Hardware assisted clock synchronization with the IEEE 1588-2008 precision time protocol

Kyriakakis, Eleftherios; Sparsø, Jens; Schoeberl, Martin

Published in:
Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS 2018

Link to article, DOI:
10.1145/3273905.3273920

Publication date:
2018

Document Version
Peer reviewed version

Hardware Assisted Clock Synchronization with the IEEE 1588-2008 Precision Time Protocol

Eleftherios Kyriakakis
Technical University of Denmark (DTU)
Kogens Lyngby, Denmark
elky@dtu.dk

Jens Sparsø
Technical University of Denmark (DTU)
Kogens Lyngby, Denmark
jspa@dtu.dk

Martin Schoeberl
Technical University of Denmark (DTU)
Kogens Lyngby, Denmark
masca@dtu.dk

ABSTRACT

Emerging technologies such as Fog Computing and Industrial Internet-of-Things have identified the IEEE 802.1Q amendment for Time-Sensitive Networking (TSN) as the standard for time-predictable networking. TSN is based on the IEEE 1588-2008 Precision Time Protocol (PTP) to provide a global notion of time over the local area network. Commonly, off-the-shelf systems implement the PTP in software where it has been shown to achieve microsecond accuracy. In the context of Fog Computing, it is hypothesized that future industrial systems will be equipped with FPGAs. Leveraging their inherent flexibility, the required PTP mechanisms can be implemented with minimal hardware usage and can achieve comparable synchronization results without the need for a PTP-capable transceiver. This paper investigates the practical challenges of implementing the PTP and proposes a hardware architecture that combines hardware-based time-stamping with a rate adjustable clock design. The proposed architecture is integrated with the Patmos processor and evaluated on an experimental setup composed of two FPGA boards communicating through a commercial-off-the-shelf switch. The proposed implementation achieves sub-microsecond clock synchronization with a worst-case offset of 138 ns.

CCS CONCEPTS

• **Computer systems organization** → **Embedded hardware**; *Embedded software*; • **Networks** → *Network protocols*;

KEYWORDS

IEEE 1588-2008, Precise Time Protocol, Hardware assist, Clock synchronization, WCET analysis, FPGA implementation

ACM Reference Format:
Eleftherios Kyriakakis, Jens Sparsø, and Martin Schoeberl. 2018. Hardware Assisted Clock Synchronization with the IEEE 1588-2008 Precision Time Protocol. In *26th International Conference on Real-Time Networks and Systems (RTNS '18), October 10–12, 2018, Chasseneuil-du-Poitou, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3273905.3273920

RTNS '18, October 10–12, 2018, Chasseneuil-du-Poitou, France

ACM ISBN 978-1-4503-6463-8/18/10…$15.00
https://doi.org/10.1145/3273905.3273920

1 INTRODUCTION

The IEEE 802.1 TSN task group [16] is in the process of standardizing Ethernet into a time-sensitive, deterministic, communication technology, by defining a range of sub-standards based on IEEE 802 networks. TSN is gaining popularity, in automotive and industrial automation networks over commonly used Fieldbus protocols, such as PROFINET and EtherCAT [4]. This is due to its support for: mixed-criticality traffic, high bandwidth and well-defined set of open standards [24, 28] that allow for interoperability between network devices. The deterministic communication of such systems is based on time-scheduled traffic with bound end-to-end latencies [3] and thus it requires a global notion of time to accurately synchronize network operations.

To ensure a global time reference among network devices, TSN employs the IEEE 1588-2008 Precision Time Protocol (PTP) [15] PTP enables accurate clock synchronization over master-slave network hierarchies, where the master is usually equipped with a high precision source of time (i.e., GPS, atomic clock). The worst-case precision of this clock synchronization correlates to the available scheduling accuracy of all network-based operations. There are two key mechanisms that influence the achieved clock precision of PTP, the method of time-stamping and the clock adjustment implementation [5]. These mechanisms are either implemented in software or via dedicated IEEE 1588-2008 compatible Ethernet PHY [26].

In the context of Industry 4.0, it is hypothesized that most upcoming industrial embedded systems, such as Fog Nodes, will be equipped with Field-Programmable Gate Array (FPGA) devices [9]. Based on the current price range of IEEE 1588-2008 compatible PHY transceivers, which is two times more expensive than standard PHYs and the decreasing prices of FPGAs, we argue that by taking advantage of the inherent flexibility of FPGAs, the PTP mechanisms can be implemented in the existing hardware without the need of costly PTP-capable PHY transceivers. These mechanisms can still provide comparable accuracy to PTP-capable PHY transceivers and at the same time minimize the cost of real-time systems by 30% while provide design flexibility and hardware reusability.

This paper explores the challenges of clock synchronization with PTP and proposes a hardware architecture that combines, a hardware IEEE 1588-2008 clock adjustment unit together with a PTP message recognition and timestamping unit. The proposed hardware is integrated within the FPGA-based platform T-CREST [21] and evaluated using the worst-case execution time (WCET) optimized processor Patmos [22]. The design is evaluated on a simple network composed of two T-CREST nodes, acting as a master-slave PTP pair, and connected through a single commercial-off-the-shelf switch.

The achieved clock synchronization between the two nodes is evaluated regarding two metrics, accuracy as average mean and jitter as standard deviation. The results are compared against a WCET analyzable software-based implementation of PTP. The contributions of this paper are:

- A hardware design that allows for network nodes to synchronize with sub-microsecond accuracy using standard PHYs.
- A WCET analysis of the PTP software-stack implementation.
- An evaluation of the achieved synchronization and the identified parameters affecting its accuracy and jitter.

The paper is organized in 6 sections: Section 2 provides the reader with a background on the fundamental concepts involved in the IEEE 1588-2008 PTP as well as a short introduction on the T-CREST platform and the WCET optimized processor Patmos. Section 3 reviews the challenges of implementing PTP and the different approaches that have been used. Section 4 presents the proposed hardware architecture and describes its integration with the T-CREST platform. Section 5 describes the experimental setup and presents the evaluation of the collected data from the implementations. Finally, Section 6 concludes the paper.

2 BACKGROUND

Different protocols have been developed over the years to achieve time synchronization between networked devices. The Network Time Protocol (NTP) [14] is one the most widely used protocols due to its compatibility with the Internet protocol as well as simple operation. It uses a polling mechanism where devices request the current time from a server to update their own notion of time. This protocol does not consider the network propagation delays along the path of the time server response and thus it can lead to significant offset and jitter. Filtering algorithms are usually implemented to reduce network-induced jitter and under special conditions (i.e., in local area networks), NTP can achieve sub-millisecond accuracies. However, this is not sufficient for real-time systems which often require microsecond precision.

2.1 IEEE 1588-2008 PTP

The IEEE 1588-2008 standard introduced PTP as an alternative mechanism to NTP to allow for sub-microsecond clock synchronization on local area networks. PTP is an Ethernet-based protocol that uses a periodic exchange of messages based on a master-slave network topology. This allows the precise calculation of each slave's clock offset, relative to its master, by considering the propagation delay of the message [5]. PTP is a distributed protocol that requires each ethernet port of an IEEE 1588-2008 compatible network device to execute the same stack of operations and implement the following fundamental blocks:

- IEEE 1588-2008 software stack
- IEEE 1588-2008 clock
- Clock adjustment
- Timestamp capturing
- Frame/Packet recognizer

These blocks can be implemented either in software or hardware and the most common cases are reviewed in Section 3. Each IEEE 1588-2008 network port can be either in a PTP_MASTER state or a PTP_SLAVE state. The port's state can be explicitly defined or implicitly by the best master clock selection algorithm. This paper focuses on the clock synchronization between two network devices, and thus the operation of the best master clock algorithm is out-of the scope of this paper. It is assumed that all ports are explicitly defined as PTP_MASTER or PTP_SLAVE.

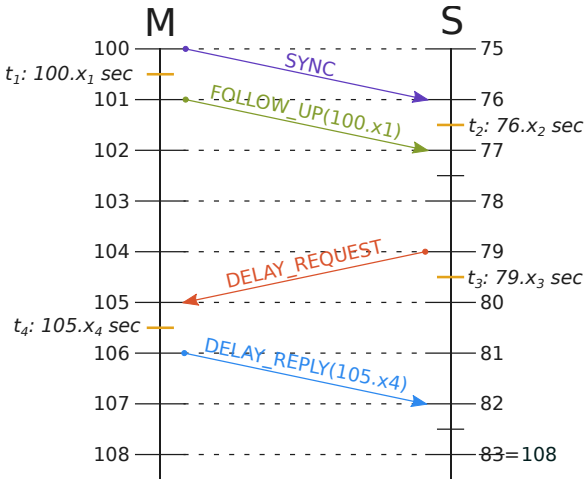


Figure 1: Simplified overview of the IEEE 1588-2008 PTP message flow. The example assumes a link delay of 1 second and variable delays x_i associated with each timestamp t_i

The calculation of the slave clock offset involves two metrics, *offset* and *delay* which are estimated using four timestamps t_1, t_2, t_3, and t_4 that are generated based on the messages, SYNC, FOLLOW_UP, DELAY_REQ and DELAY_REPLY respectively. There are two synchronization mechanisms supported by the PTP protocol, the one-message model and the two-message model. The one-message model can be used when high precision is not an application requirement and thus the message FOLLOW_UP containing the precise time of SYNC transmission (t_1) is not sent.

This paper implements the two-message model whose operation is illustrated in Figure 1 and described below. The master port **M** is responsible for periodically broadcasting SYNC messages and storing the transmission time in the timestamp t_1. Each SYNC message is followed by a FOLLOW_UP message containing the timestamp t_1. The slave port **S** keeps a receipt timestamp t_2 of the SYNC message and together with the timestamp t_1 contained in the FOLLOW_UP message it can estimate its *offset* from the master clock, but this does not take into consideration the transit time of the received messages. To calculate the propagation *delay* involved in the transmission, the slave port **S** sends a DELAY_REQ message to the master port **M** and keeps a transmit timestamp t_3. The master port **M** that receives the message replies with a DELAY_REPLY message containing the exact time it received the request t_4. The PTP slave can now accurately calculate its offset from the master taking into consideration also the transmission delay involved in their communication path. The use of the gathered timestamps by the slave in the calculations is shown in Equation 1 as described in [5].

$$offset = t_2 - t_1 - delay \tag{1}$$

where:

$$delay = \frac{dA + dB}{2}$$
$$dA = t_2 - t_1$$
$$dB = t_4 - t_3$$

The example PTP synchronization, presented in Figure 1, assumes a propagation delay of 1 second and variable delays x_i associated with each timestamp t_i. These delays correspond to the time interval between the actual transmission of a PTP message and the moment the timestamp is captured. Considering the timestamp capture delay variations in Equation 1 we calculate:

$$\begin{aligned} offset &= t_2 + x_2 - t_1 + x_1 \\ &\quad - \frac{1}{2}\left(t_2 + x_2 - t_1 + x_1 + t_4 + x_4 - t_3 + x_3\right) \Rightarrow \\ offset &= \frac{1}{2}\left(t_2 - t_1 + t_3 - t_4 + x_2 - x_1 + x_3 - x_4\right) \end{aligned}$$

Thus we can derive that the approximate jitter is:

$$jitter = \frac{1}{2}\left(x_2 - x_1 + x_3 - x_4\right) \quad (3)$$

As illustrated in Equation 3 the need for time-predictable timestamp capturing (timestamping) is a crucial step in calculating the precise clock offset and every component that handles a PTP message, until the timestamp is registered, increases the synchronization error by a small amount. The IEEE 1588-2008 standard defines that timestamping should occur precisely when the last bit of the start-of-frame (SOF) byte of an Ethernet frame is received as illustrated in Figure 2. However, depending on the timestamping technique used this cannot always be achieved. In general, there are three common ways of implementing timestamping as discussed in literature [5]:

(1) Software-based timestamping, is handled in software at the reception/transmission of an Ethernet frame from the MAC layer. The application has to interface with the MAC controller, pack/unpack the frame and check for valid PTP message, or at the transmission of a PTP frame.
(2) MAC-based timestamping, is handled by dedicated hardware on the MAC layer, most often implemented by an FPGA or a micro-controller. The hardware unit is responsible for parsing and timestamping the received frame from the PHY.
(3) PHY-based timestamping, is handled by a dedicated PHY device that incorporates both an IEEE 1588-2008 Clock and a PTP frame recognition & timestamping unit.

2.2 Experimental Platform

The implementation and evaluation described in this paper is built around the open-source platform T-CREST [21].

T-CREST is an FPGA-based multi-core platform that has been developed for on-going research in real-time applications and is based around the WCET-optimized processor Patmos [22]. Patmos is a time-predictable, dual-issue, RISC processor that has been designed with focus on WCET analysis. It uses special WCET-optimized instruction and data caches along with private scratchpad memories for instructions and data. It is supported by an LLVM-based [11] compiler, also optimized for WCET and by the WCET analysis tool *platin* [8].

The tool *platin* performs static analysis to compute the WCET of a certain code segment by using the information generated and preserved during compilation to determine a control flow graph. Together with low-level timing information of the processor architecture it can calculate a safe WCET of the analyzed code segment.

This work is integrated with the T-CREST platform as part of an on-going effort to provide support for time-triggered communication over TSN networks. Patmos is used to provide a time-predictable execution of the PTP software stack and together with the WCET analysis tool *platin* it is used to identify software-based causes of jitter in the PTP clock synchronization.

3 RELATED WORK

This section reviews the challenges and common approaches of implementing PTP, including timestamping and clock correction methods, as well as presents a state-of-art clock synchronization implementation.

Regarding the implementation of a PTP timestamping mechanism, different solutions have been presented that each tries to address different system challenges.

Software-based timestamping is very simple to implement in existing systems as it does not require any additional hardware. However, it can introduce significant jitter since the application runs in user space and its performance cannot be guaranteed. Both the processor load and the delay with handling interrupts or checking flags impact the precise moment of timestamping and lead to error offset and jitter. This is investigated in [2], where it was shown that under special conditions (i.e., low network traffic, high bit-rate connection) this implementation can achieve sub-millisecond clock synchronization.

MAC-based timestamping can be found implemented in modern commercial micro-processors, such as the STM32F105xx or the STM32F107xx [23] where the clock synchronization has been characterized by [29]. To the best of our knowledge, the implementation challenges and performance of this method have not been thoroughly investigated for IEEE 1588-2008 PTP. Related works such as [6] have investigated the concept of packet reception and timestamping using the AS6802 [10, 27] synchronization algorithm but the implementation or the precision of the achieved clock synchronization are not discussed. Designs for hardware-based PTP timestamping have been presented in [17, 20], but they have not been implemented in a real system nor evaluated in terms of their clock synchronization precision.

Finally, PHY-based timestamping has been implemented in commercial packages such as the Texas Instruments PHYTER [26]. This off-the-shelf component has been used in different works [12, 18] where it is presented that it can ensure nanosecond accuracy clock synchronization.

Regarding the clock correction mechanism, the IEEE 1588-2008 standard does not define an algorithm or procedure for adjusting the slave clock despite this having a significant influence on the overall precision of the system. Different approaches have been implemented to adjust the clock including:

- Clock rate adjustment by pulse addition and swallowing. An algorithm has been proposed by [30] and the procedure is also described in [5].

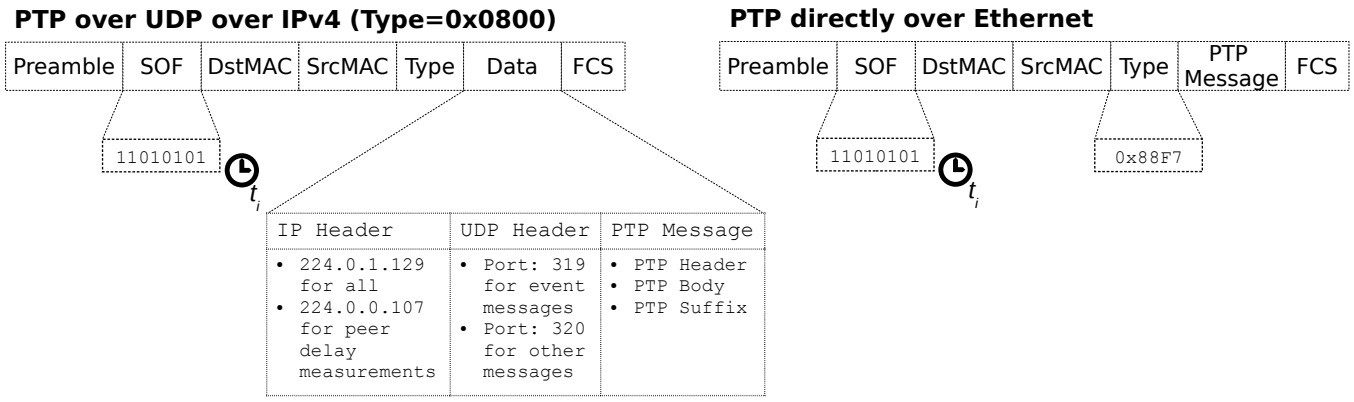


Figure 2: PTP Ethernet frame format depending on Ethernet Type. Moment of timestamping t_i is illustrated at SOF byte.

- Error register maintenance instead of clock correction. This technique is discussed in literature [5]. The error register is updated with the current offset from the master clock while the slave clock is never modified.

Finally, PTP has also been investigated in simulation to identify common sources of jitter and their effects on the synchronization as well as to estimate the best achievable clock synchronization in multi-hop large-scale networks [7]. The results show that the precision reduces as the number of hops increases and guaranteed precise synchronization proves challenging as the network scales.

At this point it is worth mentioning that research at the European Organization for Nuclear Research (CERN) has demonstrated the versatility of PTP and has shown that sub-nanosecond accuracy is possible. The developed application called White Rabbit [13] made use of PTP and Synchronous Ethernet standards on a custom network built on fiber optic links. The application achieved sub-nanosecond precision in the range of 135.25 ps with a standard deviation of approximately 6 ps.

This paper differs from related work as it aims to realize and evaluate an FPGA implementable architecture, able to provide nanosecond precision between master-slave node pairs using the IEEE 1588-2008 PTP on standard LAN networks without the use of dedicated PTP-capable Ethernet PHY hardware.

4 DESIGN AND IMPLEMENTATION

This section presents the hardware architecture that integrates with T-CREST, the proposed hardware-assist logic, for timestamping and clock adjustment, and finally the PTP software stack that is used to control and evaluate the design.

4.1 Hardware Architecture

The proposed hardware architecture, presented in Figure 3, is integrated with the T-CREST platform as a single IP core that interfaces with the Patmos processor. Its functionality is to snoop on the media independent interface RX/TX channels between the PHY and the MAC controller for PTP messages and provide times stamps for their arrival and departure accordingly. The unit is composed of three functional entities:

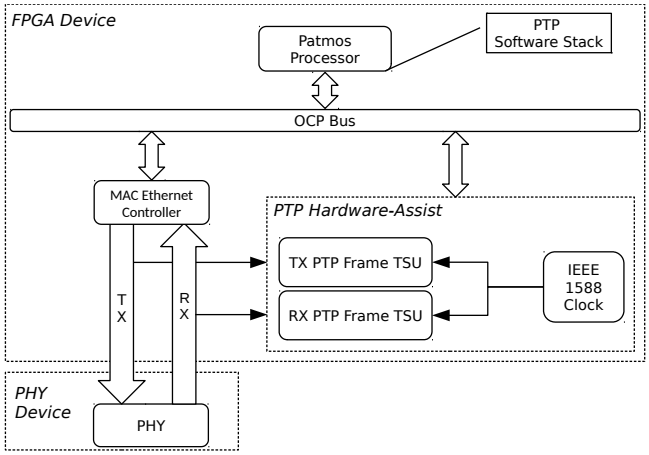


Figure 3: Implementation of PTP Hardware-Assist unit inside a T-CREST node. PHY TX/RX signals are split between the MAC controller and the hardware-assist PTP unit.

(1) The two *RX/TX timestamp units* (TSUs) that parse and timestamp a received or transmitted PTP frame, based on the SOF byte (see Figure 2). The units also include an interrupt/flag signal that is raised when a PTP frame has been parsed successfully and a valid timestamp is available for reading.
(2) The *IEEE 1588-2008 Clock* is composed of two counters representing seconds and nanoseconds that operate under a pre-scaled frequency. This unit is augmented with the proposed clock adjustment mechanism. In addition, it includes a configurable timer interrupt which can be used to schedule time-triggered network operations.
(3) The *PTP software stack*, executing on the Patmos processor, is responsible for the PTP message exchange as well as the offset calculation. The software stack is also responsible for managing the time-stamping, either by reading the *RX/TX timestamp units* or by reading the *IEEE 1588-2008 Clock* as well as for controlling the clock adjustment mechanism.

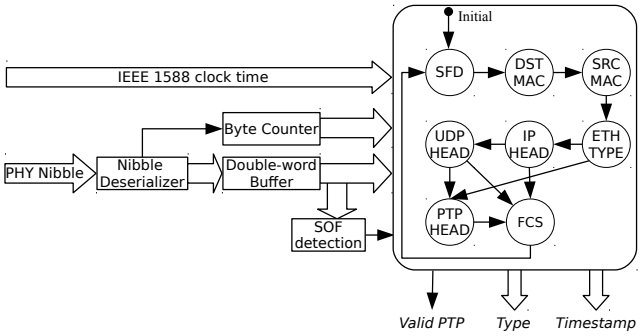


Figure 4: Implementation of the proposed RX/TX PTP timestamp unit (TSU).

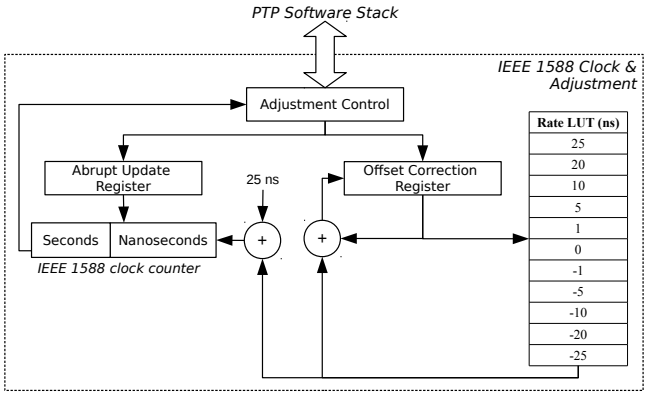


Figure 5: Implementation of the proposed IEEE 1588-2008 clock adjustment mechanism. Time-step of 25 ns **is related to a clock frequency of** 40 MHz **and can be tuned accordingly.**

4.2 RX/TX Timestamp Unit

The *RX/TX timestamp unit* (TSU) is presented in Figure 4. The hardware unit uses a finite state machine (FSM) to parse Ethernet frames as they are communicated between the PHY and the MAC controller. Incoming nibbles are de-serialized into bytes and consecutively stored into a double-word (64-bit) buffer. The FSM is initialized in the SFD (start-of-frame-detect) state where it waits and checks the double-word buffer until the frame's preamble and SOF sequence (`0x5555555555555D5`) is detected, at which moment it registers the current *IEEE 1588-2008 Clock* time and transitions to the next state to start parsing the incoming frame. Stepping through the FSM is done by keeping a record of how many bytes have been received at each state and resetting the counter when transitioning to a new state. States DSTMAC and SRCMAC read the Ethernet frame's destination and source MAC address respectively. State ETHTYPE is responsible for recognizing the received Ethernet frame's type and it provides support for the two PTP frame formats (see Figure 2). After the reception of the Ethernet type the state transitions to the IP state and consecutively to the UDP state where it parses the respective protocol headers. If the FSM detects that an IP header does not contain a valid UDP packet or that the UDP header does not contain a valid PTP message it proceeds to transition to state FCS (frame-check-sequence) where it remains until the remaining bytes have been received. When the FSM reaches the PTPHEAD state, it registers the stored IEEE 1588-2008 time along with the PTP message type and a valid bit indicating that a PTP timestamp is available for reading.

4.3 Clock Adjustment

The proposed *IEEE 1588-2008 clock & adjustment* mechanism is presented in Figure 5, and is composed of three parts:

(1) The *clock counter* which consists of a 48-bit nanosecond counter and a 32-bit seconds counter and complies with the time format specified by the IEEE 1588-2008 standard.
(2) The *abrupt update register* which can be used to instantaneously update the clock to a specific time value
(3) The *offset correction register* which is used to gradually correct the offset by adjusting the clock rate.

For large offsets (i.e., when a new node is connected to an already synchronized network or epoch changes), the time can be updated through the *abrupt update register*. For small values, the offset can be written directly in the *offset correction register*. The register indexes a look-up table (LUT) based on a set of configurable thresholds. The threshold values for this implementation are chosen empirically and are divided into the following categories for both positive and negative rates:

- $\pm$1 ms to $\pm$1 us
- $\pm$1 us to $\pm$100 ns
- $\pm$100 ns to $\pm$50 ns
- $\pm$50 us to $\pm$1 ns

The LUT controls the amount added/subtracted to or from the base time-step of the *clock counter* nanosecond counter. The operation increases or decreases the *offset correction register* according to the indexed LUT value and stops when its value reaches zero. This mechanism is based on the pulse addition and deletion technique discussed in Section 3 and works by gradually correcting the clock offset by increasing or decreasing the rate (time-step) of the *clock counter*. The LUT is indexed by a set of configurable thresholds for the value of *offset correction register*. The LUT rate values are chosen empirically, according to the resolution of the *clock counter* and allow to double the rate or completely stop the counter. Further fine tuning depending on the system's requirements can be applied.

Choosing between correcting the clock offset using the *abrupt update register* or the *offset correction register* is managed in software and can be configured by the `PTP_NS_OFFSET_THRESHOLD` parameter in code.

4.4 PTP Software Stack

The PTP software stack runs on the Patmos processor and involves the execution of a simplified PTP protocol were the master/slave mode is explicitly defined. Although open-source software projects that implement the full IEEE 1588-2008 standard are available [19], they are not developed with WCET in mind and are hardly time-predictable, thus they could not be used in our evaluation.

The PTP software stack is responsible for the following tasks:

- Initializing Patmos in master or slave port mode.

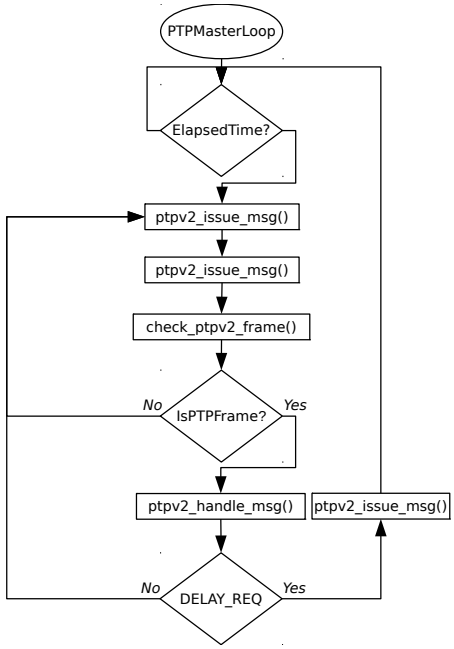
- Performing the clock synchronization, depending on the port mode.
- Reporting the clock offset at each synchronization interval.

The software is implemented in a way that both the PTP_MASTER and the PTP_SLAVE share the same codebase with the following functions:

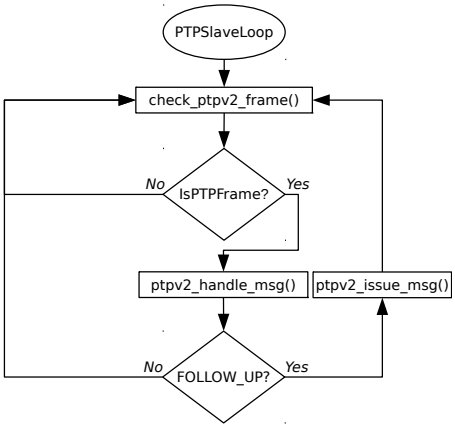
(1) `ptpv2_issue_message()` involves creating, sending and time-stamping the transmission of a PTP message. Checking the completion of the transmission and registering the timestamp is done by function `read_tx_timestamp()`.
(2) `check_ptpv2_frame()`, involves reading the MAC receive buffer, checking the ethernet type field of the frame and responding accordingly. When a PTP frame is detected the function `ptpv2_handle_message()` is called.
(3) `ptpv2_handle_message()` involves the unpacking, timestamping and the possible clock offset calculation/correction depending on the received PTP message type. Registering the time of reception timestamp is done by calling the function `read_tx_timestamp()`. Correcting the clock offset is done by first calling the functions `ptp_calc_one_way_delay()` and `ptp_calc_offset()`, for seconds and nanoseconds respectively, and finally calling the function `ptp_correct_offset()` for adjusting the clock.

Since both the PTP_MASTER and the PTP_SLAVE are explicitly defined, their operation was implemented as two simple cyclic procedures presented in Figures 6a & 6b respectively. The PTP_MASTER is responsible for issuing SYNC and FOLLOW_UP messages at a fixed rate as well as checking for any received PTP messages and replying to DELAY_REQ messages. The PTP_SLAVE is responsible for checking for any received PTP messages and, if a FOLLOW_UP message is received, replying with a DELAY_REQ message.

As presented in [2], multi-tasking can have a negative impact on the clock synchronization precision of software-based PTP. Taking this into account, to allow us to compare the performance of the proposed hardware-assist mechanisms with the best-case software execution, the application is implemented on a single task environment on the Patmos processor.



(a) PTP_MASTER loop



(b) PTP_SLAVE loop

Figure 6: Program flow of PTP master and slave

5 EVALUATION

This section presents the experimental setup over which the proposed hardware architecture was evaluated as well as its hardware resources. Finally, the collected results from the WCET analysis and the clock synchronization are discussed.

5.1 Experimental Setup

The presented hardware architecture was synthesized on two FPGA Terasic DE2-115 boards and explicitly configured as a PTP master/slave pair. The clock synchronization was evaluated on a simple experimental setup composed of the two FPGA boards communicating over a single off-the-shelf switch via a 100 Mbps Ethernet. Each FPGA board used a PLL to generate the internal logic clocks. The Patmos processor was operating at frequency of 80 MHz. The *IEEE 1588-2008 clock* was operating at a frequency of 40 MHz and had a resolution of 25 ns. The PLL input was provided by a commercial off-the-shelf oscillator operating at a nominal frequency of 50 MHz with an accuracy of 50 ppm.

5.2 Hardware Resources

The hardware was synthesized for an Altera Cyclone IV FPGA [1]. Table 1 presents the hardware utilization of the proposed PTP Hardware-Assist IP. The values outside the parentheses indicate the aggregate resources used by the entity, while the value inside the parentheses indicate the utilization of the specific entity alone.

The hardware cost of the proposed hardware-assist unit is minimal. The utilization of PTP Hardware-Assist is only 1.7 % of the total available resources of the Cyclone IV FPGA device (114480 Logic Elements) and when compared to the the small-sized processor Patmos, it is 11 % of its total size (13503 LUTs and 8325 Logic Registers).

Table 1: Hardware-assist Architecture Resource Utilization

Entity	Combinational LUTs	Logic Registers
PTP Hardware-Assist	1485 (82)	1182 (142)
MIITimestampUnit	454 (376)	402 (327)
DeserializePHYbyte	13 (13)	11 (11)
DeserializePHYBuffer	65 (65)	64 (64)
RTC	431 (431)	234 (234)

Table 2: WCET Analysis of PTP Software Stack

Function	WCET	
	Clock Cycles	Time (at 80 MHz)
`ptpv2_issue_msg()`	2560141	32 ms
`readTXTimestamp()`	5	62.5 ns
`check_ptpv2_frame()`	684	8.55 us
`ptpv2_handle_msg()`	3893	48.6 us
`readRXTimestamp()`	5	62.5 ns
`ptp_correct_offset()`	66	850 ns
`ptp_calc_offset()`	4	50 ns
`ptp_calc_one_way_delay()`	7	87.5 ns

5.3 WCET Analysis

To reveal possible sources of jitter, as well as to estimate the processor load involved in the execution of the PTP software-stack, a formal WCET analysis was performed using the tool *platin* [8] and the results are presented in Table 2.

The WCET revealed that the worst-case delay between the arrival of a PTP message and the software capturing the timestamp to be 8.6 us. This includes the software packing/unpacking the frame plus registering the time of arrival/departure that amounts to $685 + 5 = 689$ WCET clock cycles. As shown in Equation 3 this can lead to significant error in the calculated clock offset and consecutively lead to jitter. If hardware-based timestamping is used, the worst-case delay of 689 clock cycles does not introduce any jitter, since the timestamp has already been captured in hardware and thus the software only needs to read the stored value.

Furthermore, the WCET analysis of functions `ptpv2_issue_msg()` and `ptpv2_handle_msg()` showed that there is a significant overhead in the processor to execute the PTP protocol.

We propose as future work a complete in-hardware implementation of the PTP synchronization. Building up from the presented results, the proposed hardware-assist architecture can be extended with the addition of a PTP message generator controller. This is hypothesized to both minimize jitter but also significantly reduce the processor load especially in multi-tasking environments where the precision of PTP can be reduced significantly [2]. This will also allow for greater scalability on large scale networks, were devices require more than one ethernet ports to synchronize using PTP.

5.4 Clock Synchronization

To evaluate the clock synchronization, the PTP_SLAVE was configured to report, over serial a port, the calculated clock offset at each PTP synchronization interval (after all four timestamps were gathered). To best evaluate the performance of the proposed implementation, four sets of results were collected by testing different combinations of implementation, namely:

- software-based timestamping
- hardware-based timestamping
- abrupt clock updates
- clock rate control adjustment

As a base of comparison to the evaluation of the results and to determine the static drift between the master and slave clocks on the two FPGA boards, measurements were gathered using PTP but without implementing any corrections or adjustments to the slave clock. Figure 7 presents the PTP slave's clock offset as calculated after an initial correction and no further adjustments. The relative drift was estimated at an average of 34 us/sec, which corresponds to 34 ppm and illustrates the need for accurate synchronization.

First, the effects of the proposed hardware-based timestamping against software-based timestamping were compared in terms of jitter as standard deviation. Figure 8 presents the results from 18000 collected samples from two different measurements with a *SYNC* period of 0.5 ms. The calculation used only abrupt updates for correcting the clock offset, to clearly reveal the influence of the timestamping mechanism. The hardware-based timestamp mechanism managed to reduce jitter to a standard deviation of 49.8 ns, while software-based timestamping could only achieve a standard deviation of 95.3 ns.

Secondly, the effects of the proposed rate control mechanism were investigated in terms of accuracy (avg. mean) of the calculated offset. The measurements presented in Figure 8 show that there is an avg. mean offset of 154 ns. This offset is introduced by the time it takes to read the clock, add the delay and write-back the new value into the clock. Figure 9 presents and compares the improved avg. mean using the proposed rate control mechanism against the achieved accuracy using only abrupt updates. The data were collected from two different measurements that both used a 0.5 ms *SYNC* message period and hardware-based timestamping. The achieved achieved accuracy of rate-control was within 17 ns with a std. deviation of 48.7 ns.

The results show that the timestamping method influences the jitter of the calculated offset, while the clock adjustment method was mainly responsible for the accuracy of the achieved synchronization but also revealed a slight improvement in jitter.

The proposed PTP Hardware-Assist implementation offers submicrosecond clock synchronization of an avg. mean of -17.3 ns and a jitter of approx. 48.7 ns with a worst-case clock offset of 138 ns. The results improve the worst-case synchronization offset of 500 ns that was achieved when using only software-based timestamping and abrupt updates. The presented results also achieve better worst-case offset when compared to the related architecture of the STM32F107xx microprocessor, which as characterized by [29] it achieves a worst-case offset of 260 ns. Moreover, the achieved performance is comparable to the clock synchronization of a commercial PTP-capable Ethernet PHY transceiver that is presented in [12, 18], and allows for an avg. mean of 10 ns and a jitter of approximately 50 ns.

We propose as future work to extend the experimental setup and evaluate the presented architecture over a large-scale TSN network

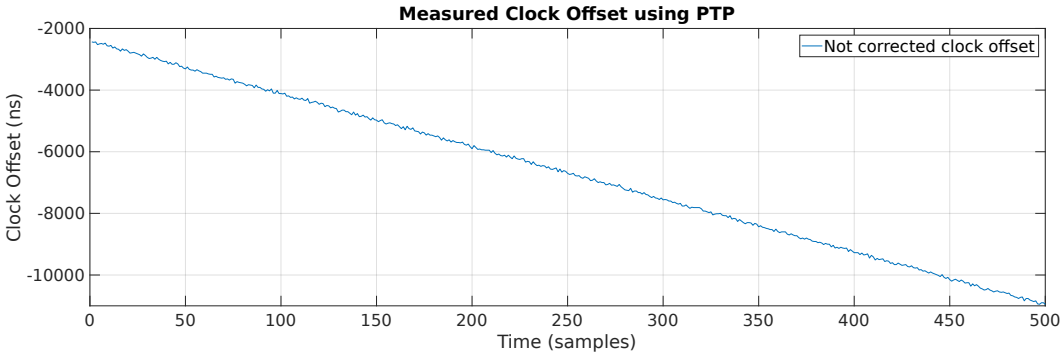


Figure 7: Measured clock offset between the two FPGA boards (using hardware-based time-stamping).

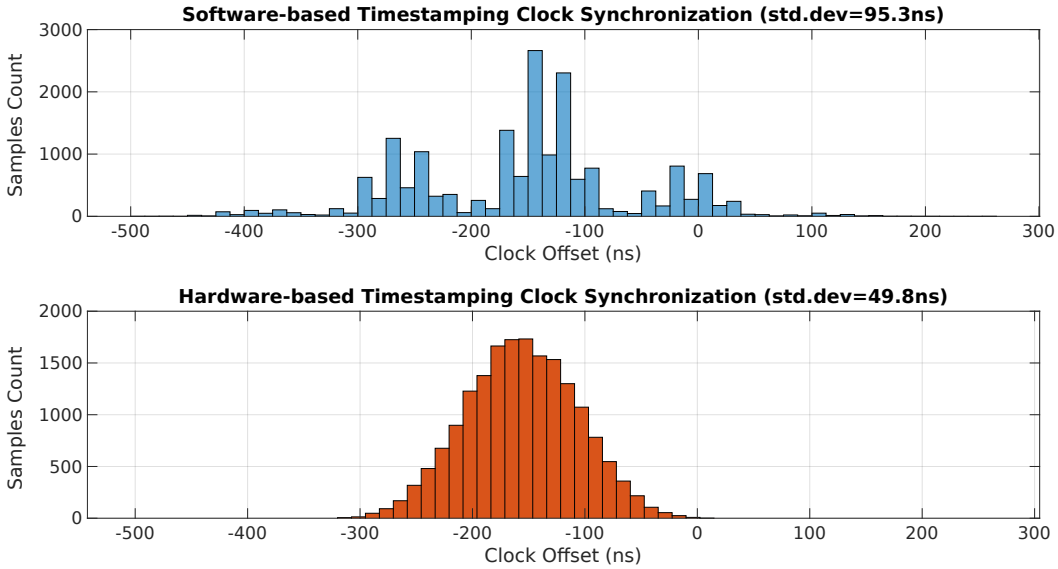


Figure 8: PTP-Slave clock offset timestamping method comparison between software-based (top) and hardware-based (bottom).

composed of multiple T-CREST nodes. It is hypothesized that traffic load and multiple-hops will have a negative effect on the clock synchronization precision between network nodes due to an increased transit time of PTP messages, as shown in the analysis of [7] [2]. In addition we plan to increase the resolution of the *clock counter*. Finally, the WCET analysis highlighted the need for hardware-based timestamping as well as revealed how time intensive the execution of PTP is. Based on the results we plan to investigate a complete in-hardware solution for performing the PTP synchronization, including transmission of PTP frames, that will effectively reduce the processor load of network devices as well as provide transparent to the user clock synchronization.

5.5 Source Access

The presented hardware-architecture is integrated with the open-source project T-CREST which is hosted at [25]. The PTP Hardware-Assist design can be found at *https://github.com/t-crest/patmos/tree/master/hardware/src/main/scala/ptp1588assist*. The PTP software is part of the ethernet lib of Patmos and is available at *https://github.com/t-crest/patmos/tree/master/c/ethlib*. To monitor the live offset from a connected PTP_SLAVE T-CREST node, a visualization script was developed available at *https://github.com/t-crest/patmos/blob/master/c/ethlib/other/plotPTPOffset.py*

6 CONCLUSION

This paper investigated the IEEE 1588-2008 Precise Time Protocol, which provides a global time reference for IEEE 802.1 TSN networks. We explored the design of a hardware-assisted implementation of PTP using a standard Ethernet PHY transceiver and finally implemented a hardware architecture that can achieve sub-microsecond precision.

The proposed architecture was successfully integrated with the T-CREST and synthesized on FPGA with minimal hardware overhead. The PTP software stack was implemented on the time-predictable

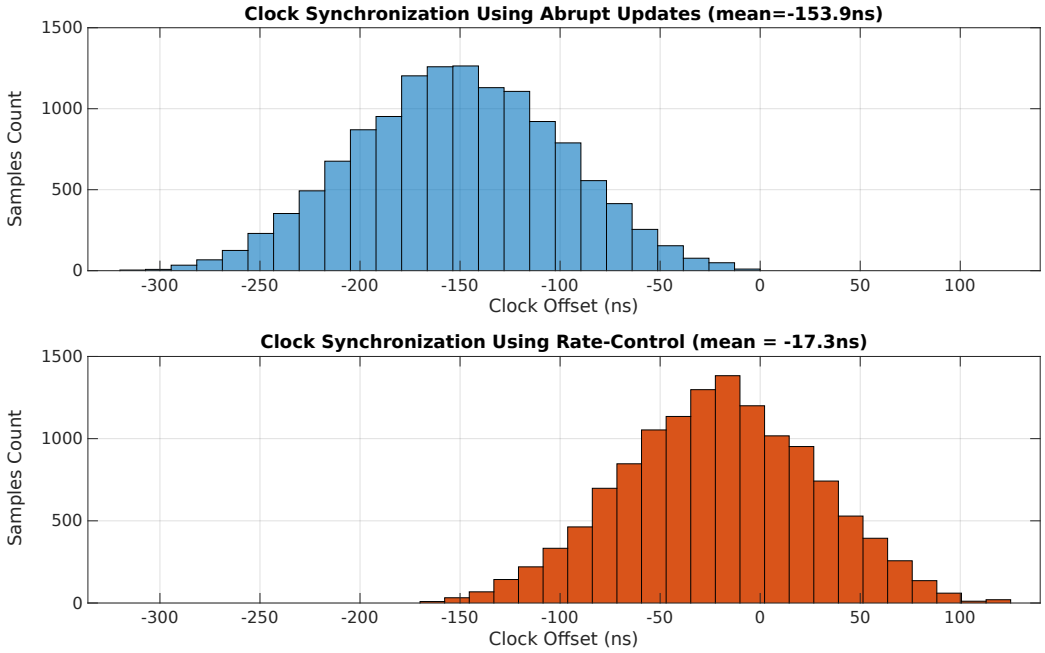


Figure 9: PTP-Slave clock offset adjustment method comparison between abrupt-updates (top) and rate-control (bottom).

Patmos processor which allowed for a full WCET analysis of the application.

The clock synchronization was evaluated on an experimental setup, composed of two FPGA boards implementing the proposed architecture and communicating through a commercial off-the-shelf switch at 100 Mbps. The evaluation was performed using two metrics, jitter as standard deviation and accuracy as average mean, and data were collected over a variety of timestamping and clock adjustment combinations.

The results showed that the proposed architecture greatly improved the precision of software-based timestamping and it achieved comparable results with commercial off-the-shelf PTP-capable Ethernet PHY transceivers, showing that FPGA-based PTP clock synchronization is feasible.

ACKNOWLEDGMENTS

This is work was part of the Fog Computing for Robotics and Industrial Automation (FORA) European Training Network (ETN) funded by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 764785.

REFERENCES

[1] ALTERA 2016. *Cyclone IV FPGA Device Family Overview*. ALTERA. Retrieved 03/07/2018 from https://www.altera.com/en_US/pdfs/literature/hb/cyclone-iv/cyiv-51001.pdf

[2] L Benetazzo, C Narduzzi, and M Stellini. 2007. Analysis of clock tracking performances for a software-only IEEE 1588 implementation. In *Proceedings of the Instrumentation and Measurement Technology Conference Proceedings (IMTC)*. IEEE, 1–6.

[3] Silviu S Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. 2016. Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM, 183–192.

[4] Peter Danielis, Jan Skodzik, Vlado Altmann, Eike Bjoern Schweissguth, Frank Golatowski, Dirk Timmermann, and Joerg Schacht. 2014. Survey on real-time communication via ethernet in industrial automation environments. In *Proceedings of the Emerging Technology and Factory Automation (ETFA)*. IEEE, 1–8.

[5] John C. Eidson. 2006. *Measurement, Control, and Communication Using IEEE 1588* (1st ed.). Springer Science & Business Media.

[6] Friedrich Groß, Till Steinbach, Franz Korf, Thomas C Schmidt, and Bernd Schwarz. 2014. A hardware/software co-design approach for ethernet controllers to support time-triggered traffic in the upcoming IEEE TSN standards. In *Proceedings of the Fourth International Conference on Consumer Electronics–Berlin (ICCE-Berlin)*. IEEE, 9–13.

[7] Marina Gutiérrez, Wilfried Steiner, Radu Dobrin, and Sasikumar Punnekkat. 2017. Synchronization quality of IEEE 802.1 AS in large-scale industrial automation networks. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 273–282.

[8] Stefan Hepp, Benedikt Huber, Jens Knoop, Daniel Prokesch, and Peter P. Puschner. 2015. The platin Tool Kit - The T-CREST Approach for Compiler and WCET Integration. In *Proceedings 18th Kolloquium Programmiersprachen und Grundlagen der Programmierung, KPS 2015, Pörtschach, Austria, October 5-7, 2015*.

[9] Intel 2018. *Intel's Fog Reference Design Overview*. Intel. Retrieved 22/06/2018 from https://www.intel.com/content/www/us/en/internet-of-things/fog-reference-design-overview.html

[10] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. 2005. *The time-triggered ethernet (TTE) design*. IEEE. 22–33 pages.

[11] Chris Lattner and Vikram S. Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *International Symposium on Code Generation and Optimization (CGO'04)*. IEEE Computer Society, 75–88.

[12] Yan Lin, Li Hao, and Tian Dan. 2011. Research and implementation in synchronized system of data acquisition based on IEEE 1588. In *Proceedings of the 10th International Conference on Electronic Measurement & Instruments (ICEMI)*, Vol. 2. IEEE, 198–201.

[13] Maciej Lipiński, Tomasz Włostowski, Javier Serrano, and Pablo Alvarez. 2011. White rabbit: A PTP application for robust sub-nanosecond synchronization. In *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*. IEEE, 25–30.

[14] David L. Mills. 2006. *Computer Network Time Synchronization: The Network Time Protocol*. CRC Press.

[15] Institute of Electrical and Electronics Engineers. 2008. *1588-2008 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. IEEE.

[16] Institute of Electrical and Electronics Engineers. 2016. Time-Sensitive Networking Task Group. Retrieved 22/06/2018 from http://ieee802.org/1/pages/tsn.html

[17] Jae Won Park, Jin Ha Hwang, Won Young Chung, Seung Woo Lee, and Yong Surk Lee. 2011. Design time stamp hardware unit supporting IEEE 1588 standard. In *SoC Design Conference (ISOCC), 2011 International*. IEEE, 345–348.

[18] Jeff Preston, Dan Blankenship, Les Hoy, MF Ohmes, Andrey Gueorguiev, and Juergen Stein. 2010. Novel timing method using IEEE 1588 and synchronous Ethernet for Compton telescope. In *Proceedings of the Nuclear Science Symposium Conference Record (NSS/MIC)*. IEEE, 1404–1407.

[19] PTPd. 2015. PTPd. Retrieved 02/06/2018 from https://github.com/ptpd/ptpd

[20] Mingzhu Qi, Xiaoli Wang, and Zhiqiang Yang. 2011. Design and implementation of IEEE1588 time synchronization messages timestamping based on FPGA. In *Electric Utility Deregulation and Restructuring and Power Technologies (DRPT), 2011 4th International Conference on*. IEEE, 1566–1570.

[21] Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Alexander Jordan, Evangelia Kasapaki, Jens Knoop, Yonghui Li, Daniel Prokesch, Wolfgang Puffitsch, Peter Puschner, André Rocha, Cláudio Silva, Jens Sparsø, and Alessandro Tocchi. 2015. T-CREST: Time-predictable Multi-Core Architecture for Embedded Systems. *Journal of Systems Architecture* 61, 9 (2015), 449–471. https://doi.org/10.1016/j.sysarc.2015.04.002

[22] Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, and Daniel Prokesch. 2018. Patmos: A Time-predictable Microprocessor. *Real-Time Systems* 54(2) (Feb 2018), 389–423. https://doi.org/10.1007/s11241-018-9300-4

[23] ST Microelectronics 2017. *STM32F107VC*. ST Microelectronics. Retrieved 03/07/2018 from https://www.st.com/en/microcontrollers/stm32f107vc.html

[24] Wilfried Steiner and Stefan Poledna. 2016. Fog computing as enabler for the Industrial Internet of Things. *e & i Elektrotechnik und Informationstechnik* 133, 7 (2016), 310–314.

[25] T-CREST. 2017. Patmos Source. Retrieved 02/06/2018 from https://github.com/t-crest/patmos

[26] Texas Instruments 2015. *DP83640 Precision PHYTER–IEEE 1588 Precision Time Protocol Transceiver*. Texas Instruments. Retrieved 03/07/2018 from www.ti.com/lit/ds/symlink/dp83630.pdf

[27] TTTech. 2011. *AS6802: Time-Triggered Ethernet*. SAE International.

[28] TTTech. 2015. Deterministic Ethernet & TSN: Automotive and Industrial IoT. *Industrial Ethernet Book* 89 (July 2015). Retrieved 03/07/2018 from https://www.tttech.com/fileadmin/content/general/secure/pdf/IEB89_2015-Deterministic-Ethernet-TSN_Automotive-and-Industrial-IoT.pdf

[29] Guang You Yang, Yi Zheng, Zhi Yan Ma, and Xin Yu Hu. 2012. The Implementation of IEEE 1588-2008 Precision Time Protocol on the STM32F107. *Key Engineering Materials* 522 (2012), 868–873.

[30] Weidong Ye. 2009. IEEE1588 Clock servo algorithm. In *Proceedings of the 9th International Conference on Electronic Measurement & Instruments*. IEEE, 5274861.