electronics

MDPI

Article

A Survey on IEEE 1588 Implementation for RISC-V Low-Power Embedded Devices

Alejandro Arteaga *, Leire Muguira , Jaime Jiménez , José Ignacio Gárate and Armando Astarloa Cuéllar

Department of Electronic Technology, Faculty of Engineering, University of the Basque Country UPV/EHU, 48013 Bilbo, Spain; leire.muguira@ehu.eus (L.M.); jaime.jimenez@ehu.eus (J.J.); joseignacio.garate@ehu.es (J.I.G.); armando.astarloa@ehu.eus (A.A.C.)
* Correspondence: aarteaga025@ikasle.ehu.eus

Abstract: IEEE 1588, also known as the Precision Time Protocol (PTP), is a standard protocol for clock synchronization in distributed systems. While it is not architecture-specific, implementing IEEE 1588 on Reduced Instruction Set Computer-V (RISC-V) low-power embedded devices demands considering the system requirements and available resources. This paper explores various approaches and techniques to achieve accurate time synchronization in such instruments. The analysis covers software and hardware implementations, discussing each method's challenges, benefits, and trade-offs. By examining the state-of-the-art in this field, this paper provides valuable insights and guidance for researchers and engineers working on time-critical applications in RISC-V-based embedded systems, aiding in selecting the most-suitable stack for their designs.

Keywords: PTP; IEEE 1588; System-on-Chip; high accuracy; clock synchronization; RISC-V

check for updates

Citation: Arteaga, A.; Muguira, L.; Jiménez, J.; Gárate, J.I.; Cuéllar, A.A. A Survey on IEEE 1588 Implementation for RISC-V Low-Power Embedded Devices. *Electronics* **2024**, *13*, 458. https://doi.org/10.3390/electronics13020458

Academic Editor: Akash Kumar

Received: 28 November 2023
Revised: 9 January 2024
Accepted: 16 January 2024
Published: 22 January 2024

1. Introduction

The Precision Time Protocol (PTP), standardized in IEEE 1588 [1], defines a method for precisely synchronizing the system clocks of different connected devices over a Local Area Network (LAN) or even a sensor one. Employing the PTP standard, it is possible to achieve clock synchronization with sub-microsecond accuracy, allowing real-time data interchange in communication networks.

IEEE 1588-2002 (PTPv1) [2], IEEE 1588-2008 (PTPv2) [3], and its later version, IEEE 1588-2019 (PTPv2.1) [4], have been developed as a hierarchical Master–Slave clock synchronization protocol. These standards can account for delays incurred at the end nodes and delays introduced by other network elements, such as Ethernet switches supporting PTP (boundary or transparent clocks).

PTPv1 established the fundamental framework for achieving sub-microsecond clock synchronization in LAN environments, but it had some limitations, including a lack of support for specific network topologies and scalability issues. PTPv2 addressed these limitations and offered improvements, making it more suitable for complex and diverse network infrastructures. This update brought the concept of boundary and transparent clocks, which enabled more-robust time synchronization across networks with multiple segments and heterogeneous devices.

The latest version of the standard, PTPv2.1, which is currently active, introduces refinements to enhance accuracy, flexibility, and interoperability. PTPv2.1 includes additional features, such as enhanced security mechanisms, support for unicast operation, and optimizations for low-power and resource-constrained devices, making it an attractive option for modern time-critical applications. Table 1 provides a summary of the standard's evolution.

There are several ways of implementing PTP and its stack. The synchronization accuracy varies depending on the chosen implementation method and the placement of timestamps [5]. To achieve optimal accuracy, timestamping for incoming and outgoing messages is ideally performed as close as possible to the physical layer, often accomplished through a hardware-based implementation. However, tasks related to the protocol, such as message processing, data set updates, execution of the Best Master Clock Selection Algorithm (BMC), and the clock control loop, are commonly carried out in software. The combination of hardware and software implementations enables efficient and precise time synchronization for distributed systems, making hardware-assisted PTP an attractive solution for various applications [6].

Table 1. PTP versions.

Version	Year	Standard	Features and Improvements
PTPv1	2002	IEEE 1588-2002	Initial release, basic clock synchronization
PTPv2	2008	IEEE 1588-2008	Enhanced accuracy, boundary and transparent clock support
PTPv2.1	2019	IEEE 1588-2019	Corrections and clarifications to PTPv2

The presented research investigates state-of-the-art solutions to enhance resource-constrained RISC-V System-on-Chip (SoC) synchronization capabilities with high precision. In contemporary applications, accurate timing plays a pivotal role in ensuring the seamless operation and coordination of various systems, including, but not limited to Industrial Automation [7], Power Grid Management [8,9], and Transportation Systems.

The RISC-V architecture provides a range of options for integrating peripheral modules and expanding the capabilities of internal CPUs. Consequently, this study highlights the limitations of current options for integrating PTP into RISC-V systems. This identification of constraints serves as a valuable foundation for the subsequent phases of our research, which are dedicated to proposing and developing a high-precision synchronization capability tailored to RISC-V-based SoCs.

This paper explores the context and reviews the literature concerning the RISC-V architecture (Section 2). It provides insights into the PTP architecture's components (Section 3) and discusses existing general PTP implementations (Section 4). (Section 5) analyzes optimal alternatives for PTP on low-power RISC-V, concluding with future directions in (Section 6).

2. Background Review

The IEEE 1588 standard is not specific to any particular processor architecture. Its implementation on RISC-V low-power embedded devices may vary depending on the system requirements and available resources. It can be categorized according to the implementation type as:

- Software-based implementation: The IEEE 1588 standard can be implemented using software running on Master and Slave devices. However, software-based solutions have limitations in terms of precision and accuracy, especially if the device's hardware lacks support for timestamping or high-resolution timers.
- Hybrid implementations: In some cases, software and hardware assistance can be used to implement IEEE 1588 on low-power RISC-V devices. This approach improves the accuracy and precision of the synchronization process.
- Hardware-based implementation: Certain RISC-V processors or Systems-on-Chips (SoCs) may incorporate a full-stack PTP implementation without relying on any software processing component. The central goal of this strategy is to attain better levels of performance. With this hardware-based approach, the architecture places a more-significant emphasis on accuracy and precision than previous implementations.

The RISC-V architecture is an open-source Instruction Set Architecture (ISA) designed to be modular, extensible, and highly customizable. It is not tied to a single company, making it an attractive choice for tailored systems [10].

The modularity and variability characterize RISC-V. This means the architecture can accommodate various word lengths, allowing for implementations with different bit widths tailored to diverse applications. Additionally, RISC-V adheres to a coherent and sequentially consistent memory model, simplifying programming and code porting across various implementations. Moreover, the architecture offers the possibility of optimization for low power consumption by selecting suitable extensions and configurations [11].

This architecture is a versatile foundation for creating customized processor cores, particularly through RISC-V soft cores on FPGAs. It is an excellent fit for diverse projects like embedded systems, real-time applications, and accelerators.

The central aim of this endeavor is to identify solutions tailored specifically to RISC-V implementations that do not incorporate Memory Management Units (MMUs). These solutions can optimize resource usage and simplify the process, excluding Linux-like systems.

These compact, low-power microprocessors are widespread for applications in the IoT and industrial sectors. The spotlight has been on these small CPUs in academic settings and recent open-source initiatives. In this context, the simplest PULP-based systems are microcontrollers that can be configured to use any supported 32-bit RISC-V core from the PULP platform [12], along with the addition of memory and some peripherals. The mentioned RISC-V-compatible processors are RI5CY, Zero-Riscy, and Micro-Riscy. Furthermore, advanced versions also allow for adding accelerators to the system. This comprises a family of open-source heterogeneous cores ready to be employed in various contexts. Additionally, they are designed to deliver high energy efficiency and low power consumption for battery-powered IoT devices.

CV32E40P is a small and efficient in-order core based on the RISC-V architecture [13]. In 2016, under the name RI5CY, it became a RISC-V core, and in 2020, it started being maintained by the PULP platform, contributing to the "Open Hardware Group". RI5CY was implemented to address energy efficiency in applications deployed on Digital Signal Processors (DSPs).

Next, the need for a simpler and smaller core led to the creation of the Zero-Riscy processor, designed as a straightforward and efficient core. The development of Ibex began in 2015 under the name "Zero-Riscy" as part of the PULP platform for energy-efficient computing. A significant portion of the code was generated by streamlining the RV32 CPU core known as "RI5CY", showcasing the potential for creating an exceptionally compact RISC-V CPU core. In December 2018, lowRISC took over the development of Zero-Riscy and renamed it Ibex [14].

Lastly, Micro-Riscy is a parameterized variation of Zero-Riscy with a minimal area, designed to create the smallest-possible RISC-V core. To achieve this, support for the "E" extension was added under the codename "Micro-Riscy". Within the PULP ecosystem, this core is employed as a control core for PULP, PULPino, and PULPissimo.

One notable solution designed for IoT applications is the PULPino platform [15]. This marked the initial open-source release within the PULP ecosystem and garnered considerable attention. PULPino is a micro RISC-V controller offering a choice between two processors: RI5CY or Zero-Riscy. Its open-source and parameterizable core design make it adaptable to various scenarios.

The PULPissimo platform was developed [16], representing an advanced iteration of the PULPino microcontroller. The critical alteration involves the logarithmic interconnection between the core and the memory subsystem, facilitating the existence of multiple access ports. These ports are subsequently employed by an integrated Micro Direct Memory Access unit (uDMA), capable of directly transferring data between peripherals and memory. Additionally, this platform incorporates optional accelerators known as Hardware Processing Engines (HWPEs).

3. Overview of Precision Time Protocol Architecture and the Protocol Stack

The IEEE 1588 standard is a comprehensive protocol architecture designed to achieve clocks' precise synchronization and syntonization across devices within networked distributed systems, providing accuracy at a sub-microsecond level. Clock syntonization is a fundamental concept within this standard, encompassing harmonizing the frequencies of two independent clocks to ensure they operate at an identical rate. This can be observed in the relationship between Clock 1 and Clock 2 (b), as illustrated in Figure 1. Moreover, clock synchronization pertains to coordinating the current time between these synchronized clocks, ensuring they agree on the same time reference [17]. Clocks operating in perfect harmony can be observed in Figure 1, particularly in the depiction of Clock 1 and Clock 2 in (c).

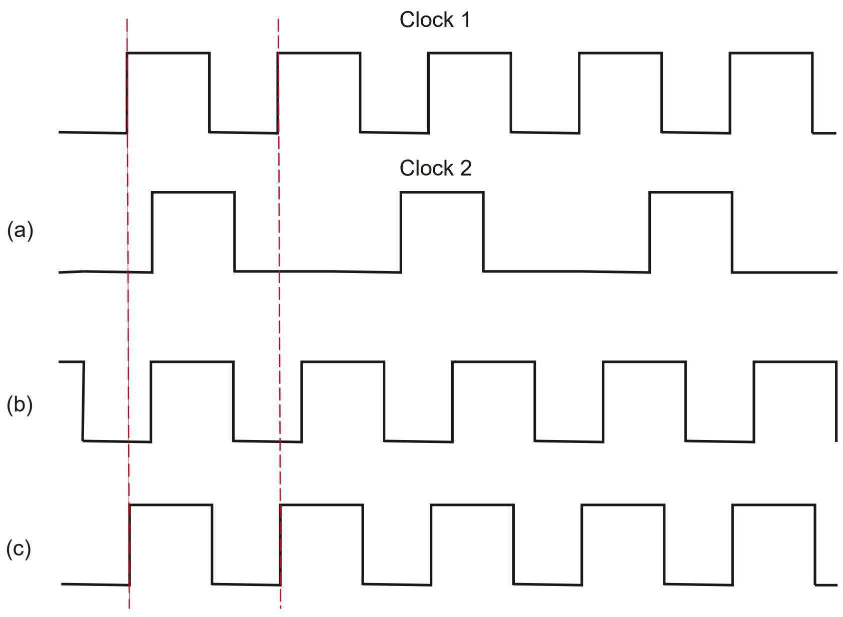

Figure 1. Clock syntonization and synchronization. (**a**) Clock 2 has a different rate than Clock 1. (**b**) Clock 1 and Clock 2 operate at the same rate. (**c**) Clock 1 and Clock 2 are synchronized.

All device clocks are typically synchronized to a Grandmaster clock within the PTP domain. The Grandmaster clock functions as the primary time reference for the entire network, while other clocks, referred to as Slave clocks, synchronize their time with the Grandmaster to attain precise time alignment. This process occurs through PTP message exchanging and capturing the timestamping for transmitted and received packets, allowing the Slave clocks to adjust their local time to match the time of the Grandmaster clock. This continuous exchange and adjustment ensures that all devices in the network maintain accurate and synchronized time.

The protocol distinguishes between two types of messages: Event messages, which facilitate the exchange of timing information between devices, and General messages, which serve various communication purposes within the PTP protocol [17]. Figure 2 is a simplified PTP exchange message to understand synchronization:

- Sync messages: The Master clock sends Sync messages to all Slave clocks with its current time.
- Follow-Up messages: Immediately after sending a Sync message, the Master sends a Follow-Up message. This message carries more-precise timing information related to the transmission of the Sync message. It allows the Slave clocks to calculate the propagation delay more accurately.
- Delay Request (Delay-Req) messages: The Slave clocks send Delay-Req messages to request the round-trip delay time between themselves and the Master clock. These messages contain timestamps representing the time when the message was transmitted.

- Delay Response (Delay-Resp) messages: In response to the Delay-Req messages, the Master clock sends Delay-Resp messages back to the Slave clocks. These messages carry the same timestamps as the received Delay-Req messages. By comparing the timestamps in the Delay-Req and Delay-Resp messages, the Slave clocks can calculate the propagation delay more accurately.

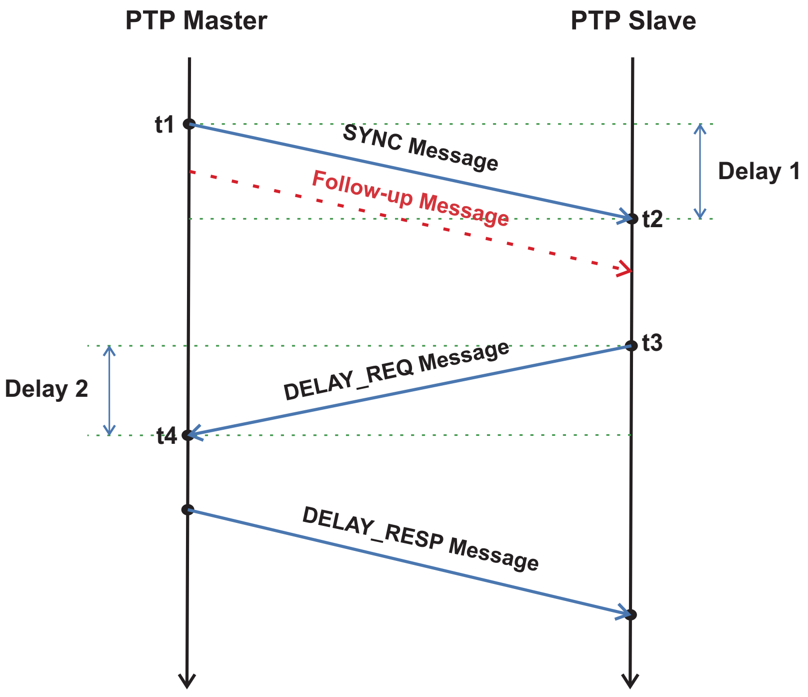

Figure 2. Simplified PTP exchange messages.

At this point, the Slave can calculate the delay and the offset from the Master clock.

$$Delay = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \tag{1}$$

$$Offset = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{2}$$

The standard supports two mechanisms to calculate delay: End-to-End (E2E) and Peer-to-Peer (P2P). In the E2E mechanism, the Slave measures the total delay between itself and the Master, encompassing the entire path. On the other hand, the P2P mechanism requires each device, including switches and routers, on the path between the Master and Slave to measure the delay between itself and its direct neighbor. This distributed measurement approach enables a more-accurate estimation of the propagation delay through the network [17].

The PTP stack is a protocol implementation responsible for managing messages and coordinating time synchronization among devices in a PTP network. Here are some of its key features:

- Message handling: The stack handles the generation and processing of different message types, such as the Sync, Delay-Req, Delay-Resp, Follow-Up, Announce, Signaling, and Management messages.
- Timestamping: The stack captures precise timestamps for transmitted and received PTP messages, which are used to calculate propagation delays and adjust clock frequencies.
- Best Master Clock Algorithm: This software algorithm selects the most-accurate clock source (Grandmaster clock) among the available clocks in the network. The criteria to determine the better clock are described in [18].

- Clock synchronization: Using the information from the PTP messages and timestamps, the stack facilitates the synchronization of Slave clocks with the Grandmaster clock.
- Event notification: The stack may provide event notification capabilities to inform applications or other parts of the system when certain PTP events occur, such as changes in clock status or network conditions.
- Management and configuration: In some cases, the stack may include management and configuration functions, allowing users to customize settings, manage devices, and monitor synchronization performance.

The PTP stack can be implemented in various programming languages and tailored to specific hardware platforms, such as FPGA, ASIC, or general-purpose processors. It allows devices to participate in PTP networks and achieve high-precision time synchronization, making it essential in applications that require accurate timekeeping and coordination.

4. General Approach to Precision Time Protocol Implementation

4.1. Software-Based Precision Time Protocol Implementation

In software-based implementations, time synchronization is managed through the processing power of general-purpose Central Processing Units (CPUs). These implementations rely on software timers and algorithms to compensate for network delays and finely adjust the clock frequencies [19], achieving accurate synchronization across distributed systems.

Software-only solutions can perform timestamping at the application layer or within the Network Interface Card (NIC) driver, as shown in Figure 3. One of the notable advantages of software-based PTP is its compatibility with standard computing platforms, eliminating the need for specialized hardware components. This seamless integration of PTP functionality into existing systems reduces the implementation costs and enhances overall deployment efficiency. However, a disadvantage of software-based PTP is the relative variation of the message delay due to the protocol stack, which can introduce jitter and impact synchronization accuracy.

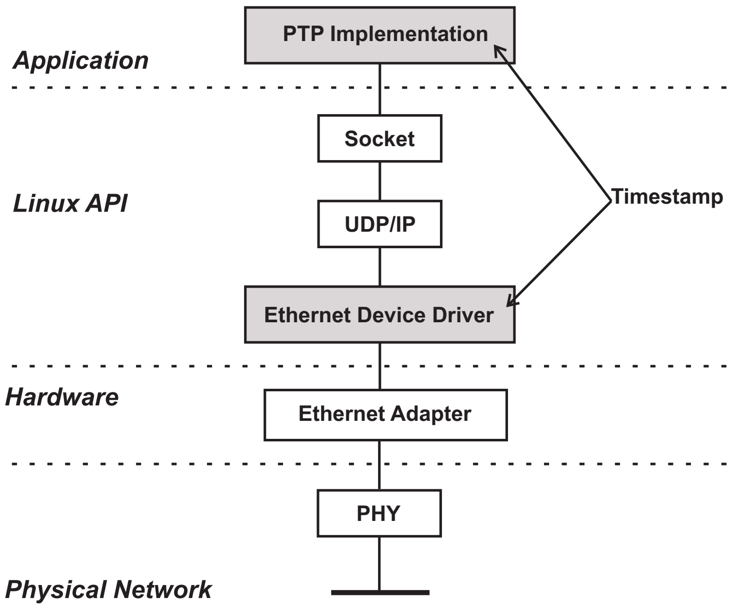

Figure 3. Timestamping in software-based implementations.

The PTP daemon (PTPd) [20] is an open-source software solution initially developed to support the PTPv1 standard. Its first release in 2005 was tailored to Unix-based operating systems. Over time, PTPd has evolved substantially, incorporating improved protocol stack versions. By 2010, it had seamlessly integrated support for the PTPv2 standard.

Another notable solution for PTP software-based implementations is the LinuxPTP project [21,22]. This open-source initiative, originally engineered to endorse PTPv2, has continuously adapted to address the evolving requirements of time-sensitive applications. This adaptability has solidified its reputation as an invaluable tool for achieving highly accurate time synchronization within Linux-based systems.

These solutions have gained widespread popularity in the industry. These well-supported implementations offer reliable and high-performance solutions for time synchronization tasks. The open-source nature of these software stacks allows continuous improvements, bug fixes, and feature enhancements. These software stacks can operate as software-only solutions or with hardware timestamps. In cases where software support is not feasible, they work in software-only mode.

On the other hand, Domain Time II [23] represents another software solution for achieving precise time synchronization across an entire network infrastructure. Its versatility includes support for PTPv2 and PTPv2.1, empowering organizations to achieve unmatched time accuracy and consistency across interconnected devices. It is worth noting that it is offered under a commercial license.

It is essential to consider that software-based PTP implementations may face limitations regarding latency and determinism [24,25]. In time-critical applications, the synchronization accuracy can be adversely affected by jitters caused by the inherent variability in software processing times and potential delays introduced by operating systems and network stacks [26].

To mitigate these challenges, ongoing research optimizes software-based PTP implementations, reduces latency [27], and enhances determinism. In [28], a PTP clock architecture implementation for the Linux Kernel to improve synchronization time was shown.

4.2. Hybrid Precision Time Protocol Implementation

A hybrid PTP implementation consists of a combination of software-based and hardware-based approaches to achieve the IEEE 1588 protocol. In these implementations, specific tasks are handled by the software part, and the PTP stack can be run on both operating systems and embedded CPUs. Conversely, critical tasks such as timestamping are offloaded to specialized hardware components. Some implementations utilize the Physical Layer Interface (PHY) to implement this [29], while others use Ethernet MAC IPs for this purpose [30], as shown in Figure 4. This approach aims at leveraging the advantages of both approaches to optimize performance, accuracy, and efficiency in real-time applications.

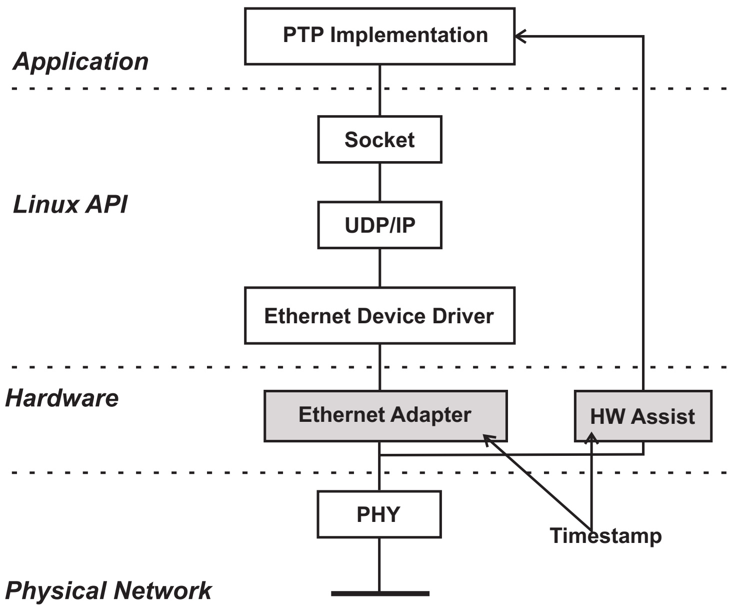

Figure 4. Timestamping in hybrid implementations.

Hardware assisting is a crucial element in achieving shorter synchronization times. Specialized hardware components, such as hardware timestamping counters in NICs, capture the precise time when PTP messages are received and transmitted at the hardware level. Hardware significantly reduces message processing latency by bypassing software processing, providing a more-accurate measurement of message propagation delays. In some hybrid implementations [29], the clock control loop, responsible for adjusting the local clock's frequency and phase, is offloaded to hardware. This hardware-based clock control loop offers better determinism and lower jitter, enhancing the overall precision of time synchronization.

While hardware handles timestamping and the clock control loop, the message processing and PTP stack functions remain in the software. The software-based message processing manages the exchange and handling of PTP messages.

In [31], an example of the PTP protocol implementation using a mixed solution was presented. This design utilizes the freeRTOS operating system running on a Xilinx platform and incorporates PTPd for specific functionalities. In this solution, the software-based components in the application layer are responsible for handling specific PTP tasks, including the BMC algorithm and packet processing. Meanwhile, the hardware layer complements the software implementation with dedicated timestamping and real-time clock management modules. Figure 5 shows how synchronization times improve when using hardware assistance.

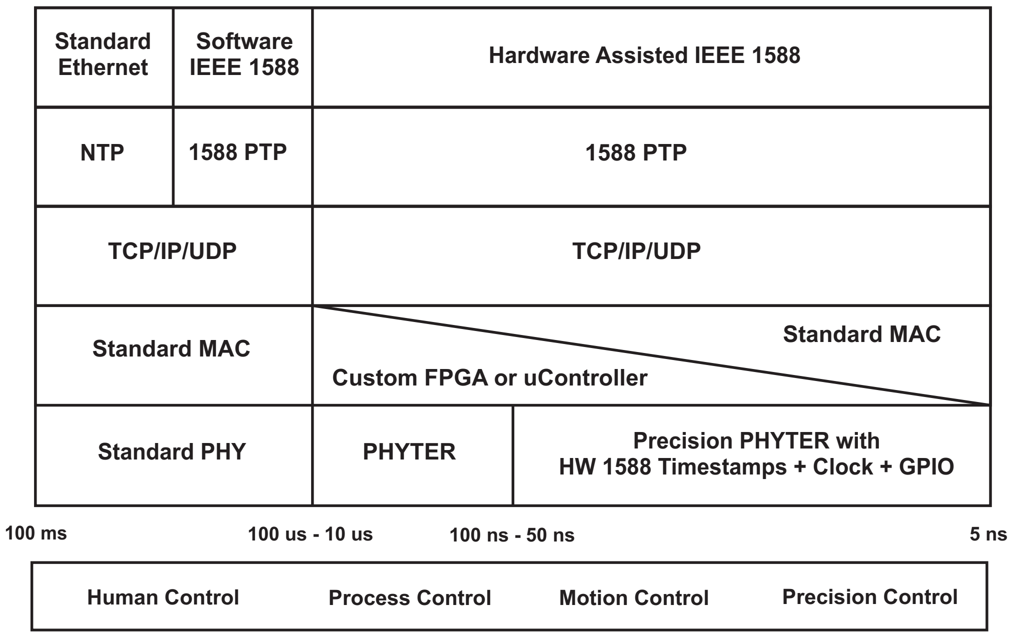

Figure 5. Implementation choices to achieve better time synchronization [32].

Numerous research papers have documented the implementation of the PTP protocol using hybrid-based approaches. In [33], an IEEE 1588 prototype on wireless LAN was implemented using an Altera Excalibur embedded development board. This board includes an ARM9 processor and a Programmable Logic Device (PLD).

Another example of a hardware-assisted implementation can be found in [34]. The Nios microprocessor is employed to implement the PTP protocol stack using PTPd without reliance on an operating system or network API. This approach showcases the direct integration of PTPd within the Nios microprocessor architecture, enabling PTP functionality to be executed without the overhead of an operating system. The absence of an operating system and network API simplifies the design, reducing potential latency and enhancing the efficiency of the PTP implementation.

Furthermore, Oregano Systems [35] provides a PTP stack solution known as "syn1588", designed as a user space application capable of running on standard PCs and various microcontrollers. A comprehensive PTP node implementation utilizing sys1588 was introduced in [36].

Alternative approaches to clock synchronization have been developed, with notable examples including the White Rabbit (WR) project [37]. The WR project achieves exceptional synchronization accuracy, boasting synchronization times in the sub-nanosecond range. This achievement is realized by extending the capabilities of the PTP protocol.

In the context of the WR project, a significant contribution is the development of a PTP daemon named PTP Ported Silicon (PPSi) [38]. PPSi is meticulously designed for streamlined portability across diverse hardware architectures and exhibits a high degree of modularity to facilitate the incorporation of protocol extensions. Notably, the foundational protocol code of PPSi remains consistent across all supported architectures. These range from conventional Linux PCs to specialized softcore processors operating within FPGA environments.

The flexibility and adaptability of PPSi across various hardware configurations offer a promising avenue for achieving hardware-assisted PTP synchronization.

Hybrid solutions are commonly adopted by companies that develop and market IP modules for FPGAs and Application-Specific Integrated Circuits (ASICs). By providing hardware-assisted PTP IP cores, these companies enable customers to implement precise time synchronization designs more efficiently and accurately. For instance, in the work presented by [39], the authors shared key insights into implementing IEEE 1588 using an Intel (Altera) FPGA IP core.

In [40], an open-core IP solution was presented, where a PTP software stack was implemented using PTPd. This IP core includes the implementation of a Real-Time Clock (RTC) and a Timestamping Unit (TSU).

Another notable solution employing the hybrid approach is offered by the System-on-Chip engineering (SoC-e) company [41], which has developed solutions based on IP modules over FPGA devices. They offer both IP cores for customers to integrate into their designs and finished products that utilize the same technology.

By leveraging a hardware-assisted PTP implementation, these companies address the challenges of achieving precise time synchronization in real-world applications. Their solutions can significantly improve synchronization accuracy, reduce processing latency, and enhance the overall performance of time-sensitive systems.

4.3. Hardware-Based Precision Time Protocol Implementation

A hardware-based PTP implementation relies on dedicated hardware components to achieve precise time synchronization and coordination according to standards. In this approach, all essential PTP tasks are exclusively executed by specialized hardware modules, as illustrated in Figure 6, without the involvement of any software components. Timestamping is typically implemented on the PHY or Ethernet MAC IPs.

Using only hardware implementations can achieve extremely high precision and low latency in time synchronization. This makes them well-suited for applications that require sub-microsecond or nanosecond accuracy.

These solutions offer deterministic and predictable performance, eliminating the variability introduced by software-based processing and operating systems. Using FPGA or ASIC devices, hardware-only solutions can be customized to match specific application requirements. This allows for tailored PTP functionality and adaptability to various network setups.

However, it also has some limitations. It may require more-specialized hardware expertise, and the initial development costs may be higher than off-the-shelf software solutions. Additionally, it may have limited flexibility for future protocol updates or changes, as they rely solely on the capabilities of the hardware modules.

As an illustrative example of a hardware-based PTP implementation, "1588tiny", a Slave node, was introduced in [42]. Remarkably, this solution operates entirely in hardware without needing a local or embedded CPU to execute the PTP stack. This approach unquestionably outshines architectures that rely on a CPU regarding resource efficiency. However, it is important to note that 1588tiny has certain limitations, such as being limited to Slave-only mode. Despite these limitations, its hardware-focused design contributes to its exceptional resource utilization efficiency.

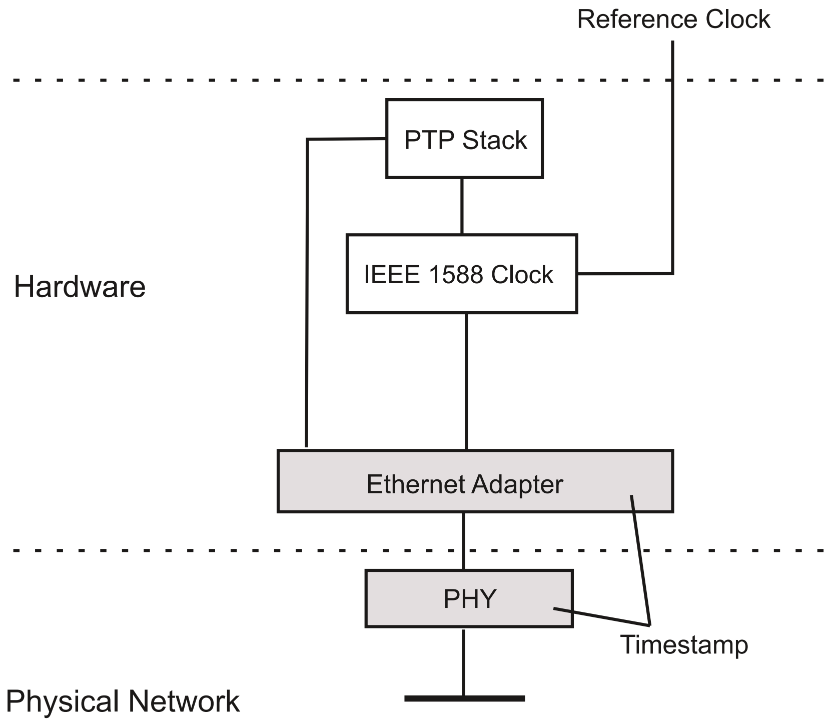

Figure 6. Timestamping in hardware-based implementations.

Another full-hardware PTP implementation was presented in [43]. In this work, a hardware-centric approach was taken to implement a switch with redundancy, and the core functions of the PTP stack, including the BMC and management functions, were directly realized using VHDL. Embedding the PTP stack directly into the hardware allows for streamlined and low-latency time synchronization within the Ethernet switch.

Another advantage of hardware-only implementations is their portability. Integrating PTP functionality into a CPU stack often demands extensive modifications and additions to the existing software stack. On the other hand, incorporating PTP IP into a hardware solution typically necessitates only configuration adjustments and establishing interface signal connections within the FPGA. This streamlined process enables more-accessible adaptation and integration of PTP capabilities across different hardware platforms, reducing the complexity associated with software modifications.

5. Analysis of Alternatives for Reduced Instruction Set Computer-V Architecture

Alternatives to PTP Implementations in the context of RISC-V provide a comprehensive examination of the various options available when considering the integration of PTP protocols within the architecture. This analysis compares different PTP implementation strategies for RISC-V, evaluating simplicity, efficiency, portability, community support, power consumption, latency considerations, and development effort. By exploring these alternatives, this analysis equips decision-makers with the insights to select the most-suitable PTP implementation approach that aligns with their project's objectives and RISC-V architecture considerations.

When implementing a solution on the RISC-V architecture, it is essential to consider resource availability and whether to opt for an implementation with or without a Memory Management Unit (MMU), depending on the requirement to run an OS. It is noteworthy that Linux can be tailored for deployment on RISC-V, even in scenarios where an MMU is absent. An illustrative example is the Buildroot environment, configuring QEMU-emulated machines with 32-bit RISC-V without an MMU [44]. The decision to opt for an MMU-less configuration should be guided by a meticulous assessment of specific requirements, performance goals, and resource constraints.

However, noteworthy attempts have been made to port PTPd to smaller OS options such as Iwip-ptpd [45]. Additionally, the Zephyr project [46] supports PTP, showcasing alternative solutions for achieving precise time synchronization in environments with specific OS constraints.

Table 2 displays a selection of the previously discussed implementations that are potential alternatives for implementing the PTP stack in RISC-V. Solutions such as Linux-PTP [21], Domain Time II [23], FreeRTOS-Xilinx Zynq [31], and ha1588 [40] are OS-based and in particular require an MMU and significant resources for implementation.

Table 2. Comparison between PTP implementations.

PTP Implementation	Stack PTP	Supported PTP Version	Supported Modes	License	OS need	Programming Languages	Implementation Type
LinuxPTP project [21]	PTPd, PTP clock API	PTPV2	Master, Slave	Open-source	Yes	C	Software
PTPd project [20]	PTPd	PTPV1, PTPV2	Master, Slave	Open-source	Yes	C	Software
Domain Time II [23]	Proprietary	PTPv2, PTPv2.1	Master, Slave	Proprietary	Yes	C	Software
FreeRTOS-Xilinx Zynq [31]	PTPd	PTPV2	Master, Slave	Proprietary	Yes	VHDL, C	Hybrid
PTP with Nios Processor [34]	PTPd	PTPV1	Master, Slave	Proprietary	No	VHDL, C	Hybrid
Syn1588 (Oregano Systems) [35]	Proprietary	PTPV2.1	Master, Slave	Proprietary	No	VHDL, C/C++	Hybrid
ha1588 (open-core) [40]	PTPd	PTPV1, PTPV2	Master, Slave	Open-source	Yes	Verilog, C	Hybrid
1588Tiny (SoC-e) [42]	Proprietary	PTPV2	Slave	Proprietary	No	Verilog/VHDL	Hardware
White Rabbit project [37]	PTPd, PPSi	PTPV2, PTPV2.1	Master, Slave	Open-source	No	VHDL, C/C++, Phyton	Hybrid

On the other hand, solutions like PTP with the Nios Processor [34], Syn1588 by Oregano Systems [35], and 1588Tiny by SoC-e [42] distinguish themselves by not necessitating an operating system or MMU implementation. Instead, they rely on proprietary software and hardware components to fulfill their PTP functionalities. While these approaches offer distinct advantages in specific scenarios, they might present challenges in understanding their development process and reusability.

The White Rabbit project is another compelling option. It operates under an open license, does not require an MMU implementation, and can implement the IEEE 1588 stack based on open-source PTPd and PPSi daemons. Its versatility makes it particularly attractive for projects prioritizing adaptability and open-source components.

Based on this latest implementation approach, it would be prudent to analyze these two PTP daemons to understand them better and determine which is better for implementation in an RISC-V project. Table 3 provides a feature comparison between PTPd and PPSi.

PTPd is primarily designed to run on systems with an MMU. It relies on dynamic memory allocation, a feature typically facilitated by an MMU. Although adapting PTPd for use on systems without an MMU is possible, it may require significant modifications and adjustments to the codebase to replace dynamic memory allocation with statically allocated memory.

Additionally, PPSi was designed to be more lightweight and suitable for embedded systems, including those without an MMU. PPSi minimizes resource usage and offers high portability across different platforms, making it a strong choice for RISC-V implementations with resource constraints.

Table 3. Comparison between PTPd and PPSi.

Features	PTPd	PPSi
Implementation	Well-suited for complex network setups.	Ideal for simpler network scenarios.
Flexibility	Highly flexible.	Basic functionality.
Adaptable to RISC-V platforms	Yes	Yes
Portability	May require adjustments for different architectures.	Designed for easy portability across architectures.
Latency considerations	May introduce slightly higher latency.	Designed to minimize latency and jitter.
Power consumption	May consume more power due to advanced features.	Optimized for reduced power usage.
Resource constraints	May consume more resources than optimized solutions.	Ideal for platforms with limited resources.
Community support	Strong community presence with active contributors.	Smaller community.
Supported PTP versions	PTPV1, PTPV2	PTPV2
Programming languages	C	C, Python
Dependency on MMU-based OS	Optimized for MMU-based OSs, such as Linux.	Can be integrated into both MMU-based and non-MMU-based OSs.
Memory management	Utilizes dynamic memory allocation, which may be facilitated by an MMU.	Primarily uses statically allocated memory.
Suitable use cases	Well-suited for applications where precise time synchronization is required and hardware with an MMU is available.	Ideal for applications with strict memory constraints, such as embedded systems, IoT devices, and hardware platforms that lack an MMU.

When comparing power consumption between them, it is important to consider that PTPd offers more features, while PPSi is designed to be more resource-efficient and, consequently, more power-efficient. However, power consumption may vary based on the specific implementation and hardware used.

Regarding latency considerations, it is essential to analyze the performance and responsiveness of these implementations, as these factors are critical in time-sensitive applications. PTPd may introduce slightly higher latency than PPSi due to the design features.

In conclusion, the decision between PTPd and PPSi for RISC-V architectures should be guided by specific project requirements and constraints. PTPd offers many features and benefits alongside strong community support, while PPSi emphasizes resource efficiency and adaptability. However, considering crucial factors such as latency, power consumption, resource limitations, portability, and the lack of MMU dependency, PPSi is the better choice for RISC-V implementation, especially in scenarios with limited resources.

6. Conclusions

This paper addresses various aspects of implementing IEEE 1588 within the context of low-power embedded devices based on the RISC-V architecture. We underscored the significance of PTP in achieving precise time synchronization in distributed systems, emphasizing its capability to synchronize system clocks with sub-microsecond accuracy, thereby enabling real-time data exchange in communication networks.

Moreover, different RISC-V devices' time synchronization approaches have been explored, each offering advantages and trade-offs. This allows developers to choose the most-suitable method based on specific project requirements.

The presentation of an evaluation criteria framework has further contributed to informed decision-making, considering factors such as simplicity, efficiency, portability, community support, power consumption, latency, and development effort in selecting a PTP stack implementation approach tailored to the RISC-V architecture.

Looking ahead, our research will address the constraints associated with existing options for integrating the Precision Time Protocol into RISC-V systems. Leveraging the inherent flexibility of the RISC-V ISA and emerging SoC architectures, we will focus on formulating and implementing a finely tuned precision synchronization capability for RISC-V-based SoCs. These efforts cater to targeted applications in sectors such as Industrial Automation, Power Grid Management, and Transportation Systems.

Author Contributions: Conceptualization, A.A. and A.A.C.; methodology, A.A.C.; formal analysis, A.A. and A.A.C.; investigation, A.A.; writing—original draft preparation, A.A.; writing—review and editing, A.A., A.A.C., L.M., J.I.G. and J.J.; supervision, A.A.C., L.M., J.I.G. and J.J.; project administration, A.A.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the ECSEL Joint Undertaking in the H2020 project IMOCO4.E, grant agreement No.10100731, and by the Basque Government within the fund for research groups of the Basque University System IT1440-22 and KK-2023/00015.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: The documents and software libraries foundational to this work have been made accessible online. Interested parties can find them for reference and further exploration.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the study's design, in the collection, analyses, or interpretation of the data, in the writing of the manuscript, nor in the decision to publish the results.

References

1. *IEEE 1588;* IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. IEEE: New York, NY, USA. Available online: https://standards.ieee.org (accessed on 23 September 2023).
2. *IEEE Std 1588-2002;* IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. IEEE: New York, NY, USA, 2002; pp. 1–154. [CrossRef]
3. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002);* IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. IEEE: New York, NY, USA, 2008; pp. 1–269. [CrossRef]
4. *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008);* IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. IEEE: New York, NY, USA, 2019; pp. 1–499. [CrossRef]
5. Lu, L.; Zhang, C.; Liu, Y.; Zhang, W.; Xia, Y. IEEE 1588-Based General and Precise Time Synchronization Method for Multiple Sensors. In Proceedings of the 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO), Dali, China, 6–8 December 2019; pp. 2427–2432. Available online: https://ieeexplore.ieee.org/document/8961658 (accessed on 23 September 2023).
6. Idrees, Z.; Granados, J.; Sun, Y.; Latif, S.; Gong, L.; Zou, Z.; Zheng, L. IEEE 1588 for Clock Synchronization in Industrial IoT and Related Applications: A Review on Contributing Technologies, Protocols and Enhancement Methodologies. *IEEE Access* **2020**, *8*, 155660–155678. Available online: https://ieeexplore.ieee.org/abstract/document/9154372 (accessed on 5 November 2023). [CrossRef]
7. Lan, Y.K.; Chen, Y.S.; Hou, T.C.; Wu, B.L.; Chu, Y.S. Development Board Implementation and Chip Design of IEEE 1588 Clock Synchronization System Applied to Computer Networking. *Electronics* **2023**, *12*, 2166. Available online: https://www.mdpi.com/2079-9292/12/10/2166 (accessed on 16 July 2023). [CrossRef]
8. Khan, M.; Hayes, B. IEEE 1588 Time Synchronization in Power Distribution System Applications: Timestamping and Accuracy Requirements. *IEEE Syst. J.* **2023**, *17*, 2007–2017. [CrossRef]
9. Akbarzadeh, A.; Erdődi, L.; Houmb, S.; Soltvedt, T.; Muggerud, H. Attacking IEC 61850 Substations by Targeting the PTP Protocol. *Electronics* **2023**, *12*, 2596. [CrossRef]
10. Cui, E.; Li, T.; Wei, Q. RISC-V Instruction Set Architecture Extensions: A Survey. *IEEE Access* **2023**, *11*, 24696–24711. [CrossRef]
11. Yu, H.; Yuan, G.; Kong, D.; Chen, C. An Optimized Implementation of Activation Instruction Based on RISC-V. *Electronics* **2023**, *12*, 1986. Available online: https://www.mdpi.com/2079-9292/12/9/1986 (accessed on 16 August 2023). [CrossRef]
12. PULP Implementation. Available online: https://pulp-platform.org/implementation.html (accessed on 22 August 2023).

13. Schiavone, D. Open Source Hardware the New Reality CV32E40P Project | OpenHW Group. Available online: https://www.openhwgroup.org/resources/blog/open-source-hardware-the-new-reality-cv32e40p-project/ (accessed on 22 August 2023).
14. Davide Schiavone, P.; Conti, F.; Rossi, D.; Gautschi, M.; Pullini, A.; Flamand, E.; Benini, L. Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications. In Proceedings of the 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), Thessaloniki, Greece, 25–27 September 2017; pp. 1–8. [CrossRef]
15. Traber, A.; Gautschi, M. "Pulpino_Datasheet". Available online: https://pulp-platform.org/docs/pulpino_datasheet.pdf (accessed on 22 August 2023).
16. Pulpissimo/Doc/Datasheet/Datasheet.pdf at Master·Pulp-Platform/Pulpissimo. Available online: https://github.com/pulp-platform/pulpissimo/blob/master/doc/datasheet/datasheet.pdf (accessed on 22 August 2023).
17. Waldhauser, S.; Jaeger, B.; Helm, M. Time Synchronization in Time-Sensitive Networking. Volume 51, pp. 51–56. Available online: https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2020-04-1/NET-2020-04-1_10.pdf (accessed on 1 August 2023).
18. Arnold, D. What Makes a Master the Best? Available online: https://blog.meinbergglobal.com/2013/11/14/makes-master-best/ (accessed on 1 August 2023).
19. Benetazzo, L.; Narduzzi, C.; Stellini, M. Analysis of Clock Tracking Performances for a Software-only IEEE 1588 Implementation. In Proceedings of the 2007 IEEE Instrumentation & Measurement Technology Conference IMTC 2007, Warsaw, Poland, 1–3 May 2007; pp. 1–6, ISSN 1091-5281. [CrossRef]
20. PTPd. Available online: https://github.com/ptpd/ptpd (accessed on 27 July 2023).
21. Introduction. Available online: https://github.com/nwtime/linuxptp (accessed on 28 July 2023).
22. Machnikowski, M.; Reddy, R.; Fodor, Z. Challenges with linuxptp on Telco RAN deployments. In Proceedings of the 2021 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS), NA, FL, USA, 27–28 October 2021; pp. 1–4, ISSN 1949-0313. [CrossRef]
23. Domain Time II Features. Available online: https://www.greyware.com/software/domaintime/v5/overview/features.asp (accessed on 17 October 2023).
24. Correll, K.; Barendt, N. Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol. In Proceedings of the Conference on IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, Winterhur, Switzerland, 10–12 October 2005; IEEE: New York, NY, USA, 2006.
25. Chen, P.; Yang, Z. Understanding Precision Time Protocol in Today's Wi-Fi Networks: A Measurement Study. pp. 597–610. Available online: https://www.usenix.org/conference/atc21/presentation/chen (accessed on 11 October 2023).
26. Kovácsházy, T.; Ferencz, B. Performance evaluation of PTPd, a IEEE 1588 implementation, on the x86 Linux platform for typical application scenarios. In Proceedings of the 2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings, Graz, Austria, 13–16 May 2012; pp. 2548–2552, ISSN 1091-5281. [CrossRef]
27. Luckinger, F.; Sauter, T. Software-Based AUTOSAR-Compliant Precision Clock Synchronization Over CAN. *IEEE Trans. Ind. Inform.* **2022**, *18*, 7341–7350. [CrossRef]
28. Cochran, R.; Marinescu, C. Design and implementation of a PTP clock infrastructure for the Linux kernel. In Proceedings of the Control and Communication 2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Portsmouth, NH, USA, 25 October 2010; pp. 116–121, ISSN 1949-0313. [CrossRef]
29. Zhao, B.; Wang, N. The Implementation of IEEE 1588 Clock Synchronization System Based on FPGA. In Proceedings of the Fifth International Conference on Intelligent Control and Information Processing, Dalian, China, 18–20 August 2014; pp. 216–220. Available online: http://ieeexplore.ieee.org/document/7010342/ (accessed on 31 July 2023).
30. Dong, M.; Qiu, Z.; Pan, W.; Chen, C.; Zhang, J.; Zhang, D. The Design and Implementation of IEEE 1588v2 Clock Synchronization System by Generating Hardware Timestamps in MAC Layer. In Proceedings of the 2018 International Conference on Computer, Information and Telecommunication Systems (CITS), Alsace, Colmar, France, 11–13 July 2018; p. 5. [CrossRef]
31. Pandey, P.; Pratap, B.; Pandey, R.S. Implementation of FreeRTOS based Precision Time Protocol (PTP) application as per IEEE1588v2 standards for Xilinx Zynq UltraScale Plus MPSoC devices. In Proceedings of the 2019 International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 17–19 July 2019; pp. 1968–1973. [CrossRef]
32. AN-1728 IEEE 1588 Precision Time Protocol Time Synchronization Performance. Available online: https://www.semanticscholar.org/paper/AN-1728-IEEE-1588-Precision-Time-Protocol-Time/e4a7159e97c06bfceff48bf4952b0a3ff188cb06 (accessed on 31 July 2023).
33. Kannisto, J.; Vanhatupa, T.; Hannikainen, M.; Hamalainen, T. Software and hardware prototypes of the IEEE 1588 precision time protocol on wireless LAN. In Proceedings of the 2005 14th IEEE Workshop on Local & Metropolitan Area Networks, Crete, Greece, 18 September 2005; p. 6, ISSN 1944-0375. [CrossRef]
34. PTP Version 1 Implementation on FPGA with NIOS d FPGA with NIOS Processor an d Gigabit MAC IP—Buscar Con Google. Available online: https://indico.cern.ch/event/28233/contributions/1631190/attachments/519246/716383/IN2P3_PTP_implementation.pdf (accessed on 16 August 2023).
35. syn1588® PTP Stack | Oregano Systems. Available online: https://www.oreganosystems.at/products/syn1588/software/syn1588r-ptp-stack (accessed on 2 August 2023).

36. Müller, T.; Kerö, N. A fully integrated versatile PTP node. In Proceedings of the 2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings, San Francisco, CA, USA, 24–28 September 2012; pp. 1–6, ISSN 1949-0313. [CrossRef]
37. White Rabbit Official CERN Website. Available online: https://white-rabbit.web.cern.ch/ (accessed on 2 August 2023).
38. Projects/PPSi. Available online: https://ohwr.org/project/ppsi (accessed on 2 August 2023).
39. Mallela, C.; Choo, Y.Y.; Bridgers, V. Building a 1588 system solution—Key learnings. In Proceedings of the 2016 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS), Stockholm, Sweden, 4–9 September 2016; pp. 1–6, ISSN 1949-0313. [CrossRef]
40. Overview: Hardware Assisted IEEE 1588 IP Core—OpenCores. Available online: https://opencores.org/projects/ha1588 (accessed on 29 August 2023).
41. PTP—Precision Time Protocol. Available online: https://soc-e.com/ptp-precision-time-protocol/ (accessed on 5 August 2023).
42. 1588 Tiny IP Core: IEEE1588v2 CPU-Less Slave Clock. Available online: https://soc-e.com/products/1588tiny-ieee-1588-v2-slave-only-hard-ip-core/ (accessed on 29 August 2023).
43. Kirrmann, H.; Honegger, C.; Ilie, D.; Sotiropoulos, I. Performance of a full-hardware PTP implementation for an IEC 62439-3 redundant IEC 61850 substation automation network. In Proceedings of the 2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings, San Francisco, CA, USA, 24–28 September 2012; pp. 1–6, ISSN 1949-0313. [CrossRef]
44. Embedded Linux Conference Europe 2020 (ELC-E). Available online: https://www.elinux.org/images/3/3d/Linux_riscv_elec2020.pdf (accessed on 13 September 2023).
45. Bennion-Pedley, J. lwip-ptp, 2023. Available online: https://github.com/BOJIT/ptpd-lwip (accessed on 14 December 2023).
46. Zephyrproject-Rtos/Zephyr, 2023. Available online: https://github.com/zephyrproject-rtos/zephyr (accessed on 14 December 2023).