens Sparsø, and Martin Schoeberl. Hardware assisted clock synchronization with the ieee 1588-2008 precision time protocol. In *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, pages 51–60. ACM, 2018.

[12] G Leen and D Heffernan. TTCAN: a new time-triggered controller area network. *Microprocessors and Microsystems*, 26(2):77–94, 2002.

[13] Igor Mohor. Documents: "ethernet IP core specification" and "ethernet IP core design document", 2002. Online: http://opencores.org/project,ethmac (Accessed 2018-02-07).

[14] Luca Pezzarossa, Jakob Kenn Toft, Jesper Lønbæk, and Russell Barnes. Implementation of an Ethernet-based communication channel for the Patmos processor. Technical Report DTU Compute 2015-02, Technical University of Denmark, 2015. http://orbit.dtu.dk/files/110841187/tr15_02_Pezzarossa_L.pdf.

[15] Martin Schoeberl and Rasmus Ulslev Pedersen. tpip: A time-predictable tcp/ip stack for cyber-physical systems. In *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, pages 75–82, May 2018.

[16] Martin Schoeberl, Luca Pezzarossa, and Jens Sparsø. A multicore processor for time-critical applications. *IEEE Design & Test*, 35:38–47, 2018.

[17] Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, and Daniel Prokesch. Patmos: A time-predictable microprocessor. *Real-Time Systems*, 54(2):389–423, Apr 2018.

[18] TTTech. TTETools – TTEthernet development tools v4.4. www.tttech.com.

[19] Weiying Zeng, Mohammed Khalid, and Sazzadur Chowdhury. A qualitative comparison of flexray and ethernet in vehicle networks. In *Proc. Canadian Conference on Electrical and Computer Engineering*, pages 571–576. IEEE, 2015.