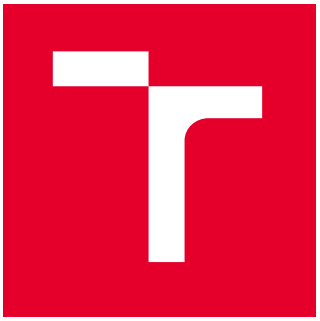


BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

COMMUNICATING DEVICES SYNCHRONIZED USING THE IEEE 1588 STANDARD

KOMUNIKUJÍCÍ ZAŘÍZENÍ SYNCHRONIZOVANÉ POMOCÍ STANDARDU IEEE 1588

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — **Ilya Voloshin**
AUTOR PRÁCE

SUPERVISOR — **Ing. František Rusnák**
VEDOUCÍ PRÁCE

BRNO 2026

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Bachelor's Thesis

Bachelor's study program **Automation and Measurement**

Department of Control and Instrumentation

Student: Ilya Voloshin ***ID:*** 247494

Year of study: 3 ***Academic year:*** 2025/26

TITLE OF THESIS:

Communicating devices synchronized using the IEEE 1588 standard

INSTRUCTION:

The aim of the thesis is to design software that will enable communication between available STM Nucleo development platforms. Individual platforms will communicate with each other using Ethernet. Communication using the PTP protocol is essential, which will allow devices to synchronize time with each other. The thesis will theoretically evaluate the possibilities of using Single Pair Ethernet for time synchronization.

1. Get to know the STM Nucleo platform and study the hardware and software options.
2. Get to know the PTP protocol.
3. Conduct research and possibilities of using Single Pair Ethernet.
4. Verify the functionality of available communication libraries for the STM Nucleo platform.
5. Design software that will implement time synchronization using PTP messages.
6. Implement the software on the platform.
7. Evaluate the results of time synchronization using Ethernet.

RECOMMENDED LITERATURE:

A. Arteaga, L. Muguira, J. Jiménez, J. I. Gárate, and A. A. Cuéllar, “A Survey on IEEE 1588 Implementation for RISC-V Low-Power Embedded Devices,” Electronics (Switzerland), vol. 13, no. 2, 2024, doi: 10.3390/electronics13020458.

L. de Vito, S. Rapuano, and L. Tomaciello, “One-way delay measurement: State of the art,” IEEE Transactions on Instrumentation and Measurement, vol. 57, no. 12, pp. 2742–2750, 2008, doi: 10.1109/TIM.2008.926052.

Date of project specification: 9.2.2026 ***Deadline for submission:*** 27.5.2026

Supervisor: Ing. František Rusnák

doc. Ing. Miroslav Jirgl, Ph.D.
Chair of study program board

WARNING:

The author of the Bachelor's Thesis claims that by creating this thesis he/she did not infringe the rights of third persons and the personal and/or property rights of third persons were not subjected to derogatory treatment. The author is fully aware of the legal consequences of an infringement of provisions as per Section 11 and following of Act No 121/2000 Coll. on copyright and rights related to copyright and on amendments to some other laws (the Copyright Act) in the wording of subsequent directives including the possible criminal consequences as resulting from provisions of Part 2, Chapter VI, Article 4 of Criminal Code 40/2009 Coll.

ABSTRACT

This bachelor's thesis deals with time synchronization between STM32 Nucleo-F767ZI development boards using the IEEE 1588 Precision Time Protocol. It compares basic time synchronization methods, describes the principles of PTP, and evaluates Ethernet communication options on the STM32 platform. Based on this analysis, firmware was designed and implemented using direct transmission of PTP messages in Ethernet frames and hardware timestamps from the Ethernet MAC. The functionality of the solution was verified in a direct connection, in a multi-slave topology through a standard Ethernet switch, and under additional network load. The thesis also theoretically evaluates the possibility of using Single Pair Ethernet as a physical layer for transmitting PTP messages.

KEYWORDS

STM32, IEEE 1588, Precision Time Protocol, PTP, Ethernet, Single Pair Ethernet, SPE, Time synchronization, Communication

ABSTRAKT

Tato bakalářská práce se zabývá synchronizací času mezi vývojovými deskami STM32 Nucleo-F767ZI pomocí protokolu IEEE 1588 Precision Time Protocol. V práci jsou porovnány základní metody časové synchronizace, popsán princip PTP a zhodnoceny možnosti práce s Ethernetem na platformě STM32. Na základě této analýzy byl navržen a implementován firmware využívající přímý přenos PTP zpráv v ethernetových rámcích a hardwarové časové značky Ethernet MAC. Funkčnost řešení byla ověřena v přímém propojení, ve víceslavové topologii přes běžný Ethernet switch a při zatížení sítě. Práce se také teoreticky zabývá možnostmi použití technologie Single Pair Ethernet jako fyzické vrstvy pro přenos PTP zpráv.

KLÍČOVÁ SLOVA

STM32, IEEE 1588, Precision Time Protocol, PTP, Ethernet, Single Pair Ethernet, SPE, Synchronizace času, Komunikace

Typeset by the `thesis` package, version 4.13; `https://latex.fekt.vut.cz/`

ROZŠÍŘENÝ ABSTRAKT

Přesná časová synchronizace je důležitou součástí moderních distribuovaných měřicích, řídicích a komunikačních systémů. Pokud více zařízení pracuje ve stejné síti a současně měří, řídí nebo zaznamenává události, musí být jejich lokální časové základny vzájemně sladěny. Bez společného času je obtížné správně porovnávat naměřená data, určovat pořadí událostí nebo řídit procesy, které vyžadují časovou koordinaci. Pro běžné počítačové sítě se často používá protokol NTP, avšak jeho přesnost je pro mnoho průmyslových a vestavných aplikací nedostatečná. Výrazně vyšší přesnosti lze dosáhnout pomocí protokolu IEEE 1588 Precision Time Protocol, zejména pokud jsou použity hardwarové časové značky na úrovni síťového rozhraní.

Cílem této bakalářské práce bylo navrhnout a implementovat software umožňující komunikaci a časovou synchronizaci mezi dostupnými vývojovými deskami STM32 Nucleo-F767ZI pomocí Ethernetu. Synchronizace je založena na výměně PTP zpráv mezi jedním hlavním zařízením a jedním nebo více podřízenými zařízeními. Součástí práce bylo také seznámení s platformou STM32 Nucleo, studium protokolu IEEE 1588, porovnání vhodných softwarových přístupů pro práci s Ethernetem a teoretické zhodnocení možností využití technologie Single Pair Ethernet jako fyzické vrstvy pro časovou synchronizaci.

V teoretické části práce jsou nejprve porovnány základní metody synchronizace času, především NTP, GPS a PTP. Dále je popsán princip činnosti protokolu IEEE 1588, jeho základní zprávy, výpočet zpoždění komunikační cesty a odchylky hodin a rozdíly mezi softwarovou, hardwarově podporovanou a plně hardwarovou implementací. Vzhledem k tomu, že výsledná implementace používá přenos PTP zpráv přímo v ethernetových rámcích, je v práci také vysvětleno umístění PTP v modelu ISO/OSI. Implementované řešení lze chápat jako PTP na druhé vrstvě s hardwarově podporovaným časovým značkováním.

Pro praktickou realizaci byla zvolena vývojová deska STM32 Nucleo-F767ZI. Tato platforma obsahuje mikrokontrolér STM32F767ZI s Ethernet MAC periferií, která podporuje hardwarové časové značky podle IEEE 1588. Při porovnání možných přístupů k práci s Ethernetem byly uvažovány varianty HAL s lwIP, přímá práce s Ethernet periferií pomocí HAL, knihovna Mongoose Embedded a FreeRTOS+TCP. Jako nejvhodnější přístup pro tuto práci byla zvolena přímá práce s Ethernetem pomocí HAL a registrů Ethernet MAC. Tento přístup nevyžaduje IP adresaci ani kompletní TCP/IP stack a současně poskytuje přímý přístup k časovacím mechanismům potřebným pro PTP.

Výsledný firmware používá přenos PTP zpráv přímo v ethernetových rámcích označených EtherType `0x88F7`. Implementace pracuje s pevně definovanými rolemi zařízení. Jedna deska je nakonfigurována jako master a periodicky odesílá zprávy

`Sync` a `Follow_Up`. Slave zařízení přijímá tyto zprávy, generuje zprávu `Delay_Req`, přijímá odpověď `Delay_Resp` a ze čtyř hardwarových časových značek vypočítává odhad zpoždění cesty a časovou odchylku vůči masteru. Pro víceslavovou topologii byl firmware rozšířen o jednoznačnou identifikaci slave zařízení, filtrování odpovědí podle cílové identity a frontu požadavků na straně masteru.

Korekce hodin na straně slave je rozdělena do dvou částí. Velké počáteční chyby jsou kompenzovány omezenými fázovými kroky, aby se lokální čas rychle přiblížil času masteru. Po dosažení menší odchylky přebírá hlavní roli frekvenční regulace pomocí PI regulátoru, který upravuje addend registr PTP časové základny. Tím se mění rychlost běhu lokálních PTP hodin bez nutnosti častých tvrdých časových skoků. Součástí firmwaru je také filtrování chybných měření. Vzorky s nepravděpodobným zpožděním cesty nebo s příliš velkým skokem odchylky nejsou použity pro regulaci, aby síťové výkyvy nepoškodily stav regulátoru.

Experimentální ověření bylo provedeno několika způsoby. Nejprve byla testována přímá ethernetová linka mezi jedním masterem a jedním slave zařízením. V této konfiguraci byla komunikační cesta velmi stabilní a implementovaný algoritmus dosáhl submikrosekundové synchronizace. Interní logy ze slave zařízení ukázaly střední hodnotu odchylky blízkou nule, směrodatnou odchylku v řádu stovek nanosekund a 99. percentil absolutní odchylky pod přibližně 0,8 μs. Měření fyzických PPS výstupů osciloskopem potvrdilo, že hodnoty z logů odpovídají skutečnému časovému rozdílu mezi deskami.

Druhá sada experimentů ověřovala činnost systému ve víceslavové topologii přes běžný Ethernet switch, který nebyl PTP-aware. Tato topologie je náročnější, protože switch přidává zpoždění a může způsobovat proměnlivé frontování rámců. Přesto se obě slave zařízení v běžném nezatíženém provozu udržovala v submikrosekundové oblasti. Naměřené zpoždění cesty se ve srovnání s přímým propojením zvýšilo přibližně z 1 μs na 8,1–8,4 μs, ale dlouhodobá synchronizace zůstala stabilní. Externí PPS měření opět potvrdilo, že reálný časový rozdíl mezi zařízeními zůstává převážně v rozsahu několika stovek nanosekund.

Poslední část experimentů zkoumala vliv dodatečného ne-PTP síťového provozu. Při zatížení 60 Mbit/s a 70 Mbit/s zůstala synchronizace zachována, ale většina PTP vzorků byla odmítnuta filtrem zpoždění cesty. To ukazuje, že běžný switch bez podpory PTP způsobuje při vysokém zatížení výrazné proměnlivé zpoždění. Při zatížení 75 Mbit/s již systém nedostával dostatek použitelných měření, stabilní synchronizace byla ztracena a slave zařízení musela provádět opakovanou resynchronizaci. Po odstranění zatížení se systém automaticky vrátil do stabilního stavu.

Práce se dále zabývá možnostmi využití Single Pair Ethernetu pro přenos PTP zpráv. SPE nemění samotný protokol PTP, ale nahrazuje fyzickou vrstvu Ethernetu přenosem po jednom symetrickém páru vodičů. Z toho vyplývá, že imple-

mentovaný princip přenosu PTP zpráv na druhé vrstvě by bylo možné přenést i na SPE. Dosažitelná přesnost by však závisela na konkrétní variantě SPE a na místě, kde jsou vytvářeny časové značky. Nejvhodnějším kandidátem pro přímé pokračování této práce je 100BASE-T1, protože zachovává rychlost 100 Mbit/s a plně duplexní point-to-point komunikaci. Varianty 10BASE-T1L a 10BASE-T1S jsou také použitelné, ale vyžadují pečlivější řešení zpoždění PHY, rozhraní MAC–PHY a v případě 10BASE-T1S také sdíleného přístupu k médiu.

Dosažené výsledky ukazují, že navržený firmware je vhodný pro synchronizaci několika STM32 desek v řízené lokální ethernetové síti. Hlavním omezením není samotný výpočet PTP ani výkon mikrokontroléru, ale proměnlivé zpoždění sítě při použití běžného switche bez podpory PTP a při vysokém zatížení. Pro další rozvoj projektu se proto nabízí několik směrů: použití PTP-aware switche nebo prioritizace PTP provozu, přenesení systému na SPE, implementace Best Master Clock Algorithm pro automatickou volbu grandmasteru a přepracování firmwaru do vhodnější architektury založené na FreeRTOS. Tyto úpravy by mohly zvýšit robustnost, škálovatelnost a použitelnost systému v realističtějších vestavných a průmyslových aplikacích.

VOLOSHIN, Ilya. *Communicating devices synchronized using the IEEE 1588 standard*. Bachelor's Thesis. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Control and Instrumentation, 2026. Advised by Ing. František Rusnák

Author's Declaration

Author:	Ilya Voloshin
Author's ID:	247494
Paper type:	Bachelor's Thesis
Academic year:	2025/26
Topic:	Communicating devices synchronized using the IEEE 1588 standard

I declare that I have written my thesis independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis. I have not violated the principles and recommendations of the Brno University of Technology regarding the use of generative artificial intelligence.

As the author, I furthermore declare that, with respect to the creation of this thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll. of the Czech Republic, Section 2, Head VI, Part 4.

Brno ....

author's signature*

*The author signs only in the printed version.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisor, Ing. František Rusnák, for his guidance, valuable consultations, and help in choosing a thesis topic that was both technically challenging and genuinely interesting to me.
I would also like to thank my mother, Marina Voloshina, for giving me the opportunity and support to study abroad. Her help and encouragement made this important stage of my education possible.
Finally, I would like to thank my secondary school computer science teacher, Maria Kulbaeva, who helped me take my first steps in research projects, conference presentations, and the development of my own technical ideas.

Declaration on the use of AI-based tools

AI-based tools, in particular DeepL, Grammarly, and ChatGPT, were used to support the preparation of this thesis. DeepL was used during the writing process to search for suitable words, synonyms, and formulations. Grammarly was used to check the text's grammar and identify spelling errors.
ChatGPT was used as an auxiliary tool for analyzing selected parts of the development board reference manual, converting tables prepared in Excel into LaTeX code, and suggesting possible ways of implementing formatting ideas in LaTeX. It was also used to provide additional feedback on selected text passages and to help identify weak points in the structure and wording of the thesis. In addition, ChatGPT supported the creation of Python scripts used for processing logs from the development boards and for real-time plotting of measured data.
All content, code, results, and interpretations were carefully reviewed by the author and evaluated for correctness. The use of these tools was intended to improve the linguistic, technical, and formal quality of the thesis and did not replace the author's own technical decisions, implementation work, measurements, analysis, or final conclusions.

Contents

Introduction **17**

1 Comparison of Time Synchronization Technologies: GPS, NTP, and PTP **19**
1.1 Network Time Protocol (NTP) 19
1.2 Global Positioning System (GPS) 21
1.3 IEEE 1588 Precision Time Protocol (PTP) 22
1.4 Comparison of Synchronization Methods 23

2 Detailed Analysis of the IEEE 1588 Precision Time Protocol (PTP) **24**
2.1 Principles of Operation 24
2.2 Standard Evolution 25
2.3 PTP Implementations 26
2.3.1 Software-Based Implementation 26
2.3.2 Hybrid (Hardware-Assisted) Implementation 26
2.3.3 Hardware-Based Implementation 28
2.4 Summary 29

3 The ISO/OSI model and the location of PTP in the stack **30**
3.1 Detailed Description of the ISO/OSI Layers 30
3.2 Placement of PTP in the ISO/OSI Model in the Context of Ethernet 32
3.2.1 PTP Directly over Ethernet 32
3.2.2 PTP over UDP/IP 33
3.2.3 Fully Hardware-Based PTP Implementations 33
3.2.4 Software and Hardware-Assisted Timestamping 34
3.2.5 Summary for the Developed System 34

4 Introduction to the STM Nucleo Platform and Study of Its Hardware and Software Capabilities **35**
4.1 Hardware Architecture of the STM Nucleo-F767ZI Platform 35
4.2 Key Hardware Components relevant to the implementation of the IEEE 1588 Precision Time Protocol 36
4.2.1 STM32F767ZI Microcontroller 36
4.2.2 Hardware Support for IEEE 1588 PTP 37
4.2.3 Clocking and Time Base 37
4.2.4 Ethernet Physical Interface 37
4.3 Software Development Environment 37

5 Comparison of libraries and approaches for working with Ethernet 39
5.1 HAL + lwIP (STM32CubeIDE) . . . 39
5.2 Direct Ethernet handling using HAL . . . 40
5.3 Mongoose Embedded . . . 41
5.4 FreeRTOS+TCP . . . 42
5.5 Summary of the Comparison . . . 43

6 Description of the PTP Implementation 45
6.1 Overall Firmware Architecture . . . 45
6.2 Hardware Timestamping Implementation . . . 47
6.3 PTP Frame Handling . . . 49
6.4 Master Firmware . . . 49
6.4.1 Two-Step Sync Generation . . . 51
6.4.2 Delay Request Queue for Multi-Slave Operation . . . 51
6.5 Slave Firmware . . . 51
6.5.1 Slave Identity and Frame Filtering . . . 53
6.5.2 Receive Callback and Event Structures . . . 53
6.5.3 Delay Request Generation and Timestamp Acquisition . . . 54
6.5.4 End-to-End Offset and Path Delay Calculation . . . 54
6.5.5 Filtering and Rejection of Bad Measurements . . . 55
6.6 Clock Correction Strategy . . . 55
6.6.1 Fine Frequency Correction Using Addend . . . 56
6.6.2 Experimental Options That Were Not Kept in the Baseline . . . 58
6.7 Logging and Measurement Support . . . 58
6.8 Summary . . . 59

7 Experimental evaluation of the implemented algorithm 60
7.1 Measurement methodology . . . 60
7.1.1 Log-based measurement . . . 61
7.1.2 PPS measurement . . . 61
7.2 Direct master–slave connection . . . 62
7.3 Multi-slave Synchronization through an Ordinary Ethernet Switch . . . 65
7.4 Behavior of outliers and rejected samples . . . 68
7.5 Network stress test . . . 69
7.6 Summary of obtained results . . . 73

8 Single Pair Ethernet as a Physical Layer for IEEE 1588 Time Synchronization 74
8.1 Single Pair Ethernet Overview . . . 74
8.2 PTP Transport over SPE . . . 77

8.3 Timing Mechanisms Affected by the SPE Physical Layer 79
8.3.1 Timestamp Reference Plane 79
8.3.2 MII, RMII, and PHY Internal Delay 80
8.3.3 Multidrop 10BASE-T1S and PLCA 80
8.3.4 Bandwidth and Network Load 81
8.4 Experimental Results Reported in Related Work 81
8.4.1 100BASE-T1 and gPTP in Automotive Ethernet 82
8.4.2 SPE PHY Comparison with flexPTP 82
8.4.3 10BASE-T1L Timestamping Accuracy 83
8.4.4 10BASE-T1S Hardware Timestamping Example 83
8.5 Comparison with the Implemented Firmware 83
8.6 Summary of the Possibilities for Using PTP over SPE 86

9 Possible Future Development of the Project 88
9.1 Transfer to Single Pair Ethernet 88
9.2 Best Master Clock Algorithm 89
9.3 Migration to an RTOS-Based Firmware Architecture 90

Conclusion 91

Bibliography 94

List of appendices 96

A Selected Firmware Code Listings 97

B Supplementary Measurement Material 102
B.1 Log-based Measurement Interface 102
B.2 Supplementary Measurement Tables 104

C Contents of the Electronic Appendix 110

List of Figures

1.1 Typical NTP synchronization subnet showing hierarchical structure and fallback paths. 19
1.2 General architecture of GPS-based synchronization systems. 21
1.3 Simplified message exchange in IEEE 1588 Precision Time Protocol. . 22
2.1 Timestamping in a software-based PTP implementation. [1] 27
2.2 Hybrid IEEE 1588 implementation. [1] 27
2.3 Fully hardware-based IEEE 1588 implementation. [1] 28
3.1 Standard representation of the ISO/OSI model 30
4.1 Simplified schematic view of the STM32 Nucleo-F767ZI board highlighting components relevant to Ethernet communication and IEEE 1588 PTP synchronization 36
6.1 Flowchart of the implemented end-to-end PTP synchronization procedure. 46
6.2 Flowchart of the master firmware workflow. 50
6.3 Flowchart of the slave firmware workflow. 52
6.4 Slave receive callback processing. 53
7.1 Measurement setup used for the direct master–slave synchronization experiment. 62
7.2 PPS offset distribution for the direct master–slave Ethernet connection. Each window of samples contains 30 oscilloscope measurements. 63
7.3 Logged offset and filtered offset in the direct master–slave Ethernet connection after the initial transient. 64
7.4 Measurement setup used for the multi-slave synchronization experiment through an ordinary Ethernet switch. 65
7.5 Logged offsets of two slaves synchronized through an Ethernet switch in a representative multi-slave run. 66
7.6 PPS offset distributions for two slaves synchronized through an Ethernet switch. Each window contains 30 oscilloscope measurements. . . 67
7.7 Long-term log-based offset of two slaves synchronized through an Ethernet switch without additional network load. 68
7.8 Network stress test: baseline switch-based multi-slave operation before additional traffic. 69
7.9 Network stress test: switch-based multi-slave operation under 60 Mbit/s additional Ethernet traffic. 70
7.10 Network stress test: switch-based multi-slave operation under 70 Mbit/s additional Ethernet traffic. 70

7.11 Network stress test: loss of stable synchronization under 75 Mbit/s additional Ethernet traffic. 70
7.12 Network stress test: initial recovery after the 75 Mbit/s load was removed. 71
7.13 Network stress test: stable operation after recovery from the overloaded interval. 71
8.1 Simplified view of 4B3T coding and PAM-3 modulation in a 10BASE-T1L physical layer. 76
8.2 Simplified view of 4B/5B coding and DME encoding in a 10BASE-T1S physical layer. 76
8.3 Conceptual transfer of the implemented Layer 2 PTP approach from the current 100BASE-TX setup to a possible SPE setup. 78
8.4 Simplified timestamp reference plane in an SPE-based PTP system. . 79
8.5 Simplified 10BASE-T1S multidrop segment. 81
8.6 Reported PPS offset ranges for Ethernet and SPE-based PTP synchronization. 86
B.1 Example of the PC-side Python live logging and plotting window used during multicast log-based measurements. 103

List of Tables

1.1 Comparison of main time synchronization technologies [2]. 23
2.1 Main versions of the IEEE 1588 standard [1]. 25
5.1 Comparison of Ethernet approaches for PTP implementation 44
6.1 Main roles in the implemented PTP system. 47
7.1 Main components of the experimental setup. 60
8.1 Main SPE variants relevant to IEEE 1588 time synchronization. . . . 75
8.2 Comparison of reported synchronization or timestamping results. . . 85
B.1 Direct master–slave connection: external PPS results. 104
B.2 Direct master–slave connection: log-based results. 105
B.3 Multi-slave connection through a switch: log-based results. 106
B.4 Multi-slave switch topology: PPS results for slave 1. 107
B.5 Multi-slave switch topology: PPS results for slave 2. 108
B.6 Summary of the 0–60–70–75–0 Mbit/s network stress test. 109

List of Listings

A.1 PTP hardware timestamping initialization. 97
A.2 Conversion from nanoseconds to STM32 PTP subsecond update format. 98
A.3 Direct RX descriptor access for received PTP event timestamps. . . . 99
A.4 Direct TX descriptor access for the `Delay_Req` hardware timestamp. . 100
A.5 Core end-to-end delay and offset calculation on the slave. 101

Introduction

Precise time synchronization is an important requirement in modern distributed measurement, control, and communication systems. When several embedded devices operate in the same network, their local clocks are never perfectly identical. Oscillator tolerances, temperature changes, processing delays, and communication latency gradually create time differences between devices. In simple applications this may not be critical, but in systems where events must be measured, logged, compared, or controlled with a common time reference, clock synchronization becomes a necessary part of the communication infrastructure.

Several methods can be used for time synchronization. Network Time Protocol (NTP) is widely used in computer networks, but its accuracy is usually not sufficient for precise embedded or industrial applications. Global Positioning System (GPS) can provide an accurate external time reference, but it requires additional hardware, antenna placement, and satellite signal availability. For local Ethernet-based systems, IEEE 1588 Precision Time Protocol (PTP) provides a more suitable solution, especially when hardware timestamping is available. PTP allows devices connected through a network to estimate the offset between their clocks and compensate for the communication path delay.

The aim of this thesis is to design, implement, and evaluate software for time synchronization between STM32 Nucleo development boards using Ethernet and IEEE 1588 PTP messages. The practical implementation is based on STM32 Nucleo-F767ZI boards. This platform was selected because the STM32F767ZI microcontroller includes an Ethernet MAC peripheral with hardware timestamping support, which makes it possible to capture transmit and receive times close to the Ethernet interface. The developed system uses a hardware-assisted approach: the time-critical timestamps are generated by the Ethernet hardware, while PTP message processing, delay calculation, filtering, and clock correction are implemented in firmware.

The implementation focuses on a controlled embedded Ethernet network rather than on a complete general-purpose PTP stack. One board is configured as a fixed master and one or more boards are configured as slaves. PTP messages are transmitted directly in Ethernet frames using EtherType `0x88F7`, without IP or UDP encapsulation. This reduces the amount of required networking software and allows the firmware to work directly with Ethernet frames, DMA descriptors, MAC registers, and the STM32 PTP time base. The slave firmware calculates the clock offset from the standard PTP timestamp exchange and corrects its local clock using bounded phase updates and fine frequency correction through the PTP addend register.

An important part of the work is the experimental evaluation of the implemented

synchronization algorithm. The system is tested in a direct master–slave Ethernet connection, in a topology with one master and two slaves connected through an ordinary Ethernet switch, and under additional non-PTP network load. The evaluation uses both internal log-based measurements and external PPS measurements. This combination makes it possible to analyze the internal behavior of the firmware and also verify the real physical timing difference between boards.

In addition to the practical implementation over standard 100BASE-TX Ethernet, the thesis also evaluates Single Pair Ethernet (SPE) as a possible physical layer for IEEE 1588 time synchronization. SPE is becoming increasingly relevant in automotive and industrial automation systems because it preserves the Ethernet communication model while reducing cabling complexity. Since PTP is not tied to one specific Ethernet physical layer, it is meaningful to analyze whether the implemented Layer 2 PTP approach could be transferred to SPE links. The thesis therefore discusses relevant SPE variants, such as 100BASE-T1, 10BASE-T1L, and 10BASE-T1S, and compares expected timing limitations with results reported in related work.

The thesis is organized as follows. Chapter 1 compares the main time synchronization technologies: NTP, GPS, and PTP. Chapter 2 describes the IEEE 1588 Precision Time Protocol and its implementation approaches. Chapter 3 discusses the placement of PTP in the ISO/OSI model, with emphasis on the Layer 2 transport mapping used in this work. Chapter 4 introduces the STM32 Nucleo-F767ZI platform and its hardware support for Ethernet and IEEE 1588 timestamping. Chapter 5 compares possible software approaches for Ethernet communication on STM32 and explains the selection of direct HAL-based Ethernet handling. Chapter 6 describes the implemented master and slave firmware. Chapter 7 presents the experimental evaluation of the synchronization accuracy and robustness. Chapter 8 analyzes the possibilities of using Single Pair Ethernet as a physical layer for PTP. Chapter 9 outlines possible future development of the project, including SPE implementation, BMCA support, and migration to an RTOS-based firmware architecture.

1 Comparison of Time Synchronization Technologies: GPS, NTP, and PTP

Accurate time synchronization is a fundamental requirement for modern distributed systems, particularly in real-time communication networks and measurement applications. The paper *"One-Way Delay Measurement: State of the Art"* [2] provides a detailed overview and comparison of three major synchronization methods—**Network Time Protocol (NTP)**, **Global Positioning System (GPS)**, and **IEEE 1588 Precision Time Protocol (PTP)**—highlighting their advantages, limitations, and achievable accuracy.

Each of these technologies provides clock alignment between devices in a network, but they differ significantly in terms of precision, cost, hardware requirements, and suitability for specific network types. According to [2], NTP typically achieves synchronization accuracy in the order of milliseconds, GPS in the order of microseconds, and PTP in the sub-microsecond range when hardware timestamping is used.

1.1 Network Time Protocol (NTP)

The **Network Time Protocol (NTP)** is one of the oldest and most widely adopted methods for time synchronization over computer networks. It is based on a hierarchical structure of primary and secondary servers, where each layer synchronizes its time with a more accurate server above it. A typical NTP subnet hierarchy is illustrated in Figure 1.1, showing primary reference servers, intermediate servers, and clients organized in a tree-like configuration [2].

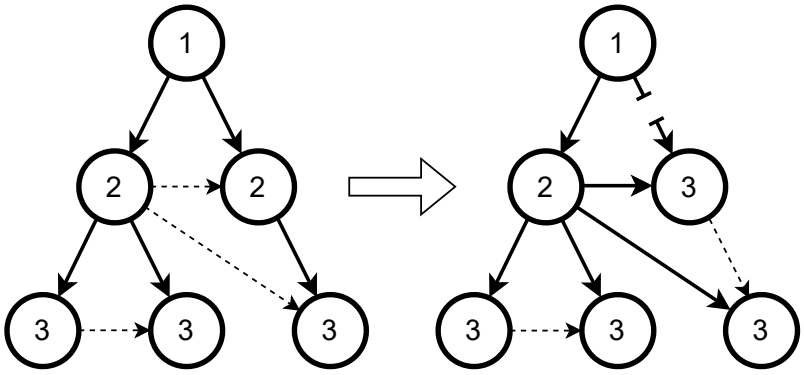


Fig. 1.1: Typical NTP synchronization subnet showing hierarchical structure and fallback paths.

In the common client-server mode, one NTP synchronization exchange consists of two network messages: a request sent by the client and a response sent by the server. The exchange is based on four timestamps:

- t_1 – time when the client sends the request;
- t_2 – time when the server receives the request;
- t_3 – time when the server sends the response;
- t_4 – time when the client receives the response.

In the NTP message structure, the client sends its transmit timestamp in the request. The server copies this value into the originate timestamp field of the response and adds its own receive and transmit timestamps. The final receive timestamp is recorded locally by the client when the response arrives. Therefore, unlike PTP, NTP does not use separate delay request and delay response messages for path delay estimation; both the clock offset and the network delay are estimated from the same request-response transaction.

The round-trip delay δ is calculated as:

$$\delta = (t_4 - t_1) - (t_3 - t_2), \tag{1.1}$$

where $(t_4 - t_1)$ represents the total elapsed time measured by the client, and $(t_3 - t_2)$ represents the time spent processing the request on the server side. Assuming approximately symmetric propagation delay in both directions, the one-way path delay can be estimated as $\delta/2$.

The clock offset θ between the client and the server is estimated as:

$$\theta = \frac{(t_2 - t_1) + (t_3 - t_4)}{2}. \tag{1.2}$$

This value represents the correction that should be applied to the client clock, under the assumption that the forward and reverse network delays are equal. Network delay asymmetry therefore directly affects the accuracy of the estimated offset.

Under standard network conditions, NTP provides accuracy within approximately **1–10 ms** in local area networks (LANs) and up to tens of milliseconds over wide area networks (WANs). Its accuracy depends on various factors such as:

- network congestion and latency asymmetry;
- jitter in message propagation;
- software timestamping uncertainty within operating systems.

Although NTP is sufficient for non-real-time systems, its accuracy is generally inadequate for industrial or embedded applications requiring microsecond-level or sub-microsecond synchronization. Hardware-assisted NTP implementations can improve timestamp accuracy, but standard NTP operation is still mainly designed for synchronization over general-purpose IP networks. In contrast, PTP provides a more specialized synchronization mechanism for local networks, supports hardware

timestamping as a fundamental part of accurate implementations, and can use PTP-aware network devices such as boundary clocks and transparent clocks to reduce the influence of intermediate network equipment.

1.2 Global Positioning System (GPS)

The **Global Positioning System (GPS)** can also be used as a reference for time synchronization, offering significantly higher accuracy than NTP. Each GPS receiver synchronizes its internal clock using signals from at least four satellites transmitting their positions and the coordinated universal time (UTC). A simplified overview of GPS-based synchronization architecture is shown in Figure 1.2.

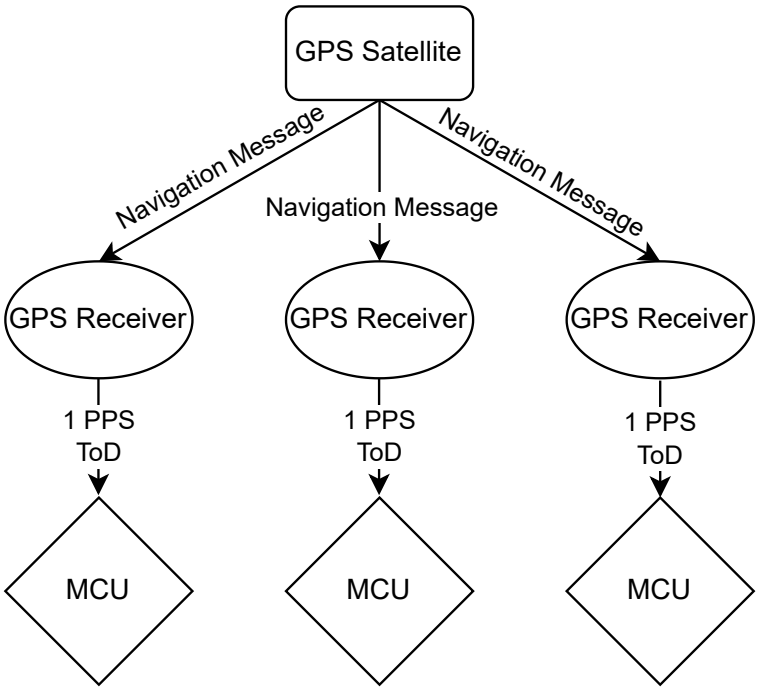


Fig. 1.2: General architecture of GPS-based synchronization systems.

GPS-based synchronization enables time alignment accuracy within **microseconds** or even tens of nanoseconds under optimal conditions. However, despite its superior precision, GPS presents several disadvantages:

- high installation and maintenance costs due to the need for antennas and receivers;
- dependency on satellite visibility, which limits its use indoors or underground;

- vulnerability to signal interference and environmental conditions.

As highlighted in [2], GPS-based solutions are typically suitable for large-scale distributed networks or scientific measurements but are often impractical for compact embedded systems or indoor industrial environments.

1.3 IEEE 1588 Precision Time Protocol (PTP)

The **IEEE 1588 Precision Time Protocol (PTP)** was introduced to achieve precise synchronization over Ethernet networks without relying on external satellite systems. PTP operates by exchanging timestamped messages between a *Master* and one or more *Slave* devices. The fundamental synchronization process includes the transmission of `SYNC`, `Follow_Up`, `Delay_Req`, and `Delay_Resp` messages, as shown schematically in Figure 1.3.

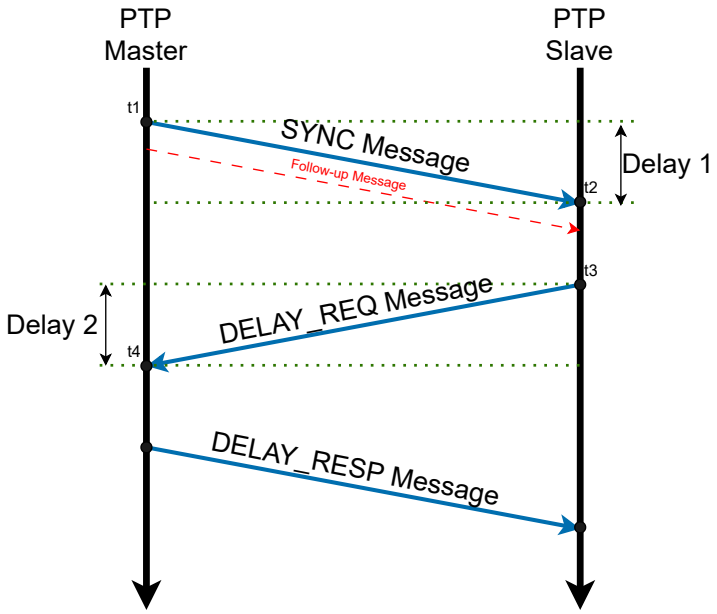


Fig. 1.3: Simplified message exchange in IEEE 1588 Precision Time Protocol.

The synchronization accuracy of PTP strongly depends on the implementation method:

- **Software-based PTP** (timestamping at the application layer) achieves precision of about 10–100 µs;
- **Hardware-assisted PTP**, with timestamping performed directly in the network interface controller (NIC) or PHY, can achieve sub-microsecond or even **100 ns** accuracy.

The IEEE 1588 standard has therefore become a de facto choice for **industrial automation**, **measurement systems**, and **embedded applications** requiring deterministic timing. Compared with NTP and GPS, PTP provides a balance between achievable precision, implementation cost, and deployment simplicity, especially in local Ethernet environments.

1.4 Comparison of Synchronization Methods

A summary comparison of the three synchronization techniques is presented in Table 1.1, based on the data from [2] (Table II).

Table 1.1: Comparison of main time synchronization technologies [2].

Method	**Typical Accuracy**	**Required Hardware**	**Applicable Network**	**Main Limitations**
NTP	1–10 ms	None (software only)	LAN / WAN	Insufficient accuracy for industrial or time-critical systems.
GPS	0.1–1 µs	GPS receiver and antenna	Global	High cost, satellite visibility required, impractical for indoor environments.
PTP (IEEE 1588)	<1 µs (down to 100 ns with HW)	NIC or PHY with hardware timestamping	LAN / Industrial Ethernet	Requires hardware support; limited to local networks.

2 Detailed Analysis of the IEEE 1588 Precision Time Protocol (PTP)

The **Precision Time Protocol (PTP)**, standardized as **IEEE 1588**, enables submicrosecond synchronization between distributed devices connected through Ethernet networks. It has become the preferred solution in industrial automation, power systems, and embedded communication due to its ability to provide deterministic timing using standard network hardware [1]. While the previous chapter introduced PTP as one of several synchronization methods, this section provides a detailed analysis of its operation principles and implementation approaches.

2.1 Principles of Operation

According to the IEEE 1588 standard, all devices participating in synchronization form a so-called **PTP domain**. Within this domain, one node is selected as the **Grandmaster Clock** — the primary source of time for the entire network. All other devices, referred to as **Slave Clocks**, adjust their local time based on periodic synchronization messages received from the Grandmaster.

The overall communication is organized hierarchically, which ensures high reliability and fault tolerance of the system. If the current master device fails or becomes unavailable, the protocol can automatically elect a new Grandmaster using the **Best Master Clock Algorithm (BMCA)**. This algorithm evaluates each device according to its clock quality, stability, and priority values defined in the network configuration.

The basic communication model of PTP thus follows a *master–slave* architecture, where the Grandmaster provides the global time reference, and the slaves continuously synchronize their clocks through message exchanges. The fundamental message types and their sequence have already been illustrated in Figure 1.3.

The synchronization process consists of two main phases:

1. **Offset correction:** Measuring and compensating for the time difference between master and slave clocks;
2. **Delay measurement:** Determining and removing the propagation delay caused by the network link.

During each synchronization cycle, four key timestamps are recorded:

- t_1 — transmission time of the `SYNC` message from the master;
- t_2 — reception time of the `SYNC` message at the slave;
- t_3 — transmission time of the `Delay_Req` message from the slave;
- t_4 — reception time of the `Delay_Req` message at the master.

Using these timestamps, the slave computes the network delay and clock offset according to the following equations:

$$Delay = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \tag{2.1}$$

$$Offset = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{2.2}$$

The slave then adjusts its local clock by subtracting the calculated offset from its current time. This procedure is executed periodically to maintain synchronization with the Grandmaster Clock and to compensate for clock drift or environmental influences such as temperature changes.

PTP supports two synchronization mechanisms:

- **End-to-End (E2E):** The propagation delay is measured directly between the master and each slave;
- **Peer-to-Peer (P2P):** Each intermediate network node measures its own link delay and forwards the timing correction downstream.

The latter approach provides improved accuracy in switched Ethernet networks, as it compensates for the latency introduced by each network segment or hop.

2.2 Standard Evolution

Since its first publication, the IEEE 1588 standard has evolved significantly to improve accuracy, scalability, and applicability to modern embedded systems. Table 2.1 summarizes the most relevant revisions.

Table 2.1: Main versions of the IEEE 1588 standard [1].

Version	**Year**	**Key Improvements**
PTPv1	2002	Initial release introducing the concept of master–slave clock synchronization in local networks.
PTPv2	2008	Enhanced precision and scalability; added boundary and transparent clocks to compensate for switch delays.
PTPv2.1	2019	Improved energy efficiency and security; introduced optimizations for low-power embedded devices.

The introduction of *Transparent Clocks* and *Boundary Clocks* in PTPv2 was a major milestone, allowing compensation for switch and router latencies. This makes it feasible to achieve microsecond or even sub-microsecond accuracy in multi-hop Ethernet environments.

2.3 PTP Implementations

The IEEE 1588 Precision Time Protocol can be realized in several different ways, depending on the available hardware resources, desired synchronization accuracy, and system constraints. Three main approaches are commonly used in practice: **software-based**, **hybrid (hardware-assisted)**, and **hardware-based** implementations. Each approach offers a distinct balance between cost, complexity, and achievable precision.

2.3.1 Software-Based Implementation

In a **software-only implementation**, all timestamping, message processing, and clock adjustments are handled entirely by the system's central processor and operating system. The transmission and reception timestamps are generated in software, usually at the application or driver level. A schematic diagram of this implementation is shown in Figure 2.1. While this approach is the simplest to implement and the most flexible, it introduces significant non-determinism due to operating-system scheduling, interrupt latency, and network-stack delays.

Such implementations typically achieve synchronization accuracy in the range of **10–100 µs**. They are suitable for non-critical applications where microsecond-level precision is not required, such as general-purpose computing or distributed data logging. Software-based PTP stacks (e.g., `ptpd` or `linuxptp`) are commonly used on desktop or embedded Linux systems, where hardware timestamping is unavailable or not required.

2.3.2 Hybrid (Hardware-Assisted) Implementation

A **hybrid PTP implementation** combines software flexibility with hardware precision. In this architecture, timestamping is performed at the hardware layer—typically within the Ethernet MAC or PHY—while protocol logic, message handling, and offset correction are executed in software. This design reduces jitter caused by operating-system and driver latency, ensuring deterministic timestamp generation close to the physical transmission layer. A schematic diagram of this implementation is shown in Figure 2.2.

Hybrid implementations generally achieve synchronization accuracy between **100 ns and 1 µs**. They represent the most common solution for embedded controllers such as STM32-based systems, where cost and energy efficiency are critical but high timing precision is still required. In this architecture, software (e.g., STM32 HAL Ethernet driver) manages PTP frame handling and clock correction, while hardware ensures accurate timestamping of transmitted and received packets.

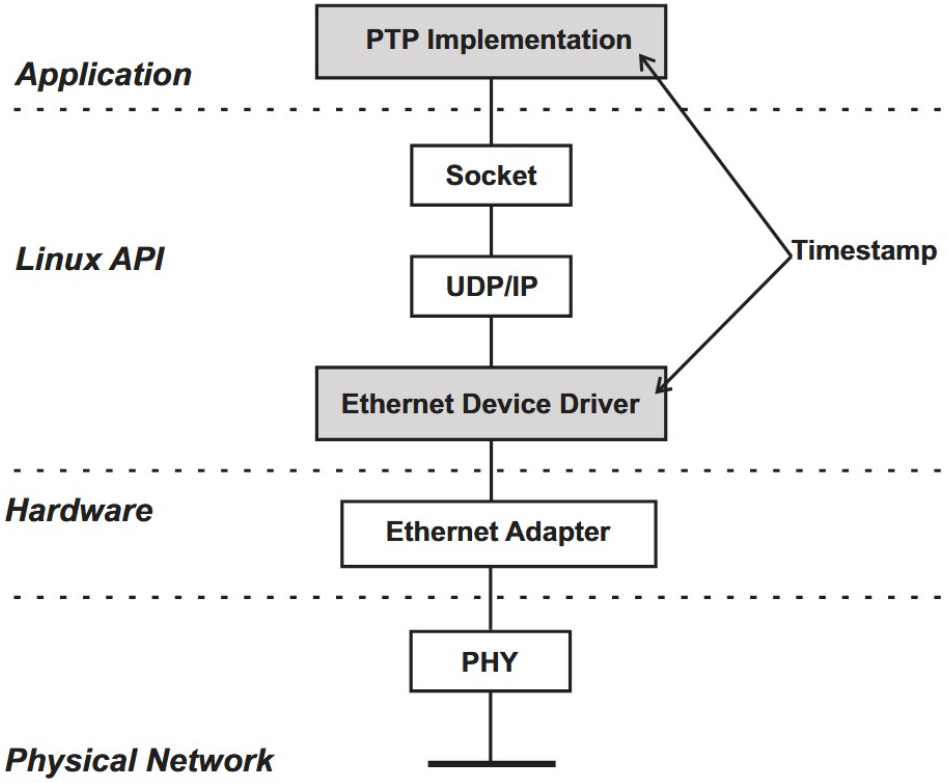


Fig. 2.1: Timestamping in a software-based PTP implementation. [1]

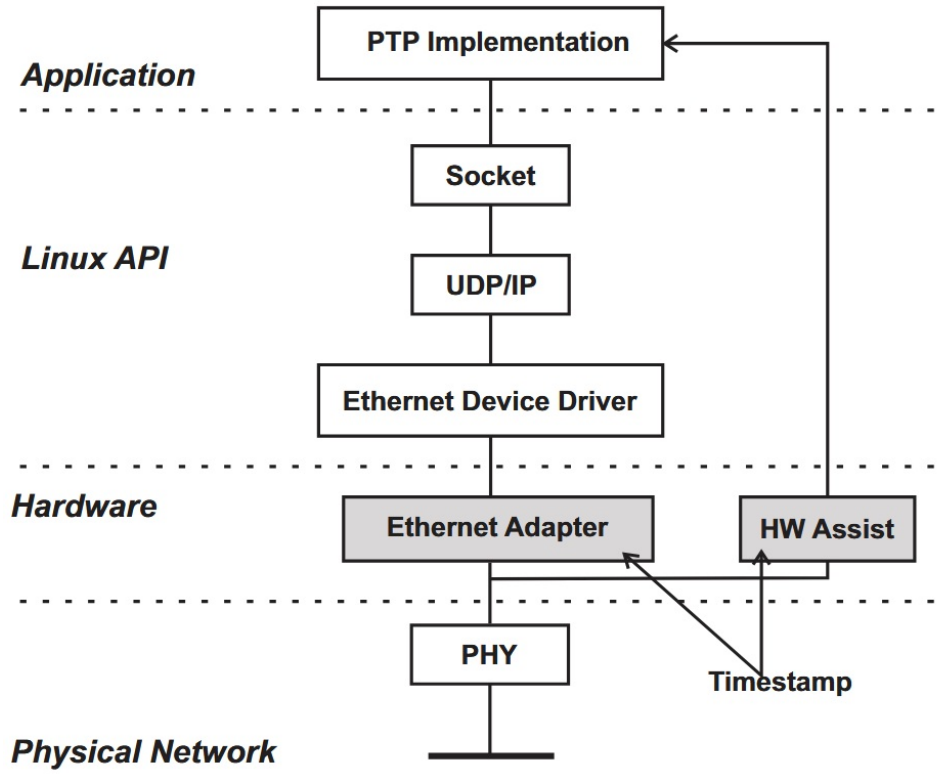


Fig. 2.2: Hybrid IEEE 1588 implementation. [1]

2.3.3 Hardware-Based Implementation

A **hardware-based PTP** architecture implements the entire protocol stack—including timestamp generation, message parsing, delay computation, and clock correction—within dedicated digital logic, such as FPGA or ASIC. This approach minimizes latency variation and provides fully deterministic performance, as all time-critical operations are executed in hardware without operating-system interference. A schematic diagram of this implementation is shown in Figure 2.3.

Hardware-Based PTP systems achieve the highest synchronization accuracy, often within tens of nanoseconds. They are primarily used in specialized applications such as telecommunication backbones, power-grid automation, and scientific instrumentation—where absolute determinism and reliability are critical. The main drawbacks of this approach are reduced flexibility, higher power consumption, and increased development cost.

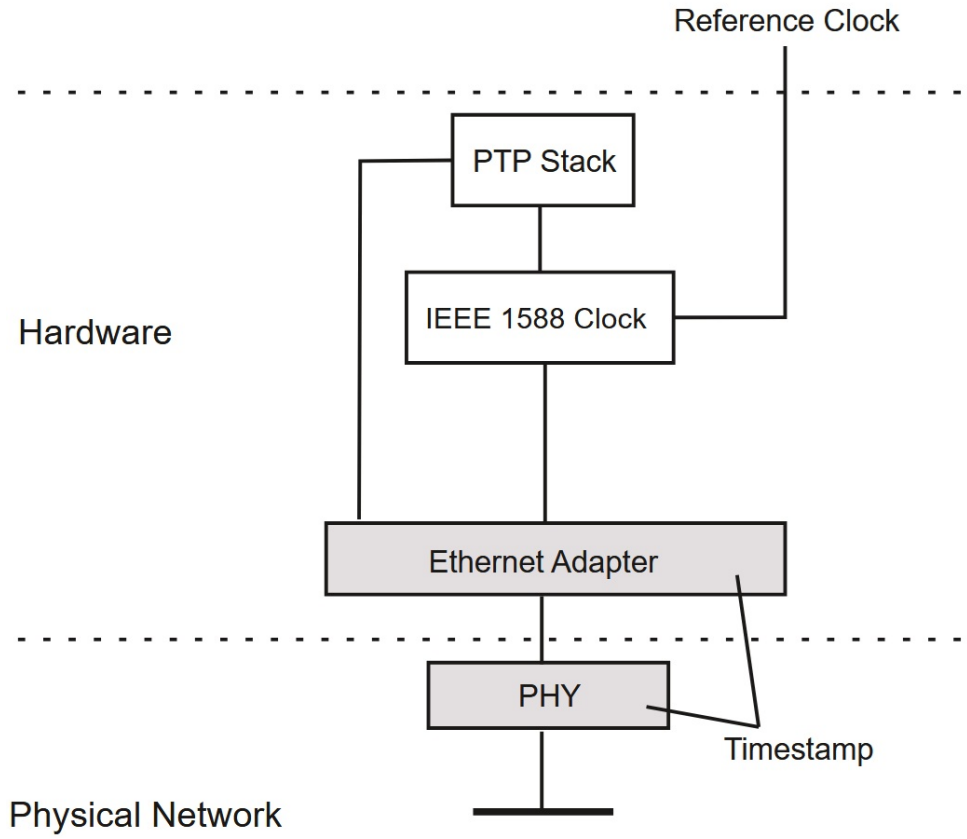


Fig. 2.3: Fully hardware-based IEEE 1588 implementation. [1]

2.4 Summary

The comparison of the three implementation approaches shows clear trade-offs between achievable precision, complexity, and cost. Software-based PTP is simple and flexible but limited in accuracy due to operating-system and network delays. Hybrid implementations, combining software control with hardware timestamping, provide sub-microsecond precision and are well suited for embedded systems such as the **STM32 Nucleo-F767ZI**. Full-hardware solutions achieve the highest accuracy—tens of nanoseconds—but at the expense of flexibility and implementation effort.

For the purposes of this thesis, the **hybrid approach** is adopted as the most practical option. It offers a balanced compromise between accuracy, resource requirements, and system complexity, making it ideal for verifying IEEE 1588 synchronization over Ethernet (100BASE-TX).

3 The ISO/OSI model and the location of PTP in the stack

The ISO/OSI (Open Systems Interconnection) model is a seven-layer reference architecture used to describe data transmission processes in communication networks. Figure 3.1 illustrates this layered structure for two communicating devices, where each layer provides services to the layer above and relies on the layer below. The model introduces a strict hierarchy of layers with clearly separated functions, which simplifies protocol design, comparison of different networking technologies, and integration of equipment from different vendors.

The key idea of the model is that each layer uses the services of the layer below and provides an abstraction to the layer above. This allows individual layers (for example, the physical or network layer) to be modified or upgraded without redesigning the entire protocol stack, provided that the interfaces between layers remain consistent.

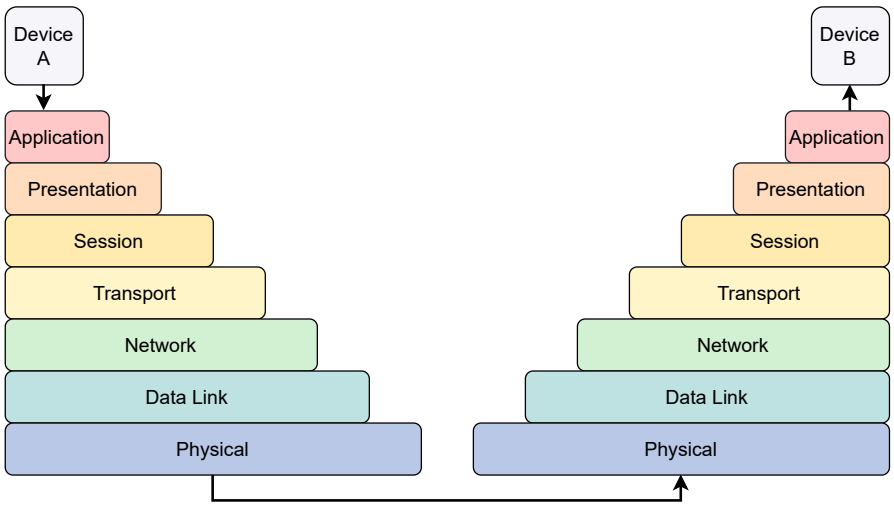


Fig. 3.1: Standard representation of the ISO/OSI model

3.1 Detailed Description of the ISO/OSI Layers

Layer 1: Physical Layer

The physical layer is responsible for transmitting bits over the physical medium. At this layer, the type and parameters of cables, connectors and transceivers are defined, as well as signal encoding and modulation schemes, permissible line length, and electromagnetic compatibility requirements. Data at this level are treated as

a continuous stream of bits without any frame or packet structure. The physical layer is typically implemented in hardware: cables, connectors, transmission lines and transceivers.

Layer 2: Data Link Layer

The data link layer provides data transmission between nodes sharing a common physical medium. It forms frames by adding service information to the payload and implements link-layer addressing (for example, MAC addresses in Ethernet). At this layer, error detection mechanisms (CRC and other integrity checks) are implemented and, in some technologies, medium access control. For upper layers, the data link layer provides the service of reliable frame delivery between neighboring nodes within a single link-layer domain.

Layer 3: Network Layer

The network layer is responsible for logical addressing and routing of packets between nodes located in different networks. It defines network addresses, routing rules and mechanisms for fragmentation and reassembly of packets when traversing segments with different maximum transmission units. The network layer provides end-to-end packet delivery from source to destination through one or more routers.

Layer 4: Transport Layer

The transport layer provides end-to-end data transfer between application processes running on different nodes. It segments data streams into blocks, controls flow, and, if required, ensures reliable delivery with acknowledgements and retransmissions. The transport layer also multiplexes several logical connections over a single network connection, providing application protocols with a uniform interface for exchanging data.

Layer 5: Session Layer

The session layer manages logical sessions between applications. Its functions include establishing, maintaining and properly terminating communication sessions, as well as providing mechanisms for dialogue synchronization and recovery after failures. In modern protocol stacks, many of these functions are integrated into the transport and application layers, but the concept of a session remains useful for a structured description of interactions.

Layer 6: Presentation Layer

The presentation layer is responsible for representing data in a consistent format, independent of the internal storage methods used on a particular platform. At this layer, data format conversions (for example, character encodings) are performed, and, if necessary, data compression as well as encryption and decryption. The presentation layer thus ensures mutual understanding between applications even if they use different internal representations of information.

Layer 7: Application Layer

The application layer is the top layer of the model and provides the interface between the network and application programs. At this layer, protocols implementing specific network services operate, such as file transfer, remote access, device management or time synchronization services. The application layer defines the formats and semantics of messages exchanged by applications and the rules for their processing.

3.2 Placement of PTP in the ISO/OSI Model in the Context of Ethernet

The position of PTP in the ISO/OSI model is not fixed by the protocol alone. It depends on the selected transport mapping and on the implementation point at which PTP messages are recognized, timestamped and processed. Therefore, different PTP implementations can be associated with different parts of the communication stack.

In Ethernet-based systems, the most important distinction is whether PTP messages are transported directly in Ethernet frames or encapsulated into UDP/IP packets. In addition, the timestamping and clock-correction mechanisms may be implemented either in software, in a hardware-assisted Ethernet peripheral, or entirely in dedicated hardware logic.

3.2.1 PTP Directly over Ethernet

When PTP is transported directly over Ethernet, the PTP message is placed immediately after the Ethernet header and the frame is identified by the dedicated EtherType `0x88F7`. No IP or UDP headers are present in this mapping.

Such an implementation is naturally associated with the data link layer. PTP traffic is selected by Ethernet frame fields, uses MAC-level addressing, and is delivered inside a Layer 2 Ethernet domain. For this reason, direct Ethernet PTP can be regarded as a Layer 2 transport mapping of PTP.

This variant is suitable for local Ethernet networks, embedded systems, industrial communication, and other controlled environments where all participating clocks are located in the same Layer 2 domain or are connected through PTP-aware Ethernet equipment. Since the packet structure is simple and the PTP frame can be recognized close to the MAC interface, this mapping is also convenient for hardware timestamping.

3.2.2 PTP over UDP/IP

PTP may also be transported over UDP/IP. In this case, the PTP message is placed into the UDP payload, the UDP datagram is carried by IPv4 or IPv6, and the resulting IP packet is transmitted over Ethernet or another link-layer technology.

From the ISO/OSI point of view, this variant places PTP above the transport layer, while its delivery depends on Layer 3 and Layer 4 mechanisms. In engineering descriptions, such an implementation is often referred to as an IP-based or Layer 3 PTP implementation, because IP addressing and routing determine the network scope of PTP message delivery.

This mapping is appropriate when synchronization traffic must pass through IP-based infrastructure, for example between different subnets or across a larger routed network. Compared with direct Ethernet transport, it provides greater topological flexibility, but the timing behavior becomes more dependent on packet routing, buffering, and network-device processing. Therefore, high-accuracy PTP over UDP/IP typically requires hardware timestamping and PTP-aware network elements such as boundary clocks or transparent clocks.

3.2.3 Fully Hardware-Based PTP Implementations

In a fully hardware-based PTP implementation, the protocol engine is implemented in dedicated logic, for example in an FPGA, ASIC or specialized Ethernet controller. Such logic may recognize PTP frames, parse message fields, generate protocol messages, capture timestamps, calculate delays and update the local clock without direct software participation.

This type of implementation is located close to the boundary between the physical and data link layers, because timestamp capture may occur near the MAC/PHY interface. However, the implemented functions still operate on Ethernet frames and PTP message fields. Therefore, a fully hardware-based PTP engine should not be classified as a pure Layer 1 protocol. It is more accurately described as a hardware implementation of Layer 2 or higher-layer PTP processing placed close to the Layer 1/Layer 2 boundary.

3.2.4 Software and Hardware-Assisted Timestamping

Timestamping affects the accuracy and determinism of PTP, but it does not by itself define the ISO/OSI placement of the transport mapping. The layer classification is determined mainly by the packet encapsulation, while timestamping describes where the time of transmission or reception is captured.

If a timestamp is taken by firmware or software after the packet has been received and passed to the processor, the implementation can be described as software timestamping. In this case, the PTP transport may still be Layer 2 or UDP/IP-based, but the timestamp is produced at a higher software-processing stage. This approach is simple, but it includes interrupt latency, DMA handling time and software execution delay in the timestamp uncertainty.

If the timestamp is captured by the Ethernet MAC or by a dedicated network controller, the implementation can be described as hardware-assisted timestamping. The main PTP state machine may still run in software, but the critical time values are measured closer to the actual frame transmission or reception point. This improves timing precision without changing the logical placement of the PTP transport mapping in the ISO/OSI stack.

3.2.5 Summary for the Developed System

The developed system uses PTP directly over Ethernet. PTP messages are transmitted and received as Ethernet frames identified by EtherType `0x88F7`; IP addressing, routing, UDP ports and transport-layer multiplexing are not used for PTP traffic.

Therefore, the implemented PTP transport mapping is classified as a **Layer 2 Ethernet-based implementation**. At the same time, the use of hardware timestamping means that the timestamp capture mechanism is placed close to the MAC/-PHY boundary. Consequently, the developed system can be described as a Layer 2 PTP implementation with hardware-assisted timestamping.

4 Introduction to the STM Nucleo Platform and Study of Its Hardware and Software Capabilities

The **STM32 Nucleo-F767ZI** development board represents a versatile and widely used platform for embedded system prototyping and research. It provides access to the capabilities of the high-performance **STM32F767ZIT6** microcontroller, which integrates a rich set of peripherals, including Ethernet support essential for time-synchronized communication. In addition to the hardware capabilities of the STM32 Nucleo platform, effective utilization of the development board requires an appropriate software development environment. For STM32 Nucleo boards, the canonical and manufacturer-recommended development environment is **STM32CubeIDE**, which provides an integrated framework for project configuration, code development, and debugging. The environment is tightly coupled with the STM32 ecosystem and is designed to simplify peripheral configuration and application development on Nucleo platforms.

4.1 Hardware Architecture of the STM Nucleo-F767ZI Platform

The STM32 Nucleo-F767ZI development board is designed for testing and application development on microcontrollers from the **STM32F7** family produced by STMicroelectronics. The central component of the board is the STM32F767ZIT6 microcontroller based on the **ARM Cortex-M7** core, operating at a frequency of up to 216 MHz. The device includes 2 MB of flash memory and 512 kB of SRAM, which allows it to handle both basic and computationally demanding tasks.

The board integrates an on-board debugger **ST-LINK/V2-1**, which enables programming and debugging of the microcontroller without the need for an external hardware programmer. This simplifies firmware development and testing, which is important for an experimental implementation that requires repeated flashing, debugging, and verification of Ethernet behavior.

The STM32F767ZIT6 microcontroller also supports a wide range of communication interfaces, including **Ethernet MAC**, **USB OTG**, **USART**, **SPI**, **I^2C**, **CAN**, and various timer peripherals. The presence of a dedicated **Ethernet MAC** block makes this platform suitable for research in the field of network communication and precise time synchronization protocols such as PTP.

4.2 Key Hardware Components relevant to the implementation of the IEEE 1588 Precision Time Protocol

Figure 4.1 presents a simplified schematic representation of the STM32 Nucleo-F767ZI development board. The figure highlights hardware components that are involved in Ethernet communication, time synchronization, and platform extensibility.

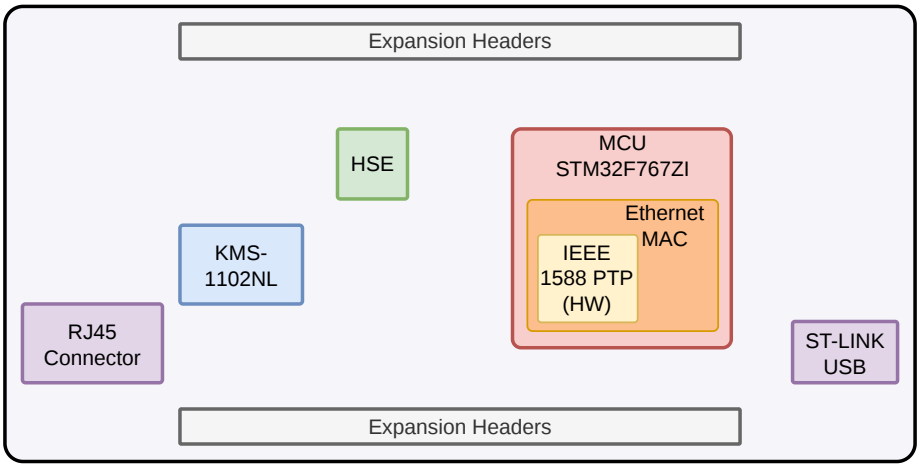


Fig. 4.1: Simplified schematic view of the STM32 Nucleo-F767ZI board highlighting components relevant to Ethernet communication and IEEE 1588 PTP synchronization

4.2.1 STM32F767ZI Microcontroller

The central component of the platform is the STM32F767ZI microcontroller. Of particular importance for this work is the integrated Ethernet peripheral, which includes an Ethernet MAC with hardware support for the IEEE 1588 Precision Time Protocol (PTP).

The Ethernet MAC incorporates dedicated logic for hardware timestamping of transmitted and received Ethernet frames. Timestamp generation is performed autonomously at the MAC level, without CPU intervention, allowing precise capture of transmission and reception events. This hardware-based approach significantly reduces the impact of software-induced latency and jitter on time synchronization accuracy.

4.2.2 Hardware Support for IEEE 1588 PTP

IEEE 1588 PTP support in the STM32F767ZI is implemented as part of the Ethernet peripheral. According to the STM32F7 reference manual, the Ethernet MAC includes timestamping functionality, a PTP system time counter, and registers for time initialization and correction [3].

Timestamping is performed at the boundary between the Ethernet MAC and the physical transmission medium, which represents the closest accessible point to the actual network interface. This design enables high-precision synchronization by minimizing uncertainties introduced by buffering, interrupt handling, and software processing.

4.2.3 Clocking and Time Base

The external crystal oscillator used on the board has a finite frequency tolerance. According to the oscillator datasheet, its frequency stability is approximately ± 150 ppm [4]. This value should not be interpreted as the final synchronization error of the implemented system, because PTP continuously estimates and compensates the difference between the local clocks. However, it illustrates why a one-time clock alignment is not sufficient. A frequency deviation of 150 ppm corresponds to a free-running drift of approximately 150 μs per second, which would be too large for sub-microsecond synchronization if it were not corrected periodically.

Therefore, the clock source defines the free-running behavior of each board. The role of the synchronization algorithm is to periodically estimate this drift and correct the local time base during operation.

4.2.4 Ethernet Physical Interface

The physical layer of the Ethernet interface is implemented using an external magnetic module (KMS-1102NL) and an RJ45 connector. The magnetic module provides galvanic isolation and impedance matching between the digital circuitry of the microcontroller and the Ethernet cable.

Although these components do not participate directly in timestamp generation or protocol processing, they define the electrical characteristics of the transmission path through which PTP messages are exchanged.

4.3 Software Development Environment

Software development was carried out using the integrated development environment **STM32CubeIDE**, officially provided by STMicroelectronics. This environ-

ment combines the microcontroller configuration tool **STM32CubeMX**, a compiler, source code editor, and debugger into a single interface.

STM32CubeIDE supports automatic project generation for a specific target board or microcontroller, including configuration of the clock tree, input/output ports, and peripheral modules. Peripheral control is primarily implemented through the **HAL (Hardware Abstraction Layer)** library, which provides hardware-independent access to microcontroller functionalities.

The choice of STM32CubeIDE was motivated by its multi-platform support, built-in integration with the Nucleo-F767ZI board, and compatibility with the ST-LINK debugger.

5 Comparison of libraries and approaches for working with Ethernet

As part of the preparation for the subsequent implementation of time synchronization according to IEEE 1588 (PTP), several approaches to Ethernet communication on the STM32 NUCLEO-F767ZI platform were considered:

1. standard STM32CubeIDE components: HAL Ethernet + lwIP,
2. direct work with the Ethernet peripheral using HAL,
3. the Mongoose Embedded library,
4. the FreeRTOS+TCP stack.

The comparison was not limited to a specific PTP transport variant. Instead, the approaches were evaluated from the point of view of their general suitability for implementing PTP with hardware timestamping on STM32F7. The main criteria were the possibility and complexity of implementing PTP message exchange, access to STM32 Ethernet MAC registers and IEEE 1588 hardware timestamping peripherals, multicast support, integration effort, documentation quality, and the availability of examples. A comparative overview of the considered approaches is provided in Table 5.1.

5.1 HAL + lwIP (STM32CubeIDE)

The HAL + lwIP combination is a standard and widely used solution for Ethernet communication on STM32 microcontrollers. HAL provides access to the Ethernet MAC, DMA descriptors, PHY configuration, and peripheral registers, while lwIP implements the TCP/IP stack, including IP, ICMP, UDP, TCP, and multicast-related functionality. This approach is well supported by STM32CubeIDE: the initial project structure, peripheral configuration, and middleware settings can be generated automatically, which simplifies the first stages of development and allows basic Ethernet functionality to be tested relatively quickly.

From the point of view of PTP implementation, this approach has several advantages. Since PTP can be transported over UDP/IP, lwIP can be used to handle IP addressing, UDP communication, and multicast traffic. This is useful when the system is expected to coexist with other IP-based services or when standard network tools are needed for debugging and diagnostics. In addition, HAL remains available for working with the low-level Ethernet peripheral, which is important for accessing IEEE 1588 hardware timestamping functionality.

Another advantage is the large amount of documentation, examples, and community experience. STM32CubeIDE, ST application examples, and the broader lwIP

community provide many reference points for configuring Ethernet, testing ping, creating UDP communication, and enabling multicast-related functionality. This reduces the risks associated with the initial configuration of the network stack.

However, lwIP does not provide a ready-to-use IEEE 1588 implementation. The PTP protocol logic, including message handling, timestamp processing, offset and delay calculation, and clock correction, must still be implemented separately or ported from an existing PTP stack. In addition, high-precision PTP requires that hardware transmit and receive timestamps be reliably connected with the corresponding PTP event messages. Since lwIP abstracts packets through its own buffer structures and network interface layer, additional integration work may be needed to pass timestamp information between the Ethernet driver and the PTP logic.

Thus, HAL + lwIP is a strong option for IP-based network applications and for PTP variants transported over UDP/IP. Its main disadvantages are the absence of built-in PTP support and the additional work required to integrate precise hardware timestamping into the stack data path.

5.2 Direct Ethernet handling using HAL

Another approach is to use HAL and the STM32 Ethernet peripheral directly, without adding a full TCP/IP stack. In this case, the developer works with Ethernet transmission and reception at a lower level: Ethernet frames, DMA descriptors, MAC configuration, PHY settings, interrupts, and IEEE 1588 timestamping registers are controlled directly from the firmware.

The main advantage of this approach is full access to the STM32F767 Ethernet hardware. PTP accuracy strongly depends on the ability to obtain timestamps as close as possible to the actual moment of frame transmission or reception. Direct work with HAL and MAC registers makes it possible to configure the Ethernet MAC time base, enable timestamping, read transmit and receive timestamps, adjust the hardware time counter, and control the timestamp addend used for clock frequency correction. This makes the approach well suited for studying and implementing the low-level mechanisms required by IEEE 1588.

Another advantage is that unnecessary protocol layers can be avoided. If the target system does not require IP-based services, then a TCP/IP stack may add complexity, memory consumption, and additional abstraction without being essential for the synchronization task. Direct Ethernet handling allows the firmware to focus only on the required communication and timing functionality.

This approach also gives the developer complete control over the packet format. PTP may be implemented either with a custom low-level frame processing path or with direct handling of standard PTP Ethernet frames. Such flexibility is useful in

systems where all nodes are located in the same local Ethernet segment and where IP address assignment is not required for the synchronization mechanism.

The main disadvantage is the increased amount of manual development work. Without a TCP/IP stack, the developer must implement packet construction, parsing, filtering, buffer management, error handling, and protocol state processing manually. There is also a higher risk of errors related to low-level Ethernet details, such as DMA descriptor ownership, cache coherency, interrupt handling, MAC filtering, and timestamp validity. In addition, this approach does not provide general network functionality such as ICMP, DHCP, TCP, or HTTP unless it is implemented separately.

Therefore, direct HAL-based Ethernet handling provides the best low-level control and the most transparent access to IEEE 1588 hardware features, but it requires more manual implementation effort and is less convenient for applications that also need general-purpose IP networking.

5.3 Mongoose Embedded

Mongoose Embedded is a lightweight networking library for C/C++ embedded systems. It is commonly used for rapid development of network services such as HTTP servers, WebSocket interfaces, MQTT communication, and other IoT-oriented protocols. Depending on the target platform and configuration, Mongoose can operate on top of an existing TCP/IP stack or use its own networking implementation and low-level drivers.

The main advantage of Mongoose is development simplicity. It provides a compact event-driven API and a large number of ready-made examples. This makes it convenient for quickly demonstrating Ethernet connectivity on an embedded board. For example, a simple HTTP server can be implemented with relatively little code, allowing a browser-based interface to interact with the microcontroller.

From the PTP point of view, Mongoose is less specialized. It does not provide a built-in IEEE 1588 implementation and is not primarily intended for high-precision time synchronization. PTP message handling, timestamp processing, offset calculation, and clock correction would still have to be implemented separately. If PTP were implemented over UDP, Mongoose could simplify some parts of network communication, but the precision-critical timestamping functionality would still require low-level access to the Ethernet MAC and its IEEE 1588 registers.

A further limitation is that Mongoose is mainly oriented toward application-layer network services. Although it is useful for HTTP, MQTT, and similar protocols, PTP requires close interaction with the Ethernet driver and hardware timestamping mechanism. Therefore, additional modifications to the low-level driver or platform

port would likely be needed in order to associate PTP event messages with precise hardware transmit and receive timestamps.

The documentation and examples for Mongoose are clear and convenient for learning the library. However, most examples are focused on application-level communication rather than on hardware timestamping or IEEE 1588-specific functionality. As a result, Mongoose is suitable for rapid network prototyping and demonstration interfaces, but it is less suitable as the main basis for implementing precise PTP on STM32F7.

5.4 FreeRTOS+TCP

FreeRTOS+TCP is a TCP/IP stack designed for close integration with FreeRTOS. It provides IP, UDP, TCP, and socket-like communication mechanisms in a way that fits naturally into an RTOS-based architecture. Networking can be organized using tasks, queues, semaphores, and priorities, which is useful in larger embedded systems where network communication must coexist with other real-time tasks.

The main advantage of FreeRTOS+TCP is its architectural consistency with FreeRTOS. If the firmware is already based on FreeRTOS, the stack can provide a structured way to separate network processing from application logic. This can improve maintainability in complex projects and allows the developer to use RTOS mechanisms for synchronization and task scheduling.

For PTP implementation, FreeRTOS+TCP can be considered when PTP is transported over UDP/IP and when the rest of the application already depends on FreeRTOS. In such a case, the stack may be used for UDP communication and multicast-based message exchange, while the PTP state machine and clock correction algorithm are implemented at the application level.

However, FreeRTOS+TCP does not provide a ready-made IEEE 1588 implementation. As with the other approaches, the complete PTP logic must be implemented or ported separately. In addition, precise hardware timestamping requires extending the network interface layer so that transmitted and received packets can be associated with timestamps from the Ethernet MAC hardware. This increases integration complexity.

Another disadvantage is the manual porting effort for STM32 Ethernet. Compared with the HAL + lwIP configuration generated by STM32CubeIDE, FreeRTOS+TCP often requires more manual work to integrate the network interface, configure buffer management, and adapt the driver to the specific hardware. Multicast support and IGMP-related behavior may also require additional configuration or refinement depending on the selected stack version and project settings.

Thus, FreeRTOS+TCP is a reasonable option for RTOS-oriented TCP/IP applications, especially when FreeRTOS is already a central part of the firmware architecture. However, for PTP with hardware timestamping on STM32F7, it still requires significant additional work at the driver and timestamp integration level.

5.5 Summary of the Comparison

Based on the comparison, the practical implementation was performed using direct Ethernet handling through HAL and STM32 Ethernet MAC registers. The selected implementation transmits PTP messages as raw Ethernet frames identified by EtherType `0x88F7`. This transport variant is suitable for the developed system because all boards are connected within the same local Layer 2 Ethernet network, where IP addressing is not required for the synchronization task.

Avoiding IP and UDP removes the need for address assignment, ARP processing, UDP sockets, and TCP/IP stack configuration. At the same time, direct HAL-based Ethernet handling provides access to the STM32F767 Ethernet MAC mechanisms required for IEEE 1588 operation. According to the STM32F7 reference manual, the Ethernet peripheral provides hardware timestamping, a PTP system time counter, time update registers, an addend register, and DMA descriptor fields for transmit and receive timestamps [3]. These mechanisms are necessary for precise timestamp acquisition and local clock correction.

This approach is also compatible with STM32CubeIDE: the toolchain is still used to configure the Ethernet peripheral, GPIOs, clock tree, and the basic HAL project structure, while the PTP-specific frame handling and timestamp processing are implemented manually. Therefore, direct HAL-based Ethernet communication was selected as the most suitable approach for the implemented PTP system. It avoids unnecessary networking overhead while preserving the required level of control over raw Ethernet frame transmission, reception, hardware timestamping, and clock correction.

Table 5.1: Comparison of Ethernet approaches for PTP implementation

Criterion	**HAL + lwIP**	**Direct HAL Ethernet handling**	**Mongoose Embedded**	**FreeRTOS+TCP**
Primary purpose	General TCP/IP communication on STM32 with ST ecosystem support	Low-level Ethernet communication and direct peripheral control	Rapid development of embedded network services and IoT interfaces	TCP/IP communication integrated with FreeRTOS
Integration into STM32CubeIDE	High: project generation and middleware configuration are supported	High for peripheral initialization; protocol logic is implemented manually	Medium: external library and platform configuration are required	Medium/low: manual network-interface integration is often required
Documentation and examples	Very high: ST materials, lwIP examples, and broad community support	Medium: HAL documentation is available, but complete PTP examples with hardware timestamping are limited	High for HTTP/MQTT/WebSocket examples; limited for IEEE 1588 use cases	Medium: official documentation is available, but fewer STM32-specific examples
Access to STM32 registers and IEEE 1588 peripheral	Good through HAL and driver-level code, but lwIP adds an abstraction layer	Very high: direct access to MAC registers, DMA descriptors, and timestamping logic	Limited unless the low-level driver or port is modified	Possible, but requires integration through the network-interface layer
Hardware timestamp integration	Possible, but requires connecting driver timestamps with lwIP packets	Most direct and transparent	Requires additional low-level modifications	Requires extension of packet metadata and driver code
Multicast support	Supported depending on lwIP configuration	Possible through direct MAC filtering and manual frame processing	Depends on selected configuration and driver support	Possible, but may require additional configuration or refinement
Need for IP addressing	Usually required for normal stack operation	Not required if raw Ethernet frames are used	Usually required for typical use cases	Usually required for normal stack operation
Main advantages	Convenient TCP/IP support, many examples, easy testing with standard network tools	Full hardware control, low overhead, suitable for precise timestamp handling	Simple API, fast prototyping, convenient for demonstration services	Natural RTOS integration, task-based architecture, socket-like API
Main disadvantages	No built-in PTP; timestamp integration requires extra work	More manual implementation work; no general TCP/IP services by default	Not focused on hardware timestamping or PTP precision	No built-in PTP; more complex STM32 driver integration
Overall suitability for PTP with hardware timestamping	High for UDP/IP-based PTP with additional driver integration	High for low-level and raw Ethernet-based PTP implementations	Medium/low; useful mainly for auxiliary network services	Medium; suitable mainly when FreeRTOS-based TCP/IP architecture is required

6 Description of the PTP Implementation

This chapter describes the firmware developed for time synchronization between STM32 Nucleo-F767ZI boards using the IEEE 1588 Precision Time Protocol. The PTP operating principle and the meaning of the four timestamps were introduced in Chapter 2 and Section 1.3. The implementation described here follows the software approach selected in Chapter 5: PTP messages are transported directly in Layer 2 Ethernet frames with EtherType `0x88F7`, while the STM32F767ZI Ethernet MAC provides hardware timestamps for transmitted and received event messages.

The developed software is not a general-purpose complete PTP stack with automatic grandmaster election. Instead, it is a focused embedded implementation of the subset needed for the thesis experiments: one STM32 board is configured as the master and one or more STM32 boards are configured as slaves. This fixed-role architecture removes the need for the BMCA and makes the synchronization process easier to analyze: the master always generates the reference time, while each slave only performs delay measurement, offset calculation, and local clock correction. At the same time, the implementation still preserves the essential PTP message exchange: `Sync`, `Follow_Up`, `Delay_Req`, and `Delay_Resp`.

6.1 Overall Firmware Architecture

The final firmware consists of two variants. The master periodically transmits the reference time and answers delay requests. The slave receives the reference time, performs end-to-end delay measurement, estimates its offset from the master, rejects invalid samples, and corrects its local PTP clock.

The complete synchronization exchange is summarized in Figure 6.1. The important point is that the slave does not correct its clock from a single received packet. A valid correction is made only after the full timestamp set t_1, t_2, t_3, t_4 has been obtained from the end-to-end PTP exchange described earlier in Section 2.1.

Both firmware variants are based on STM32CubeIDE generated projects. HAL is still used for Ethernet initialization, packet transmission and reception, and callback registration. The PTP-specific communication path is implemented manually, because the firmware must control raw Ethernet frame contents, MAC addressing, timestamp request bits in DMA descriptors, and the STM32 Ethernet timestamping unit directly. This corresponds to the hardware-assisted implementation principle discussed in Section 2.3.2: timestamping is performed by the Ethernet hardware, while protocol logic and clock correction are implemented in software.

The master uses the fixed MAC address `00:80:E1:00:00:01`. The slave boards are based on the same IDE project, but each board is flashed with a slightly modified

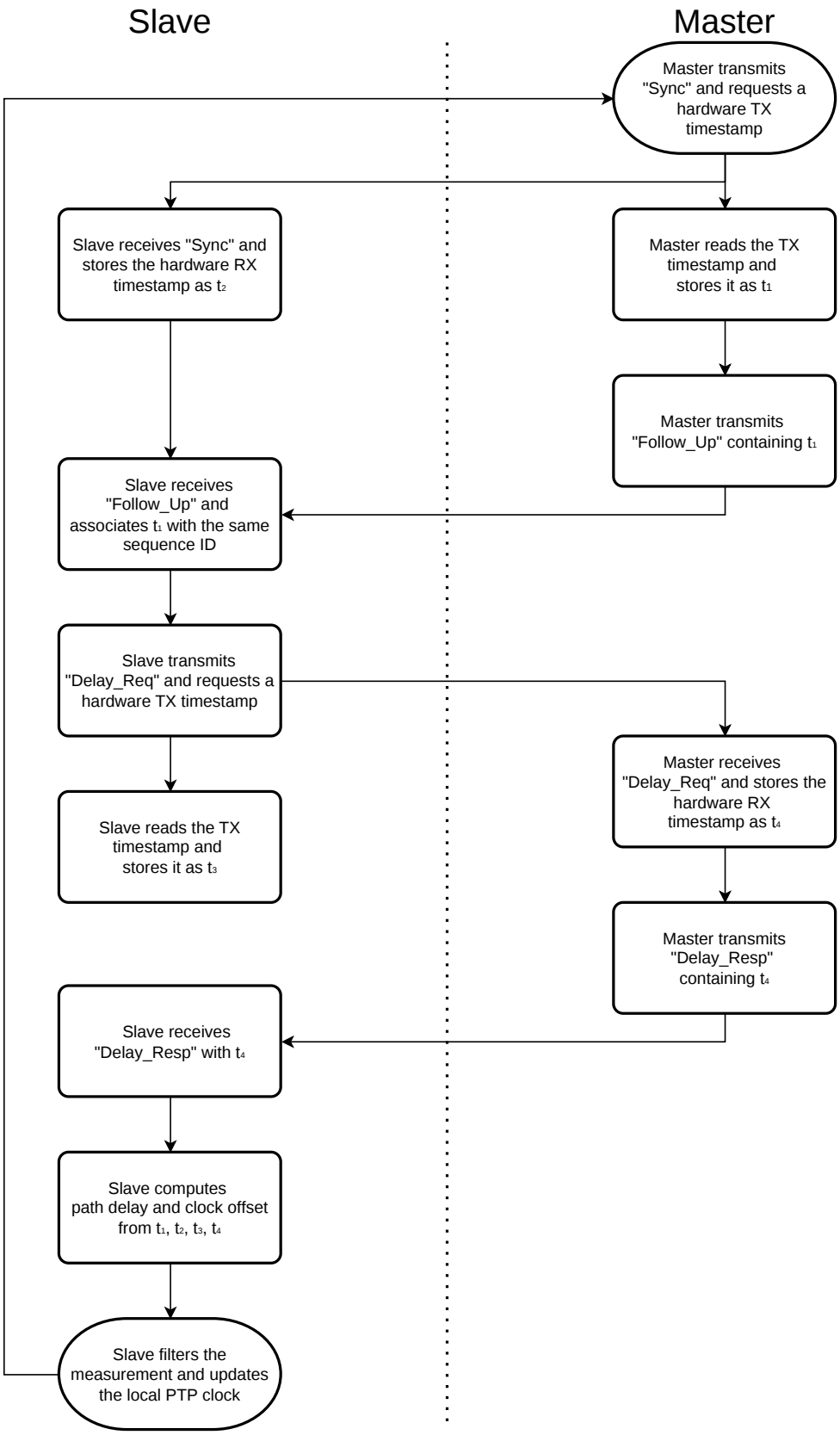


Fig. 6.1: Flowchart of the implemented end-to-end PTP synchronization procedure.

Table 6.1: Main roles in the implemented PTP system.

Node	**Role**	**Main firmware responsibilities**
Master STM32	Reference clock	Transmits the reference time, timestamps outgoing event frames, receives delay requests, and sends delay responses.
Slave STM32	Controlled clock	Receives master timestamps, sends delay requests, computes offset and path delay, filters samples, and corrects the local PTP clock.
Ethernet switch	Optional interconnect	Forwards Layer 2 frames in the multi-slave tests; it is not PTP-aware in the tested baseline.
PC	Measurement support	Captures UART logs, plots measured data, and can generate background Ethernet traffic for stress tests.

firmware build in which `SLAVE_ID_LAST_BYTE` is adjusted manually. This byte acts as a simple per-board identifier and is included in both the Ethernet address and the PTP port identity. This keeps the project reusable for several slave boards while still allowing each slave to be distinguished during `Delay_Req`/`Delay_Resp` processing.

The PTP multicast MAC address `01:1B:19:00:00:00` is used for multicast PTP traffic inside the local Layer 2 domain. In the final multi-slave version, `Delay_Resp` is sent as a unicast frame to the slave that generated the corresponding request. This prevents a slave from processing timing information that belongs to another board. The main roles of the nodes used in the implementation are summarized in Table 6.1.

6.2 Hardware Timestamping Implementation

The most important low-level part of the firmware is the acquisition of hardware timestamps from the STM32 Ethernet peripheral. Precise PTP synchronization cannot rely on the time when firmware processes a packet. The timestamp must correspond to the moment when the Ethernet MAC actually transmits or receives the PTP event frame.

The STM32F767 Ethernet peripheral provides IEEE 1588 hardware timestamping support. The STM32F7 reference manual describes timestamping through the Ethernet PTP registers and through timestamp fields associated with Ethernet

DMA descriptors [3]. At the beginning of the implementation, the HAL functions intended for working with Ethernet hardware timestamps were tested. In the tested configuration, the HAL timestamp access path did not provide the required hardware timestamps. Therefore, HAL was kept for initialization, packet transfer, and callbacks, but the timestamp path itself was implemented by direct access to Ethernet DMA descriptors and PTP registers.

The PTP hardware is initialized by enabling timestamping, PTPv2 message recognition, PTP-over-Ethernet support, subsecond rollover mode, and fine update mode. The relevant part of the real initialization code is shown in Listing A.1. This code configures the Ethernet PTP timestamp control register directly.

The time base uses a subsecond increment of 5 ns with a system clock of 216 MHz. The default addend is calculated from

$$A_0 = \left\lfloor \frac{2^{32} \cdot 10^9}{f_{HCLK} \cdot I_{SS}} \right\rfloor, \tag{6.1}$$

where f_{HCLK} is the STM32 high-speed clock frequency and I_{SS} is the PTP subsecond increment in nanoseconds. The subsecond increment defines the size of one PTP time-base update step. The addend controls how often these update steps are generated by the hardware accumulator in fine update mode. In fine correction mode, the addend register is accumulated by the hardware, and the accumulator carry generates update pulses for the system time counter [3]. Therefore, A_0 represents the nominal clock rate, while the current addend can later be increased or decreased by the slave frequency servo.

A specific practical issue of the STM32F7 Ethernet peripheral is the representation of the low time-update register. The firmware cannot write raw nanoseconds directly to this register. For time initialization and phase updates, a nanosecond value is converted into the STM32 subsecond update format. In this representation, one second corresponds mathematically to 2^{31} subsecond units, written in hexadecimal as `0x80000000`. The most significant bit is used as the sign bit for negative time updates, while the remaining bits contain the positive subsecond magnitude [3]. Therefore, the positive magnitude written to the register is limited to `0x7FFFFFFF`. Listing A.2 shows the helper function used for this conversion.

For received PTP event messages, the timestamp is read from the RX descriptor that belongs to the received frame. The HAL receive callback is entered after the descriptor index has already advanced, therefore the firmware first moves the index one position back and then reads descriptor words `d[6]` and `d[7]`. Listing A.3 shows the real code fragment used for this access.

For transmitted PTP event messages, the firmware explicitly requests a hardware transmit timestamp before the frame is passed to the Ethernet driver. This is

done by setting bit 25 in the selected TX descriptor. After the DMA has returned ownership of the descriptor, the firmware checks the timestamp status bit and reads the timestamp fields directly from the same descriptor. Listing A.4 shows the real code used on the slave for the `Delay_Req` transmit timestamp t_3. The master uses the same principle for the `Sync` transmit timestamp t_1.

This descriptor-based access was the decisive implementation point. The HAL timestamp functions did not provide the required hardware timestamps in this firmware, so the final implementation obtains the event timestamps from DMA descriptor fields directly. As a result, the timestamps used by the protocol are taken at the Ethernet MAC level: t_1 for the master's `Sync` transmission, t_2 for the slave's `Sync` reception, t_3 for the slave's `Delay_Req` transmission, and t_4 for the master's `Delay_Req` reception.

The firmware also enables the Ethernet PPS output by writing `ETH->PTPPPSCR = 0x0U`. This provides a 1 Hz signal on the `ETH_PPS_OUT` pin and allows the master and slave clocks to be compared externally with an oscilloscope.

6.3 PTP Frame Handling

The firmware does not use lwIP, UDP sockets, or an external PTP library. All PTP traffic is handled as raw Layer 2 Ethernet frames, which follows the direct Ethernet approach justified in Section 5.2. Each frame starts with an Ethernet header: destination MAC address, source MAC address, and EtherType `0x88F7`. The PTP common header begins immediately after the Ethernet header.

The important design decision is that event messages are transmitted through the same HAL Ethernet path as normal frames, but with explicit descriptor-level timestamp requests added before transmission. General messages do not require transmit timestamps and are used to carry the timestamps that were already captured by hardware.

The implementation uses two-step synchronization. The master sends a `Sync` frame first, waits until the hardware TX timestamp has been captured, and then sends `Follow_Up` with the exact timestamp value. This avoids inserting an approximate firmware timestamp into the `Sync` frame before the actual transmission time is known as described in Section 1.3.

6.4 Master Firmware

The master firmware acts as a fixed reference clock for the experimental system. It does not contain a servo because its PTP time base is treated as the reference

time. Its tasks are limited to periodic two-step synchronization, reception of delay requests, and generation of delay responses. The resulting workflow is shown in Figure 6.2. The master main loop is a compact scheduler. It first checks whether the TX timestamp for the latest `Sync` became available. If a timestamp is available, it sends the corresponding `Follow_Up`. It then processes queued `Delay_Req` events and sends the required `Delay_Resp` frames. Finally, if no timestamp or follow-up operation is pending and the synchronization period has elapsed, the next `Sync` frame is transmitted. Completed TX buffers are released at the end of the loop.

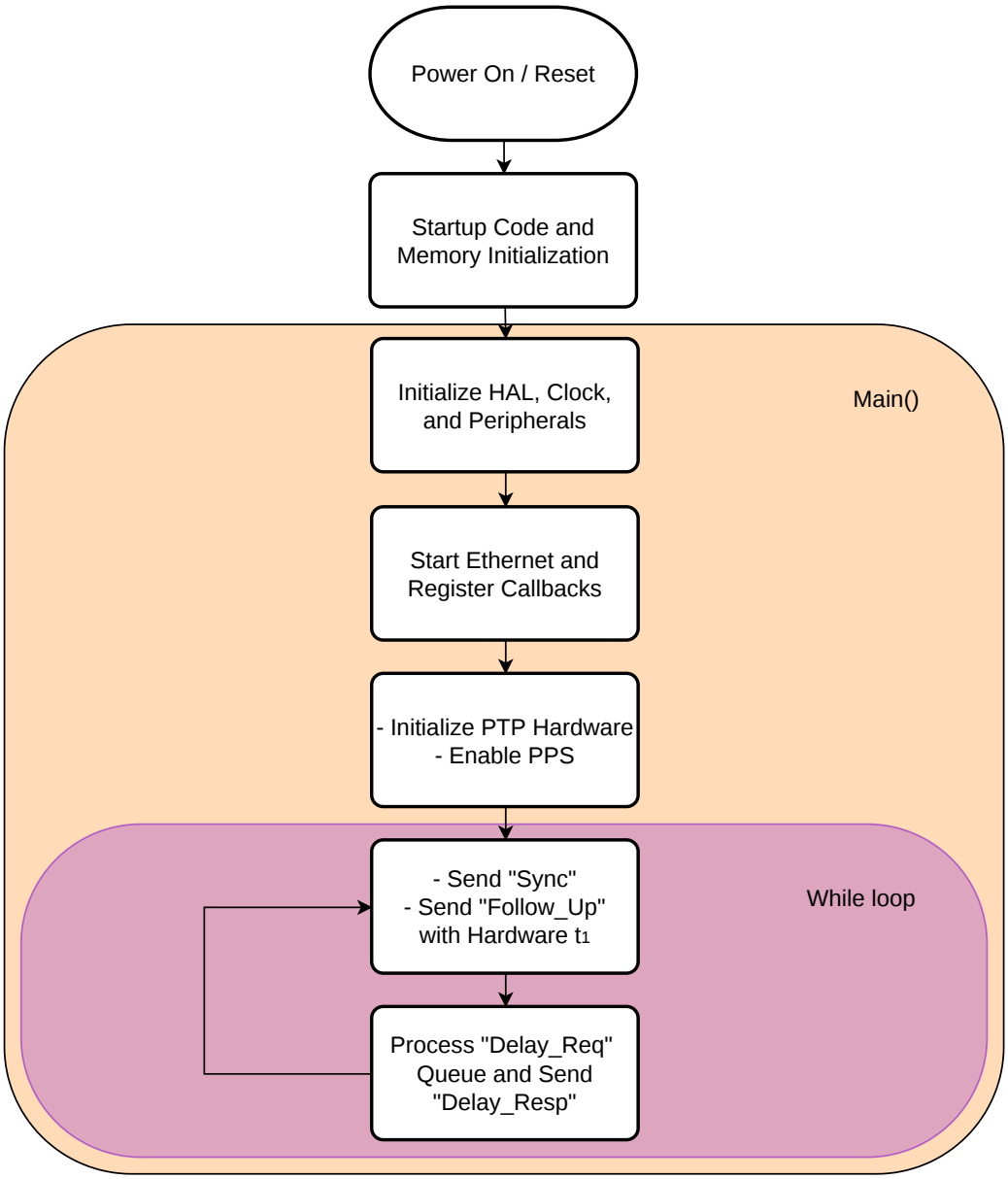


Fig. 6.2: Flowchart of the master firmware workflow.

6.4.1 Two-Step Sync Generation

The master periodically sends a `Sync` frame. In the current configuration the synchronization period is 125 ms. Before the frame is transmitted, the master stores the current TX descriptor index and sets the timestamp request bit in that descriptor. After transmission, the descriptor is checked until ownership returns to the CPU and the timestamp status bit is set. The captured seconds and nanoseconds fields are then stored as t_1, and a matching `Follow_Up` message with the same sequence ID is sent.

This separation is necessary because the precise transmission time is not known when the `Sync` frame is built. The `Sync` message only starts the timing event, while `Follow_Up` carries the hardware timestamp captured after the frame has actually been transmitted by the Ethernet MAC.

6.4.2 Delay Request Queue for Multi-Slave Operation

In the initial single-slave version, the master could store one pending `Delay_Req` event in a global structure. This was sufficient for a direct two-board test, but it was not safe for a topology with more than one slave. If two slaves sent `Delay_Req` messages before the main loop transmitted the first `Delay_Resp`, the later request could overwrite the earlier timing information.

The final master firmware therefore uses a small ring queue for received delay requests. Each queue item stores the request sequence ID, the hardware RX timestamp t_4, the requesting port identity, and the source MAC address of the requesting slave. The receive callback only parses the frame, reads the RX descriptor timestamp, and pushes the event into the queue. The main loop later removes items from the queue and sends a unicast `Delay_Resp` to the stored destination MAC address. In the current baseline, the queue length is 16 and the master processes up to four delay responses per loop iteration.

This queue-based structure is the main change that made the master usable in the one-master, multi-slave switch topology. It prevents timing information from different slaves from being mixed and keeps the interrupt-level receive path short.

6.5 Slave Firmware

The slave firmware contains the main synchronization algorithm. It receives and matches PTP messages, freezes the relevant timestamp pair for a delay measurement, generates `Delay_Req`, obtains the hardware TX timestamp, calculates path delay and offset, rejects invalid samples, and updates the local PTP clock. The workflow is shown in Figure 6.3.

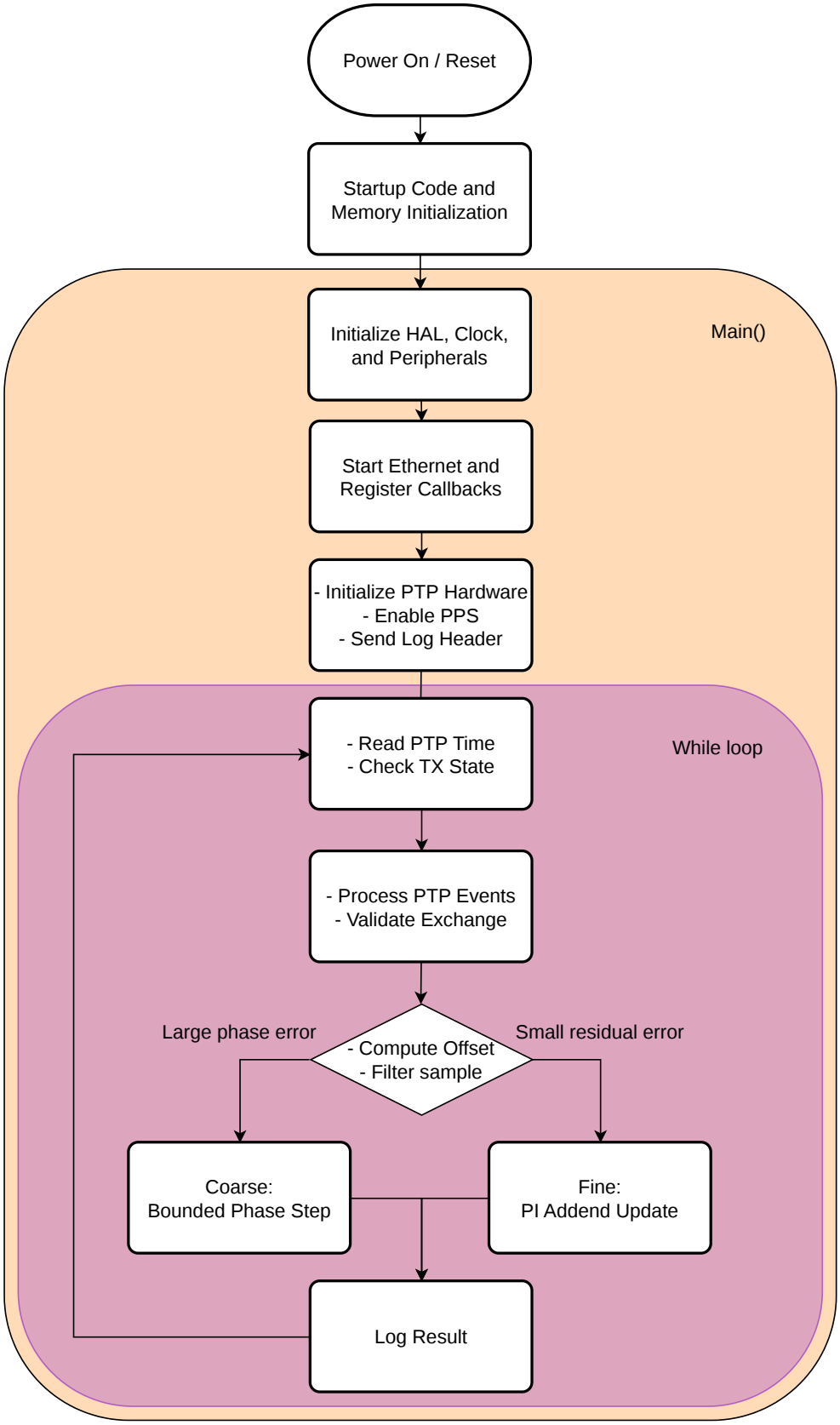


Fig. 6.3: Flowchart of the slave firmware workflow.

6.5.1 Slave Identity and Frame Filtering

The slave project is parameterized by `SLAVE_ID_LAST_BYTE`. This value is used in the slave MAC address and in the PTP port identity. During flashing, the value is adjusted manually for each slave board. The final firmware also checks `Delay_Resp` frames by destination MAC address and by `requestingPortIdentity`. This prevents one slave from accepting a response intended for another slave in a multi-slave topology.

The Ethernet MAC address generated by STM32CubeIDE is overwritten during Ethernet initialization. The PTP hardware initialization then configures filtering so that the slave receives frames addressed to its own unicast MAC address and to the PTP multicast address, without accepting unrelated Ethernet traffic as valid PTP input.

6.5.2 Receive Callback and Event Structures

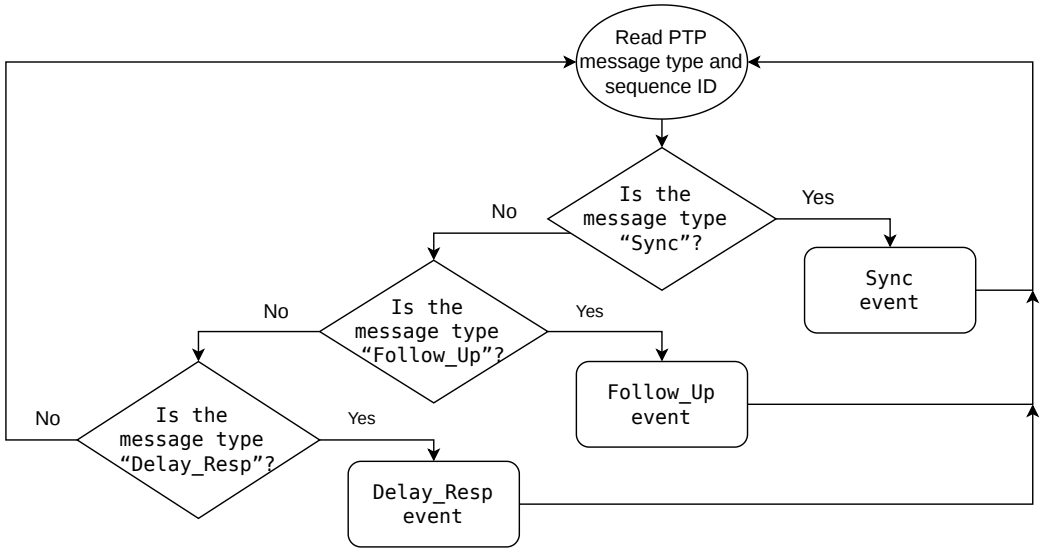


Fig. 6.4: Slave receive callback processing.

The slave receive callback reads all available Ethernet buffers in a loop as shown in Figure 6.4. This is important in switch-based and multi-slave operation because several frames can be queued before the callback is entered. Each received frame is filtered by EtherType `0x88F7`. The callback then reads the PTP message type and sequence ID, takes the RX timestamp from the descriptor when it is available, and updates a small pending-event structure.

For `Sync`, the slave stores the sequence ID and the hardware RX timestamp t_2. For `Follow_Up`, it parses the origin timestamp t_1. For `Delay_Resp`, it verifies that

the destination identity matches this slave and then parses t_4. The callback does not run the clock servo. It only stores the received event information so that the main processing function can consume it deterministically.

6.5.3 Delay Request Generation and Timestamp Acquisition

After a matching `Sync`/`Follow_Up` pair is received, the slave freezes the corresponding t_1 and t_2 values for the next delay measurement. This freezing step is critical: without it, a newly received `Sync` could overwrite t_2 while the firmware is still waiting for the previous `Delay_Resp`. Such a state would mix timestamps from two different synchronization cycles.

The `Delay_Req` frame is then transmitted as a PTP event message with a hardware TX timestamp request. The timestamp request and timestamp readback are implemented through the TX descriptor access shown earlier in Listing A.4. If a matching `Delay_Resp` does not arrive within the configured timeout, the exchange is dropped and the next clean `Sync`/`Follow_Up` pair is used.

6.5.4 End-to-End Offset and Path Delay Calculation

When a matching `Delay_Resp` is received and t_3 is available, the slave converts the four timestamps into nanoseconds and performs the standard end-to-end calculation introduced in Section 2.1. The quantities used in the firmware are:

$$\Delta_{sync} = t_2 - t_1, \tag{6.2}$$

$$\Delta_{delay} = t_4 - t_3, \tag{6.3}$$

$$\text{meanPathDelay} = \frac{\Delta_{sync} + \Delta_{delay}}{2}, \tag{6.4}$$

$$\text{offset}_{raw} = \frac{\Delta_{sync} - \Delta_{delay}}{2}. \tag{6.5}$$

The raw offset can be shifted by the constant `PTP_STATIC_OFFSET_BIAS_NS`. In the current baseline this value is zero, but the define remains in the code because external PPS measurements can reveal a stable systematic bias. Listing A.5 shows the corresponding core calculation in the slave firmware.

For the frequency servo, the firmware also forms two comparable time values:

$$\begin{aligned} T_M(k) &= t_1(k) + \text{meanPathDelay}(k), \\ T_S(k) &= t_2(k). \end{aligned} \tag{6.6}$$

Here T_M is the estimated master time at the instant of slave reception, and T_S is the slave timestamp of that same event. Their evolution between accepted samples is used by the PI frequency controller.

6.5.5 Filtering and Rejection of Bad Measurements

A practical PTP servo must not react to every computed measurement. Switch buffering, background traffic, missing timestamps, or delayed responses can produce individual samples that are physically implausible. If such a sample is allowed to enter the servo, it can create an unnecessary phase step or corrupt the PI integrator. For this reason, the slave applies a reject filter before updating the clock.

The current baseline includes three rejection conditions. First, the estimated mean path delay must be inside a configured interval. In the multi-slave through-switch baseline, this interval is 6000 ns to 10000 ns. Second, in fine mode, an extremely large absolute offset is rejected if it is not large enough to intentionally force a coarse-mode re-entry. Third, in fine mode, a sudden jump relative to the last accepted raw offset is rejected. The corresponding thresholds are 3 ms for the fine-mode absolute offset and 200 μs for sample-to-sample jumps. If too many consecutive samples are rejected, the slave executes a forced resynchronization, clears the pending message state, returns to coarse mode, and waits for a new clean `Sync`/`Follow_Up` pair.

After a sample is accepted, the slave updates a first-order IIR filtered offset. Let o_k denote the newly accepted raw offset sample at synchronization cycle k, and let $\overline{o}_k$ denote the filtered offset after this sample. The implemented filter is

$$\overline{o}_k = \frac{7\overline{o}_{k-1} + o_k}{8} = \alpha o_k + (1-\alpha)\overline{o}_{k-1}, \qquad \alpha = \frac{1}{8}. \tag{6.7}$$

The value $\alpha = 1/8$ was selected as a simple compromise between noise suppression and response speed. It reduces the influence of individual accepted samples while still allowing the PI controller to follow slow clock drift. The power-of-two denominator is also convenient for integer implementation on the microcontroller. The filtered value $\overline{o}_k$ is used by the PI controller and is logged over UART.

6.6 Clock Correction Strategy

The final slave correction algorithm is a two-level servo. Large phase errors are corrected by bounded time updates. Once the clock is close enough to the master, fine synchronization is maintained by changing the PTP addend, which adjusts the frequency of the local PTP clock. This separation is important because a frequency servo alone would need too much time to eliminate a large startup error, while

continuous hard phase steps would create jitter after the system is already close to synchronization.

This division follows the two correction mechanisms provided by the STM32 Ethernet PTP unit. The reference manual distinguishes between coarse correction, where the time update value is added to or subtracted from the system time, and fine correction, where frequency drift is compensated over time using the addend register [3]. In the implemented servo, coarse regulation is phase regulation performed through bounded time updates, while fine regulation is frequency regulation performed through addend adjustment.

6.6.1 Fine Frequency Correction Using Addend

After the slave enters fine mode and the holdoff counter expires, the fine frequency servo becomes active. In this mode the firmware does not directly change the current PTP time. Instead, it changes the addend of the STM32 PTP time base, and therefore changes the rate at which the slave PTP clock runs.

The controller does not use the raw offset sample directly. First, every accepted raw offset o_k is passed through the first-order IIR filter introduced in Section 6.5.5. It determines how strongly a newly accepted offset sample affects the filtered value. With this value, only one eighth of the new sample is added at each update, while seven eighths come from the previous filtered offset. The result is the logged variable `offset_avg_ns`, which is then used as the input of the PI frequency controller.

Based on the estimated master and slave reception times defined in Equation 6.6, the firmware computes the diagnostic frequency error shown in Equation 6.8. This value compares how much master time and slave time elapsed between two accepted synchronization samples and expresses their relative rate difference in parts per billion.

$$\begin{aligned} M_k &= T_M(k) - T_M(k-1), \\ S_k &= T_S(k) - T_S(k-1), \\ e_{f,k} &= \tfrac{(M_k - S_k)\cdot 10^9}{S_k}. \end{aligned} \tag{6.8}$$

Here, M_k is the elapsed estimated master time, S_k is the elapsed slave time, and $e_{f,k}$ is logged as a diagnostic frequency error in ppb. It is useful for observing the behavior of the slave clock, but the actual PI correction is based on the filtered offset $\overline{o}_k$. Before the controller is evaluated, a small deadband is applied as shown in Equation 6.9.

$$e_k = \begin{cases} 0, & |\overline{o}_k| < 30\,\text{ns}, \\ \overline{o}_k, & \text{otherwise.} \end{cases} \tag{6.9}$$

The deadband prevents the controller from reacting to very small residual noise around zero. The sign convention in the firmware is that a positive offset means that the slave clock is ahead of the master and must therefore be slowed down.

The PI output is expressed in parts per billion. The firmware uses fixed-point scaling internally, but conceptually the controller output can be written as

$$u_{\mathrm{ppb}}(p) = -F_{\mathrm{PI}}(p)E(p), \tag{6.10}$$

where $E(p)$ is the filtered offset error after the deadband, and the minus sign represents the firmware sign convention. The output u_{ppb} is then converted into the target addend:

$$A_{\mathrm{target}} = A_0 \left(1 + \frac{u_{\mathrm{ppb}}}{10^9}\right), \tag{6.11}$$

where A_0 is the default addend. Before being written to the hardware, the addend change is limited. In the final configuration, one update step is limited to 2000 addend units, and the total allowed deviation from the default addend is limited to ± 1000000 units. These limits prevent one bad or unusually large sample from producing an excessive frequency correction.

In standard controller notation, the implemented frequency controller can therefore be represented as a PI regulator:

$$F_{\mathrm{PI}}(p) = K_P \left(1 + \frac{1}{T_i p}\right) \tag{6.12}$$

The proportional coefficient used in the firmware is

$$K_P = \frac{1}{40}\,\frac{\mathrm{ppb}}{\mathrm{ns}}. \tag{6.13}$$

The integral coefficient used in the discrete firmware implementation is

$$K_{I,d} = \frac{1}{2500}\,\frac{\mathrm{ppb}}{\mathrm{ns}} \tag{6.14}$$

per nominal synchronization cycle. Since the nominal cycle time is $T_0 = 125\,\mathrm{ms}$, this discrete coefficient corresponds to the integral time

$$T_i = \frac{K_P}{K_{I,d}} T_0 = \frac{1/40}{1/2500} \cdot 0.125 = 7.8125\,\mathrm{s}. \tag{6.15}$$

Therefore, the implemented controller can be written in the standard PI form as

$$F_{\mathrm{PI}}(p) = \frac{1}{40}\left(1 + \frac{1}{7.8125p}\right). \tag{6.16}$$

The K_P and K_I coefficients for the implemented controller were determined experimentally.

6.6.2 Experimental Options That Were Not Kept in the Baseline

Several design ideas were implemented or evaluated during development but were not kept as active baseline mechanisms. They are still useful for understanding why the final algorithm has its current form.

First, a direct phase step to zero was rejected. It was simple, but it caused overshoot and repeated transitions between coarse and fine regions. Second, the first integer-ppb PI integrator was replaced by the scaled integrator described above, because small integration increments were rounded to zero and the servo could not reliably remove the residual error. Third, a fine phase capture mechanism was tested. Its purpose was to apply very small phase corrections when the frequency servo was already close to lock, but it did not provide a consistent improvement on the tested setup and sometimes added unnecessary dynamics. Therefore, it remains disabled in the baseline. Fourth, a warm-start mechanism based on a stored addend was explored, but it did not produce a stable enough improvement to justify its complexity in the final configuration.

6.7 Logging and Measurement Support

The slave firmware prints one CSV line after every newly processed `Delay_Resp`. The header is printed once at startup and includes the estimated offset, raw offset, filtered offset, mean path delay, frequency error, addend state, phase and frequency update counters, timestamp counters, delay-request timeout counters, ignored-response counters, PI internal values, reject counters, consecutive reject count, and resynchronization count.

The logging is deliberately detailed. It does not only show the final synchronization error; it also shows how the servo produced that result. For example, `current_addend` and `last_addend_step` show the action of the frequency controller, while `reject_mpd_count` and `reject_jump_count` show whether the network path is producing invalid samples. This made it possible to compare direct cable, switch-based, multi-slave, and traffic-loaded scenarios without changing the firmware logic.

During the experiments, the incoming UART logs were processed by a Python script on the PC and visualized in real time. An example of this live logging and plotting window is shown in Figure B.1. The Python script was used only for measurement support; the synchronization loop itself runs completely on the STM32 boards.

The PPS output provides an independent measurement path. The UART log reports the internal PTP offset computed from packet timestamps, whereas the

oscilloscope measurement compares the physical 1 Hz PPS output of the master and slave. During development, these measurements were used to verify that the sign convention and magnitude of the logged offset correspond to real clock alignment.

6.8 Summary

The final firmware was reached through several development stages. The first functional milestone was a slave implementation tested against a PC running PTP software. This verified Ethernet frame reception, PTP parsing, hardware RX timestamp extraction, TX timestamp requests, delay request generation, and basic offset calculation. However, the PC master used software timestamping, which introduced high jitter and made delay measurements unsuitable for the final accuracy target. This confirmed the practical importance of the hardware-assisted approach discussed in Section 2.3.2.

The next stage was the STM32-to-STM32 direct-link version. It confirmed that the hardware timestamping path worked on the selected Nucleo-F767ZI platform described in Chapter 4. During this stage, detailed UART logging and PPS-based oscilloscope verification were added.

The final baseline extends the implementation to one master and multiple slaves through an ordinary Ethernet switch. This required unique slave identities, unicast `Delay_Resp` messages, identity checks on the slave, a delay request queue on the master, and protection against mixing timestamp pairs on the slave. The implemented firmware therefore realizes a hardware-assisted Layer 2 PTP synchronization system on STM32F7: the master generates two-step synchronization messages and answers delay requests, while the slave obtains all four event timestamps from Ethernet hardware, computes end-to-end path delay and offset, filters invalid samples, and corrects its clock using bounded phase updates and PI addend control.

The main implementation decision was to combine HAL-based Ethernet operation with direct access to DMA descriptor timestamp fields and STM32 PTP registers. This made it possible to keep the project compatible with STM32CubeIDE generated initialization code while still obtaining timestamps at the Ethernet MAC level. Logging, counters, and PPS output are integrated into the firmware so that the behavior of the algorithm can be evaluated experimentally in the following chapter.

7 Experimental evaluation of the implemented algorithm

This chapter evaluates the synchronization accuracy and robustness of the implemented IEEE 1588 PTP system. The evaluation was performed on the STM32 Nucleo-F767ZI test stand described in Section 6.1. The implemented system uses PTP directly over Ethernet with EtherType `0x88F7`, hardware timestamps from the STM32 Ethernet MAC, and a two-stage slave servo consisting of bounded coarse phase correction and fine frequency correction through the PTP addend, as described in Chapter 6.

The purpose of the experiments was not only to show that the slave clock converges to the master clock, but also to verify how stable the synchronization remains in longer runs, in a multi-slave topology through a regular Ethernet switch, and under additional non-PTP network traffic. For this reason, the evaluation combines two measurement methods:

- log-based evaluation from the UART CSV output of the slave firmware, described in Section 6.7;
- external PPS evaluation using the physical 1 Hz PPS outputs of the boards, described in Section 6.2.

7.1 Measurement methodology

The log-based measurements provide a large number of samples and make it possible to inspect the internal state of the algorithm, including the estimated offset, mean path delay, PI controller output, reject counters, timeout counters, and resynchronization counters. The PPS measurements provide an independent check of the real physical timing at the output pins and were used to validate that the internal log values correspond to the observed timing behavior.

Table 7.1: Main components of the experimental setup.

Component	**Role in the experiments**
STM32 Nucleo-F767ZI master	Reference PTP clock and PPS source.
STM32 Nucleo-F767ZI slave boards	Synchronized clocks under evaluation.
Ethernet switch	Non-PTP-aware Layer 2 switch for multi-slave tests.
PC / laptop	UART logging, live plotting, and network traffic generation.
Oscilloscope	External PPS offset measurement.

7.1.1 Log-based measurement

Each slave periodically prints a CSV line after a processed `Delay_Resp` message. The most important logged quantities for the evaluation are:

- `offset_ns` – the estimated slave offset from the master, calculated from the four PTP timestamps;
- `offset_avg_ns` – the filtered offset used by the PI frequency controller;
- `mean_path_delay_ns` – the estimated end-to-end path delay;
- `rejected_sample_count` and related counters – the number of samples rejected before they were allowed to affect the servo;
- `resync_count` – the number of forced resynchronizations after a long sequence of rejected samples.

During the experiments, the incoming UART logs were processed by a Python script on the PC and visualized in real time. An example of this live logging and plotting window is provided in Figure B.1.

The logged offset is calculated from the standard end-to-end PTP Equations 2.1 and 2.2. All four timestamps are hardware timestamps. Therefore, the log-based offset does not include software interrupt latency in the timestamp itself. However, the value is still an internal estimate based on the PTP message exchange and on the assumption that the path delays are sufficiently stable and approximately symmetric.

For statistical evaluation, the initial part of each run was excluded. Unless stated otherwise, the statistics were computed from sample 7000 to the end of the selected log. This removes the startup transient, during which the slave may still be in coarse phase correction mode and the PI frequency controller has not yet reached a steady operating point.

The following statistics are used in the tables:

- mean offset;
- standard deviation of the offset;
- 95th and 99th percentile of the absolute offset;
- minimum and maximum offset;
- mean and standard deviation of the path delay.

The percentiles of the absolute offset are especially useful because the mean offset can be close to zero even if the signal contains occasional positive and negative deviations.

7.1.2 PPS measurement

For external verification, the PPS outputs of the STM32 Ethernet PTP unit were measured. The master and slave PPS signals were compared on an oscilloscope

and the time difference between their edges was recorded. Each PPS measurement window contains 30 measured offsets. The logic of connecting an oscilloscope is shown in Figures 7.1 and 7.4.

The PPS method has a smaller number of samples than the UART logs, but it has an important advantage: it observes the physical timing output of the board. Therefore, it verifies the final effect of the synchronization algorithm rather than only the internal variables computed by the firmware. In the presented results, the PPS values are generally consistent with the log-based measurements, which confirms that the log output is a suitable diagnostic tool for the rest of the experiments.

7.2 Direct master–slave connection

The first evaluation was performed with a direct Ethernet connection between one master board and one slave board. This configuration removes the switch from the path and therefore provides a baseline for the implemented algorithm. Figure 7.1 shows the measurement setup used for the direct master–slave experiment.

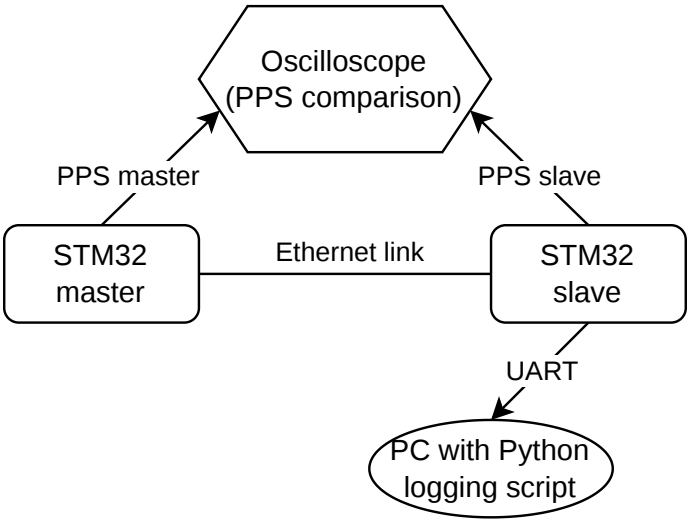


Fig. 7.1: Measurement setup used for the direct master–slave synchronization experiment.

Figure 7.3 shows a representative log-based result for the direct connection. The raw offset remains centred around zero and the filtered offset follows the long-term trend used by the PI controller. The resulting deviations are in the sub-microsecond range, with occasional short outliers.

The statistical summary of the direct-connection logs is given in Table B.2. The mean offset is small in all runs, from 5 ns to 22 ns. The standard deviation is between 173 ns and 243 ns. The 99th percentile of the absolute offset remains below 771 ns in all four tests. The measured mean path delay is approximately 1 μs, with a very small standard deviation of 8–9 ns. This indicates that the direct Ethernet link is highly stable and introduces almost no delay variation.

The external PPS measurements for the same direct topology are shown in Figure 7.2 and summarized in Table B.1. The PPS results confirm the log-based behavior. Most measured windows have standard deviation below 253 ns, and the 99th percentile of the absolute PPS offset is below 605 ns. The only wider window is the first selected interval of Unicast test 4, where the standard deviation reaches 335 ns. Even in that case the maximum measured PPS offset remains 624 ns.

The direct-connection measurements demonstrate that the core algorithm works correctly when the path delay is stable. The residual error is dominated by small measurement noise, oscillator drift between updates, and the dynamics of the PI addend controller.

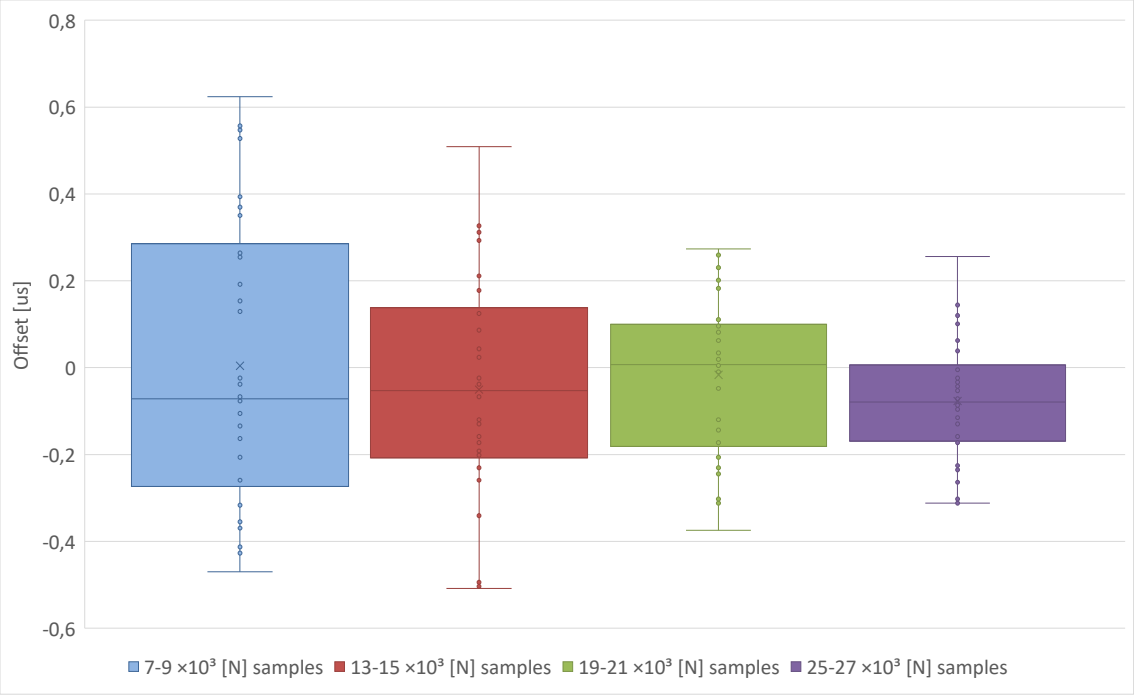


Fig. 7.2: PPS offset distribution for the direct master–slave Ethernet connection. Each window of samples contains 30 oscilloscope measurements.

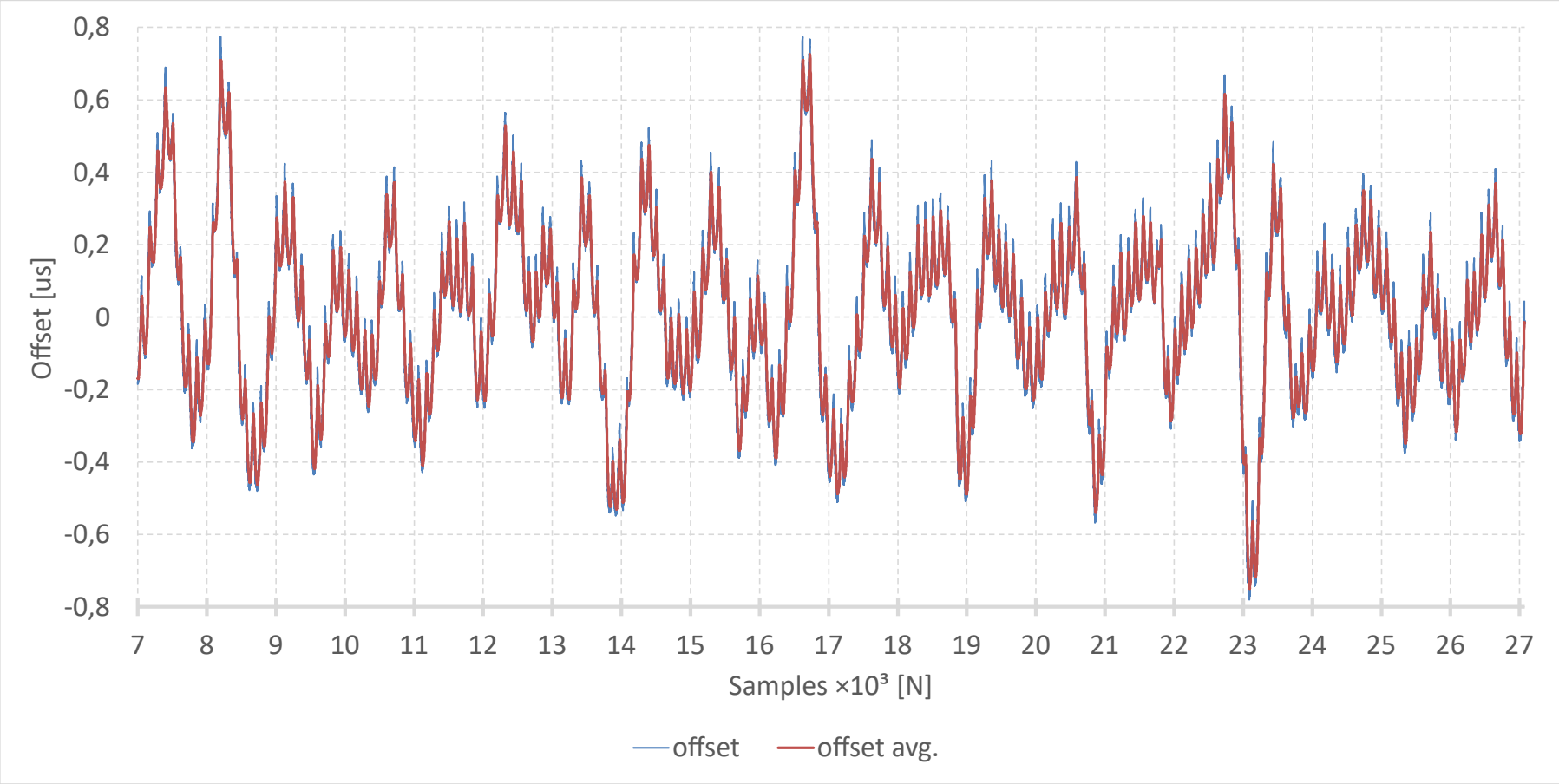


Fig. 7.3: Logged offset and filtered offset in the direct master–slave Ethernet connection after the initial transient.

7.3 Multi-slave Synchronization through an Ordinary Ethernet Switch

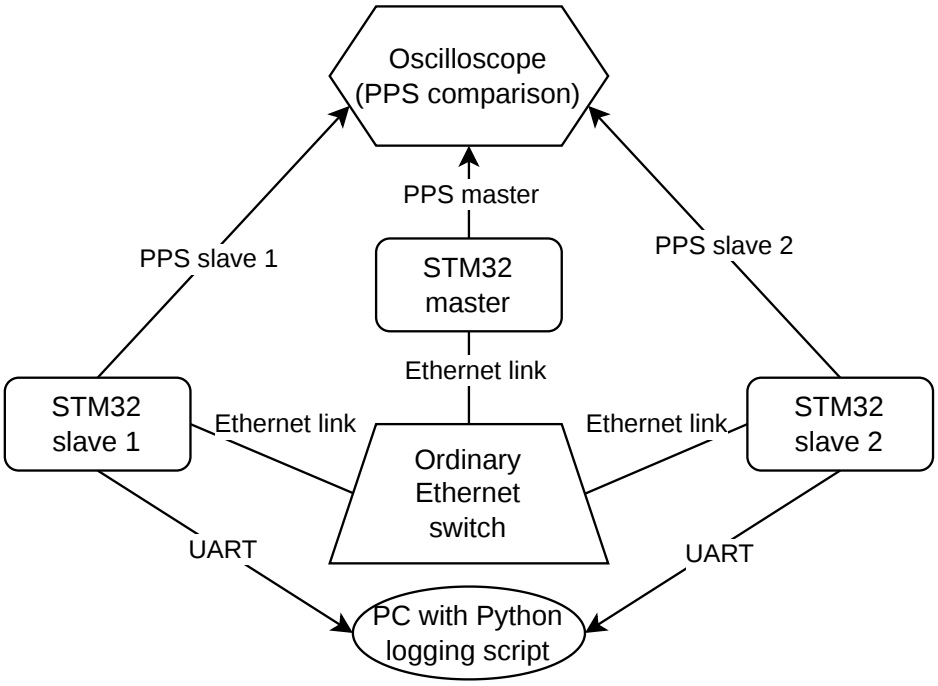


Fig. 7.4: Measurement setup used for the multi-slave synchronization experiment through an ordinary Ethernet switch.

The second evaluation used one master and two slaves connected through an ordinary Ethernet switch. The switch was not configured as a PTP transparent clock or boundary clock. It only forwarded Ethernet frames. This configuration is more demanding than the direct connection because the switch adds forwarding delay and can introduce additional delay variation. Figure 7.4 shows the measurement setup used for the multi-slave experiment.



Fig. 7.5: Logged offsets of two slaves synchronized through an Ethernet switch in a representative multi-slave run.

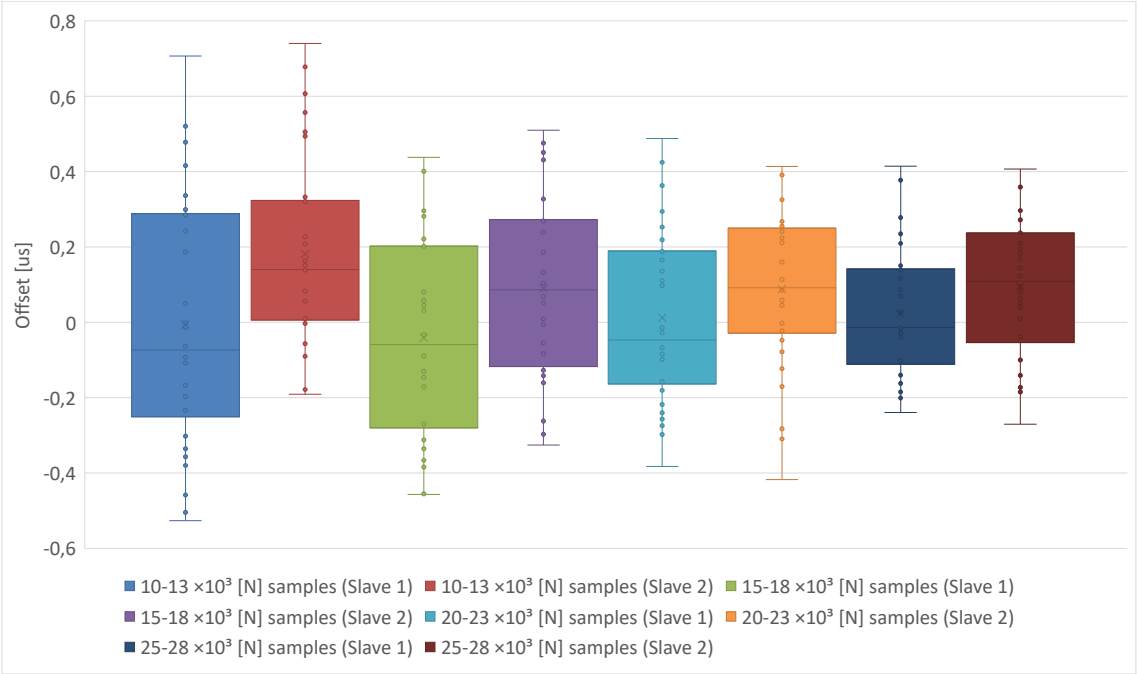


Fig. 7.6: PPS offset distributions for two slaves synchronized through an Ethernet switch. Each window contains 30 oscilloscope measurements.

Figure 7.5 shows that both slaves remain synchronized around zero offset. Compared with the direct connection, the signal contains more spikes. This is expected because the switch introduces additional forwarding delay and occasional queuing. Nevertheless, the normal operating region remains within the sub-microsecond range for the majority of samples.

Table B.3 summarizes three multi-slave runs and one longer run. The mean offset remains close to zero for both slaves. In the long test, slave 1 has a mean offset of 3 ns and slave 2 has a mean offset of 11 ns. The 99th percentile of the absolute offset is 704 ns for slave 1 and 586 ns for slave 2. The mean path delay increases to approximately 8.1–8.4 μs because of the Ethernet switch, but its standard deviation remains around 100 ns in normal operation. Therefore, the switch adds a significant constant delay, but the ordinary unloaded switch does not destroy synchronization stability.

The PPS verification for the switch topology is shown in Figure 7.6. The corresponding values are listed in Tables B.4 and B.5. The PPS measurements again confirm the log-based results. The majority of windows remain within several hundred nanoseconds. The largest PPS deviations occur in individual windows, for example slave 2 in Multicast test 2, but even there the 99th percentile remains below 675 ns.

The long run in Figure 7.7 confirms that the switch-based multi-slave system does not slowly drift away from the master. Both slaves remain close to zero over

more than 43000 evaluated samples after the initial transient. Occasional spikes are visible, but they do not accumulate into a long-term error.

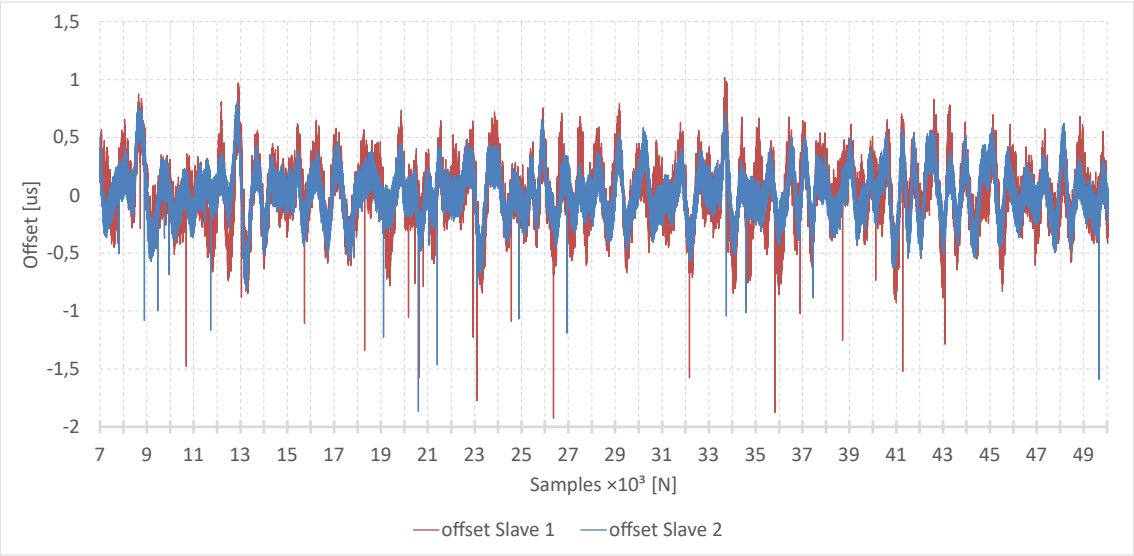


Fig. 7.7: Long-term log-based offset of two slaves synchronized through an Ethernet switch without additional network load.

7.4 Behavior of outliers and rejected samples

The switch-based graphs contain short offset spikes even without an artificial stress test. Such spikes are expected in a non-PTP-aware switched Ethernet network. They can be caused by temporary switch buffering, arbitration between frames, delayed `Delay_Resp` frames, or short bursts of unrelated traffic. A PTP end-to-end measurement assumes that the two one-way delays are sufficiently stable during the cycle. If one direction is delayed more than the other, this asymmetry appears in the offset equation as a false clock error.

For this reason, the slave does not accept every computed sample. The path-delay filter is especially important. In the final switch baseline, samples are accepted only if the estimated mean path delay remains within the configured interval. For the switch topology this interval was set to 6000–10000 ns. A sample with a much higher mean path delay is not a reliable indication that the slave clock has drifted. It more likely indicates that one of the packets was delayed in the switch or in the Ethernet path. If such a sample were used by the PI controller, the regulator would correct the clock in response to network queuing rather than real oscillator drift Section 6.5.5.

Therefore, an outlier may appear in the log as a computed `raw_offset_ns` or as a diagnostic value, but it is not necessarily passed into the servo. When a sample

is rejected, the reject counters are incremented and the addend controller is not updated from that measurement. This behavior explains why visible individual disturbances do not normally destabilize the system. The rejection logic prevents sporadic high-delay samples from corrupting the frequency controller.

7.5 Network stress test

After confirming the normal switch-based operation, the system was tested under additional non-PTP Ethernet traffic. The stress traffic was generated by a Linux laptop connected to the same switch. The generated frames used a non-PTP EtherType, so they were not interpreted as PTP messages by the firmware. Their purpose was to load the Ethernet link and the switch, creating queueing conditions that could disturb PTP message timing.

One test run was performed with several load levels in sequence. After an initial unloaded interval, the generated network load was increased to 60 Mbit/s, then to 70 Mbit/s, and finally to 75 Mbit/s. After the system lost stable synchronization at 75 Mbit/s, the load was removed and the recovery was observed.

Figures 7.8–7.13 show the selected windows of this test. Table B.6 summarizes the same run numerically. The effective cycle value in the table was estimated from PC timestamps in the CSV log. It is therefore not a precise hardware measurement of the master period, but it is useful for comparing windows from the same experiment.

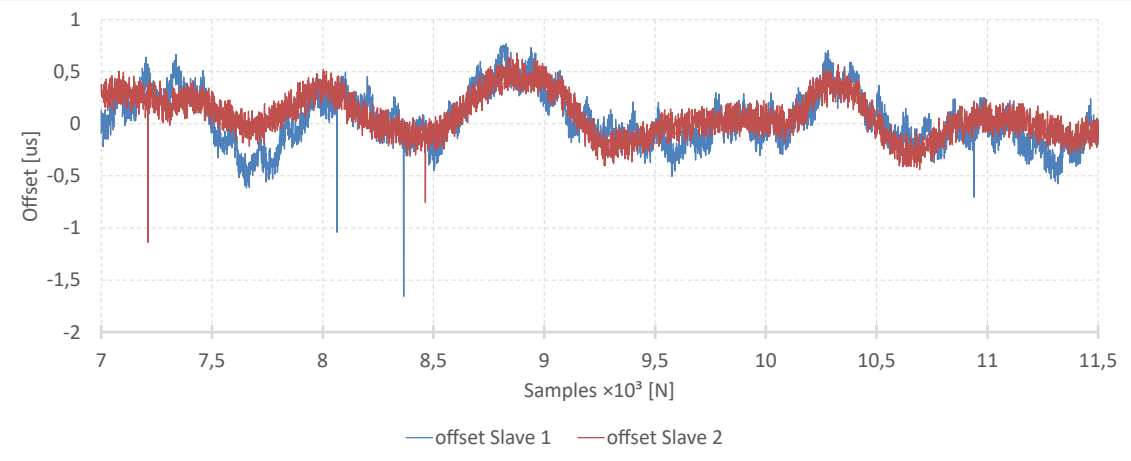


Fig. 7.8: Network stress test: baseline switch-based multi-slave operation before additional traffic.



Fig. 7.9: Network stress test: switch-based multi-slave operation under 60 Mbit/s additional Ethernet traffic.

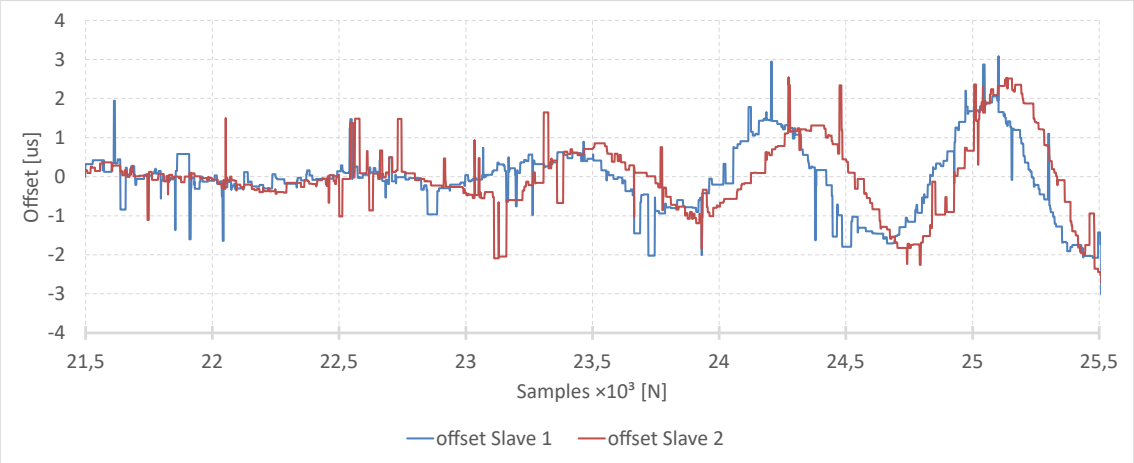


Fig. 7.10: Network stress test: switch-based multi-slave operation under 70 Mbit/s additional Ethernet traffic.

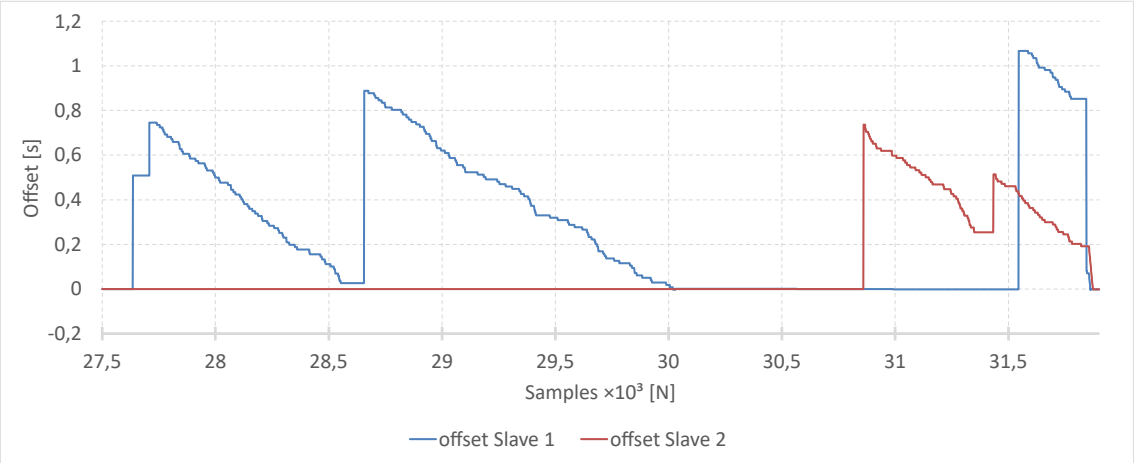


Fig. 7.11: Network stress test: loss of stable synchronization under 75 Mbit/s additional Ethernet traffic.

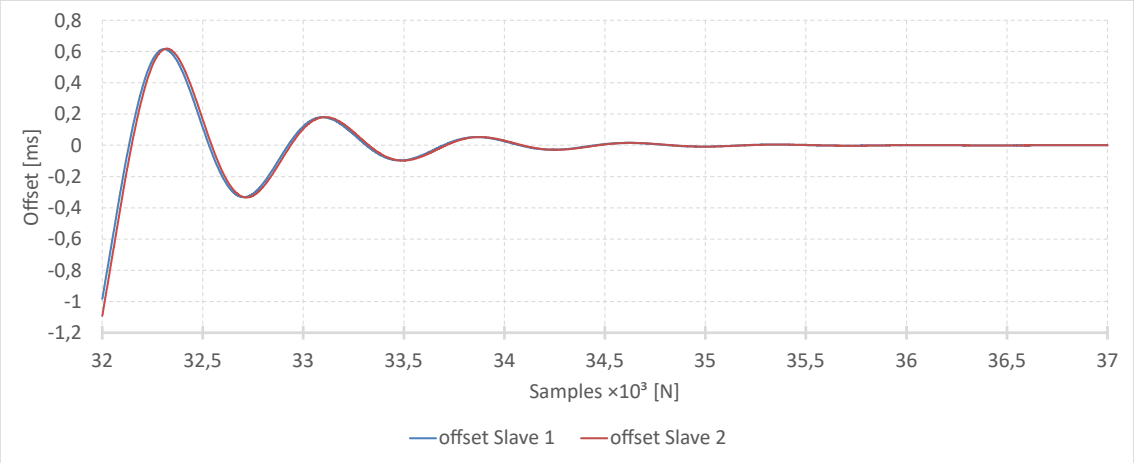


Fig. 7.12: Network stress test: initial recovery after the 75 Mbit/s load was removed.

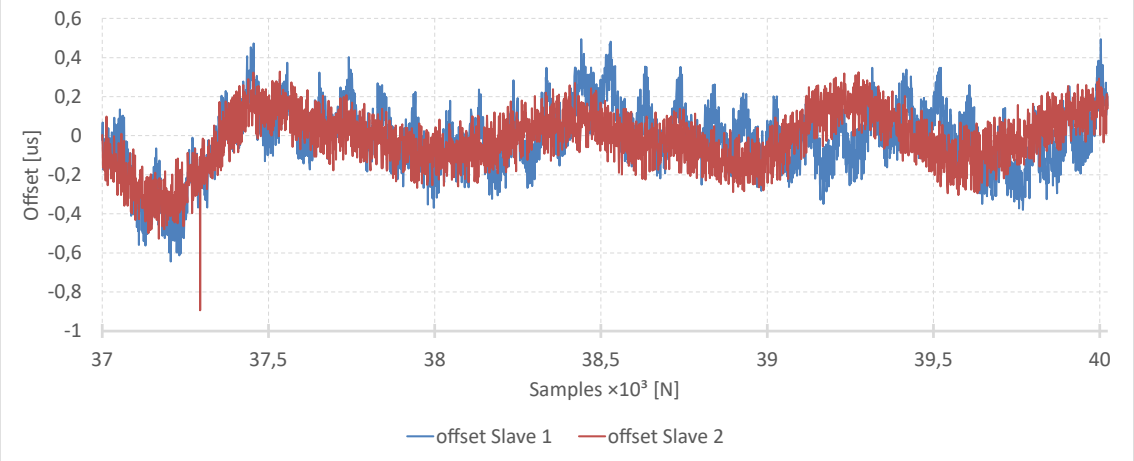


Fig. 7.13: Network stress test: stable operation after recovery from the overloaded interval.

The unloaded part of the stress run behaves similarly to the normal long switch test. Both slaves remain within several hundred nanoseconds for most samples. At 60 Mbit/s the offset becomes more disturbed, but the system still remains synchronized. The important point is that more than 82 % of samples in this window were rejected by the path-delay filter. This means that most PTP cycles were affected by excessive switching delay and were therefore not suitable for servo correction. However, enough valid samples still remained for the frequency controller to keep the slave clocks close to the master.

At 70 Mbit/s, the same behavior becomes stronger. Almost 89 % of samples are rejected, and the 95th percentile of the absolute offset increases to approximately 1.8 μs for slave 1 and 2.3 μs for slave 2. Nevertheless, no forced resynchronization occurs in this interval. This shows that the reject filter is effective: it prevents the controller from reacting to most high-delay samples while still allowing the system to use the remaining valid measurements.

At 75 Mbit/s the system crosses its practical stability limit for this non-PTP-aware switch setup. Both slaves still attempt to recover, but the valid sample rate is no longer sufficient. Slave 1 shows several recovery attempts and performs four forced resynchronizations in the selected window. During one of these attempts it temporarily approaches a controlled state, but then loses it again as the high-delay samples continue. Slave 2 remains regulated for part of the interval, but eventually also loses stable synchronization and performs two forced resynchronizations. The offset reaches hundreds of milliseconds and, for slave 1, exceeds one second. This is not a normal servo error but a loss of useful PTP measurement conditions under excessive network load.

The effective logged cycle interval also increases during the load sequence: from approximately 131 ms without load to 134.6 ms at 60 Mbit/s, 137.8 ms at 70 Mbit/s, and 140.2 ms at 75 Mbit/s. This supports the conclusion that the load changes not only the path-delay values, but also the effective timing of the PTP exchange observed by the logger. Since the values are derived from PC timestamps, they should be interpreted as an experimental indicator rather than as the exact internal master period.

After the 75 Mbit/s traffic is removed, the system recovers automatically. Figure 7.12 shows the initial recovery transient. The offset first has a large low-frequency excursion while the slaves return from the disturbed state. Figure 7.13 shows the later stable part of the recovery. In that final window, both slaves again operate in the same range as the normal unloaded tests, with 95th percentile absolute offset below 350 ns for both slaves.

7.6 Summary of obtained results

The experimental evaluation confirms that the implemented STM32-based Layer 2 PTP system is able to achieve stable sub-microsecond synchronization in a controlled local Ethernet network. In the direct master–slave topology, the mean path delay remained close to 1 μs with only about 8–9 ns standard deviation. The PI addend controller kept the slave clock close to the master, and the external PPS measurement confirmed that the logged offsets corresponded to real physical timing differences.

The system also operated correctly in the one-master, two-slave topology through an ordinary Ethernet switch. Although the switch increased the mean path delay to approximately 8.1–8.4 μs and introduced occasional outliers, the unloaded switch topology still provided stable sub-microsecond synchronization for the majority of samples. In the long multi-slave test, the 99th percentile of the absolute offset was 704 ns for slave 1 and 586 ns for slave 2, which confirms that no long-term drift accumulated.

The main limitation of the tested setup is operation through a non-PTP-aware switch under high background traffic. At 60 and 70 Mbit/s, the system remained synchronized only because the firmware rejected most high-delay samples and used the remaining valid measurements. At 75 Mbit/s, the number and duration of invalid samples became too large, causing repeated resynchronizations and loss of stable steady-state synchronization. After the traffic was removed, the forced-resynchronization logic allowed the slaves to return to stable operation without manual reset.

Additional firmware tests with synthetic CPU load showed that the dominant practical limitation is not the computational cost of the PI servo or timestamp processing, but the availability of clean and sufficiently symmetric PTP message timing in the Ethernet network. Overall, the results show that the implemented algorithm is suitable for synchronizing several STM32 boards in a controlled local network. For more reliable operation under heavy traffic, a PTP-aware switch, traffic prioritization, or separation of PTP traffic from high-bandwidth background traffic would be required.

8 Single Pair Ethernet as a Physical Layer for IEEE 1588 Time Synchronization

The implemented synchronization algorithm was experimentally tested using standard 100BASE-TX Ethernet on STM32 Nucleo-F767ZI boards. This physical layer was selected for the practical part of the work because it is directly supported by the available development boards and provides convenient access to the STM32 Ethernet MAC hardware timestamping unit. However, in modern automotive and industrial automation systems, Single Pair Ethernet (SPE) is becoming increasingly important as an alternative Ethernet physical layer. Its main motivation is the ability to preserve the Ethernet communication model while reducing cabling complexity and enabling compact, cost-effective connections to sensors, actuators, and distributed embedded devices.

Since precise time synchronization is an essential requirement in many distributed measurement, control, and automation systems, it is reasonable to evaluate whether the PTP approach implemented in this thesis could also be used over SPE links. This chapter therefore analyzes SPE not as a second implementation platform, but as a possible physical layer for transferring the same IEEE 1588-based synchronization principle to another Ethernet medium. The focus is placed on which SPE variants are relevant for time synchronization, how PTP messages can be transported over SPE, and what timing limitations can be expected from the physical layer.

SPE itself does not define a new time synchronization protocol. It defines Ethernet physical layers that operate over one balanced pair of conductors. Therefore, as long as the MAC layer and the selected PTP transport mapping remain Ethernet-compatible, PTP messages can be transmitted over SPE in the same logical way as over conventional Ethernet: either directly as Layer 2 Ethernet frames with EtherType `0x88F7`, or encapsulated in UDP/IP. What changes is not the basic PTP message exchange, but the timing behavior of the physical link. For time synchronization, the important factors are PHY latency, timestamp reference plane, link asymmetry, the MAC–PHY interface, serialization time, and, in the case of multidrop 10BASE-T1S, medium access behavior.

8.1 Single Pair Ethernet Overview

Single Pair Ethernet is a family of IEEE 802.3 physical-layer standards that transmit Ethernet frames over one twisted pair instead of the two pairs used by 100BASE-TX or the four pairs used by 1000BASE-T. The motivation is to reduce wiring

mass, connector size, and installation cost while keeping the Ethernet frame format and the possibility of using existing Ethernet-based software and network tools. This is especially attractive in automotive systems, building automation, process automation, and sensor or actuator networks.

The SPE family is not a single interface. It contains several physical layers optimized for different distances, data rates, and topologies. The most relevant variants for this thesis are summarized in Table 8.1. The 10 Mbit/s variants 10BASE-T1L and 10BASE-T1S were introduced by IEEE 802.3cg. 10BASE-T1L targets long point-to-point industrial links, while 10BASE-T1S targets short point-to-point and multidrop links. 100BASE-T1 is a higher-speed automotive-oriented SPE variant and is important because its data rate is close to the 100BASE-TX links used in the practical experiments of this thesis.

Table 8.1: Main SPE variants relevant to IEEE 1588 time synchronization.

Variant	**Nominal rate**	**Typical reach**	**Topology**	**Relevance for PTP**
10BASE-T1L	10 Mbit/s	up to about 1000 m	full-duplex point-to-point	Long industrial links; PHY latency and MAC–PHY timestamping accuracy are critical.
10BASE-T1S	10 Mbit/s	15 m point-to-point, up to 25 m multidrop	half-duplex or full-duplex point-to-point; half-duplex multidrop	Low-cost sensor networks; multidrop and PLCA require special timestamping and protocol handling.
100BASE-T1	100 Mbit/s	automotive short reach	full-duplex point-to-point	Closest SPE alternative to 100BASE-TX in this thesis; good candidate for sub-microsecond synchronization.
1000BASE-T1 and faster T1 variants	1 Gbit/s and above	automotive and industrial short reach	full-duplex point-to-point	Relevant for high-bandwidth systems, but outside the experimental scope considered here.

The physical-layer encoding is one of the reasons why the 10 Mbit/s SPE variants cannot be treated as a perfectly transparent replacement for the 100BASE-TX link used in the practical part of the thesis. Figure 8.1 shows the simplified idea of 4B3T coding and PAM-3 modulation used for 10BASE-T1L. The purpose of this figure is not to describe the full PHY specification, but to highlight that the SPE PHY introduces its own coding and symbol timing below the Ethernet MAC. These PHY-level operations contribute to the difference between the MAC timestamp point and the actual line-side timing.

A second simplified coding example is shown in Figure 8.2. It illustrates the 10BASE-T1S case, where 4B/5B coding and DME encoding are used before trans-

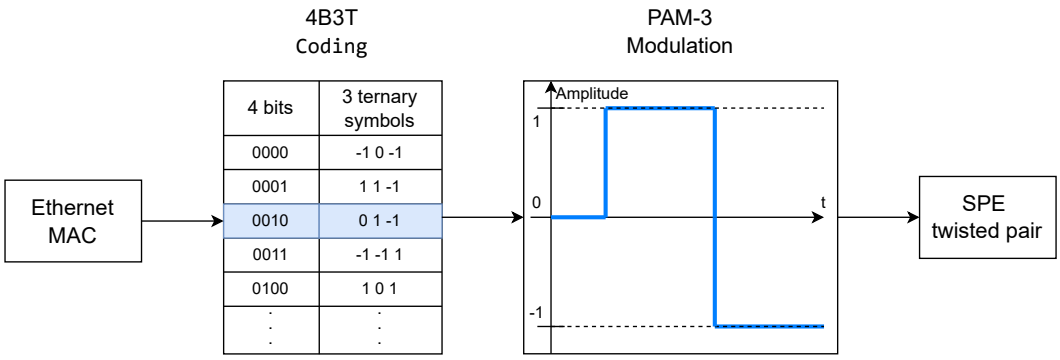


Fig. 8.1: Simplified view of 4B3T coding and PAM-3 modulation in a 10BASE-T1L physical layer.

mission on the single-pair medium. This again supports the main engineering point of this chapter: the Ethernet frame service above the PHY remains compatible with PTP, but the achievable timing accuracy depends on how the selected PHY transforms and schedules the physical transmission.

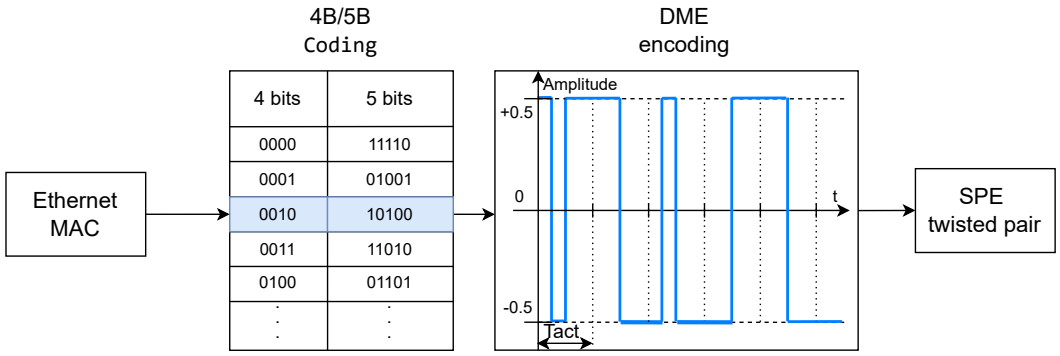


Fig. 8.2: Simplified view of 4B/5B coding and DME encoding in a 10BASE-T1S physical layer.

The industrial motivation for 10BASE-T1L is the replacement of long two-wire fieldbus links with Ethernet. The IEEE 802.3cg tutorial describes 10BASE-T1L as a 10 Mbit/s long-reach PHY and 10BASE-T1S as a short-reach PHY with optional multidrop operation and PLCA support [5]. Texas Instruments also emphasizes the practical industrial advantages of 10BASE-T1L: reuse of existing two-wire cabling, lower power operation, reduced gateways, and operation over long cable distances [6]. In the automotive domain, the introduction of one-pair Ethernet was driven by the need to reduce cable cost and mass while keeping Ethernet communication. Matheus describes BroadR-Reach, later standardized as 100BASE-T1, as the breakthrough that made Ethernet practical for automotive runtime applications over unshielded

single-pair cabling [7].

For this thesis, SPE is important mainly as a possible replacement of the physical layer below the Ethernet MAC. The implemented firmware already uses raw Ethernet frames and hardware timestamping in the MAC. Therefore, the logical PTP packet format would not have to change when moving from 100BASE-TX to an SPE PHY. The necessary hardware change would be replacing the conventional twisted-pair PHY and RJ45 interface by an SPE PHY and the corresponding single-pair connector or evaluation board.

8.2 PTP Transport over SPE

As concluded in Section 3.2.5, the developed system implements PTP as a Layer 2 Ethernet-based protocol using raw Ethernet frames with EtherType `0x88F7` and hardware-assisted timestamping in the STM32 Ethernet MAC. This means that the implemented PTP logic is not directly bound to a particular copper physical layer. The physical medium used in the practical part of this thesis is 100BASE-TX, but the PTP message format and the Layer 2 transport principle remain Ethernet-based.

This relation is summarized in Figure 8.3. In the current implementation, the PTP firmware communicates through the STM32 Ethernet MAC timestamping unit, the 100BASE-TX PHY, and the RJ45 cable. In a possible SPE implementation, the PTP firmware and the logical Ethernet frame handling could remain very similar, while the conventional PHY and cabling would be replaced by an SPE PHY and a single twisted pair. The figure is intentionally simple because the purpose is to identify the boundary where the practical changes would occur: below the MAC timestamping unit, not in the PTP message exchange itself.

From this point of view, replacing conventional 100BASE-TX with SPE would primarily mean replacing the physical layer and the external PHY, while keeping the same logical Ethernet frame format. IEEE 1588 PTP can still be transported over Ethernet in the two usual ways: directly as Layer 2 Ethernet frames with EtherType `0x88F7`, or encapsulated in UDP/IP. Since SPE transports standard Ethernet frames, it does not prevent either transport method. The same `Sync`, `Follow_Up`, `Delay_Req`, and `Delay_Resp` messages could therefore be used over an SPE link.

However, such a transfer would not be completely automatic. The main condition is that the MAC, driver, and PHY path must provide a reliable way to associate each PTP event message with the correct transmit or receive timestamp. In the current implementation, timestamps are taken by the STM32 Ethernet MAC. If an external SPE PHY is added between the MAC and the physical medium, the timestamp reference point no longer coincides exactly with the point where the frame enters

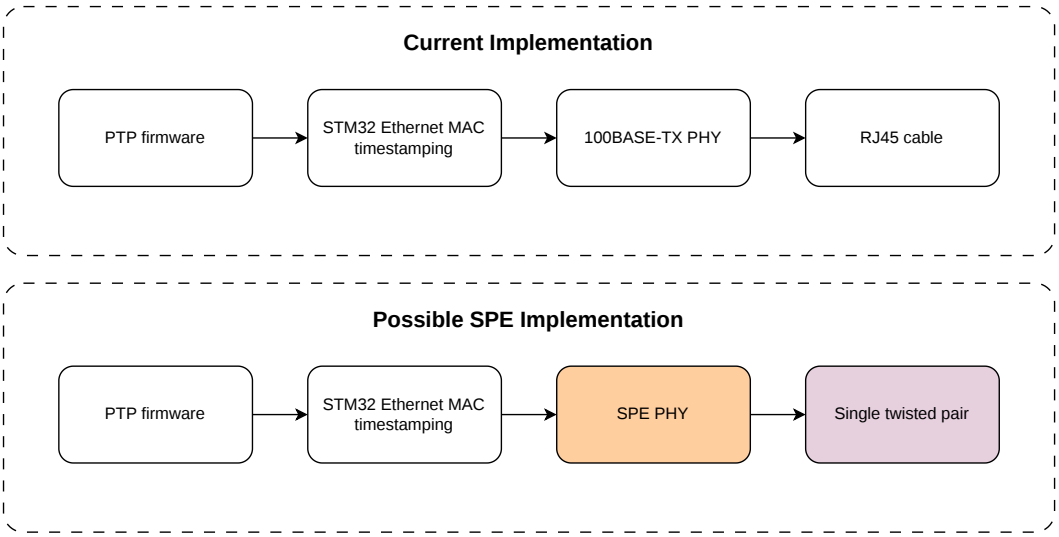


Fig. 8.3: Conceptual transfer of the implemented Layer 2 PTP approach from the current 100BASE-TX setup to a possible SPE setup.

or leaves the cable. The difference is caused by PHY-related transmit and receive latency, and it becomes an important source of constant offset, asymmetry, and jitter.

For a point-to-point SPE link, such as 100BASE-T1 or 10BASE-T1L, the transfer of the implemented Layer 2 approach would be conceptually straightforward. The firmware would still build and parse PTP Ethernet frames, request transmit timestamps, read receive timestamps, and correct the local PTP clock. The main required changes would be related to PHY configuration, link setup, possible MAC–PHY interface differences, and compensation of PHY latency and link asymmetry. Therefore, point-to-point SPE can be considered the closest SPE equivalent to the conventional Ethernet topology used in the practical part of this thesis.

For multidrop 10BASE-T1S, the situation is more complex. Multiple devices share one physical segment, and half-duplex operation can introduce medium access delay. The PLCA Reconciliation Sublayer improves utilization and bounds access latency by assigning transmit opportunities after a beacon, but it still changes the timing path compared with a simple full-duplex point-to-point link [5]. Microchip also notes that standard Pdelay messages need software customization on multidrop segments, because multiple devices may generate delay requests and the clock source must return the correct timestamp to the correct requesting device [8]. Therefore, 10BASE-T1S is usable for time-aware systems, but it should not be treated as a direct drop-in replacement for a switched full-duplex Ethernet topology.

8.3 Timing Mechanisms Affected by the SPE Physical Layer

The PTP equations used in this thesis, Equations 2.1 and 2.2, assume that the measured path delay is sufficiently stable during one synchronization cycle and that the forward and reverse delays are approximately symmetric. When this assumption is violated, a delay variation in one direction is interpreted as a false clock offset. This is true for conventional Ethernet and for SPE, but SPE introduces several additional physical-layer effects that must be considered.

8.3.1 Timestamp Reference Plane

A timestamp can be taken at different points in the communication path. In the STM32 implementation of this thesis, the timestamp is captured by the Ethernet MAC hardware, as described in Section 6.2. This removes software interrupt and driver latency from the timestamp itself, but it still leaves the latency between the MAC timestamp point and the actual physical medium. In a conventional Nucleo-F767ZI setup this includes the MAC, external PHY, magnetics, and cable. In an SPE setup it includes the MAC–PHY interface, the SPE PHY internal transmit and receive paths, line coding, echo cancellation or medium access functions, and the single-pair cable.

Figure 8.4 shows this issue in a simplified form. The timestamp taken by the MAC is not necessarily the same as an ideal line-side timestamp at the MDI or cable reference plane. If the difference between these two points is constant and symmetric, it can be compensated. If it depends on PHY role, link establishment, interface alignment, or traffic conditions, it becomes a source of residual synchronization error.

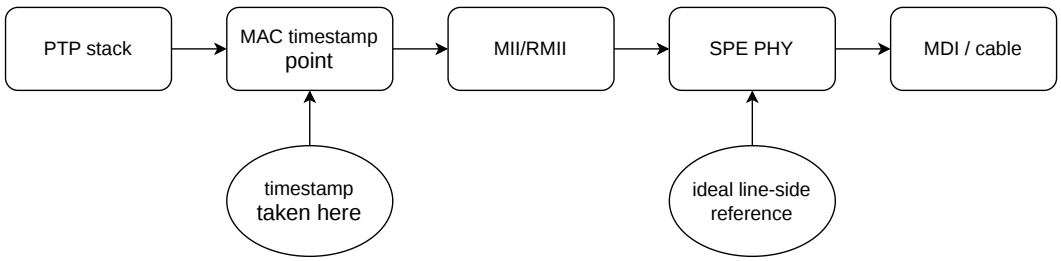


Fig. 8.4: Simplified timestamp reference plane in an SPE-based PTP system.

For accurate PTP or gPTP, the ideal timestamp reference plane is close to the medium-dependent interface. If timestamps are captured at the MAC–PHY interface, constant egress and ingress latencies can be compensated by fixed correction

values. Dynamic or link-dependent latency, however, becomes much harder to remove. This is one of the main conclusions of the 10BASE-T1L timestamping study by Brunner, Da Silva Miranda, and Ostertag, which shows that MAC–PHY interface timestamping on current 10BASE-T1L components may introduce several hundred nanoseconds of error depending on PHY type, interface, PHY role, and link establishment state [9].

8.3.2 MII, RMII, and PHY Internal Delay

The MAC–PHY interface matters because it determines where the MAC can observe a frame. Brunner et al. show that 10BASE-T1L uses 4B3T coding with a symbol duration of 133.33 ns and a 400 ns time for one 4-bit PCS code group. With MII, alignment between PCS code groups and symbols can create discrete latency positions. With RMII, additional effects appear because the MAC and PHY use a common reference clock and the PHY must perform clock-domain crossing through an elasticity buffer. The result is that the phase relation between MAC and PHY may add timestamp uncertainty up to hundreds of nanoseconds in some configurations [9].

This is directly relevant for an STM32-based system. The Nucleo-F767ZI Ethernet MAC normally uses an external PHY through an RMII-like interface in the standard board configuration. If an SPE PHY is connected through RMII, the implementation must not assume that the hardware timestamp taken at the STM32 MAC is automatically equivalent to a PHY-level timestamp. For moderate submicrosecond synchronization this may still be acceptable. For strict gPTP or TSN requirements, a PHY or MAC-PHY device with packet timestamping closer to the line side is preferable.

8.3.3 Multidrop 10BASE-T1S and PLCA

10BASE-T1S is attractive because it can reduce the number of switches by allowing several low-cost nodes to share one short single-pair segment. For PTP this is also the main complication. In a multidrop segment, transmission may be delayed by the medium access mechanism. PLCA bounds this delay, but the delay is still part of the timing path. If the transmit timestamp is taken when the frame leaves the MAC, but the PHY or PLCA layer delays the actual transmission, the timestamp does not represent the real line event.

Figure 8.5 shows the topology idea in a simplified way. A clock source or PLCA coordinator and several follower nodes share one single-pair bus. This differs from the ordinary switch-based topology tested in Section 7.3: in a switch topology each

node has its own point-to-point link to the switch, while in multidrop 10BASE-T1S several nodes use the same physical medium.

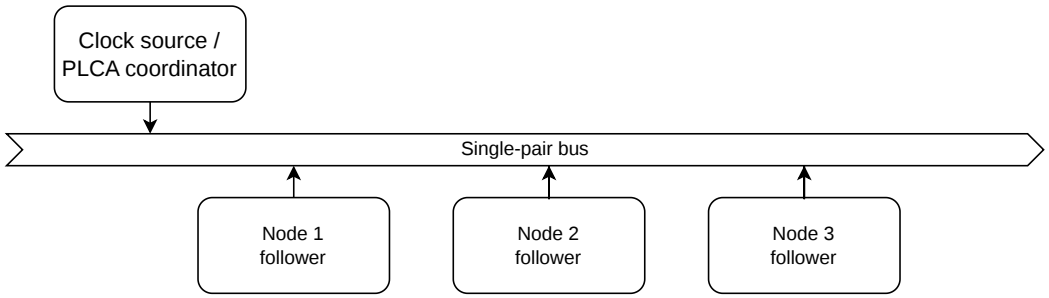


Fig. 8.5: Simplified 10BASE-T1S multidrop segment.

The LAN8650/1 application note addresses exactly this issue. It explains that the PLCA elastic buffer can add a bounded but variable delay, and therefore accurate timestamps should be taken in the PHY when the packet is actually transmitted and received. The LAN8650/1 includes packet pattern matchers and timestamp units to capture timestamps at the end of the SFD for transmit and receive packets [8]. This makes 10BASE-T1S synchronization feasible, but it means that an MCU implementation should use the timestamping features of the MAC-PHY device rather than relying only on MCU-side MAC timestamps.

8.3.4 Bandwidth and Network Load

The data rate also affects the robustness of synchronization. At 10 Mbit/s, a frame occupies the medium ten times longer than at 100 Mbit/s. PTP event messages are small, but background traffic and medium access delays become more significant. This is consistent with the results of Wiesner and Kovácsházy: in their SPE evaluation platform, 100BASE-T1 showed the best timing performance, while the 10 Mbit/s SPE variants showed larger offset ranges [10]. It is also consistent with the stress tests in this thesis in Section 7.5, where a non-PTP-aware switch under high traffic load produced many rejected samples and eventually loss of stable synchronization at the highest tested load level.

8.4 Experimental Results Reported in Related Work

Several related works provide useful reference points for evaluating SPE as a PTP physical layer. They do not all measure exactly the same quantity, so they must be compared carefully. Some works report complete synchronization offset measured by PPS comparison, while others investigate only PHY timestamping accuracy or one-way latency effects.

8.4.1 100BASE-T1 and gPTP in Automotive Ethernet

Delbosco evaluates gPTP synchronization over 100BASE-T1 in an automotive-oriented setup. The baseline offset is reported around a mean value of 34.40 ns with a standard deviation of 27.48 ns, and the observed offset distribution is concentrated mostly between approximately 0 ns and 50 ns. The same work reports a mean peer delay around 15.5 μs, dominated not by the physical two-meter 100BASE-T1 cable but by switch residence time, PHY latency, and the chosen timestamp reference planes [11].

This result shows two important points. First, 100BASE-T1 can support high-quality synchronization when the hardware and gPTP path are designed for it. Second, the measured path delay in a real embedded system is often dominated by internal delays and switch architecture rather than by cable propagation. This is important when comparing to the results of this thesis, where an ordinary non-PTP-aware switch also increases the measured path delay from about 1 μs in the direct link to about 8 μs in the switch topology, as discussed in Section 7.3.

8.4.2 SPE PHY Comparison with flexPTP

Wiesner and Kovácsházy compare 100BASE-TX, 100BASE-T1, 10BASE-T1L, and 10BASE-T1S using a modular MCU evaluation platform and the flexPTP implementation. Their measurements use PPS comparison between a timeTransmitter and a timeReceiver. With a PI servo, they report approximately $\pm$200 ns worst-case offset for 100BASE-TX, approximately $\pm$50 ns for 100BASE-T1, and offsets mostly below $\pm$500 ns for the 10 Mbit/s SPE variants. Their statistical table gives PI/PID min/max values of -193.94 ns / +214.06 ns for 100BASE-TX, -56.08 ns / +75.92 ns for 100BASE-T1, -498.02 ns / +497.98 ns for 10BASE-T1L, and -270.68 ns / +349.32 ns for 10BASE-T1S [10].

The comparison is highly relevant because it uses microcontroller-based hardware and evaluates physical PPS outputs. However, it is not a direct comparison with this thesis. The hardware platform, PHYs, servo implementation, clock configuration, and topology are different. Their results are mainly point-to-point PHY comparisons, while this thesis also evaluates a multi-slave topology through an ordinary switch and a high-load switch stress test. Therefore, the numbers can be used to estimate the order of magnitude expected from different physical layers, but not to predict exact results for the STM32 Nucleo-F767ZI implementation without additional measurements.

8.4.3 10BASE-T1L Timestamping Accuracy

Brunner et al. do not evaluate a complete PTP servo in the same way as the previous work. Instead, they investigate timestamping accuracy and one-way latency effects of 10BASE-T1L PHYs with MII and RMII interfaces. Their main conclusion is that MAC–PHY interface timestamping on available 10BASE-T1L components is affected by multiple static, pseudo-static, and dynamic errors. These errors can reach several hundred nanoseconds and may exceed strict IEEE 802.1AS-2020 or TSN profile requirements if they accumulate over multiple links [9].

This work is important for system design because it explains why good internal PTP logic is not sufficient by itself. Even if the PTP implementation and servo are correct, the timestamp may still correspond to the wrong reference point. For a future STM32-to-SPE implementation, this means that the selected PHY and MAC–PHY interface must be evaluated experimentally, especially if the target is not only sub-microsecond synchronization but strict TSN-grade time alignment.

8.4.4 10BASE-T1S Hardware Timestamping Example

The Microchip LAN8650/1 application note demonstrates that 10BASE-T1S time-aware systems are possible when the MAC-PHY device provides the required timestamping features. The document describes hardware packet timestamping, event capture, event generation, and 1PPS output. In the simple follower example, the reported 1PPS comparison shows a maximum time difference of 100 ns, a mean of 8 ns, and a standard deviation of 25 ns [8].

This result is promising, but it should be interpreted as an implementation-specific demonstration rather than a general result for all 10BASE-T1S systems. It uses dedicated LAN8650/1 hardware features and a simplified synchronization approach. It does not directly represent a generic STM32 MAC timestamping implementation connected to an external PHY, and it does not remove the need to handle multidrop addressing and delay-response association correctly in a real multi-node PTP profile.

8.5 Comparison with the Implemented Firmware

The experimental results of the implemented STM32-based PTP system over conventional 100BASE-TX Ethernet were analyzed in Chapter 7. For the comparison in this section, these results are used only as a reference baseline. The most relevant cases are the direct master–slave connection, the switch-based two-slave topology, and the network stress test, described respectively in Sections 7.2, 7.3, and 7.5.

In the direct and unloaded switch-based cases, the implemented system demonstrated sub-microsecond synchronization accuracy according to both log-based and PPS measurements. Under high non-PTP network load, the results showed that the main limitation of the developed architecture is not the PTP message format itself, but delay variation introduced by non-PTP-aware network elements and traffic-induced queuing. These observations are important for comparison with SPE studies, because SPE changes the physical layer and, depending on the variant, may also introduce additional PHY latency, timestamping uncertainty, or medium-access delay.

Table 8.2 compares the results from the selected SPE-related studies with the results obtained in this thesis. The table is not intended as a strict performance ranking, because the compared systems differ in physical layer, timestamping reference point, hardware platform, servo algorithm, topology, and measurement method. Instead, it summarizes which results are directly useful for comparison, which results are only qualitative, and which limitations must be considered before relating them to the developed STM32 implementation.

Figure 8.6 presents the PPS offset ranges from Wiesner and Kovácsházy together with the PPS offset ranges measured in this thesis. The figure uses minimum and maximum PPS offset values rather than only the mean value, because the width of the interval makes the practical spread of the synchronization error visible. The values from this thesis are included only as 100BASE-TX reference measurements: the direct link represents the best case of the implemented firmware, while the switch topology shows the effect of an ordinary non-PTP-aware switch.

The comparison shows that the direct STM32 implementation in this thesis reaches the same general order of magnitude as other microcontroller PTP implementations over 100 Mbit/s Ethernet, but it is less precise than the best 100BASE-T1 and gPTP results reported in the selected works. This is expected: the thesis system uses a simple custom PTP implementation, an ordinary Ethernet PHY, an ordinary switch in the multi-slave case, and no TSN queueing or switch-level PTP correction.

The comparison also shows that SPE itself is not a single answer. 100BASE-T1 appears to be the most promising SPE candidate for a future version of the implemented system because its data rate and full-duplex point-to-point operation are close to standard 100 Mbit/s Ethernet, and published results show very good synchronization performance. 10BASE-T1L and 10BASE-T1S are also usable for PTP, but they require more careful timestamping design. Their lower rate, longer symbol times, PHY processing, and, in the case of 10BASE-T1S, multidrop medium access can introduce additional timing uncertainty.

Table 8.2: Comparison of reported synchronization or timestamping results.

Source	**Physical layer / topology**	**Metric**	**Reported result**	**Adequacy of comparison with this thesis**
This thesis, direct link from Section 7.2	100BASE-TX, direct STM32 master–slave	Internal PTP log and external PPS	Log std. **173–243 ns**; log $p_{99}(\|offset\|)$ below **771 ns**; PPS p_{99} below **605 ns**	Direct baseline for the implemented firmware, but not SPE. Useful as a reference for 100 Mbit/s MCU-based hardware timestamping.
This thesis, switch topology from Section 7.3	100BASE-TX through ordinary non-PTP-aware switch, two slaves	Internal PTP log and external PPS	Log std. **214–335 ns**; log $p_{99}(\|offset\|)$ **575–945 ns**; path delay about **8.1–8.4 μs**	Relevant for evaluating switching and queueing effects. Not directly comparable with point-to-point SPE PHY tests because the switch dominates part of the delay variation.
Delbosco [11]	100BASE-T1 automotive gPTP setup with switch	gPTP offset and peer delay	Mean offset **34.40 ns**, std. **27.48 ns**; peer delay about **15.5 μs**	Good qualitative comparison for 100BASE-T1 and gPTP. Not strictly direct because the platform, protocol profile, timestamp reference planes, and switch architecture differ.
Wiesner and Kovácsházy [10]	100BASE-TX, 100BASE-T1, 10BASE-T1L, 10BASE-T1S point-to-point MCU platform	PPS offset, PI and Kalman servos	PI min/max: 100BASE-TX **-193.94/+214.06 ns**; 100BASE-T1 **-56.08/+75.92 ns**; 10BASE-T1L **-498.02/+497.98 ns**; 10BASE-T1S **-270.68/+349.32 ns**	Most useful quantitative SPE reference. Still not a strict direct comparison because firmware, PHYs, topology, servo, and measurement conditions differ from this thesis.
Brunner et al. [9]	10BASE-T1L, MII/RMII PHY latency study	Timestamping error and one-way latency, not full servo offset	Several-hundred-nanosecond timestamp inaccuracies possible; errors depend on PHY, interface, role, and link establishment	Not a servo result, but highly relevant for explaining why MAC-side timestamping may limit SPE synchronization accuracy.
Microchip LAN8650/1 [8]	10BASE-T1S MAC-PHY with hardware timestamping	1PPS comparison in a simple follower example	Maximum time difference **100 ns**; mean **8 ns**; std. **25 ns**	Shows potential of dedicated 10BASE-T1S timestamp hardware. Not directly generalizable to external PHYs or to a full multi-node IEEE 802.1AS implementation.

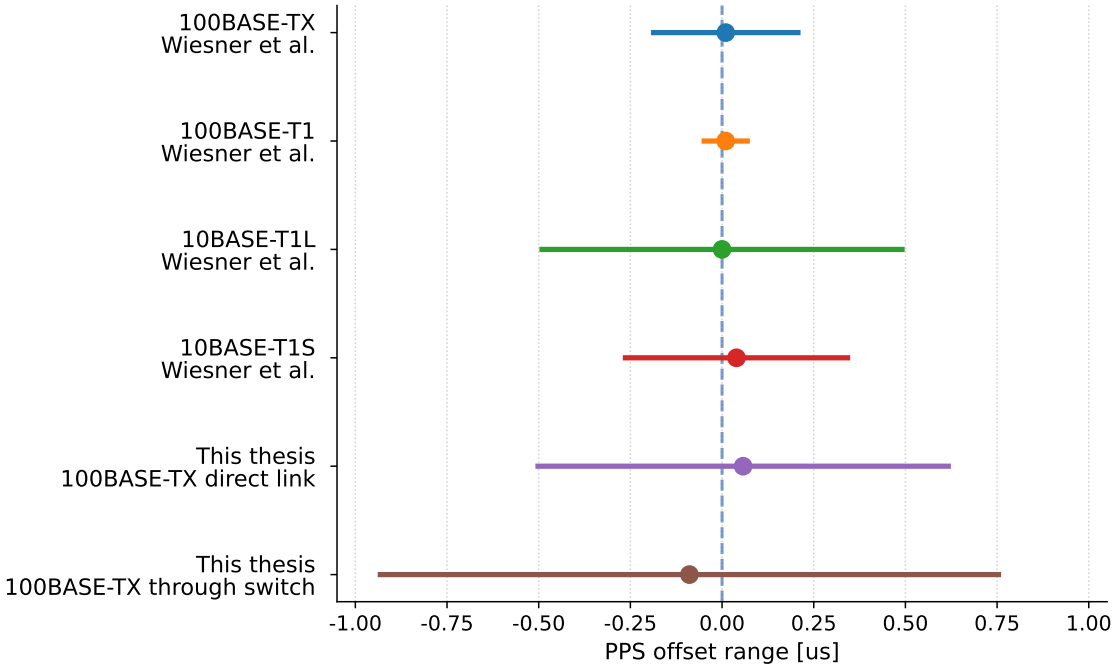


Fig. 8.6: Reported PPS offset ranges for Ethernet and SPE-based PTP synchronization.

8.6 Summary of the Possibilities for Using PTP over SPE

Single Pair Ethernet is technically suitable as a physical layer for transmitting IEEE 1588 PTP messages because it preserves the Ethernet frame service above the PHY. The PTP implementation approach used in this thesis, based on raw Layer 2 frames and hardware timestamps, could therefore be transferred to an SPE link without changing the logical PTP message format.

However, the achievable synchronization accuracy would depend strongly on the selected SPE variant and on the timestamping architecture. 100BASE-T1 is the most direct and promising candidate for sub-microsecond synchronization because it offers 100 Mbit/s full-duplex point-to-point operation and published measurements show very low offset values. 10BASE-T1L is attractive for long industrial links, but MAC–PHY timestamping can introduce several-hundred-nanosecond errors unless PHY latency and interface effects are handled carefully. 10BASE-T1S enables low-cost multidrop Ethernet segments, but it requires PHY-level timestamping and protocol adaptation for shared-medium operation.

The comparison with the results of this thesis should therefore be understood as an order-of-magnitude engineering comparison rather than a direct benchmark. The STM32 system developed here demonstrates that a simple hardware-assisted

PTP implementation can achieve sub-microsecond synchronization on ordinary Ethernet in normal conditions. SPE can support similar or better results, especially with 100BASE-T1 or dedicated timestamping MAC-PHY devices, but only if the physical-layer timing behavior is included in the system design rather than treated as an ideal transparent cable.

9 Possible Future Development of the Project

The implemented system demonstrates that IEEE 1588 time synchronization can be realized on STM32 Nucleo-F767ZI boards using raw Ethernet frames and hardware timestamping in the STM32 Ethernet MAC. The developed firmware is sufficient for the experimental goals of this thesis: a fixed master, one or a few slaves, Layer 2 PTP message exchange, offset calculation, filtering, and clock correction through the PTP addend. At the same time, the current implementation remains a focused laboratory prototype. Further development could make it more flexible, fault-tolerant, and suitable for more realistic embedded network scenarios.

The most relevant directions for future work are the transfer of the system to Single Pair Ethernet, implementation of automatic grandmaster selection, and redesign of the firmware architecture using a real-time operating system.

9.1 Transfer to Single Pair Ethernet

The analysis in Section 8.2 shows that the implemented Layer 2 PTP approach is not fundamentally tied to the 100BASE-TX physical layer. Since SPE preserves the Ethernet frame model, the same PTP messages could in principle be transported over an SPE link. Therefore, a future version of the project could replace the conventional Ethernet PHY with an SPE PHY while keeping the main PTP message format and synchronization logic.

For a direct replacement of the current laboratory setup, 100BASE-T1 would be the most natural SPE candidate because it preserves a 100 Mbit/s data rate and full-duplex point-to-point operation. For long industrial links, 10BASE-T1L would be more relevant, while 10BASE-T1S would be interesting for low-cost short multidrop segments. However, these variants would not be equally simple to integrate. The firmware would need to configure the selected external PHY, verify the MAC–PHY interface, and compensate the additional transmit and receive latency introduced by the PHY.

The most important practical issue is the timestamp reference point. In the current implementation, timestamps are taken in the STM32 Ethernet MAC. With an external SPE PHY, this timestamp no longer corresponds exactly to the moment when the frame appears on the physical medium. If the PHY latency is constant, it can be measured and compensated. If it depends on link establishment, MAC–PHY interface alignment, or medium access behavior, it becomes a source of synchronization error. For high-accuracy SPE synchronization, a PHY or MAC-PHY device with line-side timestamping would therefore be preferable.

The low-speed SPE variants also require additional attention to traffic load. The stress test in Section 7.5 showed that non-PTP traffic can disturb synchronization even in a 100 Mbit/s switched Ethernet network. On 10 Mbit/s SPE, the same amount of traffic would occupy a much larger part of the available bandwidth. For 10BASE-T1S, the situation is even more specific because several nodes can share one physical medium, and PLCA configuration or other access-control mechanisms become part of the timing behavior [5, 8]. Therefore, an SPE implementation should include traffic prioritization, careful filtering of invalid delay samples, and, for multidrop networks, adapted handling of delay request and response messages.

9.2 Best Master Clock Algorithm

The current implementation uses fixed device roles: one board is compiled as the master and the other boards are compiled as slaves. This makes the system simple and deterministic for laboratory evaluation, but it also creates an obvious limitation. If the master is disconnected, reset, or fails, the slaves have no mechanism for selecting a new time reference. As a result, the synchronization system depends on one manually assigned device.

A useful future extension would be the implementation of the Best Master Clock Algorithm (BMCA). With BMCA, all devices in the same PTP domain periodically exchange Announce messages and compare the quality and priority of available clocks. The network can then automatically select the best grandmaster. If the current grandmaster disappears, the remaining devices can detect the timeout and elect a new grandmaster instead of losing synchronization completely.

Adding BMCA would require several architectural changes. First, the firmware would need a common code base capable of operating both as a master and as a slave. Second, the PTP data set fields used for clock comparison would have to be stored and updated, including priority values, clock identity, clock class, accuracy, and variance. Third, Announce message generation, reception, timeout handling, and state transitions would need to be added. Finally, the clock servo would have to be enabled only when the device operates as a slave, while Sync and Follow_Up generation would become active only when the device becomes the selected master.

Such an extension would make the project closer to a complete PTP system. It would also make the network more robust, because synchronization would no longer depend on a single manually configured master device.

9.3 Migration to an RTOS-Based Firmware Architecture

Another important direction is migration to a proper RTOS-based firmware architecture. During the work, an initial experiment was made with FreeRTOS, and the slave firmware was transferred to a basic FreeRTOS project without major problems. This is expected because FreeRTOS is natively supported by STM32CubeIDE. However, this experimental version was not included in the thesis because it was not yet a correct RTOS design. It mostly reproduced the existing firmware structure inside an RTOS environment instead of fully separating responsibilities into well-defined tasks.

The motivation for continuing this work is practical. Experiments with artificial CPU load showed that the synchronization can degrade when other processing interferes with the PTP part of the firmware. In the current bare-metal architecture, the main loop must handle PTP processing, timestamp polling, logging, and auxiliary actions in one execution flow. If additional functionality is added later, it becomes harder to guarantee that PTP processing is always executed in time.

A well-designed FreeRTOS implementation could solve this problem by assigning priorities to different parts of the firmware. The PTP task should have a higher priority than diagnostic logging or application-level tasks. Ethernet receive events and timestamp acquisition could wake the PTP task through queues or task notifications. Logging could be moved to a lower-priority task so that UART transmission does not block time-critical synchronization logic. Additional application functions could then be added without directly disturbing the PTP state machine.

A possible task structure is the following:

- a high-priority PTP task responsible for message matching, delay calculation, filtering, and clock correction;
- an Ethernet event or driver task responsible for receiving frames and passing compact events with timestamps to the PTP task;
- a low-priority logging task responsible for UART or USB diagnostic output;
- optional application tasks for future measurement, control, or network services.

This architecture would make the firmware easier to understand and maintain. It would also simplify future extensions, because new functionality could be added as separate tasks instead of being inserted into one large main loop. At the same time, the RTOS version must be designed carefully. Shared data between the Ethernet driver, PTP task, and logging task must be protected, and priority inversion must be avoided. The goal is not only to run the existing code under FreeRTOS, but to make the synchronization path deterministic and isolated from non-critical processing.

Conclusion

The aim of this thesis was to design, implement, and evaluate software for time synchronization between STM32 Nucleo development boards using Ethernet communication and the IEEE 1588 Precision Time Protocol. The main objectives of the assignment were fulfilled. The STM32 Nucleo-F767ZI platform was studied, the principles of IEEE 1588 PTP were analyzed, suitable Ethernet software approaches were compared, and a working PTP-based synchronization system was implemented and experimentally evaluated on the selected hardware platform.

The practical implementation was based on direct Layer 2 Ethernet communication using raw Ethernet frames with EtherType `0x88F7`. This approach was selected because it avoids unnecessary TCP/IP stack overhead and gives direct access to the STM32 Ethernet MAC, DMA descriptors, hardware timestamping registers, and PTP time base. The final firmware implements a fixed-role PTP system consisting of one master and one or more slave devices. The master generates two-step `Sync`/`Follow_Up` messages and responds to `Delay_Req` messages. The slave receives and matches PTP messages, obtains hardware timestamps, calculates end-to-end path delay and clock offset, rejects invalid measurements, and corrects its local PTP clock.

During development, several practical implementation problems had to be solved. The first prototype used a PC running PTP software as a temporary master, but the software timestamping on the PC introduced high jitter and unstable delay measurements. This confirmed that both sides of the synchronization path had to use hardware timestamping. Further problems were related to the STM32 PTP time base configuration, correct acquisition of transmit timestamps from DMA descriptors, repeated generation of `Delay_Req` messages, and excessive sensitivity of early servo versions to raw offset samples. These issues were resolved by correcting the PTP clock configuration, explicitly requesting transmit timestamps, improving state handling, and introducing filtered and bounded clock correction.

The final slave clock correction strategy combines two mechanisms. Large initial phase errors are corrected by bounded time updates, while steady-state synchronization is maintained by a PI frequency controller that adjusts the PTP addend register. This separation proved to be more stable than direct phase correction alone. Several additional ideas were tested during development, including fine phase capture and warm start based on a stored addend. These mechanisms were not included in the final baseline because they did not provide a sufficiently consistent improvement for the tested setup.

The firmware was also extended from a simple two-board direct connection to a multi-slave topology. This required unique slave identities, correct filtering of

`Delay_Resp` messages, unicast responses from the master, a queue of received delay requests, and protection against mixing timestamp pairs from different synchronization cycles. As a result, the final implementation is not only a direct master–slave demonstration, but also a small experimental platform for evaluating synchronization behavior in more realistic local Ethernet conditions.

The experimental evaluation confirmed that the implemented system achieves stable sub-microsecond synchronization in normal operating conditions. In the direct master–slave connection, the measured path delay was approximately 1 μs with very small variation, and both the internal log-based evaluation and external PPS measurements confirmed synchronization errors in the range of several hundred nanoseconds. In the multi-slave topology through an ordinary non-PTP-aware Ethernet switch, the mean path delay increased to approximately 8.1–8.4 μs, but the synchronization remained stable and sub-microsecond for the majority of samples.

The network stress test showed the main limitation of the implemented architecture. With additional non-PTP traffic of 60 Mbit/s and 70 Mbit/s, the system remained synchronized, but a large part of the samples had to be rejected because of excessive path-delay variation. At 75 Mbit/s, the ordinary Ethernet switch introduced too much queuing delay and the slaves no longer received enough valid measurements to maintain stable synchronization. After the additional traffic was removed, the system recovered automatically. This demonstrates that the implemented algorithm is robust under normal controlled Ethernet conditions, but operation under high load through a non-PTP-aware switch remains limited by network delay variation.

The thesis also fulfilled the theoretical part of the assignment related to Single Pair Ethernet. SPE was evaluated as a possible physical layer for IEEE 1588 time synchronization. The analysis showed that the implemented Layer 2 PTP approach could in principle be transferred to SPE, because SPE still transports standard Ethernet frames above the physical layer. However, such a transfer would require additional hardware and timing analysis. The most direct candidate is 100BASE-T1, while 10BASE-T1L and 10BASE-T1S would require more attention to PHY latency, timestamp reference point, MAC–PHY interface effects, and, in the case of 10BASE-T1S, multidrop medium access.

Some parts of a complete industrial PTP system were intentionally outside the final implementation scope. The developed firmware does not implement the Best Master Clock Algorithm, so the master role is fixed at compile time. It also does not use a PTP-aware switch, transparent clock, boundary clock, VLAN priority, or a complete RTOS-based architecture. A preliminary FreeRTOS migration was considered, but it was not included as a final result because it did not yet represent a properly structured RTOS firmware design. These limitations do not prevent the

system from meeting the goals of the thesis, but they define clear directions for future development.

Overall, the work demonstrates that STM32 Nucleo-F767ZI boards can be synchronized using a custom IEEE 1588 Layer 2 implementation with hardware timestamping. The implemented system provides a functional and experimentally verified baseline for studying PTP synchronization on embedded Ethernet devices. The most important future improvements would be the implementation of the Best Master Clock Algorithm, migration to a well-designed FreeRTOS-based architecture, use of PTP-aware or QoS-capable network equipment, and practical verification of the same synchronization principle over Single Pair Ethernet.

Bibliography

[1] Alejandro Arteaga, Leire Muguira, Jaime Jimenez, José Gárate, and Armando Astarloa. A survey on IEEE 1588 implementation for RISC-V low-power embedded devices. *Electronics*, 13:458, 01 2024. doi:10.3390/electronics13020458.

[2] Luca De Vito, Sergio Rapuano, and Lucio Tomaciello. One-way delay measurement: State of the art. *IEEE Transactions on Instrumentation and Measurement*, 57(12):2742–2750, 2008. doi:10.1109/TIM.2008.926052.

[3] STMicroelectronics. *RM0410 Reference manual: STM32F76xxx and STM32F77xxx advanced Arm-based 32-bit MCUs.* STMicroelectronics, July 2024.

[4] Jiangsu Yangjie Electronic Technology Co., Ltd. *SMD Crystal Resonator Datasheet.* Jiangsu Yangjie Electronic Technology Co., Ltd. Datasheet. URL: https://www.lcsc.com/datasheet/C889706.pdf.

[5] George Zimmerman, Peter Jones, Jon Lewis, Piergiorgio Beruto, Steffen Graber, and Heath Stewart. IEEE P802.3cg 10Mb/s Single Pair Ethernet: A Guide. IEEE 802.3cg 10Mb/s Single Pair Ethernet Task Force tutorial, January 2019. Tutorial presented at the IEEE 802.3cg 10Mb/s Single Pair Ethernet Task Force Interim Meeting.

[6] Texas Instruments. Single-Pair Ethernet, the future of industrial communications: 10BASE-T1L IEEE 802.3cg single-pair ethernet phy. Texas Instruments New Product Update webinar slides, 2020. New Product Update: Ethernet — "Single-pair Ethernet, the future of industrial communications – 10BASE-T1L IEEE 802.3cg single-pair Ethernet PHY".

[7] Kirsten Matheus. One-pair Ethernet in the Automotive Industry. In *SPE Technology Days*, Virtual event, September 2020. BMW AG. Presentation at SPE Technology Days, September 22–23, 2020.

[8] Microchip Technology Inc. *AN1847: Time Synchronization using the LAN8650/1.* Microchip Technology Inc., June 2025. Application Note, DS60001847C. URL: https://ww1.microchip.com/downloads/aemDocuments/documents/AIS/ApplicationNotes/ApplicationNotes/LAN8650-1-Time-Synch-AN-60001847.pdf.

[9] Kilian Brunner, Luis Da Silva Miranda, and Martin Ostertag. Timestamp accuracy of 10BASE-T1L for TSN applications. In *2023 IEEE Third International Conference on Signal, Control and Communication (SCC)*. IEEE, 2024. Conference held in Hammamet, Tunisia, 1–3 December 2023. doi: 10.1109/SCC59637.2023.10527511.

[10] András Wiesner and Tamás Kovácsházy. Time synchronization performance of Single-Pair Ethernet (SPE) solutions. In *2025 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 1–6. IEEE, 2025. Paper 11281755. doi: 10.1109/ISPCS66324.2025.11281755.

[11] Fabio Delbosco. Automotive Ethernet and 100BASE-T1: State of the art of communication protocols and experimental evaluation of time synchronization using gPTP. Master's thesis, Politecnico di Torino, Turin, Italy, 2025. Master of science program in Computer Engineering; advisors: Stefano Di Carlo and Alessandro Savino. URL: https://webthesis.biblio.polito.it/id/eprint/38595.

List of appendices

A Selected Firmware Code Listings **97**

B Supplementary Measurement Material **102**

B.1 Log-based Measurement Interface 102

B.2 Supplementary Measurement Tables 104

C Contents of the Electronic Appendix **110**

A Selected Firmware Code Listings

Listing A.1: PTP hardware timestamping initialization.

```
static void PTP_HW_Init(void)
{
  uint32_t ptptscr = 0U;

  ETH->MACFFR |= ETH_MACFFR_RA | ETH_MACFFR_PM;
  ETH->MACIMR |= (1UL << 9);
  ETH->PTPSSIR = (uint32_t)PTP_SUBSEC_INCREMENT_NS;

  ptptscr |= ETH_PTPTSCR_TSE;
  ptptscr |= ETH_PTPTSCR_TSPTPPSV2E;
  ptptscr |= ETH_PTPTSCR_TSSPTPOEFE;
  ptptscr |= ETH_PTPTSCR_TSSSR;
  ptptscr |= ETH_PTPTSCR_TSSARFE;
  #if PTP_ENABLE_FINE_MODE
  ptptscr |= ETH_PTPTSCR_TSFCU;
  #endif
  ETH->PTPTSCR = ptptscr;

  ptp_default_addend = PTP_CalcDefaultAddend();
  ptp_current_addend = ptp_default_addend;
  #if PTP_ENABLE_FINE_MODE
  (void)PTP_ApplyAddend(ptp_default_addend);
  #endif

  (void)PTP_ApplyTimeInitialize(0U, 0U);
}
```

Listing A.2: Conversion from nanoseconds to STM32 PTP subsecond update format.

```
#define PTP_NS_PER_SEC      1000000000ULL
#define PTP_SUBSEC_PER_SEC  0x80000000ULL

static uint32_t PTP_NsToSubsec(uint32_t ns_val)
{
  uint64_t tmp;

  if (ns_val >= 1000000000UL)
  {
    ns_val = 999999999UL;
  }

  tmp = ((uint64_t)ns_val * PTP_SUBSEC_PER_SEC +
  (PTP_NS_PER_SEC / 2ULL)) / PTP_NS_PER_SEC;

  if (tmp > 0x7FFFFFFFULL)
  {
    tmp = 0x7FFFFFFFULL;
  }

  return (uint32_t)tmp;
}
```

Listing A.3: Direct RX descriptor access for received PTP event timestamps.

```
idx = heth_local->RxDescList.RxDescIdx;
idx = (idx == 0U) ? (ETH_RX_DESC_CNT - 1U) : (idx - 1U);

desc = (ETH_DMADescTypeDef *)heth_local->RxDescList.RxDesc[idx];
d = (uint32_t *)desc;

/* For received PTP event frames on this setup, d[6]/d[7]
carry the valid RX timestamp. */
if ((d[6] != 0U) || (d[7] != 0U))
{
  rx_hw_nsec = d[6];
  rx_hw_sec  = d[7];
  rx_hw_valid = 1U;
}

if (ptp_message == PTP_SYNC)
{
  g_sync_evt.pending = 1U;
  g_sync_evt.seq_id  = ptp_seq_id;

  if (rx_hw_valid != 0U)
  {
    g_sync_evt.sec  = rx_hw_sec;
    g_sync_evt.nsec = rx_hw_nsec;
    sync_rx_ts_valid_count++;
  }
  else
  {
    g_sync_evt.sec  = rx_sec;
    g_sync_evt.nsec = rx_nsec;
  }
}
```

Listing A.4: Direct TX descriptor access for the `Delay_Req` hardware timestamp.

```
delayreq_tx_desc_idx = heth.TxDescList.CurTxDesc;
txdesc =
(ETH_DMADescTypeDef *)heth.TxDescList.TxDesc[delayreq_tx_desc_idx];
td = (uint32_t *)txdesc;
td[0] |= (1UL << 25);

tx_t3_sec  = 0U;
tx_t3_nsec = 0U;

if (HAL_ETH_Transmit_IT(&heth, &TxConfig) == HAL_OK)
{
  delayreq_pending = 1U;
  delayreq_tx_ts_waiting = 1U;
}
else
{
  ptp_tx_req_fail_count++;
}

/* Later, after transmission, the same descriptor is checked. */
txdesc =
(ETH_DMADescTypeDef *)heth.TxDescList.TxDesc[delayreq_tx_desc_idx];
td = (uint32_t *)txdesc;

if ((td[0] & (1UL << 31)) == 0U)
{
  if ((td[0] & (1UL << 17)) != 0U)
  {
    tx_t3_nsec = td[6];
    tx_t3_sec  = td[7];
    tx_ts_seen++;
    delayreq_tx_ts_waiting = 0U;
    return 1U;
  }
}
```

Listing A.5: Core end-to-end delay and offset calculation on the slave.

```
t1 = PTP_ToNs(delayreq_ref_t1_sec, delayreq_ref_t1_nsec);
t2 = PTP_ToNs(delayreq_ref_t2_sec, delayreq_ref_t2_nsec);
t3 = PTP_ToNs(tx_t3_sec, tx_t3_nsec);
t4 = PTP_ToNs(resp_t4_sec, resp_t4_nsec);

delta_sync  = t2 - t1;
delta_delay = t4 - t3;

mean_path_delay = (delta_sync + delta_delay) / 2LL;
raw_offset_e2e  = (delta_sync - delta_delay) / 2LL;
ptp_offset_e2e  = raw_offset_e2e + PTP_STATIC_OFFSET_BIAS_NS;
```

B Supplementary Measurement Material

B.1 Log-based Measurement Interface

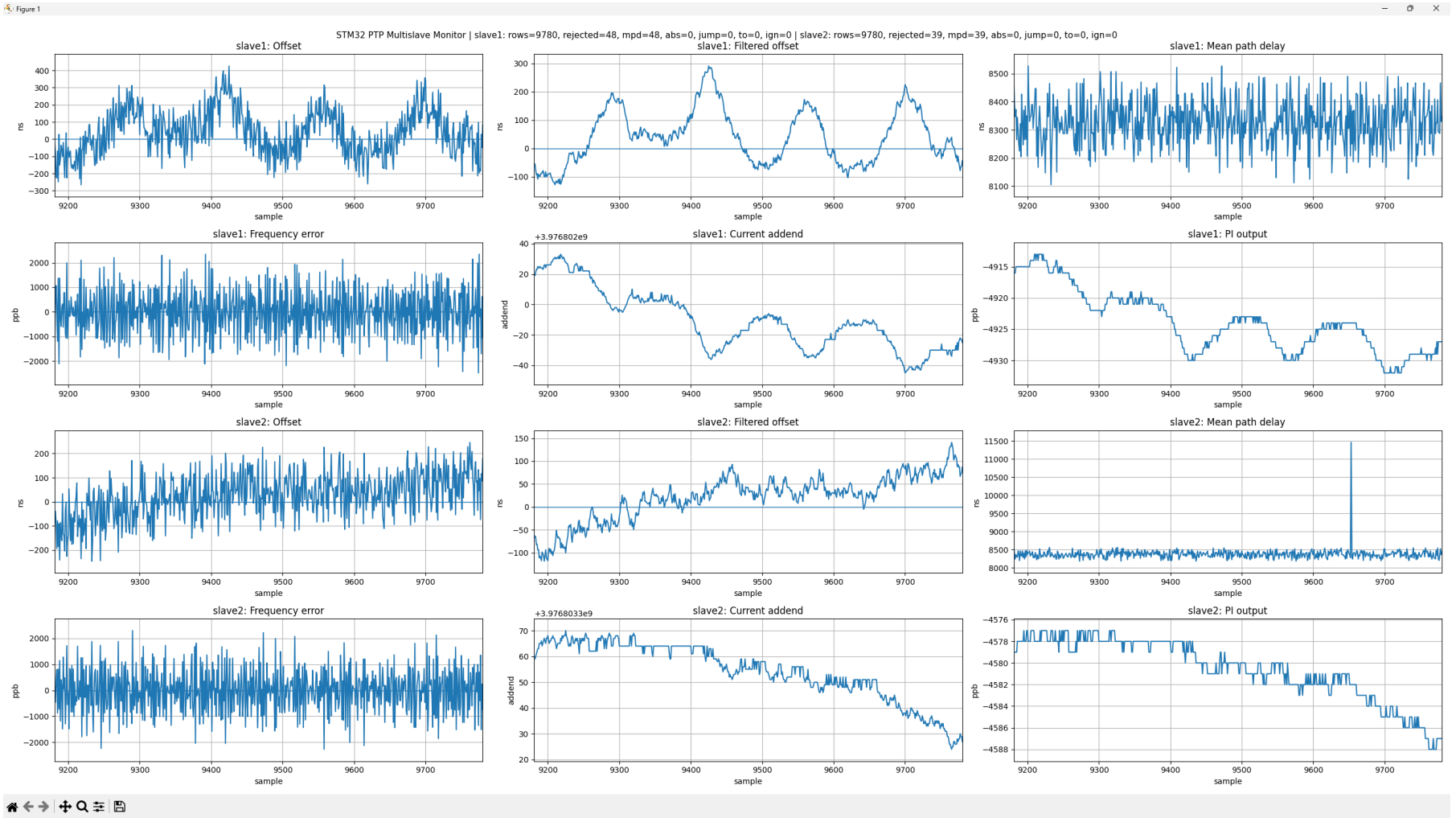


Fig. B.1: Example of the PC-side Python live logging and plotting window used during multicast log-based measurements.

B.2 Supplementary Measurement Tables

Table B.1: Direct master–slave connection: external PPS results.

Measure-ment	N_{PPS}	Mean offset [ns]	Offset std. [ns]	p_{95} ($\lvert offset \rvert$) [ns]	p_{99} ($\lvert offset \rvert$) [ns]	Min. [ns]	Max. [ns]
Unicast test 1	30	44	208	384	420	-356	430
Unicast test 2	30	-44	188	236	264	-381	273
Unicast test 3	30	-38	166	216	320	-322	360
Unicast test 4 7000–9000 samples	30	4	335	552	605	-470	624
Unicast test 4 13000–15000 samples	30	-50.56	253	320	456	-509	509
Unicast test 4 19000–21000 samples	30	-16.48	182	248	269	-374	274
Unicast test 4 25000–27000 samples	30	-76.92	144	135	223	-312	256

Table B.2: Direct master–slave connection: log-based results.

Measure-ment	Samples used	N samples	Mean offset [ns]	Offset std. [ns]	p_{95} ($\|offset\|$) [ns]	p_{99} ($\|offset\|$) [ns]	Min. [ns]	Max. [ns]	Mean path delay [ns]	Path delay std. [ns]
Unicast test 1	7000–end	42342	10	243	488	771	-970	1331	964	9
Unicast test 2	7000–end	2448	13	195	379	469	-502	540	968	9
Unicast test 3	7000–end	8309	22	173	325	405	-429	478	983	9
Unicast test 4	7000–end	20072	5	240	474	649	-780	774	980	8

Table B.3: Multi-slave connection through a switch: log-based results.

Measure-ment	Slave	Samples used	N samples	Mean offset [ns]	Offset std. [ns]	p_{95} ($\|offset\|$) [ns]	p_{99} ($\|offset\|$) [ns]	Min. [ns]	Max. [ns]	Mean path delay [ns]	Path delay std. [ns]
Multicast test 1	Slave 1	7000–end	20699	10	254	493	623	-1648	842	8353	102
Multicast test 2	Slave 1	7000–end	21231	1	327	644	920	-1943	927	8351	97
Multicast test 3	Slave 1	7000–end	21126	6	218	425	585	-1787	890	8335	103
Multicast long test	Slave 1	7000–end	43028	3	276	541	704	-1926	1016	8139	105
Multicast test 1	Slave 2	7000–end	20699	44	258	531	688	-2384	903	8162	108
Multicast test 2	Slave 2	7000–end	21231	10	335	643	945	-1888	1004	8155	103
Multicast test 3	Slave 2	7000–end	21126	4	232	442	575	-1982	867	8167	100
Multicast long test	Slave 2	7000–end	43028	11	214	421	586	-1864	816	8339	105

Table B.4: Multi-slave switch topology: PPS results for slave 1.

Measurement window	N_{PPS}	Mean [ns]	Std. [ns]	p_{95} ($\lvert offset \rvert$) [ns]	p_{99} ($\lvert offset \rvert$) [ns]	Min. [ns]	Max. [ns]
Test 1 samples 10000–13000	30	-6	335	504	653	-526	707
Test 1 samples 15000–18000	30	-41	256	354	354	-457	438
Test 1 samples 20000–23000	30	12	228	397	470	-382	488
Test 1 samples 25000–28000	30	24	172	333	404	-239	414
Test 2 samples 10000–13000	30	16	295	454	616	-526	663
Test 2 samples 15000–18000	30	-14	459	627	627	-939	761
Test 2 samples 20000–23000	30	19	296	465	553	-670	567
Test 2 samples 25000–28000	30	76	132	223	266	-274	279
Test 3 samples 10000–13000	30	9	266	491	553	-526	564
Test 3 samples 15000–18000	30	90	238	512	512	-297	652
Test 3 samples 20000–23000	30	90	143	294	370	-209	391
Test 3 samples 25000–28000	30	51	224	375	406	-354	417

Table B.5: Multi-slave switch topology: PPS results for slave 2.

Measurement window	N_{PPS}	Mean [ns]	Std. [ns]	p_{95} ($\lvert offset \rvert$) [ns]	p_{99} ($\lvert offset \rvert$) [ns]	Min. [ns]	Max. [ns]
Test 1 samples 10000–13000	30	182	249	646	723	-191	741
Test 1 samples 15000–18000	30	90	236	465	465	-326	510
Test 1 samples 20000–23000	30	88	208	365	407	-418	414
Test 1 samples 25000–28000	30	95	183	361	394	-270	407
Test 2 samples 10000–13000	30	143	278	637	662	-313	672
Test 2 samples 15000–18000	30	140	317	675	675	-311	704
Test 2 samples 20000–23000	30	96	231	502	572	-315	589
Test 2 samples 25000–28000	30	73	273	578	624	-282	634
Test 3 samples 10000–13000	30	81	241	443	464	-295	470
Test 3 samples 15000–18000	30	80	175	371	371	-294	536
Test 3 samples 20000–23000	30	77	144	328	350	-208	358
Test 3 samples 25000–28000	30	52	219	382	457	-292	486

Table B.6: Summary of the 0–60–70–75–0 Mbit/s network stress test.

Condition	Sample window	Slave	Rows	Rejected	Rejected [%]	p_{95} ($\lvert offset \rvert$) [ns]	p_{99} ($\lvert offset \rvert$) [ns]	Max. consec. rejects	Resyncs	Effective cycle [ms]
0 Mbit/s	7000–11400	Slave 1	4401	8	0.2	493	630	1	0	131.0
0 Mbit/s	7000–11400	Slave 2	4401	5	0.1	436	531	1	0	131.0
60 Mbit/s	11900–16318	Slave 1	4419	3651	82.6	976	1969	28	0	134.6
60 Mbit/s	11900–16318	Slave 2	4419	3643	82.4	718	1373	31	0	134.6
70 Mbit/s	21200–25618	Slave 1	4419	3930	88.9	1794	2053	47	0	137.8
70 Mbit/s	21200–25618	Slave 2	4419	3923	88.8	2256	2583	62	0	137.8
75 Mbit/s	27500–31812	Slave 1	4313	3981	92.3	$8.88 \cdot 10^8$	$1.06 \cdot 10^9$	63	4	140.2
75 Mbit/s	27500–31812	Slave 2	4313	3943	91.4	$5.44 \cdot 10^8$	$6.51 \cdot 10^8$	63	2	140.2
Recovery start	32000–36905	Slave 1	4906	2	0.0	465789	615273	1	0	149.5
Recovery start	32000–36905	Slave 2	4906	1	0.0	485095	714204	1	0	149.5
Recovery stable	37000–39961	Slave 1	2962	0	0.0	350	490	0	0	167.2
Recovery stable	37000–39961	Slave 2	2962	1	0.0	279	395	1	0	167.2

C Contents of the Electronic Appendix

The electronic appendix contains the source files, measurement data, and auxiliary scripts used during the development and evaluation of the implemented PTP synchronization system. The appendix is organized into several directories according to the type of files.

```
appendices ..... electronic appendix root directory
├── PPS_measurements ..... processed PPS measurements
│   ├── PPS_multicast_test.xlsx
│   └── PPS_unicast.xlsx
├── PTP_implementation_projects ..... STM32CubeIDE projects
│   ├── PTP_master ..... master firmware project
│   └── PTP_impl_01 ..... slave firmware project
├── script_for_logging ..... Python logging scripts
│   ├── ptp_live_plot_multicast.py
│   └── ptp_live_plot_unicast.py
└── UART_logs ..... raw CSV experiment logs
    ├── PPS_multicast_test1.csv
    ├── PPS_multicast_test2.csv
    ├── PPS_multicast_test3.csv
    ├── PPS_unicast_test1.csv
    ├── PPS_unicast_test2.csv
    ├── PPS_unicast_test3.csv
    ├── PPS_unicast_test5.csv
    ├── ptp_multislave_log_multislave_through_switch_50k_samples.csv
    └── ptp_multislave_log_stresstest_60_70_75_mbit.csv
```

The directory `PTP_implementation_projects` contains the complete STM32CubeIDE projects for the master and slave firmware. The directory `UART_logs` contains raw CSV logs used for the experimental evaluation. The directory `PPS_measurements` contains processed oscilloscope PPS measurements. The directory `script_for_logging` contains Python scripts used for UART capture and live plotting.

The firmware projects were developed and tested in **STM32CubeIDE v.1.19.0**. The projects target the STM32 Nucleo-F767ZI development board and can be rebuilt from the included STM32CubeIDE project files and CubeMX `.ioc` configurations. The Python scripts require a standard **Python 3.13.13** environment with packages for serial communication and plotting. The UART logs and PPS measurement files are included so that the statistical evaluation presented in Chapter 7 can be reproduced or checked independently.

Public Repository

The developed firmware and auxiliary scripts are also available in a public GitHub repository:

`https://github.com/LaGr0mpe/Bachelor_Thesis_Code`

The version corresponding to this thesis submission is located in the folder `thesis-submission-2026`. The electronic appendix submitted with the thesis contains the reference copy of the files used for the results presented in this work.