Study of the theoretical bounds and practical limits of time synchronization protocols using an Ethernet FPGA platform

Carles Nicolau Jené

Tesi Doctoral UPF / 2010

Dirigida per
Dra. Dolors Sala i Batlle
Departament de Tecnologíes de la Informació i les Comunicacions

Make everything as simple as possible, but not simpler.
Albert Einstein

Acknowledgements

I express my gratitude to my thesis advisor, Prof. Dolors Sala, for giving me the opportunity to be part of her networking research group and for joining different projects and research events along these years.
This work would also not exist without the financial support of the Department of Information and Communication Technologies of Universitat Pompeu Fabra. Thanks.
My gratitude to Prof. Enrique Cantó for his help on the daily problems with the FPGAs, and to all the tribunal for accepting the attendance to this thesis defense: Dr. Carlos Macián, Prof. Enrique Cantó, Prof. José Luís Marzo, Prof. Josep M^a. Fuertes and Prof. Boris Bellalta.
I am grateful to my research group colleagues for their time dedicated on me: Eduard, Khan, Javier, Ali and Darko.
I will be lifetime indebted to my parents for their love and faith on me, for listening to, for advising and for supporting me day after day along the years. I also wish to thank my brother, for his optimism and everlasting self-worth injections on me. Thanks.
Lastly, I want to dedicate this Thesis to my Dear Олеся. Без тбоей любви и веры в меня, я не смог бы закончит последнюю часть этого длинного и трудного пути. Спасибо тебе. Я тебя очень люблю.

Abstract

The goal of synchronization is to align/synchronize the time and frequency scales of all nodes within a network. In industrial applications, synchronization enables simultaneous triggering of distributed events and synchronous data acquisition at different nodes. For wide distributed systems, such as Internet, clock synchronization is advantageous for maintaining end-to-end Quality of Service (QoS).

Ethernet is the technology of choice for the future networks. Its low cost, the ever increasing data rates and low complexity and maintenance are key enablers for adopting it at all geographical scales and applications, ranging from the Network Provider to the industrial level. However, low cost and simplicity that characterizes the legacy Ethernet are only part of its attraction. The challenge is that it was initially conceived as a 'best-effort' and asynchronous oriented technology, limitations that difficult its adoption to handle, for example, time-sensitive applications in the industrial field, or carrier-class transport of services, from the Network Provider perspective. To better support new applications with tight synchronization requirements, standardization bodies and equipment manufacturers are making considerable efforts to extend its functionalities and release solutions to meet the synchronization requirements of new applications.

High accuracy time synchronization is a key enabler for offering such carrier-class QoS and handling distributed applications with stringent synchronization needs. Today's Ethernet-based approaches that deliver time synchronization rely on timestamped packets that distribute to the network. The acts of timestamping and sending the packet are crucial for achieving high accuracy synchronization, as they are exposed to a number of delay variabilities from the source to the destination node that impair the synchronization accuracy between nodes.

As the timestamping is a key component for actual synchronization protocols, the main goal in this work is to evaluate the impact of these sources of inaccuracies of Ethernet layers on the synchronization accuracy between nodes. The followed evaluation method is based on a real prototype utilizing low-cost platform Field Programmable Gate Arrays (FPGAs). The inherent complexity of these devices poses an additional challenge to the evaluation process, especially if the addressed synchronization accuracies are at the level of few nanoseconds. Therefore, this work also discusses and proposes methods to overcome platform-dependent limitations.

Additionally, this work proposes a different perspective for Ethernet technology which consists on envisioning the legacy Ethernet with a time synchronization functionality. We believe that such a new capability would allow Ethernet to better handle time sensitive applications and to be independent/compatible from/with the higher layers while keeping its initial philosophy: low-cost, simplicity and asynchronous technology.

Resum

L'objectiu de la sincronització és alinear/sincronitzar les escales de temps de tots els nodes d'una xarxa. En aplicacions industrials, la sincronització permet l'inici simultani d'esdeveniments distribuïts o l'adquisició de dades de forma síncrona als diferents nodes. En grans sistemes distribuïts, com per exemple l'Internet, la sincronització és beneficiosa per mantenir Qualitat de Servei (QdS) entre dos nodes distants entre si.

Ethernet és la tecnologia d'el·lecció per les xarxes del futur. El seu baix cost, les contínues actualitzacions de velocitat i la baixa complexitat i manteniment són els activadors per adoptar-la a tots els nivells geogràfics i aplicacions, des de Proveïdors de Xarxa en xarxes metropolitanes, fins a aplicacions industrials en xarxes locals. No obstant, el baix cost i simplicitat que caracteritzen a Ethernet constitueixen només una part del seu interés. El problema és que aquesta va ser originalment concebuda com una tecnologia de serveis mínims i asíncrona, dues limitacions que dificulten la seva adopció en aplicacions amb fortes restriccions de temps, tant en el camp industrial com en el transport de serveis de qualitat d'operadora. Per tal de suportar noves aplicacions amb fortes restriccions de temps, diversos organismes d'estandarditzación i fabricants d'equipament estan treballant activament per extendre les seves funcionalitats i llançar solucions per tal de complir amb nous requeriments de sincronització.

La sincronització de temps d'alta exactitud és clau per oferir serveis d'alt QdS i suportar aplicacions distribuïdes que necessitin fortes restriccions de temps. Les solucions d'avui dia basades en Ethernet que entreguen sincronització de temps es basen en distribuïr paquets amb una marca de temps a la xarxa. Les accions d'inserir la marca de temps i enviar el paquet són decisives per aconseguir sincronització d'alta exactitud ja que estan exposades a un nombre de variabilitats de retard des de l'origen fins el destí que empitjoren l'exactitud de la sincronització entre nodes.

Degut a que l'acció d'inserir la marca de temps és un component clau pels protocols de sincronització actuals, l'objectiu principal en aquesta Tesi és avaluar l'impacte d'aquestes fonts d'inexactitud de les capes d'Ethernet en la sincronització entre nodes. El mètode d'avaluació està basat en un prototipus real utilitzant plataformes basades en matrius de portes lògiques programables per camp (de l'anglès, Field Programmable Gate Arrays (FPGA)) de baix cost. La inherent complexitat d'aquests dispositius suposa un repte addicional al procés d'avaluació, especialment si s'adreçen exactituds de sincronització de nivells de pocs nanosegons. Aleshores, aquesta Tesi també debat i proposa mètodes per vèncer les limitacions dependents de la plataforma.

A més, aquesta Tesi proposa una perspectiva diferent per a la tecnologia Ethernet, la qual consisteix en extendre l'Ethernet inicial amb una funcionalitat de sincronització. Creiem que una funcionalitat com aquesta permetria a Ethernet suportar aplicacions amb fortes restriccions de temps amb independència de, i compatibilitat amb capes més altes tot mantenint la seva filosofia inicial: baix cost, simplicitat i tecnologia asíncrona.

Publications derived from this work

The results obtained from this Thesis derived on the following publications:

⇒ C. Nicolau, *A Zero-Nanosecond Time Synchronization Platform for Gigabit Ethernet Links.* In Proceedings of the 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom), Berlin, Germany, May 2010.

⇒ C. Nicolau, D. Sala, E. Cantó, *Clock Duplicity for High-Precision Timestamping in Gigabit Ethernet.* In Proceedings of the 19th International Conference on Field Programmable Logic and Applications (FPL), Prague, Czech Republic, August-September 2009.

⇒ C. Nicolau, D. Sala, *Flat-Soft Synchronization for Gigabit Ethernet.* In Proceedings of the 28th International Conference on Computer Communications, IEEE Infocom 2009 Student Workshop, Rio de Janeiro, Brazil, April 2009.

Contents

List of Figures

List of Tables

1.1 The Need for Synchronization in Ethernet

In the context of packet-based computer networks, synchronization is becoming a critical factor since the advent of the World Wide Web. Whereas in the early days it was only the exchange of 'best-effort' data, it now encompasses time-sensitive data, such as high-quality voice and video communication and a myriad of services that need exact knowledge of global time to render consistent results. Traditionally, time-sensitive data was transported by digital networks, such as Synchronous Digital Hierarchy (SDH) and Plesiochronous Digital Hierarchy (PDH). Digital networks are costly infrastructures in which the data transportation relies on providing precise and controlled timing to the overall equipment. The principle of digital networks consists on providing a high-stable timing signal coming from an expensive clock (e.g., a Primary Reference Clock (PRC)) to the physical layer of each node of network, thus making the network fully synchronous. Nowadays, access transport networks are evolving from dedicated-service, narrowband and highly bit-optimized PDH/SDH-based networks to multiservice, wideband and over-provisioned packet-based networks. This turn has been driven by an increasing demand of new broadband services, such as carrier-class voice, video over IP, IPTV, new applications, such as mobile wireless broadband services, and packet-switched concepts. All these applications demand synchronization accuracies that range from sub-microsecond down to nanosecond levels.

For high-capacity low-cost services to be commercially viable, telecommunication operators must decrease their per-bit production costs. Indeed, the technology that is on the spotlight since many years is Ethernet. It is becoming the technology of choice at local, metropolitan and wide area networks (WAN), as it offers simplicity, high data rates, plug'n'play network connection and more bandwidth at a lower cost than traditional WAN services. Ethernet was designed as a two-layer technology. In the Layer 2, or the *data link layer*, there was specified how the data is organized and sent over the network. In the Layer 1, or the *physical layer*, there was described the network medium and the signalling specifications. Currently, it operates from 1Mbps to 100Gbps, in different Layer 1 technologies (optical and electrical), and over large distances and different topologies (point-to-point and shared). The line capacity upgrades of Ethernet over the past years aimed at accommodating the increasing bandwidth needs many times. Modifications on the capacity do not involve any protocol functionality exten-

sion, but on over-provisioning the architecture to be able to provide quality of service[1] (QoS).

The work in this Thesis commits the adoption of Ethernet to Provider Networks and industrial field through a small extension of the existing architecture. The introduction of provider capabilities was initiated with the Ethernet in the First Mile (EFM) specification [IEEE Std. 802.3 (1998)] covering all type of broadband residential deployments (copper, point-to-point fiber and passive optical networks (EPONs)), together with the management capabilities. As far as the synchronization is concerned, EPON specification introduced a time synchronization capability based on timestamping for the purpose of coordinating the slotted access of point-to-multipoint networks. The synchronization capabilities in EPON were entirely defined at Layer 2. As far as the industrial field is concerned, Ethernet has attracted interest only recently. Actually, a number of vendors are offering industrial communication products based on Ethernet and TCP/IP as a means to interconnect field devices to the first level of automation. Others restrict their offer to communication between automation devices such as programmable logic controllers and provide integration means to existing fieldbuses [Decotignie (2005)]. In the area of industrial communications, synchronization is also delivered by means of timestamps within specific messages.

Kopetz was the first author who pointed that the synchronization accuracy among networked nodes using timestamping techniques is mainly affected by the sum of time variabilities during the insertion of the time information in the message, the transmission and propagation of the time message, and during the reception at the receiver side [Kopetz & Ochsenreiter (1987)]. Proper tuning of these main parameters will lead to a better synchronization. In order to know which are the synchronization limitations of Ethernet in shared topologies, the EPON timestamp and synchronization functionality should be refined in terms of jitter and resolution. Thus, this Thesis aims at characterizing accurately a Layer 2 synchronization mechanism for native Ethernet. A proper optimization would allow Ethernet to better support time sensitive applications with optimal QoS. Our objective is to improve Ethernet capabilities within an acceptable range while respecting its architecture simplicity principle.

1.2 Synchronization Provisioning for IEEE 802.3 Ethernet

Ethernet-based networks are asynchronous networks, i.e., they do not share a common timing signal as in e.g., SDH, but each node's physical layer is provided with a low-cost timing source for pacing the transmission and reception of the information. This principle decreases Ethernet's cost dramatically and allows to embrace a wide set of technologies built on top of it. The timing source is generally a low-cost quartz crystal oscillator from which a square clock signal is obtained and used to pace other circuitry within the node. Ethernet's Layer 1 are provided with hardware synchronization blocks that reconcile/synchronize

[1] In this Thesis term quality of service (QoS) refers to achieved service quality rather than the resource reservation control mechanisms.

the local timing from the neighbor node, thus creating a network of *free-running* nodes capable to communicate at any time and relegating services running on upper layers from synchronization.

Accordingly, synchronization transfer methods in Ethernet networks rely on synchronization protocols that distribute the timing using time marks (*timestamps*) inside the messages. Synchronization protocols define the message semantics, the rules for exchanging messages and the state information to be kept by the node. In a synchronization network there is usually a main node, commonly known as *best clock*, to which all the other nodes of the network synchronize to. Although there are different communication schemes (i.e., unicast, multicast or broadcast), the essence of synchronization protocols is similar: there is a periodic exchange of time messages from which the propagation delays are inferred and used to improve the synchronization. The nodes that need to be synchronized adjust their respective clocks accounting for the propagation delay. The more accurate, precise and deterministic the delay is, the more accurate the synchronization between nodes will be. It is precisely on the calculation of this parameter where synchronization fails; to calculate the delay, a node utilizes the remote timestamps which arrive indeterministically due to jitter sources present along the path of the message and within the same node.

To provide synchronization capabilities to Ethernet, several approaches have been proposed. Two distinguishable aspects exist between them: whether they provide synchronization in the form of *frequency* or *time/phase*, and whether they use dedicated hardware to carry out critical functions or not. The combination of choice depends on the synchronization needs of the application. For instance, backhaul cellular technologies require precise frequency and phase synchronization to the level of few parts-per-billion (ppb) and microseconds (μs) to efficiently share the radio access; time division multiplexed (TDM) networks have also the strong requirement of few ppb's frequency synchronization across the entire network, as accurate multiplexing and demultiplexing of data depends on that capability; a distributed measurement network running at ten gigabit speed requires time synchronization accuracies below few tenths of nanoseconds (ns) to distributedly log the interarrival times of the packets.

A technology that is gaining momentum among Tier 1 telecommunication operators to distribute frequency through Ethernet is Synchronous Ethernet (SyncE) [ITU-T G.8262 (2007)]. SyncE distributes frequency directly to the Ethernet physical layer. SyncE transceivers are provided with PLL-based mechanisms that pass timing from node to node in the same way timing is transported in, e.g., SDH. In IEEE 802.3 Ethernet networks, frequency synchronization is not mandatory as they do not need timing synchronization to work and, most important, they have been optimized for cost efficiency. Considering these requirements, other technologies built over Ethernet, such as the Network Time Protocol (NTP) [Mills (1992b)], or the more recent Precision Time Protocol (PTP)/IEEE Std. 1588 [IEEE Std. 1588 (2008)] are being developed and enhanced for high-quality time distribution.

Key to understanding synchronization in Ethernet networks is that each node suffers from an unsurmountable physical limitation: the local oscillator's fre-

quency drifts at a nominal rate of $\pm 50\mu$s/s (100 ppm), thus making nodes to drift apart few seconds per day from the ideal rate. Moreover, the drift rate can change due to temperature variations or aging, thus further impairing the synchronization. Oscillators' drift is the cornerstone problem of all synchronization mechanisms that drive synchronization protocols to re-synchronize periodically using messages. It is precisely in the process of the time message exchange where the synchronization accuracy between nodes is more impaired because the messages suffer from two error sources: *fixed propagation delay* and *propagation delay variability* (or *jitter*). Synchronization protocols face the complex task of estimating and cancelling the fixed propagation delay piggybacked on each timestamp. In the case of symmetric paths, the propagation delay can be inferred optimally (from the round trip delay (RTD)). On the other hand, jitter cannot be eliminated, but it has to be accommodated by the applications running at upper layers. The jitter is critical in synchronization and it raises from different sources in the path of the message:

⇒ *Internode jitter.* Corresponds to the time variability of an inbound/outbound synchronization message to cross the layered-stack. In IP-based synchronization schemes like PTP and NTP, this time penalty stems from the priority-based behavior of the operating system which queues the time message for a variable amount of time before sending. The most accurate solution to eliminate the gross part of the internode jitter is to generate and introduce the timestamp after the message has been queued. To carry out this approach, it is necessary to use specialized hardware capable to recognize the time packet while it is being sent/received. A common practice in Ethernet architecture consists on triggering the timestamp between the Media Access Controller (MAC) and the Ethernet transceiver (PHY). The paradigm of generating and inserting the time in a message by means of dedicated hardware is known as *hardware timestamping. Software timestamping* alludes to the same concept except that the timestamping process run on top of the jittery schedulling tasks of the operating system. Both timestamping paradigms establish the division line of performance and cost among the time synchronization protocols. Software-based protocols can be implemented in low-cost commodity hardware at expenses of poor synchronization accuracy, while hardware-based synchronization protocols leverage specialized hardware to generate timestamps with very low jitter. The use of hardware reports improvements of three or four orders of magnitude on the synchronization accuracy in many implementations.

⇒ *Network jitter.* Corresponds to the time variability of a synchronization message go across the layered-stack of the intermediate elements from a source node to a destination node. The network jitter is aggravated by the residence time variability that the packets experiment in the internal queues of the network elements (NE) like switches and routers. As switches are store-and-forward devices, the packets received on one port are stored temporarily while the device figures out which forwarding port(s) to send them

to. The time the packet is delayed is called the *residence time*. In scenarios with dense and variable network traffic, the residence time is variable and unpredictable. Moreover, topologies with several NEs suffer from the jitter introduced by each element, thus further aggravating the end-to-end synchronization. Although switch and router technologies and architectures have greatly improved over the years, packet delay variability (PDV) is still the more relevant limiting point to achieve accurate synchronization. Synchronization protocols, like PTP, tackle this problem in two ways. First, by defining profiles, i.e., specific network topologies with a maximum number of NEs to accommodate a permissible level of jitter. And second, by leveraging hardware assistance. To achieve *ns*-level synchronization, hardware assistance is a must. Hardware-assisted switches subtract, in the same NE, the time that a message is retained in the queue and modify the time information when the message is being transmitted. This technique is not widely available yet in commercial off-the-shelf products because there can be diminishing returns where more is not always better nor necessary. In the context of PTP these switches are called *boundary* and *transparent clocks*.

The actual technological trend in communication nodes points toward an increase in the use of specific hardware for the support of communication tasks. Although pure software systems have greatly increased their performance, hardware domains still achieve orders of magnitude better results. Known truth is that software systems feature more flexibility and short time-to-market than hardware systems, but the actual rapid evolution and creation of new standards and protocols has pointed hardware manufacturers towards more flexible and programmable hardware platforms to their products. This fact, taken in conjuction with the integrated circuit (IC) technological evolution (chip size reduction, power reduction and speed increase) and the integration of multiple systems in a single chip (SoC), is facilitating the integration of new functions into the hardware domain. A technology that is gaining force in the market is Field Programmable Gate Array (FPGA). FPGAs are programmable devices that have been developed as a form of intermediate approach: hardware design on a high-performance platform, optimal resources and programmability as the devices/systems within can be reprogrammed. In broad stokes, state-of-the-art FPGAs integrate thousands of programmable devices, hundreds of third part Intellectual Property (IP) blocks and microprocessor cores into one single part.

Over the last decades, the world of computing platforms has been progressing towards very complex systems for the sake of better efficiency. Actually, computing platforms are provisioned with extra resources capable to communicate at dizzying speeds. Architecture complexity has lead to the need to consider platforms as "black boxes" in order to simplify the design process. On the one hand, abstractions simplify the programmability as the designer 'only' needs to foucs on the application functionality. On the other hand, the fact of considering systems as black boxes lead to omit the hardware's behavior. Computation relies on hardware, and hence it cannot be ignored. Hardware systems are an integral part of the design, and thus the software must be designed to operate with it

accordingly. High-level programming languages and traditional programming is time-agnostic and provide functionality through ordering. Computer systems are provided with operating systems, with a rich suite of services that lack of a notion of time. To put an example, higher-level modern programming languages do not support the use of periodic interruptions in their semantics or counting the number of microprocessor clock cycles of a piece of code in a critical inner loop. Physical systems are intrinsically concurrent and temporal. Actions and reactions happen simultaneously and over time, consequently time plays an essential part in the system's behavior.

Embedded systems are on the other side of computing platforms. They are small computers that adhere to the world of embedded computing, characterized for having a limitation in the number of hardware resources and a direct interaction with them. In the embedded system world, time matters, and thus embedded-system applications are designed with low-level programming languages, more capable to provide notion of time and direct mapping to the hardware resources. For these reasons, embedded programming challenges to manage common hardware resources efficiently, reliably and, most important, in a predictable amount of time. In this view, the evaluation of synchronization protocols in embedded platforms entails an added problem that is *predictability*, or the lack of deterministic response and execution times in the interaction between software and hardware. Although FPGAs with processor cores provide an alternative architectural division between hardware and software, when time synchronization is needed at the level of *ns*, their interaction becomes more stringent. The software part, the "intelligent" side, hosts protocol management functions and rather complex arithmetic operations, while hardware is consigned to repetitive, non-complex but time-restrictive tasks. Hardware is put at the service of executing the requests of the software.

To summarize, the level of synchronization in Ethernet-based packet networks depends on the amount of jitter that exists between nodes. Jitter becomes more gross as more high in the protocol stack the timestamps are obtained. In IP-based networks the major part of the *internode jitter* is introduced by the stochastic behavior of the operating system. In pure Layer 2 Ethernet-based networks, the jitter sources arise from the time variabilities committed by the hardware mechanisms in the MAC and the PHY of the end nodes to transmit, propagate and receive the messages. This work focuses on accurately and precisely deriving the jitter at and below the MAC.

To tackle the problems stated above, this Thesis introduces a time synchronization functionality at Layer 2 for IEEE 802.3 Ethernet based on a readaptation of the EPON synchronization protocol. The synchronization functionality resides at the control path of the MAC, while the protocol is part of the MAC control sublayer. The protocol defines a set of control headers for delivering the synchronization and a message exchange pattern. The proposed model is implemented and evaluated in a low-cost FPGA-based embedded platform. The core of the time functionality falls in the hardware part, in the form of a timestamping block that permits to timestamp on-the-fly the time messages. The timestamping block characterizes for having a high-resolution time counter that

permits to timestamp very precisely the time messages. The timestamping block also records the time of flight of the time messages for deriving the jitter of the MAC and the PHY.

1.3 Technical Approach

This Thesis aims at evaluating what is the potential of Ethernet as a technology for delivering high-precision time synchronization. To that end, this work follows a methodology based on analysis, design and implementation:

⇒ *Analysis & study.* First, the existing technologies for delivering synchronization in Ethernet networks need to be explored and carefully studied to know their shortcomings and limitations. At the same time, a study of the actual synchronization needs of specific applications and services will give a set of specifications for the design phase.

⇒ *Design.* This phase comes up with a new model for IEEE 802.3 Ethernet to provide a time synchronization service. The design extension is fully prospected at Layer 2 and re-adopts the synchronization mechanism introduced in EPON.

⇒ *Implementation and Evaluation.* Ethernet functions are defined in the hardware domain, thus our conceptual design will be implemented in a hardware platform, a low-cost generic platform FPGA. As hardware design is an inherently complex task due to the number of physical limitations incurred in the design process, some parameters are very difficult to verify correctly. This Thesis will introduce a set of methods that will allow a correct verification of the proposed synchronization functions with independence of the platform limitations. Hence, this work will emphasize the implementation and evaluation aspects of the proposed mechanism.

1.4 Summary of Contributions

The contributions of this Thesis are:

⇒ An architectural extension proposal for IEEE 802.3 Gigabit Ethernet consisting on the introduction of a time synchronization functionality at Layer 2 (see Chapter 4). The synchronization function re-adopts the ranging procedure of the Ethernet Passive Optical Network (EPON) synchronization protocol. The function is fully prospected in the control plane of the MAC and introduces new control messages.

⇒ An optimal implementation of the time synchronization mechanism on a low-cost platform FPGA (see Chapter 5). The core of the mechanism is a hardware timestamping unit (TSU) that allows to timestamp on-the-fly the ingress/egress control frames. The TSU contains a high-speed time counter

that allows to timestamp the frames and synchronize with the maximum allowed precision given by the platform, $3.\widehat{33}$ *ns*.

The implementation of the synchronization function on a generic platform, the use of a high-resolution counter and the oscillator clock drift entail several challenging problems. The first one and most critical is that the clock frequency of the counter is asynchronous to the rest of the clock signals within the TSU. This can lead to unpredictable errors [J. Stephenson (Altera, Corp.) (2009)] in the captured timestamps that can totally invalidate the evaluation results (see Section 6.3.2). Second, the propagation time derived in the ranging procedure is not accurate, as it suffers from the processing delays within the node (see Sections 5.5.4 and 6.3.4). Third, the accumulated clock offset due to the clock drift when targeting synchronization accuracies at the level of nanoseconds impairs seriously the synchronization accuracy, therefore the real offset cannot be correctly verified (see Section 6.3.1).

The need to address these three limitations have lead to the following relevant contributions:

⇒ A hardware mechanism based on a digital circuit called *flancter* that totally prevents from timestamping errors in presence of multiple asynchronous clock sources (see Section 5.4.4). Our *flancter-flag* blocks accommodate the counter frequency of the TSU with the transmission and reception frequencies of the MAC and PHY interfaces. This block can be re-used in other designs with disparity of frequencies.

⇒ A method to calculate with high accuracy the propagation time in the ranging process, and thus to synchronize optimally. The method consists on capturing several distributed timestamps along the MAC layer during the synchronization message journey and re-use them to subtract the node processing delays. To be able to obtain these timestamps, the TSU architecture has carefully been designed (see Sections 5.4.2 and 5.4.3).

⇒ Two methods to evaluate the time synchronization accuracy with independence from the oscillator drift. The first one is based on the capture and comparison of the distributed timestamps between two nodes. The second one is based on the comparison of the instantaneous time of two remote nodes using programmable interval timers (PIT) (see Section 6.3.4).

The combination of the synchronization protocol, the hardware timestamping and a careful analysis of the platform architecture has rendered synchronization accuracies of zero nanoseconds in a point-to-point configuration (see Section 6.3.4).

1.5 Outline of the Thesis

After introducing the problem and the objectives in this chapter, the remainder of the thesis is organized in six chapters as follows:

Chapter 2 provides a summary of terms and concepts most used and relevant to this Thesis.

Chapter 3 reviews the state of the art work most related to this Thesis and summarizes the actual needs for synchronization of end-users and applications. It also reviews the problems that timestamping mechanisms usually suffer from.

Chapter 4 presents the first contribution of this Thesis, which is a time synchronization model for Gigabit Ethernet. It describes the followed conceptual approach while leaving aside implementation details.

In Chapter 5, we cover an important part of this Thesis which is the implementation of our conceptual model in a platform FPGA. Here, we present almost the rest of the contributions and explain in detail the main design components of our synchronization platform: the main hardware subblocks of our timestamping unit, the concept of distributed timestamping to re-synchronize and calculate the RTDs optimally and the *flancter-flag* blocks to prevent from timestamping errors.

Chapter 6 presents the experimental results for characterizing the timestamping mechanism, the internode jitter and the synchronization accuracy achieved in a point-to-point configuration. The last contribution of this Thesis is presented here: a method to evaluate the synchronization accuracy at nanosecond level with independence of the clock drift.

In Chapter 7, we draw conclusions and identify directions for future work.

This chapter presents most relevant terms and concepts used in this thesis. Especially it aims at differentiating the different ways that the term synchronization can be understood, along with its metrics and the methods to disseminate synchronization in a network of geographically dispersed nodes. Last, this chapter gives a tradeoff of actual computing platforms and why this thesis uses a platform FPGA as a validation tool.

2.1 What is Synchronization?

Synchronization is the act or result of synchronizing, namely, to represent or arrange events to indicate coincidence or coexistence [Merriam-Webster, Inc. (2010)]. In the context of networks, synchronization can be understood as the process that deals with the distribution of time and frequency across all the nodes of the network [Bregni (2002)]. Thus, the goal of a synchronization process is to align (i.e. synchronize) the time and frequency scales of all clocks of a network by using the communication capacity of their interconnecting links.

2.1.1 Different Meanings, Different Abstractions

The term *synchronization* has different meanings and challenges depending on the area of work, the level of abstraction and the context. For a software specialist, synchronization deals with the consistency between two files, i.e. which of the two files were lastly modified. For an expert on digital communications, the term is familiar on the acquisition and tracking of a clock in a receiver, with reference to the periodic timing information contained in the received signal. And for a digital circuit designer, synchronization deals with the circuit mechanisms to ensure reliable data passing among different groups of logic paced at different clock frequencies. Regardless of the abstraction level and area of expertisement is referring to, synchronization in digital systems is crucial as it ensures correct information flow and that operations follow in the correct order to obey a precedence and timing to obey deadlines. Each level of abstraction relies on the features of the abstraction level below and hides unnecessary details to the level above. Whichever is the abstraction criterion in describing hardware and software systems, the entities (networks, nodes, systems, blocks) are mutually correlated at any level, and the correct interoperation relies on a correct temporal coordination [Messerschmitt (1990)].

2.1.2 What Is A Clock?

The term *clock* has also different meanings depending on the level of abstraction and the context too. In the world of electronics, a clock is a digital signal consisting on periodic pulses. The pulses are generated by an oscillator, which relies on the performance of a resonator as a stabilizing element. Digital systems are entities that at register level exchange binary information at periodic intervals paced by the clock signal. As telecommunications equipment rely on smaller digital subsystems, their synchronization performance is at expenses of the subsystems below. Furthermore, in order to have an optimal coordination and a reliable information transfer among digital subsystems, the equipment has to share the same clock signal. In the real world there are no two or more clock signals that oscillate at the same frequency, as the resonator output frequency is affected by manufacturing deficiencies and environmental conditions (humidity, temperature, aging).

In personal computers, the clock is known as *system clock* and it is rendered to the user in the format of `year, day, hour:minute:second` [ISO 8601:2004 (2004)]. The formatting task is accomplished by a high-priority routine within the operating system after reading one of the several existing timing sources, either within the motherboard or the microprocessor, i.e., from the Real Time Clock (RTC) or the Timestamp Counter (TSC). Both counters are hardware registers that sum up ticks at a multiple[1] of the on-board resonator's frequency to keep track of real time.

In the context of synchronization of networks, every node of a network that has to synchronize to a reference node is a *clock*. The reference nodes have a better quality in terms of frequency stability, as they are generally locked to a high stable frequency signal coming from an atomic resonator. Atomic clocks are a kind of devices based on an atomic material and deliver a high stable frequency signal, generally in the flavours of 1, 5, and 10 MHz. Reference clocks in networks are dedicated high-performance computers that are locked to an atomic resonator in order to offer a high stable time to those client nodes with less stability.

2.1.3 Why Do We Need Synchronization?

The need for synchronization is global, in daily life, in electronic systems and in networks. Networks are built of electronic devices that are paced by drifty clocks that run at different frequencies. Consider for example the disparity of frequencies of the components in an electronic system. A processor of a personal computer may operate at 3 gigahertz and the data information travelling through the busses of a printed circuit board may run at hundreds of megahertz. In this case, the provision of synchronization mechanisms to adapt the information transfer speeds between the CPU and the bus is a must, otherwise the information would be lost. All the electronic digital systems are subjected to this ultimate level, thus

[1] It is common to fold the resonator's frequency by specialized frequency synthesis circuits inside the digital chips as actual quartz resonators can only resonate in the range of tens of kilohertz to tens of megahertz.

different re-synchronization mechanisms are provided along the information path, i.e., from the physical to the user level.

The lack or loss of synchronization has a different effect depending on where it takes place. For example, a loss of synchronization in the transfer of information from one register to another may raise to a misinterpretation of one or more bits. The type and mechanisms to provide synchronization depend on the synchronization abstraction level, while the specific requirements depend on the application needs. Consider another example, a voice communication over IP network (VoIP). The time delays and variabilities (jitter) are splitted along the path from the source to the destination nodes. Every node contains electronic circuitry that run at a different frequency and low-level synchronization mechanisms that add latency and jitter on the time needed for the packet to reach the destination. Besides, the intermediate elements (i.e., the switches and routers) store the packets for a variable amount of time that depends on the traffic load in the other ports. As in the case of VoIP, the packets need to be delivered to the user within a specific time interval to be sequentially and orderly processed on-time in the destination node for proper playout. However, if one or several packets containing the encoded voice data of the user is excessively delayed, the destination node will notice gaps of time with no voice.

All the waiting time counts, either coming from the latency and jitter added by digital circuits or the intermediate software processes that handle the packet before delivering it to the user. The crux of the matter is what are the synchronization requirements at a specific level of abstraction, and if that level can accommodate the delay and jitter introduced by lower layers.

2.1.4 Time, Phase and Frequency Synchronization

Synchronization is a generic concept that depicts the act of aligning the timing references of two or more entities. The timing references can be treated in different physical quantities, i.e. either in frequency, phase and time. Figure 2.1 shows an elementary view of two signals, A and B, being synchronized (subplots a, b, c) and unsynchronized (subplots d, e, f). Columns show the type of synchronization, i.e., frequency, phase and time:

- *Frequency synchronization*: As shown in Figs. 2.1a and 2.1d, frequency synchronization consists on matching the frequencies ($f_a = f_b$) of both signals, regardless of the phase difference ($\Delta\phi \neq 0$) between them. Conversely, the two signals are unsynchronized if the frequencies differ ($f_a \neq f_b$). In the context of networks, the term frequency synchronization is also know as *syntonization*.

- *Phase synchronization*: As shown in Figs. 2.1b and 2.1e, phase synchronization implies that the rising edges of the reference signal occur at the same instant ($f_a = f_b$, $\Delta\phi = 0$). Phase synchronization is more restrictive than frequency synchronization as it requires frequency and phase matching at the same time. In the context of networks, this term might include

the notion of frame timing, i.e., the point in time when the time slot of an outgoing frame has to be generated.

- *Time synchronization*: As shown in Figs. 2.1c and 2.1f, in the context of networks, time synchronization refers to the alignment the timescales of two geographically dispersed real-time clocks. Two nodes are time-synchronized if their timescales match ($C_A = C_B$). Notice from the figure that time synchronization is one way of achieving phase synchronization.

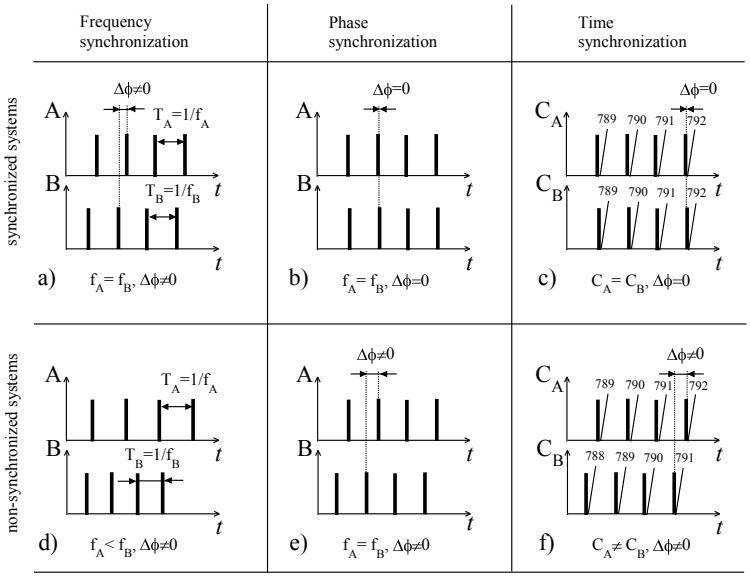

Figure 2.1: Physical quantities to represent synchronization.

2.1.5 Timing Between Signals and Systems

By definition, two digital signals or systems or networks are synchronous to each other if they share the same clock signal for transmitting and receiving the information. Synchronization of data passing between two nodes/networks is only necessary if the timing of the systems is different. If both systems work with the same clock, then they are synchronous and changes in the data from one system are always made at the same time in its clock cycle. The receiving system knows when the data is stable relative to its own clock and can sample it at that time, thus no synchronization is necessary. Figure 2.2 is inherited from [Messerschmitt (1990)] and illustrates a taxonomy of the timing relationship between two digital signals or systems. *Mesochronous* relationship is given to the systems that

do not share the same clock signal but they are tightly coupled, e.g., with the help of a phase locked loop (PLL), with small phase difference. Two systems are *plesiochronous* when both clock frequencies may be nominally the same, but the phase difference can drift over a period of time in an unbounded manner. Synchronization of two plesiochronous systems can be achieved by predicting when conflicts might occur, and avoiding data transfers when the two clocks conflict. If the time frame of one system is completely unknown to the other, there is no way that a conflict can be avoided and it is essential that data transfers are synchronized every time. In such a case the path is *heterochronous*.

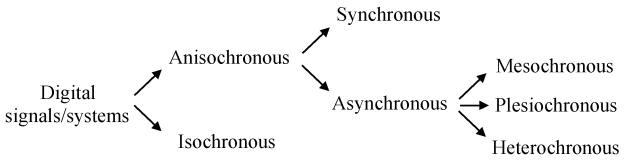

Figure 2.2: Synchronous and Asynchronous relationship between two digital signals, systems or networks.

2.2 Synchronization in Networks

2.2.1 Synchronous and Asynchronous Networks

Ethernet is well-known to be an "asynchronous technology". Each node of an Ethernet network has a free-run clock for sending and receiving information. As each Ethernet node runs at a different speed, synchronization mechanisms at the receiver node are required to *adapt* the sender datarate to its local clock. This first stage synchronization is entirely performed in the physical layer by hardware mechanisms. In Ethernet technology, the Ethernet transceiver (PHY) carries out this task [IEEE Std. 802.3 (2005)]. Once the information is synchronized and within the node, other types of synchronization are performed. For the specific case of Ethernet, each frame contains a special and fixed header (i.e., the *preamble*) to differentiate the frames.

Asynchronous networks are a type of networks with independent timed processors and hardware mechanisms to adapt the different datarates. This free-running characteristic is what differentiates asynchronous and synchronous networks. Asynchronous networks are architecturally simpler compared to synchronous counterparts, but the overhead of establishing the local synchronization is significant. On the other hand, in synchronous networks, all clocks are completely bound together through hardware mechanisms and equipment geographically dispersed. Many Network Operators are actually replacing synchronous technologies, such as SDH, to reduce cost in equipment technology, network setup and maintenance. Digital networks, such as SDH, where initially deployed in the 70s for the transmission of voice in digital telephone networks. Nowadays, with the advent of Internet and cellular mobile technologies, data traffic points

towards an ever increasing bandwidth demand, a parameter that traditionally synchronous technologies or circuit-switched networks cannot accommodate.

2.2.2 Actual Synchronization in IEEE 802.3

According to the definitions stated in Section 2.1.5, native Ethernet is a heterochronous technology, i.e., Each node of an Ethernet network has a free-running clock that paces the transmission of the data to the media. Ethernet was defined as a two-layer technology that addressed, in the Layer 2, how the data was organized and sent over the network medium, and in the Layer 1, how were the signalling and synchronization mechanisms. The actual synchronization provisioning in Ethernet is performed in the physical layer (PHY) to recover the analog signals from the cable to the digital data delivered to the MAC (L2). In the context of networks, an "asynchronous technology" is a technology that

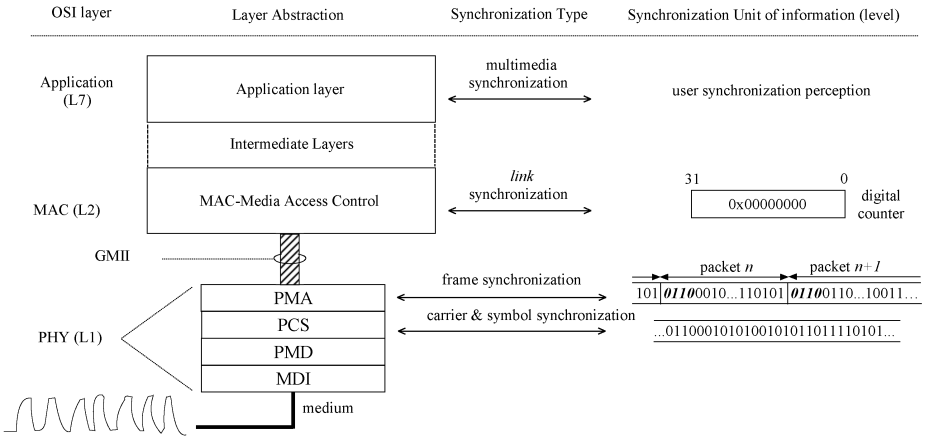

Figure 2.3: Actual synchronization functionalities in the OSI model.

do not share with other nodes a common timing from a high quality frequency standard source, but it uses a free-running crystal oscillator (XO) on-board to communicate with its adjacent node. In the case of Ethernet, this task is carried out by the PHY. During the link setup between two nodes, the autonegotiation function of the PHY decides which node acts as a master and which one as a slave. Hereafter, the master generates the transmit clock locally from the XO and the slave recovers the master clock from the received data and uses this recovered clock to receive data properly. After each packet transmission, the MAC enters in the interframe gap period (IGP), where the underlying PHY injects *idle* control codes to keep synchronized with the adjacent node until the next packet transmission. Without this Layer 1 (carrier, symbol and frame) synchronization it would not be possible for the PHY of the receiving node to receive incoming packets. This fact poses several pros and cons. On the one hand, the systems and the overall network is simpler and much cheaper than other technologies, such

as SONET/SDH or PDH. On the other hand, the overhead of establishing local synchronization is be significant for certain applications in terms of latency and jitter. Figure 2.3 illustrates different representations and synchronization techniques in the path of a message through the layered stack of an Ethernet node. The information in the medium is transported by means of analog pulses that are degraded by different sources of noise along the path. The PHY transceiver on Ethernet boards have the task to convert the incoming information from analog to binary format (*carrier & symbol synchronization*). This conversion is performed in the PCS sublayer, which contains complex hardware mechanisms such as noise filters, carrier recovery circuits, de-scramblers, decoders and so on. All Ethernet frames have a fixed header at the beginning of the frames that is used by the PHY to partition the string of decoded bits into frames (*frame synchronization*). The splitting of each frame into bytes is performed in the PMA sublayer and then it is passed to the MAC. Thus, time synchronization at Layer 2 (*link synchronization*) can be represented as a string readable format coming from a digital counter. Synchronization within the MAC is represented in multiples of byte (8, 16, 32 bits, and so on). It must be noted that frame synchronization does not result in link synchronization, but it is a method to obtain the frames from a bitstream of logical '0' and '1'. To provide time synchronization at Layer 2, there must be hard hardwared mechanisms within the MAC to update the digital counter and note the ingress and egress times of a frame.

2.2.3 Time Synchronization Protocols

A protocol specifies a collection of rules that describe message formats and the patterns for exchanging those messages. The same applies to time synchronization protocols which aim at aligning the timescales of geographically dispersed real-time clocks. They distribute time as a machine-readable string in specific messages and exchange them between nodes. Remote nodes collect the time inside the packets to find an agreement of global time. The strings with the time information come, either from a digital counter located in the hardware resources of the node, or from a software variable as a wasy to represent real time by means of a "synthetic" counter.

[Anceaume & Puaut (1997)] classify clock synchronization protocols in three major components. The first one is the *re-synchronization detection component*, which triggers the periodic indication for two nodes to synchronize their real-time clocks. The second one is the *remote clock estimation component*, which estimates the values of remote clocks in presence of error sources such as delay and jitter. And third, the *clock correction component* which corrects the local clock according to the result of the second component. The correction of the local clock can either be applied by brute force (state correction) or progressively (rate correction).

Part of the work of this Thesis can be better understood and classified according to those components and definitions. First, we define a set of messages to time synchronize at Layer 2 and a synchronization exchange pattern. We also set a series of periodic time-triggered events to start different processes, such as

remote clock observation, remote re-synchronization (i.e., remote clock state correction) and propagation delay calculation. And third, we estimate the remote clock values.

2.2.4 Delay, Jitter and Timestamping

As opposite to traditional digital networks (e.g., SDH/SONET or PDH), asynchronous networks or packet-based networks do not physically carry a common and dedicated frequency signal over the physical transceivers, but they transport time marks, or *timestamps*, embedded in specific time messages. The use of messages entails two additional problems that degrade the synchronization of a network. The first is the *internode jitter* or the jitter inside the node that is measured as the time variability of the synchronization message to cross the layered-stack. This error is mostly due to the scheduling of the operating system and internal message queues in retrieving/inserting the timestamp from/to the message. The most effective solution to the inter-node jitter is to timestamp at the lowest accessible level of the stack in order to bypass intermediate hardware. A common practice for Ethernet architecture is to trigger the ingress/egress timestamps between the MAC and the Media Independent Interface (MII). This practice is known as hardware timestamping. Although a gross amount of jitter is removed from the timestamps, they still accommodate the jitter and delay introduced by the physical layer. In Section 6.3.3, we will evaluate the delay and jitter introduced by the MAC and the PHY of a real platform.

The second problem is the network jitter or the node-to-node time variability of the synchronization message. This problem is due to the intermediate elements (e.g. switches) between two nodes. The solution to this problem is to use smart and well engineered synchronization protocols to infer the real delay in the network even under non-ideal load conditions and network heterogeneity, e.g., the well-know Network Time Protocol [Mills (2006b)] (see Section 3.2.2). Software-based synchronization protocols are algorithmically complex and not feasible to be implemented in hardware.

2.2.5 Synchronization Algorithms

As mentioned before, synchronization can be understood differently depending on its level of abstraction. In a similar way, synchronization algorithms have different meanings and approaches depending on the level of abstraction and whether they are designed to correct the frequency or the phase/time of a clock.

A synchronization algorithm as a whole can be understood as an "intelligent" entity that is capable of cancelling the error that might exist between an input and its reference, being the inputs either frequency or phase/time. In the case of time synchronization, they are also known as clock synchronization algorithms (CSA)[2]. As stated in Subsection 2.2.3, one of the critical blocks of time synchronization protocols is the *remote clock estimation component* (RCEC), which

[2]Clock synchronization is understood as the act of synchronizing the nodes of a network in frequency and in time/phase [Mills (2006a)].

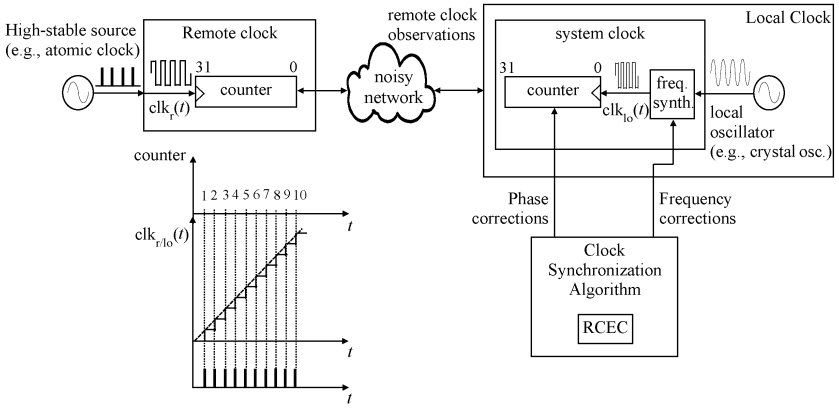

Figure 2.4: Clock synchronization algorithm scheme.

provides "intelligence" to the receiving system to infer the real delay of a message to go across the noisy network. Specifically, the task of CSAs is to estimate and compensate for the clock values between the remote clock and the local clock. Figure 2.4 gives an overview of the main blocks of a synchronization scheme. The remote clock provides high-stable time samples to the local clock. The RCEC applies the correction to the local clock according to the result of (more or less complex) computations from the timestamps. The counter can be, either modified abruptly (state correction), or smoothly (frequency correction). The former is computationally less complex, as the instantaneous value of the local counter has to be replaced by the remote timestamp. However, there are other potential problems on this type of clock correction related to e.g., the timescale similarity of the two clocks and timestamp errors when overwritting the local counter. Thus, usign a clock state correction approach, it is necessary to provide a reliable hardware infrastructure in order to prevent from timestamp errors. When the synchronization is performed in frequency mode, the modified variable is the clock signal (clk_{lo}) that comes from a frequency synthesizer circuit (*freq. synth*) within the node. To do that, it is necessary to have control of the frequency synthesizer, and to be able to infer from the timestamps the error term to correct [Aweya et al. (2006)], [Aweya et al. (2007)].

State correction is the method chosen in this Thesis to re-synchronize, as it the best choice to evaluate some synchronization components (see Chapter 6). In Section 5.4.3, we present a hardware block that we use to re-synchronize two contiguous nodes and to carry out the synchronization performance evaluation.

2.3 High-Performance Computing Platforms

In the last years, there has been a progressive increase in bandwidth demand per user worldwide, a trend that has pointed equipment manufacturers towards an increase of performance to accommodate more data in less time. At the same time, this has lead equipment to be provided with high-speed resources, such as communication interfaces, microprocessors, busses and memories. To accommodate the speed upgrades, equipment manufacturers are introducing architectural approaches to further increase the speed to process data. In one word, the trend is to parallelize the operations to obtain more efficiency.

2.3.1 Platform Technologies

Simplicity, high-performance, time-efficiency and flexibility are ever-desirable features that cannot co-exist at the same level. For this reason three different technologies have arised over the last years. Each one fits in a particular niche of application with different needs: general purpose processors (GPPs), Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs).

GPPs is the most used technology to those applications where time-efficiency and performance is not of a matter. They are relatively easy to program and there is a big community of experts that master the software skills for their programming. However, GPPs present a clear bottleneck in terms of performance: the computational tasks are carried out by a single entity, the microprocessor (μP), which in heavy workload scenarios lead to a loose of performance. This problem has given rise to move from centralized to distributed μP-based architectures in an attemp to become "more concurrent" for the sake of performance. Concurrency always comes at expenses of parallelism or redundancy of resources. Actually, parallelism exists in many different flavours in GPPs, such as multiprocessor systems, multicore systems, multithreading, grid and cluster computing, and so on. The rationale of these alternatives consists on folding up several times the μP entity in order to increase the computational time-efficiency by sharing parallel tasks. Despite these approaches have pointed to a performance improvement relative to single μP architectures, there is still a gap between the μP-based architectures and other technologies with more presence of specialized hardware.

ASICs and GPPs represent the two extremes in the flexibility - performance - cost trade-off. ASICs are custom chips designed for a specific set of very repetitive and time-demanding tasks with limited functional variability. ASICs, and custom hardware systems in general, exploit task parallelization in thousands of isolated circuits within a chip. Each circuit performs a specific and invariable task. Functional variability in the hardware domain implies a huge increase in circuit resource provisioning and utilization. Accordingly, these circuits are not suitable for the realization of prototyping functions since the chip has to be substituted. Another added drawback is in the design development which requires knowledge in different disciplines, such as logic design, analog electronics and chip layout design, thus making it inaccessible to small teams. Prior to the submission

the fabrication of the chip, an intense verification process of its functionality must be carried out as their updates are extremely expensive and, once the fabrication process has started, it is impossible to correct last minute bugs. As a way of illustration, the total cost of an ASIC can be easily a million dollars accounting for engineering costs and chip production [ETH - Microelectronics Design center (2010)]. ASIC designs are then only suitable for fixed functions that can be used in millions of identical chips, so that the investment will pay off.

FPGAs are in the middle way of the trade-off exposed at the beginning of this section. State-of-the-art FPGAs pose high degree of flexibility (or programmability), hardware performance and a cost comparable to computer-based systems. FPGAs are provided with a software plane that provides controlability and "intelligence" to a bunch of programmable hardware resources. In this sense, the hardware can be user-customized to a specific application requirements to perform time-demanding tasks which, at the same time, can interact with the software. In FPGA-based systems, the logic is directly designed using hardware description languages like VHDL or Verilog, which are intrinsically adapted to carry out parallel processing and pipelining. While designing with HDL requires expertisement in digital logic synthesis, the software part is programmed with standard programming tools, such as C/C++ flavours. Conversely to ASICs, today's FPGAs are supported by vendor specific tools that facilitate its overall design process. Traditionally FPGAs have been committed to early stages of product release but nowadays they are more present in deployed infrastructures, offering thus the possibility for in-situ hardware updates.

For the reasons explained above, the FPGA technology has been chosen to prototype in this Thesis.

2.3.2 FPGA-based Embedded Platforms

FPGAs are programmable logic devices that can be re-programmed to implement any function within the device resources. The internal architecture of an FPGA consists of an array of logically uncommitted elements that can be interconnected to carry out a specific application [Brown & Rose (1996)]. The three basic internal resources that use to characterize FPGA architectures are: a) combinational logic blocks (CLBs) or logic elements (LEs), b) interconnection (between the CLBs), and c) I/O or I/O blocks (IOBs). The combination of LEs and interconnect is known as FPGA fabric, which is shown in the right side of the Figure 2.5. The CLB can perform the combinational functions of several logic gates and/or the sequential functions of memory circuits. A CLB is further subdivided in slices, which in turn, contain function generators (Look-Up Tables (LUTs)), flip-flops, combinational logic, multiplexers and carry logic. Provided that an FPGA contain thousands of CLBs (depending on the FPGA model), a large amount of combinational and sequential logic functions can be embodied in the fabric. The IOBs contain programmable logic to be inputs or outputs. They are intended to interface to the external pins of the FPGA and they allow several I/O signalling standards to be interfaced depending on the setup. The interconnections are

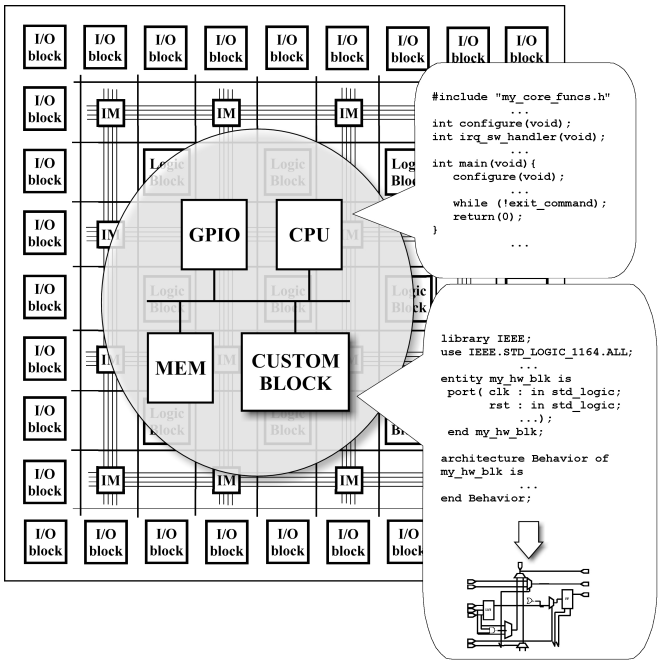

Figure 2.5: Generic FPGA internal architecture. Programmable hardware (*custom block*) is described with HDL languages. The application software executed by the CPU can be described using standard programming languages, such as C.

predefined tracks logically divided into channels that link all the configurable logic.

Early FPGAs were used as a glue to connect various elements within a system. Nowadays, and thanks to the technology improvement, FPGA devices have provided such increase in capability that they have been moving from the edge to the center of the design and they can be considered as a fundamental computational element. Today's FPGAs are platforms that integrate a wide variety of hard and soft intellectual property (IP) cores on a single device whose hardware and firmware can be upgraded at any time. Hard IP cores are a dedicated part of the integrated circuit, whereas soft IP cores are implemented utilizing general-purpose FPGA logic cells. The shaded circle in Figure 2.5 denotes a group of hard IP cores (*GPIO*, *CPU* and *mem*) and a soft IP core (*custom block*) hooked on a bus for communicating themselves.

FPGA's reconfigurable logic is particularly suitable for parallel algorithms implementation. However, sequential algorithms, especially those that don't demand huge processing power, are easier to implement as a program for a CPU. The CPU core controls the functionality of the hardware part and does more complex calculations, while the hardware parts are responsible of more intensive operations.

A FPGA-based platform is a predesigned architecture that designers can use

to build systems for a given range of applications. The programmability of the architecture reduces system development time yet enables a single Platform FPGA to be targeted at multiple applications. This programmability also enables designers to optimize systems throughout the development cycle and offers co-design flexibility for trading-off hardware and software design implementations.

2.3.3 Hardware/Software Co-design

Hardware/Software (HW/SW) Co-design refers to the simultaneous consideration of hardware and software within the design process of certain applications. HW/SW partitioning maps a design onto the target architecture that includes a CPU and several IP blocks. A decade ago HW/SW Codesign was a discipline that intended to explore the design space of an application to create a suitable platform. Actually HW/SW partitioning is moving towards a pactical design task thanks to reconfigurable computing. FPGA's internal architecture is what HW/SW partitioning targets, i.e., reconfigurable logic to create custom hardware blocks specialized to repetitive and time-demanding tasks, and embedded CPU's to allocate standard applications. Making the best use of FPGA-based platform implies to map the application needs into the specific blocks of the internal architecture, either to the CPU or to the programmable hardware. This entails a problem that has been lasting for long time, which is the communication uneffectiveness between the programmable substrate and the CPU. There are several sources of delay, such as the bus arbitration delays, physical communication delays and re-synchronization delays that can totally make useless the processing and speed gain of the hardware logic.

Modern design requires a designer to have a unified view of software and hardware, seeing them not as completely different domains, but rather as two implementation options along a continuum of options varying in their design metrics (hardware cost, performance, power, flexibility, etc.). In the context of time synchronization the interaction of the software to the hardware is critical. For example, it is of no use having specialized high-resolution hardware to timestamp frames if then the time information is unduly delayed and overriden by software latencies. Or, also, it is of no use for a time synchronization protocol to timestamp in a non-predictable fashion and timestamp events unaccurately.

All these factors of real platforms must be taken into account in the evaluation of designs and, especially, they cannot be obviated in an evaluation using a platform FPGA.

2.3.4 The Cost of Hardware Design

Contrarily to pure simulation systems, where the testing of the fundamental characteristics of a design does not depend on the platform architecture and technology, in the world of hardware design, the platform characteristics plays a fundamental role to verify the functionality of a design. Designing with hardware implies to interact with real physical devices subjected to uncontrollable and unknown sources of error that difficult the overall design process.

The methodology for designing with FPGA slightly changes from manufacturers; Figure 2.6 shows a generic FPGA design flow. In the first stage, the system functionality of the hardware is described with a hardware description language (HDL) (e.g., VHDL or Verilog), which does not consider architectural properties nor any physical restriction. In the synthesis phase, the circuit behavior described by the HDL code is turned into netlists of logic gates. So far, the design is purely conceptual and its functionality is verified through Behavioral simulation. The translation stage opens the pool of the physical design processes, and merges the netlists from the synthesis phase. The mapping stage ensures that the merged netlists fit into the available logic resources, dependent of the FPGA model. The last two phases are the placement and routing, which determine the positions of the logic, the I/O Blocks and the overall connectivity along the chip. The results of each phase must be individually verified for good functionality conformity through simulations at different levels. Each phase in the design process adds more complexity to the system by adding more physical restrictions. This is especially noticeable when crossing the physical design phase border, where the design process becomes completely different. The design flow is left at expenses of the physical design tools, which are basically complex computational processes that translate an implementation-agnostic description to a real system with limited resources and propagation delays.

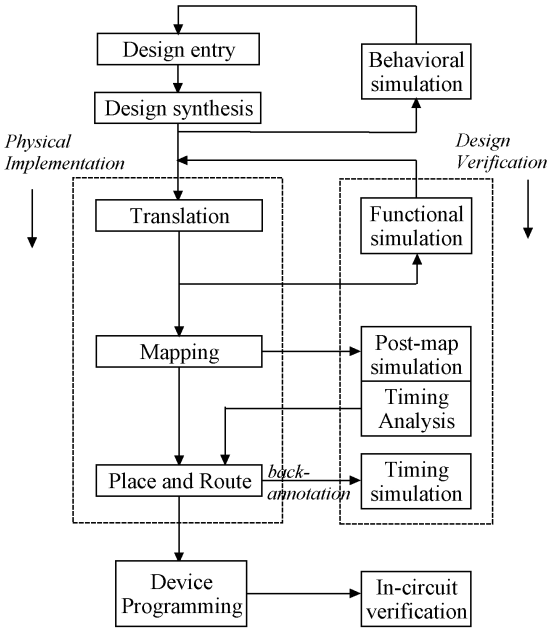

Figure 2.6: Generic FPGA design flow.

In order to have a successful physical implementation, the physical constraints must be provided before the synthesis stage. Physical constraints specify the fundamental timing properties of the physical design, e.g., clock frequencies of clock

signals, cross domain crossings, etc. If the physical constrains are too relaxed, the physical design process will be likely coped at expenses of a modest performance. On the other hand, if the physical delay constraints are too constrained, in the best case, the time needed for the tools to end up the physical design phases will grow exponentially. In the worst case, the physical process will preempt reporting errors like lack of chip space or bad timing.

The FPGA design tools basically report three fundamental design parameters after the Place & Route: the maximum frequency (speed) at which the internal devices can properly work, the number of used programmable devices and the overall chip power consumption. Designs tightly constrained[3] in terms of those three parameters are likely susceptible to misfuction or, if not, to show a much lower performance than the targeted at the initial phases of the design. For those hard constrained designs, it is necessary to use the Place & Route tools to fix the critical errors manually. As it will be seen in Chapter 5, some of the requirements to implement our platform will force us to manually fix the critical routing problems. On this cumbersome task, there is an added difficulty: the design tools are prone to errors that mislead the real final result report, especially on hard constrained designs. A reported frequency within the FPGA might be different from the real one, or the reported number of used slices might exceed the real available resources of the FPGA.

In order to evaluate a specific functionality in a network of FPGA-based nodes (like in the case of the system evaluated in this Thesis), it is necessary to mirror the hardware part and build the software design part according to the functionality of each particular node. The reason is that the FPGA design tools produce a different result after each run. The development tools rely on complex algorithms that might be affected, e.g., by a change of input parameters, or simply, by a change of the computer at which the tools are running. Thus, of utmost importance is, after the routed design functionality is correctly verified, to duplicate the hardware part and work independently with the software program.

In summary, in the world of FPGA design, there is a long gap in performance (speed, used programmable resources and power consumption) from the initial specifications to the final implemented design. To take advantage of the many benefits FPGAs provide, the designer needs to be versed in a wide range of skills from systems, hardware and software perspectives. Few technologies require as broad an experience base. Designing with FPGAs means aforced steep learning curve, complex design entry, multiple tool options and a myriad of design decisions.

[3] Designs demanding, e.g., low power consumption, high internal frequencies and small programmable area used.

§ 3. State of the Art of Synchronization in Ethernet-based Networks

This chapter reviews the actual work most related to the contributions presented in this Thesis. It is divided into two main blocks: synchronization protocols, with emphasis on time synchronization protocols, and its essential component: timestamping. In the former, we cover the most prominent works by classifying them following a common criteria. In the later, we review the existing timestamp methods.

3.1 Introduction

As explained in Chapter 1, a crucial aspect for achieving accurate synchronization between two nodes is in the timestamping process, i.e., in the jitter and delay introduced by the chain of mechanisms that retrieve a timestamp, insert it in the packet and send it to the network. The higher the number of elements in the chain for obtaining a timestamp, the higher will be the delay and jitter. Therefore, to get rid of gross inaccuracies, it seems obvious to allocate the timestamping function at the lowest level as possible. Indeed, today's synchronization approaches partition applications and distribute independent modules strategically within the architecture in order to have a deterministic behavior. This is the case of timestamping blocks. They are hardware mechanisms, which inherently characterize for having a low-level of jitter and latency.

Besides the delay and jitter of the timestamp reading, there are also other sources of uncertainties, such as the variability for inserting the timestamp in the message, transmitting it and, in the reception node, for receiving it. The journey of a message follows a path that consist on a series of digital mechanisms that store, process and forward the message, and add delay and variability. At the same time, the variability introduced by the digital circuitry of these blocks stems from the skew of the electrical clock signal within the chip or board module. Although the variability is seen as a whole from the user perspective, it can be dissected at different conceptual levels to observe the uncertainty added by each logic device.

In the next sections, we will review up-to-date methods and tools to evaluate the timestamp accuracy at different levels within the layered stack. Before that, a classification of the synchronization schemes, protocols and applications are presented. Special emphasis is given to study the synchronization solutions applied

to Ethernet. We will also discuss different tradeoffs and scenarios of choice using Ethernet among the available synchronization technologies.

3.2 Protocols

Currently, three standardization bodies cope the actual progress on synchronization protocols for packet-based networks: the Institute of Electrical and Electronics Engineers (IEEE), the International Telecommunication Union (ITU) and the Internet Engineering Task Force (IETF). Each body addresses synchronization requirements, targets and scope for specific applications. Next, the solutions provided by each of these bodies will follow.

3.2.1 Pure Hardware Approaches

In this work, we understand *pure hardware* approaches as those methods and mechanisms that deliver synchronization, either in frequency or in phase/time flavours, using methods that basically rely on specialized hardware. These methods are described next following the classification presented in the previous section.

Synchronous Ethernet

Many existing networks have a strong requirement of frequency synchronization across the entire network, as accurate multiplexing and demultiplexing of data depends on that capability, e.g., TDM networks. Transport of frequency has traditionally been performed for years using well-known principles, engineering rules and experience. This knowledge is being applied to Ethernet-based packet networks to replace legacy equipment. A technology that is gaining momentum for distributing frequency over Ethernet is Synchronous Ethernet (syncE). The principle of syncE is similar to SDH, i.e., to create a chain of clocks over Ethernet traceable to an external high stable frequency source (PRC/PRS) (see Figure 3.1). The first clock of the chain receives the PRC/PRS signal and propagates it down to the chain. To that end, each Ethernet physical layer of the chain contains a clock recovery mechanism (e.g., PLL) that recovers the clock from the parent node. SyncE capable devices are multiport specific cards provided with high stable backup oscillators for high precision clock recovery. The stability of the oscillators in syncE is ± 4.6 ppm and is specified in G.781 ITU-T's recommendation [ITU-T G.781 (1999)]. Up to date, syncE has been described by experts within the ITU-T Study Group 15 [ITU-T Q13/15 SG (2008)], the responsible group for studying timing and synchronization, and it is going to be specified within ITU-Ts G.Pacmod and G.Pacloc groups. SyncE has inherited other properties described in ITU-T recommendations, such as G.8261 [ITU-T G.8261 (2008)], G.8262 [ITU-T G.8262 (2007)], G.8264 [ITU-T G.8264 (2007)], G.781 [ITU-T G.781 (1999)] and so on. These standards specify different requirements such as the jitter and wander tolerances, supported frequencies, clock specifications, clock selection and quality levels, error responses, noise transfer

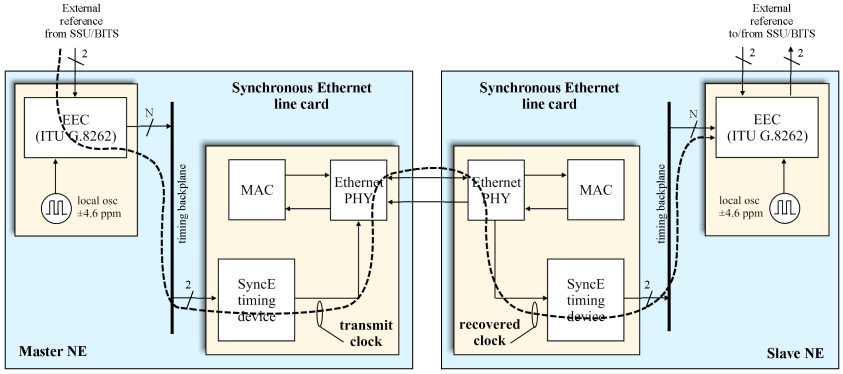

Figure 3.1: Synchronous Ethernet cards architecture.

limits, holdover performance, deployment scenarios, interworking requirements, etc. Furthermore, syncE is sponsored by several main Tier 1 telecommunication operators and semiconductor manufacturers, a fact that points it as the candidate technology for carrier-grade frequency delivering over Ethernet.

SyncE is a solution to deliver synchronization over the Ethernet media types defined in IEEE 802.3 [IEEE Std. 802.3 (2005)]. Theoretically, it is a low-cost solution to distribute frequency compared to traditional TDM-based equipment that provides packet delay variation independency and integrability with existent SONET/SDH networks. Although its potential, SyncE is still in its development phase and actual deployments are propietary networks. Corporate users of legacy data connections are reluctant to the substitution of their equipment with Ethernet alternatives due to possible sevice interruptions of initial trial periods and the costs incurred during the first deployments. Actually, the downside of syncE is its high cost. Up-to-date there are few syncE silicon alternatives (e.g., [Zarlink Semiconductor Inc. (2010)], [Maxim Integrated Products Inc. (2010)]), a fact that increases its cost dramatically compared to traditional solutions. Besides, there is yet no single formal document that standardizes it, but it is described by several ITU-T specifications.

The work in this Thesis deviates from syncE principles for two main reasons. First, syncE is an approach for frequency distribution over Ethernet nodes for replacing TDM dedicated links. Secondly, syncE redefines the Ethernet's physical layer (see Figure 3.1), a fact that will lead to increase the cost of native Ethernet cards. For a comprehensive tutorial of syncE the interested reader is referred to [Cisco Systems, Inc. (2010)].

Ethernet Passive Optical Networks

The first attempts to shift the legacy Ethernet to wider areas started in the Ethernet in the First Mile Alliance (EFMA) in 2001. EFMA[1] proposed the standardization of IEEE802.3ah, Ethernet in the First Mile (EFM), a collection

[1] Actually EFMA is integrated in Metro Ethernet Forum [Metro Ethernet Forum (2010)].

of standards that mainly describe a new standard for copper and fiber based PHY interfaces, as well as functions for management and monitoring of links on PHY layer. IEEE802.3ah is now integrated into IEEE802.3 [IEEE Std. 802.3 (2005)]. In a first mile environment, the Ethernet PHY interfaces require different characteristics (frequency spectrum, number of wires, cable length) to bear harsh environmental conditions like temperature range and humidity.

Together with new PHY interfaces, EFM also standardized the Ethernet Passive Optical Network (EPON) to cover the entire broadband access technologies under the same management architecture. EPON is a full duplex single fiber network with point-to-multipoint (PtMP) topology, in which passive optical splitters are used to enable a single optical fiber to serve multiple premises. An EPON architecture consists of an optical line terminal (OLT) and a number of optical network units (ONUs) near end users.

In an EPON, downstream communication is broadcast to each premise sharing a single fiber, while the upstream communication is a shared media that follows the multipoint control protocol (MPCP [IEEE Std. 802.3 (2005)]). MPCP is implemented in the MAC control layer and coordinates the medium access of the ONUs based on the time division multiple access (TDMA) principle. The control functionality of MPCP works continuously in two alternating phases, the *discovering* and the *reporting* phases (see Figure 3.2a). At start-up and periodically, MPCP starts discovering new users (ONUs) that want to join the network. Each ONU is in a different physical distance to the OLT and needs to anticipate or delay its time to access the network. In order to provide optimal network performance, the OLT measures the physical distance to each ONU using a *ranging mechanism*. As shown in Figure 3.2b, the OLT sends its local time (t_0) the ONU. When the ONU receives the message, it sets its local time according to the value in the timestamp field. When the OLT receives messages from ONUs, it uses the received timestamp value to calculate or verify a round trip time (RTT) in the timestamp field. The RTT is equal to the difference between the timer value (t_2) and the value in the timestamp field (t_1). The OLTs MAC control client uses this RTT for the ranging process, and to notify later the slot time to each ONU. The RTT is continuously updated in the OLT and in the ONUs to detect timestamp drift error, i.e., when the differences between OLT's and ONU's clocks exceeds some predefined threshold.

Timing is needed for MPCP as it is an slotted protocol. ONUs continuously report their current filling level of their transmission queues to the OLT, which signals each ONU the start time and duration of its next time slot. Either during the register process or during the bandwidth allocation, timestamps are mapped into a specific field of multipoint control protocols data units (MPCPDU) (see Figure 3.2c). MPCPDU are basic IEEE 802.3 that are always encoded as Control frames (*Type* field) and have another field (*Opcode*) to identify the specific MPCPDU being encapsulated. The timestamps are obtained from a 32-bit counter located in the MAC that increment every 16 *ns* (62.5 MHz). In [Symmetricom (2010)], there is a brief tutorial of PON architectures for the interested reader, and in [Kramer & Pesavento (2002)] there is a description about EPON's working details.

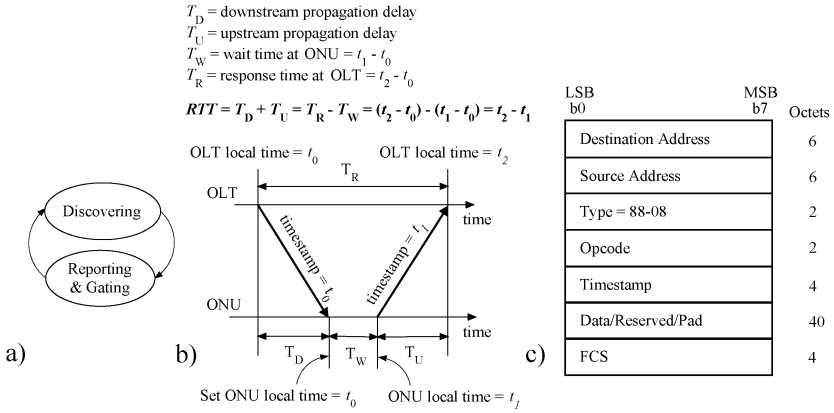

Figure 3.2: EPON operation (a)). EPON ranging mechanism for RTT calculation (b)). Multipoint control protocols data units (MPCPDU) (c))

In this Thesis we follow the principle of calculating the propagation delay of the Ethernet layers with the ranging mechanism of EPON (see Section 4.3.3).

General Positioning System

One of the most popular methods to synchronize with high accuracy is by using the dissemination time service of Global Positioning System (GPS) [U.S. Naval Observatory (1999)]. GPS system is an earth-orbiting-satellite based navigation system consisting of 24 satellites that provides users worldwide with twenty-four hour a day precise position in three dimensions and precise time traceable to global time standards. Each satellite, or space vehicle (SV), contains a pool of atomic clocks with which computes a *GPS-Time*, a continuous measurement of time from an epoch started at January 6, 1980 at midnight (0 hours 0 minutes 0 seconds) of Universal Time Coordinated (UTC). SVs communicate themselves and find a common GPS-Time to broadcast to the Earth at nominal intervals of 1.5 s in a message that also contains its spatial and temporal coordinates. GPS receivers on Earth calculate from (at least) four satellites its three-dimensional spatial position and the time with reference to the UTC. Through the Standard Positioning Service (SPS) offered by GPS, the information is intentionally degraded taking into account the delay of the GPS carrier signals as they pass through the ionosphere, which varies in density and thickness from 50 to 500 kilometers due to solar pressure. Once the GPS receiver processes the information and reduce the noise, it delivers two types of information that can be used as a reference to synchronize. The first is a set of subframes containing several parameters such as the SV clock correction parameters, the GPS week number, SV health, and so on. This information is used in many navigation systems to show the three-dimensional position and the UTC time. The second type of information, and more important for real-time applications, is the 1PPS signal which is an electrical signal steered by the GPS receiver as a means to provide

a frequency standard similar to atomic clocks. When the GPS receiver is stable (i.e., locked to, at least, four satellites), the 1PPS signal is equivalent to a reference signal of an atomic standard, and thus it can be used to control real-time applications. Although GPS receivers are relatively inexpensive when compared to conventional atomic clocks, their usability is far from that of an atomic standard. Atomic standards are plug'n'play equipments that do not depend on the satellites conditions, but they deliver a very high stable frequency signal from the moment they are powered on [Lombardi et al. (2007)].

Coupling to user computer equipment can be simplified by the use of GPS receivers that operate as plug-in cards that fit within personal computers [Hough (1991)]. With careful measurement of the latency time and repeatability required to read and transfer the GPS receiver time measurements on the PC bus, remote PC applications can be synchronized. This is precisely one of the main hurdles of computer-based architectures, i.e., the latency of the busses is usually too high and variable, and totally smudge the stability of the GPS signal. The latency and variability of the buses can only be eliminated with specialized equipment and high-end network cards, such as [Endace (2009)] (see Section 3.2.3).

In this Thesis, we design a hardware module that leverages the 1PPS signal delivered by a GPS receiver very accurately (see Section 6.2.1). This signal is used as a time reference to evaluate precisely the synchronization.

3.2.2 Pure Software Approaches

Since early Internet ages, time or time of day synchronization (ToD) has been traditionally addressed using software approaches. The IETF has standardized time services since the release of Internet Control Message Protocol (ICMP) [Postel (1981)] or the Time protocol [Postel & Harrenstien (1983)]. IETF has actually two opened work directions. The first one is the working group for coordinating the Network Time Protocol (NTPv4) [Mills (1992b)] and extensions. The second line concerns on next generation timing over native IP MPLS-enabled IP Packet Switched Networks (PSNs). The working group responsible for coordinating this task is called The Timing over IP Connections and Transfer Of Clock (TICTOC) [IETF-TICTOC (2010)]. Up to date, its task is to collect requirements for various applications (such as Cellular backhauling, Circuit Emulation Services (CES), Instrumentation and Measurement, and so on) using NTPv4 and IEEE1588v2. NTP has become the *de facto* standard for ToD dissemination over the internet. Next, a breadth description of the NTP protocol is presented.

The Network Time Protocol

NTP was originally designed by David L. Mills at the University of Delaware in the middle 80s and is actually maintained and enhanced by a group of researchers [NTP (2010)]. NTP usually render a synchronization accuracy of hundreds of milliseconds over the Internet, enough for more relaxed distributed applications [Liskov (1993)]. The operational details of the last NTP version (NTPv4) are specified in the reference design [Mills (2006b)].

A node running NTP service works by repeatedly making time observations to a number of remote clocks. It adjusts the value of the system clock periodically in order to bring its value closer to the correct time. The protocol specifies the format of packets sent over the network, and the manner in which a computer conforming to the protocol must respond to the received packets. The specification includes suggested clock synchronization algorithms for managing the value of the local clock and a framework for other users to use particular clock adjustment algorithms. [Troxel (1994)], [Ridoux & Veitch (2009)], [Marzullo & Owicki (1983)] are some examples of clock synchronization approaches that utilize the NTP flow. NTP also categorizes machines into a hierarchy of time references. Primary machines are synchronized directly to time sources outside of the network, such as radio [Microsistemes Timing & Synchronization Solutions (2010)] or atomic standards. Secondary machines synchronize to the primary servers and others in a specifically configured subnet. The subnet is configured dynamically by a spanning tree algorithm using hierarchical levels and minimum path delay.

The main hurdle that NTP has to go through is the delay and the time variability of the intermediate network elements. Using NTP, the accuracy and stability of clocks on the Internet is generally in the tens of millisecond range [Mills (1991)], [Mills (1995)]. However, to reach this level of accuracy, NTP must be tuned correctly and allowed to stabilize for one or two weeks. Another drawback that an NTP client faces, and software approaches in general, is the time undeterminism caused by the operating system in reading the local time. To optimize these accesses, the author of NTP has proposed several changes to the operating system's kernel that basically consist on assigning high priority execution to NTP services [Mills (1994)].

Other Clock Synchronization Approaches

As seen in Section 2.2.5, clock synchronization algorithms are computational processes that synchronize the physical clocks of distributed computer networks. Many algorithms have been designed for maintaining synchronization of physical clocks over the years and it is not feasible to review them all here. However, all the approaches have common basic features: **1)** a connectionless messaging protocol; **2)** a mechanism for exchanging clock information among clients and/or servers; **3)** algorithms for mitigating the effects of nondeterminism in message delivery and processing, and for updating local clock based on information received from a server. They differ in some aspects: **1)** whether the time information is obtained from a software routine or specialized hardware; **2)** whether the network is kept internally consistent or synchronized to an external stardard time source. There is extensive literature describing variations on these methods. Most of them describe statistical and heuristic methods for e.g., better outlier rejection, reduction of the effects of time variability, resiliency in case of remote crashes or falseticking, and so on. [Anceaume & Puaut (1997)], [Anceaume & Puaut (1998)], [Ramanathan et al. (1990)] and [Schneider (1987)] are some fairly comprehensive surveys that analyse and compare them.

3.2.3 Hybrid Hardware/Software Approaches

The x-factor of time synchronization hybrid approaches relative to the pure software solutions is their capability to isolate the jitter introduced by the operating system when dealing with synchronization client requests. A synchronization client is a pure software process that executes a specific message interchange and an algorithm to adjust the local time of the client node. The synchronization client retrieves the timestamp, packetizes it and send it to the destination nodes. The action of timestamping is crucial as it is subjected to a time variability that depends on the computational load of the processor's node. In strong load computational scenarios, the *time sample* suffers from a waiting time that lasts from the time of the sample obtention to the time the sample sending. This problem that has been tackled in different flavours by [Kopetz & Ochsenreiter (1987)], [Höller et al. (2002)] and [Micheel et al. (2001)]. The principle of these solutions consist on capturing the time sample when the message is being sent. To carry out this approach, it is necessary the use of dedicated hardware of recognize specific headers of a synchronization message to generate the timestamp. As seen in Chapter 2, the concept of generating a timestamp by means of hardware is called hardware timestamping. IEEE 1588, or the Precision Time Protocol (PTP), was initially designed with this perspective in mind. As the core work of this thesis in some ways builds on features of the Precision Time Protocol (PTP), some parts of it would be difficult to understand without a general understanding of the workings of PTP. A summary of PTP follows, with emphasis on those aspects relevant to the current discussion.

The Precision Time Protocol

The Precision Time Protocol (PTP), or IEEE 1588, as defined in the IEEE 1588-2002 [IEEE Std. 1588 (2002)] and 1588-2008 standards [IEEE Std. 1588 (2008)], is officially entitled the Standard for Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. It was originally created by John Eidson in Agilent Technologies, Inc. two decades ago with the purpose of synchronizing the physical clocks of networked measurement and control systems in small Ethernet LANs. Similar to NTP protocol, the time distribution method of PTP relies on carrying timing information inside UDP packets. The nodes of a PTP network infer correct reference time using metrics of remote time offset and round trip delay.

The PTP synchronization operation is performed in two phases (see Figure 3.3a). In the first phase the slave correct his time difference with the offset measurement and at the end of it the slave clock is synchronized in frequency with he Master Clock (MC). In this phase the master multicast *Sync* type messages at cyclically defined intervals (typically every 2 seconds). However, since there is an internode jitter the time information in the *Sync* message is not precise. The precise time is sent afterwards inside a *Follow_up* type message. Upon reception of the *Follow_up* message, the slave calculates the correction value, i.e. the offset, and adjusts its time accordingly, thus synchronizing also in frequency.

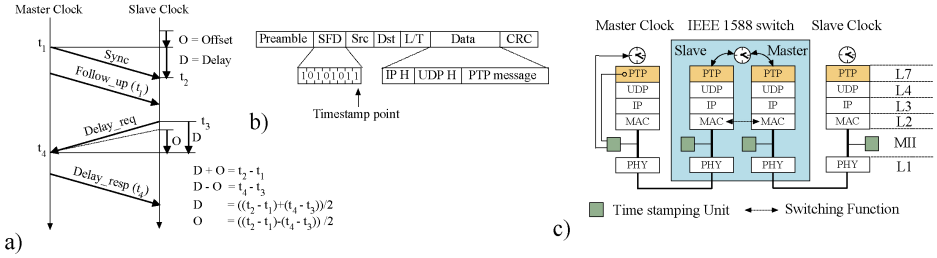

Figure 3.3: Synchronization procedure (a)). PTP header format (b)). PTP node architecture (c)).

The second phase of the synchronization process aims at calculating the network delay time between slave and master. The slave starts sending a *Delay_request* type message to the master and determines the exact transmission time of the message. The master creates a timestamp when it receives the packet and sends the reception time back to the slave in a *Delay_resp* message. At this point the slave can derive the network delay (D).

The PTP standard defines network messages, internal datasets and events, specifications for Ethernet-based implementations and implementation suggestions. It does not include any reference implementation for adjusting a clock, but it provides a set of rules for allowing devices from different manufacturers and different implementations to interoperate. Its submicrosecond accuracy target is consolidating it as a *the facto* standard for time/phase distribution especially over networked distributed measurement and control systems, industrial automation and test and measurement environments and LANs supporting multicast communications.

The key points that makes PTP rendering the sub-microsecond time/phase synchronization accuracy are:

⇒ *Internode jitter assistance.* In Ethernet architecture, PTP tackles this problem by allocating a specialized hardware unit closer to the Media Independent Interface (MII) (see Figure 3.3). PTP packets contain a protocol-specific header which makes the unit to generate timestamps for both the ingress and egress packets. The protocol suggests the use of hardware-assisted timestamping for best synchronization accuracies. However, an implementation compliant might also disregard this requirement at expenses of less accuracy, which in some cases equals NTP's performance.

⇒ *Network jitter assistance.* The network jitter arises from the time needed by the intermediate elements to store and forward the packets. In *store-and-forward* switches this time score penalty might be around hundreds of microseconds. PTP overcomes this problem leveraging specialized devices capable to compensate the residence time that a synchronization message would experience in the queue of a switch (see Figure 3.3c). In the context

of the PTP protocol these devices are called *Boundary clocks*. The clock is regenerated at the node in a multistage master slave relationship.

Despite addressing internode and network jitter, PTP has several limitations that can affect the synchronization accuracy:

⇒ *Number of synchronization messages.* The node acting as a master broadcasts periodically to slave clocks two messages (*Sync* and *Follow_up*, see Figure 3.3a) to communicate its local time. The use of two messages owes to the fact that the timestamping logic cannot provide in time the egress timestamp. In microprocessor-based architectures, the timestamps are obtained from a high-speed counter within the microprocessor. As the timestamp logic has to go across bus arbitration delays and jitter, the timestamps of egress packets cannot be inserted in the current packet, but they need a second packet to be transmitted. This limitation degrades the accuracy between nodes as the time conveyed in the first message is an estimation, and hence the node has to wait for a second packet (with the more precise timestamp). The time from the first to the second timestamp can be up to 2 seconds. As the clock oscillators in common nodes have generally a nominal frequency drift of ± 50 ppm, the waiting time might lead to offset errors of ~ 100 μs.

⇒ *Network delay assymetry.* The fact that PTP was originally designed for data acquisition in small LANs leads to assume the transmission delays are near constant and symmetric. Such assumption is not true in many other packet-based networks and determines the error committed on the offset and roundtrip calculations. E.g. the Internet is the extreme case where assymetries lead to poor synchronization accuracies. The wider the network is, the greater the impact of assymetries on the synchronization accuracies is. PTP is provisioned with a Spanning Tree based protocol that minimizes this problem by constructing a symmetric network topology so that the propagation times can be considered equally. In the context of PTP, the algorithm is known as Best Master Clock Algorithm (BMCA). Besides the topology construction, BMCA is committed to elect the node with the best clock quality in order to set it as the root of the PTP network, and the node by which other clocks synchronize to.

⇒ *Interactivity with software.* PTP is a synchronization protocol specifically designed to split functionalities on a hardware part and software part. The role of the hardware is to accept instructions from the software, and move bits around accordingly. The whole "intelligence" resides in the control software, which realizes more complex computational operations. The downside of this paradigm is on the response delay time in the software-hardware communication. This delay time becomes more critical when the targeted accuracy is at the level of *ns*. In many cases the software stack can take up to hundreds of μs to react.

The work of IEEE has recently been closed with the standardization of the second version of IEEE 1588, unofficially known as IEEE1588v2 [IEEE Std. 1588 (2008)]. This new release has filled a gap for being used in the area of telecommunications, more concretely in backhaul wireless synchronization. Many existing mobile wireless technologies have a strong requirement on time synchronization (see Section 3.2.4). Distinctive characteristics in this last release are:

⇒ *Layer 2 option.* The last revision of the PTP standard considers a new normative for transporting PTP over IEEE 802.3 Ethernet (Annex F of [IEEE Std. 1588 (2008)]), a feature that is more akin to the work in this Thesis. This new specification benefits from avoiding Layer 3 services to transport timing, and thus the use of long UDP packets (due to additional information overhead produced by IP or UDP protocol (see Figure 3.3b)). The use of shorter messages (64 bytes) offers the possibility to built PTP with scarced hardware and software resources, and thus an IEEE 1588 approach susceptible to be integrated in a single Ethernet chip. To the time of the edition of this document, we found one work implementing this new *IEEE 1588 Layer 2* specification [Kutschera et al. (2009)]. The authors target synchronization accuracies of tenths of nanoseconds between contiguous nodes. Their implementation is based on a proprietary core [Oregano Systems - Design & Consulting Ltd. (2009)] that contains a dedicated MAC with PTP-message filtering, a clock unit (inherited from [Höller et al. (2002)]) and a general purpose 8-bit mc8051 microcontroller that runs the PTP protocol stack.

⇒ *Transparent Clocks.* In the context of PTP, boundary clocks are specialized switches capable of signalling the internetwork residence time of the packets. On the other hand, transparent clocks represent one step further in the sense that they cancel the residence time in the switch. The use of Transparent clocks allows that the networks elements in the PTP network do not have to support a full master-slave configuration, and thus diminishing the complexity and the cost [Nylund & Holmeide (2005)].

⇒ *Message size.* The size of the synchronization messages is reduced in order to reduce the transmission delay of the packet (see Figure 3.3b).

There is a large bibliography of PTP in the Internet. For the readers interested about the PTP protocol working details, they are referred to the standard webpage [National Institute of Standards and Technology (NIST) (2010)] and to the author's book [Eidson (2006)]. The work in this Thesis leverages PTP's paradigm, i.e., hardware-assistance for precise timestamping of packets as the cornerstone for achieving high accuracy synchronization.

Standard for Audio Video Bridging Transport (802.1as)

Another active area of work in which IEEE 1588 is being integrated is for the audio and video bridging (AVB) transport in residential Ethernet environments. In the last years, the market of home multimedia networks has been growing. Consumers

want to access multimedia content and resources stored anywhere in the house using their computers and entertainment devices. This growing need, together with the potential of Ethernet, has lead to provide Ethernet the capability to distribute high quality digital audio and video reliably [Teener et al. (March 2005)]. This new application is in standardization process and is coordinated by the 802.1as Working Group (WG). The IEEE 802.1 WG is chartered to concern itself with and develop standards and recommended practices in several areas such as security, interworking, audio/video bridging, and so on. Regarding to the audio/video bridging applications, three PARs are responsible for developing stardards: 802.1as [802.1as WG (2010)], for timing and synchronization; IEEE 802.1Qat, for a stream reservation protocol; and 802.1Qav, for forwarding and queuing enhancements for time-sensitive streams.

Once standardized, 802.1as will be officially titled as the "Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks". It provides the protocol and the procedures for audio and video applications across bridged LANs (with fixed and symmetrical transmission delays) by re-adopting a big part of the synchronization functionality of the last IEEE 1588 release. This includes the maintenance of synchronized time in the context of IEEE Stds 802.1D and 802.1Q networks during normal operation, addition, removal, and failure or reconfiguration of network components. The proposed standard could be applied to all 802 networks composed of full duplex Ethernet links.

The definition of this specification arised because the existing time synchronization standards, PTP and NTP, operate at layer 3 and impose too much operational complexity and implementation costs on the development of audio-video equipment. Concretely, this specification defines how to synchronize a common time base across an entire AVB network in a residential Ethernet network, limiting networks with no more than seven bridges (see Figure 3.4). The common time base is in the form of a Real Time Clock (RTC), a large counter which consists of a 32-bit nanoseconds field and a 48-bit seconds field. For Fast-Ethernet networks the clock resolution is 40 ns, while in Gigabit Ethernet networks is 8 ns. A single device on the network is designated as the clock master by automatic resolution using the PTP's Best Master Clock Algorithm (BMCA) while all other devices resolve to be slaves. Using the 802.1as, all slave devices will regularly update their own RTC to match that of the network clock master. As seen in the Figure 3.4, AVB networks are fully compatible with IEEE 802.3 legacy traffic (*legacy traffic*) but priority is given to AV traffic (*AV traffic*).

Although the 802.1as is majorly based on IEEE 1588, their specifications have some differences. The first is on the network topology construction and the master clock selection within the AVB network. Each protocol elects a reference clock based on different datasets of the protocol headers. PTP gives priority to the quality of the clock (e.g., stratum number, clock identifier, and PTP variance (related to Allan variance)), while in AVB falls on the user choice according to a preference levels. More differences are on the frame formats and data types, and on the frequency and phase compensation algorithm. However, the major difference is on the message semantics. On one hand, AVB uses a two-way message

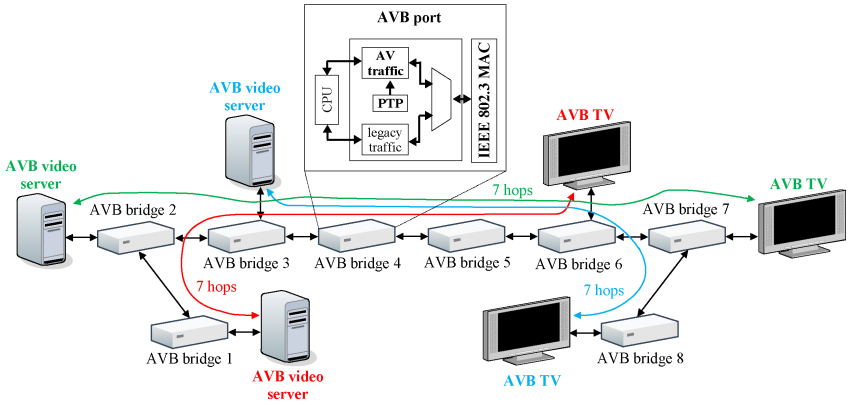

Figure 3.4: Audio-Video Bridged network.

scheme, while PTP uses more frequent one-way messaging. The last difference is that the timestamps of AVB messages always refer to the time of two messages ago, while the timestamps in PTP refer to the current time of the originator.

3.2.4 Specific Needs, Specific Applications

In the previous sections, we have reviewed most prominent State-of-the-Art to deliver synchronization. In absolute terms, a synchronization network deployment is not better than other, but its suitability depends on the synchronization needs of the end application. In the context of networks, the complexity and diversity of the scenarios involved leads to consider several parameters at the time of chosing a particular or a combination of technologies for delivering synchronization. Next, we summarize a series of fundamental requirements and tradeoffs.

Fields of Application

An important parameter to be considered for actual applications is the type of synchronization needed, i.e., either in frequency and time/phase. For example, packet switching was originally introduced to handle asynchronous data, but the ongoing evolution of packet network drives the new requirements in the form of frequency, phase and time distribution. In the case of packet networks, two main aspects summarize the need for synchronization: **1)** The support of new services and interworking with legacy TDM networks, and **2)** Support of the end-application operations, e.g., cellular mobile wireless base stations. In the first case, the equipment deployed at the edges of packet networks generally drives the synchronization requirements. Such requirements include: **1)** proper recovery of the long-term accuracy of the original timing reference, and **2)** controlling phase noise within the limits given by actual standards (e.g., ITU-T G.8261 [ITU-T G.8261 (2008)] and ITU-T G.823 [ITU-T G.823 (2000)]).

Typical today's end-application that require tight time/phase and frequency

synchronization are broadband mobile wireless applications, including last mile broadband, hotspots, cellular backhaul and full mobile high speed broadband access. According to [Infonetics Research (2009)], backhaul investments experienced a jump by 19% in 2008 and forecast a jump by 24% in 2009. The increasing demand for innovative data services and T1/E1 high costs to support the growth in traffic, lead telecom operators to consider other solutions for frequency and time/phase distribution to the wireless base stations. For the sake of the discussion, a high frequency stability is required by CDMA, GSM and TDMA specifications to avoid that centre frequencies of on-air channels drift, resulting in co-channel interference and in problems at hand-off. Moreover, time stability is required by CDMA specifications to keep pilot sequences in all cells within strict time alignment to assure uninterrupted handoffs when the user transits from one cell to another. New services such as 3GPP Long Term Evolution (LTE) will require even more stringent frequency and time stability. Table 3.1 is reproduced from [Cisco Systems, Inc. (2010)] and gives a summary of the timing requirements of the main wireless technologies.

Table 3.1: Timing requirements of the main wireless technologies.

Synchronization Type	Application	Targeted Quality
Frequency	TDM support (e.g., Circuit Emulation over Packet (CEoP) or Circuit Emulation Service (CES)	PRC traceability, i.e., reference signal from Stratum 1 [ITU-T G.810 (1996)]
	Long Term Evolution (LTE) base stations [3GPP2 (2010)]	better than $\pm 5\times10^{-8}$ (± 0.05 ppm). $\pm 10\times10^{-8}$ (± 0.1 ppm) for frequency division duplex (FDD).
	IEEE 802.16 (WiMAX)	*Unsynchronized Orthogonal Frequency:* Division Multiple Access (OFDMA): better than $\pm 2\times10^{-6}$ (± 2 ppm).
	Digital Video Broadcasting Terrestrial / Handheld (DVB-T/H)	down to a few parts per billion (ppb).
Time →phase (relative time) →ToD (absolute time)	3GPP2 Code Division Multiple Access (CDMA)	3 μs < time alignment error < 10 μs
	Universal Multiple Telecommunications Service (UMTS) Time Division Duplex (TDD)	Inter-cell synchronization accuracy better than $\pm 2.5\mu$s between base stations. $\pm 1.25\mu$s from common source.
	DVB-T/H single frequency network (SFN)	All transmitters within a single-frequency network must broadcast an identical signal to within 1 μs accuracy.
	802.16D/e TDD	better than 5 μs.
	IP service-level agreement (SLA), performance measurement, correlation of logs	Metro Ethernet Forum Service Operation And Management (SOAM) for one-way delay measurement [Metro Ethernet Forum (2010)], [Clouston et al. (1998)] and [ITU-T Y.1731 (2006)]. The short-term goal is to improve precision to <1 *ms*. Target is at few orders of magnitude below average delay, i.e., 10 to 100 μs. For correlation, the finer the timestamping, the faster the correlation.

Another area of application for Ethernet is in the industrial field. In contrast to the Ethernet of corporate LANs, *industrial Ethernet* has to meet requirements such as time synchronization and real-time operation. Nowadays, there are a myriad of different real-time Ethernet solutions supporting different QoS requirements. In [Decotignie (2005)], there is a classification of actual and current solutions according to their degree of similarity with legacy Ethernet.

Accuracy, Geographical Dispersion and Cost Budget

The synchronization tightness and geographical dispersion are two key variables for choosing a synchronization approach, whether being pure synchronization hardware mechanisms, hybrid or pure software solutions. Figure 3.5 classifies synchronization mechanisms depending on the synchronization accuracy, the geographical dispersion of the distributed system and cost. Pure hardware mechanisms are more feasible to be adopted in those applications that require short-distance communications and high accuracy. Conversely, pure software approaches such as NTP and PTP (without hardware assistance) fit better on applications requiring communication distances ranging from hundreds of meters to kilometric distances. The cost plays an important role in this classification. The closer the distances are, the cheaper are the solutions based on hardware mechanisms, e.g., on PLL. On the other hand, pure hardware solutions devoted to synchronize long distance applications become more expensive, as in the case of SDH networks or syncE.

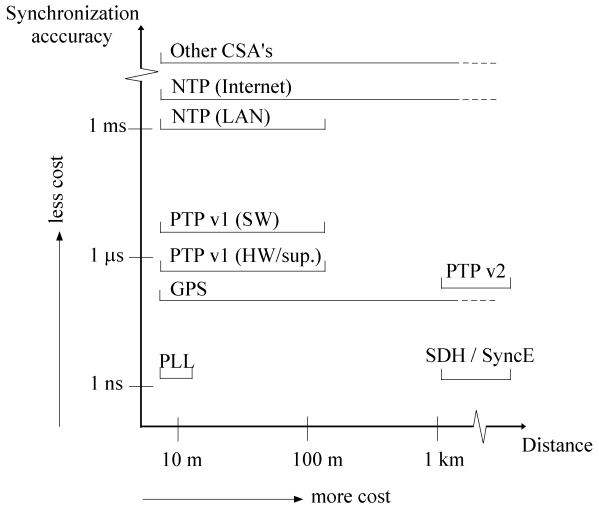

Figure 3.5: Synchronization classification according to synchronization the accuracy, geographical dispersion and cost.

Existing Infrastructure

Another key factor to consider when chosing a particular technology to deliver synchronization is the coexistence with other technologies. This is particularly noticeable in the field of telecommunications. The actual trend of telecommunication operators is to gradually replace circuit-based infrastructure to packet-based technologies, with more bandwidth capabilities and less cost-per-bit. The network transition from legacy TDM to packet technologies started many years ago and is gradually migrating to the access networks. Some parts of the network

will continue to use TDM elements for some more years until they reach their end-of-life. Today's networks are a hybrid mix of circuit and packet technologies, particularly in the access and metro. Therefore, timing and synchronization technologies will also evolve progressively in order to accommodate this heterogeneity. The telecommunication standardization bodies (ITU-T, IEEE and IETF) are carrying out a cooperative work so as to complement among technologies and synchronization requirements. In [Garner (2007)], there is an interesting summary of the actual timing techniques and synchronization aspects for packet networks (specified in [ITU-T G.8261 (2008)]).

3.3 Timestamping in Ethernet-based Networks

In Ethernet-based networks, timestamps are affected by the packet delay variabilities introduced by the intermediate elements of the network and the jitter of the digital logic mechanisms of the ingres/egress paths within the nodes. Unlike ATM, the timestamps used in Ethernet are consumed above the Layer 2 by applications. This section explains the problems of variability and delay of the timestamping processes in Ethernet technology, and actual works that help to measure and soothe them.

3.3.1 Error Sources

Timestamping mechanisms are exposed to three main problems: **1)** the *latency and jitter to read the clock*, **2)** the *clock drift* and **3)** *clock reading errors*. As shown in Figure 3.6a, the time snapshot from a time source (i.e., a counter) is not done instantaneously, at t_1, but it takes an additional delay (Δt) that sums up to the instantaneous value. Thus the time is obtained at t_2. This limitation becomes an important problem when the interval of a measurement requires high precision or the network interfaces speeds increase. For example, at 10 Gbps, a minimum-length frame may arrive every $\sim$61 *ns*, which means that in order to log the arrival of a minimum-length frame, the latency to read the counter should be well below this value. This requirement might be very challenging for most of the actual PC-based architectures. As they do not have hardware support, the latencies for reading the counter might be in the range of few microseconds.

Second, the clock drift is illustrated in Figure 3.6b. It shows the time evolution of two counters, an ideal counter, C_i, with no drift, and a real counter, C_r, with a variable drift. As explained before, the counters of PC-based architectures are paced by cheap quartz oscillators that inevitably drift from absolute time and from one another due to their intrinsic frequency error. Therefore, timestamps are also affected by clock drift component. In this example we wish to read C_r at true time t. Due to the latency in reading the counter (Δt), we will obtain a timestamp with a counter value given by $C_r(t\prime)$. Then, at true time $t\prime$ the timestamping error from the clock drift can be expressed as $\epsilon(t\prime) = C_r(t\prime) - t\prime$.

The third potential error is shown in Figure 3.6c. The counter C is a digital ensemble that sums up ticks at the speed of f_1 and period $T_1 = \frac{1}{f_1}$. R is the

register that stores C's instantaneous values upon a request on the control input *store*. In order to correctly transfer the value from C to R, the period of *store*'s control input, T_2, must be equal to T_1. In other words, the rising edges of *store*'s control input and f_1 must be synchronized, otherwise the transferred bits are not stable in the destination register (R). In digital logic design, this problem is known as *metastability* and it is a primary concern, and in some cases a difficult challenge to solve, when designing digital systems with several clock speeds.

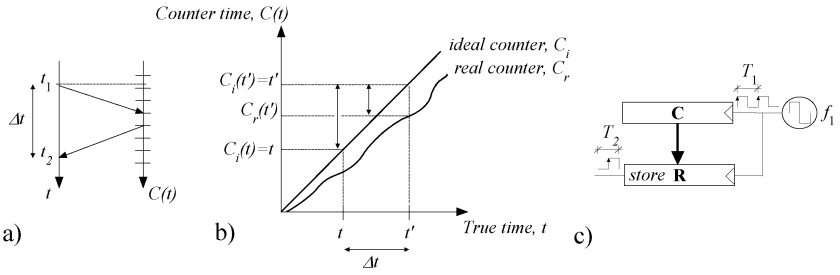

Figure 3.6: Delay in a timestamp reading (a)). Clock and timestamping errors (b)). Clock reading errors (c)).

Next, we review the methods and tools aiming at solve and minimize some of the three errors explained above.

3.3.2 Methods for Timestamp Accuracy Measurement and Error Prevention

Timestamping accuracy stands for the degree of conformity of a measured or calculated value to its definition, while precision, or resolution, refers to the degree by which the time is resolved. The timestamp accuracy is influenced primarily by two error sources: how often and how accurate a counter is updated and how fast is possible to access to this counter. There are several methods to evaluate the timestamp accuracy, most of them can be well classified whether they leverage any kind of specialized hardware or commodity hardware with software tools.

Hardware methods consist on a combination of specialized hardware and software tools for analyzing the accuracy of the timestamp mechanisms. Hardware benefits from the inherent stability to obtain precise timestamps which can be processed later. On the other hand, software methods make use of standard PCs with packet capture software tools and statistical methods to analyze the timestamps.

Software Methods

The burden to read the clock, as well as the resolution with which it can be read, started to be studied many years ago by Mills in [Mills (1992a)]. The main optimisations proposed by the author consisted on lowering the system noise by placing the timestamping operation in the (low-level) driver code of the network

interface. Following this approach and re-nicing the priorities of the OS, the timestamp requests take priority over other processes. In [Mills (1994)], there is a detailed implementation of this approach for NTP protocol with Linux OS.

In [Dietz et al. (1995)], the author investigates techniques for detecting instabilities (e.g., clock drift or uneven ticking) and estimating the timestamping accuracy without a precision time source. The author describes the sources of instabilities of Sun and DEC workstations at three levels: hardware, kernel, and network time service. To make use of unstable clocks in measurements, the author leverages the high stability of the oscillator over short interval periods of one hour, and use statistical methods to draw a "synthetic" clock.

[Paxson (1998)] discusses the problem of comparing pairs of timestamps between unsynchronized clocks of a wide-area network and how to calibrate them. If timestamp comparisons can be compared reliably, i.e., without coarse differences, one-way transit time measurements can be used to infer fundamental network properties such as delay, bottleneck, link speed, available bandwidth, and so on. The author proposes an algorithm for detecting when two clocks are not adjusted or have skewed for an interval of time, and the procedures to "clean" them.

[Pásztor & Veitch (2002)] introduced a new trend for synchronization based on the use of the Time Stamp Counter (TSC) register found in modern microprocessors, which gives a high resolution time source, for example 1 nanosecond for a 1 gigahertz processor. The authors propose a high accurate monitor solution for active network measurement. The key of these solutions is to consider the rate fluctuations of the TSC register over different measurement intervals. Leveraging the behavior of the TSC in different observation windows, the solution exploits a network of NTP servers and create a synchronized software clock with a stability of 0.1 ppm.

More recent works are found in [Wac et al. (2007)] and [Nguyen (2007)]. In the first one, it is evaluated the impact on timestamp accuracy of two software application tools developed with C# and Java. To obtain realistic end-to-end performance measurements, they emulate application-level streaming traffic through data with fixed inter packet times. The experiment was carried out in a distributed passive measurement infrastructure based on DAG 3.5E cards for reference link-level measurement. The authors found worse performance on Java application due to its interaction with the operating system (OS), and the system clock resolution under the Windows OS influencing application timestamps accuracy.

In the second work, it is used a test setup based on the packet capturing software Tcpdump [Tcpdump (2010)] and Smartbits packet generator [Spirent Communications (2010)] to measure the timestamping accuracy of motherboard's clock and several PCI-based Network Interface Cards (NIC) cards. The work offers a quantitative analysis between several parameters, such as the interrupt rate per second employed by the interrupt moderation mechanism, the time-stamping accuracy of the generator, CPU usage and packet loss rate. Among the many different configuration scenarios, the author found a worst case timestamping error of 1 *ms*.

Hardware Methods

One of the most prominent and recent works addressing the problems of timestamping was tackled by [Donnelly (2002)]. In his dissertation, he gives a detailed description design requirements for high-precision data packet capturing, covering related aspects such as timestamping, clock synchronization and node architecture analysis, for different technologies, like SONET and Ethernet. With regards to timestamping, it discusses different aspects such as resolution, format, reliability, accuracy and repeatibility. In his work, it also widely describes and analyzes the architecture and performance of a specialized NICs called DAG cards. DAGs leverage re-programmable hardware and embedded processors to provide globally synchronized time and high-precision and reliable timestamping of all the packet arrivals on high-speed network links with sub-hundred nanosecond resolution. DAGs were initially designed by the author's research group, in the University of Waikato [WAND - Network Research Group (2010)], to provide hardware-based measurements for ATM technology. Nowadays, DAG cards are founded by Endace [Endace (2009)] to support a wider variety of technologies (such as SONET and Ethernet) and speeds. DAGs are a well-known hardware tool in the network measurement community as they characterize for reporting interarrival packet times with no packet losses.

In [Arlos & Fiedler (2005)], the same author gives a comparative of the performance in terms of packet inter-arrival times and data loss of the popular Tcpdump [Tcpdump (2010)] and Windump (now WinPcap [WinPcap (2010)]) free software tools for packet capturing running on off-the-shelf PCs. They use a DAG card as a reference to timestamp the packets with a precision of 100 *ns* and to observe the packet losses of the software tools. Under lightly loaded links, the authors noticed that both software tools displayed packets with faulty timestamps and large tails in the inter-arrival times. Besides, they found the software was not able to indicate some packet losses. Based on their measurements, they conclude the both software tools cannot be used for time sensitive analysis or modelling.

In [Arlos & Fiedler (2007)], it is presented a method to estimate the timestamp accuracy obtained from measurement hardware and software. The method consists on using the link layer to generate equally sized PDUs, sent them back-to-back at the physical layer and observe the interarrival times using DAG cards. From the interarrival times, they found that the worst-case timestamp accuracy can be bounded within the limits of the maximum resolution of the measurement system.

[Nicolau et al. (2009)] proposes a method to timestamp with high precision and determinism non-standard IEEE802.3 control frames. The method consists on "duplicating" the TSC register of the μP and allocate it next to the digital logic that timestamps the packets. With this method the latencies and variabilities of the buses, arbitration and re-synchronization logic are bypassed. This technique, however, poses two problems. The first one is on the offset between both counters when a transfer of the counter's content is required. This offset can be variable if the bus protocol is not deterministic. The author measures statistically the transfer time by measuring repeteadly the execution times of the

low-level instructions and the two counters accounting for the average delays (see Section 6.3.5). The second problem is that the speeds with which the two counters sum up the ticks are different. This problem can be addressed by creating a symmetric clock signal tree in order to control the propagation delay through the digital logic cloud.

[Zhou et al. (2010)] proposes a method called HATS to evaluate the impact of the timestamp precision on network measurements based on the NetFPGA platform [NetFPGA (2009)]. The authors implement a timestamping hardware mechanism that timestamps packets with 48 bits and 8 *ns* resolution. The purpose of the authors is to evaluate using advanced statistical methods the mean of the timestamp dispersion at three *levels*: at the application level, using IPerf packet capture tool [IPerf (2010)], at the kernel level, using Tcpdump, and in hardware, using their own hardware block HATS. As far as the timestamping reliability is concerned, the authors overlook or do not mention how they snapshot the time to obtain timestamps free of capture errors.

3.4 Conclusions

The bodies which carry the most weight of the synchronization evolution are several Tier 1 mobile operators, standardization bodies and equipment manufacturers. There is a consensus among them that the preferred solution for frequency delivery using Ethernet, as specified in ITU-T Recommendations, is syncE [ITU-T G.8262 (2007)]. For delivering both phase and time, the preferred solution is IEEE 1588 [IEEE Std. 1588 (2002)]. While syncE approach is still in its early stages, IEEE 1588 has been consolidated as the standard par excellence in industrial environments and, since the release of the new version [IEEE Std. 1588 (2008)], in the telecommunications sector too. IEEE 1588 specifies a particular exchange of UDP messages between a node acting as a master and several nodes acting as a slave. The master inserts its localtime in a message, i.e., a timestamp, and sends the messages to the other nodes. The slaves infer the propagation delay of the messages from the series of timestamps, and use it to synchronize to the master with better accuracy.

The key of the IEEE 1588 excellence is because it specifies that the timestamps must be generated when the messages are being sent/received at the lowest layers. This way the time information is not influenced by the operating system delays (scheduling behavior) or the elements of the node architecture (busses, intermediate memories, etc). This requirement can only be meet with the help of specialized hardware. IEEE 1588 leverages this property to all the nodes of a network running the protocol, including specialized routers too.

The protocols that deliver synchronization over Ethernet use the timestamps as the fundamental units to synchronize accurately. The accuracy of the synchronization majorly depends on the reliability of the timestamps. The protocols based on Ethernet maintain the "asynchronicity", namely as opposite to synchronous networks, every node contains its own clock to send/receive the information and to generate the timestamps. Moreover, every node has multiple

and different clock signals to control the internal circuitry. The timestamping circuitry is also asynchronous to other parts within the node, a fact that can lead to obtain erroneous timestamps. We have found platforms based on specialized hardware and software tools that measure the accuracy of the timestamps and the latency to read a counter. Although the need for timestamping reliability is a well-known problem [Arlos & Fiedler (2005)], we have not found literature proposing specific solutions to avoid faulty timestamps.

In the field of synchronization, the techniques for delivering synchronization are specified in standards, which formally describe a series of recommendations and requirements for a solution to be standard compliant. Standards specify which requirements and properties are needed to establish the performance threshold. They keep out implementation decisions, and thus different implementations of a same standard might show different performance degrees. Therefore, in the area of synchronization, there is a wide range of technologies and synchronization techniques that should be seen as complementary for meeting different time and frequency needs in specific deployments [ITU-T G.8264 (2007)]. Therefore, the interoperability is a must.

This chapter is devoted to state the problems that this Thesis addresses and the followed conceptual approach to cope with them. It also presents the first contribution of the Thesis.

4.1 Delay Components in Timing Message Delivery

The main tasks of time synchronization in a distributed network are to provide a common timescale for all the network nodes and the right temporal coordination among all nodes engaged in a collaborative and distributed interaction with the physical environment. Timing mismatch arises from different initial setup times of nodes and time variations introduced by local oscillators running at different frequencies. These sources of error cause the local time of different nodes to drift apart over a time interval.

For the sake of argument, consider two dispersed clocks with different local time that need to be synchronized one to the other. One of the nodes sends its local time to the other node. In the hypothetical case of no delay in the message delivery, the receiving node would immediately know the difference of its time to the sender's time. Unfortunately, in real networks, various delays affect the message delivery, making time synchronization much more difficult than this.

Synchronization protocols aim at estimating the delay of one node to the other through a series of timing transmissions that follow a specific message exchange pattern. Due to the stringent temporal constraints, part of the protocol functions, such as the timestamping, has to be supported at hardware layers to reduce delay uncertainties. Since the first introduction in the early 80s by Kopetz [Kopetz & Ochsenreiter (1987)] and later by Schmid [Schmid et al. (1999)], hardware timestamping has become a requirement for achieving high-precision synchronization.

At hardware level, the latency and jitter introduced by logic devices is at few orders of magnitude less compared with those uncertainties introduced by software at upper layers. Existing time synchronization protocols accommodate the latency and jitter introduced by intermediate layers, thus the accuracy of synchronization is tied to those limits. For example, in the case of Ethernet, the latency and jitter introduced by the digital logic of the MAC and the PHY layers would account for pure Layer 2 time synchronization mechanisms.

To understand the Layer 2 model perspective addressed in this Thesis, consider the message delay components in Figure 4.1:

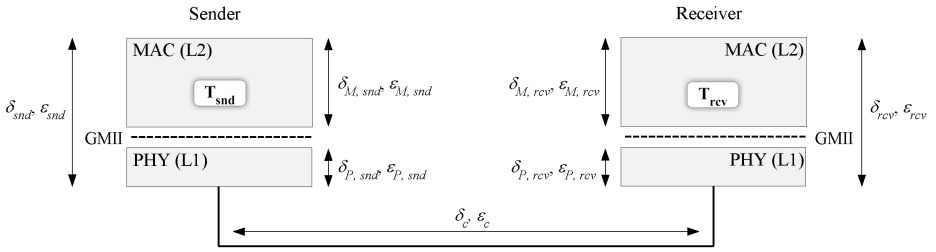

Figure 4.1: Delay and delay uncertainty components of a time message delivery.

⇒ The *time-to-send*, or the time for the MAC to build the message, $\delta_{M,snd}$, and send it to the PHY layer, $\delta_{P,snd}$. The time-to-send is non-deterministic and can vary within the interval of few nanoseconds, depending on the variability of the digital logic within the MAC, $\varepsilon_{M,snd}$. The complex analog and digital logic mechanisms inside the PHY also sum up variability to this message delay component, $\varepsilon_{P,snd}$.

⇒ The *propagation time*, or the time to transmit a message from the sender to the receiver through the cable, δ_p. In the case of Ethernet technologies using twisted pair cabling, the propagation delay is a function of the dielectric materials of the cable and its length. The propagation time variability, ε_p, is almost negligible when compared to inter-layer components. For a Category 5 (CAT-5) Ethernet cable, the propagation delay is $\sim 5\ ns/m$, while its variability is $\sim 0.2\ ns/m$. For a fiber, the propagation delay is $\sim 3\ ns/m$, while its variability can be at subpicosecond level.

⇒ The *time-to-receive*, or the time needed by the PHY layer of the receiver to process a receiving packet and transmit it to the MAC layer, $\delta_{P,rcv}$. And second, the time needed by the MAC to annotate the time of arrival of the message, $\delta_{M,rcv}$. As in the transmission case, in the reception, the PHY circuitry, $\varepsilon_{P,rcv}$, and the reception logic of the MAC, $\varepsilon_{M,rcv}$ increment the jitter budget.

The whole latency of a message timing delivery for a Layer 2 architecture is given by the sum of the latencies added by each layer during the transmission and the reception, i.e.,

$$\delta_{tot} = \sum \delta_{snd} + \delta_{rcv} = \delta_{M,snd} + \delta_{P,snd} + \delta_{P,rcv} + \delta_{M,snd} \tag{4.1}$$

The uncertainty of a message delivery in a Layer 2 architecture is given by the sum of the individual uncertainties added by each layer during the transmission and the reception, i.e.,

$$\varepsilon_{tot} = \sum \varepsilon_{snd} + \varepsilon_{rcv} = \varepsilon_{M,snd} + \varepsilon_{P,snd} + \varepsilon_{P,rcv} + \varepsilon_{M,snd} \tag{4.2}$$

Consider again the Figure 4.1. In order to synchronize the receiver node to the sender node, i.e., $T_{snd} = T_{rcv}$, it is needed to know and cancel the delay (δ_{tot}) introduced by the MAC and the PHY layers of the sender (SND), the receiver (RCV) and the cable. Theoretically, the synchronization accuracy achievable should be within the limits of the variability of the layers (ε_{tot}).

4.2 Goals and Approach

According to the problem stated in the previous section, the main goal of this Thesis is to validate in a real point-to-point configuration that the level of synchronization achievable at MAC level is limited by the sum of variabilities introduced by each Ethernet layer of a timing message delivery, i.e.,

$$-\varepsilon_{tot} < T_{snd} - T_{rcv} < +\varepsilon_{tot} \tag{4.3}$$

To prove the Equation 4.3, we need a synchronization mechanism capable of obtaining the latency introduced by the intermediate layers of the sender and the receiver (δ_{tot}) and communicating it to the partner link.

We will validate the Equation 4.3 in a real platform using 1000 Base-T Ethernet technology. Regarding to the synchronization mechanism, we use the time exchange pattern already defined in Ethernet standard [IEEE Std. 802.3 (2005)] for EPON topologies. EPON synchronization scheme is based on a ranging mechanism principle which performs a Round Trip Delay (RTD) calculation to infer the propagation time of a link ($2 \times \delta_{tot}$) and communicates it to the receiving node.

Ethernet is a technology specified to be implemented in hardware. Because of that, in this work we also rely on hardware tools to verify Equation 4.3. We choose standard FPGAs provided with Gigabit Ethernet connectivity to implement our design. The inherent complexity of FPGAs poses additional challenges when the time comes to translate a conceptual statement to the real platform. On top of that, the fact of addressing accuracies at the level of few nanoseconds strongly difficults the evaluation process and the veracity of the results. Thus, this Thesis also emphasizes the challenges incurred during the implementation phase and derives methods to evaluate the synchronization accuracy at nanosecond level in FPGA prototyping.

4.3 Layer 2 Network Model

Metro Ethernet Forum [Metro Ethernet Forum (2010)] stated that nowadays more than 90% of all data traffic starts and finish its journey from and in an Ethernet network. Although most of the internal networks of the Service Providers/Enterprises can connect disparate LANs through Ethernet, such an approach has several drawbacks, like non-guaranteed QoS, slow failure-recovery time, limited Virtual LAN (VLAN) tag space and single Spanning Tree based uneven load distribution with possible bottlenecks [Myers et al. (2004)]. IEEE task groups and a big number of researchers are working to overcome these drawbacks in best possible ways [Chiruvolu et al. (2004)], and many of them have provided realistic solutions [Kim et al. (2008)], [Kim & Rexford (2007)]. Since many years there is an interest to provide Ethernet with more capabilities in an intend to port it to wider geographical areas. One example is on the operation, administration and maintenance (OAM). The requirements for OAM functions focus on monitoring parameters such as connectivity, delay, delay variation (jitter) and status monitoring. Accurate synchronization is essential to provide those OAM parameters. One approach to provide synchronization might be with the use of a GPS at each Ethernet node. However, this solution is impractical and expensive. Another problem in wide area networks is that the roundtrip measurement is not accurate because the reverse direction might not take the same path.

As far as the local environments is concerned, pure Layer 2 implementations of the prominent PTP protocol have been requested in several areas, especially in the industrial field. A pure Layer 2 model would enable easier silicon-based solutions and more efficient switch technology. Time synchronization at Layer 2 has only to accommodate the jitter introduced by Ethernet layers, which is much less compared to other approaches based on upper layers, e.g., IP-based solutions. Despite of the advantages of a pure Layer 2 approach to deliver synchronization, there is a clear direction towards IP-networking. IP networks are easier to manage and do not suffer from well-known scalability problems of pure Layer 2 networks. For a Layer 2 paradigm to be fully reliable, these other functions must be addressed too.

This Thesis explores a Layer 2 network model to provide synchronization in an attempt to evaluate the synchronization accuracy achievable in Ethernet with independence of all these other limitations explained.

4.3.1 The Control Plane

The control plane can be understood as a set of architectures and protocols that enable a network to operate in a dynamic, self-organizing mode. The control plane concept pursues to provide controllability, observability and measurability. From the perspective of an infrastructure operator or provider, the separation of control plane functions from forwarding functions is not a new idea, but it is the traditional perspective of traditional networks such as SDH or ATM [Sexton & Reid (1997)]. A network operator needs to have a complete view of the built configuration of its infrastucture to take decisions about resource usage.

A control plane separated from the forwarding path is easier to centralise, both conceptually and physically. In the context of distributed systems, centralizing functionalities offers several advantatges such as resource assignment and reconfiguration, infrastructure cost minimisation, protection against attacks and, most important, better and new service offerings [Reina (2010)].

The control plane paradigm in the legacy Ethernet started in the amendment 802.3z [IEEE Std. 802.3 (1998)] with the introduction of a Flow-control functionality. The Flow-control was specifically designed to prevent switches (or end stations) from discarding incoming frames due to buffer overflow in short-term transient overload conditions. When a receiving node detects an overflow is coming it sends a PAUSE message to the transmitter to request a stop in the transmission. The PAUSE function is defined in a separate interface in standard Ethernet implementations to differentiate the data path from the control path.

The work in EFM under the standard IEEE 802.3ah further developed the concept of the control plane with EPON networks [Kramer & Pesavento (2002)]. Trying to meet the requirements of residential and business access networks, this standard focuses on different PHY specifications to the actual 802.3 specifications with minimal modifications in the MAC.

As seen in Figure 4.2, we want to follow the control plane concept and allocate a time synchronization functionality in the legacy Ethernet for the purpose of bounding the jitter introduced by Ethernet layers. The time synchronization functionality (*time sync*) and time synchronization client (*time sync client*) are both allocated in the MAC control and the MAC control sublayer, respectively. The first block performs hardware operations, such as timestamping, while the second block controls the protocol operation. With this approach, we follow the Ethernet philosophy which is to keep its asynchronous nature by not modifying the Layer 1.

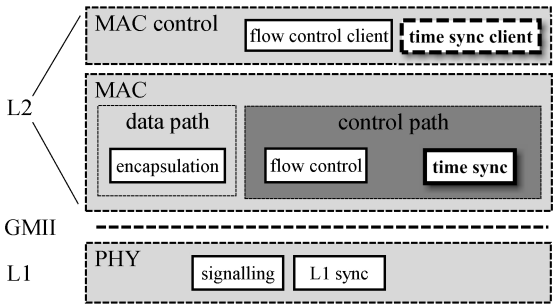

Figure 4.2: Proposed time synchronization extension in the Ethernet architecture.

4.3.2 Point to Point Layer 2 Architecture

In this Thesis, we use a point-to-point (PtP) topology to verify that the realistic synchronization accuracy between two nodes can be bounded within the jitter limits introduced by Ethernet layers in the path from a sender to a receiver. A

PtP topology is the optimal scenario as it is not influenced by other intermediate underterminism sources, such as the bridges in a bridged Ethernet scenario. The jitter measurement obtained in a PtP topology could be extrapolated to a multihop scenario with n nodes if the residence time of the frames within the bridges could be perfectly subtracted. Although not addressed in this Thesis, we provide means to compensate the residence time in a supposed bridged scenario for future extensions (see Section 4.3.4).

4.3.3 Synchronization Mechanism

Synchronization mechanisms in packet networks have one major objective, which is to set the same time in different nodes. To that end, protocols define different strategies for exchanging messages between nodes that contain the local time of each node. From the remote timestamps, they infer the journey delay of the packets following different computations. When a receiving node is aware of the delay of an incoming packet, it can use it to compensate the journey time of the packet and synchronize more accurately. As explained in Section 4.1, theoretically, if the sum of the latencies can be optimally cancelled, the synchronization error is confined within the jitter introduced by the intermediate elements in the message path.

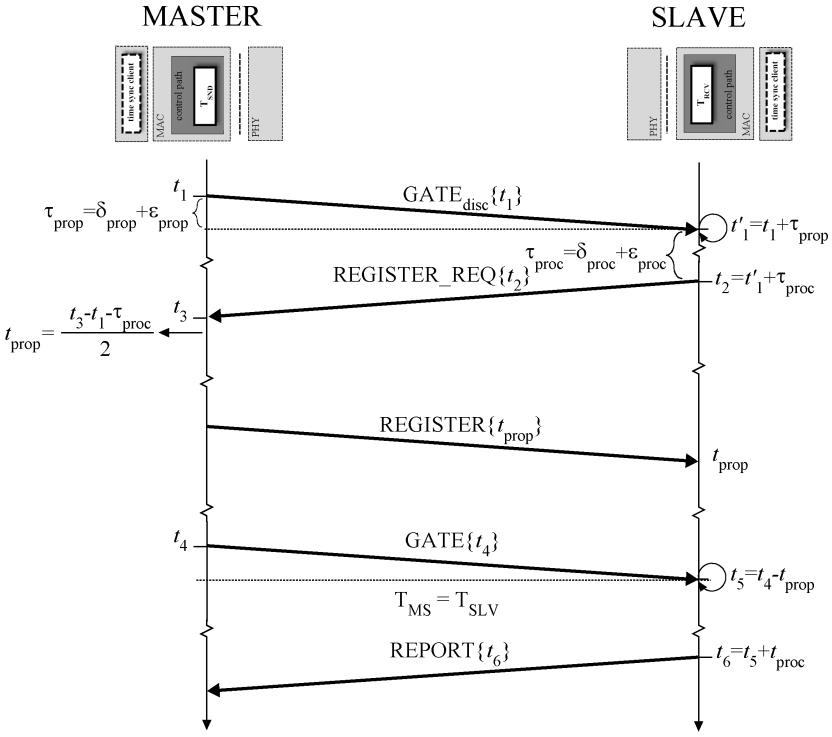

Figure 4.3: Synchronization message exchange pattern.

Figure 4.3 shows the EPON ranging mechanism for propagation delay calculation [IEEE Std. 802.3 (2005)]. This message exchange pattern characterizes for its simplicity and suitability to be implemented in hardware, and in point-to-point configurations. In this scheme, the node on the left is acting as a master and the one on the right as a slave. The master node performs two basic functions: *1)* coordinates the sending of synchronization messages, and *2)* calculates and inform to its link partner about the propagation time of synchronization messages.

The master node starts sending a message with its local time (t_1). The slave uses the timestamp within the message to replace its own local time abruptly. At this point, the offset between the master and the slave is given by the propagation time of the message to go across the stack of the master and the sender, i.e., τ_{prop}. Note that τ_{prop} corresponds to the fixed and variable propagation time of Equations 4.1 and 4.2, respectively. After a short interval of time given by $\tau_{proc} = \delta_{proc} + \varepsilon_{proc}$[1], the slave replies with a message containing its local time (t_2). When the master receives the reply from the slave, it notes its time of arrival (t_3) and starts the calculation of the one-way propagation time following the Equation 4.4.

$$t_{prop} = \frac{t_3 - t_1 + \tau_{proc}}{2} \tag{4.4}$$

The use of Equation 4.4 is valid under two considerations. The first is that the processing time must be far smaller than the propagation time through the stack, i.e., $\tau_{proc} << \frac{t_3 - t_1}{2}$. In those cases where this condition is not meet, the calculated propagation times will not be precise enough, and thus the synchronization accuracy will be impaired. The second consideration relates to the symmetry of the transmission and reception paths of Ethernet layers. The reception logic within the PHY layer usually takes more time to convert the bitstream into bytes than the transmission logic [Müller et al. (2004)]. This situation is more dramatic in those configurations of different Ethernet vendors and implementations. However, in the cases with the same Ethernet vendor, the link can be considered symmetric.

Once the master has calculated the delay compensation, it sends it to the slave node in another message (REGISTER$\{t_{prop}\}$). The slave will use it to cancel the propagation time in the next message (GATE$\{t_4\}$) to synchronize accurately to the master node (T_{MS}=T_{SLV}).

4.3.4 Protocol Data Units

Protocol Data Unit (PDU) is the term used to describe data as it moves from one layer of the OSI model to another. There are also two kinds of PDUs: data and control PDUs. While the former contain data for the upper immediate layer, control PDUs are in charge of the whole protocol behavior of functions such as

[1] τ_{proc} corresponds to the latency and jitter for the slave to process the incoming message and generate a reply message.

establishment or service break, flow control, error control, and so on. They do not contain information from upper layers, thus they are fully processed at the layer for which the function is managed. As explained before, Ethernet defines a control flow functionality within the Layer 2 that employs specific control messages. The case of the multipoint control protocol (MPCP) specification of the actual Ethernet standard [IEEE Std. 802.3 (2005)] defines a Multipoint MAC Control sublayer as an extension of the MAC Control sublayer of the original Ethernet. The standard specifies a generalized architecture and protocol functions for MAC Control. It is specified such that it can support new functions to be implemented and added to the standard.

The idea in this Thesis is to use the MAC Control protocol to carry out synchronization functions. In turn, synchronization functions within the MAC are triggered depending on the value specified in a concrete field of the Control PDUs. Recall from Figure 4.3; This message exchange pattern is extracted from the MPCP specification for point-to-multipoint (PtMP) topologies. Each message performs one specific function that is processed within the MAC Control sublayer and it is not passed to the upper levels. In Figure 4.4, it is shown the generic structure and encoding of MPCP PDUs that we will use in this Thesis. From the control frames defined in native Ethernet, they differ from the *Timestamp* and *Compensation* fields.

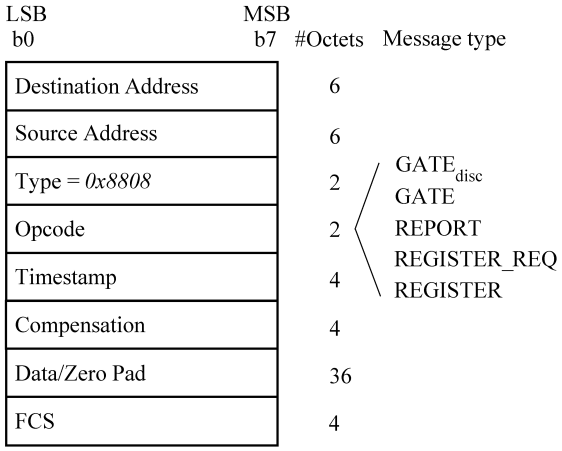

Figure 4.4: Ethernet MAC control frame transporting synchronization information.

They are 64 byte fixed-length frames organized in eight fields with different functionalities:

- **Adresses identification:** two 6-byte fields for identifying the destination (DA) and source (SA) addresses.
- **Control identification:** one 2-byte *Type* field containing the fixed value of *0x8808* for distinguish them among data frames.

- **Control function identification:** one 2-byte *opcode* field for defining and adding new control functions. The MPCP specification defines several control opcodes for carrying out different P2MP functions. We, instead, will re-use a set of opcodes for our evaluation purposes.
  - **GATE$_{disc}$.** Requests that the recipient must synchronize its localtime with the timestamp within the syncPDU, and acknowledge the update.
  - **REGISTER_REQ.** Notifies the recipient that the remote node is synchronized.
  - **REGISTER.** Notifies the recipient the propagation time of a syncPDU.
  - **GATE.** Notifies the recipient to report the localtime.
  - **REPORT.** Notifies the recipient the localtime of the remote node.
- **Local time transportation:** one 4-byte *Timestamp* field conveying the content of the local time at the time of transmission of the MPCPDUs.
- **Compensation time transportation:** one 4-byte field conveying the accumulated residence time of the syncPDU after crossing n bridges.
- **Data/Reserved/Pad:** a 36-byte long field used for the payload of the MPCPDUs. When not used they are filled with zeros on transmission, and ignored on reception.
- **Frame check sequence:** one 4-byte field for frame check sequence (FCS) generated by the underlying MAC after the Control sublayer has sent the Control PDU.

In the next section, we describe the operation mode of the synchronization mechanism as well as the main variables that govern its behavior.

4.3.5 Operation of the prototype

The synchronization mechanism operation principle has been inherited from the standard [IEEE Std. 802.3 (2005)] and adapted for prototyping. As shown in Figure 4.5, it consists of two recursive phases, the *discovery* and *normal operation* phases. In the former, the propagation delay calculation is initiated by the master with the use of a GATE$_{disc}$ message and notified by the slave with a REGISTER_REQ message. In the same *discovery* phase, the master calculates the propagation time and communicates it to the slave in a REGISTER message. Just after the notification of the propagation time, the master enters into the *normal operation* mode to perform two tasks. The first task is to periodically ask the localtime of the slave using GATE messages. The slave recognizes the normal operation mode when receiving a GATE message after the discovery phase. It has an internal state variable (*op_mode*) that uses to set the operation mode. The periodicity of the GATE message transmission is given by τ_{GATE} parameter. The second task of the master within the normal operation phase is to re-synchronize slave's localtime using GATE$_{disc}$ messages at a periodic interval of τ_{sync}. As it

will be seen in the next chapters, the periodicity of the re-synchronizations will determine the amount of accumulated offset between the master and the slave. In practice, the choice of τ_{sync} is given by the synchronization requirements of the application, e.g., the amount of permitted clock offset between two clocks. Finally, the master completes an operation cycle when transitions back to the *discovery* phase after an interval of time of τ_{disc}.

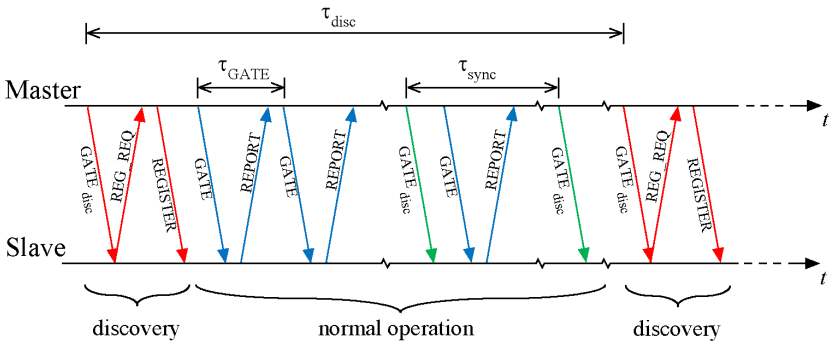

Figure 4.5: Timeline of synchronization messages exchange pattern.

4.4 Conclusions

Leaving aside the problems of Ethernet scalability [Myers et al. (2004)], an "all Ethernet" network architecture could be very efficient to deliver synchronization. Actual protocols that run at higher levels have to accommodate the jitter introduced by the lower layers. However, a pure Layer 2 solution only has to accommodate the jitter of its two layers. Ethernet is defined to be implemented in hardware, and hardware characterizes for its low level of jitter. Theoretically, the synchronization accuracy achievable should be bounded within the limits of the jitter introduced by the Layers 2 and 1. In a point-to-point scenario, the synchronization accuracy should be given by the jitter of the Ethernet layers of the sender and the receiver. This is precisely what this Thesis addresses, to verify in a real platform that the synchronization accuracy in an "all Ethernet" network could be very optimal.

To verify our goal we have introduced a synchronization prototype mechanism that calculates the fixed delay components, i.e. the latency, from the sender to the receiver. If the receiving node can perfectly know the latency of an incoming synchronization message, it will be able to synchronize optimally to the sender. Therefore, once the link partners are synchronized, the jitter can be neatly measured.

There is already a definition in the Ethernet standard to provide synchronization in point-to-multipoint topologies [IEEE Std. 802.3 (1998)], such as Ethernet Passive Optical Network (EPON). EPON synchronization mechanism characterizes for its message exchange simplicity and for its suitability to be implemented in hardware. Besides, EPON functionalities are entirely defined in the control

layer, which in the case of synchronization offers a better service as it is separated from the forwarding path. In an attemp to maintain the initial architecture of Ethernet, we have embedded the EPON concept in an evaluation platform.

§ 5. Time Synchronization Implementation for Gigabit Ethernet

This chapter presents the core part of this Thesis which is the implementation of the design presented in Chapter 4 using a standard FPGA provided with Ethernet connectivity. Commercial platforms have different features that oftentimes pose additional challenges when comes to implement a conceptual design. This chapter explains the architectural changes to overcome all these limitations, and describes in detail the hardware implementation of the key components of the synchronization platform.

5.1 Objectives and Requirements

The major objective in the development of this platform is to provide a low-cost, programmable and flexible framework for testing Ethernet upgrades. By definition, Layer 2 is largely defined at hardware level, a fact that lead us to consider hardware tools for function definition. FPGA-based embedded platforms are the best tools for this purpose as they contain multiple hardcored Intellectual Property (IP) blocks for specific hardware tasks in many different areas, such as communications, image processing, etc. Besides the IP blocks, they include a programmable region where specific and custom hardware tasks can be implemented. To implement our conceptual model presented in Chapter 4, we develop a hardware block capable of carrying out hardware timestamping of synchronization messages and internal platform events.

Our design characterizes for the following high-level design principles:

⇒ *Simplicity.* When designing with hardware, less is more. The less complexity in terms of used programmbable resources a design is, the better the overall functionality will be. Therefore, a primary requirement in the design of our block is simplicity.

⇒ *Upgradeability.* Hardware design characterizes to be unflexible when it comes to add new functions. Another requirement for our block is to design it in such a way that improved and new functionalities can be easily integrated with minor impact on the development time.

⇒ *Re-usability.* Implementations differ from the conceptual design due to the disparity of technologies. This mismatch is especially noticeable in designs with platform FPGAs. Hardware is described using HDL languages, and a

unique characteristic of FPGAs compared to other technologies, e.g. ASICs, is the flexibility of the code. Re-usability in the HDL code helps to port the hardware description to other platform FPGAs, and therefore, to validate hardware designs on different platforms.

⇒ *Layer 2 perspective.* Ethernet is defined as a two-layer technology with synchronization functions in the lowest layer. This Thesis envisions a legacy Ethernet with a time synchronization function in the MAC. Therefore, another requirement is to maintain and do not modify the synchronization mechanisms of the PHY.

5.2 A Low-Cost Platform FPGA

The following subsections present the chosen platform FPGA to implement the design. We identify some architectural limitations that hamper the similarity between the conceptual design and the implementation. We will also cover with detail the core work of this Thesis, the Timestamping Unit (TSU), which is a specialized hardware core that will help us verify the goal of this Thesis.

5.2.1 Platform Overview

To implement our design, we have chosen the ML403 platform FPGA from Xilinx [Xilinx, Inc. (2006)]. It is a standard platform that targets different markets and applications such as industrial, telecommunications, medical, digital video and embedded computing. A simplified block diagram is shown in Figure 5.1.

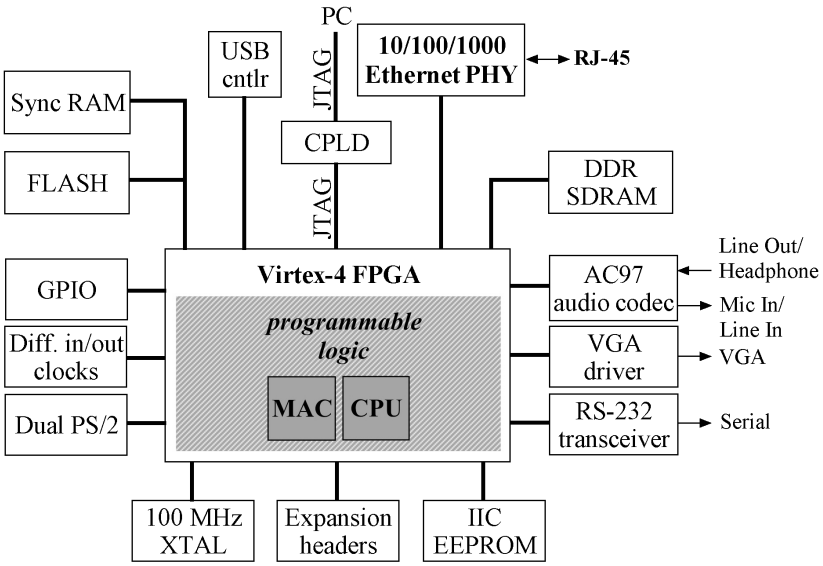

Figure 5.1: Block diagram of the ML403 evaluation board.

The overall platform is centered around the FPGA chip, a Virtex^{TM}-4 FPGA (XC4VFX12FF668). The FPGA chip embeds a PowerPC 405 (PPC) CPU, an Ethernet media access controller (MAC) and programmable logic to implement custom hardware functions. The programmable logic also serves to create the internal bus logic and *wrapper* logic to access to the peripherals on-board. The board has three different types of components: clocks, memories and connectors and interfaces. The overall FPGA chip is paced by a 100 MHz crystal oscillator and some other on-board drivers. The platform has different types of memories, ranging from DDR SDRAM, Flash or Synchronous RAM. As per the Ethernet connectivity, it mounts a Marvell Alaska 10/100/1000 Mbps (tri-speed) Ethernet transceiver (PHY) that connects directly to the MAC within the FPGA. The PHY is individually paced by a 25 MHz crystal oscillator to generate (with internal frequency synthesizers) the frequencies for the transmission and reception.

Platform FPGAs offer the possibility to interconnect the resources and have a complete system. However, to ensure the good functionality of the overall design, to accelerate the design process and save programmable resources, it is recommendable to choose the components (e.g., memories and connectors and interfaces) needed for the targeted application. In the next sections, we present the synchronization platform adapted to our evaluation purposes. A review of the internals of the FPGA follows.

5.2.2 FPGA Overview

The primary reason for choosing a platform FPGA to implement a design is the FPGA model. The whole Virtex-4 FX family is featured for embedded platform applications including hard-IP core blocks, such as the PowerPC^{TM} processor, tri-mode Ethernet MACs, 622 Mb/s to 6.5 Gb/s serial transceivers, dedicated DSP slices, high-speed clock management circuitry (frequency dividers/synthesizers), and source-synchronous interface blocks.

Knowledge of the overall amount of programmable resources is somehow needed for some specific functions. For example, the designer should know how many frequency synthesizers needs for a specific design; however, it may not be necessary to know how many slices the chip has for the design, as the number of used slices are controlled by the FPGA software development tools. While in the early stages of the design some design details can be abstracted, in the physical design phase they become crucial, especially if the design is hard constrained. The development tools usually report errors related to internal timing and resource limitations, thus the knowledge of the location of the resources and IP blocks within the FPGA layout is needed to debug error reports at some stage. Table 5.1 summarizes the number of hard-IP blocks and programmable logic resources of the Virtex-4 (XC4VFX12).

5.2.3 Limitations and Challenges

The price of generality comes at expenses of some architectural limitations. Generic platforms aiming at satisfying different needs, applications and require-

Table 5.1: Summary of the hard-IP blocks and programmable resources of the Virtex-4 (XC4VFX12-FF668).

	Configurable Logic Blocks (CLBs)				DSP	Block RAM						
Device	Array Row×Col	Logic Cells	Slices	Dist. RAM(Kb)	Slices	18 Kb Blocks	Block RAM(Kb)	DCMs	PowerPC Processors	Ethernet MACs	I/O Banks	User I/O
XC4VFX12	64×24	12312	5472	86	32	36	648	4	1	2	9	320

ments often cannot meet constrained requirements, in terms of speed, connectivity and chip size for implementing the hardware design. At the time to implement our conceptual design, we encountered the following hurdles:

- **MAC inaccessibility**. As seen before, today's FPGAs characterize for embedding third party intellectual property blocks that perform multiple different hardware tasks. On the one hand, these blocks are fully tested, compliant to international standards and easy to integrate within the system. On the other hand, access to their source code for reverse engineering is restricted. Moreover, their use for testing requires the payment of a time-limited license. We have not been exempted from these limitations in our platform either and, as per the Ethernet case, we do not have access to the MAC controller within the FPGA chip.

- **Maximum speed**. A primary requirement for our design is to provide timestamps with the maximum resolution possible. To provide clocking to the FPGA chip and part of the on-board transceivers, the platform use a 100 MHz XTAL oscillator. The FPGA has several frequency synthesizers (DCMs) that multiply/divide the frequency of the crystal oscillator. The maximum frequency that the DCMs can generate is limited to 300 MHz, and thus the maximum timestamping granularity achievable for this platform is $3.\widehat{33}$ *ns*.

- **Lack of a high-bandwidth interface**. Our platform lacks of a high-bandwidth interface, such a PCI, for dumping the data from the ML403 board to the PC for, e.g., post-processing purposes. To overcome this limitation, we perform an *offline* evaluation, that consists on storing the data to a massive memory on-board for the duration of a test. Afterwards, the data within the memory is transferred through the serial port (or JTAG) to the PC for post-processing.

The limitations exposed above and the internal structure of the FPGA have lead us to consider some other important modifications on the conceptual design presented in the previous chapter. Next, we show the changes to be introduced in the proposed time architecture.

5.3 Hardware Design

The next subsections explain the adaptations of our conceptual model due to architectural limitations of the FPGA. There is a description of the chosen blocks tailored to the functionalities of the synchronization platform and their specific role. There is also described in detail the architecture of our custom hardware block, the design challenges encountered during the design phase and the solutions to cope with them.

5.3.1 Architectural Adaptations

The architectural limitations of this commercial platform have forced us to take several design considerations. The foremost architectural change for this design is to bypass the MAC inaccessibility. In this Thesis, we envisioned the legacy Ethernet with a time synchronization functionality in the control plane of the MAC. Therefore, it might seem plausible to, first, design and implement a standard compliant MAC and then add our synchronization functionality. However, the time overhead and effort for designing a MAC from scratch is considerable, and would slow down the process of evaluation. Here, we want to rapidly work around this problem and have exploited the actual resources while taking some architectural decisions. Figures 5.2a to 5.2c show the adaptation of our initial conceptual model to the actual platform FPGA.

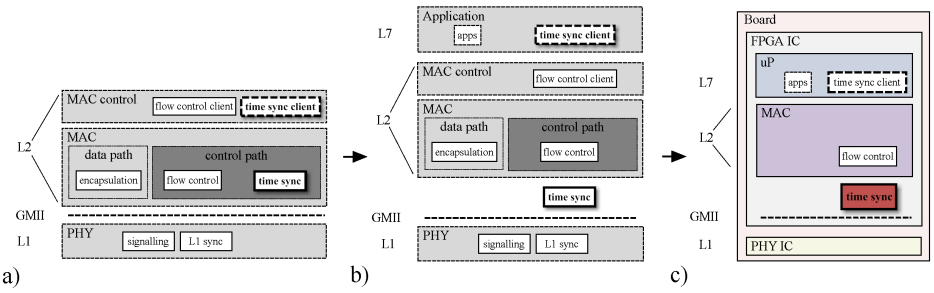

Figure 5.2: Proposed time synchronization extension in the Ethernet architecture (a)). Reallocation of the synchronization functionality considering the MAC inaccessibility (b)). Mapping of the synchronization functionality into the physical components of the platform FPGA (c)).

Figure 5.2a illustrates the initial design prospected in Chapter 4, and Figure 5.2b the proposed change. Virtually, the synchronization function is located in the control plane of the MAC, but it is physically between the GMII and the MAC, and below the flow control functionality. As we have access to the data and the flow control interfaces of the MAC, we have chosen to disseminate the time information by transparently changing the flow control information, i.e., the PAUSE messages. To that end, we will need a specialized unit capable of replacing on-the-fly the ingress/egress flow control information. Figure 5.2c shows

the modification in the FPGA chip. The software client that manages the flow control hardware function is defined in the MAC control client sublayer of most of the commercial NICs. The flow control client is de/activated manually by the user through the NIC Application Program Interface (API). Once the flow control routine is running, and when the internal reception FIFO of the NIC is close to overflow, it sends a notification and the flow control routine request the MAC to transmit a PAUSE. In our case, as we do not have access to the MAC control sublayer either, we have reallocated the synchronization protocol in the Application Layer. Figure 5.2c shows the physical allocation of the software part, which is implemented with low-level ansi C code and executed by the microprocessor block.

Considering all these limitations and in an intend to consume the minimum programmable resources as possible, in the next section we present the overall view of the synchronization platform and its working principles.

5.3.2 Synchronization Platform

Taking into account the architectural limitations explained before, we have chosen specific IP blocks within the ML403 platform to fulfill our needs. Figure 5.3 shows the main components and interconnects of the overall synchronization platform. The whole architecture is centered around the PPC 405 microprocessor (PPC) [Xilinx, Inc. (2007a)]. The PowerPC processor is a 32-bit licensable embedded core implementation of the IBM PowerPC^{TM} RISC processor. It characterizes to integrate a scalar 5-stage pipeline, separate instruction (*iplb*) and data paths (*dplb*), instruction and data caches, a JTAG port for debugging, multiple timers and a memory management unit (MMU), with 1.52 DMIPS/MHz performance, and for being capable to work at a peak frequency of 450 MHz. The PPC is supported by CoreConnect^{TM} technology, a high-bandwidth 64-bit architecture that runs from 100 to 133 MHz. In our platform, the PPC core and bus interface are running at 300 MHz and 100 MHz, respectively, in order to obtain the maximum performance for instruction execution and bus transfers. The functionality of the PPC is to execute different software applications and interact with the components on-chip.

The PPC accesses high-speed and high-performance system resources through the PLB bus, which provides separate 32-bit address and 64-bit data buses for the instruction (*iplb*) and data (*dplb*) sides. The processor clock and PLB must run from an integer ratio of 1:1 to 16:1. In our design, the PLB operate at its maximum speed, 100 MHz, while the PPC operates at 300 MHz. Although not shown in the figure for a better clarity, the arbitration of the PLB bus is done by a *PLB arbiter*, which receives bus requests from the devices attached to it and grants the bus to one of them.

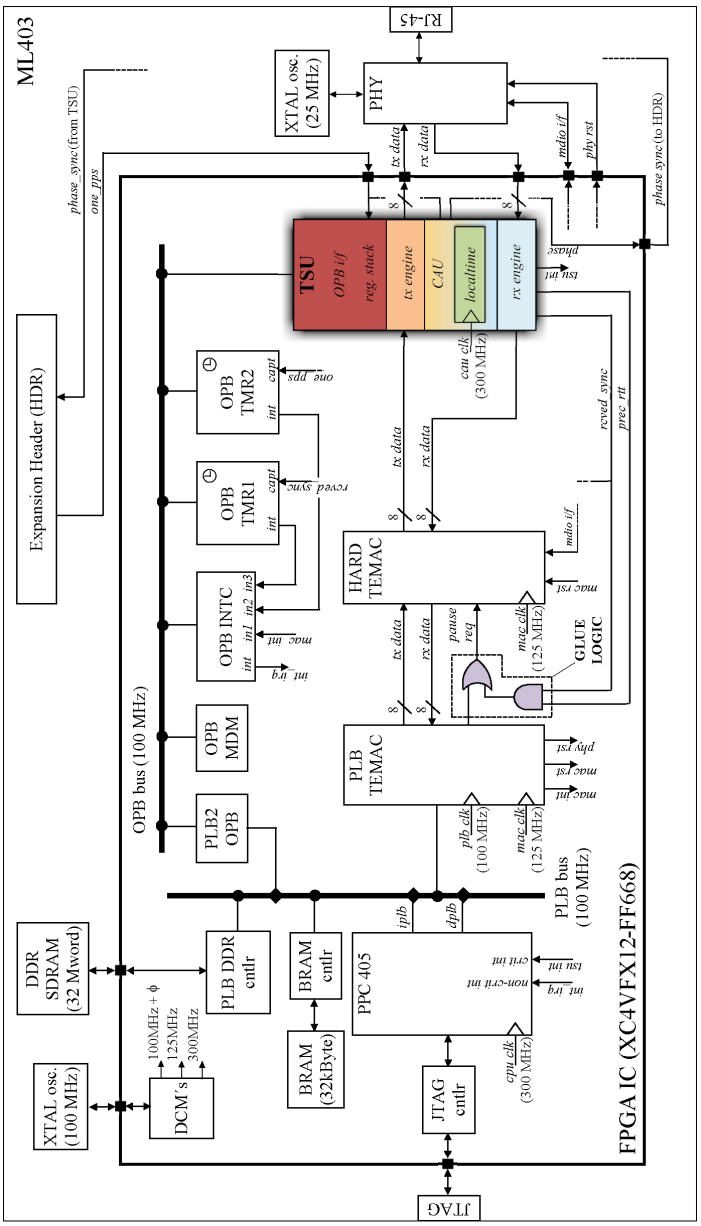

Figure 5.3: Embedded platform architecture and TSU allocation.

The devices adopt the role of master (noted as a '◇') or slave (noted as a '•'). One particular feature of the PLB is that it is an indeterministic bus, meaning that the bus latency may vary, depending on whether an operation is occupying the bus or not. As it is seen in Chapter 6, this represents a problem for time synchronization applications. Despite of the PLB indeterminancy, it is the best choice to interconnect those blocks demanding rapid accesses, such as the case of the memories. The DDR controller (*PLB DDR cntlr*) serves as an interface between the PLB bus and the external DDR SDRAM memory (*DDR SDRAM*), which hosts all the software application code (instructions and data). The Block RAM controller (*BRAM cntlr*) manages the flow of data between the Block RAM memory (*BRAM*), inside the FPGA chip, and the PLB. We use the BRAM memory to store some interruption handler routines that need rapid access by the PPC. Another block that is connected to the PLB bus is the PLB to OPB bridge (*PLB2OPB*), which translates 64-bit PLB transactions to 32-bit OPB transactions.

The OPB bus is the secondary bus and it is tailored for less complex, lower performance peripherals. The OPB has a shared 32-bit address and 32-bit data bus, and can run up to 100 MHz, with peak data rate transfers of 3.2 Gbps. The OPB hosts 5 peripherals in our design: a debug module (*OPB MDM*), 2 capture/timer blocks (*OPB TMR1/2*), an interruption controller (*OPB INTC*) and our custom block, the Timestamping Unit (TSU). It is specifically allocated here to replace on-the-fly the ingress/egress control frames (see Section 5.3.3). As per the rest of the blocks listed, the first one is a controller for debugging and monitoring purposes. It allows us to access to internal registers of the PPC during its operation. The timer/capture blocks have a primary function, which is to trigger an interruption through the output *int* when receive an external trigger on *capture* input or when an internal timeout expires. We have configured both timers to timeout after an specific interval and for capture detection. The interruption output (*int*) of the timers is fed into the interruption controller (OPB INTC), which is used to expand the number of interrupt inputs of the platform, prioritize over them and notify to the CPU the interruption request (*int_req*).

The *HARD TEMAC* is a standard compliant core provided by Xilinx that serves as a wrapper for the two MACs on-chip. The *HARD TEMAC* contains several configuration registers that can be written at any time to change the configuration of the silicon MACs. These EMACs may be configured for full or half-duplex operation and support several media interfaces including MII, GMII, RGMII, SGMII, and 1000Base-X. The Hard TEMAC also supports MII management of physical devices, PHY, VLAN frames, jumbo frames, configurable inter-frame gaps, in-band frame check sequences, FCS, for both transmit and receive, auto padding on transmit, FCS stripping on receive, flow control through Pause packets, receive address filtering, and provides raw statistics vector outputs. It is important to note that the MAC component (together with the CPU) consumes no FPGA programmable resources since it is built into the silicon of the FPGA.

HARD TEMAC can be accessed through the *PLB TEMAC*, which translates the PLB bus protocol to the MAC interface specified in the standard. *PLB*

TEMAC has also been designed incorporating the applicable features described in IEEE Std. 802.3-2002 [IEEE Std. 802.3 (2002)]. The *PLB TEMAC* enables memory mapped access to registers and memory mapped or DMA access to packet FIFOs, which in turn interface to the client transmit and receive interfaces of the *HARD TEMAC* to support transmission and reception of Ethernet frames.

The Digital Clock Manager (DCM) is used as a digital frequency synthesizer to derive the several frequencies used by all the devices on-chip and the DDR memory on-board. It generates the frequencies of 100 MHz (for busses, OPB peripherals and TSU's OPB interface, and the *JTAG*), 125 MHz (for the *PLB TEMAC* and *HARD TEMAC*), and 300 MHz (for the TSU's and CPU's counters).

Another key block in the platform for our evaluation purposes is the *GLUE LOGIC* block. As explained before, EPON synchronization mechanism calculates RTDs to infer the propagation time from the sender to the receiver Ethernet layers. The task of the *GLUE LOGIC* block is to create a hardware loop to bound incoming syncPDUs without the intervention of the PPC. The user set the TSU in *prec_rtt* mode (see Section 5.4) and, upon the arrival of a non-errored syncPDU, TSU's *rcved_sync* assertion pulse makes the *HARD TEMAC* block to immediately generate another PAUSE frame.

The CPU and the MAC are hardcoded in the silicon of the FPGA, thus they almost[1] do not consume logic resources. However, the rest of IP blocks do use programmable area that depends on the number of functionalities desired for each of them. In FPGA design, the less IP blocks added, the better the design will be (in terms of area consumption, speed and correct functionality). Table 5.2 summarizes the programmable resources used to implement the platform of the Figure 5.3.

Table 5.2: Summary of the programmable resources utilized for TSU implementation and percentage of the overall amount.

Slices	LUTs	FFs	Equivalent Gate Count
4956 out of 10944 (45%)	5262 out of 10944 (48%)	4938 out of 10944 (45%)	1560072

5.3.3 Message Handling

This Thesis follows the concept of having separate paths to split the functionalities of forwarding and control. This concept was introduced in the legacy Ethernet with the flow control, which aims at preventing the MAC FIFOs being overflowed due to transient traffic congestion situations. Rather than the operation mode of the flow control, we are interested on the flow control message semantics, as we will re-use them to generate synchronization messages (syncPDU). In the MAC of

[1] All hardcored blocks need minimal use of programmable resources for glue logic at the output interface of the IP block.

our platform, we have access to the flow control interface through two dedicated lines [Xilinx, Inc. (2007b)].

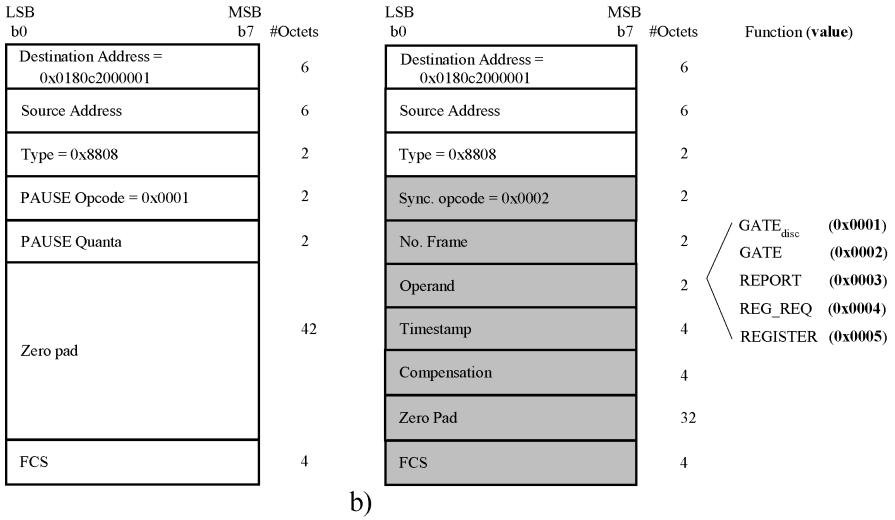

Figure 5.4: Pause control frame (a)). Synchronization protocol data unit (syncPDU) (b))

Figure 5.4 depicts the generalized MAC Control frame format of the legacy Ethernet. All Control frames are exactly 64 bytes in length, not including the Preamble and Start-of-Frame Delimiter. The frame starts with the reserved Destination Address (DA) that contains the fixed value of `0x0180c2000001`. When the receiving MAC parses the frame and matches this value, it knows that it must enter into the *control mode*. The Source Address (SA) is filled with the MAC address of the flow control requester. The next field contains the unique Type field that has been reserved for Ethernet MAC Control, `0x8808`. Within the Data field of the frame, the first two bytes identify the MAC Control opcode, or the control function that is being requested by the frame. Currently, there is only one control function defined, thus this value is constant in the MAC and is set to `0x0001`. Next to the opcode field, it follows two more bytes to indicate the Pause quanta value which, in the case of the flow control, it sets the time that the transmitter must be idle. The remainder of the frame is padded with zeros.

In Figure 5.4b, it is shown the modified PAUSE frame, the syncPDU. Its length has not been modified and only some of the fields have been replaced. It starts with the DA field which is kept with the same code `0x0180c2000001` for the receiving TSU to be able to recognize an incoming syncPDU. The SA field is filled with the MAC address of the flow control requester. The Type field is not modified either. The modification starts from byte 15 until the end of the frame, byte 72. The first two bytes explicit the synchronization control mode, with the value `0x0002` and tag *Sync. Opcode*. The field *#Frame* is for

indicating the number of synchronization frame being sent. *Operand* field is used to differentiate the synchronization messages during the message exchange mechanism (see Section 5.5). *Timestamp* is a 4-byte field that contains the value of the TSU's counter that is triggered during the transmission of the 16^{th} byte of a syncPDU. *Compensation* field contains the value to subtract to the receiving timestamp in a bridged Ethernet scenario. The remaining bytes are zero padded until the *FCS* field, which contains a new cyclic redundant code (CRC) for the newly generated syncPDU.

5.3.4 Platform Functionality

Sending and receiving synchronization data is handled cooperatively by the TSU and some intermediate hardware elements of the platform, such as the timers, the microprocessor and the MAC, and software routines stored in the BRAM memory. To better understand the functionality of the platform, Figure 5.5 dissects a transmission and a reception of a syncPDU.

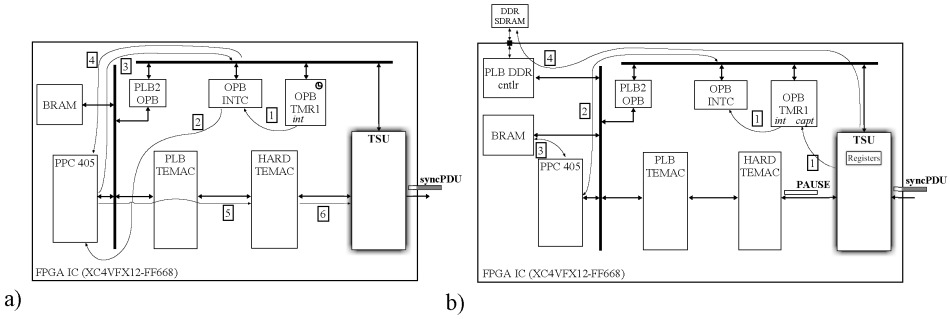

Figure 5.5: Steps involved in sending (a)) and receiving (b)) a syncPDU.

Sending a synchronization packet requires the steps shown in Figure 5.5a. In step 1, the OPB TMR1 timeout expires and sends an interruption request to the OPB INTC. The OPB INTC notifies the PPC that some of the OPB peripherals is requesting an interruption, in step 2. In step 3 of the figure, the PPC asks back to the OPB INTC which peripheral notified an interruption. In step 4, the OPB INTC provides an interruption vector to the PPC, which starts the interruption process. In the interruption process, the PPC executes an interruption handler which basically writes in those registers of the TSU that will map the NUM and OPER fields of the TSU (see Section 5.5). The instructions of the routine are stored in the BRAM memory for faster accesses. In step 5, after the TSU pre-loading, a PAUSE frame is requested to the PLB TEMAC, which signalls the HARD TEMAC for generating it. Finally, when the MAC starts sending the PAUSE, the TSU replaces some of the fields on-the-fly and creates the syncPDU at the GMII.

Receiving packets can be seen as the reverse process, i.e., the incoming syncPDUs are replaced to PAUSE frames with its *pause quanta* field zero-padded (to

avoid setting the receiving MAC in PAUSE mode). Figure 5.5b depicts the steps for receiving a syncPDU from the network to the DDR memory. In step 1, the TSU has transformed the incoming syncPDU into a PAUSE, which is already in the MAC. Then, when the TSU finishes checking that the incoming syncPDU has no errors, it generates a pulse through the *rcved_sync* output. This output is fed into the *capt* input of the OPB TMR1. In step 2, the timer initiates the same interruption handshaking process with the CPU as in the transmission. In step 3, the CPU is already executing the interruption routine for the reception, which basically consists on dumping in the DDR memory the content of some of the TSU registers, i.e., those that store the information during the reception (step 4). In the case that the TSU detects an incoming syncPDU has a CRC error, the TSU does not assert *rcved_sync* output and the overall process does not start. Besides, it notifies the MAC to discard the PAUSE frame through the MAC interface.

5.4 Timestamping Unit

This section explains in detail the internal architecture of our Timestamping Unit, their main functional blocks and the reasoning for some design choices. We start by collecting and clarifying the objectives and requirements of the TSU, and a detailed explanation of its internal architecture and main design challenges follows.

5.4.1 Requirements and Functionalities

Our TSU is a hardware block that was conceived to provide an infrastructure with which to measure the latency and jitter of Ethernet layers and to assess the time synchronization accuracy with a link partner. To meet these goals, we find of importance to design a hardware block capable of meeting the following requirements and functionalities:

⇒ *Control frame generator.* To separate the control path from the forwarding path and create the control frames, we would need to either redesign a new MAC from scratch or to provide a block in parallel with the actual MAC to create the control frames. Both approaches lack of simplicity and might not be the right approach to cope with our objectives rapidly. A better and simpler option is to reuse the actual control frames and replace on-the-fly those fields of interest. This requirement lead us to partition functionalities in the TSU, i.e., the transmission and the reception component. In each part, we must provide logic to replace the fields of our interest and regenerate the FCS field of the newly generated syncPDU. Besides, the reception part must contain a block to check against CRC errors in the incoming syncPDU.

⇒ *High precision timestamping.* The latency and jitter introduced by hardware mechanisms are few orders of magnitude smaller than those introduced

by an operating system. To be able to measure them, we need to generate timestamps with the highest resolution possible. Due to the limitations of our platform FPGA, we are restricted to timestamps of $3.\widehat{33}$ *ns* resolution.

⇒ *High precision propagation time cancellation.* As explained in Chapter 4, if a receiving node is able to cancel the propagation time of a syncPDU, it will be able to synchronize with its link partner with an accuracy given the jitter introduced by Ethernet layers. The propagation delay value is communicated by the sender to the receiving node using the message exchange pattern explained in Figure 4.3. In the TSU, we must provide a hardware mechanism capable of subtracting the propagation delay from the received timestamp.

⇒ *On-the-fly timestamping.* An on-the-fly timestamping property requires to timestamp transparently, i.e., to replace the fields of a frame with minimal or no latency. In digital logic design, this requirement poses some advantages and limitations. First, "no latency" implies avoiding the use of FIFO-based architectures to store and forward the frame. This simplifies greatly the design of the overall system, but on the other hand, FIFO avoidance forces to store only the *last sent/received* information. In communication IP blocks, this translates to read/write more often to, e.g., the registers of the block, thus posing an increase of utilization in terms of bus accesses. Another added drawback is on the latency introduced by some hardware subblocks, e.g., the cyclic redundancy check (CRC) block. In this case, it will be necessary to implement it using latches instead of flip flops. The use of latches in digital design is strongly not recommended as it can harm the reliability of the design. A latch is an asynchronous memory unit that can suddenly change the stored information due to, e.g., a glitch in its input.

⇒ *Asynchronous system.* Digital systems that require to be synchronous, the electrical clock signal of the receiver must track that of the transmitter. In our platform, this would represent to re-use the transmission clock to provide the timestamps too, thus we would loose timestamp resolution and refinement for our evaluation purposes. If we want maximum timestamp resolution, we are forced to use multiple and non-related clock sources to pace different circuit portions: a clock to transmit, a clock to receive and the clock for obtaining the timestamps[2]. In digital logic design, this scenario is also known as clock domain crossing (CDC). Those designs with CDCs are prone to loose or mislead the digital information due to a phenomenon called *metastability* [J. Stephenson (Altera, Corp.) (2009)]. They require a careful design and mechanisms to protect against information looses and misinterpretations. With this background, we must provide a mechanism capable of protecting the information subjected to clock domain crossings, such as the case of the timestamps.

[2]The three clocks interact and the different circuits they pace exchange information asynchronously.

The next sections explain in detail the architecture components of the TSU, as well as the design approach followed to fulfill the requirements bulleted above.

5.4.2 Distributed Timestamping

As seen in the previous Chapter, a key parameter that will help us to determine the level of jitter, and thus the achievable synchronization accuracy at Layer 2, is the latency of a syncPDU to go across the layered stack. The finer the propagation time calculation is, the finer the measured jitter will be. To infer the propagation delay, we use the message exchange pattern defined in Chapter 4. To obtain precise RTDs, we have added a small glue logic that allows the receiver node to bounce immediately a synchronization frame. However, there is still a small source of delay error coming from, first, the time that the MAC needs to generate a new PAUSE, and second, the time that lasts from the first syncPDU byte to the time for the TSU to take the timestamp snapshot (i.e., the timestamp to send). To better understand this situation, consider the Figure 5.6a.

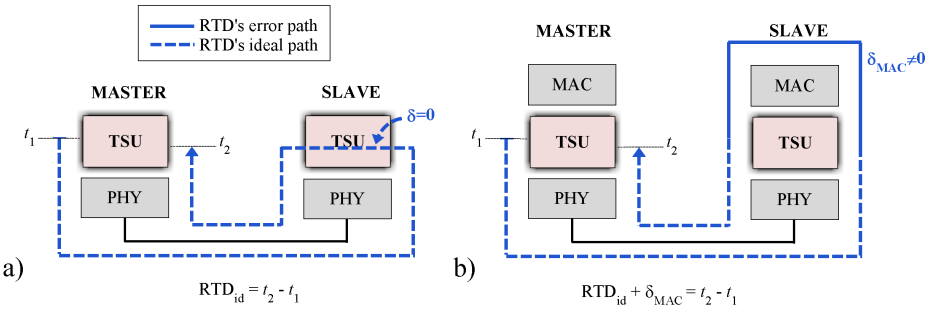

Figure 5.6: syncPDU journey with no inter-layer delay (a)). Platform's syncPDU journey, with additional delay to regenerate the syncPDU reply (b)).

The time for a syncPDU to travel from its node to the link partner and bounce back is not influenced by the internal delay of the layer ($\delta = 0$), i.e., $RTD_{id} = t_2 - t_1$. In this situation, we would obtain precise RTD and interlayer propagation time calculations. However, due to the architectural limitations of our platform, slave's reply needs to be regenerated again in the MAC, and therefore adding an additional error on the calculated RTD given by the latency of the MAC's internal circuitry, i.e., $RTD_{id} + \delta_{MAC} = t_2 - t_1$ (see Figure 5.6b).

In order to correct the latency introduced by the MAC in the reply, we have introduced the concept of *distributed timestamping*. It consists on capturing different timestamps during the syncPDU journey and use them strategically to cancel MAC's latency. Figure 5.7 illustrates our approach. We have chosen significative time points like the start of the syncPDU, the time point where the timestamp is triggered and the timestamp when the last byte is sent. The distributed timestamps keep a symmetry in their position in both the transmission

and reception paths. Table 5.3 summarizes the logical variables assigned to these timestamps together with the TSU's associated register.

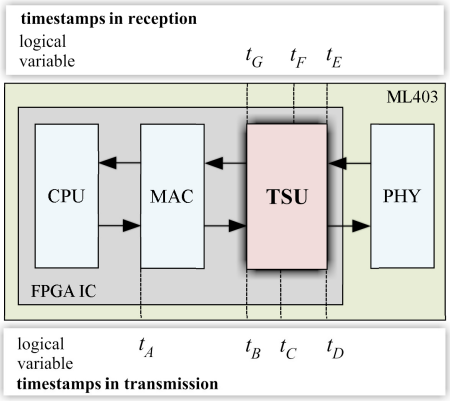

Figure 5.7: Distributed timestamping architecture.

Table 5.3: Description of the distributed timestamps (logical variables) collected during a syncPDU transmission and reception.

Mode	Logical Variable	Description
Transmission	t_A	syncPDU transmission request event
	t_B	first transmitted PAUSE byte
	t_C	timestamp to be transmitted, (taken at the 16^{th} syncPDU's byte)
	t_D	last transmitted syncPDU's byte
Reception	t_E	first received syncPDU's byte
	t_F	syncPDU's time of arrival (taken at the 16^{th} syncPDU's byte)
	t_G	last received syncPDU's byte

The distributed timestamping architecture is not only conceived for obtaining more precise RTD calculations, but it also gives us a more flexible evaluation platform for deriving internal timings (such as the MAC and PHY latencies) or for evaluating the synchronization accuracy between two link partners. Consider again the Figure 5.7. From timestamps t_A and t_B, we can obtain the MAC latency, or, with proper means for looping back outbound frames, the timestamps pairs (t_B,t_E), (t_C,t_F) and (t_D,t_G) can provide information about the PHY latency to send and receive a syncPDU. In the next Chapter, we take advantage of the distributed timestamping architecture and evaluate different synchronization components such as the PHY and MAC latency and the synchronization accuracy. Next, a description of the main components of the TSU follows.

5.4.3 TSU Architecture Description

In Figure 5.9 it is shown the overall architecture of the TSU. It is divided into four main regions: The transmission engine block (*tx engine*) (in light yellow), the reception engine block (*rx engine*) (in light blue), the clock adjustment unit (*CAU*) (in light green), the data register block (*data register block* and *control register*) (in red) and bus interface logic area (*bus interface logic*) (in red). Next, a detailed description of each region comes.

Transmission Engine

The main task of the *tx engine* is to detect the egress PAUSE frames sent by the MAC and replace on-the-fly some of their fields to create a syncPDU (recall Figure 5.4). All the blocks within *tx engine* have been designed and interconnected according the signalling protocol for sending a control frame. Control frames characterize to be fixed-length frames of 64 bytes, a fact that gives us certain advantages in the design process. Figure 5.8 shows the timing of a PAUSE frame. When a control frame is being sent, the `TX_EN` signal is asserted by the MAC during the transmission of the 72 bytes. *tx engine* starts the replacement operation once detects the Start of Frame Delimiter (*SFD*) code. SFD differentiates from Preamble field (*PRE*) in the last nibble, and it is used to notify the PHY that the payload is starting at the next byte.

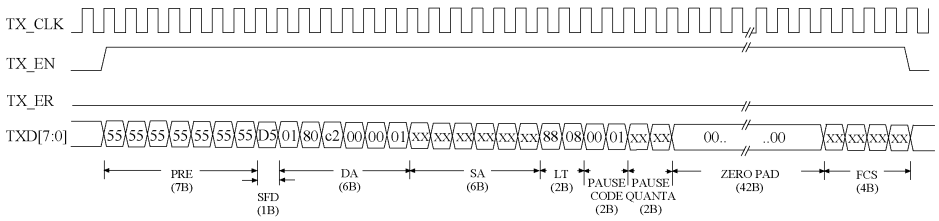

Figure 5.8: PAUSE frame transmission waveform across MAC interface (adopted from [Xilinx, Inc. (2007b)]).

tx engine's part of the Figure 5.9 shows several functional blocks that perform different functionalities during the frame transmission. For instance, the block tagged *pattern recognizer* is committed to detect and inform whether the DA field of an incoming frame contains a PAUSE code (01-80-c2-00-00-01) or not. The rest of the blocks perform the following tasks: *CRC gen* creates a new CRC code for the new generated syncPDU. *word to byte* are two shift registers that slice 32-bit data (coming from the CAU and data register zone into 8-bit data needed by the GMII. *MUX 1* and *MUX 2* multiplex different output data coming from the different blocks.

The main component of the *tx engine* block is the transmission finite state machine (*tx FSM*). It is a Moore FSM that starts upcounting upon the rising edge of `TX_EN` signal and activates or deactivates the internal blocks accordingly.

Its working principle is straightforward: it leverages the fixed length of the frame to know the position (in bytes) of each field. It starts activating the first block, *pattern recognizer*; if this detects that a PAUSE frame is flowing, it asserts the *hit* signal to the *tx FSM* which starts activating/deactivating the next blocks. The *tx FSM* also provides triggering to the CAU block for obtaining the distributed timestamps during the transmission (*start tx*, *timestamp2txmit* and *end tx*).

The block *synchronizer* is an output stage for synchronizing the incoming PAUSE frame to the modified one. It consists of a two-stage 32-bit registers synchronized to the transmission clock (*tx clk*) that provides us a lead gap of 2 GMII clock cycles (16 *ns*) to perform the replacement operations on-the-fly.

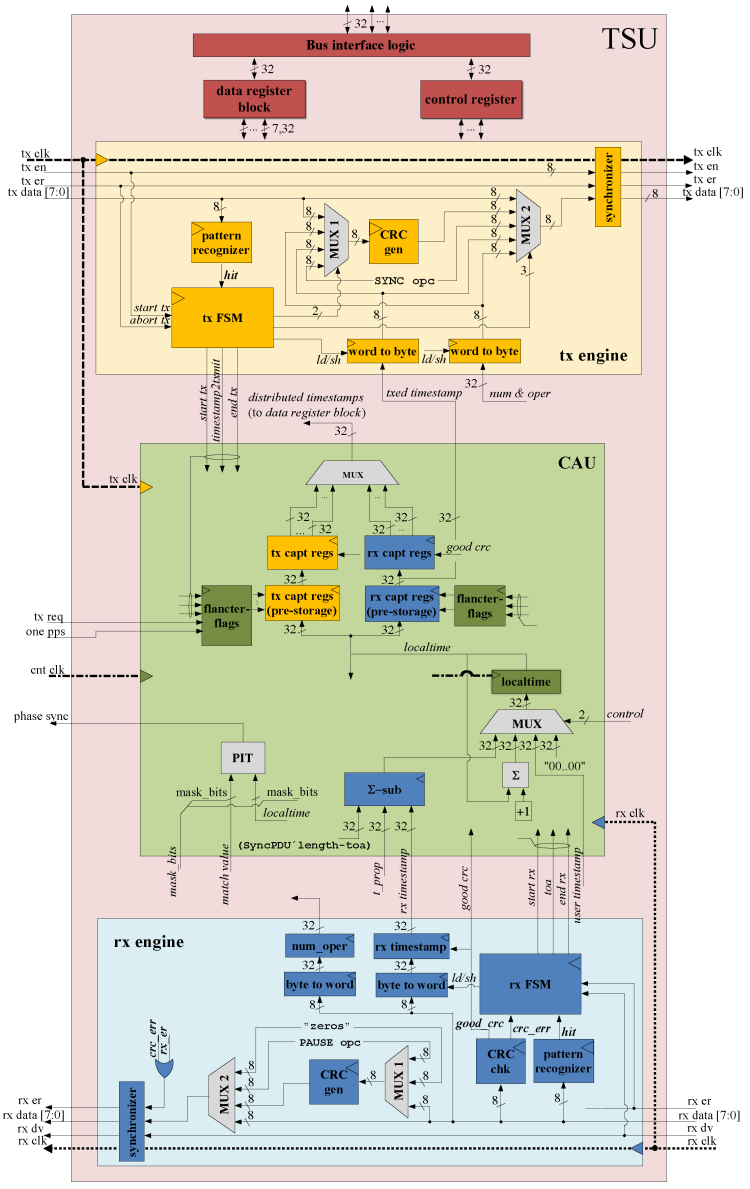

Figure 5.9: Internal architecture of the Timestamp Unit.

Reception Engine

The role of *rx engine* is similar to *tx engine* which is to transform syncPDUs to PAUSE frames. Here, the blocks are designed to perform the reverse operation, i.e., to transform incoming syncPDU delivered by the PHY to PAUSE frames for the MAC (see Figure 5.10). The FSM recognizes that a syncPDU is ingressing after receiving the notification of the *pattern recognizer* block, which asserts a logic '1' through *hit* signal. *Rx FSM* is aware of the syncPDU skeleton and enables/disables the internal blocks accordingly. In the reception, the incoming information is transformed from bytes to words through the *byte to word* 8-bit shift registers. Both 32-bit words, i.e., the received timestamp (*rx timestamp*) and Number and Operand fields (*num_oper*) are pre-buffered in their respective shift registers inside the *rx engine* block. Once the CRC checker block (*CRC chk*) detects that the incoming frame has no CRC errors, it asserts *good_crc* and the values are transferred into the registers of *data register block*.

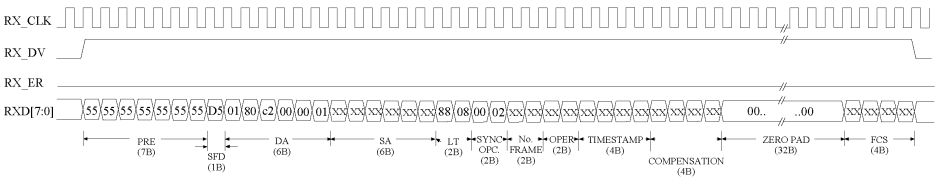

Figure 5.10: SyncPDU frame reception waveform across PHY interface (adopted from [Xilinx, Inc. (2007b)]).

In case of a CRC error, it asserts *crc_err* signal for 3 GMII clock cycles to notify to the MAC that the PAUSE frame that has already been converted is erroneous. As in the transmission case, a synchronizer is also needed to reconcile the incoming frequency (*rx clk*) at the PHY side to the output frequency at the MAC side. The synchronizer provides a gap of two extra clock cycles to modify the incoming syncPDU on-the-fly.

As in the transmission block, the *rx FSM* triggers the distributed timestamps at the first byte of the incoming PAUSE (*start rx*), at the 16^{th} byte (*toa*) and at the end of the reception (*end rx*).

Clock Adjustment Unit

The Clock Adjustment Unit (CAU) is the centerpiece of the TSU architecture. It contains a 32-bit counter (*localtime*) that is summing up clock ticks at the frequency of 300 MHz and provides a timestamp granularity of $3.\widehat{33}$ *ns*. In our design, *localtime* counter keeps track of high resolution time at MAC level.

Some time synchronization protocols use instead the microprocessor's high-speed counter (also known as Time Stamp Counter (TSC)) to obtain the timestamps [Ridoux & Veitch (2007)], while in other architectures, such as EPON, the internal counter is physically located in the MAC. The subblock within the

Figure 5.4 corresponding to the CAU contains a MUX that multiplex three 32-bit signals: the output of the subtractor block ($\Sigma - sub$), the localtime instantaneous value and the timestamp entered by the user (*user timestamp*). The subtractor block subtracts the propagation time (in figure tagged as *t_prop*) and the syncPDU remainder (`SyncPDU'length-toa`) from a received timestamp. `SyncPDU'length-toa` allows to cancel the time that lasts from the time to trigger the arrival of the timestamp (*toa*) to the last received byte (i.e., the CAU waits for the *good_crc* assertion signal). Thus, when the CAU is configured to set the localtime to the receiving timestamp, the replaced value is free from that latency (`SyncPDU'length-toa`) and the propagation time to the partner link.

Another key block of the CAU is the programmable interval timer (PIT). It consists of a comparator that raises a 50 *ns*-wide pulse when *localtime* instantaneous value matches a user pre-set value (*match value*). *mask bits* input allows to stretch out or to narrow the interval of the pulses from ∼50 *ns* to ∼14 *s*. The output of PIT is connected to *phase sync* external output. As seen in the next chapter, this functionality is needed to evaluate the phase synchronization accuracy.

A crucial component within the CAU block is the block ensemble that consists of specialized circuits called *flancter-flags* and the four pool of registers (*tx/rx capt registers*). *localtime* is connected to *tx/rx capt registers*. When each FSM asserts the control signals for storing the distributed timestamps, the instantaneous value of *localtime* is stored in the respective register. A major problem here is the interaction of the different frequencies: the *localtime* clock frequency (*cnt clk*), which is running at 300 MHz, and the pulses coming from the *rx engine* and *tx engine*, which are synchronized with their respective clock signals (*tx clk* and *rx clk*) running at 125 MHz. If the control pulses are not aligned with *cnt clk* frequency, the values transferred from *localtime* to the *tx/rx capt registers* will be likely misinterpreted. To solve this problem, we have introduced the *flancter-flags* which reconcile the three clock domains allowing correct transfers from *localtime* to *tx/rx capt registers*. In the next section, we will explain in more detail its working principle.

Finally, the CAU has a dedicated input for using the 1PPS signal coming from a GPS receiver for evaluation purposes. As the 1PPS signal is also asynchronous to the *localtime*'s frequency, we also re-synchronize it with the help of the *flancter-flags*. In Chapter 6, the 1PPS signal is used to derive the oscillator clock drift and to obtain the real frequency of *localtime* counter.

Registers Block

The registers block is another key component in the overall architecture of the TSU as they provide the user interface to interact with the rest of the blocks, the *tx engine*, the *rx engine* and the *CAU*. The approach followed in the development of the TSU was to control and monitor their different functionalities and statuses by writing specific registers. As seen in Figure 5.9, the register block comprises a pool of 18 individual 32-bit registers (*data register block*) for storing the last transmitted and received fields of a syncPDU, as well as other information for

evaluation purposes. Data registers are directly wired to the inputs and outputs of the *tx engine*, *rx engine* and *CAU* blocks. From the OPB side, each register can be accessed following the data transfer protocol of the OPB bus. The registers zone also contains one 8-bit control register (*control register*) for setting the different TSU operation modes. The bits of *control register* are directly wired to specific inputs of the internal blocks, and can be individually written/read from/to the bus following the OPB data transfer protocol. Table 5.4 summarizes the function of each data register, as well as their addresses space for read and write accesses. Table 5.5 summarizes the functionality of each bit within the control register.

Table 5.4: Data register definition within TSU's *register block*.

Register Name	Access Type	Offset Address	Block Association	Location	Description
`tsu_ctrl`	RW	0x00	TSU	Reg. block	Control of TSU operating modes.
`rxed_ts`	R	0x04	Rx engine	Data reg.	Received timestamp.
`ext_time`	RW	0x08	CAU	Data reg.	External time.
`cau`	R	0x0C	CAU	Data reg.	Instantaneous *localtime* value.
`num_oper_tx`	RW	0x10	Tx engine	Data reg.	Transmitted Num and Oper fields.
`num_oper_rx`	R	0x14	Rx engine	Data reg.	Received Num and Oper fields.
`off_corr_tx`	RW	0x18	Tx engine	Data reg.	Propagation time to communicate.
`match_val`	RW	0x1C	CAU	Data reg.	Pulse interval period.
`match_rng`	RW	0x20	CAU	Data reg.	No. bits to mask for `match_val`.
`tx_req`	R	0x24	Tx engine	CAU	SyncPDU transmission request.
`st_tx`	R	0x28	Tx engine	CAU	First transmitted PAUSE byte.
`txed_ts`	R	0x2C	Tx engine	CAU	Timestamp to be transmitted.
`end_tx`	R	0x30	Tx engine	CAU	Transmitted syncPDU's byte.
`one_pps`	R	0x34	–	CAU	1PPS event.
`st_rx`	R	0x38	Rx engine	CAU	First received syncPDU's byte.
`toa`	R	0x3C	Rx engine	CAU	Received time of arrival of a syncPDU.
`end_rx`	R	0x40	Rx engine	CAU	Received syncPDU's byte.
`t_prop`	RW	0x44	CAU	Data reg.	Propagation time for cancellation.

Table 5.5: Control register definition within TSU's *register block*.

Bit no.	Access Type	Association	Description
0	RW	TSU	TSU general reset.
1	RW	CAU	Set TSU in re-synchronization mode.
2	RW	CAU	When asserted, *localtime*. is loaded with `ext_time`.
3	RW	CAU	Reset *localtime*.
4	RW	TSU	Disables the CAU block from inserting timestamps from *localtime*. `t_prop` is inserted in *timestamp* field.
5	W	Tx engine	Enables to bounce incoming syncPDUs without CPU's intervention.
6	W	TSU	Pulse assertion. For debugging purposes.
7	W	CAU	*localtime*'s read request.

Bus Interface Logic

The four blocks explained so far are considered to be the entire design of the TSU block and can be reused in other FPGA architectures. However, the IP blocks within embedded systems are interfaced through busses. For our design, we have chosen the OPB bus to access to the registers of the TSU for its simple data transfer protocol. As our synchronization exchange mechanism is not subjected to intense communication throughputs, we can disregard the use of more complex bus access techniques and design specific bus access logic for single read/write transfers of 32-bit. To simplify even more the bus logic interface, we avoid protection techniques and re-transmission operations, and design the logic to work as a slave.

Another aspect to consider is the addressing mode of the busses. The OPB and PLB are addressed on a byte basis, but has a 32-bit wide data bus. Therefore to access to the 32-bit wide registers of the TSU, we will need to add multiples of 4 to the peripherals base address. This translates on designing the address decoding logic to select the upper nibble of the TSU address space (see the third column of the Table 5.4).

5.4.4 Hardware Design Challenges

This subsection presents one of the contributions of this Thesis which is a hardware circuit called *flancter-flag* that provides timestamping reliability. The subsection finishes with another a design consideration when using FPGAs: the timing closure.

Clock Domain Crossing (CDC)

One of the major challenges in the design of the TSU is the presence of four clock domains. Each clock signal paces a logic resource region inside the circuit and the information transfer from one domain to another can render to subtle, intermittent and unpredictable corrupt control and data signals. The rationale for this behavior is on the phenomenon known as *metastability* [J. Stephenson (Altera, Corp.) (2009)]. Metastability is a major concern in digital logic design. It is the second main cause of ASIC chip respins and they only occur in hardware, i.e., when a design is already implemented on-chip. In the case of FPGAs, metastability appears after the place and route phase.

As shown in the Figure 5.11, the four asynchronous clock sources are: the frequency of the CAU (300 MHz) (noted with '-·' line), the GMII frequencies for the transmission (125 MHz) (noted with '- -' line) and the reception path (125 MHz + ϕ) (noted with '··' line) and the OPB bus logic (100 MHz). The frequencies of the GMII are equal but their phases are different, hence they can be considered asynchronous too. As shown in the left side of the Figure 5.11, the logic in the transmission engine block (tx engine) retrieves the timestamps from the CAU at the speed of 125 MHz but the time in the *CAU* is summed up faster. If the edges of the two clocks (*CAU* clock and *tx engine* clock) are not aligned,

timestamps will be misinterpreted unpredictively. This situation can take place in any of the two clock regions: *CAU-rx engine* and *CAU-tx engine*.

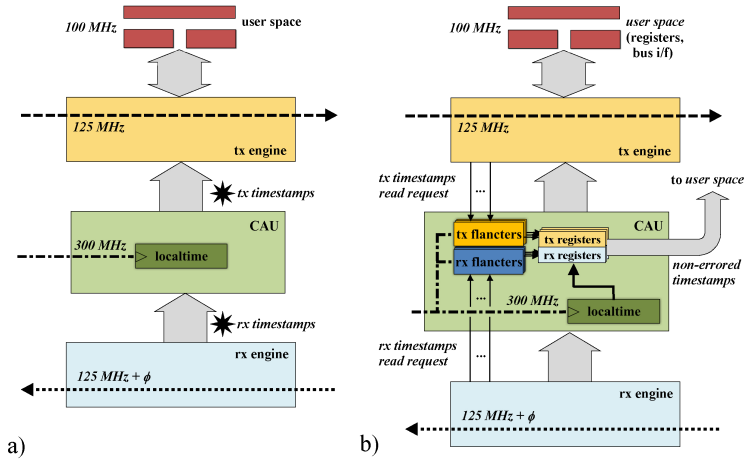

Figure 5.11: TSU without timestamping reliability mechanism (a)). TSU with timestamping reliability mechanism, with flancter-flags. (b))

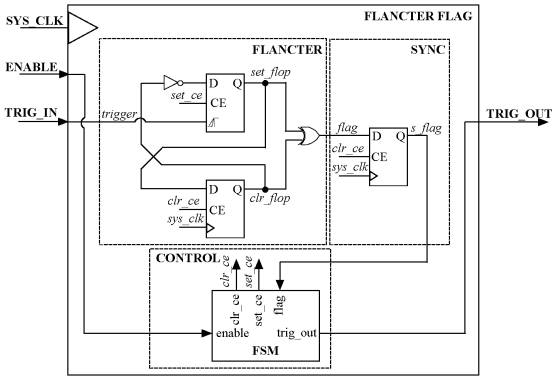

Figure 5.12: Flancter-flag circuit.

Figure 5.11b shows the integration of the flancter-flags circuits within the TSU architecture. The flancter-flags reconciliate the CAU frequency with the frequencies of the *tx engine* and *rx engine*. The flancters detect the rising edges coming from read requests of the distributed timestamps (*tx/rx timestamps read request*). Once the flancters have their internal flags on, they transfer reliably the value of *localtime* counter to the *tx/rx registers*. The flags set by the flancters are automatically resetted (by their respective control blocks) and the distributed timestamps can be read through the user space.

Timing Closure

Another big design challenge in FPGA-based designs is related to the propagation delays of digital signals inside the chip. Custom blocks are located in areas of the FPGA where routing availability can be problematic as a result of congestion around these blocks. Poor resource placement can result in timing not meeting all requirements. Conflicts can occur between resource, area, power and timing requirements in a design. When a design requires more resources, the resources have to be spread out across the target device, some of them having long interconnections. At smaller device geometries, delays are dominated by interconnect delays rather than logic delays. To have shorter net lengths, the area should be smaller. Therefore, these two requirements generally conflict.

A noticeable feature in FPGA architectures is their sensitivity to coding styles and design practices compared to, e.g., ASICs. Some FPGA design practices consist on creating appropriate hierarchical blocks, registering all block inputs and outputs, using different blocks along the FPGA die for optimal performance, adding pipeline stages to break up long data paths in multiple clocks and using *case* statements instead of *if-then-else* to enhance the speed of multiplexing. In our design, we follow several design rules given by the FPGA manufacturer to achieve timing [Whatcott, R. (Xilinx, Inc.) (2009)] and have used the minimum number of IP blocks in order to reduce the number of interconnections, and thus to reduce the congesion too.

A very challenging requirement in our design is the TSU's counter frequency, targeted to be 300 MHz, the maximum frequency achievable with this platform. At this frequency we can obtain timestamps with a granularity of $3.\widehat{33}$ *ns*, and thus more precise measurements. Since clock signals are usually high-fanout signals, they are routed via dedicated routing networks within the FPGA and can be separately managed. In our work we have respined several times the physical design process and have put a lot of effort by manually tracing the TSU's clock tree. We have also arranged the frequency synthesizer block that delivers 300 MHz clock signal next to the TSU block. To do that, knowledge of the location of the programmmable resources on-chip is required [Xilinx, Inc. (2008)].

5.4.5 Used resources

The TSU has been developed with Xilinx development tools ISE 9.1 SP3, and has been integrated within the embedded platform with EDK 9.1 SP2. Both phases of the design have been verified usign Modelsim 6.2c. Table 5.6 summarizes the hardware resource utilization of the TSU reported by the synthesis tools. In terms of slices, i.e., summing up combinational and sequential logic, the whole TSU occupies the 19% of the overall programmable resources of the FPGA. LUTs, that map the combinational functions of the TSU, occupy the 14% of the programmable area, while sequential circuitry (Regs.) consumes the 12% of FPGA resources. IOs correspond to the used number of external TSU outputs interfacing other components, including embedded system components (e.g., the OPB bus) and FPGA external pins. As explained before, the TSU uses registers

for information storage, thus no BRAMs from the FPGA silicon resources are used.

Table 5.6: Summary of the programmable resource utilization of the TSU (expressed as used amount and percentage).

Slices	Regs.	LUTs	IOs	BRAM
1091 (19%)	1357 (12%)	1609 (14%)	169	0 (0%)

5.5 Software Design

Although the big part of the design weight relies on hardware, the software also plays an important role in FPGA-based designs. The software architecture is designed as a layered structure as shown in Figure 5.13. The layered architecture accommodates the many use cases of device drivers while at the same time providing portability across operating systems, toolsets and processors. The layered architecture provides seamless integration with RTOS (Layer 2), high-level device drivers that are full-featured and portable across operating systems and processors (Layer 1), and low-level drivers for simple use cases (Layer 0). Those designs that require a direct interaction with the hardware, such as the case in this Thesis, it is recommendable to design the low-level device drivers as they give to the user an "in-the-box" solution and thus a total hardware controllability and measurability.

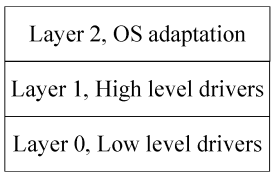

Figure 5.13: Software layered architecture.

In the next sections, we describe the key software design decisions, its operation and its adaptation to the hardware plane.

5.5.1 Requirements

In embedded systems, it is common that platform manufacturers provide implementation specific support code for a given board through a collection of low and high level drivers called board support package (BSP). BSPs conform to a given operating system and the device drivers for all the devices on-board. Our platform is also provided with a library of Layer 0 and Layer 1 device drivers to interact with the IP blocks such as the OPB timers, the PPC or the PLB TEMAC to access to the Xilinx MAC. For custom IP blocks, the software drivers must be specifically designed taking into account the layout of the registers (see Tables 5.4

and 5.5). For the design of the TSU drivers and the application functionality, we aim at meeting the following objectives and requirements:

- *Simplicity.* We aim at designing a single-threaded environment that provides basic features like standard input/output and access to processor hardware features.
- *No Error handling.* In memory-mapping strategies for designing hardware, the registers of an IP block can be accessed by writing/reading to/from a memory position in the overall memory map. While Layer 1 software abstractions often include means to verify the integrity of the association hardware peripheral-pointer variable, Layer 0 software drivers do not. Error handling adds an important amount of non-desired latency in the execution of the instructions. Thus, we disregard the use of such a functionality.
- *Small Memory footprint.* Low-memory-footprint programs are of paramount to running applications on embedded platforms or Platform FPGAs, where memory is often constrained to within a few MBs. Here, we want to find a tradeoff to meet efficiency and a footprint small enough to fit into the available RAM.
- *Minimal abstraction.* Hardware drivers characterize to have a direct vision to the hardware block, to match the device registers. On the one hand, this gives a total controllability of the hardware block. On the other hand, the API is less isolated from hardware device changes and has to be updated with every hardware change.
- *Small interface.* The design of the bus interface logic has been designed for single read/write transfers of 32 bits. While this might seem to offer a poor performance, it is rather a good approach for the hardware and software design as it is free of other intermediate mechanisms to, e.g., buffer handling packet-based transfers, steering logic for handling diferent transfer sizes, pointers, and so on.

5.5.2 TSU drivers

The TSU drivers are low-level C functions that permit us to read/write from/to the pool of registers of the TSU. As seen in Tables 5.4 and 5.5, in our block we have two types of registers: 18 data registers for storing the last transmitted and received information, and 1 control register for controlling statuses and setting different operation modes of the TSU. The read and write accesses to the registers can be done using specific functions that contain one or more single read and write transfers of 32-bit from and to the register's addresses. The readings and writings are done using specific macros that are included in the standard C libraries and Xilinx C libraries delivered with the BSP, such as `XIo_In32` and `XIo_Out32`. Listings 5.1 and 5.2 show two of the six TSU functions.

```
Xuint32 tsu_read_register(Xuint32 BaseAddr, Xuint32 offset_reg)
{
 /* instr.#1, save space in PPC stack. No PLB access */
 Xuint32 aux;
 /* instr.#2, 1 PLB access to take the value in 'BaseAddr
  + offset_reg */
 aux = XIo_In32(BaseAddr + offset_reg);
 /* instr.#3, move 'aux' to another temporary register.
  No PLB access. */
 return(aux);
}
```

Listing 5.1: Function to read from one of the TSU data and control registers.

```
void tsu_write_register(Xuint32 BaseAddr,
    Xuint32 offset_reg, Xuint32 value)
{
 /* instr.#1, 1 PLB access to store 'value' in 'BaseAddr
  + offset_reg' (in TSU) */
 XIo_Out32(BaseAddr + offset_reg, value);
}
```

Listing 5.2: Function to write to one of the TSU data and control registers.

The TSU drivers are a key design component as they will set the execution time of the instructions in the program code. The execution time of the functions depend on several parameters:

⇒ *Number of input/output parameters.* In a function call, the input parameters are stored in temporary registers within the PPC with which to perform the operations. The more number of parameters, the more assembler instructions for the PPC to store them in the temporary registers. Similarly, the more the number of registers to store the result/s of the function, the more the number of transfers.

⇒ *Bitwise operations.* Some TSU drivers contain logical bitwise operations to write to the control register. As the operations are executed by the microprocessor, the higher the number of single bit operations is, the higher the latency of a TSU driver execution will be. To lower the latency, we have configured the microprocessor to run at the maximum speed of 300 MHz.

⇒ *Number of accesses to external memory.* This is the most critical parameter as the PLB bus is an indeterministic bus and grants peripheral access requests depending on the occupancy. For our TSU drivers which characterize for having few or no arithmetic C instructions, the gross part of the execution time will be given by the number of accesses to the PLB.

Some TSU drivers will contain more low-level C instructions depending on the TSU hardware design. The use of more low-level C instructions and macros imply more accesses to the external memory to fetch and store variables, and therefore more accesses to the PLB by the PPC and more indeterminism in the

program execution time. For the sake of the discussion, consider the TSU function *tsu_read_register* (Listing 5.1). The first instruction (*instr.#1*) tells PPC that has to save 4 bytes in the *stack* region of memory, and thus to update the stack pointer accordingly. *instr.#1* does not need any parameter that requires a PLB access. In the second instruction (*instr.#2*), the PPC has to copy to one of its temporary registers the value (*value*) stored in the address memory *BaseAddr + offset_reg*. This requires at least one PLB access. The third instruction (*instr.#3*) the variable `aux`, stored in a temporary register within the PPC, is transferred to another temporary register as the output argument. It does not require to access to the PLB.

Table 5.7 summarizes the rest of the TSU software drivers, their purpose, number of low-level C instructions and the number of PLB accesses. In the next chapter, we will study the impact of TSU drivers latency on the synchronization accuracy between the *localtime* counter within the TSU and the counter within the PPC.

Table 5.7: Low-level C drivers for read/write access from/to the TSU.

Function Name	No. of C Instructions	PLB accesses	Purpose
tsu_write	1	1	Write TSU register.
tsu_read	3	1	Read TSU register.
assert_control_reg_bit	4	2	Write a logical '1' to one of the control_reg register bit.
deassert_control_reg_bit	4	2	Write a logical '0' to one of the control_reg register bit.
read_cau	5	3	Read TSU's counter.
load_cau	6	4	Write TSU's counter.

5.5.3 Basic Application Interface

One of the advantages of embedded platforms is that applications can be partitioned in independent modules. The goal of application partitioning is to ensure that the resources dedicated to a specific task are available and adequate to meet the timing requirements, while avoiding resource contention. Resource contention lead to indeterministic temporal behavior in the execution time of applications. With careful design of application partitioning, resource contention can be greatly reduced.

To partition the application subtasks, we will follow two well known paradigms in the area of test and measurement and industrial control, *time slicing* and *arming* [Kopetz (1997)]. The time-slicing paradigm consists on providing a common time scale to the modules attached to the communication system to start tasks following a temporal order. On the other hand, arming consists on isolating and reserving hardware resources so that rapid response to external signals (usually

called *capture triggers*) can be achieved. In our platform, these tasks are carried out by the timers.

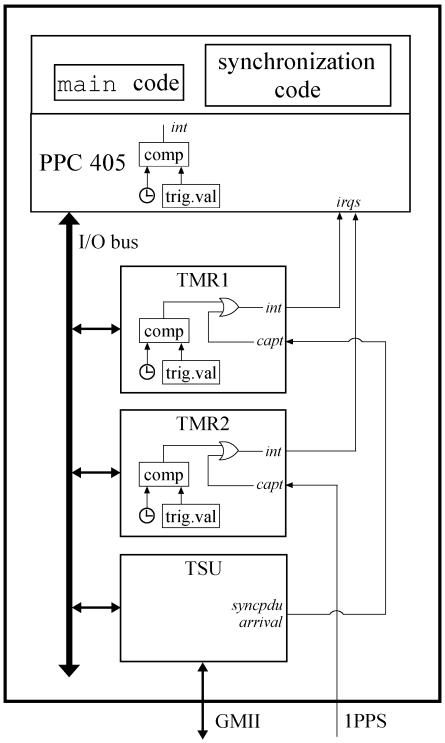

Figure 5.14: Basic application interface (BAI).

Figure 5.14 shows the basic application interface (BAI), which is a blend of the OSI layer model and a simplified architecture version of the synchronization platform seen in Section 5.3.2. BAI illustrates those hardware blocks that directly interact with the application layer, i.e., TMR1 and TMR2. The timers contain two logic blocks: a *capture block* and a *timeout block*. The first one captures external events, such as the logic level of a digital signal. The second block consists on a counter, a trigger value register (*trig. val*) and a comparator. The application code writes the desired interval time into the trigger value register. This value is continually compared to the current time of the counter, and when the two values match, an event signal is generated. This is precisely the task of the three timers, to trigger interruptions through their *int* output upon a *timeout* or a *capture* event. The interruptions are centralized and handled in the PPC. When it detects an interruption, it starts executing a specific software routine.

To associate the hardware event with the software routine, we have considered the number of possible events of the message exchange pattern (see Chapter 4). We have two possible *capture* events, the arrival of a syncPDU and the 1PPS events coming from the GPS input. As shown in Figure 5.14, these events are

wired to the *capt* input of TMR1 and TMR2, respectively. The other kind of events that we need to generate depend on the sending of the different syncPDUs by the node acting as a master (see next subsection). Recalling Figure 4.3 of Chapter 4, we need to send three different syncPDU types: GATE$_{disc}$, GATE and REGISTER. Thus, we allocate them in TMR1, TMR2 and TMR3, respectively. Table 5.5.3 summarizes the association between the timer events and the functionality of the program code.

Table 5.8: Association of the software interruption handlers with BAI's hardware blocks.

	Hardware resource	Configuration	Interval	Purpose
Master	TMR1	Capture	N/A	SyncPDU reception
		Timeout	τ_{GATE}	GATE transmission
	TMR2	Capture	N/A	1PPS event
		Timeout	τ_{resync}	GATE$_{resync}$ transmission
	TMR3	Timeout	τ_{disc}	Start of discovery process
Slave	TMR1	Capture	N/A	SyncPDU reception
	TMR2	Capture	N/A	1PPS event

5.5.4 Application Functionality

Message Exchange Pattern

The main goal of the application is to measure and to adjust the clock offset between each pair of clocks of two nodes (master-slave) in order to synchronize them perfectly. Figure 5.15 shows the timeline of the synchronization message exchange pattern at different block levels of each node. Compared to the synchronization exchange pattern seen in Figure 4.3.3, this includes the software/logical variables (stored in the DDR SDRAM) and the internal registers of the TSU. The inner columns show the instantaneous clock offset between the master and the slave and viceversa. The mid and outer columns denote which hardware registers and logical variables are being written during the process. The whole process is divided into three recursive phases: *normal operation*, *discovery* and *resynchronize* phase. In the normal mode, the master is sending *GATE* messages which are replied by the slave under *REPORT* type messages at intervals given by τ_{GATE}. The purpose of this phase is to check the clock drift of the slave relative to the master. If the master detects a specific clock offset due to the clock drift, it enters into the resynchronization mode to resynchronize the slave. Conversely, if the accumulated clock offset keeps lower than a maximum threshold value, the master enters into the *discovery* mode after a fixed and periodic interval of time given by τ_{disc}.

The RTD calculation is executed at the *discovery* phase. The master starts sending a $GATE_{disc}$ message with its localtime (t_3) and the slave adjusts its *lo-*

caltime register with this value. At this timestamp point, the time offset between the master and the slave is the propagation time, which is the time needed to go across the two PHYs and the cable (t_{prop}=2×t_{PHY}+t_{cable}). After the amount of time given by, t_{wait}=t_{pause_gen}[3] +$t_{ts\_tx\_freeze}$[4] the slave replies sending a *REG_REQ* type synchronization message with its recently synchronized time (t_4). At this moment the master has the capture register filled with the last transmitted and received values (see the left-side mid column). After some microseconds the register values are dumped in the DDR memory and the master starts the calculation of the RTD applying the equation 5.1. Due to the symmetry on the time points along the synchronization path, the second term of the equation, t_{wait}, is equal at both nodes and the master can use its own internal timing metrics ((t_C-t_B)-(t_D-t_C)) to accurately calculate the RTD value (see Section 5.4.2). The discovery phase ends when the master communicates the propagation delay, t_{prop}, (RTD>>1) to the slave in a *REGISTER* type message. The slave stores t_{prop} value in `rtt_reg` to use it in the next normal operation phase to accurately synchronize to the master.

$$\begin{aligned} RTD &= t_{response} - t_{wait} = \\ &= (t_N - t_D) - (t_{pause\_gen} + t_{tx\_ts\_freeze}) \\ &= (t_N - t_D) - (t_J - t_i) - (t_K - t_J) \\ &= (t_N - t_D) - (t_C - t_B) - (t_D - t_C) \end{aligned} \tag{5.1}$$

Of sum importance is the time interval between the last transmitted GATE_{resync} and the first GATE within the normal operation phase. We denote this interval as τ_{obs}. τ_{obs} must be selected such that the accumulated offset (due to the frequency drift) during this interval (t_6-t_5) does not affect the clock reading of the next message, i.e., the GATE send in t_6. If $\tau_{obs} = t_6 - t_5$ is too big, the read offset will be erroneous. Conversely, if τ_{obs} value is small enough such that the accumulated offset does not impair the synchronization accuracy since the last re-synchronization act. The clock reading in the next GATE message will allow us to verify whether the Ethernet jitter is bounded or not within the sum of the jitter of its MAC and PHY layers, i.e., $\pm(\varepsilon_{MAC} + \varepsilon_{PHY})$.

[3] t_{pause_gen} corresponds to the time for the MAC to generate a PAUSE frame, which is equivalent to $(t_J - t_i)$ in the slave, and $(t_C - t_B)$ in the master.

[4] $t_{tx\_ts\_freeze}$ corresponds to the time for the TSU to generate the timestamp to transmit, which lasts from the first PAUSE byte to the 16^{th} syncPDU's byte. It is equivalent to $(t_K - t_J)$ in the slave, or $(t_D - t_C)$ in the master.

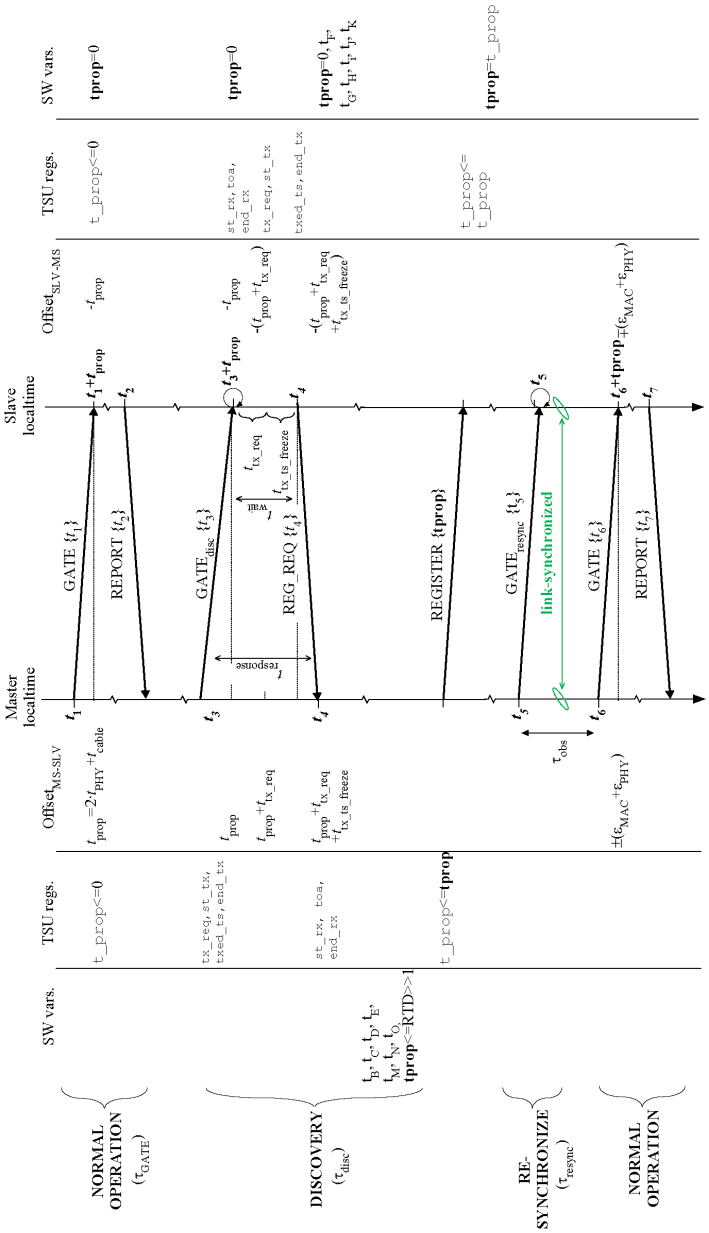

Figure 5.15: Message exchange pattern.

In the next chapter, we will measure the frequency drift between the master and slave, and we will accurately select the optimal τ_{obs} interval. Next, an explanation of the synchronization code functionality follows.

Master Events

As seen in the message exchange pattern, the master orchestrates the whole synchronization mechanism with the transmission of syncPDUs. The timer events that trigger the message transmission are the timeout events (τ_{GATE}, τ_{resync}, τ_{disc} and τ_{obs}), while the arrivals of syncPDUs and 1PPS events are handled by the capture modules. Figure 5.16, 5.17 and 5.18 show the software routines executed by the PPC out upon a timer event. Each routine, except from PPC's internal timer, hosts two events, either a timeout or a capture. In Figure 5.16, there are shown the subtasks of TMR1 routine upon the transmission of a GATE message and the arrival of a syncPDU. To send a GATE message, the *Oper* field of the syncPDU must be loaded with the operand GATE, and then trigger a PAUSE message. The TSU inserts the *Oper* field (*syncPDU pre-loading*) and replaces the rest of the fields. The distributed timestamps collected when the syncPDU is being transmitted are also automatically stored in their respective addresses of the data register block. Under the arrival of a syncPDU, the PPC starts dumping the data registers into the DDR memory (*save TSU registers*). Finally, for both events, the interruption flags are restored.

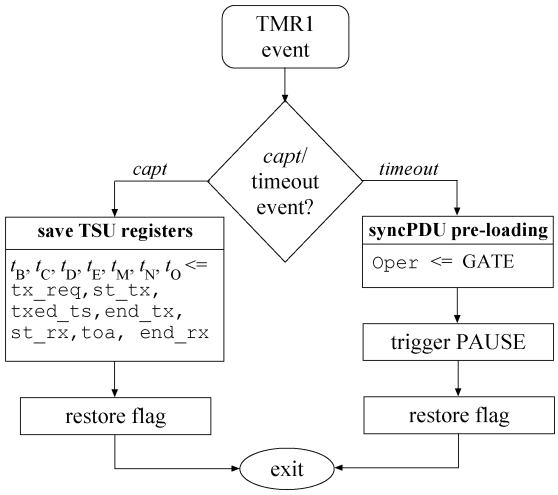

Figure 5.16: TMR1 event handling (Master). GATE message transmission (at intervals of τ_{GATE}), and syncPDU arrival.

In Figure 5.17 it is shown the software routine of TMR2. The timeout handles the transmission of a GATE_{resync} following a similar process to TMR1. Upon a 1PPS event, the PPC moves the content of *one_pps* register to the DDR memory. The response time of the PPC to read *one_pps* register do not affect its value, as it is stored by the hardware mechanisms within the CAU. The subtasks under the timeout event are slightly different from the previous figure. Here, the PPC first stores GATE_{resync} operand in syncPDU's Oper field and triggers a PAUSE

message. Immediately after an interval of τ_{obs}, it issues the transmission of a GATE message to observe the clock offset.

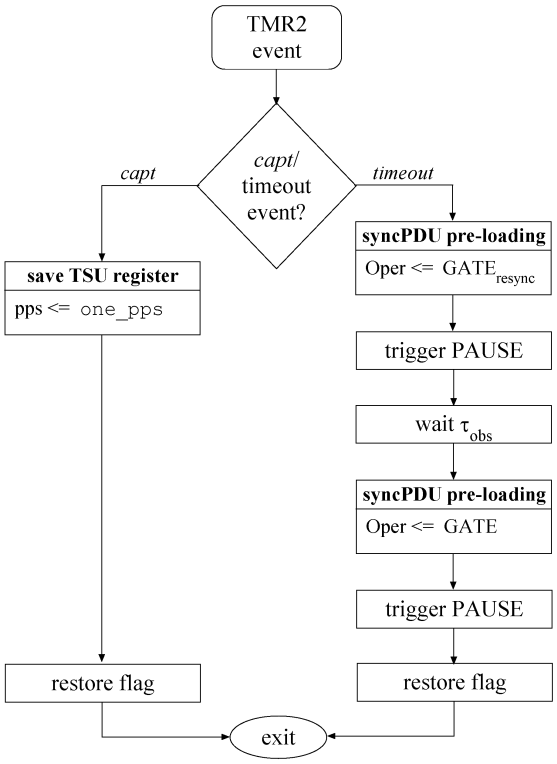

Figure 5.17: TMR2 event handling (Master). GATE_{sync} message transmission (at intervals of τ_{resync}), and 1PPS events.

The software routine executed in Figure 5.18 is triggered by a PPC's timeout event. Its functionality is to start the discovery process with which to calculate the RTD and the propagation time, and communicating it to the link partner. The procedure has similar and different subtasks, as in TMR1 and TMR2 subroutines. Similarities include the *syncPDU pre-loading 1* and PAUSE triggering, while the main difference is on the way the master communicates the propagation time to the slave (*syncPDU pre-loading 2*). Here, the TSU is set to *offset transmission mode* to inform the slave that the content of *Timestamp* field is not the master's localtime, but the propagation time. Then, before triggering the PAUSE (*trigger PAUSE 2*), the *Oper* and *Timestamp* fields are pre-loaded with REGISTER and the propagation time, respectively. The routine also concludes with the restoration of the interruption flag.

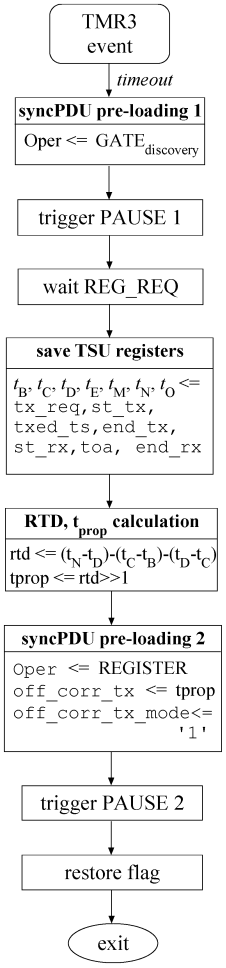

Figure 5.18: TMR3 event handling (Master). Start of the discovery process (at intervals of τ_{disc}).

Slave Events

The role of the slave compared to the master is simpler as their timers have been configured without the timeout events. Slaves timers detect incoming syncPDUs and log 1PPS events. In Figure 5.19, it is shown the software routine of TMR1, which triggers specific instructions depending on the type of received syncPDU. When the slave detects that the *Oper* field of the incoming syncPDU contains either a GATE_{resync} or a GATE_{disc}, it directly restores the interruption flag and exits the subroutine. In these two cases, the PPC does not have to interact with the TSU as the operations are performed automatically by hardware.

When the *Oper* field contains a GATE operand, the PPC preset the TSU for a reply using a REPORT message. The PPC first load the *Oper* field and

trigger a PAUSE. Thereafter, it dumps to the external DDR the contents of the data registers and exits the subroutine after restoring the flag. Lastly, when a REGISTER is received, the content within the *Timestamp* field is the propagation time and it is stored in `t_prop` register. Finally, the interruption flag is restored

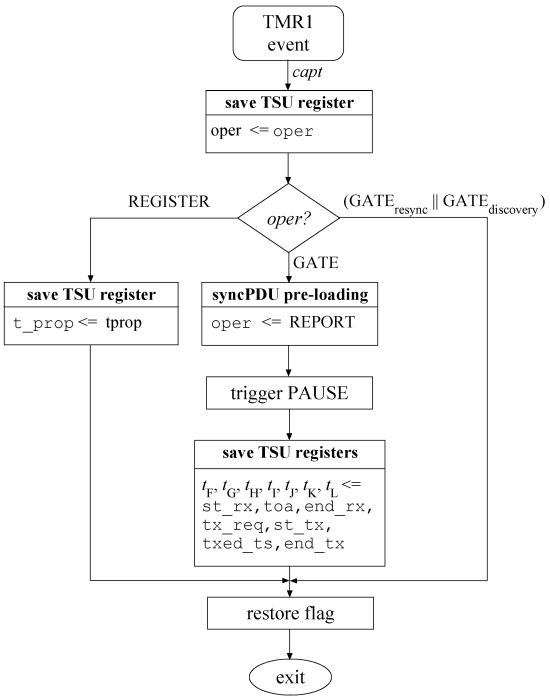

Figure 5.19: TMR1 event handling (Slave).

The subroutine executed by TMR2, in Figure 5.20, is committed to capture the 1PPS events and store the slave's localtime value in the DDR memory.

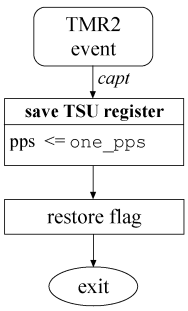

Figure 5.20: TMR2 event handling (Slave). Subroutine for storing 1PPS events

5.5.5 Memory Allocation

The wide range of different memory technologies in today's FPGAs offers the possibility to embedded processor designs of optimizing to the limit the execution speeds of application programs. On the one hand, memory can be located on the same physical chip as the processor (on-chip) to increase performance. On the other hand, in order to improve flexibility and accommodate big applications, designers can opt to attach off-chip memory modules that can vary in size and speed. Memory situated on the same chip as the processor often has a small capacity because of limited available chip area. On-chip memories may be sufficient for small applications, but are rarely large enough to support large applications.

Our FPGA has 64 KByte of BRAM along the die that can be used for storing, e.g., code portions requiring fast accesses by the microprocessor. The board also mounts two off-chip kind of memories: 1.152 MByte of ZBT SRAM and 64 MByte of DDR SDRAM on-board. ZBT SRAM characterizes for supporting 166 to 225 MHz bus operations with zero wait states, i.e., no bus latencies, while the DDR SRAM supports bus operations up to 266 MHz with latencies ranging from 1.5 to 3 ns. On top of the memory latency, there is also the latency and jitter added by the arbitration protocol of the PLB bus, which is variable and depends on transient occupancy.

For rapid accesses to memory, BRAM memory on-chip is best suited for accomodating either the execution instructions within the application program, or the data that needs to be rapidly accessed. Our FPGA development software tools reported that the amount of memory needed to host the program application was 60.1 kByte, which is less than the overall BRAM on-chip. Although it may seem plausible to allocate the program code in the BRAM memory, the FPGA design tools demand the use of BRAM for logic support in other IP blocks. Therefore, we are forced to use the external DDR SDRAM to host the program code, data and temporary data. Under this situation, we face the latency and undeterminism of the PLB bus and the logic devices inside the DDR SDRAM chip. In the next chapter, we evaluate the effect of these sources of undeterminism on the synchronization between the TSU's counter (*localtime*) and the PPC's counter.

5.6 Conclusions

In this chapter, we have covered an important part of this Thesis, which is the implementation of the platform for the evaluation of the time synchronization accuracy at Layer 2 of Ethernet technology. The approach followed in this Thesis is to maintain the initial "asynchronicity" of Ethernet, i.e., to keep the low-level re-synchronization mechanisms of Layer 1 and to accommodate at Layer 2 the jitter introduced by Layer 1.

To obtain high-precision jitter measurements of Ethernet layers, we have used a 32-bit digital counter running at 300 MHz, the maximum frequency of this FPGA model. The timestamps have then a granularity of $3.\widehat{33}$ *ns*. This design requirement has lead to another design challenge: the presence of circuit portions running at different frequencies and the interaction between them. The

GMII interface of Gigabit Ethernet interfaces work at 125 MHz, while the CAU's *localtime* frequency is 300 MHz. This frequency mismatch can impact on the reliability of the timestamps used for the evaluation. The need for a reliable frequency interoperation has lead us to provide another contribution, which is the introduction of a mechanism that allows to synchronize the three frequencies and to obtain timestamps reliably.

Another hurdle we have encountered in the design process has been on meeting the timing requirements. In FPGA design, and in hardware design too, there is a paradox as far as the timing closure is concerned. On the one hand, less is more, i.e., the less programmable resources used to implement a function, the less the routing congestion and the more probability for the FPGA development tools to successfully map the components and place and route the interconnects. On the other hand, few used resources can lead to the circuitry to be more dispersed, long tracks to interconnect them, and thus long transmission delays, less working frequency and performance. To meet the timing, we have placed and routed manually those components demanding high frequency, such as the timestamp counter (*localtime*).

The last conclusion note regarding to the physical design is on the design overload. If unnecessary hardware blocks are added, the FPGA tools will likely not be able to fit them within the programmable resources. Moreover, in the supposed case that the tools can allocate the design within the programmable area, it is not sure that the intended functionality of the design is met. For a successful functionality of the overall platform, a careful study of the application requirements and the FPGA architecture is needed.

In this chapter, the architecture presented in Chapter 5 and the synchronization mechanism proposed in Chapter 4 is evaluated. Since a meaningful evaluation of a hardware system can only be performed under consideration of the real platform to be used, a short description of it is presented. Subsequently, the tests to individually measure the jitter and latency of the MAC and the PHY will be explained. Special emphasis is put on the time synchronization accuracy evaluation. The results drawn from this experiment permits us to verify and meet the main goal in this Thesis. We further evaluate relevant synchronization components and the conclusions that can be derived thereof. A short summary of results concludes the chapter.

6.1 Criterias, Methods and Goals

In this Thesis, a pure FPGA-based prototype (synthesis and place and route) has been chosen as method to evaluate the latency and jitter, and the synchronization achieved at Layer 2. Although designing hardware is a very burdensome and complex task, pure prototyping is preferrable compared to other evaluation techniques, e.g. system emulation or pure simulation. A real prototype allows to recreate the technological limitations surrounding the system incurred in the real-life devices, especially those related with the physical limitations of the circuit. For example, the maximum system clock frequency, which influences performance by setting a limit on the cycle duration of the program, or the maximum clock frequency of the TSU counter, which set the granularity of the timestamps. Physical resource usage is another such case: The exact memory footprint, used programmable resources, etc, depends on the FPGA used. In order to use the logic resources efficiently, or even simpler, to fit the hardware functionality into a specific FPGA, the design has to be synthesized and placed and routed taking into account the exact architecture and components (FPGA, memory banks, busses, etc.).

As presented in Section 5, the TSU was designed to fulfill a number of criteria. First and foremost, how close two high-speed remote counters are to each other after a re-synchronization event. This gives us the level of synchronization achieved. To follow this evaluation criteria, the TSU has to provide high timestamp granularity to be able to measure with enough resolution the jitter and latency introduced by Ethernet layers. However, the resolution of the times-

tamps is limited by the physical resource allocation during the hardware design process. This fact brings an added uncertainty in the assessment process.

To verify the goal explained in Section 4 at the nanosecond level using low-cost crystal oscillators, their frequency drift must be taken into account. To understand well the implications of the clock drift on the synchronization evaluation, consider the nominal drift of a standard crystal oscillator technology which is known to be around 100 ppm (the worst case possible). Every 1 s, the frequency deviates 1 μs, or in a smaller scale, every 1 ms the frequency deviates 100 ns. To observe the accumulated error due to clock drift over different intervals, see Table 6.1. The significance of the error rates impacts on the observations made on the synchronization accuracy. As we are addressing nanosecond accuracies, we must provide reliable methods capable to get rid of the influence of the drift.

Table 6.1: Absolute error at different error rates and intervals.

Interval Duration	Error rate, PPM	
	50	100
1 day	4.32 s	8.64 s
1 min	3 ms	6 ms
1 s	50 μs	100 μs
1 ms	50 ns	100 ns
0.1 ms	5 ns	10 ns
10 μs	0.5 ns	1 ns
1 μs	0.05 ns	0.1 ns

We have taken into account all these considerations and, in the next sections, we propose a set of experiments and methods to correctly assess critical parameters for time synchronization, such as latency and jitter, with a timestamp resolution boundary set by the platform technology.

6.2 Evaluation Setup Description

In this section we characterize the overall synchronization platform we have used for the presented experiments. Before that, we identify key parameters that need to be addressed and then the measurement results with a proper discussion.

6.2.1 Hardware System

Figure 6.1 illustrates the measurement hardware setup used for the presented experiments. It consists of two sites equipped with a measurement PC and the FPGA. Each measurement PC is connected to its respective FPGA through the JTAG interface and the USB. FPGAs are connected through a 2 meter CAT-5 crossed cable, as well as both PCs (not shown). The station on the left is acting as a *master* and the one on the right as a *slave*. Both FPGAs receive 1 PPS signal from a GPS receiver [Trimble (2010)] through a 10 meter long cable. Both TSU's PIT of each FPGA are connected to one oscilloscope's channels. Setup and

measurement recording are controlled from and centralized at the master node. The station acting as a slave also serves as data repository for the accumulated data.

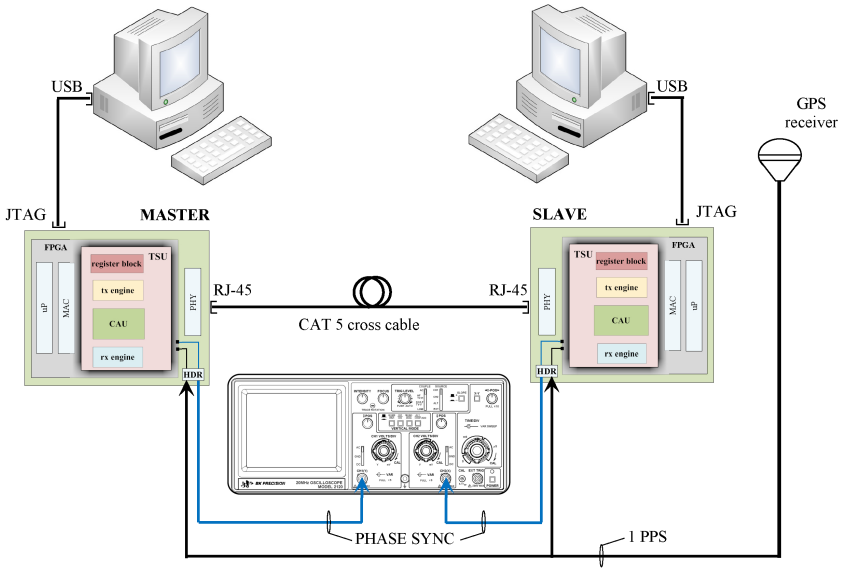

Figure 6.1: Synchronization platform setup.

6.2.2 Software System

The software used for evaluating the components consists of:

⇒ *FPGA vendor tools*, which consist of the software packages to develop applications with the FPGA embedded platform, both for the hardware portion (i.e., with HDL language) and the software/application portion (with e.g., ANSI C).

⇒ *Setup scripts*, which are custom instruction executable files centralized and triggered from the master station. As the FPGA vendor tools run under Windows operating system, the setup scripts consist on standard batch scripts. They configure each FPGA with the bitstream configuration, i.e., the bitfile for hardware reconfiguration and the executable file that contains the application for the embedded microprocessor, and setup a text file to write the stored data of the FPGA memory.

⇒ *Statistics and processing scripts*, which are custom scripts that process the ASCII files after each particular evaluation. The ASCII files contain the time traces of the sync PDUs.

6.3 Synchronization Evaluation Components

This Section is devoted to evaluate different components of synchronization using the hardware setup. We follow an incremental approach to gradually assess the components: First, we measure what is the timestamp granularity of both TSUs and the mutual clock drift between the two boards. Once known, they allow us to translate the measurements from TSU clock cycles to nanoseconds with confidence. After that, latency and jitter measurements of the MAC and the PHY follow. Next, we evaluate the time and phase synchronization accuracy achieved in a point-to-point configuration and provide reasoning from the results obtained. We conduct experiments to observe the impact of the latency and variability of the busses and memory and also compare the robustness of our timestamping mechanism against reading errors with a system lacking protection mechanisms.

6.3.1 Clock Frequency and Drift

Knowledge of the frequency of the TSU's counter is fundamental for many reasons. First, it allows to determine the timestamp granularity in units of seconds, which is the basic measurement unit to assess the synchronization accuracy, latency and jitter. Second, it permits us to properly setup the software part running in the microprocessor for the next experiments. To measure each individual frequency, we have used the 1PPS signal of a GPS receiver [Trimble (2010)] and have wired it directly to the ML403 board headers. Once the GPS receiver is correctly locked to the satellites, it delivers each second a high stable electrical signal to the *enable* input of `one_pps` event registers of each TSU. `one_pps` event register captures the number of clock cycles spanned every second by each TSU. Table 6.2 the number of clock cycles accumulated by each counter at every second over an interval of one day. The column on the right shows the number of clock cycles and the right column the percentage over the whole experiment. The 98% of the values of the master and the slave accumulate 3×10^8 clock cycles/s, which means that they are both running at 300 MHz. The remaining measured percentage, <2%, was accumulating 6×10^8 clock cycles/s, and thus it is not meaningful. This small percentage of outliers might be caused by a spurious loss of signal of the GPS receivers. In such a case, the GPS receiver misses one 1PPS pulse, and `one_pps` registers accumulate a number of clock cycles equivalent to 2 seconds.

Table 6.2: Number of clock cycles spanned by `one_pps` event registers at every second, and percentage over one day.

	No. clock cycles/s	Percentage [%]
Master	3×10^8	98.45
	6×10^8	1.55
Slave	3×10^8	98.17
	6×10^8	1.83

Another component that strongly impacts on the synchronization accuracy is the crystal clock drift or the progressive frequency deviation that each counter suffers from. For us, knowledge of the crystal drift is of most importance as it will provide a bound of the accumulated frequency offset error in the evaluation of the time synchronization accuracy. To measure the clock drift, we have used the same setup for determining the clock frequency and the traces collected at each station. We have compared and subtracted each 1 PPS event to observe the mutual drift of the clocks, i.e., slave's clock drift relative to the master's. Using our measurement setup, we do not have means to determine each individual crystal, but only the mutual clock drift. Figures 6.2a and b show the mutual clock drift of the master relative to the slave over the period of one day (86400 s) in TSU clock cycles and seconds, respectively. After one day they drift apart 258.8×10^6 TSU clock cycles. Considering that each clock cycle has a period of $3.\widehat{33}$ ns, over the period of 1 day, they drift apart 862 ms, which means a mutual clock drift of 10 $\mu s/s$. The negative slope of the accumulated clock offset reveals that slave's oscillator runs faster than master's. From the Figures 6.2a and b, we have derived the Table 6.3, which summarizes the accumulated clock offset due to frequency drift over different time intervals of interest. These values will allow us to select time intervals in which the accumulated clock offset error is not significative.

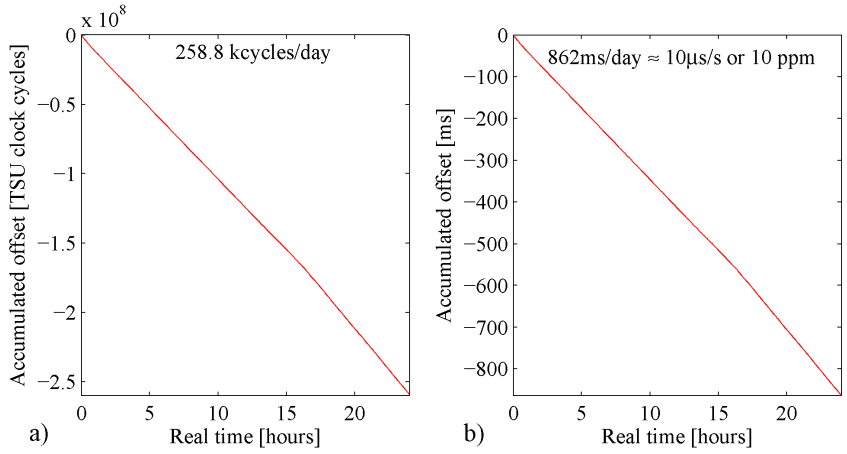

Figure 6.2: Relative clock drift, expressed in TSU clock cycles (a)) and in ms (b)).

Table 6.3: Accumulated clock offset at the rate of 10 ppm.

Interval Duration	Relative error
1 day	862 ms
1 min	598.6 μs
1 s	114.4 μs
100 ms	11.4 μs
10 ms	1.14 μs
1 ms	114 ns
100 μs	11.4 ns
10 μs	1.14 ns
5 μs	0.57 ns

6.3.2 Timestamping Reliability

As explained in Section 5.4.4, the cross domain clock scenario of the TSU can lead to erroneous timestamps that can invalidate the evaluation process. To observe how harmful the metastability phenomenon can be on the timestamps, we have undergone a very short time synchronization experiment which consist on comparing the remote clock offset with and without the *flancter-flags*[1]. In absence of timestamping errors, the remote clock offset should follow a sawtooth shape bounded in the region between 0 to $\sim 10^4$ TSU clock cycles (see Section 6.3.4). In Figure 6.3, it is shown the setup used to evaluate the timestamping reliability. The pair of distributed timestamps used are t_D, at the master, and t_G, at the slave. Although we could use whichever of the pairs of distributed timestamps, we choose these ones as they correspond to the timestamp carried in the packet (t_D) and time of arrival (t_G). In this experiment we do not use any background data to isolate possible sources of additional errors and focus exclusively on the performance of the protection mechanism.

From the plots in Figure 6.4, we observe that some clock offset values fall beyond the region close to zero (i.e., the peaks of '...' blue line). To understand this situation, consider the timestamp pair number 31, which has a magnitude of 4×10^9 and negative sign. Considering its magnitude and the width of the TSU counter (32 bits), the erroneous captured bits are within the uppermost nibble of the counter, i.e., from bit 28 to 31. From the sign, we infer that the wrong timestamp is t_G. Contrarily, non-errored timestamps (in '- -' red line) fall within the region close to zero.

[1] In this implementation, we were forced to change the architecture of the TSU and the FPGA design tools reported a TSU counter frequency of less than 300 MHz. However, this change is not relevant for the evaluation of the timestamping reliability.

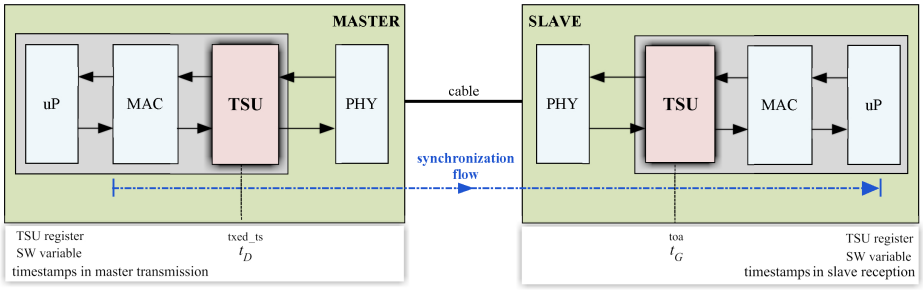

Figure 6.3: Platform setup for testing the timestamping reliability.

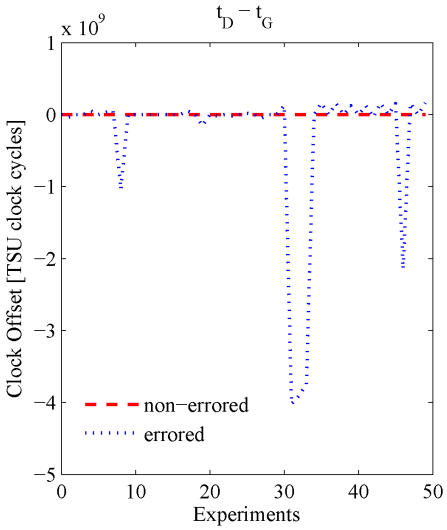

Figure 6.4: Effect on timestamping reliability with and without flancter-flags.

6.3.3 Internode Jitter

One of the objectives of this Thesis is to measure with the highest precision as possible the jitter introduced in the synchronization messages by the analog and digital logic devices within the MAC and the PHY. Actual techniques for measuring the latency of digital logic imply to have direct access to the pins of the chip to measure time intervals of interest, e.g., the period of time that lasts from the rising edge of `TX_EN` (at the sender) to the rising edge of `RX_DV` (at the receiver). These practices require the use of expensive instrumentation, such as high resolution event counters, and setups valid only in laboratory environments.

In this work, we avoid the use of expensive instrumentation and complex measurement setups. Instead, we use the same platform and the distributed timestamp architecture to derive high-precision ingress and egress frame interarrival times. Next, we explain the conducted experiments to derive the jitter introduced

by each Ethernet layer. Later, we use the results to verify the effectiveness of our time synchronization mechanism.

MAC latency and jitter measurement

The latency and the jitter introduced by the MAC of the chip is the first component to be measured to later verify the synchronization accuracy at Layer 2. In this experiment, we aim at observing the latency and variability that a MAC with a time synchronization functionality might need to generate a syncPDU. In our implementation, the MAC latency corresponds to the time interval needed by the internal digital logic to generate a PAUSE frame, while its jitter corresponds to the variation of such time interval. In order to observe the influence of the internal MAC buffers, we have set different load scenarios and captured the time series of the different distributed timestamp pairs (see Figure 6.5). Both nodes are transmitting back-to-back syncPDUs (*synchronization flow*) with and without background data (*data flow*). The chosen sizes of the background data are 64 bytes and 1500 bytes. The timestamp pairs that correspond to MAC latency measurement are (t_C,t_B), at the master side, and (t_J,t_i), at the slave side.

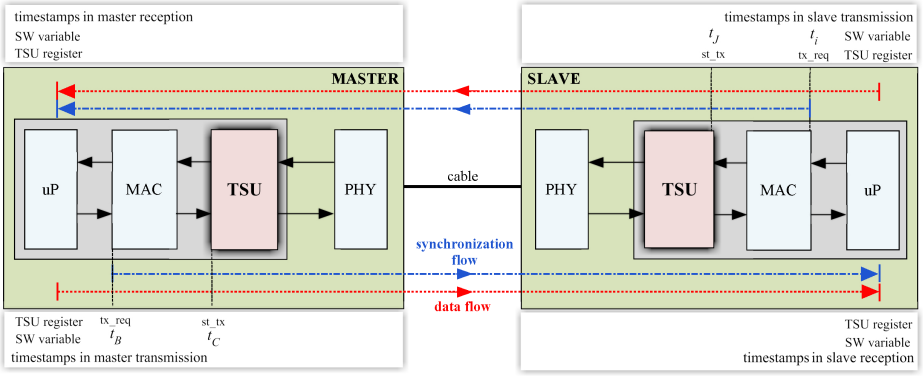

Figure 6.5: MAC latency measurement setup.

The experiments consists on sending 200k syncPDUs for each of the three load scenarios. The statistics of the experiment are summarized in Table 6.4. From the results, we observe that the latency is well confined over 2 intervals of 1 TSU clock cycle, or $3.\widehat{33}$ *ns* (the timestamp maximum resolution). The latency of the MAC to generate a PAUSE control frame is well bounded in the interval of 38 to 39 TSU clock cycles (126.7 and 130 *ns*, respectively). From the Uniform distribution of the intervals, we can deduce that the latency of the internal mechanisms that generate the control PDU is nearly constant and does not depend on the load conditions.

Table 6.4: Latency and jitter measurements of the Xilinx MAC core (expressed in number of TSU clock cycles, nanoseconds and percentage over 200k samples).

	Master (t_C-t_B)		Slave (t_J-t_i)	
	latency	jitter	latency	jitter
no load	38, 126.67 (50.1%) 39, 130 (49.9%)	1, 3.33	38, 126.67 (64.4%) 39, 130 (35.6%)	1, 3.33
w/64B data	38, 126.67 (51.1%) 39, 130 (48.9%)		38, 126.67 (61.7%) 39, 130 (38.3%)	
w/1500B data	38, 126.67 (51.9%) 39, 130 (48.1%)		38, 126.67 (62.5%) 39, 130 (37.5%)	

PHY latency and jitter measurement

The latency and the jitter introduced by the complex physical layer internal devices is the last component to be measured. The results from this experiment help us to better granularize and verify the RTD during the time synchronization process (see Section 6.3.4). The ML403 board mounts a Marvell 88E1111 10/100/1000 Mbps tri-speed transceiver that can be configured to work at whichever of the three speeds. We only conduct the experiments under gigabit speed.

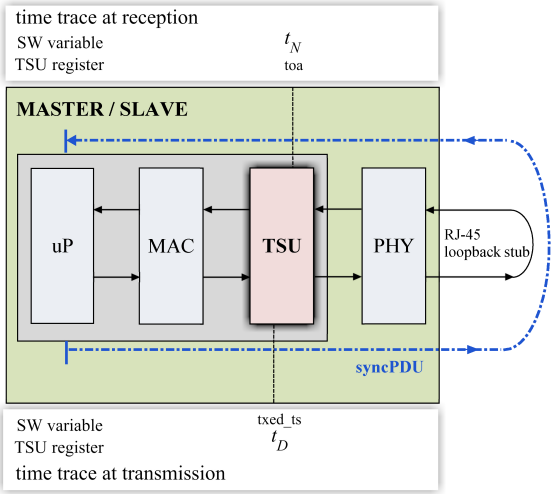

Figure 6.6: Platform setup for PHY latency and jitter measurement.

In order to measure the RTD without the influence of the cable delay, we have manually wired a RJ-45 connector in loopback mode and connected it directly to the RJ-45 PCB board's connector (stub). As shown in Figure 6.6, to perform the RTD measurements we only need one board at a time, the RJ-45 stub and a special program to trigger back-to-back syncPDUs at periodic intervals. The stub bounces back the syncPDUs and the synthetic program process the stored distributed timestamps, i.e., t_N and t_D at the master (and t_G and t_K at the

slave). Before start sending the syncPDUs, the PHY has to be configured in "external loopback mode" to avoid its internal logic detecting near end crosstalk (NEXT) interferences. The program calculates the RTD as the arrival timestamp minus the sending timestamp, i.e., $RTD_{master} = t_N - t_D$ in the master, and $RTD_{slave} = t_G - t_K$ in the slave. In this experiment we do not use background data as it does not influences the PHY processing.

Table 6.5 summarizes the results of the experiment. From the results, we observe that RTDs are randomly spread over 5 intervals of 1 TSU clock cycle or $3.\widehat{33}$ *ns*. The number of intervals depends on the PHY vendor, model and working configuration. The round trip jitter (RTJ) added by PHY is 4 TSU clock cycles or 13.33 *ns*. The percentages are similar to a Normal distribution, a fact that induces us to think that the delay and jitter introduced by the internal devices of the PHY.

Table 6.5: Round trip delay (RTD) and jitter (RTJ) measurements of 88E1111 Marvell's PHY (expressed in number of TSU clock cycles, nanoseconds and percentage over 200k samples).

RTD		RTJ	
Master	Slave	Master	Slave
101, 336.67 (0.2%)	101, 336.67 (0.8%)	4, 13.33	
102, 340 (19.3%)	102, 340 (24%)		
103, 343.33 (41.7%)	103, 343.33 (41.6%)		
104, 346.67 (36%)	104, 346.67 (31.6%)		
105, 350 (2.8%)	105, 350 (2.1%)		

6.3.4 Phase and Time Synchronization Accuracy

The experiments carried out in this section permit us to verify a major goal of this Thesis, which is to prove that the optimal time synchronization accuracy at Layer 2 could be within the jitter bounds introduced by Ethernet layers. This objective is technically viable if an optimal synchronization method is defined. However, the synchronization accuracy will be likely limited by the physical implementation of the hardware design.

Before the evaluation of time synchronization, we perform an experiment to see what is the improvement on the RTD calculation. To evaluate the time synchronization in a realistic scenario, we will make use of the message exchange pattern and leverage our distributed timestamp architecture to derive precise RTD calculations. Moreover, we will conduct a phase synchronization experiment to further reinforce the benefits of having a time synchronization function in the MAC. Phase synchronization method characterizes for being independent of the clock drift.

Precise Round Trip Delay Calculation

An optimal RTD calculation permits to cancel the fixed term of the propagation time, and thus to bound the clock offset within the jitter range. As seen in Section 5.4.2, the processing time for the logic of the receiver to generate a reply message adds an error to the final RTD value. To cancel this processing time and to obtain more precise RTD calculations, we have leveraged the distributed timestamp architecture and have applied Equation 5.1 (see Section 5).

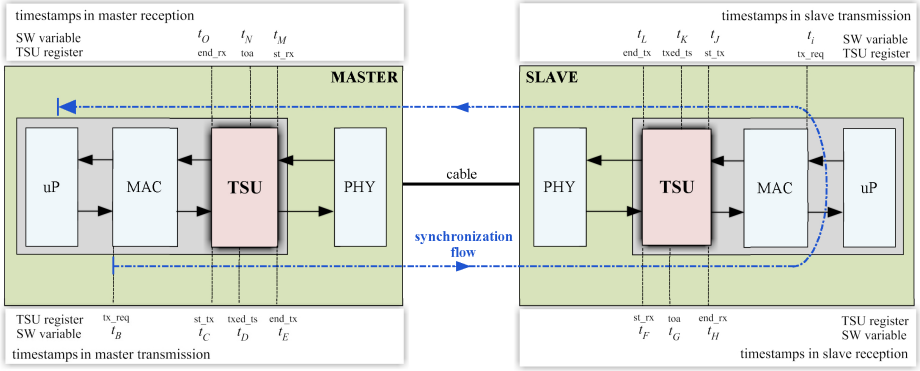

Figure 6.7: Platform setup for RTD evaluation.

To observe the refinement on the RTD calculation, we have used the setup of Figure 6.7 and have configured the master node to send back-to-back GATE_{disc} messages. As explained in the previous chapter, GATE_{disc} messages are automatically bounced and more precise RTDs can be obtained. The traces with the distributed timestamps are collected by the master for post-processing. We have undergone an experiment consisting on a set of 100k samples and evaluated the Equation 5.1 to observe the improvement on the RTD value. The left column of Figure 6.8 shows the time series of different parts of Equation 5.1, and on the right column there are their respective distributions. Subplots a) and b) denote the RTD calculation without means for compensating the processing time of an incoming syncPDU ($t_N - t_D$). Its mean (μ) is 1470.21 *ns*. Subplots c) and d) show that the error component introduced in the RTD formula, $(t_C - t_B) - (t_D - t_C)$, has a mean of 235.6 *ns*. Subplots e) and f) show the achieved improvement on the RTD calculation when cancelling the error part. From the mean value, we can well approximate that the latency to go across the stack is $\sim \frac{1221.78}{2} = 610$ *ns*. From the "Normal shapes" of the subplots b, d, f, we infer that the jitter added by digital logic follows a random behavior. The variability obtained here, 3.18 $ns < \sigma_{RTD_{prec}} < 4.43$ *ns*, is associated with the timestamps triggers, which is around 1 TSU clock cycle. In other words, the distributed timestamps can be captured with 1 TSU clock cycle variability.

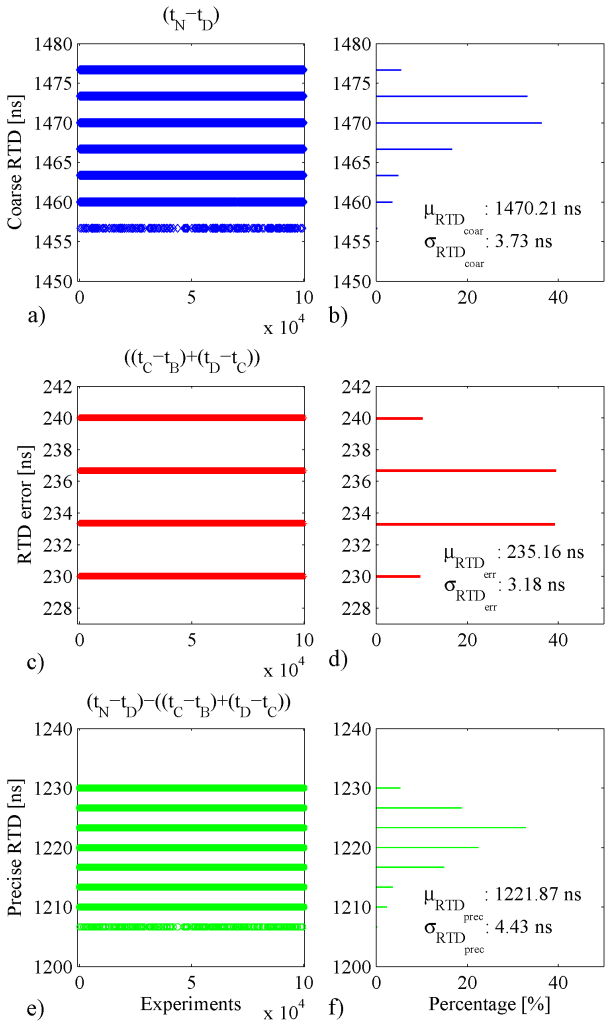

Figure 6.8: RTD calculation. RTD with no internal processing compensation (a,b)). Internal processing time (RTD error) (c,d)). RTD with internal processing compensation (e,f)).

Time Synchronization

The method used to evaluate the achieved time synchronization accuracy combines the message exchange pattern defined in EPON (recall Section 4) with distributed timestamps collected during the message journeys of synchronization messages. As stated in Section 4.1, our goal is to synchronize with the best accuracy as possible accommodating the jitter introduced by Layers 2 and 1. The fixed propagation delay value (i.e., the latency) introduced by the intermediate

layers from the master to the slave is cancelled using the RTD measurements calculated in the message exchange pattern. The criteria we use to evaluate the achieved accuracy is to observe how close to zero are the clock offset values of several distributed timestamp pairs just after the re-sychronization act. To verify it, we only consider the timestamps of the transmission path, i.e., from t_B to t_H. The reason for that is because the reply of the slave after the re-synchronization is done after some tenths of microseconds, and thus the timestamps from t_I to t_O will have the offset error accumulated during this interval.

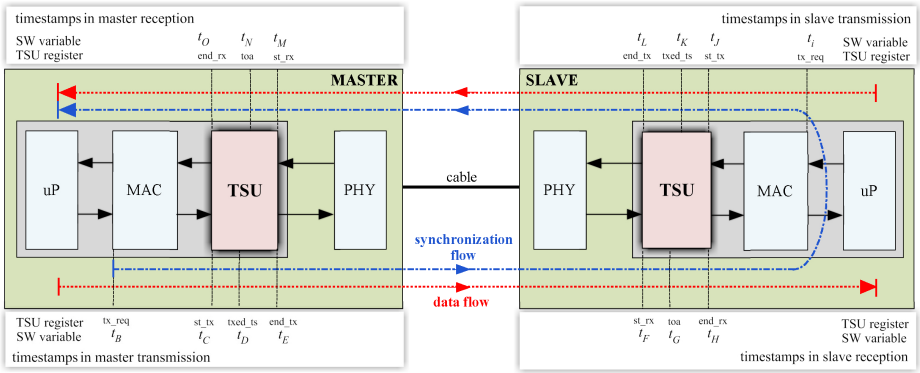

Figure 6.9: Setup used for the evaluation of time synchronization.

Figure 6.9 shows the timestamps collected by each TSU during a sync PDU journey (*synchronization flow*). The location points of the distributed timestamp pairs are chosen in the journey trip of a synchronization message in the way that they keep a symmetry in the ingress and egress paths. The exact triggering points have been carefully chosen in order to calculate more accurate RTDs. The variables on the top of Figure 6.9 refer to the software variables assigned by the software layer that are stored in the SDRAM memory after transmission/reception. The lower ones map the hardware registers of the TSU.

To observe the clock offset, we substract pairs of symmetric distributed timestamps during the *normal operation* phase of the synchronization messages exchange pattern (recall Figure 5.15). In order to correctly assess the achieved accuracy using the message exchange, we observe the clock offset just after the re-synchronization act, namely, without the influence of the clock drift. To know the interval between the re-synchronization and the first message asking to read the remote counter (τ_{obs}), we have used the Table 6.3. According to the table, the accumulated offset due to the mutual clock drift in an interval of 10 μs is 1.14 ns, which is one third of the timestamp granularity. Therefore it is a good approximation to set $\tau_{obs} = 10\ \mu s$.

To observe if the control frame generation logic also adds more jitter, we have executed the message exchange pattern under three different load scenarios (*data flow*): no background data (no bg data), with minimum-length data packets (w/64B pckts) and maximum-length data packets (w/1500B pckts).

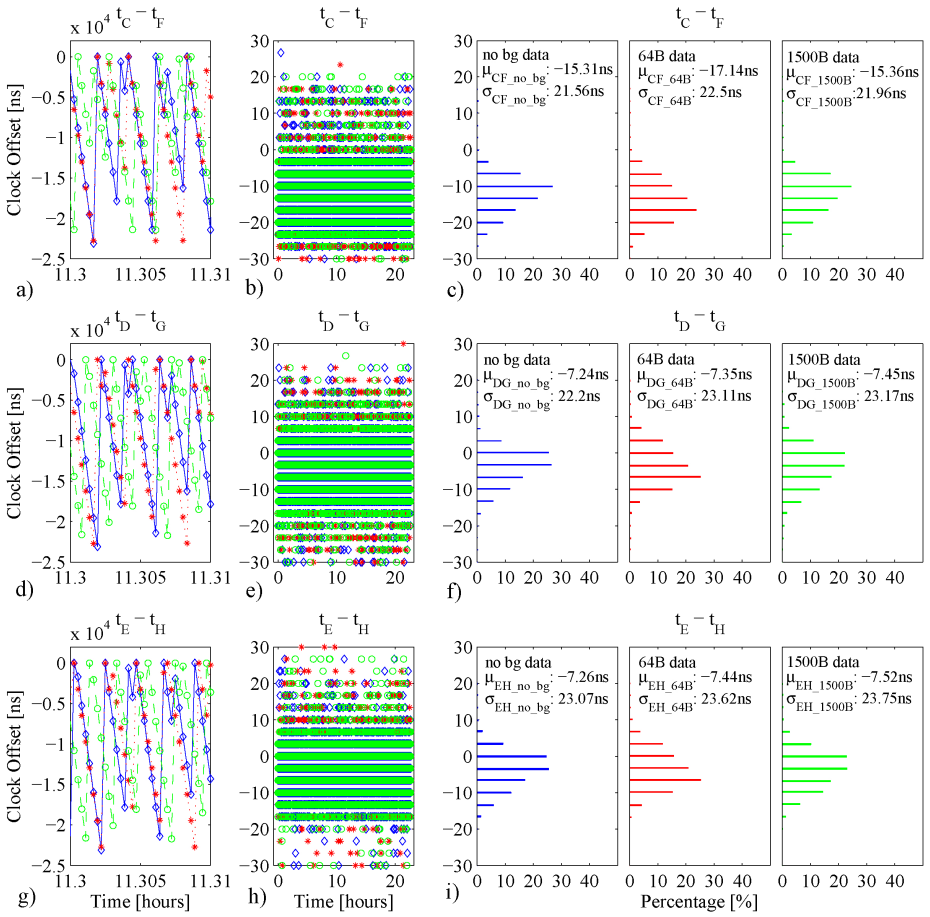

Figure 6.10: Evaluation of the synchronization accuracy (24 hour experiments). a, d, g) Running clock offset (in ∼36 *s.* window). b, e, h) Clock offset series after the re-synchronization. c, f, i) Distribution of plots b, e, h, for a each load scenario.

The subplots of the Figure 6.10 provide information about the remote clock offset in the path from the master to the slave after the re-synchronization act. The first row refers to the distributed timestamps pair t_C and t_F, the second row to t_D and t_G, and the third row to t_E and t_H. The first column of subplots shows the clock offset of the three distributed timestamp pairs over a small interval window (for better clarity). The sawtooth shape of the clock offset (in the first column) shows the synchronization process, where each "falling edge" to zero axis represents a re-synchronization act. Re-synchronization period should exactly be 7.5 *s.* However, in some cases it is smaller. This can be noted from the "non-uniformity" of the sawtooth plots. This behavior owes to the fact that, under

some circumstances, there are multiple timeout event interruptions pending to be served. The timestamp pairs that cause the falling edges are captured in the routine executed upon the timeout event of TMR2. As the interruption is not not immediately served, the handler subroutine starts prematurely in the next cycle. The sawtooth shape also reveals the clock drift of the slave relative to the master. Recall the subplot a) of Figure 6.10 and the clock offset values around 11.305 in the x-axis. After 7.5 *s*, master's counter is ∼22700 TSU clock cycles behind slave's, which means 3036 clock cycles per second or 10.2 μs/s. This value perfectly matches with the clock drift obtained in Section 6.3.1.

The second column shows the clock offset of the three timestamps pairs after the re-synchronization act. The three columns on the right show the distributions of the second column for each load scenario. Subplots b, e, h show that the achieved synchronization accuracy for the three load scenarios is confined within ±20 *ns* region. The histograms of Figures 6.10c, f, i better show the clock offset distribution in that region. From the "Normal envelope" of the clock offsets, we deduce that the jitter introduced by the digital logic devices within the PHY and the MAC is totally random.

For the three load scenarios, the mean (μ_{CF}, μ_{DG} and μ_{EH}) and the standard deviation (σ_{CF}, σ_{DG} and σ_{EH}) of the clock offset is well confined in the interval from -17.14 *ns* to -7.14 *ns*, and 21.56 *ns* to 23.75 *ns*, respectively. The precision attained when using a hardware timestamping scheme is totally independent of the software load in the time-client node. Of sum importance are the standard deviations (σ's), as they show that the jitter introduced by the two Ethernet layers is bounded in the region from 21.56 *ns* to 23.75 *ns*. The most relevant clock offset subplots are 6.10e and f. This timestamp pair (t_D and t_G) represents the points of the transmitted timestamp at the master node, and the time of arrival at the slave node, respectively. Their jitter values ($\sigma_{DG\_no\_bg}$, σ_{DG_64B} and σ_{DG_1500B}) utilizing our message exchange pattern are bounded within the region of 22.2 and 23.17 *ns*. These values correspond to the sum of the jitter introduced by the PHY and the MAC of the transmitter and the receiver of a syncPDU journey plus an unknown amount. Recall Subsection 6.3.3; the jitter introduced by the MAC was $3.\widehat{33}$ *ns* and the round trip jitter of the physical layer was $13.\widehat{33}$ *ns*. Those numerical values can be partially used to justify the jitter of this experiment (i.e., the σ_{DG}'s). The syncPDUs of re-synchronization acts follow this path: **1)** the MAC of the master ($\varepsilon_{M,snd}$), **2)** the TSU of the master ($\varepsilon_{T,snd}$), **3)** the PHY of the master ($\varepsilon_{P,snd}$), **4)** the cable (ε_{cable}), **5)** the PHY of the slave ($\varepsilon_{P,rcv}$) and **6)** the TSU of the slave ($\varepsilon_{T,rcv}$). As the receiving TSU uses the propagation delay to re-synchronize, we consider that it introduces the variability of the RTD calculation, and thus $\varepsilon_{T,rcv} = \sigma_{RTD_{prec}}$. Rearranging these six components and substituting them with numerical values, we obtain Equation 6.1.

$$\begin{aligned} \varepsilon_{syncPDU} &= \varepsilon_{M,snd} + \varepsilon_{T,snd} + \varepsilon_{P,snd} + \varepsilon_{cable} + \varepsilon_{P,rcv} + \sigma_{RTD_{prec}} \\ &= 3.\widehat{33} + 3.\widehat{33} + \frac{13.\widehat{33}}{2} + (\sim 0) + \frac{13.\widehat{33}}{2} + 4.43 = 24.42\,ns \end{aligned} \tag{6.1}$$

The result of Equation 6.1 is nearly the same as the σ's in the time synchronization experiment (Figure 6.10), thus confirming the validity of all presented evaluation methods, time synchronization mechanism and platform architecture.

Phase Synchronization

By definition, the phase synchronization accuracy can be evaluated by comparing how aligned are the rising (or falling) edges of two electrical signals. To follow the same principle in our platform, we have purposely introduced in the TSU the programmable interval timer (PIT). The PIT asserts an electrical pulse of 3.3 volts once detects that the counter of the TSU matches a specific value defined by the user. The timestamp granularity and the width of the TSU's counter determines the interval of each pulse, i.e., in our design it is $2^{32}\times 3.\widehat{33}ns$=14.316 s. As this interval is too long for evaluating how close are two pulses, we have introduced a mask register inside the register block of the TSU ($match\_rng$) to change the periodicity of the pulses from 14.316 s to 53.$\widehat{33}$ ns. For this experiment, we have chosen to generate pulses of a width of 53.$\widehat{33}$ ns.

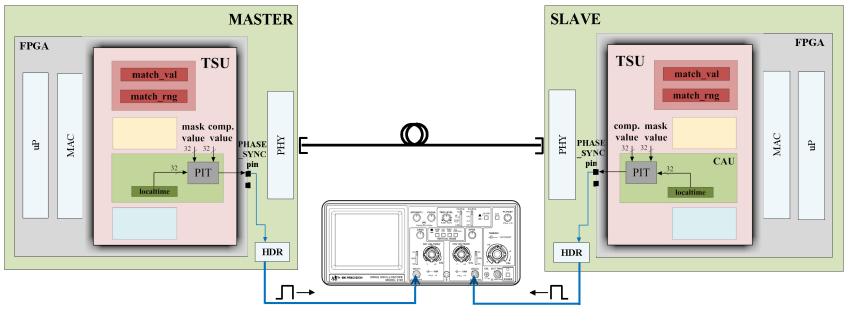

Figure 6.11: Setup the evaluation of time and phase accuracy.

To run the phase synchronization experiment, we have configured our platform as shown Figure 6.11. For each board, the output of the PIT inside the FPGA is connected to an external FPGA pin ($PHASE\_SYNC$ pin), this one to a jumper of the expansion header on-board (HDR) and each jumper to a channel of the oscilloscope. The $match\_val$ register is pre-loaded with the pulse interval value of 0×107AC, needed to generate periodic pulses at the interval of 225 μs[2]. The synchronization procedure has been slightly changed from the previous section. In this experiment we want to observe how close the two pulses are to each other after a re-synchronization event. If the rising edges of the two pulses are aligned, it means that the master and the slave are perfectly synchronized. To verify that, we have configured the timeouts of the timers of the platform to perform RTD calculations less often and re-synchronize more often, i.e., every 1 ms. The screenshot on the right of the Figure 6.12 shows the aligned pulses coming from the two boards at μs scale. The screenshot on the left of the Figure 6.12 zooms

[2] 0×107AC clk.cyc. = 67500 clk.cyc.×3.$\widehat{33}$ ns= 225 μs.

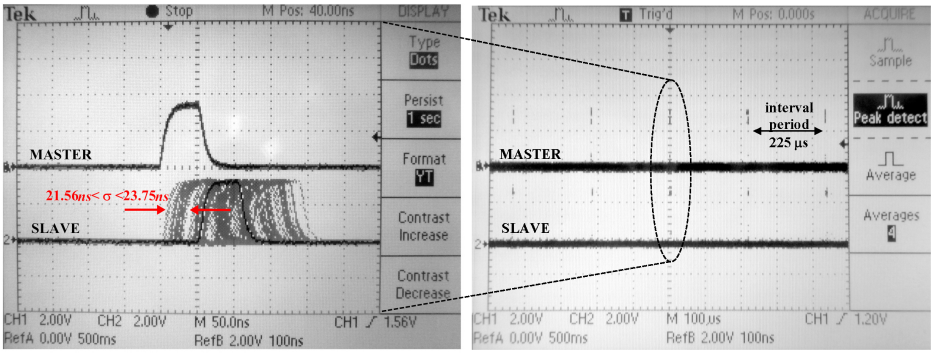

Figure 6.12: Phase error of the synchronized timing signal (right). Re-synchronization variability (left).

in one of the pair of pulses to observe the accuracy at *ns* level. From this plot, it can be seen an overlapping region of ∼25 *ns* (in red). This margin exactly corresponds to the jitter shown in Figure 6.10e. Those traces that fall out from this region are the remainings of the oscilloscope after a clock drift interval of 1 ms[3]. In this experiment we re-synchronize every 1 *ms*, and thus according to the table 6.3, the relative clock drift in 1 *ms* is around 114 *ns*. Therefore, from that value we deduce why the screenshot has traces in the region of 100 *ns*.

6.3.5 Clock Duplicity

Actual synchronization protocols, such as the IEEE 1588 or the upcoming 802.1as, consist on a software part and a hardware part. The hardware part free the ingress/egress timestamps from gross latencies and jitter, while the software part carries out a series of computational complex processes using the timestamps with which to infer an estimation value to correct the real time. It is precisely on the transfer of the timestamps from the hardware to the software part where there is a gross source of variability. Following the same reasoning as the packets travelling through a network, the timestamps within a node face latencies and jitter of intermediate elements such as bus arbitration blocks, interruption handler logic, memory controllers, and so on. When addressing synchronization accuracies of the order of nanoseconds, these sources of delay and jitter become critical. These latencies stem from the number of low-level software instructions to perform register-level transfers, memory transfers and CPU operations.

On this context, the goal of this experiment is to study the impact of software instructions on the synchronization accuracy between the TSU counter and the counter within the PPC. We want to know how many TSU clock cycles are needed to dump the PPC's counter value to the TSU's counter ($\delta_{PPC \rightarrow TSU}$),

[3] To obtain this plot we have set the oscilloscope in "persistence mode", which superimposes multiple oscilloscope waveforms on the same view.

and vice versa ($\delta_{TSU \rightarrow PPC}$). As explained in Section 5.5, the magnitude of these latencies is directly related to the number of low-level software instructions needed to perform read/write operations from/to the TSU's data register stack (recall Table 5.7). We can dissect each operation in a series of individual accesses to the PPC, the DDR memory and the TSU, all them with its associated delay. Equation 6.2a identifies the delay components in a transfer from the TSU to the PPC, and Equation 6.2b in the opposite direction. The common terms in the equations $\delta_{DDR_{wr}}$ and $\delta_{DDR_{rd}}$ correspond to the internal write and read delays of the DDR chips. Their magnitude is usually at the order of few nanoseconds. δ_{LOAD_CAU} and δ_{READ_CAU} denote the delay of *load_cau* and *read_cau* TSU drivers, respectively.

$$\delta_{TSU \rightarrow PPC} = \delta_{READ\_CAU} + \delta_{DDR_{wr}} + \delta_{DDR_{rd}} + \delta_{PPC_{wr}} \tag{6.2a}$$

$$\delta_{PPC \rightarrow TSU} = \delta_{PPC_{rd}} + \delta_{DDR_{wr}} + \delta_{DDR_{rd}} + \delta_{LOAD\_CAU} \tag{6.2b}$$

We have obtained and annotated the statistics of the timing of the time transfers 100k times using an exerciser loop. In this loop, we are continuously writing/reading to/from the TSU, depending on which delay we want to measure ($\delta_{PPC \rightarrow TSU}$/$\delta_{TSU \rightarrow PPC}$). Once the loop has done all the transfers, we proceed to average over the 100k samples. In this synthetic program the cache memory of the processor has been disabled in order to force external DDR SRAM memory accesses. The obtained metrics are summarized in Table 6.6 and the distribution functions of both transfers delays are plotted in Figures 6.13a and b. We first observe that the mean (μ) of $\delta_{PPC \rightarrow TSU}$ is greater than $\delta_{TSU \rightarrow PPC}$. This owes to the fact that the latency of *load_cau* instruction (δ_{LOAD_CAU}) is greater than the latency of *read_cau* instruction (δ_{READ_CAU}) (recall Table 5.7). The delay and variability introduced by the PPC and the memory is very low compared to the PLB indeterminism. From the distributions, we observe that the transfer times are confined within an interval of 20 to 25 TSU clock cycles.

Table 6.6: Number of TSU clock cycles to perform the transfer from CPU's counter to TSU's counter, and vice versa (percentage over 100k samples).

	μ	σ	Max.	Min.
$\delta_{PPC \rightarrow TSU}$	1604	20	1637	1534
$\delta_{TSU \rightarrow PPC}$	939	25	1047	892

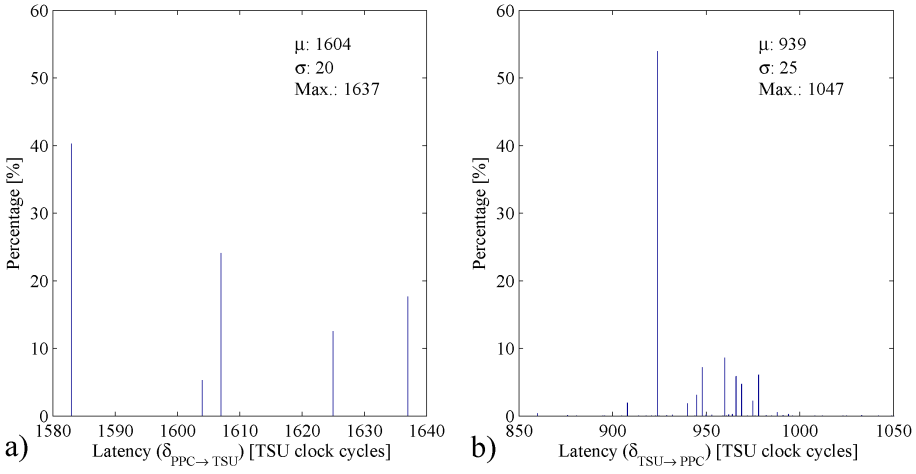

Figure 6.13: Distribution of the register transfer delays: PPC to TSU (a)) and TSU to PPC (b)).

6.4 Summary of the Results

6.4.1 Proposed Methods

In this chapter, we have presented methods to characterize and assess fundamental parameters in time sychronization at *ns*-level. A major problem incurred when addressing these accuracies is the burden of middleware and architectural components (e.g., bus arbitration and re-synchronization logic) in terms of the added latency and jitter, which can smudge completely the evaluation results. Another added problem is the clock frequency drift. Actual quartz oscillators suffer from a nominal of drift of a hundred of ppms that can lead to accumulated error offsets of one thousand of *ns* per *ms*, and thus making the evaluation process useless.

To correctly assess the synchronization parameters without the influence of these problems, we have described methods based on: **1)** distributed timestamping and **2)** programmable internal timer (see Section 6.3.4).

In the first one, we compare pairs of distributed timestamps from one node to the other. The timestamp pairs are chosen in the journey trip of a synchronization message in the way that they keep a symmetry in the ingress and egress paths. To minimize the influence of the accumulated clock offset, we have first characterized the relative clock drift between the master and the slave (Figure 6.2). Based on their relative clock drift, we have chosen the interval to read the remote (slave) clock after the re-synchronization act (τ_{obs}) in such a way that the accumulated error is not significant (Figure 6.10).

The second method consists on using PITs to completely isolate the clock drift in the measurement process (Section 6.3.4). Considering an optimal RTD

calculation, after the re-synchronization act, the pulses coming from both nodes should be perfectly aligned. Using PITs, we can evaluate the phase synchronization accuracy visually.

6.4.2 Synchronization Components

We have seen in Section 6.3.2 that the absence of a timestamping protection mechanism seriously impacts the synchronization between nodes. In a short experiment of 50 s, we obtained ∼17 erroneous timestamps, which represents the 34% of the overall data must be discarded. In absence of protection circuitry, the occurrence of erroneous data is totally random which means that in longer experiments there would be higher percentages of data that would have to be discarded. With our protection circuit based on *flancters-flags* (Section 5.4.4), we have totally eliminated possible erroneous timestamps.

One of our goals was to design a precise and accurate time synchronization mechanism with which to synchronize a *link* optimally. Reusing the message exchange pattern from EPON and leveraging our distributed timestamp architecture, we have achieved synchronization accuracies bounded within the jitter introduced by the hardware logic of Ethernet layers ($22.2\ ns < \sigma_{DG} < 23.17\ ns$). These results prove the effectiveness of a simple hardware-based synchronization method, as well as the precise architecture design.

To verify the jitter results obtained from the synchronization mechanism, we have conducted experiments to measure the latency and the jitter introduced by the MAC and the PHY layer individually. The sum of each jitter component matches with that obtained during the exchange of the messages (Equation 6.1).

§ 7. Conclusions

The actual trend in synchronization protocols for Ethernet-based networks is to partition critical functionalities, such as timestamping, and allocate them at hardware layers. Hardware characterizes for its low latency and delay variability, two parameters that greatly impair synchronization accuracy between nodes. The use of hardware for timestamping requires to design a specialized node architecture. As the legacy Ethernet is not standardized with time synchronization, actual Ethernet solutions that deliver synchronization are propietary approaches that address specific applications with specific needs in a variety of implementations. This is especially noticeable in the industrial field and the telecommunications sector.

On this background, we were motivated to pursue the following goal which is to prove in a real platform that the achievable sychronization accuracy between two nodes in a point-to-point configuration in legacy Ethernet can be bounded within the delay variability introduced by each Ethernet layer in a timing message delivery. As Ethernet is defined to be implemented in the hardware domain, the jitter introduced by its two layers should be very low and the synchronization achievable between peers too. Compared to IP-based solutions, time synchronization at Layer 2 only has to accommodate the jitter introduced by its two hardware layers.

To meet that goal, we have used the time synchronization mechanism used in EPON [IEEE Std. 802.3 (2005)]. This synchronization mechanism was defined in Ethernet standard to control the TDMA access of PtMP shared topologies. It characterizes for its simplicity and its suitability to be implemented in hardware. Its main functionality is to infer the latency of Ethernet layers from the sender to the receiver, and use it to cancel the propagation time of synchronization frames (syncPDU). Ideally, if the propagation time of a syncPDU can be perfectly cancelled, the synchronization accuracy should be within the jitter limits introduced by Ethernet layers. For this reason, this Thesis has proposed the concept of a Layer 2 time synchronization in the Ethernet technology (see Chapter 4).

This Thesis has more contributions. One of them is on the evaluation methods for *ns*-level time synchronization evaluation using FPGAs (see Chapter 6). FPGAs are inherently complex tools that, if not used properly, they may difficult the process of evaluation. In this work, we have provided a set of alternatives and guidelines to successfully overcome their limitations and validate correctly our initial proposal. The procedures and the built knowledge for using these tools optimally has allowed us to reach another important milestone, which is on

the results obtained from evaluation process. To the best of our knowledge, the synchronization platform implemented in this Thesis is the first one in achieving synchronization accuracies of few nanoseconds in a point-to-point configuration using standard FPGAs (see Section 6.3.4). From the results, we have demonstrated that a time synchronization mechanism fully implemented in the MAC would greatly benefit those actual and future applications with hard synchronization requirements.

An important contribution of this Thesis has been the design of a timestamp protection mechanism to capture timestamps coming from a high speed counter (see Section 5.4.4). If the speed at which the counter is summing up ticks is different from the control signal requesting the read of the counter, the timestamp will be subjected to a potential capture error. Whichever of the 32 bits of the counter might be misinterpreted as a logic '1' or '0', and vice versa. To prevent from these errors, we have readopted and modified a circuit called *flancter* [Weinstein (2000)]. Besides protecting the timestamp against metastability errors, our proposed protection mechanism characterizes for its reusability with other architectures and frequencies.

We have provided another important contribution after a detailed study of the the most prominent standard for time synchronization, the PTP [IEEE Std. 1588 (2008)]. PTP standard recommends the use of hardware for critical protocol functions, such as the timestamping, and the software for other more complex functions, e.g., timestamp processing. On this definition there is a limitation that depends on the internal architecture of computing platforms, which is the latency that introduces the software. At *ns*-scale, the software intervention hampers the synchronization accuracy notably. Timestamps with a resolution of few nanoseconds can be completely buried in the noise margin introduced by the software jitter. Considering this architectural limitation, we have studied the time burden of software on the synchronization accuracy (see Section 6.3.5).

7.1 Lessons Learned

We have drawn remarkable conclusions and observations along the work and the results from this Thesis:

⇒ **On the jitter**. Jitter can be defined differently depending of the context and the level of abstraction. In digital electronics, it can be defined as the deviations in a clocks output transition from their ideal positions. In computer systems, the jitter is associated to the operating system interference, caused primarily due to scheduling of daemon processes and handling of asynchronous events such as interrupts. In the context of networks, it is defined as the variation in packet transit delay caused by queuing, contention and serialization effects on the path through the network.

In the context of networks, jitter is better understood if the layered model is used. It raises from the deviations in the clocks output transition, at the physical level. To reconcile clock transitions between blocks, resynchro-

nization mechanisms are used in the hardware datapaths which add more variability. As the messages go across the layers, the interaction of the software with the hardware becomes bigger, and thus the accumulated jitter too.

⇒ **On the state of the art**. In the field of telecommunications, packet-based transportation is the foundation of future networks. The network transition from TDM-based equipment to packet technologies, such as Ethernet, started many years ago and is gradually migrating to access and metro networks. Todays networks are a hybrid mix of circuit and packet technologies particularly in the access and metro layers. There is a constant effort of several silicon manufacturers and standard bodies to provide rules for interoperability and coexistence of technologies in this migration process.

There is a clear consensus among main Tier telecommunication operators and standard bodies that the preferred solution for delivering time and frequency synchronization in Ethernet-based networks is the PTP. PTP has become the *the facto* standard for delivering synchronization in a wide variety of applications and geographical distances. Actual PTP applications range from test and measurement [The LXI Consortium (2010)], industrial automation [EPSG (2010), Profibus (2010), ODVA, Inc. (2010), EtherCAT (2010)], residential networks [P802.1as (2008)] and telecommunications [Metro Ethernet Forum (2010)]. Among all these bunch of areas of application, it is clear that a unique definition of Ethernet is not possible anymore. PTP has been specifically designed to synchronize a wide range of distributed applications with an accuracy of less than one microsecond.

⇒ **On the Layer 2 time synchronization model**. Pure Layer 2 implementations of the prominent PTP protocol have been requested in several areas, especially in the industrial field. A pure Layer 2 model would enable easier silicon-based solutions and more efficient switch technology. Time synchronization at Layer 2 has only to accommodate the jitter introduced by Ethernet layers, which is much smaller compared to other approaches based on upper layers, e.g., IP-based solutions. Despite of the advantages of a pure Layer 2 synchronization approach, there is a clear direction towards IP-networking. IP networks are easier to manage and do not suffer from well-known scalability problems of pure Layer 2 networks.

The last version of PTP protocol has included a new normative to deliver PTP services through IEEE 802.3 data frames [IEEE Std. 1588 (2008)]. This new definition is specifically tailored for small networks targeting extreme accuracies of the level of tenths of nanoseconds. This new definition opens a niche for those applications leveraging a "Layer 2 PTP" implementation.

⇒ **On the hardware design**. When designing with hardware, the correct functionality and the veracity of the measured data strongly depends on the correct allocation of the hardware components along the programmable die

during the hardware design phase. Physical allocation of hardware blocks within the chip is a time-consuming process but crucial in FPGA designs. In the FPGA world, keeping the simplicity is a must and thus Adding unecessary extra components, may likely lead a system to misfunction. If not, the implementation time of the FPGA development tools will increase dramatically. When designing with hardware, less is more.

⇒ **On the design reliability**. Besides the internal platform delays, there is the added challenge of ensuring that the information is reliable. Digital systems with more than one clock signal are prone to render errorenous results due to the interactivity between blocks that run at different frequencies. For a reliable information exchange, the blocks would need to be synchronous, i.e., to be paced by a single clock signal. Those designs that are forced to be asynchronous, as it is the case of the work presented in this Thesis, need to be provided with a synchronization mechanism that reconciles the disparity of clock frequencies.

⇒ **On the development tools**. FPGAs are provided with a bunch of development tools (EDA) that are implemented by software engineers who are human. Sometimes they make mistakes, or pick sub-optimal data structures or algorithms which produces correct but inefficient results for a certain class of circuits. EDA tools have a mixture of innovations in both algorithms and the development of heuristics. Heuristics frequently give a fast near-optimal solutions to computationally intractable problems. The reasoning is "good enough and fast is better than perfect and never". Placement and routing are areas where complex heuristics have been developed over the years. However, they are not universally applicable and, in some designs, they may render poorly conditioned data that often result in slow execution times and poor results. In the situations where efficiency suffers, users are required to make tradeoffs. Fortunately, the implementation limitations are generally minimized over the time by the same EDA companies through post-releases, service packs and new software versions.

⇒ **On the node architecture for tight clock synchronization**. As long as Moore's law validity remains, computer architectures will continue evolving as in the last three decades. PC's architecture trend is to overprovision the architecture in terms of speed, storage capacity and parallelization. While in the general processing model, the primary time constraints are throughput or execution time, for synchronization applications and time-based architectures time determinism is a primary concern. To address *ns*-level synchronization, the architecture has to be redesigned in such a way that it can execute operations deterministically. As shown in Chapter 6, the latency and the variability added by the internal components (intermediate memories, busses, re-synchronization and arbitration circuits, etc) impair the synchronization accuracy. In [Lee (2005)], there is an interesting discussion about how computer science has abstracted away the notion of time, and that a shift towards redesigning computer architectures to deliver

precisely timed behaviors would benefit those applications with hard timing requirements.

7.2 Future Directions

One of the requirements in the design of our hardware block was to provide reusability in order to port the design to other platforms. With this intention in mind, our future research points toward proving the "synchronization scalability" in a bridged Ethernet environment. In Section 6.3.4, we found that the practical jitter (σ_{DG}) introduced by Ethernet layers could be within 22.2 *ns* and 23.17 *ns*. Considering an optimal time synchronization mechanism for a bridged Ethernet scenario, we should obtain synchronization accuracies within the bounds given by the jitter introduced by each bridge port of the synchronization path. However, this would require to cancel the fixed part of the propagation time in the destination node, and provide bridges with hardware timestamping and logic to compensate the residence time of the packets within the same bridge. Figure 7.1 better illustrates this hypothesis.

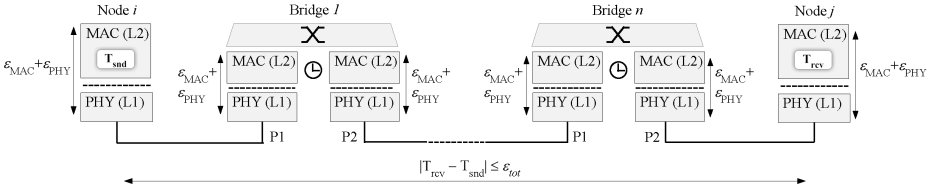

Figure 7.1: Theoretical synchronization accuracy in a bridged Ethernet scenario.

The synchronization accuracy between two nodes, i and j, separated by n bridges nodes, should be given by the sum of the jitter introduced by each Ethernet stack of the bridges, i.e.,

$$\varepsilon_{tot} = \sum_{k=1}^{n+1} \varepsilon_{MAC} + \varepsilon_{PHY} \tag{7.1}$$

For instance, in a pure Layer 2 wide area network with 10 bridges the synchronization accuracy between two nodes should be ∼260 *ns*, according to Equation 7.1 and using our measured values. To prove the synchronization scalability and these hypotheses, we must shift to another platform. Our platform has several important limitations that slow down the process of extending Ethernet with new capabilities. The most restrictive is the MAC inaccessibility that imposes us to create a shim, i.e. the TSU, closely adhered at its interface. Another limitation is the number of Ethernet ports which is restricting us to work in a point-to-point configuration and thus blurring the applicability of our method in multihop networks. For these reasons, we are also working on the migration of our synchronization functionalities in the NetFPGA platform [NetFPGA (2009)], which is a

4-port PCI-based card that enables rapid development of HW-accelerated packet processing applications. An opensource standard compliant MAC for NetFPGA is going to be released soon. This would facilitate the integration of our extensions to be tested in a multihop bridged network.

3GPP2 (2010). 3rd Generation Partnership Project 2. http://www.3gpp2.org/.

802.1as WG (2010). 802.1AS - Timing and Synchronization. http://www.ieee802.org/1/pages/802.1as.html.

Anceaume, E. & Puaut, I. (1997). A Taxonomy of Clock Synchronization Algorithms. Technical report, IRISA, No. 1103.

Anceaume, E. & Puaut, I. (1998). Performance Evaluation of Clock Synchronization Algorithms. Technical report, IRISA, No. 3526.

Arlos, P. & Fiedler, M. (2005). A Comparison of Measurement Accuracy for DAG, Tcpdump and Windump. Technical report, Blekinge Institute of Technology, Karlskrona, Sweden.

Arlos, P. & Fiedler, M. (2007). A Method to Estimate the Timestamp Accuracy of Measurement Hardware and Software Tools. In *PAM'07: Proceedings of the 8th International Conference on Passive and Active Network Measurement*, pp. 197–206. Springer-Verlag, Berlin, Heidelberg.

Aweya, J., Montuno, D. Y., Ouellette, M., & Felske, K. (2006). Clock Synchronization using a Linear Process Model. *Int. Journal Network Management*, 16(1):3–28.

Aweya, J., Montuno, D. Y., Ouellette, M., & Felske, K. (2007). Clock Synchronization for Packet Networks using a Weighted Least-Squares Error Filtering Technique and Enabling Circuit Emulation Service: Research Articles. *Int. Journal Communications Systems*, 20(6):669–694.

Bregni, S. (2002). *Synchronization of Digital Telecommunications Networks*. John Wiley & Sons, Inc., New York, NY, USA.

Brown, S. & Rose, J. (1996). FPGA and CPLD Architectures: A Tutorial. *IEEE Design Test of Computers*, 13(2):42–57.

Chiruvolu, G., Ge, A., Elie-Dit-Cosaque, D., Ali, M., & Rouyer, J. (2004). Issues and Approaches on Extending Ethernet Beyond LANs. *IEEE Communications Magazine*, 42(3):80–86.

Cisco Systems, Inc. (2010). Synchronous Ethernet: Achieving High-Quality Frequency Distribution in Ethernet NGNs. Available from: http://www.cisco.com/.

Clouston, B., Systems, C., & Moore, B. (1998). Definitions of Managed Objects for APPN. Network Working Group Request for Comments: 2455.

Decotignie, J.-D. (2005). Ethernet-Based Real-Time and Industrial Communications. *Proceedings of the IEEE*, 93(6):1102–1117.

Dietz, A. M., Ellis, S. C., & Starmer, F. C. (1995). Clock Instability and Its Effect on Time Intervals in Performance Studies. Technical report, Durham, NC, USA.

Donnelly, S. (2002). *High Precision Timing in Passive Measurements of Data Networks*. Doctoral Thesis, The University of Waikato.

Eidson, J. C. (2006). *Measurement, Control, and Communication Using IEEE 1588 (Advances in Industrial Control)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Endace (2009). DAG 4.5G2/G4 2/4 Port Network Monitoring Card. http://www.endace.com.

EPSG (2010). Ethernet Powerlink. http://www.ethernet-powerlink.com/.

ETH - Microelectronics Design center (2010). ASIC Cost Estimator. http://www.dz.ee.ethz.ch/?id=1592.

EtherCAT (2010). Ethernet for Control Automation Technology. http://www.ethercat.org/.

Garner, G. (2007). Ethernet QoS, Timing and Synchronization Requirements. *Joint ITU-T/IEEE Workshop on Carrier-class Ethernet. Available from: http://www.itu.int/ITU-T/worksem/cce/programme.html*.

Höller, R., Horauer, M., Gridling, G., Kerö, N., Schmid, U., & Schossmaier, K. (2002). SynUTC - High Precision Time Synchronization over Ethernet Networks. In *Proceedings of the 8th Workshop on Electronics for LHC Experiments*.

Hough, H. (1991). A GPS Precise Timing Sampler. *GPS World*, pp. 33–36.

IEEE Std. 1588 (2002). IEEE Std. 1588 - 2002 IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Std. 1588-2002*, pp. i–144.

IEEE Std. 1588 (2008). IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Std. 1588-2008*, pp. c1–269.

IEEE Std. 802.3 (1998). IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements. Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. *IEEE Std. 802.3, 1998 Edition*, p. i.

IEEE Std. 802.3 (2002). IEEE Standard for Information Technology–Telecommunications and Information Exchange Between Systems–Local and Metropolitan Area Networks–Specific Requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. *IEEE Std. 802.3-2002*, pp. 1–790.

IEEE Std. 802.3 (2005). IEEE Standard for Information Technology–Telecommunications and Information Exchange Between Systems–Local and Metropolitan Area Networks–Specific Requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. *IEEE Std. 802.3-2005*, pp. 1–790.

IETF-TICTOC (2010). Timing over IP Connection and Transfer of Clock (TICTOC). https://datatracker.ietf.org/wg/tictoc/.

Infonetics Research (2009). Infonetics research. http://www.infonetics.com/.

IPerf (2010). Iperf. http://dast.nlanr.net/Projects/Iperf.

ISO 8601:2004 (2004). Data Elements and Interchange Formats–Information Interchange–Representation of Dates and Times. *International Organization for Standardization, Geneva*.

ITU-T G.781 (1999). International Telecommunication Union, Telecommunication Standardization Sector (ITU-T), 1999. ITU-T. G.781: Transmission Systems and Media, Digital Systems and Networks.

ITU-T G.810 (1996). International Telecommunication Union, Telecommunication Standardization Sector (ITU-T), 1996. ITU-T. G.810: Definitions and Terminology for Synchronization Networks.

ITU-T G.823 (2000). International Telecommunication Union, Telecommunication Standardization Sector (ITU-T), 2000. ITU-T. G.823: The Control of Jitter and Wander within Digital Networks which are based on the 2048 kbit/s.

ITU-T G.8261 (2008). International Telecommunication Union, Telecommunication Standardization Sector (ITU-T), 2008. ITU-T. G.8261: Timing and Synchronization Aspects in Packet Networks.

ITU-T G.8262 (2007). International Telecommunication Union, Telecommunication Standardization Sector (ITU-T), 2007. ITU-T. G.8262: Timing Characteristics of Synchronous Ethernet Equipment Slave Clock (EEC).

ITU-T G.8264 (2007). International Telecommunication Union, Telecommunication Standardization Sector (ITU-T), 2007. ITU-T. G.8264: Distribution of Timing Information Through Packet Networks.

ITU-T Q13/15 SG (2008). Question 13/15 network synchronization and time distribution performance. http://www.itu.int/ITU-T/studygroups/com15/sg15-q13.html.

ITU-T Y.1731 (2006). International Telecommunication Union, Telecommunication Standardization Sector (ITU-T), 2006. ITU-T. Y.1731: OAM Functions and Mechansims for Ethernet based Networks. Blue Book.

J. Stephenson (Altera, Corp.) (2009). Don't let metastability cause problems in your fpga-based design. http://www.pldesignline.com/220300400.

Kim, C., Caesar, M., & Rexford, J. (2008). Floodless in Seattle: A Scalable Ethernet Architecture for Large Enterprises. In *SIGCOMM'08: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, pp. 3–14. ACM, New York, NY, USA.

Kim, C. & Rexford, J. (2007). Revisiting Ethernet: Plug-and-Play Made Scalable and Efficient. In *15th IEEE Workshop on Local Metropolitan Area Networks, 2007. LANMAN 2007*, pp. 163–169.

Kopetz, H. (1997). *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer, Boston.

Kopetz, H. & Ochsenreiter, W. (1987). Clock Synchronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, 36(8):933–940.

Kramer, G. & Pesavento, G. (2002). Ethernet Passive Optical Network (EPON): Building a Next-Generation Optical Access Network. *IEEE Communications Magazine*, 40(2):66–73.

Kutschera, C., Veigl, C., Höller, R., Rössler, P., Kerö, N., Weiss, C., Gröblinger, A., Muhr, H., & Cadek, G. (2009). Background IEEE 1588 Clock Synchronization over IEEE 802.3/Ethernet. In *3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities, 2008. TridentCom 2008*, pp. 1–10.

Lee, E. A. (2005). Absolutely Positively on Time: What Would It Take? *Computer*, 38(7):85–87.

Liskov, B. (1993). Practical Uses of Synchronized Clocks in Distributed Systems. *Distributed Computing*, 6(4):211–219.

Lombardi, M., Heavner, T., & Jefferts, S. (2007). NIST Primary Frequency Standards and The Realization of The SI Second. *Journal of Measurement Science*, 4:74.

Marzullo, K. & Owicki, S. (1983). Maintaining the Time in a Distributed System. In *Proceedings of the Second Symposium on Principles of Distributed Computing*, pp. 295–305. ACM SIGPLAN/SIGOPS.

Maxim Integrated Products Inc. (2010). DS3104 – Line Card Timing IC with Synchronous Ethernet Support . Available from: http://www.maxim-ic.com/.

Merriam-Webster, Inc. (2010). Merriam-Webster Online Dictionary. http://www.merriam-webster.com/dictionary/.

Messerschmitt, D. G. (1990). Synchronization in Digital System Design. *IEEE Journal on Selected Areas in Communications*, 8(8):1404–1419.

Metro Ethernet Forum (2010). Metro Ethernet Forum. http://metroethernetforum.org/.

Micheel, J., Donnelly, S., & Graham, I. (2001). Precision timestamping of Network Packets. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pp. 273–277.

Microsistemes Timing & Synchronization Solutions (2010). The IRIGB Web Site. http://irigb.com/.

Mills, D. L. (1991). Internet Time Synchronization: The Network Time Protocol. *IEEE Transactions on Commmunications*, 39(10):1482–1493.

Mills, D. L. (1992a). A Computer-Controller Loran-C Receiver for Precision Timekeeping. Technical report, Electrical Engineering Department Report, University of Delaware.

Mills, D. L. (1992b). Network Time Protocol (version 3) Specification and Implementation. Network Working Group Request for Comments: 1305.

Mills, D. L. (1994). Unix Kernel Modifications for Precision Time Synchronization. Electrical Engineering Department, University of Delaware. Technical report.

Mills, D. L. (1995). Improved Algorithms for Synchronizing Computer Network Clocks. *IEEE/ACM Transactions on Networking*, (3):245–254.

Mills, D. L. (2006a). *Computer Network Time Synchronization: the Network Time Protocol*. CRC Press.

Mills, D. L. (2006b). Network Time Protocol Version 4 Reference and Implementation Guide.

Müller, T., Ockert, A., & Weibel, H. (2004). PHYs and Symmetrical Propagation Delay. In *Proceedings of the 2004 IEEE 1588 Conference - NIST Technical Report NISTIR 7192*.

Myers, A., Ng, T. E., & Zhang, H. (2004). Rethinking the Service Model: Scaling Ethernet to a Million Nodes.

National Institute of Standards and Technology (NIST) (2010). IEEE 1588 Website. http://ieee1588.nist.gov/.

NetFPGA (2009). NetFPGA. http://www.netfpga.org.

Nguyen, T. T. (2007). Evaluating Timestamping Accuracy for ASUS P5LD2-VM Motherboard with Intel NICs. Technical Report 070228F, Swinburne University of Technology.

Nicolau, C., Sala, D., & Cantó, E. (2009). Clock Duplicity for High-Precision Timestamping in Gigabit Ethernet. In *International Conference on Field Programmable Logic and Applications, 2009. FPL 2009.*, pp. 379–384.

NTP (2010). NTP: The Network Time Protocol. http://www.ntp.org.

Nylund, S. & Holmeide, O. (2005). IEEE 1588 Ethernet Switch Transparency-No Need for Boundary Clocks! Unpublished.

ODVA, Inc. (2010). ODVA. http://www.odva.org/.

Oregano Systems - Design & Consulting Ltd. (2009). Syn1588 layer 2. http://www.oregano.at.

P802.1as (2008). IEEE Draft Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks. *IEEE Unapproved Draft Std P802.1AS/D2.0 Feb 2008*, pp. –.

Pásztor, A. & Veitch, D. (2002). PC Based Precision Timing Without GPS. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 1–10. ACM, New York, NY, USA.

Paxson, V. (1998). On Calibrating Measurements of Packet Transit Times. *SIGMETRICS Perform. Eval. Rev.*, 26(1):11–21.

Postel, J. (1981). Internet Control Message Protocol (ICMP). Network Working Group Request for Comments: 792.

Postel, J. & Harrenstien, K. (1983). Time Protocol. Network Working Group Request for Comments: 868.

Profibus (2010). Profibus - Profinet. http://www.profibus.com/.

Ramanathan, P., Shin, K. G., & Butler, R. W. (1990). Fault-Tolerant Clock Synchronization in Distributed Systems. *Computer*, 23(10):33–42.

Reina, M. (2010). Operationalizing A Control Plane Network. *Conference on Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference (OFC/NFOEC)*, pp. 1–22.

Ridoux, J. & Veitch, D. (2007). A Methodology for Clock Benchmarking. In *3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities, 2007. TridentCom 2007*, pp. 1–10.

Ridoux, J. & Veitch, D. (2009). Ten Microseconds Over LAN, for Free (Extended). *IEEE Transactions on Instrumentation and Measurement*, 58(6):1841–1848.

Schmid, U., Horauer, M., & Kerö, N. (1999). How to Distribute GPS–time over COTS–based LANs. In *Proceedings of the 31th IEEE Precise Time and Time Interval Systems and Application Meeting (PTTI'99), Dana Point, California.*

Schneider, F. B. (1987). Understanding Protocols for Byzantine Clock Synchronization. Technical report, Ithaca, NY, USA.

Sexton, M. & Reid, A. (1997). *Broadband Networking: ATM, SDH and SONET.* Artech House.

Spirent Communications (2010). Smartbits. http://www.spirent.com/Solutions-Directory/Smartbits.aspx.

Symmetricom (2010). Passive Optical Networks. http://www.symmetricom.com/.

Tcpdump (2010). Tcpdump/Libpcap. http://www.tcpdump.org/.

Teener, M. J., Battaglia, J., A., B., Ryu, E. H., & Kim, Y. (March 2005). Residential Ethernet Tutorial. Available from: http://ieee802.org/3/.

The LXI Consortium (2010). LAN extensions for Instrumentation (LXI). http://www.lxistandard.org/.

Trimble (2010). Trimble Timing & Synchronization. http://www.trimble.com/timing/.

Troxel, G. D. (1994). *Time Surveying: Clock Synchronization over Packet Networks.* Doctoral Thesis, Massachusetts Institute of Technology.

U.S. Naval Observatory (1999). USNO NAVSTAR Global Positioning System. http://tycho.usno.navy.mil/gpsinfo.html.

Wac, K., Arlos, P., Fiedler, M., Chevul, S., Isaksson, L., & Bults, R. (2007). Accuracy Evaluation of Application-Level Performance Measurements. In *Proceedings of 3rd Conference on Next Generation Internet Networks, 2007. EuroNGI 2007*, pp. 1–5.

WAND - Network Research Group (2010). Wand - network research group homepage. http://www.wand.net.nz/.

Weinstein, R. (2000). "The Flancter". http://www.floobydust.com/flancter/.

Whatcott, R. (Xilinx, Inc.) (2009). Timing Closure 6.1i (WP331). Available from: http://www.xilinx.com.

WinPcap (2010). WinPcap: The Windows Packet Capture Library. http://www.winpcap.org/.

Xilinx, Inc. (2006). ML401/ML402/ML403 Evaluation Platform User Guide (ug080 v2.5).

Xilinx, Inc. (2007a). PowerPC 405 Processor Block Reference Guide. Embedded Development Kit (ug018 v2.2).

Xilinx, Inc. (2007b). Virtex-4 Embedded Tri-Mode Ethernet MAC User Guide (ug074 v1.6).

Xilinx, Inc. (2008). Virtex-4 FPGA User Guide (ug070 v2.6).

Zarlink Semiconductor Inc. (2010). Timing & Synchronization – Packet Network Timing. http://www.zarlink.com/zarlink/hs/timing_PacketNetworks.htm.

Zhou, Z., Cong, L., Lu, G., Deng, B., & Li, X. (2010). HATS: High Accuracy Timestamping System Based on NetFPGA. In *Advances in Computer Science and Information Technology*, pp. 183–195. Springer-Verlag, Berlin, Heidelberg.