Sub-Microsecond Time Synchronization for Network-Connected Microcontrollers

Jens Schleusner
Institute of Microelectronic Systems
Leibniz University Hannover
Hannover, Germany
schleusner@ims.uni-hannover.de

Christian Fahnemann
Institute of Microelectronic Systems
Leibniz University Hannover
Hannover, Germany
fahnemann@ims.uni-hannover.de

Richard Pfleiderer
Institute of Microelectronic Systems
Leibniz University Hannover
Hannover, Germany
pfleiderer@ims.uni-hannover.de

Holger Blume
Institute of Microelectronic Systems
Leibniz University Hannover
Hannover, Germany
blume@ims.uni-hannover.de

***Abstract*—This paper presents a bare-metal implementation of the IEEE 1588 Precision Time Protocol (PTP) for network-connected microcontroller edge devices, enabling sub-microsecond time synchronization in automotive networks and multimedia applications. The implementation leverages the hardware timestamping capabilities of the microcontroller (MCU) to implement a two-stage Phase-locked loop (PLL) for offset and drift correction of the hardware clock. Using the MCU platform as a PTP master enables the distribution of a sub-microsecond accurate Global Positioning System (GPS) timing signal over a network. The performance of the system is evaluated using master-slave configurations where the platform is synchronized with a GPS, an embedded platform, and a microcontroller master. Results show that MCU platforms can be synchronized to an external GPS reference over a network with a standard deviation of 40.7 nanoseconds, enabling precise time synchronization for bare-metal microcontroller systems in various applications.**

***Index Terms*—PTP, Precision Time Protocol, Microcontroller, Embedded System, TSN, Time Sensitive Networking**

I. Introduction

Distributed real-time systems of network-connected embedded devices are an essential building block for a wide variety of consumer, industrial, automotive, and scientific applications, such as the network distribution of synchronized audio-video data in multimedia setups, coordination of machines for manufacturing, data transfer between Vehicle-to-Everything (V2X) nodes, and synchronization of distributed scientific measurements. Precise time synchronization over the network is a common requirement for all these Time Sensitive Networking (TSN) applications. The Network Time Protocol (NTP) is commonly used to synchronize the time of network devices to a local or internet clock. NTP is software-based and typically operates down to millisecond precision, but NTP is not designed to fulfill the synchronization constraints of real-time systems, or systems that require sub-millisecond synchronization to operate [1].

In digital audio networks with multiple network-connected speakers, a microsecond synchronization accuracy is required for precise phase adjustment and sample-synchronous playback. The Audio Engineering Society (AES) specification AES11 for digital audio transmission requires all output phases to be within 5 % of the reference frame timing ($\pm 1\,\mu s$ at 48 kHz) [2].

The automotive sector plays a major role in future deployments of TSN applications. V2X applications require sub-millisecond synchronization for network coordination and scheduling [3]. Sub-microsecond precision is needed for localization and timing-based security measures [3]. Furthermore, the In-Vehicle Network (IVN) moves from Controller Area Network (CAN)-based systems to Ethernet-based network architectures. The network is shared between critical control tasks, sensor data transfer, and entertainment applications. The scheduling of the network traffic requires precise sub-microsecond time synchronization to ensure deterministic timing for critical control signals while providing sufficient throughput for driver assistance and multimedia systems [4]. Fig. 1 shows two vehicles with synchronized IVN clock domains that exchange timestamped data via V2X.

When microsecond or nanosecond precision is required, GPS-based solutions can be deployed [5]. The GPS receiver generates a very accurate Pulse Per Second (PPS) hardware signal from the satellite data to provide frequency and offset synchronization to the device. Deploying GPS receivers to every network node is expensive and often challenging. The antenna must be deployed within line of sight to the sky, and the PPS connection requires an individual cable connection. Instead of connecting all network nodes to a PPS source, the IEEE 1588 Precision Time Protocol (PTP) is used for precision clock synchronization between a time master and all connected slave devices over an existing network connection [6]. IEEE 802.1AS generalized Precision Time Protocol (gPTP) is based on PTP but tailored towards industrial and automotive applications [7]. Data is sent either as UDP packets or as raw Ethernet frames. This work uses the term PTP as a generalized

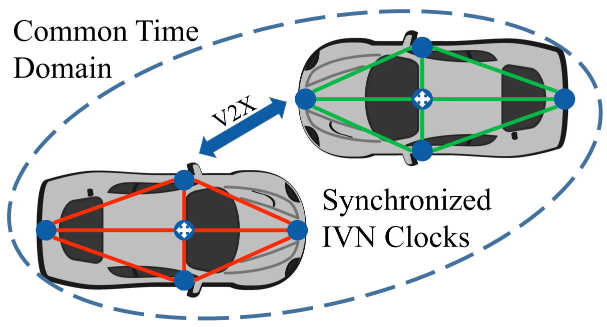


Fig. 1. Vehicles exchange timestamped data via Vehicle-to-Everything (V2X). A common time domain for the data is established with IEEE 1588 Precision Time Protocol (PTP) synchronized In-Vehicle Network (IVN) clock domains.

term for all protocol variants. The implementation of PTP relies on hardware timestamping support by the network interface for precisely measuring the receive and send time of the packets to compute the offset and drift of the local clock. PTP implementations are available for many different platforms. The Linux Kernel provides PTP functionality in its driver modules and a PTP Hardware Clock (PHC) module [8] which synchronizes the system clock to the timing information provided by the network interface [9]. The PHC driver achieves sub 1 µs synchronization performance with an Intel i210 network card [10]. Embedded Linux systems such as the Raspberry Pi Compute Module 4 (CM4) provide PTP support as well. microcontroller (MCU) implementations are available for different Real-Time Operating System (RTOS), such as FreeRTOS [11] and Zephyr OS [12], which are commonly used in industrial production-grade setups.

A. Contributions

In automotive development and research, MCUs are often used bare-metal for sensor data acquisition over the network. Numerous bare-metal drivers are commonly available for many different sensor parts. A bare-metal PTP implementation would allow network data acquisition by multiple distributed MCU-based sensor nodes with a common reference time base. Such a bare-metal implementation for MCUs is currently not publicly available. Moreover, the public research and available code for RTOS is limited to the slave mode. Expensive metrology-grade master devices are typically used to provide the reference time. This paper proposes an open-source bare-metal implementation of PTP and gPTP for the network-connected NXP i.MX RT1062 MCU using its hardware-timestamping capabilities to achieve sub-microsecond-grade time synchronization in distributed network applications. We selected this MCU due to the available Ethernet driver library [13], a manufacturer-provided application note [14] that describes the use of the integrated PTP hardware, and the availability of a lot of input-output peripherals, which makes this MCU suitable as a time-synchronized base platform in different applications. In addition to the slave mode, our implementation can be used as a time master to distribute a sub-

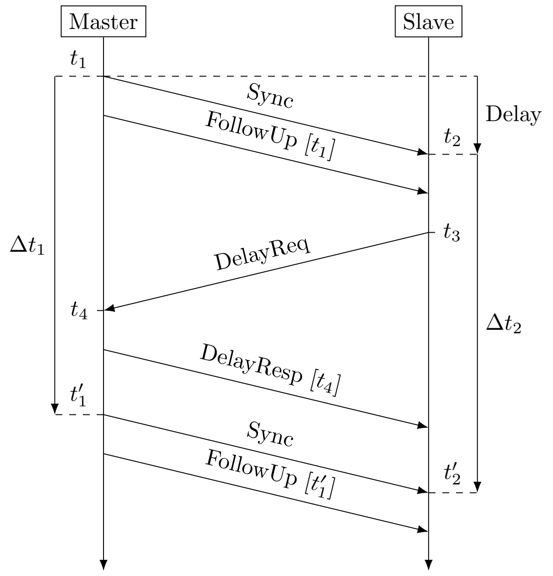


Fig. 2. Precision Time Protocol: The slave estimates the delay and offset of the local clock by exchanging messages with a master and measuring transmit and receive timestamps.

microsecond accurate GPS timing signal in a network without requiring a costly metrology time master. We evaluate the performance of our implementation in different master-slave scenarios and compare the achieved timing accuracy to CM4 embedded systems. The performance of our implementation as a GPS-synchronized time master is evaluated separately.

II. System Architecture

A. Hardware Timestamping

Precision hardware timestamping is the basis for the PTP algorithm. The Ethernet standard specifies a timestamping mechanism in the physical layer (PHY) of the OSI reference model [15, p. 368]. In embedded systems such as the CM4, the PHY is often implemented as a separate chip, which is connected via a Media Independent Interface (MII) connection to the Media Access Control layer (MAC) integrated with the main CPU [16]. Timestamping in the PHY avoids data delays caused by buffering, as it measures the timing directly at the wire [15, p. 374]. The PHY uses a hardware block configured to trigger timestamping specifically for the IEEE 1588 PTP packets. Another option is to implement hardware timestamping within the MAC. This approach is implemented on our target MCU platform and can provide timestamps for all network packets. The MAC approach enables the implementation of gPTP, which uses different peer-to-peer packet types than IEEE 1588 PTP with the same synchronization algorithm but is unavailable on the CM4.
The PHY [17] on the MCU development board has an asymmetric data delay for transmitted and received packets. This asymmetry of approximate 200 ns causes an offset when the MCU target is connected to a system with symmetric transmit and receive latency. Furthermore, the asymmetry is invisible to the PTP algorithm that only measures round-trip delays. It

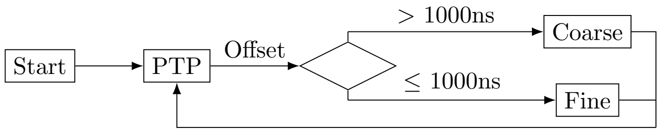


Fig. 3. For PTP-offset values above 1000 ns, a coarse offset correction is performed as a starting point for the more accurate fine clock controller.

is therefore adjusted manually specifically for this transceiver chip.

B. Precision Time Protocol

The PTP clock synchronization mechanism [6] is based on exchanging multicast network packets between a master clock and a slave clock (Fig. 2). The transmit and receive times of the messages are accurately recorded in hardware by the network cards to calculate the drift and offset of the slave clock relative to the master. The master clock is assumed to be stable and is typically synchronized to an external time source with high precision, such as a GPS receiver. The master sends Sync packets in periodic intervals (1 s), which are received by all slaves on the network. The master network card records a transmission timestamp t_1 relative to the master clock. On arrival of the Sync packet, the slave network card records a receive timestamp t_2 relative to the slave clock. In a FollowUp packet, the master sends t_1 to the slave. An important metric for clock synchronization accuracy is the clock drift. A drift of 0 indicates perfect frequency synchronization. If at least two pairs of Sync timestamps $(t_1 t_2, t'_1 t'_2)$ are available to the slave, it can compute its clock drift relative to the master.

$$\text{Drift} = 1 - \frac{\Delta t_2}{\Delta t_1} = 1 - \frac{t'_2 - t_2}{t'_1 - t_1} \tag{1}$$

The delay of the transmission line is measured to calculate the slave offset. The slave sends a DelayRequest packet to the master and records the transmission timestamp t_3. The master records the receive timestamp t_4 and sends t_4 to the slave in a DelayResponse packet. Assuming a symmetrical data path between both hosts, the slave can compute the path delay.

$$\text{Delay} = \frac{\Delta t_{4,1} - \Delta t_{3,1}}{2} = \frac{(t_4 - t_1) - (t_3 - t_2)}{2} \tag{2}$$

With a known path delay, the slave can calculate its clock offset to the master.

$$\text{Offset} = (t_2 - t_1) - \text{Delay} \tag{3}$$

All measurements are repeated periodically when a new sync message is received. Based on Drift and Offset, we implemented a two-step clock Phase-locked loop (PLL).

C. MCU Counter Correction

We selected the NXP i.MX RT1062 System on Chip with an ARM Cortex-M7 running at 600 MHz as our target MCU system. It has an integrated 100 Mbit/s Ethernet MAC with hardware timestamping capability and is available with a PHY [17] on the Teensy 4.1 development board [18]. As the backend of our implementation, we use a modified version of the QNEthernet library, which serves as the driver for the interface to the hardware timers of the Ethernet MAC [13].

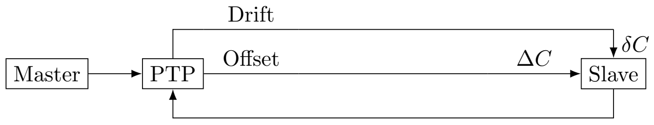


Fig. 4. The MCU implementation of coarse drift and offset correction directly corrects the local clock to within 1000 ns.

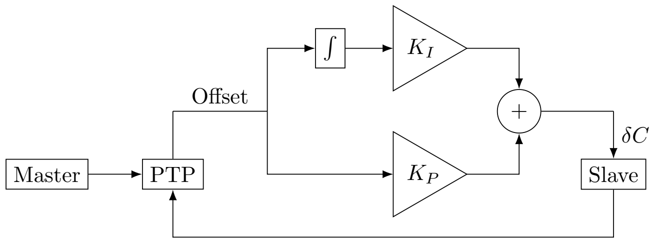


Fig. 5. The fine PI-controller-based clock PLL corrects the MCU clock offset precisely by adjusting the local clock increments.

The PTP hardware counter of the system is implemented as an adjustable nanosecond counter which is incremented by 40 (40 ns) per clock cycle of an oscillator with $f_{\text{OSC}} = 25\,\text{MHz}$. At a counter value of 10^9 (1 s), a modulo-based overflow, which keeps the remainder for the beginning of the next second, ensures the long-term accuracy of the counter [14]. In contrast to a counter reset on overflow, the modulo operation retains the overflow increments as part of the next second. Interrupts due to the transmitting or receiving of network packets, as well as external pin interrupts from an external PPS source, trigger a capture of the current counter value in hardware, which can be read by the application.
Based on the clock drift provided in Eq. 1, the counter drift per second δC is computed.

$$\delta C = \text{Drift} \cdot 10^9/\text{s} \tag{4}$$

The drift of the counter δC is corrected with an adjusted correction increment value of either 39 ns, or 41 ns. The adjusted correction increment value is used instead of the nominal 40 ns increment every n cycles of the 25 MHz oscillator to slow the counter down or to speed it up [14]. This way, the counter drift correction by δC is distributed evenly over a full second without large discontinuities of the counter value. Large values of n allow for fine corrections in the order of nanoseconds per second.

$$n = \frac{f_{\text{OSC}}}{\delta C} = \frac{25\,\text{MHz}}{\delta C} \tag{5}$$

We adopted the PI-Controller architecture of the Linux implementation PTPd [19] as a clock PLL for the fine adjustment of the slave clock. Fig. 5 shows the detailed controller architecture. The PTP algorithm provides the current offset measurement as an input error value to the controller. The output of the controller supplies the counter adjustment value δC to increase or decrease the slave counter speed. This way,

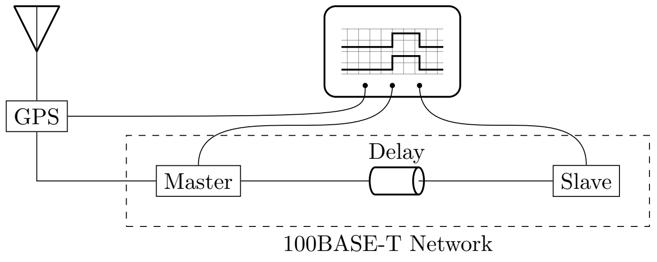


Fig. 6. The measurement setup consists of a GPS receiver that provides a reference PPS signal to the PTP master, which connects to the PTP slave via an Ethernet connection. We measure the offset between the PPS signals of the reference and the master or slave device with an oscilloscope.

offset, and drift are corrected simultaneously by increasing or decreasing the counter speed. The proportional term K_P enables the short-term control of the time offset, and the integral term K_I drives the long-term steady-state error to zero. The selection of the parameters is essential to ensure the local asymptotic stability of the controller to avoid oscillations. For our target update period of 1 s, $K_P = 1.5$ and $K_I = 0.3$ were selected according to the stability regions presented in [20] and verified empirically. We confirmed the results presented in [20] that show a strong dependency of the K_I parameter to K_P, and the update period of the PTP algorithm.

D. Coarse Clock Correction

In the startup phase of the system, the nanosecond counter of the MCU is uninitialized which results in a large clock offset up to 1 s. Therefore, for offset values above 1000 ns, a coarse synchronization of the clock frequency and a coarse offset correction is performed as a starting point for the more accurate fine clock controller. The drift and offset values are calculated based on the timestamps provided by the PTP algorithm. Fig. 3 shows the selection of coarse and fine correction. A counter offset ΔC equal to the current Offset value according to Eq. 3 is added to the hardware counter value. The CPU of the MCU platform has read and write access to the counter value with a non-deterministic latency of multiple nanoseconds. Thus, this mechanism is only suitable to apply a coarse offset correction to the counter.
Our PTP implementation targets a MCU platform without a precision oscillator. Instead, it provides a regular crystal oscillator. Given a common accuracy of 30 ppm, we expect an initial drift up to 30 000 ns over an interval of 1 s. This is significantly larger than the 40 ns increment of the counter. Relying on a clock PLL-based correction of this initial offset and drift causes ringing and a prolonged settling time due to the large input pulse. Instead, we use the measured Drift directly as a coarse correction value δC. As shown in Fig. 4, drift and offset are applied directly as ΔC and δC to the slave counter to perform the initial coarse correction.

E. MCU Master

In order to use the MCU platform as PTP master, synchronization to the reference PPS signal of the GPS is required.

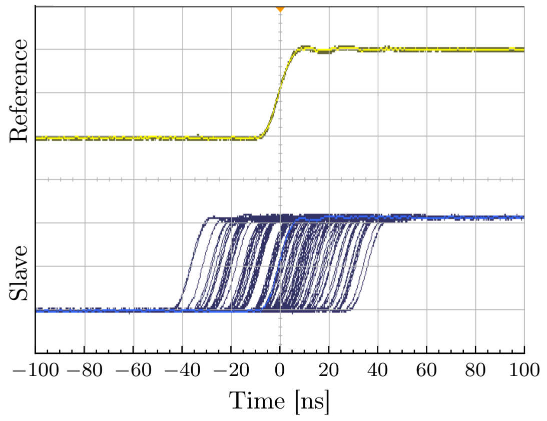


Fig. 7. The oscilloscope acquires 3600 offset measurements between a reference signal (top curve) and a slave device (bottom curves) for each master-slave pair. Data is shown at 20 ns per division.

To achieve this synchronization, we employ the same two-step synchronization algorithm described in the last sections with a fixed $\text{Delay}_{\text{GPS}} = 0$. It can be tuned based on the length and velocity factor of the PPS wire from the GPS in real-world applications. We generate an interrupt on the rising edge of the reference PPS signal and record the timestamp t_2 in hardware to compute the master offset from the reference. t_1 is by definition of the PPS signal equal to the value of the next second.

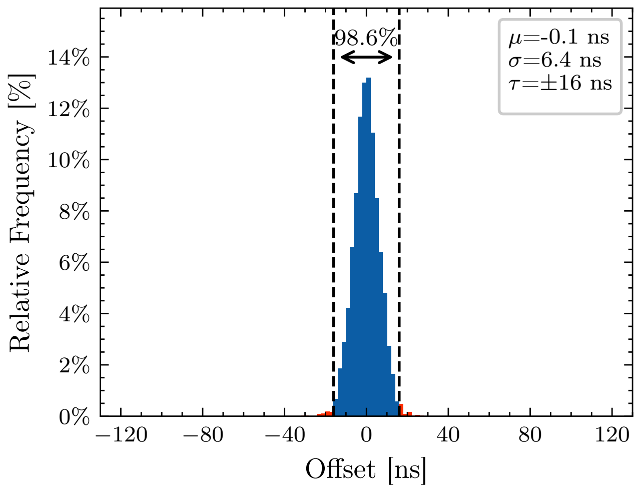
$$\text{Offset}_{\text{Master}} = (t_2 - t_1) - \text{Delay}_{\text{GPS}} \tag{6}$$

III. Evaluation

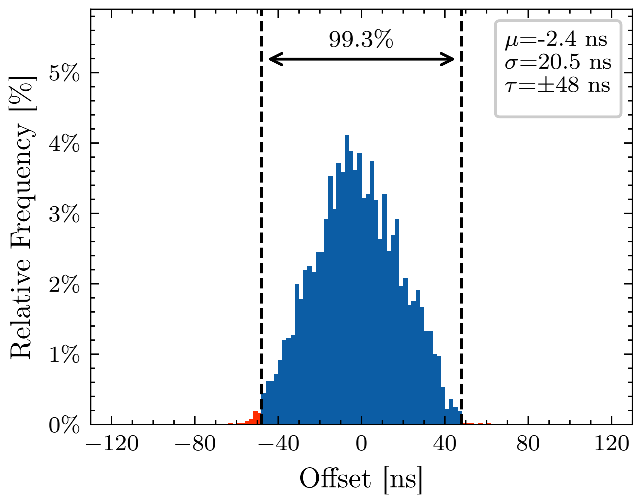
The measurement setup is shown in Fig. 6 and consists of a GPS receiver that provides a reference PPS-Signal to a PTP master. The PTP slave is connected to the master via an 100 Mbit/s Ethernet cable. We evaluate the synchronization performance of our implementation against different master- and slave devices. To isolate the impact of our algorithm, we omitted switches and other network traffic and focused on the raw time synchronization performance. We evaluate the coarse mode of our algorithm separately as it is only relevant in the first seconds after the system start and then focus on the fine clock control, which is primarily used during regular execution. The evaluated fine controller scenarios contain 3600 samples, equivalent to one hour of continuous operation after the initial coarse synchronization phase. The measurement results are plotted as a histogram of the 3600 offset values with a bin size of 2 ns. Every plot shows the mean offset μ, the standard derivation σ, and the expected range τ (defined in the following section) together with the percentage of the measurement values within that range.

A. Measurement Setup

In every measurement, the PPS signal of the GPS reference is automatically compared to the PPS signal of the current slave device. We use an oscilloscope (Keysight DSOX3034T) for the measurements of our evaluation, which measures time-domain offsets between two signals with sub-nanosecond



(a) GPS vs. CM4 slave (via CM4 master)



(b) GPS vs. MCU slave (via CM4 master)

Fig. 8. A CM4 master is connected to the GPS reference PPS signal. Fig. (a) shows the synchronization of a CM4 slave relative to the GPS with an expected offset range of ± 16 ns and Fig. (b) shows the synchronization of an MCU slave with an expected offset of ± 48 ns.

accuracy [21]. We measure the synchronization accuracy based on PPS signals generated by the devices. The PPS signals are generated in hardware on the overflow of the nanosecond counters to mark the start of a second. The increment frequency f_{OSC} of the clock oscillators defines the range τ where the PPS signal is expected to activate.

$$\tau_{\mathrm{MCU}} = \pm\frac{1}{25\,\mathrm{MHz}} = \pm 40\,\mathrm{ns} \tag{7}$$

An example of a scenario with 3600 measurements, and $\tau = \pm 40\,\mathrm{ns}$ is shown in Fig. 7.

B. CM4 Master Scenario

We use two CM4 as a reference baseline for the synchronization performance of two embedded systems. Each board supports timestamping directly in the BCM54210PE PHY with a resolution of 8 ns [22], which is five times more accurate than the 40 ns of the MCU board.

$$\tau_{\mathrm{CM4}} = \pm\frac{1}{125\,\mathrm{MHz}} = \pm 8\,\mathrm{ns} \tag{8}$$

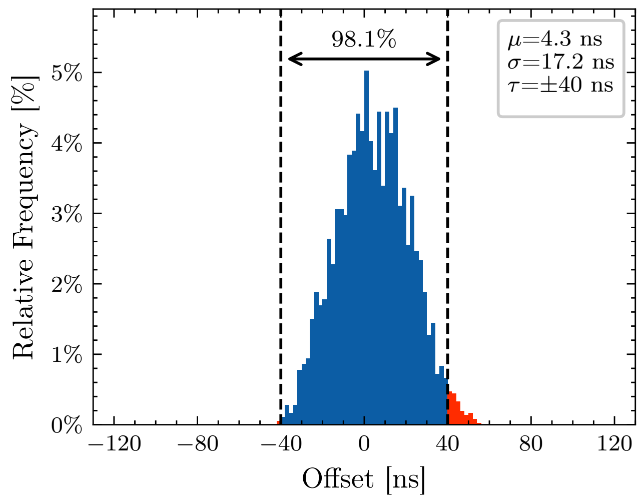
Our evaluation in Fig. 8a shows synchronization of a slave CM4 to a master CM4 with a standard deviation of 6.4 ns. 98.6 % of the measurements lie within the expected range of $2\tau_{\mathrm{CM4}} = \pm 16\,\mathrm{ns}$. After establishing a baseline for comparison, we synchronize a MCU slave to a CM4 master using our slave PTP implementation. The coarse phase of the two-step correction algorithm enables a high-speed initial synchronization. In our evaluation of the MCU performance, typically 3 s are required to achieve the sub-microsecond counter accuracy to start the fine clock PLL. We verified the coarse clock drift to lie within the expected range of 30 000 ns/s. Typically, values of 12 000 ns/s were observed in the coarse adjustment phase. In the fine PLL mode, the offsets between the CM4 master and the MCU slave are expected in a range of $\tau_{\mathrm{CM4}} + \tau_{\mathrm{MCU}} = \pm 48\,\mathrm{ns}$. Fig. 8b shows the PPS signal of the MCU slave relative to the GPS reference signal. The histogram shows a standard deviation of 20.5 ns. 99.3 % of the measurements lie within the expected range.

C. MCU Master Scenario

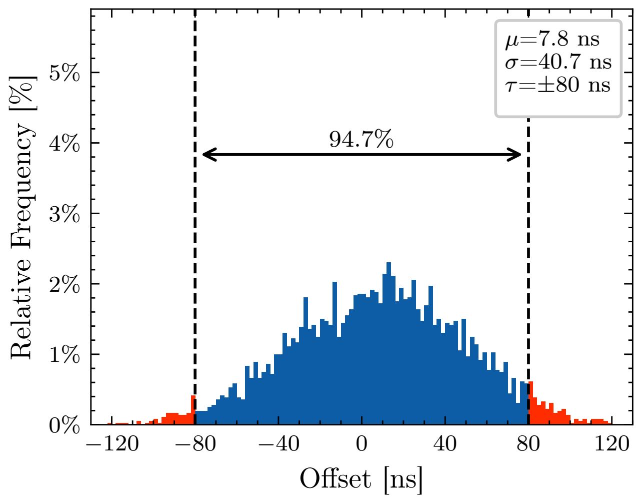
In the measurement scenario shown in Fig. 9, the CM4 master is replaced by a MCU master. In Fig. 9a, the synchronization of the MCU master to the GPS reference signal as described in section II-E is evaluated. As the influence of the network delay is omitted, the synchronization to the reference is achieved with a standard deviation of 17 ns. 98.1 % of the measurements lie within the expected $\pm 40\,\mathrm{ns}$ interval for a single MCU platform.
After synchronizing the MCU to the GPS reference, it is used as a PTP master to distribute the GPS-referenced time in the network. We evaluated the MCU master performance by synchronizing another MCU as a PTP slave. The evaluation results are shown in Fig. 9b. The tolerance ranges of both MCU add up to $2\tau_{\mathrm{MCU}} = \pm 80\,\mathrm{ns}$. 94.7 % of the measurements lie within the expected $\pm 80\,\mathrm{ns}$ range with a standard deviation of 41 ns.

IV. Conclusion

TSN is an essential aspect of modern distributed embedded real-time systems. We implemented the Precision Time Protocol for a bare-metal microcontroller platform and enable synchronization of the MCU system clock to external GPS-time signals, network time masters, and even other MCU systems acting as time masters. Our implementation is based on a two-step synchronization algorithm that combines a coarse drift correction of a regular crystal oscillator with a fine PI controller for precision offset correction on the nanosecond scale. Our implementation and photos of the evaluation setup are available at https://github.com/IMS-AS-LUH/t41-ptp . Our evaluation provides measurements that confirm the expected



(a) GPS vs. MCU master (without slave)



(b) GPS vs. MCU slave (via MCU master)

Fig. 9. A MCU master is connected to the GPS reference PPS signal. Fig. (a) shows the synchronization of the master to the GPS reference with an expected offset range of ± 40 ns and Fig. (b) shows the synchronization of an MCU slave relative to the GPS with an expected offset of ± 48 ns.

range of clock offsets for an embedded system and a MCU based on the implemented hardware counters. We showed that the clock frequency of the hardware counters significantly impacts the achieved synchronization accuracy. The CM4 running at 125 MHz contributes an inaccuracy of 8 ns to the synchronization, and the MCU running at 25 MHz contributes 40 ns of inaccuracy. We showed that a commercial MCU platform can synchronize to an external GPS reference to act as a time master on the network. Other MCU platforms can synchronize their local clocks to the master over the network connection with a standard deviation of 41 ns. Compared to a typical quartz oscillator with a tolerance of 30 ppm, the accuracy is improved by a factor of 730 down to an equivalent accuracy of 41 ppb. The implementation presented in this paper enables precision time synchronization for consumer-grade bare-metal microcontroller systems for novel applications in multimedia, manufacturing, automotive, and science.

References

[1] D. Mill, "Computer network time synchronization: the network time protocol on earth and in space," 2011.

[2] J. Emmett, "Engineering Guidelines: The EBU/AES Digital Audio Interface," *European Broadcasting Union*, 1995.

[3] K. F. Hasan, Y. Feng, and Y.-C. Tian, "GNSS Time Synchronization in Vehicular Ad-Hoc Networks: Benefits and Feasibility," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 12, pp. 3915–3924, 2018.

[4] D. Pannell, "Audio video bridging gen 2 assumptions," https://www.ieee802.org/1/files/public/docs2013/avb-pannell-gen2-assumptions-1113-v17.pdf, 2013.

[5] W. Lewandowski and C. Thomas, "GPS time transfer," *Proceedings of the IEEE*, vol. 79, no. 7, pp. 991–1000, 1991.

[6] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008), 2020.

[7] "IEEE/ISO/IEC International Standard –Part 1AS:Timing and synchronization for time-sensitive applications in bridged local area networks," ISO/IEC/IEEE 8802-1AS:2021(E), 2021.

[8] R. Cochran and C. Marinescu, "Design and implementation of a PTP clock infrastructure for the Linux kernel," in *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 2010, pp. 116–121.

[9] R. Cochran, C. Marinescu, and C. Riesch, "Synchronizing the Linux system time to a PTP hardware clock," in *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 2011, pp. 87–92.

[10] B. Ferencz and T. Kovácsházy, "Hardware assisted COTS IEEE 1588 solution for x86 Linux and its performance evaluation," in *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings*, 2013, pp. 47–52.

[11] A. Wiesner and T. Kovácsházy, "Distributed measurement system for performance evaluation of embedded clock synchronization solutions," in *23rd International Carpathian Control Conference (ICCC)*. IEEE, 2022, pp. 293–298.

[12] Zephyr Project, "Time sensitive networking," https://docs.zephyrproject.org/3.3.0/connectivity/networking/api/tsn.html, 2023.

[13] S. Silverman, "QNEthernet," https://github.com/ssilverman/QNEthernet/tree/ieee1588-2, 2023.

[14] NXP Semiconductors Application Note, "Implementing an IEEE 1588 V2 on i.MX RT Using PTPd, FreeRTOS, and lwIP TCP/IP stack," https://www.nxp.com/docs/en/nxp/application-notes/AN12149.pdf, 2018.

[15] "IEEE Standard for Ethernet (SECTION 6)," IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015), 2018.

[16] Broadcom, "BCM54210 Single Port RGMII SGMII Gigabit Ethernet Transceiver," https://www.broadcom.com/products/ethernet-connectivity/phy-and-poe/copper/gigabit/bcm54210, 2023.

[17] Texas Instruments Inc., "DP83825I Low Power 10/100 Mbps Ethernet Physical Layer Transceiver," https://www.ti.com/product/DP83825I, 2019.

[18] P. Stoffregen, "Teensy 4.1 development board," https://www.pjrc.com/store/teensy41.html, 2023.

[19] K. Correll, "Design considerations for software-only implementations of the IEEE 1588 precision time protocol," in *Proc. 2005 IEEE 1588 Conference, Zurich*, 2005.

[20] R. Maegawa, D. Matsui, Y. Yamasaki, and H. Ohsaki, "A Discrete Model of IEEE 1588-2008 Precision Time Protocol with Clock Servo using PI Controller," in *IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, 2019, pp. 531–536.

[21] Keysight Technologies, "InfiniiVision 3000T X-Series Oscilloscopes Datasheet," https://www.keysight.com/us/en/product/DSOX3034T/oscilloscope-350-mhz-4-analog-channels.html, 2023.

[22] J. Lemon, "Add PTP support for some Broadcom PHYs," https://github.com/raspberrypi/linux/blob/rpi-6.1.y/drivers/net/phy/bcm-phy-ptp.c, 2022.