TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DISSERTATION

On Enhanced Clock Synchronization Performance Through Dedicated Ethernet Hardware Support

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

Ao. Univ. Prof. Dipl.-Ing. Dr. Wolfgang Kastner
Institut für Rechnergestützte Automation,
Arbeitsbereich Automatisierungssysteme
und
Prof. Dr. Francisco Vasques
Departement of Mechanical Engineering,
University of Porto, Portugal

eingereicht an der Technischen Universität Wien
Fakultät für Informatik

durchgeführt am
Institut für Integrierte Sensorsysteme der
Österreichische Akademie der Wissenschaften
unter der Leitung von Dipl.-Ing. Dr. Thilo Sauter

von

Dipl.-Ing. Patrick Loschmidt
Wienergasse 114-116/2/5
2380 Perchtoldsdorf
Matr.-Nr. 9526652

Wien, im Dezember 2010

Patrick Loschmidt

Contents

Abstract

The evolvement of Ethernet in factory automation and test & measurement applications generated the need for high-accuracy clock synchronization. While factory communication can normally fulfil application requirements based on an average accuracy in the range of microseconds, test & measurement and several other applications (e. g. wireless position determination) seek for ultimate performance, meaning nanoseconds at most. This new demand, together with the issue of simplified node configuration, prepared the ground for a new protocol allowing for standardised synchronization in new dimensions.

The Precision Time Protocol standardised as IEEE 1588 addresses the topic of networked measurement and control systems by giving a set of rules for running high accuracy clock synchronization in a self-configuring topology. Due to the fact that not only the protocol itself is defined, but also the message timestamp point for several network technologies, it allows the implementation with hardware support. The latter enables the possibility to gain timestamps without jitter introduced by varying processing time in (software implemented) higher network layers.

The main goal of the present work is to analyse and propose a way to get from common PTP accuracy, which is about 100 ns, to systems being able to deliver sub-nanosecond performance. Although state-of-the-art devices for hardware timestamping PTP messages use the interface between the physical and data link layer, jitter sources remain, which can hinder high accuracy.

The aimed hardware support for highly accurate synchronization over Ethernet is as well targeted at the mentioned interface. This is argued by the fact that most network interface implementations use hardware blocks at this level and software routines for the higher functionalities. If the – for network synchronization indispensable – timestamps are drawn there, the effects of timely uncertainties of the software stack can be removed. Nevertheless, it is possible to operate at bit level.

An extensive analysis of existing disturbance factors on the synchronization accuracy and their (statistical) behaviour allows to design appropriate architectures and counter measures to further reduce the effects on the precision of drawn timestamps at this level. The examinations are partly done by measuring and modelling the disturbances and completed by mathematical expressions built from the knowledge of the type of the internal jitter source.

In order to show a complete picture of all relevant factors, not only the physical influence, but also structural issues are covered. These include the interaction of software with the time-keeping hardware, the analysis of external application requirements and the actual implementation of clock synchronizing hardware cells. To round network controlled systems off, the analysis describes design issues for synchronization systems and their relation to the actual hardware implementation.

Zusammenfassung

Der verstärkte Einsatz von Ethernet in der Industrieautomatisierung und die zunehmende Verbreitung in Test- und Meßsystemen bewirkte den Bedarf an hochgenauer Uhrensynchronisation auf diesem Medium. Während Anforderungen in der Automatisierungstechnik üblicherweise im Bereich von einigen Mikrosekunden liegen, fordern Test- und Meßaufgaben sowie zahlreiche andere Anwendungen (z. B. drahtlose Positionsbestimmung) extreme Genauigkeiten – oft unter einer Nanosekunde. Diese neuen Anforderungen und der Bedarf an vereinfachter Konfiguration der Netzwerkknoten schufen die Basis für ein neues Protokoll für standardisierte Uhrensynchonisation in einer neuen Dimension.

Das „Precision Time“ Protokoll wurde als IEEE 1588 Standard veröffentlicht. Mit dem Hintergrund von Netzwerk basierten Meß- und Steuersystemen definiert es eine Reihe von Regeln für hochgenaue Uhrensynchronisation in selbstkonfigurierenden Topologien. Da neben dem Protokoll selbst auch der Zeitpunkt für die Zeitstempelung von Paketen in diversen Technologien definiert ist, erlaubt es auch den Einsatz von Hardwareunterstützung. Letztere erlaubt Zeitstempel zu generieren, die von den variablen Verarbeitungszeiten in höheren (in Software implementierten) Teilen des Netzwerkstacks unbeeinflusst sind.

Ziel dieser Arbeit ist es, durch Analyse der beteiligten Komponenten die aktuelle Genauigkeit von circa 100 ns auf Subnanosekunden zu verbessern. Obwohl aktuelle Implementierung für Hardware generierte Zeitstempel von Precision Time Protocol (PTP) Nachrichten die Schnittstelle zwischen physikalischer und Sicherungsschicht nutzen, verbleiben Jitterquellen, die eine hochgenaue Synchronisierung verhindern.

Die angestrebte Lösung setzt ebenfalls auf der erwähnten Schnittstelle auf. Dies ist vorteilhaft, da die meisten Lösungen in diesem Bereich noch Hardware einsetzen, während höhere Schichten bereits in Software implementiert sind. Zeitstempel an diesem Übergang sind daher unbeeinflusst von zeitlichen Schwankungen in der Verarbeitung durch die Software, dennoch ist ein Zugriff auf die Nachricht auf Bit-Ebene möglich.

Eine detaillierte Analyse der verbliebenen Störfaktoren für die Uhrensynchronisation und deren Charakteristik erlaubt den Entwurf geeigneter Architekturen und Gegenmaßnahmen, um den Einfluss auf die Präzision der benötigten Zeitstempel zu mindern. Die Untersuchungen erfolgen zum Teil durch Messung, aber auch durch Modellierung beziehungsweise mit Hilfe mathematischer Äquivalenzmodelle unter Kenntnis der typischen Eigenschaften der Störquelle.

Um die relevanten Faktoren vervollständigen zu können, werden nicht nur die physikalischen, sondern auch strukturellen Einflüsse analysiert. Dazu zählen die Abstimmung zwischen der notwendigen Software und der Zeitbasis, die Analyse externer Applikationsanforderungen und die Implementierung einer Uhrensynchronisationszelle. Für die Anwendung in Kontrollsystemen werden auch Entwurfsfragen bezüglich des Synchronisationssystems und deren Zusammenhang mit der Implementierung behandelt.

Acknowledgement

I want to express my gratitude to everybody who helped me during the course of my studies. Concerning my dissertation I am thankful to Wolfgang Kastner and Francisco Vasques who both agreed to be the supervisors and gave a lot of helpful advice.

Special thanks also go to Thilo Sauter for his support. He made it possible for me to do all the required research since the early beginnings of the Institute. The supportive surroundings at work provided the base for my studies and their successful finishing.

Uncountable discussions, numerous critics, and countless hints were provided by my room mate, colleague and friend Georg Gaderer. There was no effort I could have asked for that he would not have taken to support my work. Thus, I am very thankful for all his help and the massive feedback to the document. He and his wife, Doris Richling, had the endurance to continuously push me towards the finish in which they now succeeded.

Special thanks go to all members of the `CSI:WN` (`C`lock `S`ynchronization `I`nvestigation: `W`iener `N`eustadt) group. In particular I want to thank my co-authors (Reinhard Exel, Aneeq Mahmood, Anetta Nagy, and Nataša Simanić) for their contributions to the topic and many discussions that helped me with a lot of enhancements to the topic of clock synchronization.

My friends Andrea Blatnek and Renate Hoffmann helped a lot with spell checking and always had an open ear for my problems, which made me endure the long way to the final document.

My parents not only supported my studies from the very beginning, but they also assisted me in any hard time I had and provided me suggestions how to solve difficult situations. Their advice always meant a lot to me and often brought me back to the essentials of a topic. Furthermore, they also helped me fixing numerous mistakes in the document with unbeatable care. As already with my previous works, whenever I needed support they found a way to give it to me.

1 Introduction

The successful spread of Ethernet based networks in the office area has prepared the ground for many interesting solutions enabling unique application requirements to be fulfilled by this general purpose networking technology. Together with the Internet Protocol (IP), interconnecting billions of nodes over the Internet via a universal approach, specialised network technologies like traditional fieldbuses lost importance. Although Ethernet and IP substantially lack real-time capabilities, classic fieldbuses and in particular sensor- and control networks are replaced by appropriate solutions based on Ethernet for the sake of simplicity and cost-efficiency.

Important services, which were not foreseen in the design of Ethernet are real-time capabilities, in particular clock synchronization. First software-based approaches to tackle the problem originated from the Internet, where the problem of correct ordering of events and actions arose from the distributed character of applications. The early software implementations without any special measures limited the accuracy to milliseconds, but were nevertheless suitable for a wide range of applications.

The goal of the scientific work is to determine the limits of Ethernet-based clock synchronization, the various influencing factors, and efficient structures to accomplish the goal of high-precision clock synchronization with sub-nanosecond performance and long-term stability. This means that a timing node should be able to precisely track an upstream timebase in a continuous manner and therefore show the same behaviour as the source for time intervals significantly larger than the synchronization period.

1.1 The Necessity of Clock Synchronization in Distributed Systems

Distributed systems are referred to as a collection of spatially separated, independently running processes. In order to form a complete system, these processes or algorithms communicate with each other using message exchange. In case the transmission delay between two processes is not negligible compared to the time between two subsequent events in a single process, the term *distributed system* applies [Lam78].

Following this definition, implementations of distributed systems can have different appearances. One would be a system consisting of spatially wide distributed computers, which jointly process some task and therefore have to be coordinated. Another example is the interaction of different function blocks (e. g., processing and storage) on a chip. Both fulfil the requirement that the delay for messages exchange can be higher than the temporal distance between two events, which hinders the establishment of a common timebase.

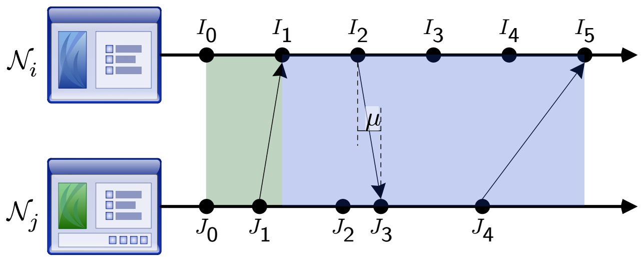


Figure 1.1: Ordering of events in a distributed system without a local timebase – parallelism as seen by node $\mathcal{N}_i$

Resulting from the interaction and dependency on the results of remote blocks, there is a general need for ordering events in a distributed system. The essential question is, how the occurrence of events in separated function blocks can be ordered with respect to each other. The introduction of appropriate definitions, together with local clocks at each spatially distributed element, gives the theoretical background for handling such systems [Lam78].

The basic principle for ordering of events in a message exchanging system of two nodes (running some sort of process), $\mathcal{N}_{i,j}$, is shown in figure 1.1. Since concurrency of events is based on the information exchanged, seen from $\mathcal{N}_i$, all events occurring between two receptions of a message appear to be concurrent, i. e., $J_{0,1}$ to I_0 and $J_{2,3,4}$ to $I_{2,3,4,5}$. From the point of view of $\mathcal{N}_j$ the events $I_{0,1,2}$ are happening in parallel to J_3 (and the events before). The parallelism can also be explained by the fact that always the remote node can only know about events after they happened and their occurrence was reported using a transmission (experiencing the delay μ). The remote node is not able to distinguish between events that took place between two message exchanges.

Based on that, another method to get ordered events in a distributed system is to provide a local timebase in each element exchanging messages with others. This allows for finer granularity without the dependency on the reception of a message after each event. The easiest approach for a general order is to have a physical local clock $\mathcal{C}_i$ keeping its value $C_i(t)$ at node $\mathcal{N}_i$. It shall follow the global real time with only a negligible small error rate κ, which is equivalent to the requirement

$$|\mathrm{d}C_i(t)/\mathrm{d}t - 1| < \kappa \ \forall i, \kappa \ll 1. \tag{1.1}$$

This requirement still does not ensure the correct ordering of events between several nodes of a system, since the individual time scales still drift with respect to each other. Synchronization of the nodes, in order to limit the maximum absolute offset between them to a small value ε, is a further requirement and expressed by

$$|C_i(t) - C_j(t)| < \varepsilon \ \forall i, j. \tag{1.2}$$

Analogue to [Lam78], events can be correctly sorted, if the value C_i of a clock after a small progress in (absolute) time, μ, is always greater than the value of all other clocks before that step. This means that the occurrence of two events in different nodes $\mathcal{N}_i$ and $\mathcal{N}_j$ can be set in a clear sequence. From the point of view of a distributed system, μ can be seen as the time required for exchanging a message between the two nodes. The condition is formulated as:

$$C_i(t+\mu) - C_j(t) > 0 \; \forall i, j, t. \tag{1.3}$$

In order to ensure the correct ordering of events, a relation between κ, ε, and μ can be established. Applying equation 1.1 to 1.3 gives, $C_i(t+\mu) - C_j(t) > (1-\kappa)\mu$. Since ε was defined to be the maximum offset between any two clocks, the inequality

$$\varepsilon/(1-\kappa) \leq \mu \tag{1.4}$$

together with 1.1 and 1.2 ensures a unique sequential order of events in a distributed system.

The message transmission time and therefore the value of μ is mostly given by the environment a distributed system runs in, i. e. the technical implementation and properties. In contrast, the drift of nodes with respect to some reference (see equation 1.1) and the offset between any two of them (see equation 1.2) can be controlled by using clock synchronization by the means of message exchange. Consequently, clock synchronization is an essential service required by any distributed system depending on an overall consistent order of events of all nodes. Commonly used methods and protocols, which provide this service for the scope of the thesis are discussed in section 3.

There are several examples for a distributed system, which require obeying the rules mentioned above to ensure proper functionality. The following list (based on an overview work on network synchronization [LGHD85]) explains some sample systems, which have an obvious need for clock synchronization:

- Within a *digital communication network*, e. g. a telecommunication network, there are several multiplexing points. These nodes can fulfil different tasks: they aggregate multiple input streams into one high-bandwidth output stream, are able to demultiplex, or extract individual low-bandwidth data streams out of a collection. For all these operations, the whole network has to be synchronous. Otherwise large buffers are required to compensate for varying data rates. Details about synchronous networks can be found in section 3.1.

- In networks, which use *Time-Division Multiplexing* each node is assigned a specific portion of time (slot) during a constant period (cycle) for all nodes in a system in which it is allowed to send data to a network. Since every node has to know when it is allowed to transmit data, synchronization is required. Most real-time networks (see section 3.3) rely on that scheme, which is also used in other digital (broadcast) networks, e. g. satellite or television.

- *Sensor and control networks* (often referred to as fieldbuses) as well as the wireless equivalents may require clock synchronization in order to provide time consistent sampling and coordinated execution of actions. The acquisition of data at a distinct point in time at a large number of sensor nodes allows to give a consistent view of the overall system and summing up of values in order to compare it to a global value, e. g. sampling of current water consumption of households in a city to be compared to the overall water consumption to detect leaks.

Often it is also required to correlate the command execution of actuators. Large systems like particle accelerators demand specific magnetic field profiles to be present at a certain point in time. The overall performance is heavily dependent on the tight timely coordination of the beam direction magnets. The Large Hadron Collider (LHC) at the European Organisation for Nuclear Research (CERN) is one example for such a system, which is covered by section 2.3.

- For *test & measurement systems* the timely coordination of multiple measurement instruments is an important issue. Furthermore, the integration of signal generators and test instruments with the data acquisition allows to build automated test environments. So far, all these instruments were connected and controlled by the IEEE-488 bus, also known as General Purpose Interface Bus (GPIB) [IEE04b]. Modern devices replace this method by LAN eXtensions for Instrumentation (LXI) (see section 3.3.6) allowing to propagate timing, control information, and measurement data over Ethernet at much higher speed over significantly simpler cables.
- By detecting and timestamping the arrival of a common signal by several nodes within a network it is possible to implement *range measurements or position determination*, respectively. For the case of two nodes, e. g. the observation of overvoltage at the end of a power supply line, it allows to calculate the position of a lightning strike using the propagation speed and the timely difference. Similarly, the method can be applied to multiple receiving nodes in order to locate a wireless transmitter by calculating the position from the Time Difference of Arrival (TDoA) of a broadcast signal. For details see section 2.2.
- As for the example of localisation, spatially distributed *phased-array antennas* can also be seen as a distributed system with the immanent need for synchronization. Examples for such an application are world wide connected radar antennas or interferometric measurements in observatories for space observations. Not only the coordinated alignment of the antennas, but also the timely correlation of detected signals is an important information for position calculation. If the antennas are used to transmit signals, the slight shift in phase of the outgoing signal between individual antennas leads to directivity without the need for mechanical components.
- The interconnection of a number of computers for the goal of high performance is also called *distributed or grid computing* and requires synchronization as well, since events have to be ordered. The need is obvious for distributed databases, which have the requirement to order the incoming data change requests as well as for the general application of data synchronization, e. g., distributed file systems, backups, and version control systems. Modern grid computing networks (or in general tasks

for parallel computing) always face the problem of sequencing occurring events, which has to be solved by the methods of synchronized distributed systems.

- In *security* critical systems synchronized clocks can be used to achieve total ordering of messages and therefore prevent fabrication attacks. A Time Variant Parameter (TVP) is used to verify the sequence of messages, which can be – if made available by a clock synchronization protocol – a timestamp from a common clock. More information about hostile attacks on clock synchronization can be found in [Gra10].
- Operations that have to ensure tasks that are crucial for *safety* often depend on guaranteed reaction times. To that end, clock synchronization is used e. g. to provide timely access to the communication medium as well as real-time processing of the required events. Furthermore, stringent communication or reaction timeouts between the involved nodes can be implemented and handled in an easier way if a common timebase is available throughout the considered network.
- Distributed systems and embedded clock synchronization are also part of *scientific research* seeking either for investigation of advantages arising from the interconnection of nodes in a network or for high accuracy in order to improve the performance of the above mentioned applications.

In general, the establishing of a world-wide time distribution system is not limited to technical communication systems, but can also be encountered in other areas. The introduction of some sort of schedule to a certain area of life soon requires synchronization in the sense of a distributed system. The roadmap for trains or the timetable for air traffic are typical examples. Besides the technical examples there are also biological ones, like the synchronous blinking of fireflies described in [Buc88]. The principle was analysed to be used as an algorithm in technical applications [WATP$^+$05]. Several proposals show the usability, especially in wireless sensor networks, where the method allows for tight synchronization while keeping the power consumption low [LE09].

1.2 Basic Terms

This section contains a collection of definitions, which are required for the area of distributed systems and clock synchronization. It is not intended to be a complete list, but will cover the terms used in the scope of the thesis.

Accuracy is the closeness of agreement between a quantity value obtained by measurement and the true value of the measurand.

Precision is the closeness of agreement between quantity values obtained by replicate measurements of a quantity, under specified conditions.

Both terms are standardised in [ISO07] and are important to be distinguished. Especially for the use case of clock synchronization, high precision within a limited number of nodes might be more important, than the accuracy. The first means corresponding simultaneous measurements by different nodes, while the latter is the agreement with a *perfect* clock, external to the system in question. In other words, the accuracy defines the mean of the time or frequency error between a clock and its perfect reference over a number of measurements. The precision is then a measure for the deviation of the error from the mean. [IEE08b]

Inner clock synchronization refers to achieving a common agreement on the clock values between all nodes of a dedicated system. This requirement is formulated in equation 1.2 stating that the offset between any two nodes of the system is below a certain boundary at any time.

Outer clock synchronization links a clock or a system of nodes to an external reference. The connection can be done via a single distinct node, e. g. the master node, or in a distributed way. The goal is always to achieve a global time scale within an enclosed system in order to get a correlation with other spatially separated systems.

The main distinction between the two terms with respect to a distributed system is a logical/structural one. Depending on whether the time reference is considered to be part of the considered network or not, the system performs inner or outer clock synchronization. Mostly, networks with only outer clock synchronization do not make much sense, whereas mere inner operation is quite common [Lis93].

Syntonization is the process of adjusting the frequencies of two processes to be identical. For two clocks, $\mathcal{C}_{i,j}$, this means in particular that they progress at the same rate and return the same duration value for a common external event. This is equivalent to the mathematical formulation $\mathrm{d}C_i(t)/\mathrm{d}t = \mathrm{d}C_j(t)/\mathrm{d}t \ \forall t$.

Delay compensation is the method to cancel out the message exchange delay between two nodes. If the transmission time in a system cannot be neglected, delay compensation is required to get correct measurements of the absolute time, i. e. appropriate accuracy.

Synchronization combines the previous two points and denotes the technique to keep the offset of two clock values, $C_{i,j}$ below a certain boundary (see equation 1.2). In other words, the measurements of the time of a single event at an arbitrary time by the two clocks do not differ by more than the uncertainty, ε.

The three techniques mentioned above denote the basic services required to be performed by a clock synchronization system. Consequently, all synchronization methods mentioned in chapter 3 implement these services using different approaches to reach the desired consistent time on all participating nodes.

Timescales are used to define the begin (time value zero) and the divisions (scale) of time. The primary timescale is called international atomic time (TAI) and advances with the rate of the international system of units (SI) second. From TAI the Coordinated Universal Time [ITU02] (UTC) is derived by adding leap seconds to

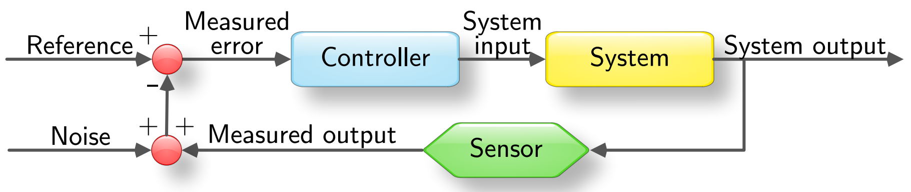


Figure 1.2: Single control loop with measurement noise

compensate for the slowing earth rotation. Nevertheless, both scales advance at the same rate and just differ by full seconds. Many technical systems requiring a timescale define their own in order to specify the start value, e. g., GPS, NTP, PTP.

Timestamping is the process of copying (freezing) the current value of the clock $\mathcal{C}$ on occurrence of an (external) event. The generated copy of $C(t)$ at an instance in time is then called *timestamp*, $\mathrm{TS}_{\mathcal{C}} = C(t)$. Often, the timestamp and some sort of event identifier (e. g., sequence number, hash code) are stored together, in case it is necessary to be able to link the timestamp to the event later on. For details see chapter 1.4.2.

Control loops consist of three basic elements shown in figure 1.2. The controller, the system, and the sensor form the well known closed loop of the control theory. For network-based control loops, the controller may be split up into the actual controller, calculating the value for the system input and the actuator, which generates the system input and is attached to the system. In digital communication systems, all components work as sampling systems, meaning that the values are exchanged between the components in a time discrete way (messages). Consequently, for digital networked control systems the considerations of chapter 1.1 concerning distributed systems apply.

Sensor and control networks are communication systems that incorporate sensors for data acquisition, actuators for influencing the considered system, and one or more (distributed) appropriate controllers. All are interconnected in form of a distributed system. Characteristic properties of such networks are a large number of sensors, which are used to generate a consistent overall picture of a specific system and relatively low processing power per node. Especially the aim for a reliable overview requires exact timing of the recorded measurement data.

1.3 Clock Synchronization in Packet-Oriented Networks

From a historical point of view, the evolution of networks first began with dedicated links between all nodes [Kle10]. This method only allows a small number of participating clients, as soon the effort to interconnect a new member to the existing system becomes unmanageable. If only unidirectional links are considered, the number of required links, M, for n nodes calculates to $M_{\text{uni}} = n \cdot (n-1)$. Even if, more realistic, bidirectional links are taken into account, larger networks are still quite inefficient due to the growth of

$$M_{\text{bidir}} = \sum_{i=1}^{n} (i-1) = \frac{n \cdot (n-1)}{2}, \tag{1.5}$$

which is still in the same order as the unidirectional version of the network, namely $\Theta(M(n)) = n^2$.

In order to build larger networks, the technique of *circuit switching* was introduced. In such a network dedicated connections are still used, but a central switching element establishes the direct link between two nodes on demand. In the optimal case every node only needs to be connected to the central element, which then reduces the order to $\Omega(M(n)) = n$. For large networks it is unlikely that all nodes can be connected to one centre point. To solve this issue, multiple switching elements are introduced, which are interconnected by dedicated links as any other node. The number of links between the elements has to be chosen in a way that the maximum number of nodes communicating with nodes of the other switching element can be assigned a link. Therefore n is the lower bound for the number of links and only valid for relatively small networks, larger versions require the afore mentioned additional connections between the switching elements. Modern digital versions of the network allow virtual links on one dedicated line using TDM. For details see chapter 3.1. This type was and still is used in telecommunication networks.

The next evolutionary step for communication networks was the introduction of *packet switching* and its special version *cell switching*. In contrast to the previously mentioned network types, packet switching allows more efficient use of the individual link capacity between nodes and/or network switches, since there is no need for static bandwidth assignments. This is due to the fact that in packet switched networks, every packet can (theoretically) take its own route through the network, which also increases fault tolerance based on redundant links. Additionally, cell switching ensures that all packets of a virtual connection (or any other common criteria) are routed on the same path. Routing is therefore semi-static but still gives the flexibility of the packet-oriented approach mentioned above.

Switching packets also allows to run many different services or user requests over a single network type taking advantage of bandwidth aggregation. This evidently increases the efficiency due to the flexible usage of resources at the expense of losing a dedicated connection, which can then only be emulated on higher protocol layers. This emulation differs from a real circuit switched connection mainly in its unknown delay behaviour.

1.3.1 Transmission Delay Boundaries

Concerning distributed systems as defined in chapter 1.1, the most important issue is the afore mentioned undefined packet transmission delay. As shown, the time for transferring data, μ, influences the achievable precision for the ordering of events. Although, for

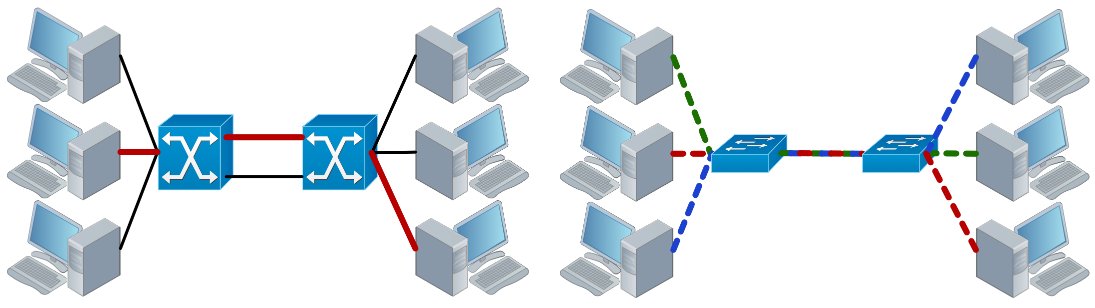

(a) Circuit switched network (b) Packet switched network

Figure 1.3: Network types

any experiment with a finite number of samples (not considering packet loss) there is an upper bound of the delay, it cannot be assumed that this is the theoretical upper bound. Moreover, the significant gap that exists between any experimentally determined and the minimum delay makes the bound impractical for the purpose of synchronizing n clocks, since all deterministic algorithms that depend on an upper bound for the maximum communication delay cannot achieve a maximum deviation better than ${}^{(\mu_{\max}-\mu_{\min})}/_{(1-1/n)}$ [CF03, WL88].

An efficient synchronization method for packet-oriented networks therefore should not assume any upper boundary on communication delays in order to work correctly. Since there is also a non negligible chance that packets can be dropped, all techniques explored in this thesis for synchronizing network nodes in packet-based distributed systems are able to cope with unexpectedly long (infinite) delays (equivalent to packet loss).

1.3.2 Packet Relaying Effects

According to the International Organisation for Standardisation (ISO) Open Systems Interconnection (OSI) Basic Reference Model [ISO94] the lower three layers of the reference stack are required to deliver packets in a network with an unlimited number of nodes. Layer 3 is the last which is involved in path decisions and provides routing and relay independent functionality to higher layers. Consequently, the layers for the physical connection, data link and network layer have to be considered to contribute to the delay of a packet, when travelling from the sending to the receiving node. The overall system is illustrated in figure 1.4.

Depending on the layer where the relaying of packets is done, the elements are called *bridge* (layer 1), *switch* (layer 2), or *router* (layer 3). Each of these network elements, which might be placed between two end nodes, can add unknown jitter to the packet transmission time and contribute to the absolute value. Therefore each of them seriously influences the clock synchronization performance for a distributed system. Further, the delay is unbound and thus requires special consideration in the event ordering algorithm, as mentioned before. The details of the delay/jitter performance of the individual protocol layers are discussed in chapter 4.

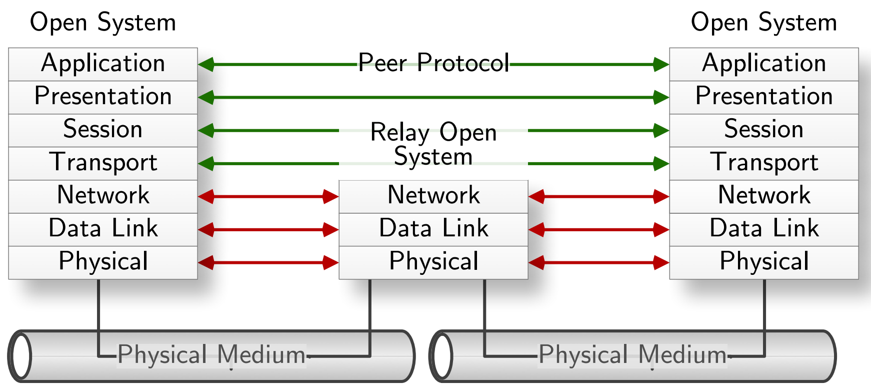


Figure 1.4: Communication involving relay open system according to the ISO OSI Basic Reference Model

It is important to take into account that switching and routing are based on addresses stored in the packet to be delivered and therefore require processing of the data contained. Since these addresses have to be read out, processed, and then taken as the basis for a packet path decision, there is not only the propagation delay through the network element but also some processing delay. The latter can be different for each packet since it might be address or data dependent. While layer 2 is normally designed to allow for constant processing delays, typical protocols on layer 3 even feature variable size headers and therefore hinder steady packet forwarding by principle since the required information arrives at different times.

Two principles are known to be used in relaying elements. On the one hand *store-and-forward* technology receives the complete packet into a buffer, then processes the required information for routing path decision, and finally forwards the packet. On the other hand *cut-through* methods only buffer the part of a packet's header which is required to select the routing path and then immediately start forwarding the packet. This method significantly reduces the residence time of a packet in the relaying element. The disadvantage is that the Frame Check Sequence (FCS) is commonly stored at the end of a packet and therefore checks cannot be done before sending the packet. As a result, also damaged packets are relayed. Some implementations use a mixed strategy, which takes cut-through as long as the number of damaged frames stays below a certain limit and switches to store-and-forward otherwise.

1.4 Hardware Support for High-Precision Clock Synchronization

Besides a protocol for exchanging accurate timing information, high-precision clock synchronization requires dedicated hardware support. This is due to the fact that software implementations can only be as accurate as the underlying hardware processing, without taking advantage of statistical effect by special algorithms. The latter can as well be applied to hardware supported systems to further increase accuracy. In this thesis the term *dedicated hardware* support refers to some additional logic, which is specialised in processing of operations for clock synchronization. This can be a special real-time clock, high-precise timestamping, or immediate event processing for example. Contrary, common general purpose processors are able to run synchronization software but cannot react to events very fast (e. g. interrupt processing latency) and therefore hinder accurate timestamping. Also the other way round, generating events at exact moments in time is hindered by task switching delays within micro controllers or a Central Processing Unit (CPU).

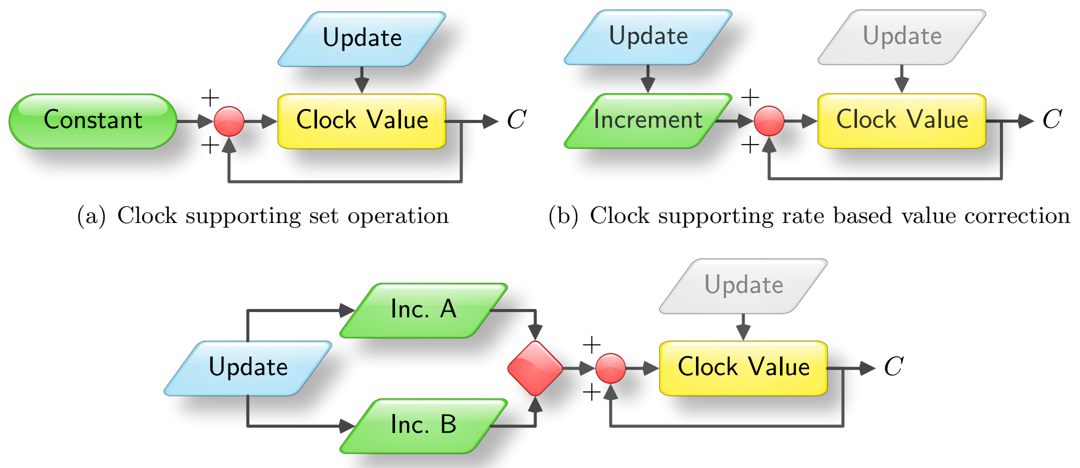
The key issue with a generalised approach is that state-of-the-art processors require several clock cycles to make a context change. This originates from the necessity of saving the current processor state (registers) and then proceed at a different program (position). Also pipeline architectures within a modern CPU delay the actual reaction to an asynchronous event, because the pipeline of pre-loaded instructions has to be flushed and filled with the code to handle the exception. Since a number of processor instructions are always required to change the context, even with clock rates in the GHz range, the delay can reach several nanoseconds. If further OSI layers are involved, an operating system or similar software requires even more instruction cycles and can therefore shift the delay even to the micro- or millisecond range.

1.4.1 Required Elements

The obviously indispensable element for hardware support is a timekeeping element, the clock $\mathcal{C}$. This central element has to be reachable by all functions mentioned in the following. In other words, the required time to determine the current value C of the clock $\mathcal{C}$ has to be negligible compared to all other processing times in the system. The term distributed system consequently does not apply to the considered hardware subsystem and avoids the utilisation of special measures for message exchange delays. The sequence of events within the dedicated hardware support block is given by a common clock signal with pre-compensated propagation delay.

The clock has to fulfil a number of prerequisites concerning C and the available operations on this value. Firstly, C should be strictly monotonically increasing, so that for any two events, which do not happen at the same time, $\mathcal{C}$ shows a different value ($C(t + \varepsilon) > C(t)\ \forall t, \varepsilon > 0$). This avoids some time values being used twice for non concurrent events due to a step back or a too slow progress of the clock. ε therefore has to be smaller than the shortest time between two events that should be distinguishable and gives the highest reachable precision. Secondly, in order to fulfil the just expressed premise, a controller is required to adjust C according to a master that has to be able to use special functions. A simple operation like *setting* the value of the clock (as illustrated in figure 1.5(a)) is not sufficient, since this would cause reoccurring time values in case the clock is ahead of time and has to be corrected.

In order to overcome this problem, many modern implementations utilise a *rate* setting approach as shown in figure 1.5(b). This method uses an increment value, which is added to the current time of $\mathcal{C}$ with every tick of an oscillator. By increasing or decreasing



(a) Clock supporting set operation

(b) Clock supporting rate based value correction

(c) Clock with two selectable increments

Figure 1.5: Structures for hardware clocks

the increment the progress of C can be speeded up or slowed down. Consequently, any inaccuracy of the node can be compensated by the controller. The obvious advantage over the previous is that C fulfils the requirement of a strictly monotonic increase even in case the node is ahead of time and has to be corrected back. This structure is called Adder-Based Clock (ABC). Figure 1.5(c) shows the next evolutionary step: the *amortisation*-based clock, which is an enhanced version of the ABC. This technique uses two values for the increment where the additional one can be set temporarily for a specific period of time. It allows to concentrate the required (possibly large) frequency change to compensate for a time offset to a specific period. Thus, it is possible to decouple e. g. round-trip delay measurements, which require a calibrated frequency, from the offset compensation.

With the functionalities mentioned above, the clock is able to keep the time and can be adjusted by a controller. To build a closed control loop, it is necessary that the hardware clock is able to timestamp external events. In case of distributed systems where message exchange is required, a function block called *timestamper* is responsible for generating a timestamp signal on arrival of a network packet or message. This signal is then used to generate a copy of the current clock value. By comparing this timestamp to the time transmitted in the message, the current error of the local time value can be determined and used to feed the controller for adjustments. The principle is shown in figure 1.6. The timestamping functionality should be available separately for incoming as well as outgoing messages to support delay compensation schemes. The latter require bidirectional communication in the non preconfigured case and therefore hardware, which is able to timestamp messages on the ingress and egress path.

Besides the basic building blocks, the timekeeping clock itself and the timestamper required for operation in distributed systems, a complete clock synchronization core (e. g.,

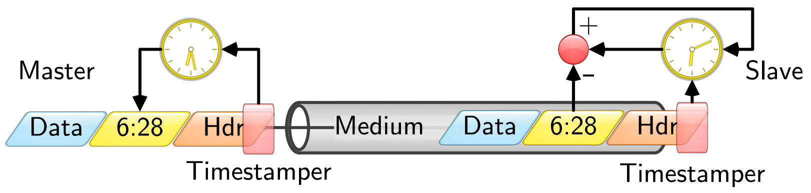


Figure 1.6: Function principle of timestampers close to the physical medium on the egress path (master) and ingress path (slave) to compare clock values of two nodes.

chapter 5) should also provide additional functionalities to make use of the accurate local time. The most important issue with these operations is that they are executed in a deterministic way. In other words, the time passing between the execution and the instant in time the clock reaches the scheduled value should be negligibly small and fixed. Commonly required functions to benefit from a highly accurate local time in hardware include:

- *Periodic timers* are used to generate an external signal in fixed equidistant time intervals. The most prominent and widely used representative of this class is the 1 Pulse Per Second (PPS) signal. This type of signal is used in many synchronization related systems like atomic clocks (e. g., caesium (Cs_{133}) or rubidium (Rb)), Global Positioning System (GPS) receiver, or the Inter Range Instrumentation Group (IRIG) standard time code B [Tel04].

- *Triggers* are used to set a signal (generate an edge) at a specific time value. The signal can be for instance an Interrupt ReQuest (IRQ) signal in a computing system to exactly trigger certain procedures. Furthermore, it can be used to transfer the occurrence of a special instant of time to any other external device.

- Many applications, like synchronous networks (see chapter 3.1) require a syntonized *frequency output* at each node in order to perform Time-Division Multiple Access (TDMA) on the network medium. Since this type of network also has some higher-level absolute event ordering, synchronization is required as well.

- Similar to output signals generated by triggers and periodic timers of the clock synchronization core, some input signals should allow *timestamping of external events* with the local time. This feature allows for example the synchronization of the core to some external time source, like the 1 PPS of a GPS receiver.

- For detailed and/or distributed network traffic analysis, *packet timestamping* for arbitrary messages is an essential function. Since it is inefficient to draw and transfer a timestamp for every incoming or outgoing packet to the software, the hardware timestamper should perform some pre-filtering and buffering. Consequently, there has to be the possibility of configuring the timestamping unit to produce timestamps for timing messages and in addition for the type of packets to be traced in the network.

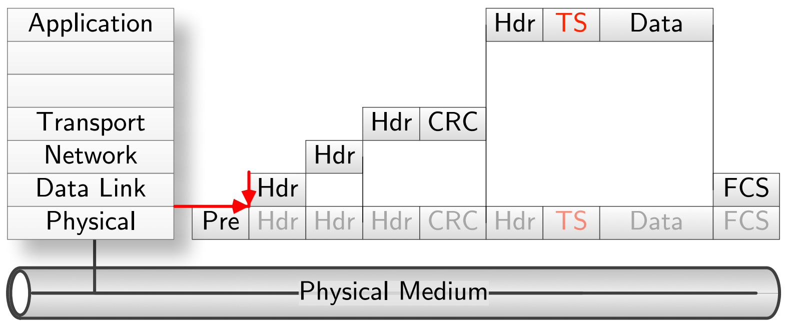


Figure 1.7: Illustration of layered assembly of an exemplary physical layer frame, using headers (Hdr), CRC, FCS, and the frame preamble (Pre). The arrows mark the last physically available interface on Ethernet, the MII, and the earliest (reasonable) timestamping point.

1.4.2 Transfer of Timestamps and the OSI Reference Model

In order to avoid as many temporal uncertainties (jitter) caused by varying processing times in the different network layers, timestamping is desired to be as close to the physical medium as possible. This requirement on the other hand causes problems with respect to the layered approach of the OSI Reference Model. Since the task of synchronization is not foreseen in the standardised representation (as well as the most commonly used lower layer protocol implementations), it was implemented as an application function (peer protocol) on layer 7. Consequently, there is a need to transfer the timestamps from the layer of origin there. The induced difficulties occur from the fact that the ingress and egress path for messages does not allow to transfer timestamps in parallel (linked to the message).

The problem induced by the layered approach of the OSI reference model is that clock synchronization actually effects several layers. While the physical layer is desired to be the source of timestamps (due to the mentioned jitter issues), the synchronization of nodes is a global networking issue. Consequently, the transmission of timestamps in a network of a certain size requires the functionality of layer 2 and 3 unless an operation with a limited number of nodes can be accepted.

Several issues related to the protocol implementation on layer 7 and the need for accurate timestamping are illustrated in figure 1.7. The timestamp, $\mathrm{TS}(t)$, can be transferred to the synchronization engine using two methods, in-band or out-of-band. While the first technique encapsulates the timestamp into the frame to be sent or received, the second uses a separate path (e. g., First In – First Out (FIFO) buffer) to make it available to the application. The latter suffers from the fact that the timestamp has to be associated to the corresponding frame by other means (e. g., comparison of CRCs, sequence numbers). This is necessary because of the possibility that any layer between the gathering of the timestamp and the usage in the protocol software might reorder or even drop the frame. In this case the chronological order would be interrupted and cause false association resulting in an additional offset for the synchronization. For the egress path this method also requires the sending of a second packet (with the actual time the frame left the node), since the timestamp for an outgoing frame is only known after its transmission.

Drawing the timestamp close to the physical layer has the advantage that there is (at least for the case of Ethernet and many other transmission standards) a fixed temporal relation between the start of the frame and the timestamping point. The resulting timestamp is then identical for all devices and jitter is only subject to physical effects and not related to varying header length of the frame. Nevertheless, if the transmit timestamp is transported in the application layer protocol data, the following issues can hinder the correct insertion at the time it is generated.

For the egress path the insertion or update of the transmit timestamp, $\mathrm{TS}_{\mathrm{tx}}(t)$, can cause several issues that have to be considered to ensure correct functionality:

- The calculation of a CRC for the application data that is stored in front of the actual data is impossible due to the sequential and non-interruptible data transfer on the medium. Furthermore, since this interface to the medium is located already after the Media Access Control (MAC), there is a strict time behaviour and additional modules are not allowed to delay data transfer for processing. Since the hardware module has to draw a transmit timestamp at the beginning of the frame, the CRC would have to be recalculated for the unknown data part. The cyclic properties of the checksum allow to modify the CRC by adding the part for the (now known) timestamp to the existing value and then inserting the timestamp. Nevertheless, this requires knowledge of the pre-existing value of the timestamp (e. g., could be solved by choosing it to be zero) and the one of the CRC. The latter requires a transmission delay of the bit-time times the length of the CRC, which might not be allowed by the specification of the interface from layer 2 to 1. For more complex (cryptographic) checksums (e. g., MD5, SHA-1), this method fails, because they cannot be updated without knowing the actual data.

- If outgoing frames have an appended FCS it requires an update to take the actual value of the timestamp into account. Again, the frame can either be delayed by the size of the FCS times the bit time to make an incremental update, or the value can be completely recalculated for the whole frame during transmission, since the final value is known at the time it has to be inserted. For the latter technique, the timestamping functionality has to be integrated into the egress path of layer 2, because a recalculation of the CRC value on the interface by a separate unit corrects the FCS for bit errors happening between layer 2 and the intermediate timestamping unit to layer 1.

- The most challenging part for inserting a timestamp into the application layer data is to cope with varying header lengths. The hardware processing unit at layer 2 has to know the position where the timestamp should be updated. Due to varying headers (e. g., packet type or link layer encapsulation) the position might change. In order to recognise the timestamp independently of these issues, the timestamping hardware has to be able to calculate the actual position out of the leading headers.

This requires knowledge and basic parsing capabilities for upper layer protocols and introduces compatibility issues. The protocol parsing ability is (at least partly) a redundant implementation of the corresponding higher layers. Consequently, they have to match the realisation there to ensure correct functionality. The approach is quite complex and heavily violates the layered approach resulting in difficult realisations or limited universality.

For the ingress path almost the same issues apply, except for the fact that the timing of the incoming frame is not critical and therefore, more complex processing even on temporarily stored frames can be done for the recalculation of the CRC. Still, the complexity of parsing upper layer protocols and the fact that the application data area must foresee a field to be updated with the receive timestamp, remains. Simply appending the timestamp to the frame at layer 2 is normally possible, since the checksum is valid only for a given length and therefore not violated. Nevertheless, there are maximum frame lengths, which might be exceeded unless the necessary length for adding the receive timestamp is reserved on the sending side.

Summing up, drawing the timestamp at the layer close to the medium and propagating it to the synchronizing application according to the OSI Reference Model is not a trivial task. The chosen method has to fit the application layer protocol as well as the implementation of the intermediate network layers. This may involve the network stack of an operating system and therefore limits the possibilities for solutions.

1.5 Contribution of the Thesis

For the mentioned application areas, so far a number of dedicated interconnection solutions were used. For example automation tasks were covered by several fieldbuses [Sau10]. Nevertheless, for integration, availability, and of course also cost reasons, these dedicated solutions were replaced by Ethernet-based networks. The latter suffer from the fact that Ethernet was not designed for time sensitive use cases. Thus, a number of real-time enhancements (see chapter 3.3) were designed.

The present work analyses the historical approaches based on synchronous solutions and draws the path to nowadays wide spread packet-oriented (Ethernet) networks. More and more applications demand for higher clock synchronization accuracy from a technology that was not designed to fulfil this purpose. Therefore, the thesis covers the enhancement of Ethernet-based clock synchronization under the pre-condition of practicability and conformance to the relevant standards. Obviously, there could be more efficient solutions than the developed one, but not in conformance to the standard or under the prerequisite of commercial-of-the-shelf physical layer devices.

The detailed characterisation of the prerequisite, i. e. Ethernet, points out the influencing factors on clock synchronization. Furthermore, several possibilities for highly accurate timestamping methods, the necessary base, are analysed for their usability in Ethernet-based solutions. The combined phase/frequency estimation method turns out to allow for extremely accurate timestamping at the edge of the physical performance of the utilized hardware elements.

Several aspects regarding oscillators, which are used as the clock source for the transmission as well as the timekeeping, are investigated for better understanding of the origin of inaccuracies. The work develops models to simulate the jitter sources and characterises the influence of the oscillator noise parameters on the performance of the overall synchronization system. Especially the interdependency between different system parameters, e. g. oscillator stability, synchronization interval, or timestamping resolution, are formulated and verified by measurements.

The overall goal of this work is to determine the limiting factors of current Ethernet-based clock synchronization systems by appropriate characterisation. This helps to find out about deficiencies in existing solutions that hinder higher accuracy. To this end a system-wide view on clock synchronization is necessary to cover the interdependencies of the involved hardware and software elements that influence the finally achievable accuracy.

The conclusion out of the analysis is then utilised to develop efficient structures for increased synchronization performance in the sub-nanosecond range. The investigation of relevant issues is started at the necessary clock source and the network link itself. Since all other logic elements depend on these two elements the decision for a specific implementation of them decides on the maximum achievable accuracy of a synchronization system.

2 Motivation of the Problem

Besides the general categories of applications for high-accuracy clock synchronization mentioned in section 1.1, this chapters exemplarily explains some applications that require precisely adjusted clocks. The selection follows the contribution of this work to the use cases that occurred in the outlined projects. The stated accuracy can be impressively seen in the given localisation accuracy, which is directly dependent on the synchronization performance of the receiving access points. Nevertheless, also distributed sensor networks with a large number of nodes do require relative high accuracies, as described in the following.

2.1 Remote Metering using Power Line

The investigated techniques to precisely synchronize distributed sensor nodes were integrated in an embedded processor described in section 6.1.3 for a large-scale remote metering application. A promising approach for low-cost clock distribution in large-scale systems is the usage of Power Line Communication (PLC) systems. For this type of communication medium clock synchronization is a crucial issue, not only because the PLC network itself requires synchronized clocks for maintaining time-sliced communication, but also the target application of remote read-out of domestic supply meter demands accurate synchronization.

In the lower levels of a hierarchical PLC system attention has to be paid to the special properties of the network with respect to varying network topologies, short message length and the like. Such an architecture has the advantage that using the existing wiring of power lines a communication infrastructure can be established in order to manage energy consumption more efficiently and inexpensively.

One issue of such networks is that fast log-on and log-off of nodes travelling from one access point to another has to be ensured. The demand originates from the fact that nodes and groups of nodes may disappearing on one side of the network and re-appear at different points – most likely at a different hierarchy as well. This is caused by the usual way of energy suppliers to switch whole net-groups from one transformer station to another, thus changing the logical hierarchy of the PLC network significantly.

Furthermore, interesting Supervisory Control and Data Acquisition (SCADA) applications that determine the loss (e. g. of water, electricity, or gas) depend on a central meter and an accurate sum of the data gathered from a high number of end-user meters. Using this communication infrastructure metering data can be retrieved without sending a physical person to read out the meter. Additionally, since the sum has to match the central amount it is necessary to synchronize the distributed meters in order to acquire the reading at single point in time to make the sum a meaningful value. Only

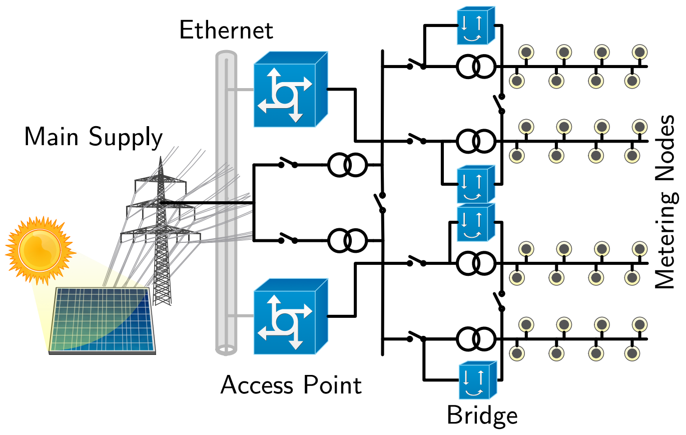


Figure 2.1: Remote metering network architecture using PLC. The switches allow failover or load balancing within the network, but can significantly change the topology. Similar structures can also be used for other supplies (e. g. water or gas)

then a global status of the system can be achieved that allows to determine the loss by subtracting the sub-meter sum from the value determined at a central point. This way, the detection of energy theft is done by advising all nodes in advance to record their meter values at a certain, predefined point in time. Thus, manipulations and energy thefts can be detected and localised more easily.

The realised implementation of the large-scale system with cascaded clocks reaches an overall accuracy of 200 µs in an area of several kilometres [GLTK06]. The solution tackles the accuracy at steady-state as well as the impact of transient effects like a coordinated, fast, and jitter-free power-up of the whole network. Figure 2.1 illustrates the network concept for the mentioned application scenario. It is split into two parts: an access network, which is based on Ethernet and the PLC structure to remotely access the domestic supply meters.

2.1.1 Access Network

One important feature of the evaluation hardware that is particularly utilised in the presented use case is the technique for amortisation (see section 5.2.2). The synchronized nodes generate a frame clock for the communication slots on the PLC network. Therefore, it is important that a node in the power-up phase does not mess up the communication. Further, the parameter estimation and lock-in for the power line is rather time consuming. Thus, periodic timers, which are used to generate the frame clock on the PLC network, are compensated for steering values that exceed a certain limit. That way, a fast adjustment of the absolute time can be achieved, while the output frequency for the PLC frame clock can be kept constant.

2.1.2 Power Line Network

Besides the central embedded processor for handling the (communication) protocols, the PLC physical layer communication is implemented using a separate Application Specific Integrated Circuit (ASIC), which contains a signal processing unit for mixing, filtering, up- and sub-sampling as well as synchronization detection. A Finite State Machine (FSM) controls all operations related to transmitting and receiving data. It implements a master-slave-based, time-slotted communication scheme. In order to detect synchronization events properly, the complete module performs an energy normalised correlation of complex synchronization sequences on the equivalent complex base band. All nodes synchronize on bit-level to the master clock. The slotted communication is established by a configured slot length, regular packets indicating the start of a slot and additional absolute time information.

Since the PLC physical layer communication is handled by a separate, specialised Digital Signal Processor (DSP), only the frame-clock events, which are automatically generated by the state machine for PLC communication, can be detected by other modules of the node. Consequently, these events are timestamped by the synchronization hardware block, the Clock Synchronization Cell (CSC), which also generates timestamps in the Ethernet case. This has the advantage that the format of timestamps is, within the PLC network as well as in the backbone networks, fully compliant to the Institute of Electrical and Electronics Engineers (IEEE) 1588 format. Nevertheless, a clock synchronization using pure IEEE 1588 is not directly applicable to PLC for a number of reasons:

- The communication delay between power line master and slave is compared to the delay in the opposite direction highly asymmetric. IEEE 1588 is not able to deal with that form of delay, since the standardised round-trip delay measurements can only take the mean of both directions into account.

- Secondly, IEEE 1588 is in general not able to synchronize clocks with a high accuracy in networks where the network delay may change at any moment due to changing repeater levels caused by network topology changes (e. g., load balancing in electricity distribution grids).

- Finally, even the relative low number of one synchronization packet per period, sent out by a master to a network with potentially hundreds of nodes can overload the limited bandwidth resources of a power line network. This is due to the interconnection using a shared medium that requires signal repeaters, which further consume time-slots. The available bandwidth therefore decreases with the number of repeater levels.

The issue that the system is in general not capable to provide symmetric communication channels in terms of latency generates issues with the assumptions made by Precision Time Protocol (PTP), which relies on a symmetric channel. Therefore, if PTP is used without adaptations this would result in a non-negligible error. In order to compensate for asymmetric transmission delays on the power line, resulting from delays of the slotted communication scheme, predefined correction values are applied.

The message transportation costs from a master are a further issue. Since in power line the upstream communication is disproportional slower than the other direction, the implementation simulates the IEEE 1588 Delay_Req and Delay_Resp messages by supplying predefined delays – calculated from the asymmetry factor and the slot delay – to the synchronization stack. The functionality is implemented by a special kernel driver module, which keeps track of the last 15 frame-clock events. Each time a telegram that contains an absolute time information from a master is received, it is matched with the appropriate clock event. This process is based on the repeater level information gathered from the network layer, which also assures that each packet is just received once. Every time a pair of absolute time and corresponding frame-clock event has been assembled it is passed on to the user process calculating the actual clock synchronization algorithm. Consequently, the line delay, which mainly consists of multiple slot delays, can be compensated. Since the accuracy requirements are in the microseconds range, there is no need for compensation of the line delay on the power line. Therefore, a number of transmissions can be saved in favour of user-payload.

The power line synchronization method also takes advantage from the fact that all messages are aligned to the system-wide frame clock as well. Thus, the adaptation layer also uses the occurrence of these messages to generate synchronization events. The drawback that no absolute time is sent with every message, can be tackled with a prediction function of scheduled frame-clock events, which is defined via the communication speed. The arrival of a timing packet containing absolute time information is used to adjust the prediction algorithm.

Tests with an evaluation board equipped with the mentioned embedded processor showed that the performance over four repeater levels – which is assumed to be the required number for a field test – can fulfil the requirements for accurate metering. [GLTK06]

2.2 Wireless Position Determination

One obvious trend in consumer as well as industrial electronics is the support of wireless network connectivity. Many personal devices provide connectivity for IEEE 802.11a/b/g. The reason for that is rather simple: One may integrate smart embedded devices into an existing network like Wireless Local Area Network (WLAN) seamlessly and benefit from access to various applications from any point. Further, not only the seamless integration but also the reduced cabling costs and the advantages of mobility are reasons to use wireless technologies. The latter argument is especially important for factory automation and industrial applications.

In that context, the history of automation networks has shown the evolution from purely dedicated fieldbuses to Ethernet-based technologies. The real-time networks mentioned in section 3.3 are almost all an example for this trend and just represent a subset of the available technologies. This step of using techniques, which originate from the office information technology domain, has nevertheless not taken advantage of wireless networks. However, the obvious increased flexibility of wireless approaches opens new possibilities for applications in automation, measurement, control, and communication.

Actually both, wired and wireless networks, have their pros and cons. The biggest advantage of having a wireless network is that one does not need to install cables and can move within the supply area without worrying about necessary infrastructure. Further, it is possible to dynamically add nodes to the shared medium. Thus, the network offers some flexibility when it comes to the number of nodes that have to be supplied with a network connection. Certainly, this is only valid within a limited range due to the fact that all clients share a limited bandwidth on one access point.

Wired networks on the other hand can provide very high data rates, which wireless networks cannot support because of bandwidth limitations and unavoidable transient disturbances on the channel. Furthermore, it is easier to provide certain security measures due to the fact that access to the network can be controlled by restricting physical access to the medium.

One missing key feature of standard wireless technology is the ability to localize clients. Whereas in wired topologies the location of a network client can be at least roughly estimated by monitoring the switch port and/or by using the knowledge of the position of the wall outlet, wireless network clients can be situated in any position surrounding the access point. This makes it impossible to implement safety features, such as the control of robots only if the Human Interface Device (HID) is in visible range to avoid unwatched operation of the device. Also roaming between access points and in advance preparation of real-time constraints before hand-over are very interesting applications for the use of positional data [GLM08].

Although there are a number of technologies for wireless sensor networks, the use of IEEE 802.11 [IEE07] compatible technology has the advantage of the simplicity it can be deployed and used with. The big incentive for WLAN users is the known technology from the office infrastructure that it can be seamlessly integrated with. For such networks a number of methods [LGS07] are available to achieve localisation. They all suffer from different deficiencies (e. g., low accuracy, dependence on special hardware, required (re-)calibration) that do not qualify them for large deployments e. g., in factory automation. Unless the mentioned ones, the presented TDoA-based approach is a service of the network infrastructure and thus can fulfil the localisation operations without the help of the node to be detected. Since there is no need to modify the mobile devices it is well suited for installations with high number of nodes (cost reasons), security issues, or continuous changing environments (calibration issues).

2.2.1 Method for Localisation

The basis for the TDoA approach is to measure the differential signal propagation delays from a mobile client to geographically distributed access points with fixed and known position. The principle of this method is illustrated in figure 2.2. It is the reversed method of known global location systems like GPS or the former positioning, navigation, and timing system, LOng RAnge Navigation Revision C (LORAN-C) [Fra83]. Since the time difference cannot be observed by a single station, all receivers have to draw a timestamp on detection of the incoming signal from the mobile node. A central node then collects all available timestamps to a set and calculates the time differences of

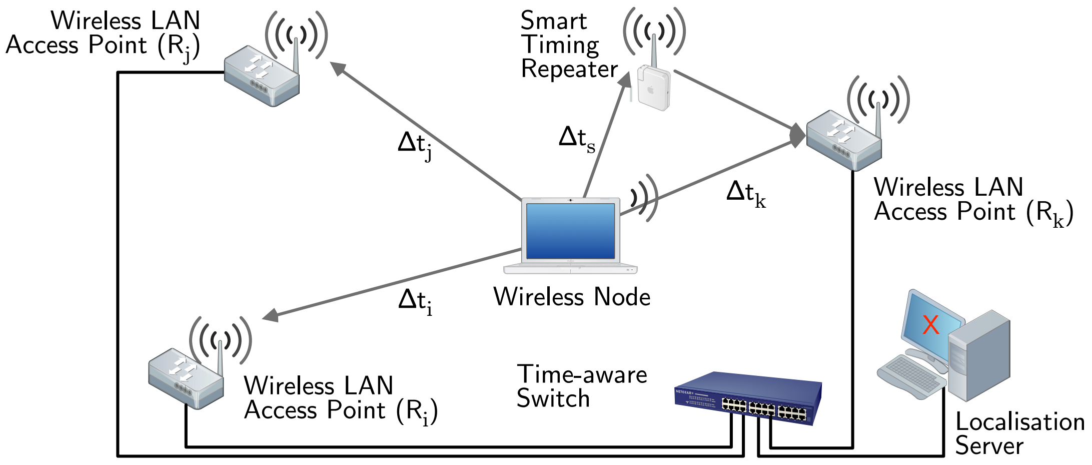


Figure 2.2: Method for 2D localisation

arrival between the individual receivers. Thus, it is required to synchronize the access points, which can be done via an IEEE 1588 highly accurate clock synchronization using dedicated hardware support as described in chapter 6. A number of methods [Sch72] can be used to calculate the emitter position from the calculated TDoA values. If there is no error in the measurements or the locations of the sensors, the emitter can be located without error or ambiguity. [Sch96]

When seeking for a wireless network, the usage of several access points, which have to be cabled, is no desirable strategy. Therefore, the concept of Smart Timing Repeaters (STRs) is introduced. These, in the ideal case power independent devices, are placed on distinct positions in the area of interest. Once such a device captures a message dedicated to localisation, it repeats this messages after a fixed timeout period. Thus, the delayed message can be received by the main access point of the network as illustrated in figure 2.2. Using the known internal delay and the position of an STR the actual propagation delay from the client to the STR can be determined. As long as the processing delay within the STR can be kept constant – hence, is known to the central data collection point – there is no need for timestamping on the device itself.

In case the previous mentioned requirement cannot be met, the issue of syntonization (adjusting only the frequency, no offset correction) has to be tackled. The problem can arise if the oscillator frequency is prone to changes and consequently, the processing delay on the node can vary. Assuming a typical processing delay of some milliseconds, the resulting error is quite small. Still, for the high accuracy required for the application of localisation, syntonization might be necessary. It is much easier to implement than a full synchronization because the STR only has to measure the interval between two synchronization packets from an IEEE 1588 master. Therefore, the communication is one-way and there is no need for path delay measurements. Further, the issue of changing path delays (reflections, signal barriers) can be avoided by using short burst of synchronization packets. This will increase the probability that two consecutive packets experience the same physical parameters on the channel, hence, increasing the syntonization accuracy.

After the system has determined at least three propagation delay differences, the position determination is based on the well-known hyperboloid position determination algorithm. Under the prerequisite that the emitter with the position $\{x, y, z\}$ has direct line-of-sight to all receivers, the signal experiences equal propagation velocity, v, form the mobile node to the fixed stations, and the position of all receivers $\{x_{\mathrm{R_i}}, y_{\mathrm{R_i}}, z_{\mathrm{R_i}}\}$ is known, the measured ingress timestamp differences, $t_{\mathrm{R_i}} - t_{\mathrm{R_j}} = \Delta t_{\mathrm{i,j}}$ can be expressed via the distance between the receivers,

$$d_{\mathrm{i,j}} = v \cdot \Delta t_{\mathrm{i,j}} = \sqrt{x^2 + y^2 + z^2} - \sqrt{(x - x_{\mathrm{R_j}})^2 + y^2 + z^2} \text{ and} \tag{2.1}$$

$$d_{\mathrm{i,k}} = v \cdot \Delta t_{\mathrm{i,k}} = \sqrt{x^2 + y^2 + z^2} - \sqrt{(x - x_{\mathrm{R_k}})^2 + (y - y_{\mathrm{R_k}})^2 + z^2}. \tag{2.2}$$

Except for special setups, which should be avoided, the three stations $\mathrm{R_i}, \mathrm{R_j}, \mathrm{R_k}$ form a plane. Consequently, in all other cases, it is always possible to find a Cartesian system of coordinates that has the first station in the point of origin, the second on the x-axis, and the third within the $\{x, y\}$-plane. This allows the rather simple presentation of the equational system built by equation 2.1 and 2.2.

If it can be assumed that $\Delta t_{\mathrm{i,j}}$ is not zero, which would imply that $x = {x_{\mathrm{R_j}}}/{2}$, the equational system can be solved towards only one parameter [Fan90]. In order to fix the full position, additional information is required. The latter can be gathered either from the time difference to a further station or the z-coordinate can be derived by other means (known or on a surface, e. g. the earth). Adding more than four receivers allows to enhance reliability or to gain more accuracy by using iterative or averaging methods [Sch96].

2.2.2 Implementation Issues

In order to implement the mentioned algorithm a crucial issue is to obtain highly accurate timestamps from the receiving stations. This on the one hand involves the wired synchronization between the access points for which the underlying technology is developed in the course of this thesis. On the other hand, also the wireless part has to be able to gather timestamps with high precision. At this point it has to be mentioned that in the best case every nanosecond of uncertainty results in 0.299 m of spacial variability. Since the positioning algorithm is not linear, the resulting error is also dependent on the location of the node with respect to the individual receivers. Therefore, jitter can also have greater influence on the finally available precision. Nevertheless, a goal of 1 m for the overall localisation error can only be achieved if the summed up jitter values from wired and wireless timestamping are safely below 3 ns. This requires both parts to be around 1 ns, which is the aimed goal for the timestamping accuracy.

The main issue with IEEE 802.11 compatible receivers is the fact that the interface between physical and media access layer is not accessible. Figure 2.3 shows the typical structure for a Commercial Off-The-Shelf (COTS) wireless interface of a node. The access to the medium is realised by using two devices, a high-frequency transceiver and a baseband processor. While the first mixes down the high-frequency signal to the baseband, the second is responsible for digitising the baseband signal, frame decoding, protocol

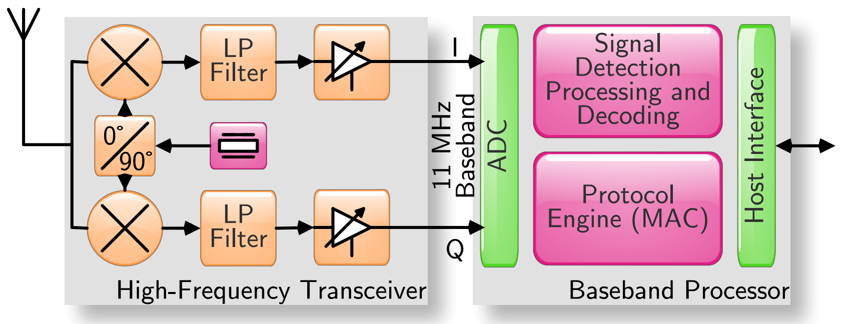


Figure 2.3: Block diagram for a typical IEEE 802.11b compatible receiver

handling, and providing a digital interface to the host system. Modern single-chipset solutions even integrate the functionality of both devices into one component. The digital host interface cannot be used for timestamping since it is not synchronous to the data on the medium. Thus, the least complex solution to realise highly accurate timestamping on the wireless interface is to use a high-frequency transceiver, sample the baseband signal, and run a timing recovery [GLN+09].

The chosen system architecture therefore recovers detailed timing information from the baseband signal, which allows to achieve a resolution much better than the symbol data rate of 11 Mbps. Using oversampling on the baseband signal with 44 MHz and recovering the detailed sub-symbol timing (signal timing and phase recovery) with additional 8 bits allows to achieve a timestamp resolution of 88.778 ps. The method used to recover the phase is similar to the developed concept of chapter 5.4.2 and requires a number of clock cycles to settle. Still, the finally measurable precision of the timestamps drawn in a prototype system shows a standard deviation of only 0.388 ns under laboratory conditions [EGL10]. Measurements with an enhanced version of the high-frequency transceiver even allowed to get below the half of the value. This precision already requires the compensation of the non-linear delay through the high-frequency transceiver, which is dependent on the attenuation and the temperature. While the first distortion can be compensated by an internal filter, the temperature dependence additionally requires a sensor within the transceiver chip in order to allow to cancel out this influence.

The timestamps gathered by at least three different receivers are sufficient to calculate enough range differences for the TDoA method to determine the actual position of the wireless node. In order to test the capabilities of the presented approach, a system [STG+09] out of four receivers was set-up. The timestamps are collected at a central instance, which is responsible for building matching tuples of the input data. This localisation node performs filtering on the timestamps to remove obvious outliers and enhance the localisation determination process later on. Additionally, a Kalman filter [Kal60] for the position is used to enhance the precision. Through appropriate analysis and filtering methods [Sch09] the final positioning accuracy can be enhanced to a standard deviation of 11 mm in the static case.

Although it is not realistic that the same precision can be achieved for moving nodes

– especially, if they change direction, i. e. non-linear trajectories – the measurements show that the combination of wired and wireless synchronization using the presented technologies can provide positional data with an accuracy of less than one meter. High-accuracy synchronization over wired networks, as developed in this thesis, allows to synchronize multiple access points with the required accuracy for even large installations that require roaming of nodes. Except for the degradation of the accuracy due to multiple intermediate switching levels, there is no actual limitation for the area that can be covered by such a positioning system.

2.3 Control System for Particle Accelerators

One of the probably most complex machines ever built by mankind is the new CERN particle accelerator. From a technical point of view the accelerator is a distributed system of sensors and actuators. For the control and monitoring of the beam these sensors and in particular the actuators have to be coordinated precisely in terms of timing. The interconnecting network is referred to as timing network, because of the critical control tasks it fulfils. With the completion of the new LHC, a renovation of this network for the injector complex is needed. Despite several incremental improvements, the current system [LBS+03], is more than 20 years old and has two major deficiencies [LGS+09]:

- *Bi-directionality*: In the current implementation a separate network infrastructure is used to allow communication for diagnostic and control information exchange. This missing upstream has also a disadvantage for the high-accuracy timing requirements. As it is not possible to measure the line delays by using the so-called round-trip technique, the usage of automatic cable delay compensation is not possible. Thus, manual, cost intensive, and unreliable calibration of the cable infrastructure has to be performed. This is currently done with one static and a mobile atomic clock, measuring the arrival of a signal with known generation time. Still this does not allow for continuous compensation as required by e. g., changes due to temperature dilatation.

- *Bandwidth*: Up to now the control system is based on an RS-485 multi-drop system, operating at 500 kbit/s. This circumstance forces the accelerator operators to use a different network for each machine, which causes cabling and software hassles, but still the available bandwidth per accelerator is rather low. To allow for higher event rates, remote firmware updates, and the transfer of diagnostic information from the field devices to the control station, state-of-the-art transmission bandwidth was set as a requirement.

These drawbacks and the interest which results from a number of other application scenarios that cannot be covered by current real-time networks led to the development of a novel sensor and control network. As the requirements in terms of timing performance, spatial distribution, and number of receivers are very high for this type of application, a new fieldbus like general-purpose control and sensor network is needed and proposed.

Scenario	Nodes	Distance	Accuracy	RMS Jitter	Update Rate
long range	1 000	10 km	1 ns	1 ns	1 ms
many receivers	2 000	5 km	100 ps	100 ps	10 µs

Table 2.1: White Rabbit system requirements

Table 2.1 summarises the requirements on the real-time network. The timing for the latter needs to maintain a precision of <100 ps (relative phase jitter) and an accuracy better than 1 ns (absolute offset of any clock to a defined timescale). The network is called *White Rabbit* timing network. Besides the technical requirements another important requirement was the long-term availability of all components, which also imposed the seek for maximum vendor independence through the usage of open source components.

The system consists of a layer 2 protocol and special Ethernet-based hardware components. The overall goal is a new Ethernet-based fieldbus-like communication system that fulfils the performance requirements on layer 2 without imposing restrictions on the upper layer traffic behaviour.

2.3.1 White Rabbit Communication Infrastructure

The proposed system is required to have long term availability (range of several decades), which makes it indispensable to use a widely spread industrial standard communication technology. Therefore, optical Gigabit Ethernet was chosen, which is well standardised, widely supported and nowadays used for many different purposes. This also allows to be cost efficient, since the necessary components are produced in large numbers. Additionally, it provides assured availability of the technology together with state-of-the-art throughput.

For the communication protocol, several existing real-time enabling Ethernet technologies have been evaluated whether they can or in the near future could fulfil the performance requirements. Additional boundary conditions were that the system should be an open standard (to ensure long-term availability) and independent real-time behaviour of the actual overall traffic volume. The most promising candidates were Ethernet Powerlink, SErial Realtime COmmunication System (SERCOS)-III, and EtherCAT. All have their particular advantages given in section 3.3, but to the final end, only Powerlink could fulfil the requirements. Nevertheless, there was a lack of Gigabit Ethernet and sufficient asynchronous communication support, which is limited to a single slot. The latter hinders efficient transport of high amounts of asynchronous data. [CSVV09] The goal is therefore to establish a network architecture that is capable of transporting real-time information with guaranteed delivery times, as well as allowing the remaining bandwidth to be efficiently used by non-prior traffic in the background.

The high requirements concerning the synchronization accuracy led to a system, where the whole infrastructure is based on a synchronous approach. This can be done by utilising Synchronous Ethernet (SyncE) [ITU08b] as discussed in chapter 3.1.1. The technology has its origins in the telecom technology. For this application, frequencies are adjusted (syntonized) via hierarchical communication clock recovery. However, for adjustment of the absolute time a second protocol is required. In case of White Rabbit this is done via the clock synchronization protocol IEEE 1588, which cannot provide sufficient precision for jitter adjustments in the picosecond range if used with common oscillators as the reference clock for slave nodes. Since these oscillators already induce significant noise even for short term intervals only two options, namely equipping all nodes with expensive stable oscillators or a synchronous approach are applicable. To limit the size and effort for a high number of slave nodes as well as to keep costs within a reasonable range, the latter option was chosen for the design of the real-time network system.

Since earlier design decisions already limit down the pool of transmission standards to the ones that conform to the Ethernet standard [IEE08a], only a small number of options are available for implementation. Out of these, the main issue is to select from either copper-based or fibre-based methods. For the reasons mentioned in the following, the latter were chosen, because they allow for some essential features that cannot be easily realised otherwise – or eventually with degraded performance.

- Fibre-based Ethernet transmission methods use high symbol rates on the medium, which not only allow for high data throughput, but also for more accurate synchronization due to an increased number of sampling points. Besides that, the chances that future improvements to Ethernet can also be realised using copper links continuously decrease, which can be seen from the fact that the delay between publishing the standard for copper and the one for fibre significantly increased from the 100 Mbit/s over the 1 Gbit/s to the 10 Gbit/s standardisation effort.

- The optical transmission also provides significant benefits in order to avoid disturbances by electromagnetic interference (e. g., caused by high energetic fields produced by particle accelerators) over the targeted large distances. Furthermore, the galvanic separation of the elements of the network allows the usage also in electrically critical areas as well as to avoid influences on other installations.

- Another great advantage of fibre-based transmission links is the fact that the link delay can be accurately calculated. In case mono-mode fibres are used, the delay can only change due to changes of the length caused by varying physical properties, e. g. temperature. The advantage of single-fibre transmission standards, e. g. 1000 Base-BX, is that the changes effect both directions in the same way. Since they use different wavelengths for the individual directions, the refractive index of the fibre causes diverse propagation speeds and therefore inequality in the link delay. The wavelengths are known, therefore the asymmetry factor can be calculated and thus a conventional round-trip delay measurement be corrected for an accurate one-way delay.

Concerning timing, the network is hierarchically organised and propagates the time and frequency information from a timing master down to a large number of nodes. An overview of the whole system structure is shown in figure 2.4. The master is synchronized to either GPS or an atomic clock. For the high-accuracy nodes, there are Gigabit Ethernet

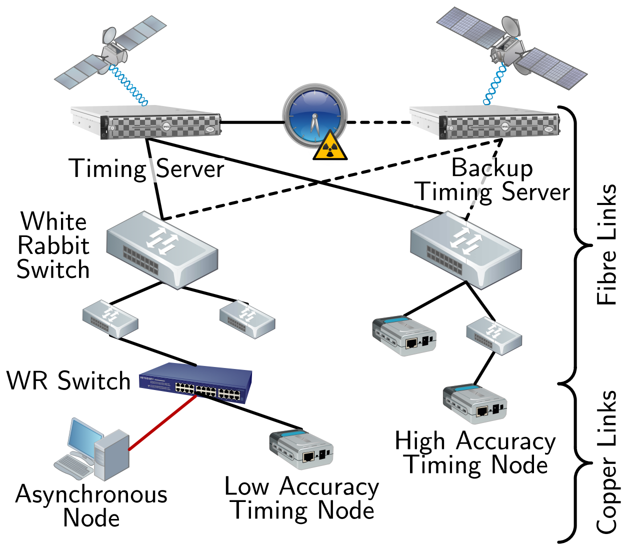


Figure 2.4: Overview of the White Rabbit network structure

links over the already mentioned mono-mode fibres, which allow for highly accurate delay compensation, since the asymmetry can be calculated out. First implementation results already show an accuracy safely below the nanosecond (±350 ps, $\sigma = 80$ ps), even under changing thermal conditions [MSW+09].

Switch

As for all up-to-date Ethernet standards, the White Rabbit network architecture uses a packet-switched approach based on star topology. The central element of the network, hosting the extended functionalities, is the White Rabbit switch. This device in its basic functionality can be used as a switch for standard Ethernet traffic. However, it also incorporates important features to allow high-accuracy clock synchronization and the usage in critical low latency applications.

The White Rabbit switch implements SyncE enabling downlink synchronization of the slave timing nodes. In order to propagate a frequency stable clock through the network, the switch behaves as a timing slave on both uplink ports, while serving as a master on all downlink ports. In addition, the switch uses IEEE 1588 messages to calculate clock offsets, measure link delays, and transmit phase measurements. The clocking structure of the switch embedded in the structure of the network is shown in figure 2.5.

To achieve sub-nanosecond synchronization accuracy between the White Rabbit timing master and its slaves nodes, the network switch uses a two-step link delay estimation [SSC+09]. First, the peer delay mechanism of IEEE 1588 version 2 measures the port-to-port propagation time on the basis of the round-trip delay estimation method. Since the mechanism depends on the available timestamp resolution, the estimation is limited to the 8 ns granularity of the Gigabit Media Independent Interface (GMII) interface. At maximum the 1 ns resolution of the symbol bit rate on the medium can be used but this

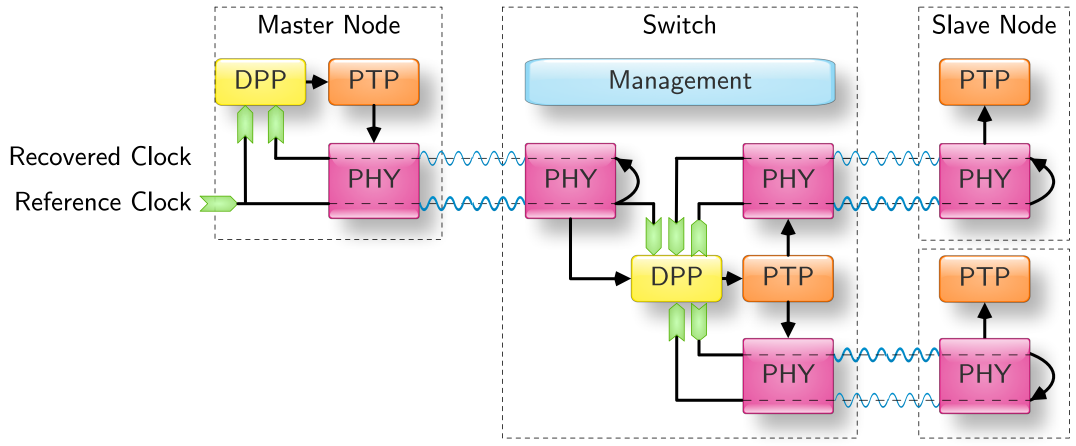


Figure 2.5: White Rabbit clocking architecture

would require non-standard modifications to the physical layer. Thus, in a second step, a digital phase compensation mechanism is used for fine grained adjustments.

As illustrated in figure 2.5 all slave ports of the system loop back the incoming receive clock to the master node. The switch does this for its uplink port, while all timing slaves have to do it for their ingress port. The phase relation of the signal returned from the loopback port is then measured in the master against the reference clock of the switch or master node within the Digital Phase Processing (DPP) block. The phase measurement and timestamping again can use the developed high precision methods for detecting events in the CSC. The value for the phase difference of the recovered clock is then transferred via an attached Type, Length, Value (TLV) element to the slave. The latter can then use the correction value to achieve a correct timestamping value below the granularity of the clock itself. Within the switch, the DPP block uses the received correction data to shift the phase in order to produce an internal synchronous clock, which is then further propagated to subsequent elements of the network.

The slaves can also implement a phase alignment block that readjusts the phase to the reference master clock in dependency of the measured link delay. This is done in addition to having their frequency locked the recovered data clock. Optionally, the phase alignment operation can also be omitted, to reduce the necessary hardware effort, and the slave can just perform a digital correction of the timestamps using the phase correction value. Thus, all slave nodes can execute control commands with respect to an absolute timing, given by the timing master, independent of the propagation delays within the network.

Protocol

For the sake of long-term availability White Rabbit uses Ethernet as its physical layer to allow the use of COTS devices. However, to enable its usage in critical time constraint applications it is necessary to add a set of rules to the data link layer. Additionally, the

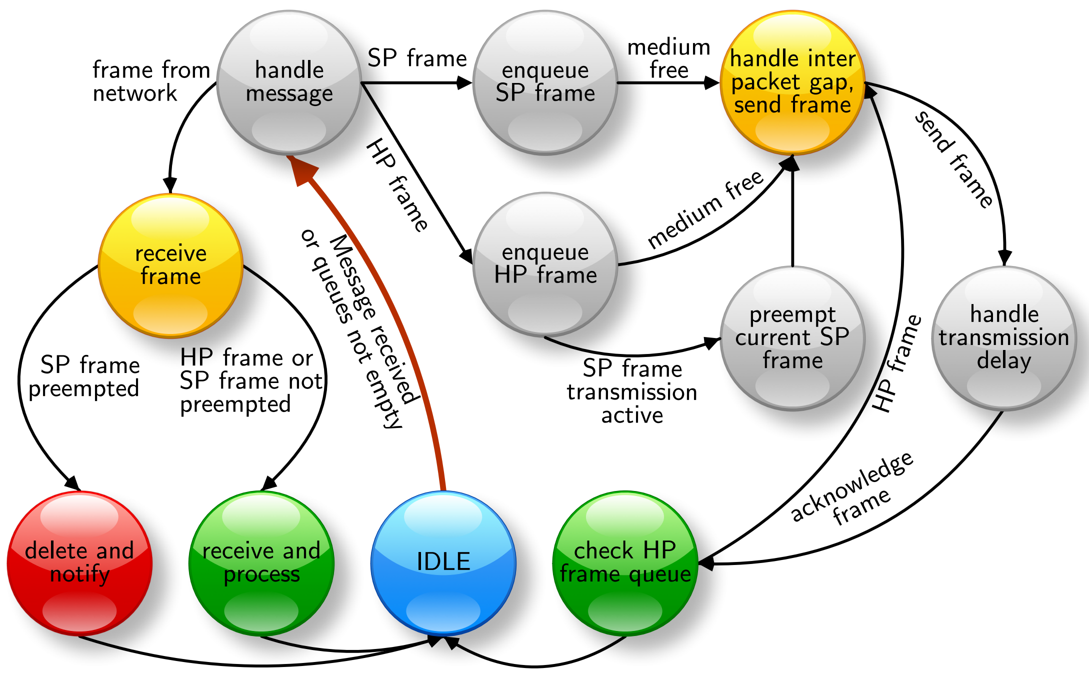


Figure 2.6: Simulation state diagram for the MAC operation

protocol aims at being fully compatible with standard IEEE 802.3 traffic, but enhances the MAC protocol with some new definitions that allow its usage in very low latency and critical applications.

In order to guarantee low latency delivery of messages, two types of packets have been defined. These are so-called High-Priority (HP) and Standard-Priority (SP) network frames. The first are specified to be transmitted with minimal transmission delay in the network and are used for very critical control messaging, supporting an upper bound of the transmission delay for a given topology. It is specified by the protocol that in case there is a request to send an HP message and the channel is not free, an ongoing SP transmission has to be preempted by the HP frame to allow its immediate delivery. The frame that was preempted will be retransmitted after the HP message has completed transmission. The technique is based on the fact that preemption in Ethernet has been proven to be very effective in reducing the transmission time of low latency frames [YOU06]. Figure 2.6 shows the state diagram for the modified MAC scheme, which differentiates between the two traffic classes and supports preemption.

Furthermore, the (potentially) critical nature of the HP frames requires that transmitted frames are guaranteed to be received by the destination nodes. In White Rabbit this is achieved by coding the transmitted HP packets with a digital fountain code (also known as rateless erasure code). To produce the digital fountain code, White Rabbit uses the Luby Transform (LT) code [Lub02]. The basic principle is that a single message with N symbols is split into multiple parts containing symbols, which code redundant information. On the receiver side only any of $\Theta(\sqrt{N} \cdot \ln^2 N)$ symbols have to arrive in order to reconstruct the original message. Previous work showed that this method is very efficient in providing a way to transfer critical information over erasure or Additive White Gaussian Noise (AWGN) channels [TNYH06] like the present structure using Ethernet.

Contrary to the above mentioned high-priority messages, standard-priority frames are used for non-critical message transmission, where the latency requirements are more relaxed or even not relevant at all. The type of message is intended to transmit non-critical control messages, additionally all other Ethernet frames are enqueued using the same priority. This way, it is possible to efficiently use all remaining bandwidth that is left over by the high-priority frames. The preemptive mechanism allows to dynamically allocate transmission capacity to important messages. Standard traffic can then *fill up* the remaining bandwidth to efficiently use the link capacities.

In order to prevent congestion in the White Rabbit network a higher layer protocol (e. g., rate constraint real-time protocol) has to manage the prioritised type of frames. Especially, the extensive use of HP messages transmitted from many nodes to the same destination can cause congestion on common links. This can cause violations of the upper delivery delay bound and therefore disturb time critical network operations. Since the standardisation of the White Rabbit architecture is limited to the lower two layers of the network stack, the congestion issue is delegated to existing solutions.

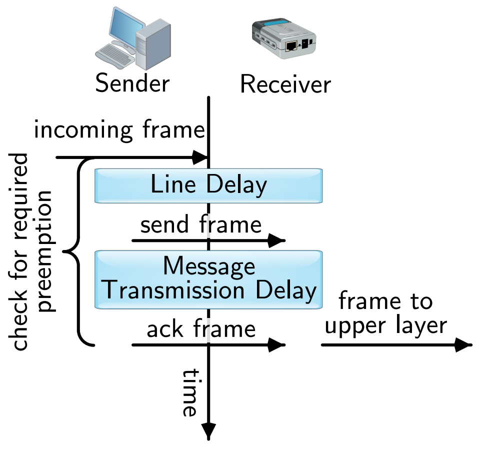
Simulation

In order to analyse the packet delivery timing of the described architecture, a simulation environment using the OMNeT++ Discrete Event Simulation (DES) system was set-up. The essential element for an accurate simulation is the integration of appropriate simulation models. The most important one for the case of the White Rabbit protocol is the media access scheme, which uses preemption of standard Ethernet frames. It is demanded to deliver higher performance in terms of real-time behaviour by the possibility to deliver a high-priority frame sent out by the master to any node in the network within at most 1 ms for the scenarios of table 2.1. Thus, the existing MAC model of OMNeT++ was modified to support preemption of standard-priority messages by high-priority frames.

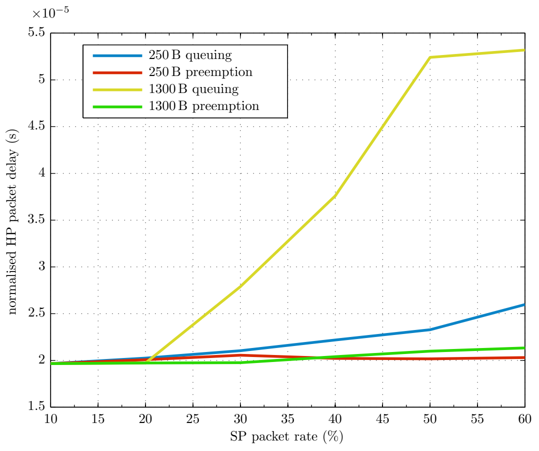
The new model of the MAC features different behaviour over the standard IEEE 802.3 compatible controllers. There is also a major difference to priority queuing. The proposed solution is able to preempt a low-priority frame while it is sent. Therefore, the solution is almost independent of the characteristics of the underlying traffic, e. g. frame length. Due to the discrete event behaviour of the simulator every delay has to be modelled by scheduling a self-message of the node to later proceed with processing a frame.

In order to handle possible preemption of SP frames in the DES correctly, several additional (non time consuming) messages are exchanged. Figure 2.7(a) shows the actual behaviour. After receiving a frame from the upper layer protocols, the modified MAC delays the frame by the line delay to mimic the physical behaviour. Next, the transmission delay (message length multiplied by the bit time) is awaited. If during the two delays no HP message arrives, the frame is acknowledged, so that the receiver can further process it. In case of a required preemption, the frame is immediately dropped.

As illustrated in figure 2.7(b) the system can guarantee an upper bound for high-priority



(a) Discrete-event-based MAC scheme



(b) Delay behaviour for high-priority messages

Figure 2.7: DES evaluation of the White Rabbit MAC protocol

messages independent of load and message size as long as there is no congestion among the class itself. In order to demonstrate the influence of the standard-priority messages on the HP delivery delay for different prioritisation mechanisms, simulation is run with two different SP payload lengths, one of 250 byte and the other 1300 byte. For both run cases, standard-priority traffic load has been varied from 10 % to 60 % of the total link capacity, while keeping the HP traffic rate constantly at 50 %. The HP delay is normalised to the full frame delivery time $(\text{msgLength} * \text{bitTime} + \text{lineDelay})$ on the link in order to better visualise the influence with different message lengths.

In a direct comparison, it is shown that preemption, unlike the simple priority queuing, is keeping the delay of the high-priority frames almost completely independent of the length or the traffic rate of the SP frames. It could be even intuitively concluded that the effectiveness is, of course, varying, as benefits of preemption are more obvious with the higher SP payload sizes, typical for block transfers. Together with the highly accurate clock synchronisation, based on the synchronous approach, the architecture is able to perform measurement and control tasks over large distances with so far unachieved accuracy.

3 State of the Art and Related Work

In order to classify the current state of the art in digital communication networks, they have to be distinguished by the main criteria mentioned in table 3.1. This thesis mainly covers mechanisms for Ethernet [IEE08a] and therefore for an asynchronous, packet-oriented, and desirably transmission delay compensated network. Chapter 8 will additionally present synchronous approaches based on Ethernet, which allow the distribution of clock signals but also require special measures on the physical layer. Due to the latter, flexibility and interoperability is reduced.

The methods presented in the following do not assume a special relation between the synchronizing nodes of a network. That means, the difference between master/slave and mutually synchronized networks is mainly reflected in different clock synchronization algorithms on the higher layers of the network. Therefore, the presented hardware elements supporting clock synchronization can be used for both methods, which makes it in the scope of this thesis unnecessary to differentiate on the selected control algorithm that steers the clocks.

Besides the classification of the used network, also the methods to syntonize network nodes can be divided into the following methods according to [LGHD85]. The additionally required determination of the transmission delay between the nodes is tackled in section 3.1.2 for synchronous approaches and in chapter 3.2 in the description of the respective protocols for packet-oriented networks.

- Using the *burst position measurement* method each node sends out regular data bursts or a pulse at predetermined instances in time. The receiving node then compares the time the burst is received with the planned schedule. The difference between the planned and the actual receiving time is proportional to the time difference between the sending and receiving node plus the transmission delay. Burst allow the usage of a weighted average of the errors, while pulses have the disadvantage of high bandwidth requirements and probably cannot use the same medium due to physical layer restrictions.

Criteria	Available, Warring Options	
communication	asynchronous	synchronous
connection	packet-oriented	circuit switched
control	master/slave synchronized	mutually synchronized
compensation	transmission delay compensated	uncompensated

Table 3.1: Criteria for classification of time and frequency distribution networks

- *Clock time scale sampling* refers to the technique of sampling a reference clock at regular intervals and then transmitting the value over a network to client nodes. The latter compare the received sample with the current local clock status. The difference between the local and the remote sample is composed of the timely error between the two nodes, the transmission delay, and the time required for sampling.
- Pseudo random noise signals can be utilised to run a *continuous correlation of timing signals*. By tracking the phase offset between the received sequence and a local generated one, again the deviation from the master node can be determined. This method is used e. g. by the GPS [Get93] and works similar to the burst position measurement.
- When the *buffer fill level measurement* technique is used, the buffer has to be filled using a clock synchronous to the transmitter and read out with the local clock rate. The fill status depends on the frequency difference of the two clocks and can be used as a measure to correct the receiving clock rate.
- Obviously, *frequency difference measurement* can also be used to keep a network synchronized by adjusting all clock generators to a given frequency. Then it is possible to benefit from synchronous transmission operation as it is explained in chapter 3.1.

The state of the art mentioned in the upcoming sections first gives an overview on synchronous network approaches. These types of networks integrate measures to transfer the source clock frequency to the destination trying to keep a constant data rate end-to-end. In contrast, asynchronous networks mainly care about data transmission and therefore do not require special measures to synchronize nodes for the sake of transmission. Nevertheless, consistent timescales and a global view of the current time is desirable for many applications as mentioned in the introduction of this thesis. Therefore, special protocols given in the second section of this chapter were designed to fulfil the task of synchronizing network nodes. The chapter is completed by a list of available real-time protocols and available COTS equipment.

3.1 Synchronous Network Approaches

Synchronization of digital communication networks was greatly pushed by the evolvement of Pulse Code Modulation (PCM) in telecommunication networks. Before the introduction of wide area computer networks (or even the Internet), these networks represented the largest (even world-wide) distributed communication system.

Whereas early telephone networks were purely analogue, the need for more efficient usage of the media first led to Frequency-Division Multiplexing (FDM) techniques and with the possibility of digital transmission to Time-Division Multiplexing (TDM) [Bre98]. FDM was used for analogue signals and shifts the individual communication channels to different carrier frequencies on the medium. In contrast, the introduction of digital transmission made it possible to use time-slots and therefore handle all transmissions in a similar way. Nevertheless, the digital communication introduced the problem of synchronizing the network in order to keep data loss due to buffer overruns caused by different clock rates of the involved nodes as low as possible.

In the early days, digital communication was used as lossless links between analogue networks. There was no need for synchronization at all since the link was point-to-point and only two nodes were involved. The next step was to use all digital networks, but in a somehow asynchronous way. Every source node could use an individual clock and the delivered data was rate and frame synchronized in the aggregating multiplexer to use TDM. This technique soon showed two major problems. First, digital switching machines are rather complicated due to multiple levels of unsynchronized signals and the inability to extract a single lower rate signal without having to demultiplex the whole aggregated data link. Second, a chain of multiplexers and demultiplexers for switching and aggregating individual sources, introduces jitter and delay into the original source signals and therefore might hinder correct recovery after some levels [Bel95].

In order to overcome the above mentioned problems, telecom industry introduced fully synchronous networks based on technologies like the American National Standards Institute (ANSI) standard Synchronous Optical NETwork (SONET) [ANS95, Boe90] or the International Telecommunication Union (ITU) representation Synchronous Digital Hierarchy (SDH). Both standards replaced the first generations of fibre optic systems in public telecom networks, which were based on proprietary architectures, multiplexing formats, and maintenance procedures. The need for interoperability led to the definition of the network architecture, frame format, bit rates, and way of multiplexing various streams into a synchronous high-speed data flow. The synchronicity of all network equipment and the special container format significantly ease the aggregation and demultiplexing for individual low data rate streams, which are common to telecom applications [BC89].

3.1.1 Synchronous Ethernet

The idea of SyncE is to transfer frequency information over the Ethernet physical layer. The ITU published several recommendations within the Telecommunication Standardization Sector (ITU-T). The functions to synchronize two nodes over a network are bundled in the *synchronization layer* [ITU99]. The required architecture of general transport networks and the necessary layers and services are defined in [ITU00c, ITU00b]. The documents recommend a common approach, independent of the actual technology used. One widely used implementation are SDH communication networks [ITU07a, ITU00a, ITU04b, ITU03]. Based on these, adaptations were made to generalise the rules and apply them to Ethernet. Therefore, the method can also be utilised for other media with constant bit rate operation in principle.

In the case of Ethernet (described by two layers in [ITU04a]) the recommendation G.8261 (Timing and synchronization aspects in packet networks) [ITU08b] is considered to be the main reference, which is extended by two others [ITU07b, ITU08a]. All these documents refer to Ethernet as defined in [IEE08a] and apply methods for using Ethernet as a synchronous network to all standardised physical media. An overview of the involvement of the different standards can be found in [FGJ$^+$08].

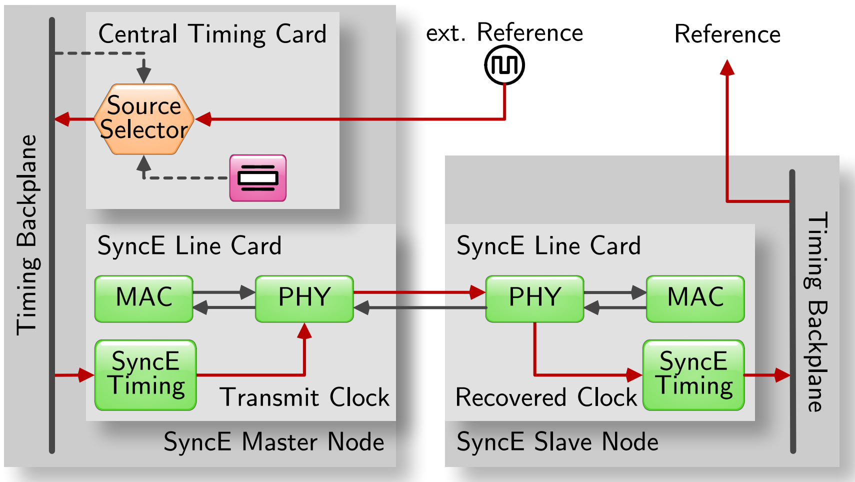


Figure 3.1: Synchronous Ethernet

Basically, the operation principle of Synchronous Ethernet is to reuse the clock recovered from the remote transmitter on the receiving node. In contrast to standard Ethernet the recovered frequency is not only utilised to synchronously sample the incoming data stream, but also to transmit further data. The mentioned ITU recommendations form a set of rules about the required minimum functionality, appropriate quality, the network management, and structural issues in order to provide interoperability between different implementations of the equipment. The SyncE approach itself can be seen as a special case of the Ethernet standard regarding clock distribution within the network elements (especially hubs and switches).

The principle is illustrated in figure 3.1, showing a master node with a central timing card and a slave node communicating over a single Ethernet based link. Based on a tree hierarchy, the source selector in the master node selects either an external reference, an internal oscillator or a dedicated uplink line card to be the clock source for the whole node. This clock is then propagated to the timing backplane. The Synchronous Ethernet module in every line card then uses this selected clock to drive the transmit clock of the physical layer device. As common for standard Ethernet, the data stream is sent synchronous with the transmit clock.

On the receiving side a Phase-Locked Loop (PLL) recovers the clock from the data stream in order to sample incoming data. In case of SyncE, this clock (receive clock on the MII) is used by the timing module and passed on to the timing backplane of the slave node. Since an upper layer protocol decided about the hierarchy, the slave node then uses this upstream clock on all line cards as a transmit clock. Further, it can provide the recovered clock to external devices as a reference. If the network has multiple stages, the slave node is configured like the master node, but the central timing card selects one line card to provide the clock to the backplane.

Besides the introduction of a protocol organising the hierarchy, only a few elements allowing the reuse of the receiving clock and the clock source selector are required over a conventional Ethernet implementation. Consequently, the additional effort for providing SyncE equipment is rather low.

3.1.2 Delay Compensation Techniques

One of the key problems still remaining when using synchronous network approaches are the effects caused by transmission path delays. These delays hinder a global synchronization and therefore high performance of digital communication networks since the phase relation between any two nodes is unknown. Also, a global notion of time in order to perform coordinated actions among all nodes of a network cannot be established. Since the effects caused by the transmission path delays on the system performance are undesirable, techniques for compensation of the delay are of great interest.

Whereas systems comparing only the arrival time of the received signal to the local clock (single ended systems) cannot fulfil the requirements, there are two different ways of accomplishing these compensations. [LGHD85] Both methods, listed in the following, eliminate the effect that in single ended mutually synchronized networks the steady-state frequency depends on the transmission path delays between the nodes [BNK89]. Further, also changes in the delays have backlash on the system frequency. Double ended systems exchange the locally measured timing error information and can therefore compensate for (changing) transmission delays.

The following two subsections will explain the principle operation of advanced systems based on an exemplary two node mutually synchronized system.

Equational Timing System

The Equational Timing System (ETS) was designed for the purpose of synchronizing telephone network switching points in PCM integrated telephone networks. These exchange points have to be synchronized in order to keep loss due to buffer overflows or bit slips as low as possible. The principle of the method is shown in figure 3.2 and can be expanded to a multi-nodal system with weighted inputs.

A phase detector determines the offset between the local oscillator and the remote clock source. Further, the remote offset is decoded from the received signal. Consequently, the offset measured at the local node, as well as the information about the deviation at the remote node can be used (via appropriate filtering) to adjust the Variable Frequency Oscillator (VFO). The detailed analysis in the original proposal presented in [YOU68] shows that the steady-state system frequency is independent of the number of nodes and the intermediate delays as long as there are sufficient interconnections.

Returnable Timing System

In contrast to the previous method, the Returnable Timing System (RTS) measures all required phase differences, which contribute to the control signal to the local oscillator, at that same node. It therefore belongs to the group of single ended synchronization systems, but was designed to deliver similar performance as double ended approaches.

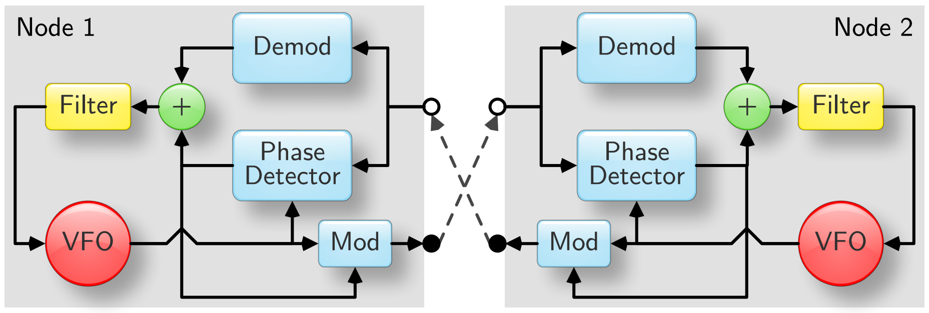


Figure 3.2: Basic interconnection of an Equational Timing System (ETS)

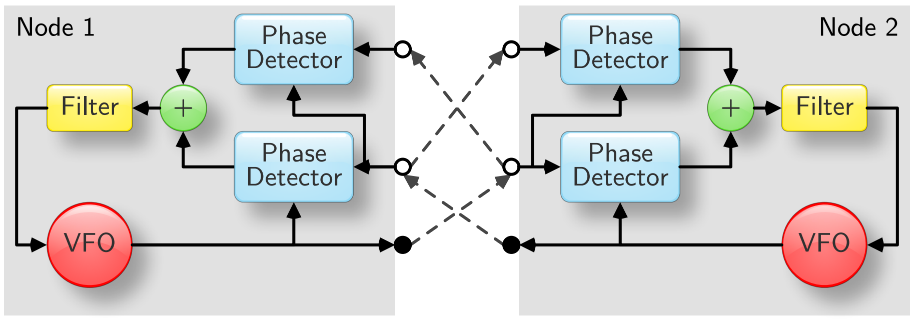


Figure 3.3: Basic interconnection of a Returnable Timing System (RTS)

This is accomplished by mirroring back the original timing information and measuring both, the received signal and the mirrored signal at the local node. The basic operation is shown in figure 3.3 and was first proposed in [LKD78] which proves that the system has similar properties as double ended systems but reduced complexity.

3.2 Protocols

In principle clock synchronization for packet-oriented networks requires determination of the offset between the reference and the clock to be adjusted. To get additional absolute synchronization, it is further necessary to determine the time (particular reference value) for comparison. This results in measuring of the read-out time, or in the case of communication links, the transmission delay. Thus, all protocols, which seek for synchronization of clocks in computer networks have to fulfil the two above mentioned tasks.

In the special case of packet-oriented networks the end-to-end transmission delay can vary for each packet. This can result for example from different routing paths, packet processing time in switching elements, and changes in the frequency of the clock used for the transmission line(s).

3.2.1 Network Time Protocol (NTP)

The roots of the Network Time Protocol can be traced back to the early 1980s when the Internet was still a collection of individual institutional networks accessible to the Defense Advanced Research Projects Agency (DARPA) Internet Project. The first documented use of synchronization technology for computer networks was in the Internet Engineering Note IEN-173 [Mil81b]. The note describes mechanism to synchronize a logical clock in a set of hosts to a single physical clock (a master node with a reference clock).

Historic Development

The first public specification of the according protocol, which was used in Distributed Computer Network (DCNET), a local network connected to the early Internet, became available through RFC-778 [Mil81a]. It already specified datagram layout and timestamp format based on the Internet Control Message Protocol (ICMP). All operations were performed with millisecond precision relative to midnight UTC.

These early methods led to the definition of version 0 of NTP [Mil85] in 1985. Beside the specification of the data exchange, the document also contains the definition of offset and delay calculation still used today. At that time accuracies of some tens of millisecond and around 100 ms could be achieved in LANs and WANs respectively.

Three years later version 1 defined in RFC-1059 [Mil88] added documentation about clock filtering, selection, and disciplining algorithms. Several enhancements like the NTP Control Message Protocol and a cryptographic authentication scheme for timing information were introduced with version 2 specified in RFC-1119 [Mil89]. The document first included a formal model and a description of the state machine to formalise the protocol operations using pseudo-code.

In 1989 a competing standard for synchronization in networks called Digital Time Synchronization Service (DTSS) came up. The main advantage were the correctness principles in the design process and the usage of an agreement algorithm invented by Keith Marzullo [Mar84, MO83]. "One problem with DTSS, as viewed by the NTP community, was a possibly serious loss of accuracy, since the DTSS design did not discipline the clock frequency." [David L. Mills] After integrating the algorithm with some adoptions to solve the jitter problems on typical Internet paths, the discussion ended with the specification of NTP version 3 in 1992 [Mil92a].

The new specification added a formal error analysis and reliable measures for selecting the best upstream time server. Further, the possibility of adding reference clock drivers was introduced, which allows for external time references (e. g., atomic clocks, GPS, PPS discipline, DCF 77). Version 3 of NTP specified in RFC-1305 is currently still valid, although there are continuous efforts for improvements, which will result in a significant revision of the NTP standard called version 4 [Mil06b]. Until now, this current development version was not formalised in a Request for Comments (RFC), nevertheless, the current software revision of the NTP daemon already supports the enhanced features of the new version.

Due to the complexity of NTP, the Simple Network Time Protocol (SNTP) [Mil92b] was introduced to avoid unnecessary large overhead in applications, where the full features of NTP are not required. The original and the reworked version 4 [Mil06c] spare out the complex clock selection and disciplining algorithms. Although it is still compatible with the full protocol version, it intends to let hosts run in a simple, stateless Remote Procedure Call (RPC) mode similar to the UDP/TIME protocol. Further, it allows for simple and dedicated time servers (e. g., GPS reference servers).

Although NTP is known as a wide-area, fault-tolerant, but not very precise protocol, there are attempts to enhance the standard implementation (the NTP daemon) to deliver nanosecond accuracy. The daemon is capable of including high precision, external reference clock but still, the timebase for the software is the timer interrupt of the hardware platform. Since the latter is in general not specially designed to support highly accurate timekeeping (high resolution, stable oscillator), special adaptation to the daemon and the operating system kernel have to be made in order to achieve nano second accuracy [MK00]. Current implementations are able to deliver a residual Root Mean Square (RMS) error in the order of 50 ns. Since the protocol (till version 3) has an inherent timestamp resolution limit of 232 ns and due to the 32 bit seconds field overflow in the year 2036, version 4 allows an alternative timestamp format of 128 bit giving 64 bit fractional seconds, which is equivalent to 0.05 as.

Protocol Principles

NTP is fully based on the connectionless service of UDP/IP. This means, the protocol in general does not maintain any state information about connections between the nodes. Each request to a server stays for itself and does not require previous actions to be processed. Consistently following this idea, there is no support for automatic server discovery.[1]

From a hierarchical point of view, the service can operate in two different modes. On the one hand there is the symmetric mode, in which hosts are equal and maintain some synchronization information about their peers. On the other hand, there is the asymmetric mode allowing the server to supply a large number of clients with timing information. This mode does not require any other state information than the one contained in the request and works like the well-known technique from process intercommunication, the RPC.

In order to estimate the accuracy of the upstream server, NTP uses so-called stratum levels. The hierarchy starts at stratum 1, which represents a primary server synchronized by outside means (direct connection to a stratum level 0[2] clock). With each NTP synchronization connection the level is increased by one (see figure 3.4). Common

[1]In NTP version 4, the *manycast* mode was introduced, allowing an NTP server to broadcast timing information to clients in the local network. Only if clients *see* such servers, they can automatically configure themselves to a server. In any other case, manual or Domain Name Service (DNS) based configuration is required. [Mil06a]

[2]The protocol itself uses stratum 0 for unspecified or unavailable hosts. Consequently, for comparison purposes this value is considered greater than any other stratum value. Nevertheless, reference clocks connected to primary servers are also called stratum 0 clocks, since they use direct connections instead of NTP packet exchange.

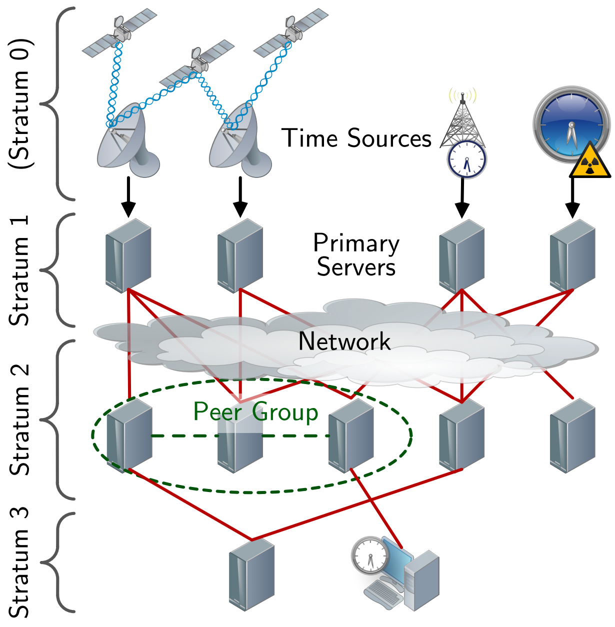


Figure 3.4: Synchronization hierarchy

installations run with the final client at maximum at stratum 4. Nevertheless, the protocol supports an 8 bit value, which is limited by the reference NTP daemon implementation to level 16, then meaning an undisciplined clock.

The basic operation of the protocol is based on the evaluation of four timestamps t_{i-3} (originate), t_{i-2} (receive), t_{i-1} (transmit), and t_i (local time) by the client. The sequence is shown in figure 3.5. The client sends out an NTP message with a transmit timestamp according to its local clock value. Next, the remote server copies the client transmit timestamp to the originate field, determines the state of the system clock to add the receive timestamp, and responds with a message further containing the transmit timestamp. After receiving the response, the client can calculate the roundtrip delay r_i and the clock offset o_i to

$$r_i = (t_i - t_{i-3}) - (t_{i-1} - t_{i-2}), \tag{3.1}$$

$$o_i = \frac{(t_{i-2} - t_{i-3}) + (t_{i-1} - t_i)}{2}. \tag{3.2}$$

The above mentioned mechanism can be used to synchronize in symmetric (server to server) or unsymmetric (client to server) mode, although there are small deviations in the way the timestamps are used and copied. For details see the appropriate NTP standard [Mil88, Mil89, Mil92a, Mil06b]. This rather simple concept together with the standardised packet format allows for a wide area of application fields without facing the problem of the necessity for different software implementations.

The described polling process to retrieve remote time information is only a part of the overall NTP node structure shown in figure 3.6. It is run for every configured upstream

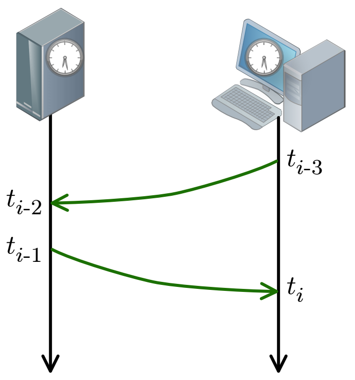


Figure 3.5: Timestamp points for offset and delay calculation

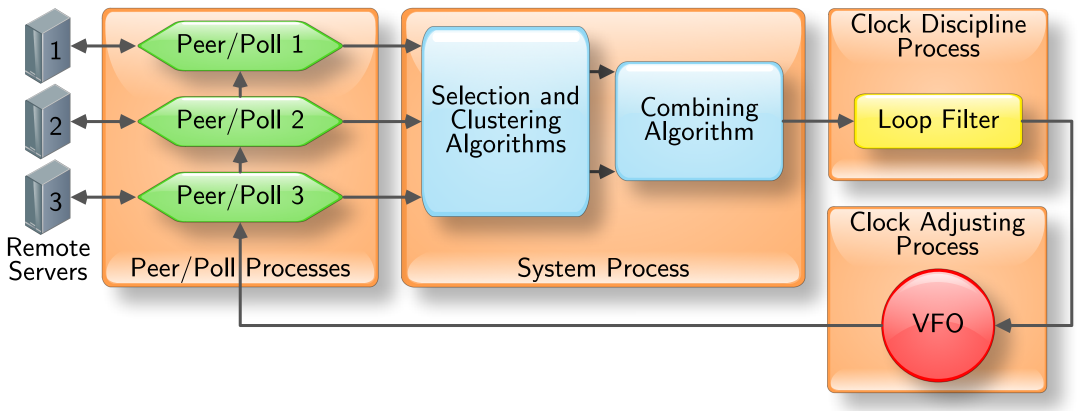


Figure 3.6: NTP implementation model according to [Mil06b]

(lower stratum) server or peer (same stratum) and makes the gathered data available for the clock *selection and clustering algorithms*. Together with the *combining algorithm* the latter selects the most accurate and reliable time source to synchronize the local system clock. Through the feedback from the clock adjusting process, the node is also able to adapt the individual server polling intervals.

Then the result is a reference time, which is fed into the control *loop filter* in order to adjust the time and the frequency of the local system clock. In contrast to other synchronization protocols, NTP (since version 2) also specifies the control loop and the adjusting algorithms. Details about the sophisticated and well studied approaches to filter out network jitter, faulty nodes, and other systematic errors can be found in the current specification [Mil06b].

3.2.2 IEEE 1588

The IEEE 1588 standard, also known as PTP, is a precision clock synchronization protocol for networked measurement and control systems, telecom applications, and automation. It is intended to synchronize independently running clocks on different nodes of a distributed system. The clear goal of the specification is to reach a high degree of accuracy and precision. Further, it defines measures for self-configuring (administration free) system topologies and allows operation, which is independent of the underlying network technology used. The protocol is based on the master/slave principle (with the exception of the initiation of link delay measurements) and is able to automatically establish the necessary relationships through the Best Master Clock (BMC) algorithm. This process ensures that within each PTP subnet[3] there is only one unique master clock. Within a complete network the ultimate timebase for a PTP subdomain[4] is called grandmaster clock. This grandmaster can, but not necessarily has to, be synchronized to a highly accurate clock source by means other than PTP.

The standard was originally published in 2002 [IEE02] and adopted in 2004 by the International Electrotechnical Commission (IEC) in [IEC04]. After extensive use in academy and industry new application fields and the lack of transparent clocks led to a revision of the standard, which was then published by the IEEE in 2008 as version 2 [IEE08b]. Transparent clocks, e. g. PTP aware Ethernet switches, in general forward all non-PTP messages, but are able to perform residence time corrections on PTP messages. Therefore, these elements do not influence the synchronization between slaves, also called *ordinary* clocks, and the elected master. An overview of the PTP network structure can be found in figure 3.7.

The maximum achievable accuracy significantly depends on the precision of the transmit and receive timestamp. Although it is possible to implement such a timestamping as software algorithm (e. g., interrupt routines) this unpredictable latency introduces an additional uncertainty. As a consequence, the achievable synchronization accuracy would

[3]A subnet is a part of a network, which is separate by a device not passing on PTP non-management messages. For the case of Ethernet this is equal to a broadcast domain.

[4]A subdomain is the logical group of clocks establishing synchronization relationships to each other, but can run independently of other subdomains.

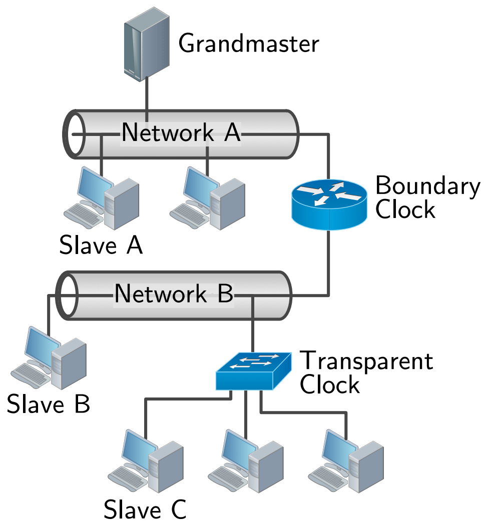


Figure 3.7: Structure of a PTP network with ordinary clocks, a boundary, and a transparent clock

be quite limited and only acceptable for low-end applications. Therefore, the node architecture should use a device, the MII scanner, which allows to take timestamps immediately before the packet enters the physical layer. This ensures that the often variable delay introduced by the MAC (e. g. using Carrier Sense Multiple Access (CSMA)) is not relevant for the time sample.

One issue still remaining in the current version of IEEE 1588 is the lack of redundancy. Although, the BMC algorithm makes sure that there is always one synchronization master available, the election process takes time in the order of multiples of the synchronization interval. Thus, during that time, the slaves run freely and unsynchronized. To avoid this undesired behaviour, the introduction of redundant masters building a group was proposed in [Gad08]. The group maintains a fault tolerant average time of all participating nodes and communicates via a so-called *speaker* node to the slaves. This is fully transparent for the slave clocks since the group appears like a single but permanently available master clock. In case one of the master node fails, the accuracy of the group might be degraded but synchronization is seamlessly available.

Network Elements

A minimal PTP network consists of two *ordinary* clocks, which are to be synchronized. The BMC algorithm selects a master clock according to predefined accuracy levels and as a last selection criteria to the Universally Unique IDentifier (UUID). After its completion, a unique master exists within a network. As shown in figure 3.7 a PTP network can consist of multiple physical networks separated by *boundary* or *transparent* clocks. The first act as a protocol slave on one side (Net A) and master on the second (Net B) for all slaves (e. g., slave B). In contrast, *transparent* clocks do not separate networks

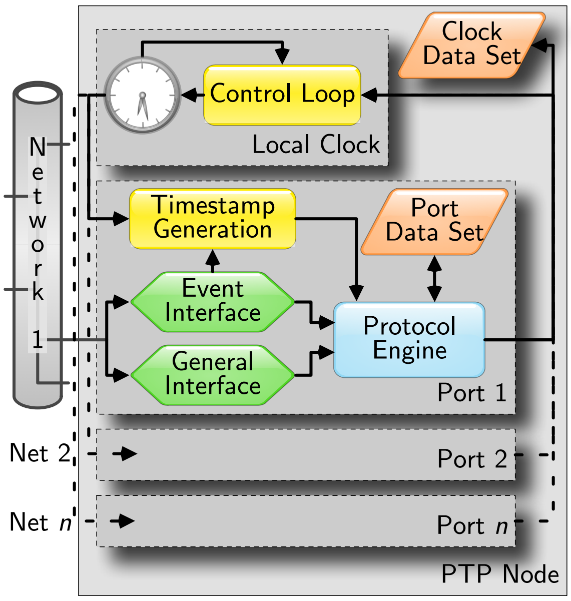


Figure 3.8: General PTP node internal structure

according to the protocol, but correct the residence time of all messages that need to be timestamped. Therefore, e. g. slave C in the figure has the same master as slave B, but both are separated from slave A through the *boundary* clock. The ultimate source for a whole PTP network is called *grandmaster* clock, which is then the master for all ordinary clocks in its subnet and *boundary* clocks.

Ordinary clock nodes have an internal structure as shown in figure 3.8 and can communicate over a single PTP port with the network. Besides the two virtual interfaces for receiving messages, whereas the *event* interface is capable of timestamping messages, the port runs a protocol engine. The latter has access to the clock and can either adjust or read it, depending on the current port state (e. g. MASTER or SLAVE).

The extension to *ordinary* clocks are so-called *boundary* clocks, which have the same structure but can synchronize over several PTP ports. The connected networks are even not required to run the same communication technology. This type of node maintains the timescale used in a defined PTP domain. Per port it may either serve as a source of time or synchronize to an upstream. The BMC algorithm makes sure that *boundary* clocks only synchronize to messages on a single port, which is then run in the SLAVE state. On an arbitrary number of other ports it may serve as a master clock or, if there are redundant paths in the network the affected ports can also be set to PASSIVE state to ensure a single upstream to the grandmaster node.

Due to the fact that a *boundary* clock synchronizes as slave to an upstream node and propagates the time of its own clock as PTP master to other nodes, it introduces an additional control loop between the grandmaster and a dedicated slave. In large PTP networks, a significant number of cascaded *boundary* clocks can show the same problematic behaviour as a chain of control loops, thus leading to instability and decreased synchronization performance [JSW04]. This issue was addressed in version 2 of the IEEE 1588 standard with the introduction of network elements being capable of compensating for the residence time on the node.

These so-called *transparent* clocks measure the time it takes a PTP message requiring timestamping to transit the device. The gathered information is then passed on to the clocks finally receiving the message. The standard defines two different types of *transparent* clocks. The first is the *end-to-end transparent* clock, which only corrects transiting PTP messages. The more enhanced version, the *peer-to-peer transparent* clock, additionally evaluates the link delay connected to each port of the node. This peer-to-peer delay measurement mechanism can then replace the standard master to slave delay measurement (see next section) by summing up all individual link delays the messages is transmitted over. Thus, the network can react much faster to a change in the master, since all distinct link delays are known and the master to slave delay does not need to be re-evaluated.

Protocol Operation

The protocol distinguishes between *general* and *event* messages. While the first are solely used to transport information between the nodes, the latter require to be timestamped by the local clock of the node on transmission (egress timestamp) and reception (ingress timestamp). As mentioned in chapter 1.4 the timestamping point can be implemented on different layers according to the OSI reference model and evidently influences the achievable accuracy.[5] Furthermore, depending on the available hardware support for timestamping, the PTP software stack can make use of one of the following two methods:

Two-step clock. If the hardware of the node is not capable of manipulating timestamps in PTP event messages, the software stack has to inform the remote side about the exact egress timestamping using a Follow_Up message. This is also due to various unknown delays between the generation of an event message and its appearance on the transmission medium, which does not allow prediction of the exact send time. Consequently, the egress timestamp is measured on the event message and passed on to the remote side using a Follow_Up message subsequently.

One-step clock. The PTP software inserts estimation values for the transmit timestamp field in event messages. During transmission specialised hardware measures the timestamp for the Start-of-Frame Delimiter (SFD) and replaces the rough time information with the accurate egress timestamp. Therefore, the Follow_Up message can be omitted, which is especially interesting for low-bandwith and pure hardware implementations of the protocol. One-step clocks are simpler to implement, since there is no need for egress timestamp buffering or Sync to Follow_Up message matching procedures.

Figure 3.9 shows the basic concept and message exchange for the PTP clock synchronization. While solid lines represent event messages requiring timestamping, dashed

[5]Unless otherwise specified in a transport-specific annex to the standard, the message timestamp point for an event message is the beginning of the first symbol after the Start-of-Frame Delimiter (SFD).

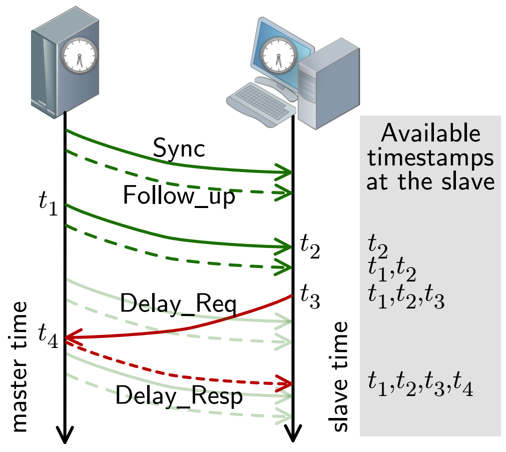


Figure 3.9: Basic synchronization messages for PTP offset and delay calculation

lines refer to general messages, which are used for data exchange only. As mentioned above, the actual presence of Follow_Up messages is dependent on the clock (one-step or two-step) type. Please note that the figure is drawn for the more general case of a two-step clock. For the case of a one-step clock the Follow_Up messages can be skipped and assumed that the Sync message egress timestamp, t_1, is immediately available.

The complete sychronization process is actually handled in two steps running (almost) independently of each other: syntonization and line delay compensation. Fist, the master syntonizes the slave by sending out Sync messages in regular intervals of $T_{\text{sI}} = 2^{\text{syncInterval}}$ whereas syncInterval should be chosen from $[0, 5]$ for version 1 and $[-7, 7]$[6] for version 2. With the assumption that there is no line delay, the slave can adjust its local clock control loop with each Sync message in a way that on reception $t_1 = t_2$ can be fulfilled. Consequently, the slave is syntonized to the master, meaning that both nodes run at the same clock frequency.

In regular intervals[7] the slave initiates a transmission delay measurement by sending a Delay_Req message. This request is answered by the master with the ingress timestamp, t_4, encapsulated in a Delay_Resp message. Subsequently, the slave knows all necessary

[6]The actual value is further restricted by a relevant profile. The default profile specifies an initialisation value of 0 (once a second), which can be altered by ± 1. For unicast implementations a default value of -4 out of the range $[-7, 1]$ is recommended.

[7]The interval is specified to have a mean value of $2^{\text{logMinDelayReqInterval}}$ s, whereas the actual transmission time should form a random uniform distribution with the afore mentioned mean value. The randomisation is used to avoid packet collisions and overload of the master with a high number of slave request in a short interval.

timestamps to calculate one-way delay, d, and offset, o, according to

$$d = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} = \frac{(t_4 - t_1) - (t_3 - t_2)}{2}, \tag{3.3}$$

$$o = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} = (t_2 - t_1) - d. \tag{3.4}$$

The method requires that the transmission delay on the communication path between master and slave is symmetric. This assumption might not hold for several reasons. On the one hand the network might use different communication speeds for each direction and therefore the delay scales with the bit rate. In order to overcome this problem a measurement principle based on block burst transmissions has been proposed in [Lee08]. On the other hand, for high-accuracy clock synchronization, even the asymmetry of cables (different wires used for receive and transmit can cause up to 50 ns per 100 m for twisted pair cables [IEC08]) or signal filters and clock recovery mechanisms in the physical layer device can have significant influence on the accuracy of the link delay measurement.

3.3 Real-Time Networks

Since the thesis is about *enhanced* clock synchronization performance, in the following a selection of real-time protocols are listed, giving better performance than implementations on top of standard Ethernet. Since for example Ethernet/Industrial Protocol (EIP) or Modbus Transport Control Protocol (TCP) do not require special functionalities on layer 2 or below, they are not considered in this context. Like many other protocols, they are based on an original fieldbus specification [Sau10] and are currently standardised in multiple parts of IEC 61158 [IEC07].

3.3.1 PROFINET

From a historical perspective, ProfiNet is an industrial Ethernet solution that evolved from the fieldbus called ProfiBus. The communication system is maintained by ProfiBus and ProfiNet International (PI) and standardised in IEC 61158 Type 10 [IEC07]. For different application requirements the organisation has specified different communication channels providing appropriate parameters for communication. All have in common that they use predefined cycles for deterministic data transfer. The separation of standard and real-time traffic in a slotted scheme makes it possible to safely integrate the widely used TCP/IP traffic of legacy applications. Nevertheless, existing networks have to be integrated through proxies. The three available transmission schemes are:

- Standard TCP/IP communication, which allows for cycle times in the range of 100 ms – This method is used for device parametrisation and configuration, reading of diagnostics data and negotiation of the communication parameters for real-time applications.

- Real-Time (RT) is used for applications requiring cycle times down to about $5 - 10$ ms. This communication path still uses standard components, but a modified software stack. The real-time frames have their own Ethertype (0x8892) in order to distinguish them from common IP frames. The communication stack as well as special switches give priority to the PROFINET RT frames allowing cyclic (scheduled) communication between the devices.

- Isochronous Real-Time (IRT) for applications demanding cycle times ranging within $0.25 - 1\,\text{ms}$. Additionally, it performs time-based communication with jitter below $1\,\mu\text{s}$. IRT traffic delivers the highest performance of all traffic classes providing isochronous transmission of process data.

While the first two methods are fully handled in software and can use traditional Ethernet hardware, the isochronous real-time mode requires special Ethernet network cards with modified media access. In order to provide accurate synchronized communication, PROFINET IRT uses a clock synchronization scheme similar to IEEE 1588 version 2. The method is called Precision Transparent Clock Protocol (PTCP) and is based on an infrastructure that is composed of cascaded transparent switches. These network elements operate similar to the transparent clock introduced with version 2 of IEEE 1588. PTCP does not define end-to-end delay measurements but uses the peer delay mechanism to evaluate the individual link delays. IEEE 1588 and PTCP can be interconnected via a boundary clock. In the Annex I of the IEEE 1588 version 2 [IEE08b], there is a table that shows the correspondence between names given by the PTP to the synchronization variables/parameters and names given by PTCP to the same objects [FFM+08]. This illustrates the close relationship between the two protocols.

For accurate synchronization PROFINET IRT requires hardware timestamping support. Consequently, the system is bound to special ASIC implementations [Sie08] like the Enhanced Real-Time Ethernet Controller (ERTEC) chip available from different companies. The versions with integrated two or four port switch support hardware timestamping for accurate packet delay measurements. This support is indispensable to allow the communication system to provide timely accurate process data.

3.3.2 TTEthernet

As with the previous real-time Ethernet solution, TTEthernet is the (time-triggered) Ethernet representation of a former fieldbus system called Time-Triggered Protocol (TTP). TTEthernet combines proven determinism, fault-tolerance and real-time properties of the time-triggered technology with the flexibility, dynamics and legacy of *best effort* of Ethernet and is therefore suited for all types of applications. [TTE09] The specification [Ste08] gives detailed information about the architecture, the synchronization protocol, and the data flow. It is fully compatible with standard Ethernet nodes and works on top of all physical layers specified in IEEE 802.3 [IEE08a] for switch-based networks. The required protocol mechanisms can be implemented in hardware or software depending on the available resources and real-time requirements.

In TTEthernet the switches are responsible for organising all data communication and therefore play a central role. All nodes deliver messages directly to a port of the switch, which is then responsible for keeping the real-time constraints. They organise data communication based on the preconfigured constraints in a way that the bandwidth is always sufficient and transmission interrupts are reduced to a minimum. The time-triggered architecture specifies three types of messages to satisfy different application needs:

Time-triggered messages. This type of message is sent over the network at predefined times. The frames have the highest priority and therefore take precedence over all other traffic classes. The timing of the individual communication links has to be precisely planned at the time of the system design. Tool supported configuration mechanisms ensure that resources' conflicts during runtime are avoided. Consequently, the frame delay is predefined and can be guaranteed. In this way, the system can ensure precise temporal messages delivery.

Rate-constraint messages. Applications that require a pre-defined bandwidth can use rate-constraint message flows. This type has lower priority than time-triggered messages, but still delays and jitter can be guaranteed to stay within preconfigured limits. The issuing of messages is planned off-line as well, but the individual frames are allowed to be queued and delayed during run-time. Since the configuration is known in advance, the temporal delivery can still be calculated and message loss is prevented.

Best-effort messages. This traffic class uses the remaining bandwidth of the network and has lowest priority. There are no guarantees concerning packet-loss or temporal delivery like in standard Ethernet implementations. All legacy TCP/IP traffic will automatically be assigned to this class.

Regarding clock synchronization, the system represents a hierarchical master/slave network that has distributed, fault-tolerant masters providing timing accuracy better than 1 µs and a precision of < 100 ns to the system. The cycle times can therefore be in the range of 10 µs, depending on the actual physical layer and the configuration. Defined masters send out synchronization messages, which have a priority equivalent to rate constraint frames. All switches act as a transparent clock, which delay synchronization messages always for the (pre-calculated) worst-case scenario. The static configuration also allows to avoid line delay measurements. Using the hierarchy, the schedule, and an optional line delay parameter, the static transmission delay can be calculated in advance. The transparent clock mechanism then compensates for network queues and other dynamic delays.

Besides the specified *transparent* and the *fault-tolerant* synchronization protocol the architecture allows also the implementation of IEEE 1588 master/slave synchronization mechanisms on top of the TTEthernet protocol. This is realised by a device that sends out IEEE 1588 synchronization messages based on the timing information gathered from the TTEthernet system.

Concluding, the protocol provides the means to maintain a global time among all devices in order to provide the time-triggered services in the network. This common time forms the basis for the other system properties, as for example temporal partitioning, diagnosis, or resource utilization. Since the protocol is dependent on the precision of the underlying hardware, the properties are crucially defined by the accuracy of detecting network events (drawing timestamps).

3.3.3 EtherCAT

The speciality of EtherCAT, specified in IEC 61158 Type 12 [IEC07] is the on-the-fly processing of Ethernet frames. Each EtherCAT device features at least two physical Ethernet ports. The input is read and processed while it is output to the second port. This causes only a very low delay (several bit-times) per device. The master sends out one frame, which can address several nodes at once. This frame is then passed from one node to the next. Each slave device can read out the data addressed for it and replaces it with the reply. Since the system always operates as a (virtual) ring, the frame from the master is always looped back and can transport the replies, as well as inter-slave communication. The design is very efficient for distributed digital I/Os [Pry08] and allows short cycle times. For a number of 1 000 I/Os the update time can be estimated to 30 µs [eth09].

The local time of the nodes is held in a 64 bit counter representing the absolute time in nanoseconds starting from 2000 Jan 1st 00:00. Low resource implementation may also use a single 32 bit counter, which gives relative timing information for 4.295 s, which is typically enough for timestamping during one communication cycle. The master can then reassemble the timing information to the absolute time. Typically the first EtherCAT slave controller after the master is responsible for maintaining the reference clock for all other devices. The actual synchronization on the EtherCAT bus runs in three steps [Bec09]:

Propagation delay measurement. Each individual EtherCAT slave device introduces a small forwarding delay of the packet travelling through the (virtual) ring structure. Additionally, the cabling between the devices introduces delay, which has to be compensated. The system accomplishes this by assuming that all devices have the same frame propagation delay and the wire delays are symmetrical. Consequently, a single frame sent out by the master is timestamped on the ingress port of both directions by each device in the ring and can then be used to calculate all inter-device delays of a system.

Offset compensation. Knowing the propagation delay, each device can easily compensate the local clock offset by calculating the difference between the received time and the ingress timestamp drawn from the local clock. The corrected time is then used by the device as the local copy of the system time.

Drift compensation. After the delays and the offsets have been evaluated, the natural drift of the local clocks is compensated by the time control loop. The latter adjusts the speed of the nodes' local clock in order to compensate for different rates of the driving oscillators.

Since EtherCAT uses the standard physical layer of IEEE 802.3 (100 Base-TX or 100 Base-FX is required) all clock synchronization functions are dependent on the behaviour of the (R)MII interface. Due to the on-the-fly processing and the delay evaluation based on round-trip measurements, the PHYsical layer entities (PHYs) must not modify the Ethernet preamble length. Internally an EtherCAT device draws timestamps from the (R)MII using the local clock to sample with 10 ns resolution. For enhancing synchronization quality, recommendations for the physical layer are given in [Bec09]:

- Reduced Media independent Interface (RMII) is discouraged since it introduces additional FIFOs at the physical layer, which increase the forwarding delay, as well as the jitter.
- A common clock source for all PHYs of an EtherCAT device avoids additional clock transitions within a device.
- The clock signals of the PHYs should have a fixed phase relation to the clock input. Especially for the transmit clock, this allows to improve the accuracy of the timestamps.

Slave implementation requires only low hardware costs, because the system acts as a single, simple registered mapped device to the master. Additionally, the master can be implemented in software on any standardised Ethernet Network Interface Card (NIC). Indispensable hardware operations, like the keeping of the reference time are then taken over by on of the EtherCAT slave controllers.

3.3.4 Ethernet POWERLINK

With respect to the OSI/ISO Reference [ISO94] POWERLINK [Kir08a], adds a sublayer to the data link layer residing on top of the standardised Ethernet MAC. The protocol expands the Ethernet CSMA/Collision Detection (CD) scheme with a polling and time-slicing mechanism, which safely avoids collisions and ensures real-time guarantees. Since collision detection is no longer required, the maximum span for Ethernet segments does not apply any more. Therefore, POWERLINK can support large and arbitrary topologies including star and line.

The management node (MN) of a system organises the complete message exchange in a deterministic, pre-planned way. All controlled nodes (CN) are only allowed to respond to queries of the management node. Consequently, collisions and subsequent retransmissions cannot happen as in pure IEEE 802.3 implementations allowing to give real-time guarantees. Figure 3.10 shows the basic communication cycle. After handling the isochronous phase, the protocol polls one device, which can then send a generic Ethernet (Ethertype other than 0x88AB) packet over the network. A proposed extension [Kir08b] allows multiple asynchronous messages per cycle providing more efficient use of the bandwidth in case of configurations with long idle phases (low amount of real-time traffic required).

All nodes synchronize the communication to the Start of Cyclic (SoC) frame at the beginning of each cycle, which can be down to 100 µs (for 100 Base-TX) duration

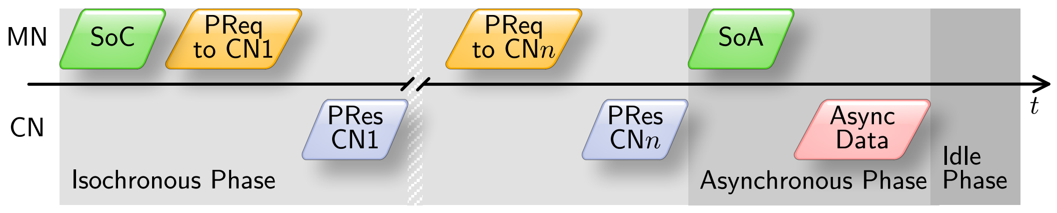


Figure 3.10: One complete POWERLINK cycle starting with a *Start of Cyclic* (SoC) frame. Communication is implemented using *Poll Request* (PReq) and *Poll Response* (PRes) messages in the isochronous phase. After the *Start of Asynchronous* (SoA) frame the exchange of one arbitrary packet is allowed.

depending on the used physical layer. Due to the implementation on top of the data link layer the protocol can achieve synchronization accuracies in the range of 1 µs even without hardware support [pow08]. Still this solution demands the use of hubs, which have far less communication jitter than switches, which are required to do frame processing.

Due to the strict master/slave mechanism, the time-slicing of POWERLINK does not depend on the synchronization quality, like other TDMA-based real-time systems at the costs of low bandwidth utilisation efficiency. The relative simple implementation makes the architecture performance [CSVV09] very dependent on the maximum delay and jitter on the path from the controlled node with the most hops to the MN. The latter has to wait for the maximum transmission time plus some safety margin in order to avoid collisions due to late responses from a controlled node.

3.3.5 SERCOS-III

The most characteristic features of the SERCOS interface are the line or ring topology and the on-the-fly processing of frames [ser07]. Each SERCOS system consists of one master handling up to 254 real-time enabled devices. All slave devices feature two Ethernet ports. One port receives data, the slave processes it on-the-fly and forwards it (with small delay) to the output port. This allows to build a chain or ring with very small communication time, i. e. short cycle times. Further, the structure makes hubs or switches dispensable and avoids collisions.

The master communicates to the slaves using a single Ethernet telegram, which contains command codes and allows the latter to insert the answer into special fields. Using this method also direct slave-to-slave communication can be handled efficiently. For real-time communication the protocol replaces the standardised MAC of Ethernet.

Concerning synchronization, SERCOS-III provides high performance because the processing engine directly processes and forwards data from/to the IEEE 802.3 physical layer. This avoids delay and jitter introduced by higher layers. The accuracy therefore is only dependent on the performance of the physical layer. SERCOS-III can guarantee accurate real-time execution of commands due to a synchronization jitter better than ± 10 ns and a simultaneity of < 100 ns network-wide [Bio06].

Summing up, this real-time network is built for fast communication and accurate command execution, but can only transport small data blocks per device. Typical applications are the control of moving axes of robots. Legacy traffic can be integrated at the end of each cycle, as done by e. g. POWERLINK.

3.3.6 LAN eXtensions for Instrumentation (LXI)

The LXI Standard [lxi08] defines a set of rules for inter-device communication on Local Area Network (LAN) developed for but intended not to be limited to instrumentation and measurement. The driving intention is the replacement of current interconnection methods, e. g. GPIB, which have drawbacks like low communication bandwidth, complex connectors, and limited node numbers. Since the standard also defines certain Application Programming Interface (API) and interface rules, also devices supporting e. g. GPIB, PCI eXtensions for Instrumentation (PXI), or standard LAN can be included. Depending on the application, three different functionalities may be implemented in a device [Ple05]:

- A web interface allows accessing and automated control of the device functions including configuration features. The functionality is made available via the standardised LAN interface, which should support peer-to-peer operation as well as master/slave operation. Devices that comply with the required rules are called LXI *class C* compatible.

- If an instrument additionally to the previous point is able to maintain a common notion of time via IEEE 1588 it is called *class B*. Furthermore, the device has to be able to timestamp actions and initiate trigger events via the LAN interface. This type of devices support time-based triggering. Triggers are configured via the network in advance and are then executed at the pre-scheduled time.

- *Class A* devices also feature a physical wired trigger system based on a multipoint Low Voltage Differential Signalling (LVDS) electrical interface. It allows triggering between the connected modules via twisted pair transmission lines. This method is recommended for the highest accuracy requirements (< 40 ns).

The better class always includes the features of the lower one as well (A includes all). The specification allows to send trigger events over LAN using User Datagram Protocol (UDP)/IP and TCP/IP. Obviously, that does not satisfy high-precision requirements. Consequently, the wired and IEEE 1588 triggers are used to address latency issues. LXI supports various modes of triggering, besides the direct ones used together with the trigger bus and software, also time-based triggers. The latter are dependent on the local timebase and the synchronization quality when used together with other modules.

Again, the accuracy of the system depends on the timestamps. The specification does not give rules about methods to acquire timestamps, but requires the system to report information about the available accuracy. Timestamps should be derived from the IEEE 1588 clock with a precision that is consistent with the event or data acquisition process and the resolution of the clock. [lxi08] The finally achievable accuracy is subject to the (hardware) implementation of the timestamping engine.

3.4 Hardware Elements

With the acceptance of IEEE 1588 in the industrial world, some chip manufacturers started integrating hardware support for accurate timestamping into their products. Although, this would have been already possible with version 1 of the standard, the growth began with version 2. This is mainly due to the fact that the concept of transparent clocks (see chapter 3.2.2) allows highly accurate synchronization between two nodes, even with an intermediate switch.

NTP was originally designed for pure software implementations and by design does not implement necessary operations (e. g., two-step clocks or transparent clocks) for hardware support. Hence, only a few dedicated realisations with explicit hardware support exist that are often special embedded systems [SJH01, Joh04] with a single timebase and on-the-fly timestamping. Such systems are built for the single purpose of a precise node for NTP and the hardware and software implementation have to be matching for a given version (i. e., especially the timestamp format). Most products therefore are specialised NTP stratum 1 servers in order to provide a precise time reference. Only in rare cases, clients are modified to support hardware timestamping as well.

3.4.1 National Instruments Synchronization Modules

National Instruments Corporation[8] currently offers a variety of add-on cards to upgrade existing Peripheral Component Interconnect (PCI) or PXI [PXI04] systems with synchronized clock signals and event timestamps.

The first generation device, the NI PCI-1588 timing and synchronization module was designed to generate events and clock signals at specified future times. Further, it is capable of timestamping input events with the internal IEEE 1588 time. Hence, providing a method for performing synchronous actions or analysis for instruments connected via Ethernet cabling.

The card is equipped with three front panel connectors (SMB) and an 8-bit Real-Time System Integration (RTSI) bus, which both can be configured as event input or trigger/clock output. Via the latter several different data acquisition modules featuring an RTSI bus can be interconnected and use the same events for reference.

According to the user manual [Nat05], the card is capable of achieving an accuracy of ±230 ns (peak-to-peak) with 33 ns standard deviation over a direct 3 m Ethernet connection. The values are based on the internal 10 MHz Temperature Compensated Crystal Oscillator (TCXO), which provides a temperature stability of ±2 ppm.

The second type and as well generation of cards are the ones of the NI PXI-66xx series. The timing and synchronization module PXI-6682 provides 47 ns (peak-to-peak) with 10 ns standard deviation over an identical connection as above [Nat07]. The reason for the enhanced synchronization performance is on the one hand the more stable TCXO, which provides a stability of ±1 ppm. On the other hand, the used chip for timestamping signals has a resolution of about 5 ns[9] compared to 16 ns of the older generation.

[8] 11500 N Mopac Expwy, Austin, TX 78759-3504, USA, `http://www.ni.com`
[9] `http://digital.ni.com/public.nsf/allkb/846CDBA8A291526E862573F0001A636E`

3.4.2 National Semiconductor PHY

National Semiconductors[10] currently provides two interesting hardware products, which are useful for high accurate clock synchronization via Ethernet. Both physical layer devices, the DP83848 and the DP83640 provide deterministic delay behaviour measured between the two MII of an Ethernet link.

The first device provides deterministic cable signal to xMII timing that avoids varying delays on link reestablishment. Applications, which use the MII signals to recover timing and for timestamping, benefit from the fixed delay [Ros06] to the signals on the cable with better synchronization precision due to the higher symmetry of the link.

Additionally, the second product (DP83640) also incorporates an IEEE 1588 compatible clock providing a timebase as well as trigger generation and capture event timestamping. Although, the integration of a clock synchronization core into the PHY could provide additional accuracy, the specification data [Nat08] suggests that no benefit was taken from this integration aspect. The timestamps provide an 8 ns resolution, which originates from the 125 MHz system base clock. Despite the fact that the device is not sufficient for high-accuracy clock synchronization, it provides a perfect means to enhance existing designs with IEEE 1588 compatible clock synchronization features. Improving working devices only requires a replacement of the PHY and additional driver support for read-out of timestamps via the MII interface. This even allows to add synchronization functionality to small embedded systems using a micro-controller.

3.4.3 Intel Gigabit Ethernet Controller

Intel[11] launched the IXP46X series of network processors in the year 2005 and two years later the 82574/82576 Gigabit Ethernet controllers. The first network processor supports up to three Ethernet interfaces and version one of the PTP standard. It uses an internal 66 MHz clock for timestamping and timekeeping, which results in a resolution of about 15 ns. Using an appropriate off-chip algorithm, the deliverable best-case synchronization accuracy is stated to be in the range of $25 - 100$ ns [Int05].

For the 82576 Gigabit Ethernet controller, the datasheet [LAN08] does not give explicit numbers for the achievable synchronization quality, just the term "sub-microsecond" accuracy. Nevertheless, the mentioned timestamping resolution of about 16 ns suggests similar performance as with the successor product but for all available Ethernet transmission speeds.

All products also support general purpose I/Os as well as precise interrupt generation to take advantage from the synchronized timebase. Unfortunately, all mentioned devices suffer from an important design issue. The synchronization logic is only able to store a single timestamp (per direction). Thus, in case the required software application is not responsive enough to read out the packet timestamp until the next arrives, the time information for the latter gets lost. This makes synchronization system implementations

[10] 2900 Semiconductor Dr., P.O. Box 58090, Santa Clara, California, USA, `http://www.national.com`
[11] 2200 Mission College Blvd., Santa Clara, CA 95054-1549, USA, `http://www.intel.com`

extremely sensitive to the load of event packets (frames that have to be timestamped). Master or high-accuracy nodes do not seem to be feasible.

3.5 Implications

The state of the art shows that the trend definitely goes towards Ethernet-based solutions for sensor and control networks as well as test and measurement solutions. Both application areas increasingly require higher accuracy to fulfil more demanding requirements. Better temporal performance allows lower guard times, faster network update times, and more precise coordination between individual nodes of a real-time network. For test and measurement setups the precision of coordination of the individual nodes' timebases directly influences the overall result gathered from distributed instruments.

So far, the state of the art for hardware assisted clock synchronization is to draw a timestamp at the time the hardware starts sending/receiving a frame to/from the medium. The rather simple approach is to detect this event by (over-)sampling using the local clock, which allows to reach synchronization accuracies far below comparable software approaches. Still this does not push the technology towards applicability in scenarios requiring sub-nanosecond accuracy.

Current implementations use a straightforward approach by implementing a counter as the timebase and the mentioned sampling of the ingress/egress packet events. The timestamps are then provided via a register interface to the software application of the protocol stack. As mentioned above the accuracy, which is achievable in the best-case is in the range of several nanoseconds if stabilised oscillators and high sampling rates are used.

The approaches utilised in state-of-the-art devices are reasonable but are not based on a systematic analysis of involved components and influence factors. A scientific way of exploring the topic of high-accuracy clock synchronization therefore has to characterise the properties of the transmission media, the required system components, and the overall architecture. As a result optimised implementations as well as data about the maximum reachable performance can be identified.

The developed timestamping method for layer 2 still depends on the properties of the physical layer and has to consider them in order to provide precise event detection suitable for highly accurate clock synchronization. Furthermore, the utilised hardware and software components have to be designed to interact in an efficient way. This especially means that the task of synchronizing clocks, which is distributed over several layers of the ISO/OSI network reference model, has to be treated as a system aspect and therefore designed in an integrated manner. The upcoming chapters of the thesis show this development by means of characterisation, modelling/simulation, and prototype implementation.

4 System Characterisation

For the purpose of synchronizing a distributed system, the timely properties of the message exchange are important. The absolute transmission delay and its variation (the jitter) are significant factors for the performance of a network regarding clock synchronization. While absolute delays can be measured and then compensated using various techniques mentioned in the previous chapter, jitter, like any other noise is the source for inaccurate clocks. Consequently, any element in a clock synchronizing system contributing to uncertainties in the detection of external events, i. e. message transmission, has to be identified, characterised, and then weighted according to the contribution to the overall jitter.

Most issues with the accurate detection of events are related to the implementation of the network node, but there are also contributions resulting from the Ethernet transmission standard used on the medium. This and the intermediate network elements, in case no point-to-point connection is used, have to be considered outside the factors originating from the node itself.

The present chapter analyses the involved components and their particular additions to the overall jitter of the message exchange in a distributed system. The precision of the hardware, which delivers the timing information, is crucial for the performance of the control loop. Not only the noise of the elements itself, but also the transitions between individual clock domains have to be considered as limiting parameters for the achievable accuracy.

4.1 System Immanent Jitter Sources

Figure 4.1 shows the jitter sources and clock transitions for a network node with Ethernet hardware timestamping support. Since the definition of Ethernet does not foresee any point for timestamping, the figure exemplarily shows the hardware support for timestamping on the MII, which is the first physically accessible digital interface defined in the standard. This is represented by the Media Independent Interface Scanner (MIIS) block between the physical layer device and the MAC, which is responsible for drawing timestamps when a packet is received from or transmitted to the medium.

Although the block diagram is not an exact representation of all possible jitter sources, it gives a good overview of the different categories of the sources for timestamping inaccuracy. It also illustrates the possible timestamping points with reference to the layered approach of the OSI Reference Model. In general it has to be mentioned that the closer the timestamping occurs to the actual ingress line interface and its clock, the less jitter can influence the measurement of the transmit/receive time instant. Nevertheless, the actual

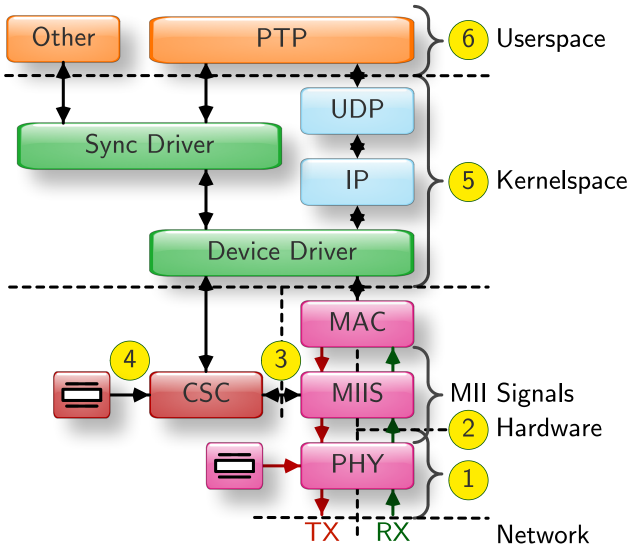


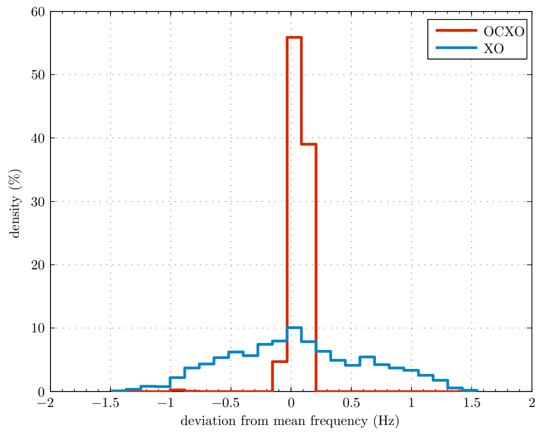
Figure 4.1: Jitter sources and clock transitions for hardware supported synchronization using an application layer protocol like NTP or PTP

implementation depends on the additional effort required over an implementation using COTS chips that is acceptable to the given application.

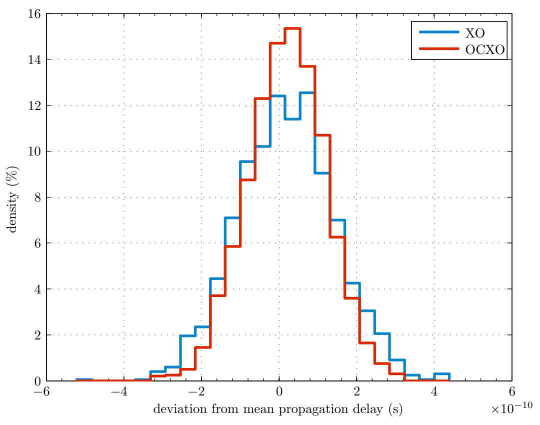
Due to time constraints of the CSMA/CD scheme used in Ethernet, most network adapters do implement the physical as well as the MAC layer in hardware in order to fulfil the short response times required. Following this common way of implementing Ethernet, hardware support for clock synchronization is defined using specialised blocks on layer 1 or 2 to increase the precision of timestamps for messages.

Hardware supported timestamping is therefore reasonable in the physical layer device ①, as shown in figure 4.1 on the MII ②, and in the media access controller. The transition of an ingress frame to the software processing domain is either done via polling the hardware or using IRQ routines. No matter which of the methods is used, this part of the software (the Operating System (OS)) is the earliest point in the software stack to draw a timestamp. Following the path of a message through the stack, a timestamp could be drawn at every function block involved in the processing of the packet. Since modern systems are normally split into a privileged processing part (kernel space of an OS) and a part running the actual applications (user space), there is also a transition (due to scheduling) between the two code execution areas. The last instant where a timestamp can be drawn is then the implementation of the protocol stack at the application layer running in the user space. It suffers from all jitter sources located in the layers below and thus is the most inaccurate way of getting timing information about incoming or outgoing events of the node.

While the figure in particular shows the transitions between individual clock domains in hardware, the different scheduling pools in software are not explicitly marked, since they



(a) Frequency distribution of the oscillators



(b) Delay behaviour of the physical layer

Figure 4.2: Histograms using 2000 samples showing the influence of the oscillator type on the transmission delay on the media

are very implementation specific and not standardised. Hence, only the most common two pools (which also differ in the privileges), i. e. kernel and user space, are shown. In the following list, detailed explanations for the jitter sources shown in figure 4.1 are given beginning at the bottom:

- Starting with the packet-oriented network, a lot of jitter is introduced by the intermediate network elements required for packet relaying. Depending on the layer they are operating on, the uncertainty can be in the range of some nanoseconds, e. g. multiport repeaters (hubs), up to several milliseconds, if (software-based) routers are involved. Not only the fact that packets are processed between the endpoints of a synchronizing system can add jitter in the network, but also varying communication paths, if no cell switching technology is used, can contribute. Further, there might be translations to other transmission technologies (e. g., satellite links) or tunnelling through networks (e. g., telephone networks), which can give large variations in the overall packet delivery time. Summing up, this item collects all jitter introduced by a communication over the network compared to a direct point-to-point link.

- The PHY[1], i. e. the physical layer device in COTS network cards, requires a transmit clock in order to generate the symbols on the transmission line. The oscillator for this clock suffers from typical noise phenomena [All87]. For the minimum requirements, the IEEE 802.3 standard [IEE08a] specifies it to be in the range of 25 % of the nominal data rate ±100 ppm. The variations in the transmit frequency directly result in different delivery times of a frame over a layer 1 link. In addition to the jitter from the data source, the clock recovery, e. g. a PLL, also generates noise on the receiving side. The type of the clock recovering circuit and the way it

[1]Within IEEE 802.3 the PHY contains the functions that transmit, receive, and manage the encoded signals that are impressed on and recovered from the physical medium.

locks to the clock signal embedded in the incoming data stream, is important to the internal frame recognition ability and all interface signals of the physical layer device. The influence of all these effects related to signal generation and recovery on the media is summarised in point ① of figure 4.1.

Figure 4.2(a) shows the distribution of 2000 samples of the frequency of two different oscillators (Crystal Oscillator (XO): 50 ppm, Oven Controlled Crystal Oscillator (OCXO): 0.3 ppm). The frequency was obtained by calculating the mean value over 1 s. Both were measured in a temperature stabilized environment (± 0.05 K). Nevertheless, the measurement points out that the behaviour of the oven controlled oscillator is still better than the XO, although temperature effects were cancelled out. Unfortunately, the resolution of the measurement device does not allow to accurately determine the stability of the OCXO.

Supplied by one of the two above mentioned clock sources, an exemplary physical layer device (SMSC[2] LAN8700 [SMS09]) shows a standard deviation of $\sigma_{\mathrm{PHY}} =$ 124 ps using the standard oscillator and $\sigma_{\mathrm{PHY}} = 100$ ps for the OCXO, respectively. Figure 4.2(b) gives the distribution of the delay between two PHYs measured from the rising edge of the transmit enable (TX_EN) to the receive data valid (RX_DV) signal of the MII. In order to achieve accurate results, a frequency counter with 25 ps precision supplied by a GPS disciplined Rubidium frequency standard was used in the measurement setup.

- Similar problems to the above mentioned ones occur in combination with the oscillator for the CSC. Obviously, the device itself again suffers from noise, as any other oscillator. Second, the CSC is required to run at a frequency, where the period is less or equal to the minimum time between two events that are to be distinguished. The generation of external events, like the 1 PPS signal, is also dependent on this clock rate. In order to get a high internal frequency, the block upscales the oscillator frequency using a PLL. The latter as well as the technology dependent jitter of internal registers and output buffers contribute to the inaccuracy of the timekeeping device, i. e. the node's clock. These issues are marked with point ④ in figure 4.1.

- One transition which is standard dependent and cannot be avoided is the translation of the line speed and encoding to the lower speed interface towards the MAC which is shown as point ②. The two interface standards for Ethernet are MII and GMII. There are reduced versions of these standards, which require less pins due to a minimised number of signals that are called RMII and RGMII, respectively. Nevertheless, both are not covered by the IEEE standard[3] for Ethernet and they only use one clock for receive and transmit, which requires the introduction of elastic buffering on the ingress path of the physical layer device. Consequently, an

[2]Smart Mixed-Signal Connectivity™, 80 Arkay Drive, Hauppauge, New York 11788, USA, `http://www.smsc.com/`

[3]The reduced versions of the Media Independent Interface standard were only specified by two documents [RMI98, Red02] of the RMII industry consortium covering several big manufacturers.

additional clock domain transition is added by the transceiver, which is not visible to the event detection block (MIIS), hence introducing additional jitter.

While RMII is an interface synchronous to the local oscillator, MII uses the recovered clock from the line on the receive path. Since the clock rate is 1/4 of the bit rate on the line, the data on the interface experiences one to five bit times latency. This results from the fact that the incoming data clock is recovered with an arbitrary phase which results in five possible edges (8 ns apart) the system can lock to. On 100 Mbit/s Ethernet this edge is normally chosen once, when the link is established, because idle patterns on the line ensure continuous clock supply. For 10 Mbit/s the process has to be done with every packet. Some physical layer chips do an additional alignment to the symbol rate, which then gives a constant delay, avoiding the jitter due to the analogue clock recovery unit [Ros06]. Measurements showing the detailed behaviour of the line and the PHY-to-PHY delay are discussed in section 4.3.

- The interface between the physical layer device and the MAC in general uses two clocks, one for transmission and one for receiving. Commonly, both cannot be assumed to be synchronous to the clock signal of the CSC. Consequently, a clock domain transition from the interface to the timekeeping device is required. It is represented by point ③. The detection of the event has to be either done by oversampling the receive/transmit event signal or in general by determining the phase offset between the node's time source and the interface clocks. Obviously, the MIIS has to adapt to the different data widths of the various interface standards. The implementation details for determining the exact occurrence of an event with respect to the clock of the CSC are discussed in chapter 5.3.

- Point ⑤ refers to numerous timely processing uncertainties introduced by the privileged software layer of all modern OSs, the *kernel*. The code flow of such systems is not linear due to the possibilities of interruptions caused by external events or scheduling. This on the one hand allows to react to asynchronous events in relative short time and on the other hand to run multiple processes using a time slicing technique, referred to as scheduling. Each task switch requires the underlying processor to save the state of the current task and switch to the new one. The time required to create a backup of the old task is the overhead for being able to react to events on demand.

  Assuming that the hardware can generate an IRQ signal, the scheduling pools (with decreasing priority) are for example the IRQ handler responsible for acknowledging the hardware event and scheduling further action, the bottom-half handler that does the actual evaluation of the caused interrupt. Further (not time critical) tasks can then be executed with the same priority as all other tasks in kernel mode. Each of the three layers could be used to generate timestamps in software with decreasing precision due to the fact that the processing could be interrupted by higher priority tasks, e. g. bottom-half handler by an IRQ handler. After the timely issues, message processing is done, e. g. routing decisions, CRC checks, decapsulation of frames in the different protocol layers. These tasks often depend on the message size, previous state, or constraints external to the actual frame. All of the mentioned criteria further add to the jitter introduced by the kernel space.

- While the processing in the kernel space is close to hardware events, scheduling in user space, illustrated in point ⑥ of figure 4.1, is normally done in time slices in the order of milliseconds. One exception is the usage of a real-time OS, which foresees special measures for fast task switching, planned scheduling, or low latency interrupt handling. The time between successive slices for one particular program can therefore easily exceed the required synchronization precision. Additionally, each task can be interrupted by kernel space functions at any time further increasing the timely uncertainty for the execution of time critical commands. The NTP daemon [ntp09] addresses this issue by an estimation of the error which is made due to inaccurate clock readings caused by the rough ticks of the system time. The errors are summed up and then corrected in multiples of the kernel internal tick interval [Mil06a].

All the above mentioned variable delays and jitter sources have influence on the precision of the receive and/or transmit timestamp of synchronization messages. This therefore directly has an impact on the achievable accuracy of the clock synchronization. Besides that, the various uncertainties cause unequally spaced readings of the reference time in the slave timing node. Consequently, there has to be taken care that the violation of the rules for time discrete controllers (equally spaced readings) does not have relevant effect on the synchronization performance.

4.2 Oscillators

Crystal oscillators are the most common clock signal sources and therefore also time references for electronic circuits. They are produced in a wide range of accuracy classes. The easy production and low electric loss led to wide spread usage. Standard oscillators are sensitive to a lot of external influence factors, like time, temperature, acceleration, supply voltage, radiation, and several manufacturing parameters [VB99]. These disturbances especially get importance when high-accuracy clock synchronization is concerned. Here the ability of the clock (which is driven by the oscillator) to hold the local time within a defined accuracy range over the period between two synchronization sampling points is of special importance. In the end, the oscillator is responsible for the accuracy of all intermediate clock readings.

Speaking in short measurement intervals, the approximate time between two synchronization messages, the most important factors are the temperature and the supply voltage variations. Although zero temperature coefficient cuts of the quartz crystal exist, many commercially available oscillators are low-cost AT-cut crystals commonly known as XOs.

Time (ageing of an oscillator) is not a real issue even in a high-precision clock synchronizing system, since the intervals between successive information updates are normally very short (range of seconds) compared to common frequency drifts due to ageing, which are approximately 0.01 ppm/day [ITU03]. All other environmental factors can be either avoided or mainly influence the basic offset in the frequency to the nominal one.

A model describing the frequency $f(t)$ of an oscillator in a realistic way is defined in [SAHW90] by

$$f(t) = f_0 + \varepsilon_{f_0} + a(t - t_0) + f_{\mathrm{n}}(t) + f_{\mathrm{e}}(t) \tag{4.1}$$

with t_0 and ε_{f_0} as the start-time of the observation and the frequency offset, respectively, a the ageing factor, $f_{\mathrm{n}}(t)$ a jitter noise term, and $f_{\mathrm{e}}(t)$ an environmental term. All terms are relatively small with respect to the nominal frequency, f_0, which means that $\varepsilon_{f_0}, a, f_{\mathrm{n}}(t), f_{\mathrm{e}}(t) \ll f_0$.

The frequency offset typically depends on the quality of the crystal and the actual cut, and can be cancelled out by calibrating the clock with a suitable adder-increment. The ageing factor is mainly influenced by the effects like mass transfer due to contamination of the material, quartz outgassing or diffusion effects as well as pressure change in the resonator enclosure [Vig04]. The short-term noise is a noise observed on the phase which is influenced by the power supply of the oscillator and the behaviour of the electrical circuitry itself. Finally, the environmental term covers influences like the temperature dependency of the oscillator and the influence of vibration on the resonance frequency. The dominating parameter for $f_{\mathrm{e}}(t)$ for actual implementations is the temperature dependency.

Using $f(t) = {}^{\mathrm{d}\varphi(t)}/{}_{\mathrm{d}t}$ the phase $\varphi(t)$ of the oscillator can be modelled as

$$\begin{aligned}\varphi(t) &= \int_{t_0}^{t} \left[f_0 + \varepsilon_{f_0} + a(t' - t_0)\right] \mathrm{d}t' + \underbrace{\int_{t_0}^{t} f_{\mathrm{n}}(t')\mathrm{d}t'}_{\varphi_{\mathrm{n}}(t)} + \underbrace{\int_{t_0}^{t} f_{\mathrm{e}}(t')\mathrm{d}t'}_{\varphi_{\mathrm{e}}(t)} = \\ &= (f_{\mathrm{n}} + \varepsilon_{f_0})(t - t_0) + \frac{a(t - t_0)^2}{2} + \varphi_{\mathrm{n}}(t) + \varphi_{\mathrm{e}}(t) \end{aligned} \tag{4.2}$$

where the integral of the phase error due to noise and environmental influences are abbreviated with $\varphi_{\mathrm{n}}(t)$ and $\varphi_{\mathrm{e}}(t)$, respectively.

4.2.1 Frequency Domain Characterisation

One key parameter for the behaviour of oscillators is the utilised technology. Nowadays many types of oscillators are available, starting at the already mentioned low-cost AT-cut crystals used in the commonly known XOs. Here, the main influence factor on the frequency, the temperature, is neither physically stabilised nor mathematically compensated. The latter is done in the so-called TCXOs (analogue) or Microcontroller Compensated Crystal Oscillators (MCXOs) (digital), where the actual temperature is taken into account and the output frequency is corrected with a preconfigured temperature-frequency calibration curve. The most expensive crystal oscillators are OCXOs. This type of oscillators stabilises the ambient temperature of the crystal to a certain temperature value. Additionally – to increase the performance – a special crystal selection process is done for the latter two types.

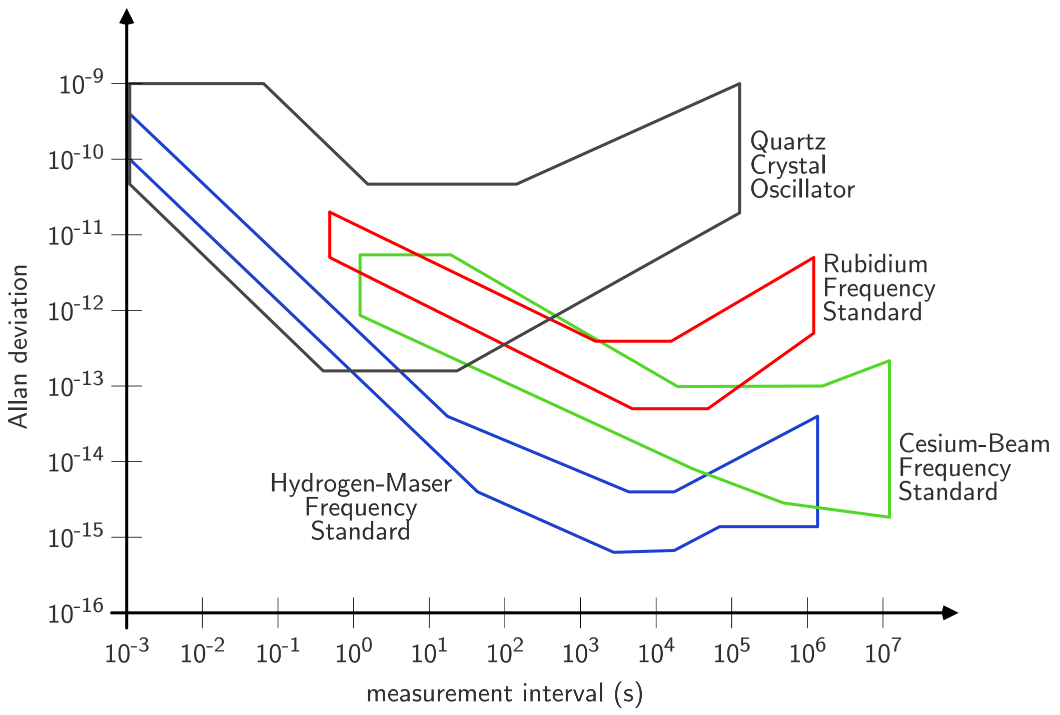


Figure 4.3: Allan deviation of various frequency sources [AAH97], indicating the stability in dependence of the considered time interval.

Besides crystal oscillators it would also be possible to use an atomic resonance frequency standard [Vig04] as the timekeeping element. Examples for such sources would be Rubidium or Cesium atomic clocks or even Hydrogen masers. All three can provide much better long term stability than any available XOs. Unfortunately, these types are rather expensive and difficult to integrate with common Printed Circuit Board (PCB) manufacturing processes due to their size and utilised components. Therefore, if at all, they are used in a timing master to provide a stable long-term reference, which allows to spend the costs only once and benefit at all nodes through clock synchronization. An overview on the achievable precision using the different oscillator type is given by figure 4.3. The used measure, the Allan deviation, is explained in section 4.2.2.

The actual, practically observable, stability of an oscillator depends on several external parameters, such as the above mentioned temperature. Furthermore, supply voltage and mechanical effects like vibrations influence the output signal. All of these effects can be summarised in terms of their impact on the frequency stability or equivalent phase stability. The following considerations take a constant set of these parameters into account and characterise in a first step only the behaviour of an oscillator in a stable environment.

In such an environment the oscillators still exhibit a variety of instabilities which are manifested in phase and frequency changes. A sub-set of these are well-known deterministic trends, like frequency and phase offset, drift, and environmental terms, which can be described with a model as published in [GLS06]. These instabilities cause a linear (or at most quadratic) time error term that can be easily corrected, for example, by clock synchronization with a control loop. However, for a complete model of an oscillator it is also necessary to include the stochastic trends. This noise, mathematically expressed by the autocorrelation of the phase, $\varphi(t)$, with itself, can be categorised depending on the Power Spectral Density (PSD), $S_\varphi(f)$, of the phase noise.

White Noise Family is probably the best known noise family. It is an uncorrelated, bandwidth-limited, random noise process which has a constant power spectral density for all frequencies in its band. However, as technical elements are bandwidth limited, the power of white noise process is also limited.

In particular, two types of white noise are contributing: the commonly called White noise Phase Modulation (WPM) ($S_\varphi(f) \propto 1/f^0$, where f denotes the frequency) and White noise Frequency Modulation (WFM) ($S_\varphi(f) \propto 1/f^2$). The first one usually denotes the phase variations, which appear to be a random white noise processes. Amplifiers usually contribute to this kind of noise in oscillators. Thus, by proper amplifier design, this process can be reduced. The second class of the white noise family is the white noise frequency modulation. Causes for this can be usually found in passive elements of the oscillator [Vig92].

Flicker Noise Family is usually also referred to as pink or flicker noise and can be identified by its a $1/f$ dependency in the PSD. In oscillators, flicker noise processes can be observed in the spectrum near the resonance frequency, which results in phase noise. Similarly, like in the white family, typically two effects of flicker noise can be observed: Flicker Frequency Modulation (FFM) and Flicker Phase Modulation (FPM) [Vig92].

Theories about the cause of flicker noise are relatively vague. Some sources [Rub05] say that FPM, $1/f$, can be related to a physical resonance mechanism in the oscillator. Usually, its source is the amplification stage itself, thus it can be found even in high-quality oscillators.

The spectrum of the FFM has a PSD proportional to $1/f^3$. Its cause is again not fully investigated. Often this type is masked by WFM or FPM in lower-quality oscillators.

Random Walk Noise can be identified by its typical $1/f^4$ spectrum. Again, this noise is relatively close to the carrier frequency and thus difficult to measure. However, the effect, namely the random (but directional) frequency change can be easily observed. Among the causes for this kind of noise are mechanical shock, vibration, temperature fluctuations, and similar effects [Vig92].

Summing up the afore mentioned components as parts of oscillator behaviour the main issue is that the noise of such clock sources is non-stationary. The measure that is used to characterise these noises in the frequency domain is the PSD, which provides information how the average power is distributed over the frequency. Since usually the oscillator noise is modelled by the five types of noises described above, the PSD can be defined with the power law [All87]:

$$S_y(f) = \sum_{\alpha=-2}^{2} h_\alpha f^\alpha, \tag{4.3}$$

where $S_y(f)$ is the one-sided spectral density of the fractional frequency fluctuations (frequency noise) and h_α are constants, which can be used to characterise the oscillator.

4.2.2 Time Domain Characterisation

In order to characterise an oscillator in the time domain, the classical variance cannot be used, since it diverges for a certain type of noise – random walk noise. Therefore, the introduction of a new measure, the so-called Allan variance, is required in order to estimate the stability of a clock. Random walk noise is the typical effect of an oscillator's frequency to depart with variable progression and direction from the nominal frequency. As for this effect the (temporal) mean value does not converge, also the classical variance does not show useful results. The Allan variance solves this issue for all noise types commonly observed in crystal oscillators. It is easy and fast to compute as well as more accurate in estimating noise processes [Eid06, Ste85, LSL84]. The Allan variance σ_y^2 of the fractional relative frequency error $y(t) = 1 - {}^{f(t)}/_{f_0}$ is defined by

$$\sigma_y^2(\tau) = \frac{1}{2(N-1)} \sum_{i=1}^{N-1} (y_{i+1} - y_i)^2. \tag{4.4}$$

In this commonly used estimation, N is the number of samples. All N samples must be evenly distributed within the observation period τ. Likewise, each frequency offset can also be estimated by evaluating the time errors $x_i(t)$. Since $y_i = \frac{x_{i+1}-x_i}{\tau}$ the Allan variance can also be approached by evaluating

$$\sigma_y^2(\tau) = \frac{1}{2(M-2)\tau^2} \sum_{i=1}^{M-2} (x_{i+2} - 2x_{i+1} + x_i)^2 \tag{4.5}$$

with $M = N + 1$ samples of x_i. Using this algorithm the Allan variance has become a de facto standard to predict the quality of clocks. Nevertheless, it should be noted that for the computation of $\{x_i\}$ a reference clock, or at least the information about its phase, is indispensable.

Finally, it has to be mentioned that a relationship between the time and frequency domain exists which is defined by [All87]

$$\tilde{\sigma}_y^2(\tau) = 2 \int_0^\infty S_y(f) \frac{\sin^4(\pi f \tau)}{(\pi f \tau)^2} \mathrm{d}f. \tag{4.6}$$

From this representation it is clear that one additional benefit of the Allan variance is that the type of error (modulation of the frequency) can be identified by the slope of the plot. For example, a slope of τ^{-1} identifies a WPM or a FPM.

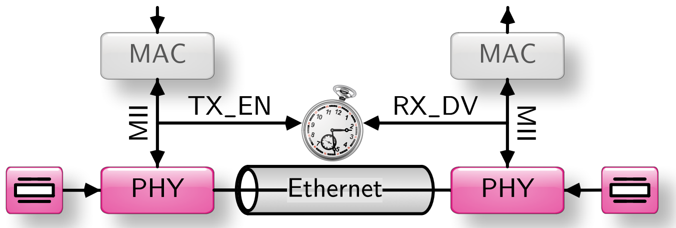


Figure 4.4: Measurement setup for Ethernet physical layer characterisation.

4.3 Ethernet Transmission Standards

The current version of the Ethernet standard [IEE08a] lists a wide range of different transmission speeds and techniques starting at 1 Mbit/s up to 10 Gbit/s. Besides the different speed also the duplex mode, either half or full, and the used transmission media discriminate the various definitions. While the optical version gained popularity with the introduction of 1000 BASE-T, many other solutions did not experience wide acceptance in industry, e. g. 10 BROAD36 (clause 11) or 1 BASE5 (clause 12). The most popular copper-based versions of Ethernet are the 10/100/1000 BASE-T standards. COTS physical layer devices are able to support all three in one chip and therefore it is possible to build triple-speed network interface cards using an appropriate MAC.

Since accurate timing is dependent on source synchronous signals that can be timestamped at the remote node, the reduced versions of the (G)MII are only useful if the timestamping unit can be integrated into the physical layer device itself. This is due to the fact that the receive path of the reduced interface versions uses signals synchronous to the local oscillator. The additional clock transition within the physical layer therefore requires data buffers for clock rate compensation and hinders high precision timestamping. Worldwide only a few number of producers for PHYs exist and among them, only National Semiconductors so far made efforts to support clock synchronization in COTS devices, like the DP83640 [Nat08].

All other implementations are based on the signals of the (G)MII. The translation of the data and timing on the physical line depends on the PHY implementation and the actual Ethernet transmission standard. The understanding of the layer 1 timing behaviour is therefore essential for all high-performance clock synchronization systems attached to the link between the physical layer and the data link layer.

The measurements shown in this chapter were all done on the same physical point-to-point link between two identical PHYs[4]. An accurate time interval counter[5] was used to measure the time between the rising edges of the two (G)MII signals, i. e. TX_EN and RX_DV, which represents the packet transmission delay. In order to avoid external influences as much as possible, network interface cards were placed in a temperature

[4]Marvell 88E1111 integrated 10/100/1000 ultra Gigabit Ethernet transceiver

[5]The used counter, the SR 620 [SRS06], has a resolution of 4 ps and a typical precision of about 25 ps, which is sufficient to characterise the link delay between the PHYs.

stabilised environment ($\Delta T \leq \pm 0.3\,\mathrm{K}$) with dehumidification. Consequently, the influence of the physical parameters on the standard oscillators is avoided. Additionally, the power supply load was kept at almost constant load to hinder influences of the supply voltage on the oscillator frequency.

4.3.1 10 Mbit/s Ethernet (10 Base-T)

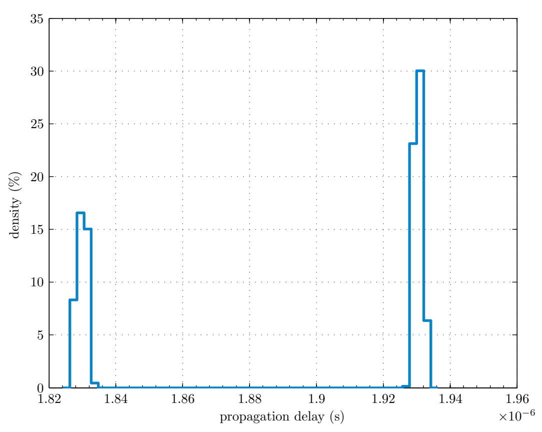
This bit rate version of Ethernet is characterised by the resynchronization of the receiver clock with every packet. The line in idle mode does not transport any information (and to be exact in this mode also no signal). Therefore, all clocks are free running if no data transmission is currently active. The preamble of each packet is used to get the receiver PLL in phase in order to sample the adjacent data of the packet at the right time, whereas the PHYs can either leave the receive clock running free or even turn it off during phases of no data reception.

According to the standard, there is no signal on the line when no data is transmitted. Furthermore, the specification (clause 12.3.2.4.4 of [IEE08a]) defines that the transmit entity switches to high level for at least two bit times to indicate *IDLE* state and then turns off the line driver. Thus, the physical layer devices on the receiving side have to resynchronize with every packet. This can cause the receiving PLL to lock to different edges in dependency of the packet rate, since the drift during the idle phase defines the offset the PLL has to compensate for. In case of a high packet rate, the drift between the line and the receive clock will give only a small offset and therefore cause the PLL always to lock to the same edge, since there is a continuous translation from the 10 Mbit/s domain to the MII. While the first uses Manchester encoding (clause 7.3.1.1 of [IEE08a]) at 20 Mbaud, the latter utilises 4 bit at 2.5 MHz. Therefore, the locking of the PLL cannot be automatically aligned to nibble boundaries. Consequently, the PHYs do buffering of bits to output always full nibbles, which also introduces a variable delay of 1–4 bit times is possible besides the half clock period delay due to the clock recovery from the Manchester encoding. Similar effects – described in the following sections – can also happen in the higher-speed modes.

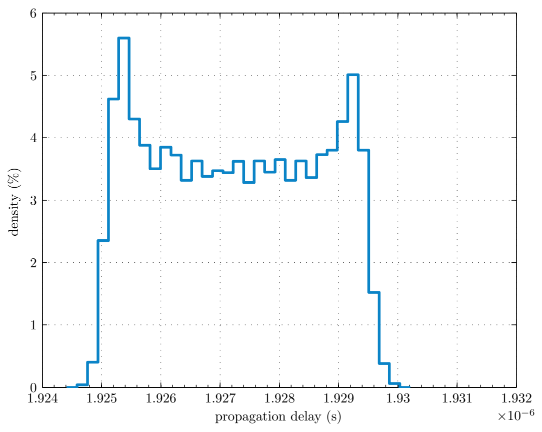
Another issue comes from the fact that the standard uses two different wire pairs of the twisted-pair cable for the receive and transmit signal. Manufacturing uncertainties lead to physically unequal lengths and effective characteristic wave impedance. This results in asymmetry of the packet transmission in addition to the PLL locking properties.

Figure 4.5 shows the measured behaviour of a 10 Base-T link. Since the PLL locking has to be done on each packet individually a separate measurement for different delays on link establishment is not necessary. The effect of the per-packet-locking can be seen in figure 4.5(a) where a low packet transmission rate was used. Consequently, the PHYs have to run their internal clock based on the local oscillator which drifts with respect to the remote side. On every new packet, the PLL then can lock on a new position that is independent of the previous state.

In the figure, the used PHY only shows two possible positions resulting from the 20 MBaud to 10 Mbit/s conversion. The line encoding is in the special case of this PHY obviously hidden by an internal buffer compensating for symbol alignment issues.



(a) 5 Hz packet rate



(b) Continuous data transmission

Figure 4.5: Delay behaviour of 10 Base-T Ethernet

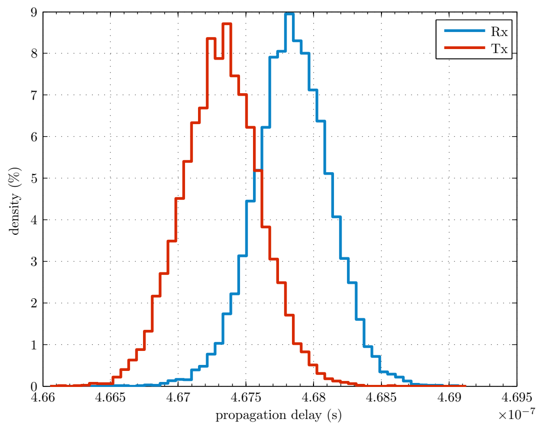
Contrary, figure 4.5(b) gives the delay for an almost continuous data transmission. The PLL of the PHY is obviously locked on the second edge of figure 4.5(a), but is able to maintain this state for the whole measurement. In other words, the right illustration is a detailed view of the second peak shown in the left figure. Both diagrams of figure 4.5 illustrate the behaviour for 10 000 samples of the line delay. Summarizing, this mode can deliver a quite good jitter performance of $\sigma = 1.387$ ns in case a continuous data transfer (not necessarily dedicated to clock synchronization) can be ensured.

4.3.2 100 Mbit/s Ethernet (100 Base-T)

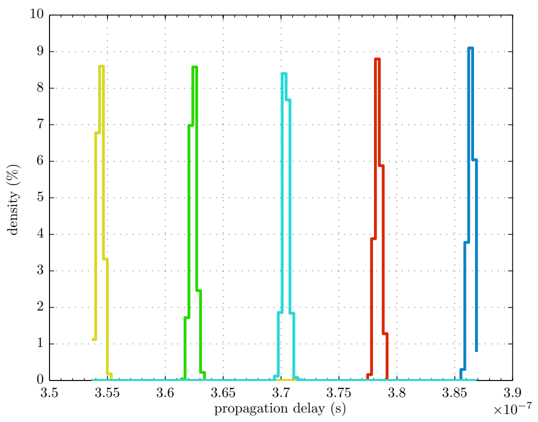
The introduction of the 100 Mbit/s mode in Ethernet brought several changes to the physical layer. Clock synchronization is effected by the usage of the 4B/5B line encoding and the *Idle* code-group (clause 24.2.2.1.2 of [IEE08a]). The coding replaces always four bit by five bit groups, which are coded in a way that long constant bit sequences are avoided to ease clock recovery. Additionally, it is possible to insert control codes, e. g. to denote the start of a transmission (/J/, /K/).

Another special code, the Idle code-group, denoted as /I/ in the specification is transferred between individual packets of the data stream. By establishing a continuous transfer on the media between the physical layer devices, it provides a continuous fill pattern to establish and maintain clock synchronization also in data idle phases, i. e. when there is no data available on the MII transmit path.

Regular transitions in the signal level on the physical media ensure that the PLL in the receiver is able to maintain a consistent and synchronous receiver clock output. Nevertheless, there is still an issue with the downsampling of the 125 MHz on the line to the 25 MHz on the MII [Ros06]. Most PHYs let the internal PLL lock on any edge of the incoming data stream on link establishment. This results in a variable delay of 1–5 bit times to the real symbol borders (a nibble in 4B/5B encoding) which is compensated by



(a) Delay history for both directions



(b) Behaviour for link reestablishment – the different colours indicate five individual measurement runs

Figure 4.6: Delay behaviour of 100 Base-TX Ethernet

small FIFO buffers. The delay is constant as long as the link is established, but adds to the asymmetry of the transmission line since the behaviour is independent on both sides of the connection. This hinders accurate packet delay measurements through round-trip delay evaluation used by most state-of-the-art clock synchronization protocols [DVRT08].

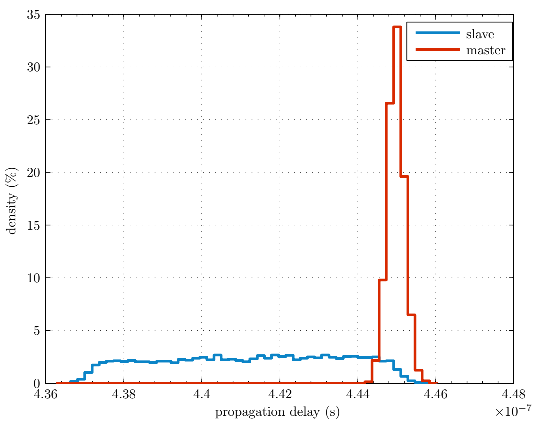
Figure 4.6 shows exemplary histograms for a direct 100 Base-T connection. Compared to the slower transfer rate the jitter of the link with continuous pattern transfer is significantly lower and shows a Gauss distribution (see figure 4.6(a)) with a low standard deviation of $\sigma = 0.286\,\text{ns}$. It indicates the typical jitter added by a PLL recovering the clock from the data signal. Due to the idle pattern, /I/, the mean value of the delay stays constant as long as the link is continuous up.

The mentioned locking behaviour of the required PLL is illustrated in figure 4.6(b)[6] for 1 000 samples on each locking position. While the standard deviation of the delay always corresponds to the afore stated value, the absolute delay changes each time the link is newly established. The mean value of the individual peaks differs exactly by 8 ns as it is expected. Thus, even the absolute delay stays always in the range of 0-32 ns, which can be enhanced by the usage of special PHYs as described in chapter 3.4.2.

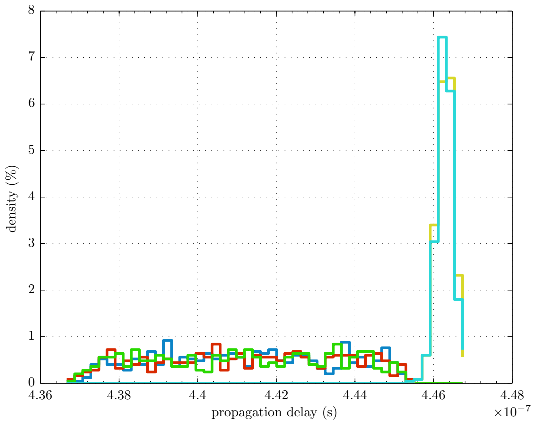
4.3.3 1 Gbit/s Ethernet (1000 Base-T)

The high data transfer requirements for 1 Gbit/s made it necessary to use all four wire pairs of standardised category 5 twisted-pair cables [IEC08] in order to keep bandwidth requirements for each pair as low as possible. Consequently, to use existing cabling, both directions are encoded on the same physical wires using echo and Near-End Cross Talk (NEXT) cancellation. This requires a synchronous approach on the media, resulting in a master/slave structure already on the physical layer. The master device (selected

[6]Since the internal structure of the Marvell 88E1111 PHY masks three of the five possible locking positions, for this measurement a SMSC LAN8700 [SMS09] PHY was used.



(a) Delay history for master/slave PHY



(b) Behaviour for link reestablishment – the different colours indicate five individual measurement runs

Figure 4.7: Delay behaviour of 1000 Base-T Ethernet

through auto negotiation by the PHY) defines the used clock on the line and the slave uses the same for its own transmission by clock recovery from the line. A detailed description of the data transfer techniques can be found in clause 40.1.3 of [IEE08a].

The receive path of the GMII is specified to deliver a clock which is either derived from the received data or it may be sourced from a nominal clock (clause 35.2.2.2 of [IEE08a]). The latter ensures that if the PHY looses the recovered clock reference it still can provide an appropriate clock signal. Further, it is explicitly stated that there is no need for a transition between the recovered clock reference and the nominal one on a frame-by-frame basis.

Due to the possible usage of a local oscillator on the receiver side, the PHY has to introduce a clock transition in order to compensate for the non-aligned phase of the incoming data stream and the interface clock. Further, a FIFO buffer is required to compensate for frequency drift of the incoming 125 MHz clock with respect to the local one, which can be in the range of ±100 ppm. This results in an 8 ns equal distributed delay on slave side, while the sender experiences a much lower, normal distributed jitter caused by the clock recovery PLL.

Figure 4.7(a) shows the delay behaviour for a Gigabit Ethernet link. The histogram was calculated from 10 000 samples for each direction. It can be clearly seen that the slave device shows the mentioned 8 ns equal distribution, which is supported by the fact that $\sigma_{\text{slave}} = 2.288\,\text{ns}$ is very close to the theoretical value of

$$\sigma_{\text{calc}} = \sqrt{\frac{T_{\text{clock}}^2}{12}} = 2.309\,\text{ns}. \tag{4.7}$$

Contrary, the master side shows the typical behaviour of a recovered clock, which results in a standard deviation one magnitude lower than the one of the slave, $\sigma_{\text{master}} = 216\,\text{ps}$.

An interesting behaviour of the 1000 Base-T link can be observed in figure 4.7(b). It shows five measurements of the delay seen from the slave device of figure 4.7(a) with a reset of the link in-between. The delay distribution stays exactly the same, only sometimes the role on the link (i. e., master or slave) is switched between the slaves. Since the transmission uses all wire pairs for both direction and the GMII clock equals the symbol rate on the line, the absolute delay does not change. This is a significant benefit over the previously mentioned standards, although the achievable standard deviation is not the peak level.

As introduced already by the 100 Mbit/s standard of Ethernet, the 1 Gbit/s mode also makes use of *IDLE* code-groups, i. e. /I/. They provide a continuous fill pattern in case the GMII is idle in order to establish and maintain clock synchronization. Further, they are also used to separate distinct carrier events as mentioned in clause 36.2.4.12.

4.4 Prerequisites for Hardware Support

The introduction of hardware supported clock synchronization in standard Ethernet equipment based on IEEE 802.3 is influenced by the availability of appropriate signals to be measured. As denoted in the previous chapter, there are various possibilities in the layered approach of a network system where the required timestamping can be done. Due to the reasons outlined in chapter 4.1 an approach as close as possible to the physical medium is preferred. The gained timestamps have to be transported through the network layers to the application. This is an important issue and therefore tackled together with clocking issues of the synchronization system in the following.

4.4.1 Media Dependency and Event Access

The Ethernet specification [IEE08a] defines for 100 Base-T and above the (Gigabit) Media Independent Interface as the interface closest to the medium which is physically available. Therefore, if COTS chipsets and no dedicated implementations are used it is the optimal access point. Consequently, state-of-the-art network card implementations use physical layer devices, which are attached via (R)(G)MII to the MAC. The signals of the interface are therefore accessible to a timestamper (detection of SFD pattern on clock synchronization packets), which can be implemented in addition to the standard signal flow in the form of a monitoring-only device. As denoted in the previous chapter, the performance of this solution is heavily dependent on the exact interface implementation, the physical layer device design, and the Ethernet transmission standard used. Nevertheless, implementations using this method can reach timestamping accuracies which are significantly lower than the achievable values in software.

For high-accuracy clock synchronization it is important that the timestamping event gathered from the accessible signals temporally matches the arrival or leaving event of a packet with close relation. This implies that the related signals have a fixed phase relation to the transmission signal on the medium. Otherwise, only statistical methods requiring a significant number of events can be used to calculate the most probable time of occurrence.

Cost reductions and size issues lead to highly integrated, combined PHY/MAC solutions. In that case, the required physical signals for a timestamper are no longer available on the MII level. Consequently, the required hardware support has to be designed as function blocks into the ASIC. In case of high integration, the assumption that the physical layer should be exchangeable – the main idea behind the layered approach – does not hold any more. Consequently, also solutions taking benefit of additional information from the PHY for higher accuracy, e. g. link length, signal quality, and PLL locking, are conceivably.

4.4.2 Timestamp Transportation Issues

Monitoring the (G)MII actually introduces a bypass for the timestamps to the MAC and is only useful, if existing function blocks or chips should be used. In any other case, the functionality of detecting the start of a packet is already available in the data link layer and can therefore be reused to draw a timestamp. An existing MAC implementation then needs to be expanded by a clock holding the timebase of the node. The close interaction between the standardised MAC functionality and the synchronization operations requires the implementation of a unified block, resulting in development effort. Alternatively, the bypass solution only requires the design of a block necessary for clock synchronization and attaching it to the digital MII signals.

Unfortunately, a bypass solution (see figure 4.8) is not feasible for timestamping on the physical layer. A second decoding unit in parallel to the PHY has to be designed in a way that the analogue parameters (impedance, isolation, etc.) of the original physical medium attachment are kept. This and the quite complex signal decoding blocks within a PHY that are required to decode the packet, would almost double the hardware effort on this layer making the solution very unattractive. Therefore, PHY integrated synchronization functions are far more efficient. Another way of solving the issues of layer 1 is to make use of PHYs, which are able to run the interface to the MAC using clocks that are deterministic and phase stable to the signal on the medium. Additionally, data must not experience variable buffering as it is the case for RMII.

Out-of-Band Solution and Timestamp Association

One issue occurring from bypass solutions is the problem of matching timestamps to the data packet they were taken for. Since the timestamps are drawn by a unit not integrated into the data path, they also have to be transferred via a separate channel and cannot be transferred in-band with the data. As most protocols and controllers for synchronizing clocks are implemented at higher layers of the network stack, the timestamp has to be matched with the specific packet it was drawn for. Due to the fact that each intermediate layer can drop (or reorder) frames it is not guaranteed that every timestamp read by software has an associated packet. The issue is illustrated in figure 4.8 for four packets whereof one is discarded.

Due to the separated data and timestamp flow, identification tags (e. g., hash value, CRC, sequence number, or address information) has to be stored together with the timestamp in order to verify the correct link between timestamp and frame at the layer

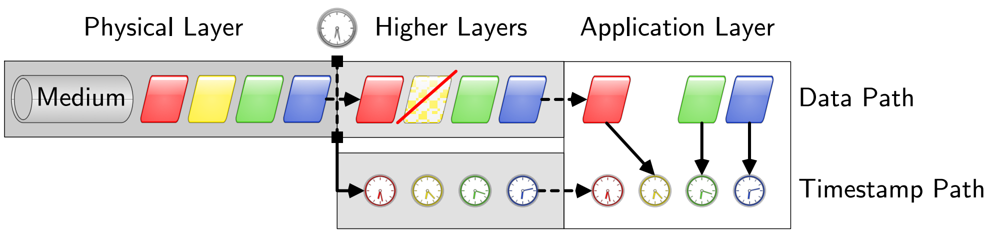


Figure 4.8: Four packets are timestamped after the physical layer using a bypass solution. As illustrated the problem of matching the timestamps to the packet can occur at the application layer due to packet re-ordering or discarding.

running the clock synchronization engine. In-band transportation of timestamps, on the other hand, demands certain prerequisites on the existing frame structure and/or hardware implementation.

In-Band Solution and Data Modification

The easiest way is obviously the replacement of timestamps with their exact value at the foreseen location in synchronization messages by hardware modules. Besides the implications on the encapsulating layered architecture as mentioned in chapter 1.4.2, also the frame must be able to transport all required timestamps. Unlike NTP, which can hold all four required timestamps for a round-trip measurement in one frame header, other protocols, e. g. PTP, just foresee the necessary space for the type of the message, i. e. egress timestamp for Sync messages. The latter then has the problem that on the receiving node, the ingress timestamp cannot be stored in a reserved space of the frame to be transported to the application layer.

Besides the simple space issues, the hardware implementation also has to match the software protocol expectations concerning time format and the used position with in the frame. The hardware processing unit therefore has to be compatible with the used protocol version which, for high volume implementations, can be an issue in case of a modification of the protocol rendering the hardware useless.

Another issue that makes direct inserting of timestamps into the frame by hardware units difficult, is the effort that has to be taken to determine the correct position of the fields to be modified. Since the timestamping is done on a layer close to the medium, a number of higher layers services and their respective data encapsulation might change the position of the timestamp field. Examples for such issues are any sort of tunnelling protocols, dynamic headers (IP version 6) or even encryption hindering timestamping completely. Since there is no universal solution to this problem, either the assumption of a fixed position must hold or the hardware unit must be able to interpret all necessary header information to determine the correct position in a frame. The latter will always be limited to a certain number of possibilities since it violates the idea of the layered network stack where each layer's function is independent of the others.

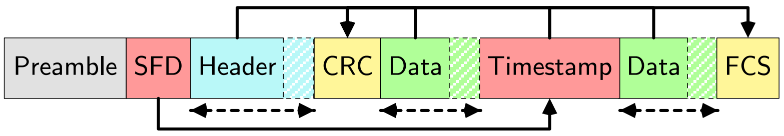


Figure 4.9: Interdependencies of various fields in a network frame including checksums (CRC, FCS) and variable size fields causing problems when sequentially processed in hardware.

The insertion of a timestamp normally also involves the recalculation of CRC values. After the MAC strict timing constraints apply in order to have a deterministic access to the medium. As an accurate timestamp can only be drawn after the access to the medium has been granted, the processing time for recalculation of timestamps is limited. This also might hinder direct insertion besides the fact that the position of all CRCs and their algorithm must be known by the hardware unit. Checksums that precede their data, e. g. the UDP checksum, also hinder hardware modification of the frame. This is due to the fact that data is sent to the medium in a sequential way and the checksum at the beginning cannot be modified after the necessary data (whole frame) is known since it has already been transmitted. UDP allows to have the checksum optional by setting it to zero [Pos80] in order to solve this issue. Unfortunately, the disabling of the checksum is not allowed for every protocol, like e. g. the widely used TCP, which also has the checksum for the whole frame in the header. Figure 4.9 shows the principle temporal problem and the related interdependencies in a frame containing a timestamp used by an application layer protocol that should be processed at layer 1 or 2 by hardware.

As indicated in a previous example, the problem of missing reserved fields for in-band delivery of timestamps occurs on the ingress path, where at the end the reference timestamp has to be compared to the local instance. Instead of modifying the frame data or using a bypass solution, there is also the option of associating the frame data with additional descriptive data structures to transport information between the network layers. There are several examples for this type of communication which were introduced due to the strict limitations of a layered network architecture. Examples are the Management Data Input/Output (MDIO) interface between physical and MAC layer allowing to exchange information about the, e. g., link status and the MAC buffer descriptors indicating FIFO overruns in hardware, wrong line symbols, frame length errors, or collisions to the network layer.

Similar communication paths usually also exist between the higher layers realized in software. The Linux kernel for example stores network frames in *socket buffers* [CRKH05], which are data structures holding the frame data and additional information, like whether the FCS was already checked or the receive timestamp. Thus, if the timestamping unit is integrated with the MAC transferring the data to the software layers and the timestamp format of the associated data structures is sufficient, there is an additional way. Further, this method does not suffer from the problem of reordered or discarded frames, which is later on discussed in section 6.2.1.

4.4.3 System Clock Structure

Another important issue for high-precision clock synchronization is the usage of a single clock source for timestamping, control, and the application. This prerequisite is relatively easy to fulfil for an embedded system driving all components from a single oscillator, since there is a constant phase relationship between all used clock rates. On the other hand, common architectures like Personal Computer (PC) based systems use a variety of oscillators. This becomes especially important if existing systems should be enhanced by a special network interface using a stabilised oscillator for keeping the synchronized time. In this case, the accurate time information is only available within the clock domain of the oscillator, i. e. usually the NIC itself. While such an add-on card can easily provide accurate output signals for further usage in external applications using hardware signals, it is not trivial to use the timebase in applications running on the main processing unit.

In case the timebase should be used on the application level on a standard PC architecture as well, e. g. ordering of distributed file system operations, the issue of multiple oscillators in a single system applies. Consequently, the access from the software to the timebase must be possible with low delay and jitter (compared to the desired accuracy) or a second synchronization method between the accurate and the system timebase has to be implemented. The transfer of the time from the hardware timebase into the software then again suffers from the problem of inaccuracies due to scheduling and other jitter sources mentioned before. Usually OSs keep the system time in software by incrementing a counter on every interrupt signal from a timer. One solution for the issue is to switch this timer signal to the accurate hardware base. This is basically an engineering task, which is dependent on the effort to modify the OS kernel and the used timekeeping hardware.

5 Evaluation Architecture for High-Performance Synchronization

All necessary functions required to maintain and use a highly accurate timebase are referred to as Clock Synchronization Cell (CSC). These include the clock itself, control and monitoring registers, a timestamping unit for network packets (see section 5.4), and function blocks to generate and timestamp external signals. The cell extends existing or new network interfaces in order to be used with hardware supported clock synchronization methods. The function block itself does not imply the use of a specific application layer synchronization protocol, but needs to be adequately adapted to the transmission standard. This thesis focuses on Ethernet-based connections. The according IEEE standard [IEE08a] therefore clearly defines boundary conditions for the implementation of such a cell.

The present chapter gives an overview of the system requirements that have to be applied and how to design the related function blocks in hardware. Besides the general functionality required for any network-based synchronization system also exemplary application specific adaptations are explained. Further, the unit drawing the essential timestamps, the MIIS, is described. The accuracy of the event sampling by this unit defines the maximum reachable performance for the overall system. Thus, different methods of acquiring exact time information are discussed. The unit can be implemented purely passive, only drawing timestamps, or also active (replacing software timestamps with their more accurate hardware equivalent). While this section will show the implementation issues, the implications of this replacement are described in chapter 4.4.

5.1 Applying System Requirements

In order to make dedicated hardware support on Ethernet-based synchronization systems a useful add-on, the achievable performance has to beat the purely software based approaches. Commercial-off-the-shelf based real-time networks can achieve synchronization accuracies down to 1 µs purely relying on standardised hardware without a special unit for gathering precise timestamps. Consequently, the aimed goal for hardware supported timestamping lies at least two to three orders of magnitude lower, targeting the nanosecond. Therefore, the CSC in principle should be capable of maintaining a timebase with at least 1 ns accuracy over one synchronization period. The latter is assumed to be in the single-digit second range, which fits the characteristics of common oscillators best. Additionally, the rigid accuracy requirements also demand the compensation of small environmental changes that can cause relevant variations within seconds. As demonstrated later on in chapter 7, even accuracies below are achievable with appropriate measures.

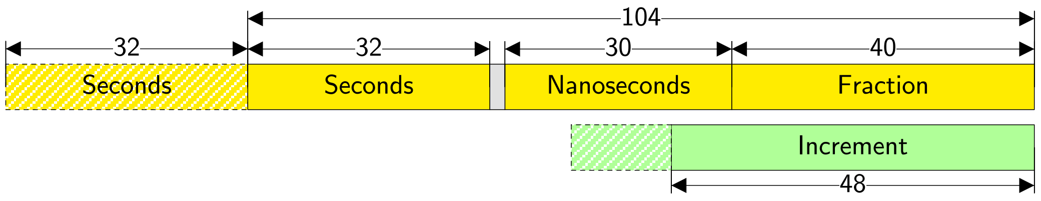


Figure 5.1: Time representation bit widths for adder-based clock design

5.1.1 Clock Resolution

For highly accurate clock synchronization, the core has to be able to compensate drift rates in the range of the desired precision over one synchronization interval, T_{sI}. For a digital adder-based clock (see figure 1.5), the precision Π, using an oscillator with the frequency f_{CSC} requires a minimum step size (the value of the Least Significant Bit (LSB) of the increment)

$$\Delta = \frac{\Pi}{f_{\mathrm{CSC}} \cdot T_{\mathrm{sI}}}. \tag{5.1}$$

For a typical synchronization interval of 1 s, an assumed maximum achievable frequency of 1 GHz, and a desired precision of ± 0.5 ps for the minimum step size evaluates to $\Delta = 10^{-21}$ s. This requires the adder-based clock $\mathcal{C}$ to maintain a register with a part for fractional seconds of 70 bit. Depending on the targeted protocol, the representation format of the time can also be changed to 30 bit nanoseconds and 40 bit fractional nanoseconds for more easy usage in hardware units for on-the-fly processing of timestamps in frames.

While the LSB directly influences the achievable accuracy, the width of the increment of the CSC defines the lowest oscillator frequency and the possible drift compensation range the cell can be used with. If an 8 bit value for the nanosecond increment is used, the cell can still run with 4.348 MHz $\pm$ 10 %. Since the (G)MII interface runs at 125 MHz, 25 MHz respectively, multiples of that frequency are useful for implementations, because the transmit path can use the same timebase as the synchronization cell allowing for perfectly precise timestamps on that path. The clock speed can then be changed in the range of [6...1/6] times the normal rate allowing for fast compensation of offsets. Wider increments are of course possible but only necessary for lower oscillator frequencies, whereas it has to be noted that wider adders also increase the complexity, decreasing the achievable clock rate for a specific architecture. Nevertheless, state-of-the-art chip technologies or Field Programmable Gate Array (FPGA) architectures should be easily capable of fulfilling the above mentioned frequency requirements.

Figure 5.1 summarises the register width requirements for an adder-based clock design. The second register width can be chosen in dependence of the utilised protocol, e. g. 32 bit for NTP till version 3 or PTP version 1, 64 bit for NTP version 4 or PTP version 2, or any arbitrary value if two-step clock implementations are used that can extend the timestamp to its full value in software. The two-step clocks are therefore more flexible when it comes to different timestamp formats, but require additional Follow_Up packets.

For protocols, such as e. g. NTP, not supporting two-step clocks, the timestamp format has to exactly match the format of the transmitted frames.

5.1.2 Performance

In order to support common triple speed Ethernet implementations, i. e. 10 Base-T, 100 Base-T, 1000 Base-T, the clock synchronization cell must be able to support timestamping of all events generated by the ingress and egress path of the node. As these events are synchronous to the packet transmission on the line, the maximum packet rate on the line defines the upper boundary of events the MII timestamper and the synchronization cell have to handle. Since the timestamping unit itself runs synchronous to the clock of the MII in order to sample data correctly, the unit inherently is capable of handling the packet rate.

On the other hand, the architecture of the synchronization cell directly influences the required performance parameters. The concerned functionality is the timestamp transfer to the upper layers. In case an integrated solution is chosen, there is again no problem, since only the data structures for keeping the extended information about the frame have to be expanded (see chapter 4.4). If a by-pass solution is chosen, the critical issue is the required buffer between the MII scanner and the application layer in the cell. The selected transmission standard influences the maximum packet rate, which has to be considered. The latter can be calculated by taking the minimum packet length, preamble (7 Byte), SFD (1 Byte), header (14 Byte), the data field (64 Byte), FCS (4 Byte) equals 72 Byte and the minimum Inter-Packet Gap (IPG) (12 Byte) into account. Depending on the utilised physical layer speed the packet rate therefore can be up to 14 880 pps, 148 800 pps, or 1 488 000 pps (packets per second), respectively. Further details are discussed in the technical report [KP02].

Although typical synchronization protocols do not occupy high bandwidths (one to two packet(s) per synchronization interval) – on the contrary, they are designed to be bandwidth efficient – there are several applications that require back-to-back packet timestamping. The obvious one is that for debugging, logging, or measurement purposes not only the packets of a specific synchronization protocol are required to be timestamped, but all other packets as well.

Another use case are democratic protocols, where a group of nodes exchanges the clock values among each other. This causes packet bursts, which have to be timestamped in order to correctly perform democratic algorithms. Such bursts require the cell to process (generate and store) timestamps within the slotTime (minimum packet-to-packet time). The time is again defined by the minimum packet size plus the IPG, which makes a total of 672 bit times, i. e. 672 ns for 1000 Base-T. The core has therefore to provide a buffer for timestamps capable of holding enough entries to allow the application to read them out after an interrupt. Due to interrupt latencies in the order of microseconds, it is advisable to choose the buffer size in the order of the number of timestamps generated by one burst of packets from the network. As it is likely that the packets arrive within a very short time (possibly back-to-back) the core should be able to process the timestamps in the minimum time mentioned before.

5.1.3 Software / Hardware Partitioning

One important issue to consider when developing hardware support for high-accuracy clock synchronization is the partitioning into hardware and software function blocks. The latter usually give more flexibility but the hardware has clear advantages, when performance matters.

When it comes to separating functions into software and hardware implementation, several considerations have to be done. Software in general should be designed to run time independent, only event triggered. This gives the advantage that it can be used on different hardware platforms without the necessity for modifications. Hardware platform independence allows software to run on hardware optimised for different purposes. Examples for this would be energy optimised or mobile devices. Without special measures (e. g., a RT-OS, defined command execution times on micro-controllers, no multi-tasking/preemption) software is not capable of keeping exact time constraints. Consequently, all high precision related issues are better realised in hardware blocks.

The basis for the above mentioned recommendation is again the negative criterion for distributed systems (explained in chapter 1.1). The transmission delay between two blocks compared to the occurrence of relevant events has to be negligibly small. For function blocks this is easily achievable by using synchronous hardware, which uses a phase compensated clock signal for all registers. Data signal delays are then by design smaller than the period of the clock signal to ensure correct functionality. Throughout the design, multiple registers can introduce delays in multiples of the clock period T_{clock}. Nevertheless, this delay is known in advance and does not increase the jitter due to the common clock. The maximum achievable precision is therefore defined by the timing properties of the sampling register(s). This includes the clock signal quality (jitter) and the setup and hold times for a single register. The remaining uncertainty is then given by three factors: the clock period, the clock jitter, and register setup/hold times. Whereas the first can be enhanced through several technologies for event detection as described in chapter 5.4, the last two depend on the used technology.

Since Ethernet-based elements for high-accuracy clock synchronization are in common only add-ons to the actual computing platform, also software function blocks on general purpose processors are used. The actual event detection and synchronization is seen as a service for already existing applications. This means that the specially designed hardware has to communicate via an interface (e. g., PCI) with the remaining components. All access from the central computing components – which includes software running there – experiences delays and jitter through clock transitions and buffering in command queues. Within a common x86 based PC a number of counters and clock sources exist that can be used to keep the time. Neither of them is eligible for high-accuracy (range of 1 ns) synchronization [BRV09]. The problem here still is the latency and jitter during event detection (ingress or egress packet), which cannot be accurately measured when being sent/received to/from the medium. This is due to the fact that the access to the medium is controlled by the MAC hardware block but the time source is kept in a separate device without direct (hardware signal) access to it. The path through software (timestamping within the IRQ handling function) suffers from latency and jitter due to task switching.

Consequently, accuracies below one microsecond are out of range.

In order to overcome the issue of transferring the timebase from the timestamping hardware to the software domain, the following solutions are applicable:

- The obviously simplest method is to synchronize the software timebase to the high-accuracy hardware. Since the operating system timebase is generated from a platform specific timer chip, this basically implements a software based algorithm for transferring the time from an extension hardware to the standard timebase of the selected platform. The issue with this method is that the timestamps for the software timebase lack the accuracy of the special hardware. The transfer therefore can never be as accurate as if the dedicated hardware is directly used. Further, cascading of control loops can impose stability problems.

- The converse technique is to keep the timebase in hardware. The advantage here is that everything is directly connected to the special hardware and incoming and outgoing events can be derived from the same clock driven by a single oscillator. The application has to be known in advance in order to design the hardware accordingly. Examples are trigger devices for LXI-based test and measurement devices. Although the application itself can directly benefit from the highly accurate timebase, software itself needs to access the clock via hardware calls and therefore again suffers from jitter and delay issues. This means that e. g. software run times cannot be measured accurately, because the access to the hardware device reduces the precision.

- In common operating systems the time is kept in a software counter and is updated by regular hardware interrupts. Additional high-resolution timers are used to increase the resolution of software generated timestamps [GN06]. By using an interrupt from the dedicated hardware support in the Ethernet network card, it is possible to use a synchronous time source for hardware timestamping as well as for the software timebase. Due to the low-level integration into the operating system, the access time to the hardware can be kept quite precise. Still, at least two different oscillators are involved: one from the timekeeping hardware and another for the processing platform for the operating system/software. The main advantage is that both are coupled very close and the hardware timestamps have a fixed offset to the software timebase defined by the interrupt latency. This offset can be compensated if the overall platform design is known.

- The highest accuracy can be achieved, if only one common (stable) oscillator is utilised for the dedicated hardware support on the network interface and the software processing. If only PLLs with a fixed natural number as multiplicator are used within a(n) (embedded) system the fixed phase relation between all components allow to calculate the constant delay from one block to the next. The jitter in this case introduced by the transfer of the time is as low as the performance of the PLLs. The latter can be as low as some picoseconds RMS even in microprocessors [Boe99].

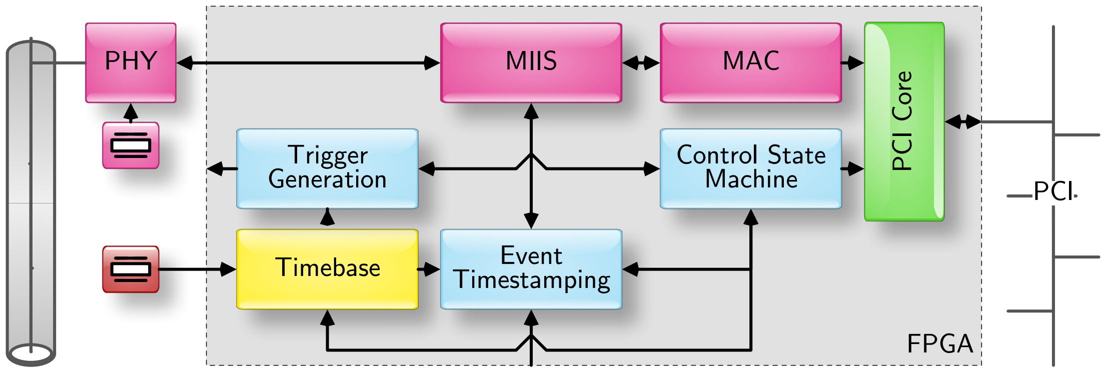


Figure 5.2: Basic prototype FPGA implementation

Due to the common oscillator, which makes the whole system synchronous, the latter solution gives the best synchronization accuracy. Nevertheless, the other solutions can be justified by either extending existing hardware without the option for modifications or for simple cost reasons. Using add-on hardware to COTS hardware platforms usually gives better universality and cost/performance rating than completely dedicated hardware platforms design only for the purpose of clock synchronization.

5.2 Basic Prototype and Proof of Concept

In order to prove the applicability of the mentioned methods and for the purpose of performance investigation an evaluation platform according to figure 5.2 was developed. The block diagram shows the main functionalities required to build a synchronization system with dedicated hardware support over Ethernet. The design was chosen according to the criteria of a low number of parts, reuse of existing function blocks, and flexibility. The latter as well as the reuse clearly votes for an FPGA-based implementation, since existing code for an Ethernet MAC or the PCI core can be utilised. Furthermore, FPGAs allow fast exchange of function blocks in order to evaluate several designs for the CSC consisting of the timebase, the event timestamping, the trigger generation, and the control state machine. The MIIS is directly connected to the event timestamping allowing to generate timestamps for ingress and egress packets on the Ethernet communication path.

The FPGA-based implementation also allows to minimise the number of chips on the PCB. Besides the FPGA itself, the board requires components for power supply, bus drivers for the communication via PCI, the oscillators, and the PHY. While the figure shows two oscillators, one (stabilised) device for the FPGA internal function blocks – especially the timebase – and the other for the PHY, the clock connection between PHY and FPGA also allows to run both from the same clock source. This gives on the sending path the advantage that one clock transition is avoided due to the synchronicity of the transmission and the timebase. The chosen architecture not only gives maximum flexibility in the function block design, but also enables the use of the card in all systems equipped with a PCI bus. The card can therefore enhance a wide range of platforms

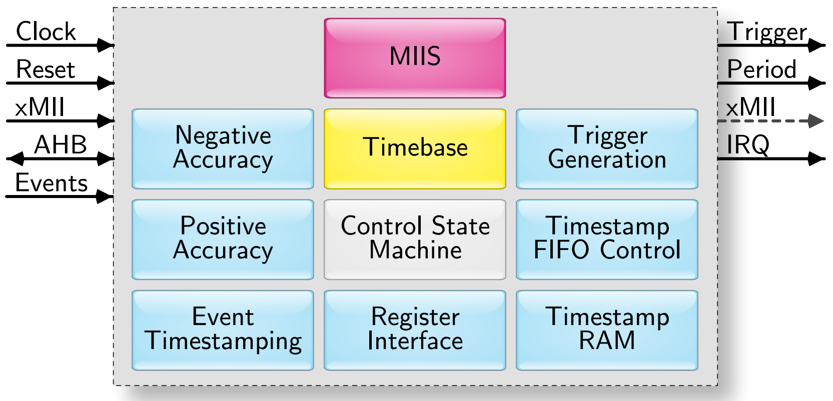


Figure 5.3: CSC internal block diagram

with an Ethernet interface including hardware support for clock synchronization.

The mentioned flexibility is also shown by the fact that the first three out of the four possible methods to use the precise timebase in software, listed in chapter 5.1.3 are supported. Synchronization using a further algorithm in software is described in 6.2.2. Since the FPGA can be be programmed with function blocks adapted to the targeted application, keeping the time fully in hardware is also possible. An example is the generation of e. g., a synchronous frame clock for transmission lines, as used in the telecom industry. The card shows up as a standard PCI network device and with special software driver support it is also possible to use the timebase directly as the source for the system time. This method – mentioned as item three in the referenced list – then avoids a second synchronization step by directly deriving the software clock from the hardware.

Unfortunately, common PCI-based system architectures do not use the same oscillator for the bus and the processing units. Consequently, the use of the PCI clock for the timebase does not allow to build up a fully synchronous node. The latter can be achieved in an embedded system where the bus (not necessarily PCI) shares a common clock with all other devices. A fixed phase relationship between the dedicated Ethernet hardware support and all other components can be assumed, which enables the software to calculate with fixed (jitter free) delays. Nevertheless, such a system does not provide the flexibility of an extendable architecture and cannot be used in a universal manner. This gives the reason why the technique of special node design is excluded from the considerations in this thesis.

The main element of a dedicated hardware support for Ethernet is the CSC, which consists itself of several function blocks shown in figure 5.3. All blocks except the MIIS are integrated parts of the synchronization cell. From a systematic point of view, the scanner on the MII could also be integrated into the MAC. Therefore and due to the task of separating the clock domains (transmit, receive, and CSC clock) it can be seen as an extra module. The functionality also has to match the interface it is attached to. The

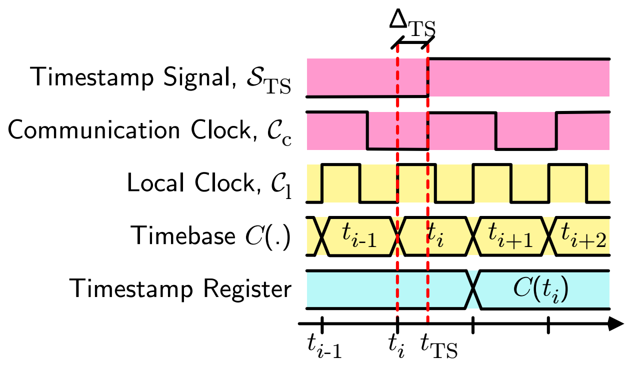


Figure 5.4: Signals involved to generate a timestamp in the CSC

GMII interface uses a different data bus width and an additional clock signal compared to the standard MII for 10/100 BASE-T. While the MIIS has to match the requirements of the transmission standard on the medium, an interface to the synchronization cell can be defined in order to keep the remaining blocks of the CSC universal. Chapter 5.4 deals with the specific requirements and solutions for precise event detection with high resolution in this block.

Actually, this function block has the task to generate event signals for ingress and egress packets in the most simple case as here for the prototype. The signals on the MII are sampled with the clock of the CSC to detect packets and store timestamps as shown in figure 5.4. The timestamp signal is synchronous to the communication clock (receiver or transmit) on the xMII. If a high level of this signal is detected at the rising edge of the CSC local clock, the current value of the timebase is frozen in the timestamp register. Due to the granularity of the clock, a temporal error, Δ_{TS}, occurs, which is equally distributed over the local clock period.

Additional functionality, like on-the-fly timestamp insertion into the packet, requires the unit to modify the data stream of the MII which is indicated by the optional outgoing MII on the right side of figure 5.3. The insertion of the timestamp requires the MIIS to insert the current time which then demands close integration with the timebase of the CSC. If this operation is desired, the scanner is further required to be compatible to the synchronization protocol used in order to be able to modify the timestamp at the correct position in the frame. Additionally, the timebase has to provide the timestamps in the format required by the protocol in use. If the time is not available in the required format it must be possible to do the required format conversion in the timespan from the triggering point (SFD) to the start of the time field in the frame. The absolute time depends on the position of the timestamp in the packet of the synchronization protocol, but has a lower limit defined by the Ethernet frame format. The minimum time is defined by the Ethernet frame header consisting of two times six octets for source and destination address plus two octets for the type/length field. The worst case for the conversion of the time can therefore be given as $\Delta t_{\mathrm{conv}} = 12 \cdot T_{\mathrm{xMII}}$, i. e. 96 ns for the GMII.

Address	Field Name	Higher Double Word	Lower Double Word
0x0	Identification	seqNR[15:0] & srcMAC[47:32]	srcMAC[31:00]
0x1	Timestamp	0x0	seconds
0x2	Timestamp	nanoseconds	fractional nanoseconds
0x3	Upper Accuracy	32 bit	9 digits
0x4	Lower Accuracy	32 bit	9 digits

Table 5.1: Timestamp FIFO entry format

5.2.1 Timestamp FIFO

For nodes or protocols that do not support one-step clocks (on-the-fly insertion of timestamps into the frame), the CSC has to store the egress timestamp for later transmission by the upper layer protocol. For the ingress path the criterion is the availability of a method to transfer the timestamp attached to the frame. Either an upper layer protocol reserves a field in the frame for this purpose, e. g. NTP, or all network layers till the one processing the synchronization protocol support the transfer of attached timestamps with sufficient resolution. If such a method is not available, also the timestamps of the ingress path have to be stored for later use by the synchronization protocol. Due to the delay between packet transmission/reception and the read-out by the synchronization protocol sufficient buffering of the drawn timestamps is required.

The timestamping FIFO has to fulfil the performance requirements mentioned in chapter 5.1.2, i. e. about 1.5 million entries per second and direction for Gigabit Ethernet. The prototype implementation uses an entry format with 64 bit word width as listed in table 5.1, which gives five FIFO lines per timestamp. It is accessible via two 32 bit virtual registers. Accessing the lower double word increments the FIFO read counter and returns the next entry, while reading the upper double word leaves the counter unmodified. The data alignment allows to optimise internal memory access delays since only one address change is necessary per long word access. This also allows to reduce the number of access in order to remove undesired entries from the FIFO by just reading the register for the lower double word. The timestamp FIFO is implemented to work with any asynchronous dual-port Random Access Memory (RAM) block in order to provide good vendor independency. The two separate blocks of figure 5.3 indicate the interchangeability of the RAM from the FIFO control. The FIFO control state machine uses a single block to store ingress and egress timestamps. The entries are distinguished by the highest memory address bit to make them separately accessible for the software. For one Ethernet interface, the memory has to be able to store 10 lines (ingress and egress) within the slotTime (minimum packet-to-packet time) in order to cover the worst case.

The FIFO control state machine first stores a long word containing a 16 bit frame sequence number and the source MAC address in order to be able to perform a timestamp-to-packet matching procedure on the application layer. This is required because packets can get dropped or resorted by intermediate network layers. After the identification field

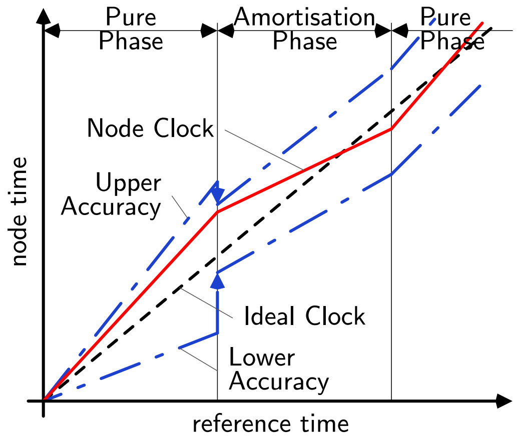


Figure 5.5: Sample diagram of the accuracy counter functionality [Gad08]

the entry covers the timestamp issued by the timestamp signal of MIIS. Since the storage procedure for one entry requires the extracted identification information from the packet and due to the format and access time of the memory, the timestamp and the status of the accuracy counters (see chapter 5.2.2) has to be buffered till the complete entry could be written to the FIFO. For that purpose, the timestamp signal of MIIS issues the freeze of the current value of all counters (timebase and accuracies) in separate registers within a single clock cycle. The state machine can then access the correct value later on. If the performance requirements are met by the state machine the timestamps are stored into the RAM before the next event can happen.

5.2.2 Accuracy Counters and Amortisation

This function block contains two additional adder-based counters, which can be used to define an upper and a lower boundary of the local clock accuracy. This is required for algorithms like [MO83], which are essential for fault tolerant solutions, e. g. the application case of [Gad08]. The accuracy counters run synchronous to the timebase but with different configurable increments. Events, which save the current value of the timebase (transfer of packets, external signals) also synchronously freeze the value of the accuracy counters for event triggered evaluation.

Figure 5.5 illustrates the usage of the accuracy counters in combination with the support for amortisation-based clock synchronization. For this functionality, the general control state machine manages two phases, *pure* and *amortisation*. Both have their individual step sizes for all internal counters (timebase and accuracies). Additionally, also the periodic timers can be programmed with separate periods for both phases in order to compensate for a changed step size of the timebase. Thus, it is possible to provide a constant output frequency even if the CSC compensates for an offset by increasing or reducing the step size from the nominal rate.

At the beginning of each amortisation phase an algorithm as [MO83] can define a new positive and negative accuracy for the node based on other remote clock readings. Due to the convergence of the algorithm the new interval is always smaller than the current one. To account for the new values, the cell allows to correct the accuracy counters by a fixed step at the beginning of the amortisation phase. A detailed description of the method is explained in [Gad08]. Each amortisation phase runs for a pre-defined interval and can be started manually, at a set time value, or the next expiration of a periodic trigger. In case the amortisation time is chosen to be a multiple of the period, the latter allows to synchronously change the step size while each period of the periodic triggers can be kept precisely the same. This feature can be used, e. g. in telecom installations, which require a stable output frequency. Additionally, the node can keep a continuous timebase and still compensate for large offsets in reasonable time.

5.2.3 Event Timestamp Registers

The prototype implementation also features a configurable number of event timestamp registers. The registers can be individually configured to freeze the current time at a configurable event, which can be chosen from an external signal, a trigger event, or a periodic trigger event. The latter two are used to determine the timebase value with full resolution (96 bit) at the time the trigger event is issued. Contrary to the ingress and egress packet events, this type is not buffered in a FIFO, which requires the software to be able to read out subsequent events fast enough. A single bit allows the detection of missed events by the software.

A common use case for this functionality is the synchronization to an external time source, e. g. a GPS receiver. Common receivers provide a 1 PPS signal and a serial connection giving the precise time according to a time standard like the one of National Marine Electronics Association (NMEA). The messages on the serial connection report the time at which the PPS signal was issued. By comparing the transmitted value with the sampled time of the cell in the event register, the offset can be calculated and adjusted by an algorithm. This way, a master synchronization node can be aligned to an external global time-scale.

5.2.4 Trigger Functionality

Besides determining the time value at the occurrence of an (external) event the core also allows to generate signals by internal trigger blocks. The simplest function is the trigger block for single events. The core generates a rising edge on a core signal as soon as the timebase shows a value equal or greater to the predefined one. Reset is done by setting a new time value for the trigger.

In addition to the one-shot triggers, also the generation of periodic events is supported by the synchronization cell. For that purpose, a period can be stored in a register, which is then utilised to define the temporal distance between two pulses generated by the CSC. Internally, the periodic events are handled like one-shot events, but the time for the next trigger event is calculated automatically. On the start event, which can be either a one-time trigger or a manual start event, the period is added to the current time value and used as the next event time. In order to keep an exact frequency independent of the clock granularity, the subsequent periodic event is always calculated from the ideal expiration time with nanosecond granularity. Still the external visible jitter depends on the clock base frequency, which for the prototype was chosen to be in the range $50 - 100\,\text{MHz}$. The maximum frequency of the periodic timer is defined by the recalculation and reset time on expiration and limited to $1/5 \cdot f_{\text{CSC}}$.

5.2.5 AHB Register Interface

The core uses the Advanced Microcontroller Bus Architecture (AMBA) Advanced High-Performance Bus (AHB) [ARM99] to connect to other system components, like the PCI controller for external access. Besides a register read/write access interface the core also provides an IRQ signal for event notification. This signal is a logical OR of the interrupt source register, which can be read to identify the event that occurred. The implementation ensures that the register is automatically cleared on read access. This way it is ensured that no event notification gets lost e. g. due to delays in a two step read-out-write-back cycle method.

Since most bus systems introduce non-deterministic access times to the register, respectively the software triggering such accesses, all parameters of the CSC, e. g. step sizes, are written to shadow registers. The value of these registers can then be synchronously set into operation either by another single register access or on a certain time value, the *apply* time. This functionality completes a set of methods provided by the synchronization cell in order to perform all operations with fixed relation to the internal clock. This way, all time critical operations are performed in hardware and can be scheduled in advance in software without degrading the overall performance.

5.3 Event-Based, High-Accuracy Cell

The core task of the CSC is to schedule actions and detect events with a resolution and precision that cannot be achieved in software. To fulfil this task, the supporting hardware blocks are required to allow scheduling of actions in advance, i. e. operations are prepared and can be executed at a precise moment in time. For the detection of events, the function block has to timestamp with high resolution and buffer the individual incidents till the software is able to take over the data. The designed resolution of the internal timebase should match the expected precision of the MIIS as well as the drift parameters of the driving oscillator. In general the communication between hardware and software is block oriented and the CSC takes the part of reacting to events with low delay and executing functions at an accurate instance in time.

On the other hand, putting too much functionality into hardware increases the complexity and has negative effects on the achievable resolution. The latter is in general dependent on the maximum reachable sampling frequency. The chapter describing the MIIS shows that for detection of reoccurring events other techniques than high sampling frequencies can be used in order to increase the resolution.

For irregular output signals and one-time input events, additional information besides the event itself is required. A common method, e. g. used for 1 PPS signals, is to use a relative low sampling frequency and to additionally provide correction data via a separate channel. This gives exact knowledge, when the signal was raised. Nevertheless, this method only tackles the output issues, therefore only half of the involved components. Additionally, methods that are based on the knowledge that a signal occurs regularly do not cover the detection of asynchronous signals. For the latter only high sampling rates can assure accurate timestamping or event generation. This functionality has to be provided by special measures described in the following subsections.

5.3.1 Prescaler

As illustrated in figure 5.1, the adder-based structure of the CSC involves a quite wide adder that increments the timebase (104/136 bits for 32/64 bits of seconds) by the step register holding 48 bits (or 64 bits if the full step register width is used). In any case, the typical FPGA internal implementation of such adders introduces significant delays, since signal routing between individual logic cells is required. Even in ASICs the structure is relatively complex although special building blocks can be used.

The calculation of the timebase with high resolution is required in order to compensate also for small drifts in the oscillator frequency. Therefore, the adder cannot be shortened in order to increase the sampling frequency for asynchronous event detection or trigger outputs. However, using an additional function block called *prescaler* helps to solve the frequency problem.

A prescaler is a part of the design – commonly a narrow-width counter or shift register – that runs with an integer multiple of the timebase clock. The latter is fed into a PLL to generate a phase stable derivative. The new high-frequency clock is then used for an e. g. 8 bit counter. Typically an 8 bit counter has at least double the speed of a 64 bit adder, which allows for the presented architecture multiplication factors of eight or even higher.

In case of an external event, the timebase and in parallel the fast counter are frozen. Both values are stored and the fractional increment, given by the fast counter value is added to the state of the timebase. The sum then gives the actual timestamp with high resolution. Similarly, also output triggers can be produced. In this case the counter is compared to a fractional value and on equality the output is generated. The counter itself is enabled by a comparison of the timebase to a predefined trigger time.

As mentioned, both issues can also be handled by a shift register. For the input, the signal is fed-in, generating a thermometer code, which can be converted into a binary value on the next edge of the timebase clock. The method also works for the output generation, by using a comparator and a predefined pattern for the fractional part.

5.3.2 Adder-Based Clock Enhancements

Based on the considerations of chapter 5.1.1 the exact implementation of the core of the CSC, the timebase function block, determines the upper limit of the achievable core (and therefore sampling) frequency for a given technology. One important optimisation step

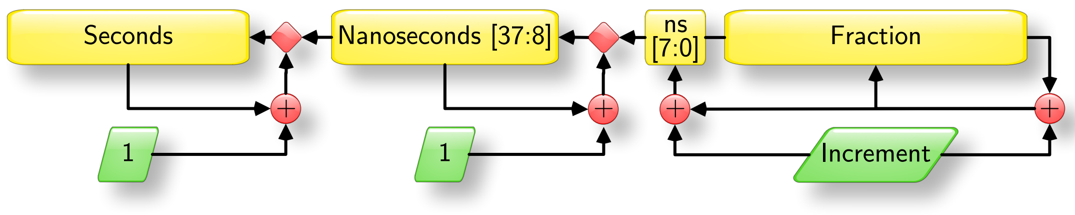


Figure 5.6: Optimised adder-based clock structure using counters

that is quite simple to implement, is to reduce the involved bit widths to a minimum. Although, the whole design is developed for a 32 bit architecture, a reduction of the critical timing path can significantly improve the performance. That is the reason for the increment to be only 48 bit instead of the full register width. The adder can therefore be less complex.

Another significant enhancement is the replacement of adder-based structures with counters. The latter can be incremented only by one, but offer approximately 10 % higher performance [Lat08]. Further improvement can be drawn from the fact that the overflow check is reduced to an *equal* comparison instead of *greater or equal* check. In the present optimised implementation, the adder only covers the increment of the fractional nanosecond part and the lower 8 bits of the nanoseconds. The overflow of the adder is checked by a *greater or equal* comparison, since the increment can vary and the overflow value is dependent on it. The overflow triggers the increment of the upper nanosecond counter [37:8] by one.

The final value of the counters is known, since the increment is defined. Thus, the overflow can be verified by an *equal* comparison which further increases the processing speed. The latter is mainly dependent on the chain of carry signals from the individual bit adders. The limited width of the adder increases the speed, but of course defines a lower boundary for the driving oscillator frequency. The chosen width of the increment also allows to check the nanosecond overflow using an *equal* comparison which again improves the performance of the core timebase function block.

Similar optimisation can be done for the periodic and standard timers. The knowledge that the upper part of the nanoseconds cannot change more than by one, the detection of the trigger event occurrence can be split into the upper part being compared to be equal and the second overflow condition enhancing the comparison speed. Especially the periodic timers benefit from this speed-up since a sequential recalculation of the next trigger point has to be done which is substantially more effort than the simple trigger condition by time. The processing speed also benefits from a greater lower bound of the periodic timer frequency, since less bits are necessary to represent the value and therefore lowers the calculation delay. The drawback of a high minimum frequency can be compensated by software support that takes over the rescheduling of periodic events.

5.3.3 Tick-Based Architecture

In contrast to the previously mentioned timebases, a tick-based architecture does not count real time values, but (fractional) ticks of the driving oscillator. Like the first, the structure can be used to compensate for inaccuracies of the source clock but does not directly give the current time rather than a corrected tick value. The latter has to be mathematically transferred into a time value based on using the source clock frequency. This operation can be done in software which reduces the required effort in hardware allowing for increased clock frequencies and therefore higher precision. Since most synchronization protocols require the real time to be transmitted, either the use of two-step clocks is enforced or a separate translation block for calculating the time value out of the ticks has to be used (in hardware). Still, this method gives advantages for the operation, because for the translation of timestamps multiple clock cycles can be used, whereas triggers, event detection, and setting of internal control values require one-cycle operation (e. g. of comparators).

Important for the achievable performance of a synchronization cell is the minimal time error that the cell can compensate for. The architecture shown in figure 5.6 has a fixed minimal step size. The minimal error, the cell can compensate for during one synchronization interval can be calculated using the definition of the precision in equation 5.1, $\Pi_{\text{std}} = T_{\text{sI}} \cdot f_{\text{osc}} \cdot \Delta_{\min}$, whereas the influencing factors are the synchronization interval, the source oscillator and the minimal settable step size, $\Delta_{\min}$. For the tick-based architecture, the step size is implicitly the period of the oscillator and therefore the error, Π_{tick}, using $\Delta_{\min} = {}^{1}/_{f_{\text{osc}}} \cdot \Delta_{\text{tick}}$, calculates to

$$\Pi_{\text{tick}} = T_{\text{sI}} \cdot \Delta_{\text{tick}}. \tag{5.2}$$

Consequently, the internal representation of the step size can be reduced by the number of bits required to note the period of the oscillator. This allows either for higher precision using the same number of bits, or a significantly reduced internal calculation bit widths. For the step size using a tick-based architecture, the number of bits can be fully used for precise representation, i. e. $\Delta_{\text{tick}} = 2^{-\text{bits}}$.

As it can be trivially shown, depending on the given application, the reduction can be as much as 50 % of the bit width (equivalent to a performance enhancement of approximately 25 %) at the price of non-direct real time read-out. The latter can be compensated by appropriate function blocks, which use multiple clock cycles to translate the ticks into time values using multiple clock cycles. Such structures require hardware multipliers and therefore quite some chip resources.

An alternative to the mentioned hardware multipliers is to avoid the usage of human readable time values in hardware completely and shift the translation to software procedures. For ingress timestamps this is no problematic issue, because the timestamp is anyway transferred to software functions and can be transformed there. The necessary support for raw values (special time representation) can be integrated in a similar manner as for human readable time values into the network stack of the operating system including the translation functions, e. g. Linux kernel versions since 2.6.30. Nevertheless, for egress timestamps it is not possible to use one-step clocks due to the required processing

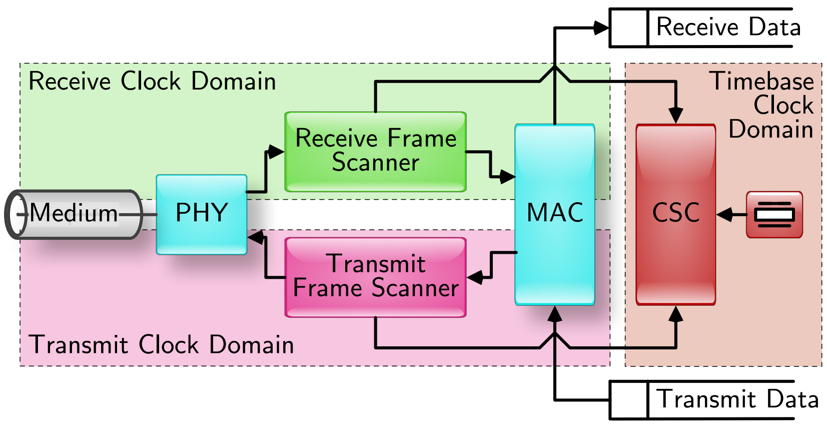


Figure 5.7: Clock domains involved in timestamping on xMII

time. Consequently, only two-step clocks can be used for such structures, which require more network frames but have no influence on the achievable accuracy.

The advantage of the latter method is that due to the required translation, any synchronization protocol can benefit from the hardware timestamping, because the translation function can be adapted accordingly in software. Further, the storage of time values greater than approximately one second can be shifted to software functions. That means that only parts of a second are counted in hardware, whereas the seconds themselves are counted in software on an overflow of the hardware adder. The required merging of the software counter with the hardware value delays the availability of the final timestamp making two-step clock functionalities unavoidable.

5.4 Media Independent Interface Scanner

The MIIS is a unit attached to the xMII of an Ethernet device. The core task is to detect the event of incoming or outgoing packets and issue the storage of a timestamp in the CSC. Also, it handles the transitions of the event signals between the different clock domains of the interface and the one of the local node as shown in figure 5.7.

In general, except for R(G)MII, the receive data path runs synchronous to the clock of the remote side of the link, while the transmit path uses a local oscillator for data transmission. While the first is always a separate clock, the latter one can be derived from the clock of the timebase and therefore one clock transition can be saved, if an integer multiple is used. The generic method is to treat both paths asynchronous to the timebase and use special methods for accurate timestamping as described in chapter 5.4.2.

5.4.1 Functionality

Either the CSC is capable of handling timestamps of all incoming and outgoing packets or appropriate pre-filtering has to be implemented in this unit. Timestamps for all packets also require the necessary measures to transport timestamps attached to the

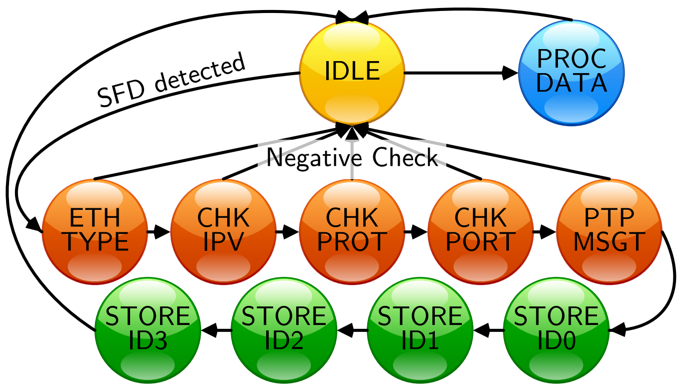


Figure 5.8: State diagram of the configurable MIIS

packet in all network layers up to the application performing the actual synchronization algorithm, as described in chapter 4.4. Since current standard network stacks of common operating systems do not support the transport of high-accurate timestamps directly to the application, a bypass solution has to be utilised.

The architecture of the MIIS supports two independent parsing paths, which are implemented in a similar way. While the first is an optimised state machine parsing the incoming data stream for specific synchronisation packets, the second parsing path allows to be programmed in order to obtain also timestamps for user-defined packets.

The logic functionality of the MIIS is shown in the state diagram of figure 5.8. The state machine starts in the *IDLE* mode, which stays active as long as the MII shows an error or no frame transmission is active. As soon as there is an active transmission, the block scans for the SFD. If the pattern is detected several checks (see second line of the figure) are performed in case the block is used to perform automatic parsing. For both methods, the scanner keeps track of the current position within the frame using an internal byte counter.

The user-defined checks are done in the *PROCDATA* state. The state machine sequentially reads position/compare/mask patterns from a memory block and checks whether the pattern (filtered by the mask) occurs at the specified position. Additionally, the scanner can be configured to shift the current byte into a register in order to verify the exact value later in software. As soon as all checks have been successfully executed and the position of the next check is specified to be lower than the current, the frame is marked as positively checked and the state machine returns to the *IDLE* state.

For the implementation supporting the IEEE 1588 protocol, the automatic parsing mode, first performs a number of checks – starting with the *ETHTYPE* till *PTPMSGT* state – to verify that only packets of the protocol are timestamped. Due to the flexible structure keeping track of the current position within the packet and additionally using a dynamically adjustable verification position, the scanner is able to support a number of frame types despite its small size. Supported frame types are (and arbitrary combinations of the listed properties): basic and Q-tagged (according to IEEE 802.1q) Ethernet frames, IPv4 and IPv6 standard header frames with dynamic header length or IEEE 1588 layer 2

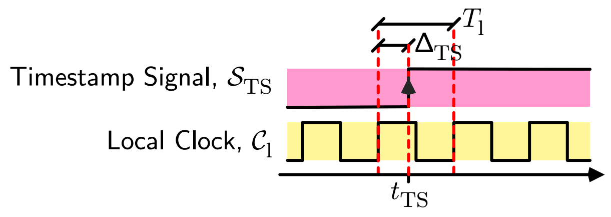

T_l
Δ_TS
Timestamp Signal, $\mathcal{S}_\mathrm{TS}$
Local Clock, $\mathcal{C}_\mathrm{l}$
t_TS

Figure 5.9: Timestamping error due to sampling on a clock boundary

frames. After the verification of the frame type, the protocol, port, and the PTP message type (only event messages are timestamped) identification information (source MAC address and packet sequence number) of the packet is stored. Hereafter, the frame is acknowledged to be stored in the timestamping FIFO of the synchronization cell.

5.4.2 Precision

One of the limiting factors of the overall clock synchronization system is the precision of the timestamps that can be drawn. The simplest version is to generate a frame detection signal from the incoming data stream and sample it with the local clock, $\mathcal{C}_\mathrm{l}$, of the CSC. Since the sampled data stream and $\mathcal{C}_\mathrm{l}$ are sourced from independent oscillators timestamp error Δ_TS is in the range of $[0...T_\mathrm{l}]$ and the resulting precision is expressed by $\sigma_\mathrm{TS} = \sqrt{T_\mathrm{l}^2/12}$. The context is illustrated in figure 5.9. Due to the complexity of the CSC, the base frequency of the cell is limited to a few hundred MHz in current FPGA technologies, which consequently limits the precision to several nanoseconds. Therefore, other methods have to be chosen to decrease Δ_TS. These techniques to enhance timestamping accuracy by reduction of the jitter Δ_TS can be grouped in the following two subcategories:

- Single-Shot methods measure the time between the assertion of the timestamp signal $\mathcal{S}_\mathrm{TS}$ with respect to the rising edge of $\mathcal{C}_\mathrm{l}$. The techniques have to provide a significantly better time resolution than T_l.
- Phase estimating methods make use of the fact that the timestamp signal is asserted synchronously to one of the communication clocks (either receive or transmit), $\mathcal{C}_\mathrm{c}$. Therefore, by evaluating the current phase offset between $\mathcal{C}_\mathrm{c}$ and the local clock, $\mathcal{C}_\mathrm{l}$, the phase of $\mathcal{S}_\mathrm{TS}$ with respect to the rising edge of $\mathcal{C}_\mathrm{l}$ can be estimated.

Both methods have in common that they measure or estimate the phase Δ_TS with respect to $\mathcal{C}_\mathrm{l}$ and therefore it can be better expressed by a relative phase offset $\delta_\mathrm{TS} = \frac{\Delta_\mathrm{TS}}{T_\mathrm{l}}$. Assuming that the value of the ABC is increased every T_l by the increment I, the value of the timestamp is calculated by

$$\mathrm{TS} = C(t_\mathrm{TS}) + \delta_\mathrm{TS} \cdot I \qquad (5.3)$$

An overview of available methods for measuring time intervals with high resolution can be found in [Kal04]. Out of these techniques, for the subject of this thesis only methods that can be implemented fully into modern digital logic devices without the necessity of external components were considered. Even though they can deliver comparable or even better performance, approaches using external Analogue to Digital Converters (ADCs) or mixers are not considered, since there is no significant expected benefit from the additional precision that compensates for the higher implementation effort and costs. In the following only the most relevant techniques for use in an FPGA or ASIC are tackled.

Single-Shot Methods

Single-shot methods measure δ_{TS} directly. That is done by determining how long after the rising edge of the local clock the timestamp signal $\mathcal{S}_{TS}$ has been asserted. For that purpose a clock cycle T_{l} is divided into $n \in \mathbb{N}^{+}$ equally spaced fractions, which reduce the timestamping variance to $\sigma^2_{\mathrm{TS}} = \frac{T_{\mathrm{l}}^2}{12n^2}$.

High-Speed Counter. This approach divides T_{l} by a short-width high-frequency counter with a period $T_{\mathrm{h}} = T_{\mathrm{l}}/n$. The counter is reset at every rising edge of the local clock and its value is frozen when $\mathcal{S}_{\mathrm{TS}}$ is active. The sampled counter value divided by n then describes the relative phase offset δ_{TS}. Since the period of the local clock is exactly a multiple of T_{h} the clock transition between these two clocks can be designed without any additional jitter and consequently the timestamping variance reduces by n^2. The result is similar to a design completely clocked with $1/T_{\mathrm{h}}$ with the advantage that only a few logic elements have to run at a high frequency.

With very low additional effort, n can be doubled using registers sampling with the opposite edge of the clock. Such register pairs using both edges, called Double Data Rate (DDR) registers are available in many devices to be used for communication links. The additional improvement by a factor of two is achieved without change in the clock rate and is a special case of the next option requiring only an inverter.

Phase Shifted Clocks. The use of phase shifted clocks is another method to partition T_{l}. For this technique, $n - 1$ additional clocks are generated, which are phase shifted by $2\pi/k$ with $k = 0...n - 1$ with respect to $\mathcal{C}_{\mathrm{l}}$. The timestamp signal is registered into n registers, each with a different generated clock. If the $\mathcal{S}_{\mathrm{TS}}$ gets active, the first i registers still sample the old state. This generates a thermometer code, which is converted to a binary code and used in the same manner as the high-speed counter. Since $\mathcal{S}_{\mathrm{TS}}$ drives n registers, special care has to be taken that the clock at the registers has the designed phase shift (i. e. the registers are timely equally spaced), since otherwise the thermometer code renders non-linear. This effect and the number of output clocks per PLL limit n to value of about 10 in state-of-the-art FPGA devices (e. g. Altera Stratix III family).

Tapped Delay Lines (TDLs) are a common approach for digitizing times with sub-nanosecond accuracy. The basic configuration of a TDL consists of a serial chain of n latches having a delay τ_{L}, a second chain of non-inverting buffers with delay $\tau_{\mathrm{B}} < \tau_{\mathrm{L}}$, and an output logic as described in [KSPP97]. The signal to be timestamped is then fed through all latches, which freeze its current state at the rising edge of a second signal (clock). The resulting thermometer code can then be evaluated after the next clock edge.

Nevertheless, it has to be considered that such a design uses asynchronous logic and therefore the delays τ_L and τ_B are not only placement, but also temperature dependent. The linearity of a TDL may be compromised by these effects and special calibration logic may be required. The possible accuracy depends on the intrinsic switching speed of the latches, which is typically in the range of 100 ps [SKS00].

Phase Estimating Methods

Phase estimating methods do not measure δ_TS directly, but rather estimate δ_TS using the fact that $\mathcal{S}_\mathrm{TS}$ is asserted synchronous to the communication clock. The relatively new approach for highly accurate time interval measurements is described in [ZSYZ08]. It is realised by means of an implementation with analogue components.

The sample implementation is based on sampling a 10 MHz atomic clock with an ADC driven by the communication clock and performing a phase estimation based on an 1024 point Fast Fourier Transform (FFT). While the results show a very low timestamping standard deviation of about 10 ps, this approach requires one ADC per timestamper and an atomic clock to achieve the mentioned performance. As with analogue single-shot methods, this is rather impractical for timestamping in the targeted application area. Consequently, a technique to implement the scheme using pure digital components has to be developed [EL09].

The frequencies of the involved clocks have to fulfil several requirements in order to benefit from the method of phase estimation: First of all, the rising edge of the local clock T_l should occur equally distributed within the clock cycles of T_c averaged over a given time span (i. e. the rising edge of the local clock should cover all phases of the communication clock with equal probability). This includes that T_c must not match T_l or a multiple thereof because in this case the rising edge would always coincide with a certain phase of T_c and the necessary averaging time span would be infinite.

In order to be able to sample T_c (or signals synchronous to it) at a number of different phase states, the local clock T_l has to be a non-multiple of the communication clock. To represent this criteria, the nominal period is given by

$$T_\mathrm{l} = T_\mathrm{c}\frac{1-\beta}{n}, 0 < \beta \ll 1 \tag{5.4}$$

with the design parameter β as the relative frequency offset factor and n as the nominal oversampling factor. A cycle slip occurs, i. e. when the rising edge of two clocks pass each other, if the difference in the periods sums up to T_l.

$$T_\mathrm{l} = \frac{1}{\epsilon}\left[\frac{T_\mathrm{c}}{n} - T_\mathrm{l}\right] \tag{5.5}$$

$$T_\mathrm{l} = \frac{1}{\epsilon}\left[\frac{T_\mathrm{l}}{1-\beta} - T_\mathrm{l}\right] \tag{5.6}$$

$$\epsilon = \frac{\beta}{1-\beta} \tag{5.7}$$

Using equation 5.7 cycle slips will happen after every $1/\epsilon$ local clock periods. This implies that $\mathcal{C}_\mathrm{c}$ has been sampled at $\lceil 1/\epsilon \rceil + 1$ different phase points over one cycle slip period and the maximal attainable accuracy is nominal $T_\mathrm{l} \cdot \beta$. Selecting a very small β results in a small frequency offset and great resolution, but this can cause $T_\mathrm{c} \leq nT_\mathrm{l}$ due to instabilities of the two involved oscillators during the long theoretical cycle slip period. Consequently, cycle slips might not happen at all or even worse, reverse cycle slips can occur, resulting in possible data loss.

Furthermore, small frequency differences are unusable since the rising edge instance is only equally distributed within T_c over a long averaging window and for short averaging windows the leakage effect [DG94] becomes dominant.

Multiple Timestamps per Frame. Timestamping a frame m times by means of the communication clock is one option to estimate the phase at the timestamping event based on the assumption that δ_TS averages to 0.5. Given that the timestamps are centred before and after the assertion of $\mathcal{S}_\mathrm{TS}$ (i. e. $\frac{m-1}{2}$ timestamps before and after the event) and that the ABC's increment is not altered during the timestamping period, the final timestamp TS is calculated by a weighted average over the timestamps TS_i following

$$\mathrm{TS} = \sum_{i=-\frac{m-1}{2}}^{\frac{m-1}{2}} \alpha_i \mathrm{TS}_i, \quad \sum_{i=-\frac{m-1}{2}}^{\frac{m-1}{2}} \alpha_i = 1. \tag{5.8}$$

This Finite Impulse Response (FIR) filter can be simplified to a window integrator with $\alpha_i = 1/m$ with some limitations, namely leakage effects. The timestamping window should cover one cycle slip period or a multiple thereof to get the timestamps equally distributed over one T_c period. Since the cycle slip period is dependent on the current frequency offset, it varies with the oscillator drift between the communication and local clock. One solution to this problem is to adjust m to cover always a multiple of the cycle slip periods or by capturing a big number of such periods and using a windowing function to minimize leakage effects. The leakage can also be reduced by selecting a rather large ϵ with the drawback of reduced resolution.

In the optimal case the resulting timestamping variance reduces to $\sigma^2_\mathrm{TS} = \frac{T_\mathrm{l}^2}{12m}$. For example, a typical IEEE 1588 frame with about 80 bytes frame length and a nibble-wide interface can deliver $m = 160$ timestamps. Given that $T_\mathrm{l} = 10\,\mathrm{ns}$ the standard deviation resolves to 228 ps.

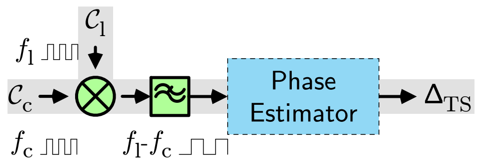


Figure 5.10: Direct phase estimation by mixing

Digital Phase Estimation. The phase of the communication clock can be directly estimated by phase detectors. Such a detector is based on mixers, which shift the spectrum of the clock to a low frequency as shown in figure 5.10. The output of the mixer is low-pass filtered to remove aliases at multiples of the input frequency and is conducted into a phase estimator. Given that the duty cycle of the clock input signal is constant, the low frequency part of the down mixed signal then is a measure for the phase difference at the inputs. Nevertheless, real filters with a low bandwidth introduce significant group delay, which has to be taken into account for the calculation of the timestamp. In order to allow for a digital implementation the mixer can be replaced by an XOR gate and the output can be filtered in the same way. A further solution would be to use an external analogue anti-aliasing filter in combination with an ADC and only perform the second filtering digitally.

A pure digital implementation without requiring external parts can be achieved, but in such a processing scheme undersampling occurs. As the XORed signal is not band-limited, sampling results in alias frequencies that can dominate the signal (e. g. if the “1 bit” sampler is sourced from a clock correlated with $\mathcal{C}_\mathrm{l}$). One feasible solution is to sample the mixed signal by a clock odd to both inputs and apply the sampled signal to a low-pass filter with very low relative bandwidth. Such filters are typically Infinite Impulse Response (IIR) type since comparable FIR filters would need a big number of filter taps. IIR filters on the other hand have a frequency dependent group delay, which means that the frequency offset between $\mathcal{C}_\mathrm{l}$ and $\mathcal{C}_\mathrm{c}$ must be estimated in order to compensate for the filter’s group delay.

Combined Phase/Frequency Estimation. Rather than estimating δ_TS directly, it is also possible to estimate the phase by its derivative, the frequency offset, together with a reference point. The principle of phase estimation by frequency estimation can be implemented as follows: Whenever the communication clock is phase aligned to the local clock (that is when a cycle slip occurs), the phase estimation $\bar{\delta}_\mathrm{TS}$ is set to zero. In every subsequent clock cycle $\bar{\delta}_\mathrm{TS}$ is incremented by the estimated inverse cycle slip period $\bar{\epsilon}$. In other words, $\bar{\delta}_\mathrm{TS}$ is the sampled integral of $\bar{\epsilon}$ over one cycle slip period. Given that the frequency is stable in the averaging interval, $\bar{\delta}_\mathrm{TS}$ ideally would reach 1.0 at the next cycle slip as depicted in figure 5.11 for $n = 1$. For the reason of better visibility, a relative high value of $1/4$ was chosen for $\bar{\beta}$ resulting in a cycle slip period of four local clock cycles. As the logic illustrated by the figure only uses positive detections of the cycle slip, i. e. when the $\mathcal{C}_\mathrm{d}$ is high, the period is doubled.

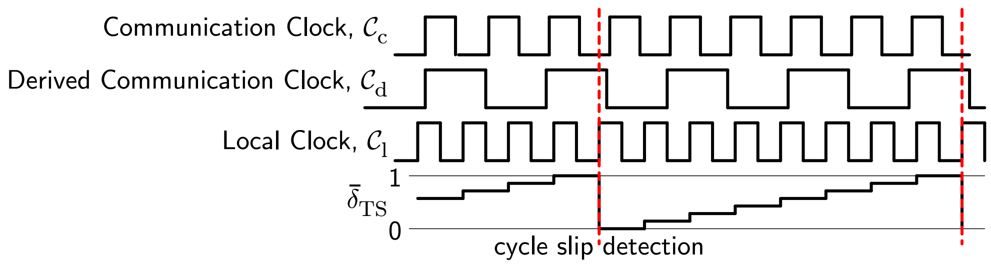

Communication Clock, $\mathcal{C}_c$
Derived Communication Clock, $\mathcal{C}_d$
Local Clock, $\mathcal{C}_l$
1
$\bar{\delta}_{TS}$
0
cycle slip detection

Figure 5.11: Phase estimation between cycle slips

The accurate detection of the cycle slip instant is critical for the start of the integration. To make the method independent of the communication clock duty cycle, a derived clock $\mathcal{C}_d$ with the period $2T_c$ is generated digitally. $\mathcal{C}_d$ is fed into a shift register with $5+n$ taps clocked with the local clock. Further, this clock is used for cycle slip checking and performing edge detection. While the first two shift taps are used for buffering, the middle taps ($2+n$ down to 2) are used for cycle slip detection. If all $n+1$ middle taps contain the same binary value, a cycle slip must have happened. The last two taps (1 down to 0) are used to detect rising and falling edges of $\mathcal{C}_d$ at which the buffered timestamp signal $\mathcal{S}_{TS}$ is checked for a high level.

The relative frequency offset $\bar{\beta}$ can be calculated by monitoring the number of rising edges of $\mathcal{C}_c$ with respect to $\mathcal{C}_l$. In average every $1/\bar{\epsilon}$ local clock cycles a cycle slip will occur, which results in a missing rising edge with respect to the local clock. In order to have a continuous value of $\bar{\epsilon}$, a low-pass filter has to be applied. The bandwidth of the filter must be narrow enough to track frequency changes of the oscillator while removing the cycle slip frequency. In general, IIR filters which have poles close to one are ideally suited for this application. Alternatively, the frequency offset $\bar{\epsilon}$ can be calculated by the cycle slip rate $1/\bar{\epsilon}$, but this requires a division block, which consumes a significant amount of logic resources. In any case, the calculation of the final timestamp involves summing up the ABC's last value plus $\bar{\delta}_{TS}$ times the increment I.

The cycle slip rate not only has to respect the accuracy requirements of the application, but also the behaviour of the physical layer. As mentioned in chapter 4.3, e. g. 10 Base-T does not provide a continuous clock supply that can be measured by the phase/frequency estimation. Therefore, the factor β has to be chosen in a way that the measurement can be done within the reception/transmission time of a single packet. Alternatively, the accuracy can be enhanced by collecting the measurements of several packets, if constant behaviour of the oscillator can be assumed. This is only true for small inter-packet delays. Obviously, the provision of a continuous link implies much higher potential for accurate timestamping.

One major advantage of the phase estimating methods over single-shot techniques is the fact that the measured signal only involves one digital processing path. Unless for

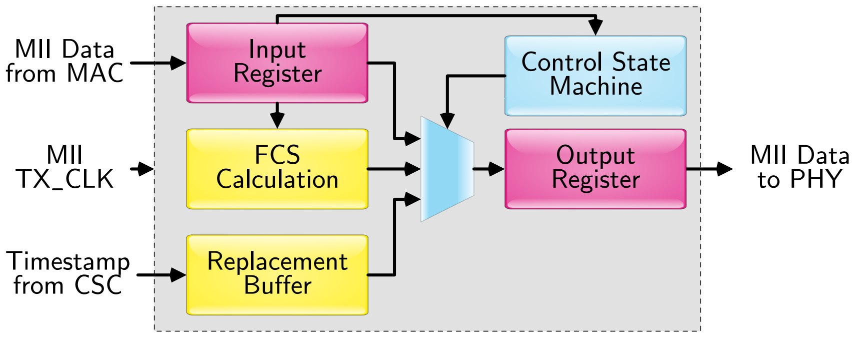


Figure 5.12: Additional functionality required for MIIS to support one-step clocks

the latter techniques only one sampling register is directly connected to the $\mathcal{S}_{\mathrm{TS}}$, or the synchronous clock signal respectively. Therefore, linearity problems due to unequal signal propagation delays to registers or between other logic elements, e. g. buffers in TDLs, cannot occur. This makes the mentioned methods also robust against temperature and other effects, which have direct influence on the signal propagation properties within an integrated circuit. Nevertheless, there might be a small (asymmetric) delay between the receive and the transmit path since they are two instances of the mentioned method. For each chip, this delay has to be calibrated once to compensate for internal placement and routing issues.

5.4.3 One-Step Clock

In order to support also one-step clock nodes, the MIIS in combination with the CSC has to support the replacement of timestamps with their actual value in special fields of the transmit packet during the transmission on the MII. As already mentioned earlier, the CSC has to provide the timestamp of the currently transmitted packet in the format of the protocol within the time between the start of the transmission and the position of the timestamp in the packet.

Figure 5.12 shows the structure of the additionally required hardware that has to be included in the MIIS. The MII is not only scanned by the MIIS but passed through. Therefore, the block has an MII towards the MAC and another one to the PHY. In between, the block can switch from pass-through to replacement operation by simply setting the multiplexer to output the data from the internal register instead of the input. The decision is taken by the controlling state machine, which verifies that the packet is a synchronization frame and controls the packet position counter. If an appropriate frame is recognised and the desired position reached, the timestamp in the frame is replaced by the data provided by the CSC. Additionally, the unit replaces the FCS at the end of the frame to make up a valid transmission.

The whole block runs with the transmit clock of the interface and has to obey the timing restrictions defined by the Ethernet standard [IEE08a] in clause 23.11. The delay from the MAC to the line (`TEN_PMA+PMA_OUT`) has to be guaranteed to match the range of $7 - 17.5$ bit times. If the structure including the PHY matches this requirement (the example in the figure adds two bit times) the unit can be used to avoid the Follow_Up messages and run as a one-step clock.

6 Synchronization System

The overall goal of a network-based clock synchronization system is – as suggested by the name itself – to synchronize the timebase of two or more network nodes. Chapter 5 tackles the necessary hardware enhancements in order to prepare a single node to be integrated in a highly accurate clock synchronization system. This chapter expands the ideas of the previous one and deals with the necessary means required to build a complete system able to keep at least two clocks aligned. The described principle can also handle bigger networks, larger distances, or multiple network hops.

Even the simplest thinkable setup, the direct connection of two network elements via a NIC, requires additional effort over the so far given enhancements to enable accurate synchronization of the timebases. For the given example at least a set of carefully selected software functions is required to allow the operation of the system. Larger systems demand the introduction of intermediate network elements (i. e., especially switches and routers) and make additional software and/or hardware support for clock synchronization unavoidable. Furthermore, function blocks are required, which provide means to measure the performance of the overall system (end-to-end) from an observer point of view. This chapter also mentions adaptations for the previously presented hardware block to fit special network elements (small nodes, switches) in order to allow any type/size of node to participate in the synchronization system.

6.1 Hardware

The core elements mentioned in chapter 5 have to be embedded into surrounding function blocks in order to provide a fully functional, clock synchronization aware network element. The timestamping functionality has to be extended by the necessary network interface function blocks. The most important network elements required to build an almost unlimited sized Ethernet network are end nodes and switches. If routers (OSI layer 3 relaying devices) are required, they can be built like a common switch but with extended (software) functionality for packet switching. Due to the layered approach any element using higher network layers can be built on the functional elements described in the following.

6.1.1 Network Interface Card

As shown in figure 6.1 a basic network card supporting hardware assisted clock synchronization consists of a number of building blocks that have to be implemented in one or more FPGAs/ASICs. In contrast to highly integrated solutions, which can implement all parts except for the oscillators and the power supply in one ASIC, the chosen development

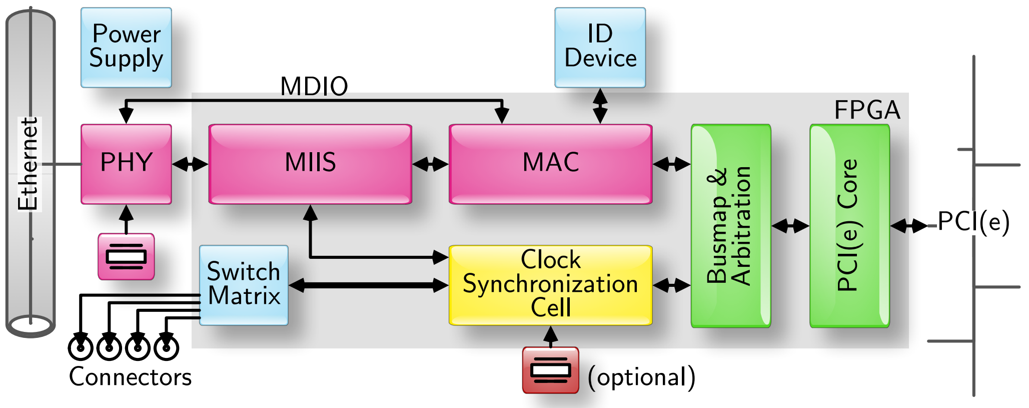


Figure 6.1: Structure of the network interface card

implementation uses an FPGA for all functional blocks except for the PHY. This method allows easy adaptation of the design to different needs, because each functional block can be exchanged by recompiling the configuration of the FPGA.

The device hosts the MIIS, the MAC, the CSC, the bus mapping and arbitration block, as well as the bus interface core itself (e. g. PCI(e)). As peripheral components the following items are required: at least one oscillator (the FPGA can be driven by a generated clock of the PHY), an identification device supplying the necessary Extended Unique Identifier (EUI)-48 number to build a unique MAC address, and the power supply for all board components.

Sampling of external events or the generation of (periodic) triggers is done by the clock synchronization cell, which transfers the signals via the switch matrix to the Sub-Miniature type A (SMA) connectors on the PCI bracket. The switch matrix can be configured to connect any of the four available connectors to one of the internal input or output signals of the CSC. Therefore, any combination of timestamping event inputs, 1 PPS, one-time trigger and periodic trigger signals can be chosen. This can be done during the operation of the card without the need for recompilation of the FPGA design, which eases the use of the card in different environments or application scenarios. Even within a hierarchical synchronization system, it might be required to configure the cards differently e. g. because the master should be tied to an external reference, while the slaves may provide various trigger signals to other components.

The communication with the host system of the network interface card is handled by the PCI(e) controller of the card. Due to the fact that the available IP cores feature different interfaces for data transfer on the client side, the bus-mapping unit is responsible for handling the transfers within the card implementation. The unit encapsulates the functional units within the card from the communication interface and therefore allows to exchange it without need to touch the internal units. This, for example, made it possible to easily generate a PCIe version of the card without great effort. Additionally, it provides arbitration functions to prioritise important data exchange operations, e. g. Direct

Memory Access (DMA) transfers of ingress packets to avoid internal buffer overflows, over other less important functions.

One of the key features of the present board design is the flexible usage of clocks to evaluate different clock synchronization schemes. Since the function blocks for MAC and CSC are hosted in the same FPGA, the synchronization cell can be operated from different clocks depending on the desired behaviour. The core clock can be selected from one of the MII clocks (RX_CLK or TX_CLK) or the special (more stable) on-board oscillator. Consequently, the CSC can operate synchronous to the remote side (useful for point-to-point synchronization if implemented on one side only), the own transmit clock and therefore avoid imprecise measurements of the transmit timestamp, or independent of the communication clocks which is useful for oversampling and required for some of the methods described in chapter 5.4.2. The latter option is further supported by the PCB layout through an additional footprint to equip the card with a highly stable oscillator, e. g. TCXO or OCXO. Since the main clocks of the design are subject to changed sources, the different functionalities have to be implemented through recompilation of the FPGA programming as they are currently not foreseen to be user (online) changeable.

Another small change in the mentioned firmware of the NIC allows to use one of the external SMA connectors as a clock input. Several application scenarios can make use of such a configuration. One would be that the master is sourced from an almost perfect, e. g. atomic, clock in order to measure the behaviour of the clock synchronization mechanism under stable conditions. An enhanced analysis could even source the master and a slave from the same clock to determine the influence of the synchronization algorithm, the intermediate network elements, or the precision/resolution of the timestamps on the overall performance.

The SMA connectors can also be used to evaluate the influence of e. g., timestamping precision, resolution, or algorithm behaviour. By modification of the FPGA firmware several important signals can be easily made available to external measurement instruments in order to perform a detailed analysis of the internal performance parameters. Several useful experiments can be made. Typically it is of great interest to see the actual transmission delay between the TX_EN signal on the master-side and the RX_DV signal on the slave-side MII. If they are measured externally, the comparison with the reported timestamps from the NIC can deliver information about the accuracy of the link delay measurements and also how much jitter originates from the physical layer itself. A differential evaluation of the e. g., RX_DV signal against the receive timestamp sampling signal allows to evaluate the jitter of the timestamps against the actual receive event. Not shown in figure 6.1 is an – by the firmware – unused connector, which would enable the system to output, e. g. packet identification tags, to an external system via a bus interface. Since for most evaluations regarding high precision there is no interest in specific packets, i. e. a single out of a burst, the identification was not used in the course of this thesis.

A number of other analysis methods can be thought of, where the availability of four external signals from the CSC internals can be very useful to evaluate the performance under real conditions. The recorded data can be used to feed simulations for the evaluation of different algorithms in order to have realistic, but reproducible conditions. Also for long distance measurements, where the comparison to an external reference, e. g. GPS, is required, the output of ingress/egress packet events can be very helpful.

From the point-of-view of the operating system (PCI(e) subsystem) both, the MAC and the CSC can be queried/written in the same way. The busmap & arbitration unit unifies the access to the functional blocks by allowing operations on two different address ranges. That way, the registers of the functional units are collected together in a larger address space and the device driver only has to manage a single PCI device in the system. The simplification opens the opportunity for unified functions, e. g. integrated IRQ handling for both function blocks.

6.1.2 Time-Aware Switch

The network interface card described in section 6.1.1 gives a network node the ability to timestamp ingress and egress packets with very high precision. The hardware assistance cancels out jitter introduced by software network stacks and data dependent access times. Nevertheless, this functionality only allows to synchronize devices within one collision domain of Ethernet (nodes connected via a hub), i. e. messages of these nodes are received by every other within the domain. Due to throughput and network size reasons, most current Ethernet installations are based on switches. Unfortunately, these central network elements introduce unpredictable packet forwarding delays. [Cos09,HM09] The processing time within the switch depends on several factors, like the internal architecture, load, packet size, or special packet treatment functions like Quality of Service (QoS) support. In order to compensate for the unknown internal processing times, timestamps within clock synchronization protocol packets have to be compensated for the forwarding delays. Figure 6.2 shows a matching architecture to the one of the network card, which can fulfil the required functionality.

Similar to the network card, the clock synchronization cell draws timestamps for all ingress and egress synchronization packets. Contrary to the functionality of the first, the switch does not store the timestamps for later usage by an application, but has to calculate the difference between the incoming and the outgoing one. This so-called *correction value* can then be either used for an appropriate field in the protocol packet (e. g. the *correctionField* of IEEE 1588 version 2) or to correct any contained message timestamp for the residence time the packet encountered at the switch.

The biggest issue with building a switch that can correct residence delays is the calculation of the correction value out of two timestamps that were produced by one packet on the ingress and the egress part. Since the egress port of a packet is not known during the reception, similar to the possibilities for the network card, the timestamps can be either attached to the packet or intermediately be stored. The latter case requires additional identification information of the packet and a central memory block that is accessible by all egress ports.

Attaching the ingress timestamp to the packet has the advantage that the delta-timestamping functionality (calculation of the residence time of a packet in the switch by evaluating the difference of ingress and egress timestamp) does not require significant additional memory resources. Nevertheless, the frame is extended by the length of the

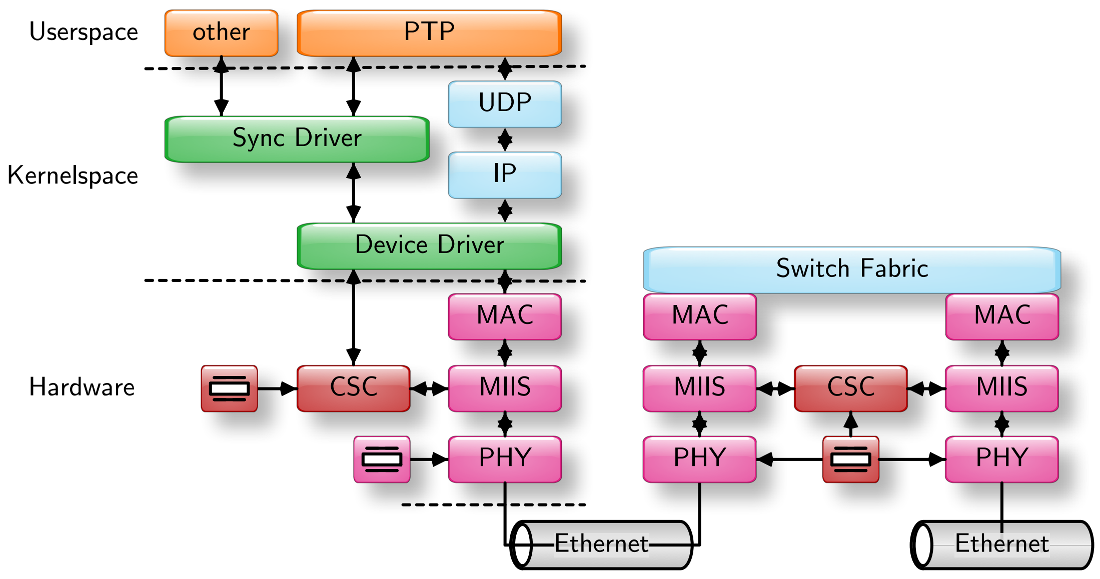


Figure 6.2: Layer architecture for a network with a timestamping switch

timestamp and depending on the architecture of the switching matrix (fabric), some additional temporary memory for frame storage might be required. The modification of the frame has to be done *on-the-fly* in order to ensure competitive performance of the switch. These changes in the frame also bear the biggest issues for the overall implementation.

The appended timestamp has to match the frame processing engine of the switch matrix. Since no existing protocol reserves space for adding temporary timestamps to a frame, the switch has to use some format that allows to transport the timestamp through the switch. The proprietary frame format then causes several issues although it is used only internally.

- Most important, to process clock synchronization packets and eventually correct timestamps, the switch has to be able to parse layer 3 and above protocols unless e. g. PTP on layer 2 is used. In general this violates the OSI reference model and limits the capabilities of the protocols the switch is programmed for. Issues arise if tunnelling or similar techniques are used to encapsulate packets. The effort to implement a number of protocols and their combination over several layers can be significant. Obviously encryption hinders correct delta-timestamping.

- Ethernet defines a frame format with a maximum length that can travel on the medium. Therefore, only protocols that define a maximum length for the packets to be corrected can be used unless the switching matrix is modified to process also packets with excess length.

- Another serious issue arises from the modification itself. Ethernet and most upper layer protocols use some sort of checksum (e. g., for Ethernet the FCS at the end) to ensure frame integrity. This checksum has to be corrected, otherwise transmission errors within the switch cannot be detected anymore. On the other hand, incorrect checksums have to be left as they are in order to allow the MAC and higher layer protocols to sort frames out if necessary.

- A further problem results from the fact that some checksums (e. g. UDP) precede the actual data which requires either the knowledge of the not yet fully received frame or the modification pattern. The first method implies the buffering of the frame till it is available as a whole to fix the checksum. This can introduce unwanted or even impossible delays for reception and transmission. In contrast, the second only requires the current checksum, the algorithm, and the knowledge of the old and the new timestamp value in the packet to do an *incremental* and on-the-fly update of the checksum.

- The timestamp that has to be corrected by the switch is located somewhere in the middle of the frame, depending on the protocol used. Since the ingress timestamp can be added only reasonably at the beginning or at the end of the frame, timing issues arise. In order to fix the timestamp in the packet, the attached timestamp has to be known by the egress correction block. This would vote for a timestamp at the beginning of the frame. Nevertheless, this requires modification of cut-through switch architectures, which require the destination MAC address to be the first information.

- Since a modern COTS switch has to comply with a number of standards it is quite difficult to ensure that the post-modification of frame on a lower layer does not have negative side effects on the upper layer protocols although the modifications only affect the switch's internal structures.

All mentioned issues and the fact that it is not feasible to implement a complete switching matrix for the sake of a prototype render the method of frame extension for the implementation of a time-aware switch, in the context of this thesis, not useful. Consequently, a by-pass solution as for the network interface card was chosen which of course also can have side effects but allows much easier integration with existing function blocks.

For the prototype implementation, the architecture shown in figure 6.3 has been chosen. The basic functionality of the on-the-fly timestamper is implemented in two blocks on each port. The ingress part draws a timestamp for all necessary clock synchronization packets and stores it together with identification information to a memory block. This operation is similar to the one of the network card described in previous chapters. On the egress part, the function block searches for the matching timestamp drawn on the receive side, calculates the difference to the current time, and performs on-the-fly correction of the timestamp contained in bypassing frame or sets the appropriate correction field.

The separated memory, which works together with the on-the-fly timestamper independently from the actual switching matrix has to be well designed in order to deliver correct functionality. Several issues arise from the fact that the two blocks are not directly

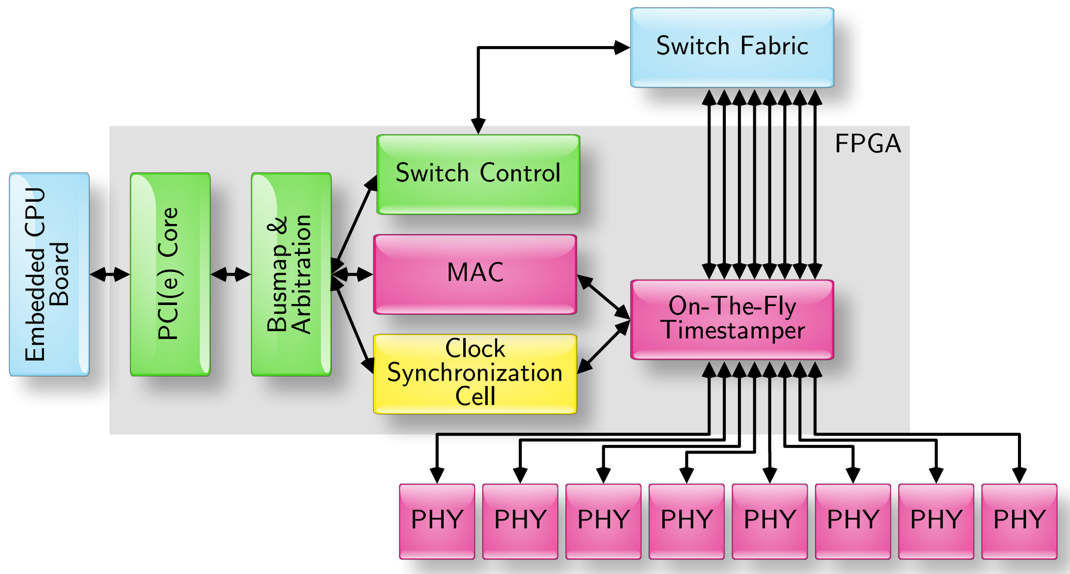


Figure 6.3: Block diagram for the timestamping switch prototype

connected except for the data flow. In order to avoid buffer overflows, there has to be a maximum life time for tuples consisting of the ingress timestamp and the identification information. It is required because frame dropping by the switching matrix can happen which causes an incoming frame never to leave the switch on any port. Such packet loss might happen due to incorrect frame checksums or congestion on the desired egress port, which overflows the internal frame buffer of the switching matrix.

The size of the memory for temporarily storing the timestamps therefore has to be chosen in a way that there is space for all incoming packets that require timestamping during the maximum time assumed for switching. Even for short synchronization intervals, e. g. 10 ms, and a high upper bound for the switching delay of 500 ms the memory of a switch for IEEE 1588 only has to hold about 50 entries plus some additional ones for further packets from the slaves (e. g., Delay_Req messages).

Although in theory, the delta-timestamping can compensate for the jitter introduced by the switching matrix the current investigations [BGNV09] show that available commercial products still suffer from significant inaccuracies, which can hinder highly accurate clock synchronization in the nanoseconds range. As mentioned by the authors, the inaccuracies are mainly due to jitter effects below layer 2 which implies that the characterisation of the transmission system mentioned in chapter 4 is also valid for switch designs. In addition to jitter sources, e. g. clock transitions, also asymmetries, and issues with the syntonization of the switch clock itself influence the performance. The syntonization to the reference clock is important in order to measure the residence time of the packet according to the timescale of the master.

Besides the most important block – the on-the-fly timestamper – the presented prototype shown in figure 6.3 features additional blocks for improved functionality. The embedded CPU board together with the MAC basically server the function of an integrated network interface card that allows the switch itself to work as a timestamping node or master. Additionally, the switch can also serve the function of a boundary clock for PTP using an appropriate software stack running on the CPU board. Although the residence time for frames on the switch is quite short, high-accuracy clock synchronization requires the switch delta-timestamping functionality to run with the same timescale as the (external) timing master. Therefore, the embedded CPU board also serves the function of a timing slave in order to syntonize the clock synchronization cell, which provides the timing information to the delta-timestamping block. Finally, the *switch control* block allows to set parameters of the switching fabric by the on-board processor. The latter allows to run the switch as a completely independent node and to control the switching parameters in dependence of the needs for the synchronization protocol.

Due to the similar characteristics of the network card interface and the switching device (except for the fact that the switch itself not necessarily has to keep the absolute time) also the performance enhancing methods described in the chapters 5.3 and 5.4 can be applied.

6.1.3 Support in Embedded Processors

For an embedded processor, the Hyperstone HyNet 32S/XS [hyp07], the CSC was adapted and integrated into the ASIC. The System-on-a-Chip (SoC) is built for small devices with special focus on real-time Ethernet implementations. Therefore, many of the industrial standard protocols, e. g. ProfiNet, EtherNet/IP, Powerlink, and EtherCAT, are supported. The processor design reflects the needs for communication control in industrial Ethernet installations and therefore integrates a number of features like an integrated Ethernet hub for two ports, a DSP core, low interrupt latency, and a special version of the CSC [LGS06].

An important function of the completely embedded system is the internal clock distribution system. The chip is driven by a single external oscillator (8 − 25 MHz) that supplies a PLL. The latter allows to generate a system frequency of up to 200 MHz. Since not all parts of the chip are able to run at the maximum speed, several dividers allow to generate lower multiples of the system frequency to operate e. g. the Ethernet relevant parts. The CSC can operate with up to 100 MHz allowing timestamps, trigger signals, and event detection with 10 ns resolution.

One integrated PHY and two independent MACs connected to individual external MIIs are available on the device. This combinations allows the embedded system to be used in various different configuration [Duc05] with up to three fully functional Ethernet interfaces. Almost all current real-time Ethernet protocols can be implemented using the available hardware. The MIIS that are connected to the interfaces from the two individual MACs allow highly accurate timestamping on all ingress and egress packets of the chip. The scanner itself can be programmed with simple commands to compare data, skip position, and finish the operation. Consequently, not only the commonly used PTP protocol can be timestamped, but also user-defined patterns.

Verification

One important aspect in the development of the CSC for the embedded system was the verification of the system with high quality against the written specification of the function block. This was required, since changes to the IP core are very expensive after the roll-out and not tolerable for the clock synchronization functionality. Consequently, a test bench operating with register operations (read/write) was developed. Besides these two operations, the test design also allows to trigger or read external events (e. g., IRQ, trigger, event input) and to receive or send frames via the MII.

An important difference to common designs that intend to check the correct functionality of an implementation, the present test bench also allows to precisely check the timing (i. e., completion) of all operations. While in common embedded systems it is sufficient that the system fulfils the requested operations, in this special case, also the point of time of the execution is relevant. Typical examples for such operations are the applying of shadow register values to their effective ones, the (exact) periodic output of a trigger pulse for the period timers, or the raising of an interrupt at a specific point in time. Obviously, for the test of design not only the principle execution of the demanded task is important but the timing as well.

To support the timely testing, the test bench features operations like “wait for x cycles” or “measure time to previously marked point in time” besides “wait for event” which simply waits for a certain event to happen. The combination of both command types allows to check the design with 100 % statement, branch, and condition coverage. Nevertheless, the values of the coverage do not give information on the (correct) execution times, therefore the tests have to be adequately chosen in order to verify the timing as well.

Although verification of the point in time of the execution of certain operations is quite important, it significantly adds testing effort due to the fact that even very small changes in the internal Hardware Description Language (HDL) coding of the function blocks can shift operations by individual clock cycles. This might not be relevant for the overall performance, but should be reported by the test bench and can be a performance indicator (e. g., can be relevant for the dead time of the control loop). The balance between functional and (additional) timing verification should be chosen with care due to the significant effort for adapting the test design even in the case of small changes to the implementation of the CSC.

Evaluation

The aforementioned embedded system in hardware revision 0 was evaluated using a prototype board together with the driver structure under µClinux. The measurement results (histogram of 10 000 samples) of a test setup using three nodes connected via a hub and 10 Mbit/s Ethernet are shown in figure 6.4. The synchronization was maintained by an IEEE 1588 version 1 software stack with an interval of two seconds.

In order to interpret the results correctly, the behaviour of the physical layer has to be considered as well (see section 4.3). For the specific physical layer device used in the processor core the delay at 10 Mbit/s shows four different values spaced 100 ns

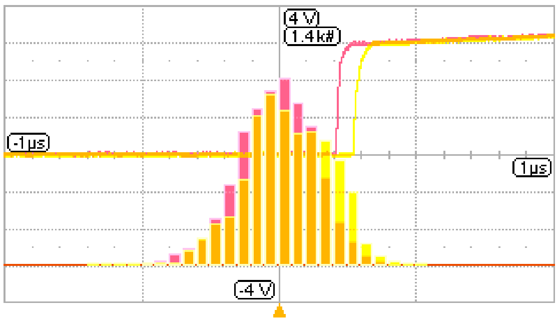


Figure 6.4: Master to slave delay histogram of two network nodes synchronized over 10 Mbit/s Ethernet. All three nodes (one reference master, two slaves) output four rising trigger edges per second for the purpose of measurement. The delay between the signal edge on the master and the corresponding one on both slaves is plotted as a histogram.

apart [GLS06]. Therefore, the transmission times of two consecutive synchronization messages can differ up to 300 ns from each other. Still, under these conditions, the absolute time synchronization accuracy is within a range of ±665 ns, with a mean deviation for slave 1/slave 2 of 26 ns/27 ps, and a standard deviation of 167 ns/142 ns. The results also show that the difference between the two slaves is lesser than the offset to the master, which results from the difficulty to measure the correct network delay also prone to the inaccuracy of the link delay measurements.

Similar measurements using 100 Mbit/s Ethernet point out that switching to that standard allows to improve the accuracy by at least a factor of five using the same hardware. This is mainly due to the fact that in contrast to the lower speed standard, the 100 Mbit/s line speed results in a physical line to MII delay, which shows a Gaussian distribution over some few nanoseconds. Therefore, not the increased data rate, but the significantly lower jitter on the physical layer then gives the better synchronization accuracy. In the first results for Fast Ethernet the accuracy stayed within a range of ±135 ns, with a mean deviation of the slave of 1.7 ns and a standard deviation of 56 ns.

Again, the measurements were taken with a synchronization interval of two seconds and further improvements can be expected, if the interval is reduced to one second. As mentioned later, also the characteristics of the control loop have great influence on the achievable accuracy. Due to the prototype stadium of the implementation, these optimisations were not realised.

6.2 Software

In order to make use of the enhanced hardware functionality described in chapter 6.1 several software components are necessary. An indispensable device driver allows access to the hardware and abstracts the standardised network operations to the API calls of the operating system. Additionally, the extended functionality (timestamping) of the network interface has to be made accessible. Besides the basic procedures required to run the interface compliant to the operating system, synchronization protocol stacks have to be modified to benefit from the high-precision timestamps. Furthermore, also linking software procedures to other time sources or bases, e. g. GPS or the operating system time, is required.

Depending on the flow of the timestamps – in-band, with the packet or out-of-band, on a separate path – different complexity in the operation of protocol stacks is required. The upcoming chapters describe the clock synchronization related software elements located at layer 3 and higher of the ISO/OSI reference stack.

The reference system was set up under the Linux operating system kernel. Thus, the explanations in this chapter refer in detail to this specific structure, although the general concept is valid for most of the modern operating systems. The concept can therefore be ported to other systems as well, nevertheless requiring appropriate adaptations of the system calls for the applications. For kernel space drivers, the situation is a bit different, since they are very tightly integrated with the basic functionality of a specific operating system. Even within the development of the driver for the Linux kernel version 2.6.x, specific APIs had to be adapted multiple times due to the heavy development on the networking subsystem.

6.2.1 Driver and Software Structure

Since PTP and NTP were designed as application layer protocols, the associated software stacks are implemented as applications in the user space. The communication with the network card providing the synchronization frames and matching timestamps is implemented via system calls to the kernel. The two basic functions – a standard compatible network device and the clock synchronization cell – are mapped to two separated devices which allow access to the MAC and the CSC hardware blocks. An overview of the architecture is given in the left part of figure 6.2.

Low-Level Driver

To allow higher flexibility with different hardware implementations, the overall required functionality for translating hardware access to a reasonable user space API has been split up into a high-level and a low-level driver. While the first one implements the communication to the user space, the latter communicates with the hardware or respective kernel subsystems. A stable API between the two driver levels allows to encapsulate the functionality of both in order to be able to easily exchange one of the layers without touching the other.

The interface between the high- and the low-level driver uses five different functions in order to exchange information. Since the information exchange should be event driven to avoid unnecessary consumption of processing time, the device driver defines two callback functions for the synchronization driver (*SyncD*). The latter can register a *device callback* and an *IRQ callback* function. The first is called by the device driver to inform the SyncD about the detection or removal of a network adapter while the latter is used to provide interrupt events together with the content of the interrupt source register (IRSRC) to the high-level driver. The IRSRC register contains information about which event was detected by the CSC.

Two additional functions allow to access the register interface of the CSC. By providing the appropriate CSC handle, delivered along with the device registration, the SyncD can read or write individual register by calling a get and a set function of the device driver. Finally, a further function allows to verify whether an IRQ event is currently processed. This rather simple interface fully allows to exchange the necessary information, to operate the CSC.

Besides the hardware abstraction of the CSC through the five interface functions, the device driver also serves as a network device driver. Although the two functionalities work completely independent of each other, it was chosen to combine them in a signal kernel module, because both are hosted by a single PCI device. Consequently, the communication with the PCI core is common to both and requires unique kernel internal data structures which would have to be doubled in case of a splitting. The network device driver functionality translates the standardised requests of the networking subsystem [CRKH05] to hardware specific commands. This way the timestamping NIC can be handled by the operating system like any other network interface. All extended functions are covered by the synchronization driver.

As already mentioned in previous chapters, the task of clock synchronization is a procedure spanning multiple layers of the OSI reference model. Therefore, not a full abstraction of the CSC can be provided to the user application. Additionally, so far there is no standardised API for accessing timestamping network interface cards. Since the device driver is responsible for all hardware specific functions, all configuration options concerning the actual implementation of the NIC and the required register addresses are provided by a dedicated interface. It contains selection switches for card revisions and different clock synchronization cores. All applications written in C/C++ can use the defined names in order to access hardware specific registers without having to care about the actual address mapping.

High-Level Driver

The SyncD makes the functionality of the clocks synchronization cell available to user space applications. For the present implementation a Linux kernel *character* device [QK06] was chosen. Such a virtual file can be opened and also read and written like any normal file. The CSC registers are directly mapped to the file positions with the same address. Using the standard [IEE04a] file access functions, e. g. seek(), read(), and write(), all data and control registers of the CSC can be retrieved or set. In case an application runs parallel threads on a single file descriptor (e. g., a separate signal handler), atomic operations for setting the file position (register address) and reading/writing the register are required. These operations are fulfilled by the pread() and pwrite() functions.

In general, not only the clock synchronization protocol itself, but also some user applications require access to the cell and its information. Therefore, the task of the synchronization driver is to simplify common operations and to separate the access of individual processes if possible. The chosen method to fulfil these tasks is the virtualisation of important registers of the clock synchronization cell. Obviously, not all functions of the CSC can be abstracted, e. g. the trigger output control, since the functionality is a dedicated hardware resource.

All event controlled and functionally grouped registers require explicit virtualisation. An example for the latter is the value for the current time, which is available via three 32 bit registers: In order to get a consistent read-out value, the hardware freezes all registers on read access to the lowest value which solves the problem for a single execution sequence. Nevertheless, in a multi-tasking environment, the problem of interleaving read accesses to the three registers can occur. To solve this issue, the SyncD performs the three sequential read accesses automatically if a user process accesses the lowest time value and buffers the result individually for each open file descriptor in virtual registers. Since the driver is the central instance to access the hardware, a mutual exclusive locking of the code for read access of the register via semaphores forces all operations in a strict sequential order. Therefore, multiple parallel processes can read consistent time values from a single hardware resource.

Another event driven operation that has to be shared by possibly multiple processes, is the IRQ notification. The CSC sets the IRQ line on certain events to notify the processor on demand without the need for polling the status, e. g. incoming/outgoing packet, external event timestamped, trigger time elapsed, or time counter overflowed. The *IRSRC* (Interrupt Source) register contains several bits indicating the origin of the event which are logically OR'ed to generate the IRQ signal of the core. In order to guarantee consistent read-out and detect the loss (missed read-out) of an interrupt, the register is automatically cleared on read access, avoiding race conditions if multiple accesses would be necessary. In parallel to the interrupt source register a second one, the *IREN* (Interrupt Enable) register with the same layout is available. Setting the corresponding bit there enables the reporting of the specified event via the IRSRC register. If an interrupt occurs, the device driver performs the read-out of the IRSRC register in a timely manner and schedules the call of the previously registered IRQ callback function which resides within the synchronization driver code and contains the necessary functionality to distribute events to multiple processes in an efficient way.

In fact the synchronization driver provides a virtual IRSRC and IREN register for each open file handle. On each access to one of the virtual IREN registers, the driver calculates the logical OR of all virtual registers and sets that value in the hardware. This way it is ensured that notifications about all desired events in the system can reach the driver. In order to notify user applications about new events, the driver allows *blocking* read calls on the IRSRC register, if the file descriptor was opened with the according flag. The program flow is blocked until a new value of the virtual IRSRC register is available. On each notification of the synchronization driver about a new value of the IRSRC register in hardware, it checks whether an application requested one of the IRQs that occurred, modifies the virtual IRSRC register and releases the blocking of the particular code. This way it is ensured that only interrupts that were requested by an application require further processing, which increases the efficiency on high loads.

Though the event driven read-out of the CSC is the more efficient and less CPU

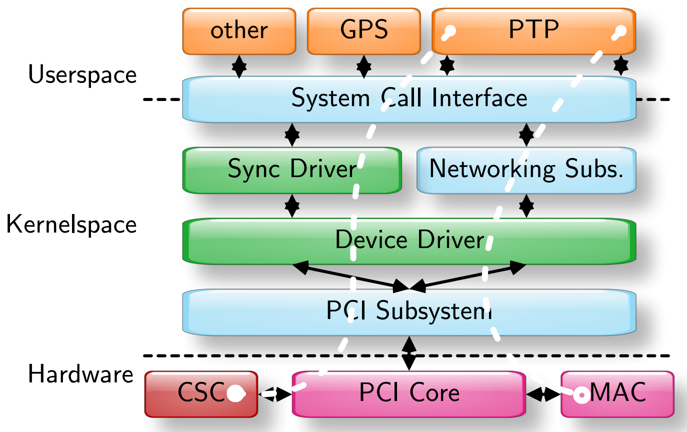


Figure 6.5: Overview on required software components for a clock synchronization system. The dashed paths show the data flow from the CSC on the left side and on the right side from the network (MAC) to and from the PTP stack.

consuming method, the driver also supports non-blocking file operations. The latter can be necessary in case the application itself runs event driven operation but according to some other criteria, e. g. periodic execution.

Software Stack Operation

A block diagram of the software architecture required on a clock synchronization node is shown in figure 6.5. The diagram illustrates the most generic way, with out-of-band packet timestamp transfer which works without any special support of the operating system. The PTP stack opens two interfaces to the kernel: one standardised network interface and a second in order to access the CSC registers. As already mentioned, the second has been implemented as a standard character device. Nevertheless, additional information about the register address has to be provided. This is done in the form of an include file containing appropriate definitions for any user application. The method avoids the introduction of new *ioctl* commands (out-of-band signalling for e. g., network interfaces) in the kernel that have to be unique and require confirmation by the Linux developer community which is hard to achieve. Exceptions are rarely made.

Receive operation. An ingress packet passes the MIIS and produces a timestamp together with an identification tag to be stored in a FIFO of the CSC while it is received by the MAC. The latter also stores an associated buffer descriptor that contains information about packet size or possible errors. The network packet itself is stored in a hardware buffer and the MAC transfers the data via a DMA to the main memory (kernel space) as illustrated in figure 6.6. Subsequently, an IRQ is issued to inform the device driver about the reception. The device driver reads the according buffer descriptor from the MAC, verifies the error bits and passes the data on to the networking subsystem.

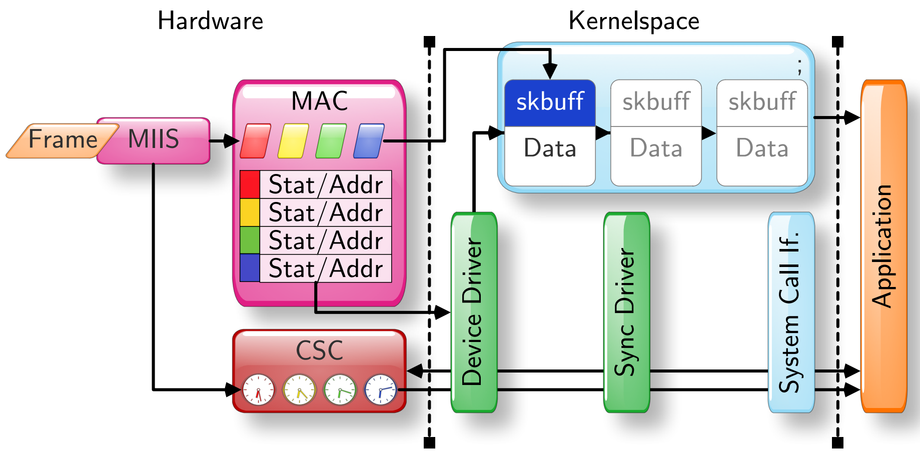


Figure 6.6: Network operation for out-of-band timestamp flow

The latter then informs the user application about the packet receptions and transfers the data for further processing.

The user application subsequently has the option to query the timestamp FIFO of the CSC. This is done via access to the character device, synchronization driver, device driver, and the PCI. The matching is done via the stored identification tag of the timestamp and the corresponding one in the packet. In case no packets are lost – which is assumed to be very likely – always the next timestamp can be expected to match the frame just received.

The issue with the method mentioned above is that only for a subset of the incoming packet, timestamps can be drawn. This is due to the fact that the FIFO in the CSC has limited size and the application (which is commonly a synchronization stack only requiring a small number of packets) would have to read out a high number of unnecessary timestamps. Nevertheless, the solution serves the function for synchronization stacks and does not require special support of the operating system. Furthermore, there is the problem of multiple applications demanding access to the timestamping FIFO. Since a read-out is destructive, all parallel user applications compete about the available timestamps. Therefore, it cannot be ensured that an application receives the timestamp for the corresponding packet. Details about how to overcome these limitations with special support of the operating system can be found in the section on Socket Control Message Flow.

Transmit operation. Sending packets is a bit more complex than receiving, since the timestamp for a packet is not immediately available for further processing. The application issues the *sendto* system call to hand over the frame data to the kernel interface. The kernel network subsystem does the necessary protocol encapsulation and hands over a pointer to the complete network frame to the device driver. The latter prepares a buffer descriptor in the MAC with the necessary length information and copies the frame to a DMA area also referenced there. As soon as the driver sets the *ready* flag, the MAC starts copying the frame data to an internal FIFO and starts transmission as soon as the medium is available.

When the frame is transmitted over the MII to the free medium, the MIIS issues the generation of a timestamp in the CSC. Again, it is stored as a tupel of the timestamp itself and a unique identifier of the packet. The MAC issues an IRQ as soon as the packet has been completely transmitted to inform the driver about the status (e. g., number of retransmissions) and that the corresponding buffer descriptor can be used again. The interrupt notification also enables the device driver to update the interface statistics (e. g., number of transmitted packets).

The required timestamp for the egress packet can be either polled or the synchronization cell is configured to raise an IRQ as soon as the timestamp has been written to the FIFO. Since the delay between the issued transmission process and the storage of the timestamp is in the range of microseconds, polling in this case is normally not a significant waste of resources, especially if the program is rescheduled (typical granularity of $1 - 10\,\mathrm{ms}$). In any case, the application is then able to read out the timestamp from the FIFO of the synchronization cell.

In contrast to the ingress operation, this two-step process cannot be avoided, due to the fact that the timestamp is only available after the complete transmission of the frame. PTP allows this behaviour with the specification of two-step clocks. The only solution that avoids this issue is to perform on-the-fly timestamping in hardware, meaning that the timestamp is inserted into the frame during the transmission process to the medium. As already mentioned in previous chapters, this implies other drawbacks like limited protocol support.

Socket Control Message Flow

Until Linux kernel version 2.6.21 packets were timestamped with microsecond precision. The kernel's internal data structure for network frames contained a field, which allowed the networking subsystem to store the actual time of reception or transmission. Due to the fact that the data structure, *socket buffer*, is directly linked to the packet data and contains information about each individual frame that is handled, timestamps are available for all packets in the operating system. Nevertheless, the timestamping by the networking subsystem of the kernel only offers the possibility to use the software clock of the operating system as a reference.

The release of the Linux kernel version 2.6.22 introduced a new internal time format capable of handling nanosecond resolution. About the same time, also the kernel internal clock source API was introduced, which allows the kernel to measure time intervals with high resolution. This functionality is realised by evaluating a high-resolution counter, e. g. the processor cycle counter, in addition to the course (regular) system time updates trigger by an IRQ. The higher resolution of course allows to better control the software clock using an appropriate algorithm, but still does not solve the issue of jitter introduced by scheduling an IRQ processing. Since hardware timestamps cannot be directly linked to network frames, high-precision clock synchronization still has to be implemented using out-of-band timestamp transfer from the CSC. Still, pure software approaches like the reference implementation of NTP do benefit from the better resolution of the kernel timestamps.

Starting with version 2.6.30 of the Linux kernel an additional structure to the socket buffer was introduced [OLS08]. The additional fields can hold a system timestamp (drawn from the software clock), a raw hardware timestamp, and a transformed version of the previous. While the second is just a value reported by the NIC and has no relation to the actual system time, the latter is transformed so that it corresponds as good as possible to the system time. It has to be noted that the correlation is not perfect and as a consequence the sorting of packets by their transformed hardware timestamp received via different network interfaces might differ from the actual order. Even on a single network card, the progress of the transformed timestamp might be non-monotonic, due to step-like corrections of the system clock.

The possibility to store the timestamp directly attached to the kernel internal structure for handling network frames offers a way to timestamp all ingress and egress packets on a network interface. To do so, modifications of the network card, especially the MAC, are required to ensure that timestamps can be seamlessly associated with the corresponding network frame through all layers of the network stack. Actually, the adaptations result in a simplification of the hardware structure, in particular of the CSC and the MIIS.

In order to directly associate timestamps with the frame they were generated for, the modified implementation adds additional 64 bits to the buffer descriptor of the media access controller as illustrated in figure 6.7. This extends the status information about every ingress and egress frame which can be read by the device driver in the course of the normal network receive or send operation. There is no need to perform a matching of timestamps from the FIFO of the CSC to the respective packet. Since the timestamps are stored together with the status in the MAC the FIFO can be completely removed. In fact also the MIIS is not required any more. This is due to the fact that every packet can be timestamped (as no filtering is required). Further, the functionality of detecting the SFD is already implemented in the MAC and can therefore be reused to generate the necessary timestamping signals to sample the core time of the CSC. As soon as the packet is completely received, it is (in the previously described way) transferred to the main memory and subsequently the buffer descriptor is updated with the status and length information together with the timestamp, which is made available by the CSC.

The device driver receives the interrupt for the packet, reads the buffer descriptor together with the receive time information and populates a newly generated socket buffer that contains the control and timestamp information required to pass the frame on to the network subsystem. Frames can then be received by any application using the *recvmsg* API call to the kernel. The timestamp itself is contained in the returned message structure in an attached Socket Control Message (SCM). Thus, only a single operation is required to receive the network packet itself and the corresponding timestamp. Additionally, the timestamp is available for all packets on demand and it always automatically corresponds to the network frame. Packet loss or multiple applications receiving from the same interface do not pose a problem.

Since the send operation is non-blocking – it does not wait for the actual transmission of the frame over the medium – the timestamp is not available at the time the function

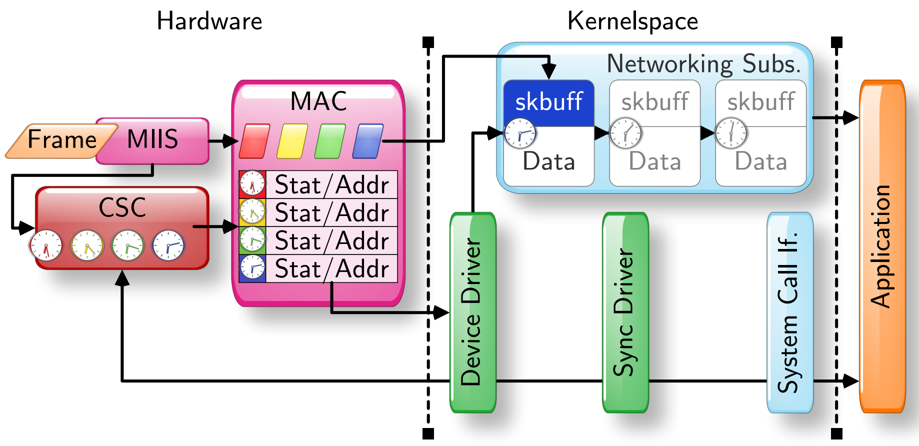


Figure 6.7: Network operation and data flow for in-band timestamp flow through extended operating system support

call by the application finishes. Consequently, egress time values have to be reported back to the application in a second step. After the application has handed over the data to the network subsystem, the driver is informed via a socket buffer about the send request. As in the normal operation flow, it copies the data to a DMA area and prepares a buffer descriptor in the MAC. The latter waits for the medium to be free and starts transmitting. On the SFD, the current time is sampled by an appropriate signal to the CSC. This time is then written together with the status information to the buffer descriptor. The driver is informed about the process using an IRQ. For the interface statistics and the read-out of the timestamp the device driver evaluates the buffer descriptor and reports back a reference to the original socket buffer plus the timestamp to the network subsystem.

The application can then subsequently read the looped-back egress message with the attached transmit timestamp from the error message queue of the socket, specially marked as “no error” and “timestamp”. For fragmented packets only the timestamp for the first fragment is reported and fed-back.

The seamless attachment of the timestamp to the network frame through all layers brings great benefits, especially for the receive operation. The association to the network frame is implicitly ensured – there is no need for identification tags generated using hash functions. This also avoids the problem of unassigned timestamps in the queue due to frames that were dropped at higher network layers. Furthermore, the direct connection between the timing information and the network packet always allows the operating system to assign them as a tuple to the socket opened by a specific application. Multiple user programs can therefore operate on the same network device without having to deal with frame and identical timestamp ordering in the MAC and CSC. As all frames can be timestamped, there is no need for large FIFOs in the synchronization cell. The transmit operation still has to use the two-step operation due to the non-blocking character and nature of the media access. Therefore, the advantage over the out-of-band signalling is only the standardised interface for the application. Since operating system functions are available, the actual implementation of the CSC is hidden to the application, except for the detailed configuration of special functionality, e. g. triggers. As for most hardware abstractions, the advantage lies in the interchangeability of compatible network cards and in the fact that the timestamps are available to any application, e. g. network packet analysers that supports the corresponding API of the operating system.

One issue that occurs due to the current realisation is the resolution of the timestamps. Current kernel implementations use 32 bits for seconds and nanoseconds (or a unified 64 bit field on supporting architectures) as the internal time representation. For protocols like NTP version 4 or PTP version 2, this resolution is not sufficient to fully take advantage of the potential of these protocols. Still, for common standard oscillators, commercial (non-dedicated) network equipment, or a synchronization precision in the nanosecond range (< 10 ns), the supplied resolution is sufficient. Only for higher precision (below one nanosecond), the read-out of additional digits is necessary to populate additional fields in the synchronization protocol frames.

6.2.2 NTP Daemon Reference Clock Driver

A reference clock driver serves the function of querying a master clock in order to update the locally managed time of the NTP daemon. In other words, the network packet operation is replaced by individual (device specific) calls in order to get the offset between the locally kept clock and the reference. In case of the present implementation it simply means that there has to be a close timely relation between the read-out of the local clock and the core time of the CSC.

The present driver was built for ntpd version 4.2.x and implements the *clock_start*, *clock_shutdown*, and for the actual functionality, the *clock_poll* function. The first two are required in order to prepare the necessary file descriptors and open the interface to the kernel level drivers. The latter is used to sample the current CSC time from time register and immediately read the system time. The sampled value is then converted to the ntpd internal time format and passed on to the daemon.

Contrary to the GPS implementation, the driver is not triggered by an external event, but the NTP daemon itself triggers the comparison. Therefore, also the interval between the read outs can be determined by the configuration of the software and even be changed during runtime according to the accuracy requirements.

The reason for the development of the reference clock driver is the very simple interface and the powerful possibilities that the NTP daemon and the surrounding support tools offer. With this integration into a well known clock synchronization protocol stack, which also serves sophisticated clock selection and steering procedures, several tasks can be easily fulfilled.

- The probably most obvious use case for the reference clock driver is that the PTP serves as an upstream (stratum 0) clock for NTP. The common assumption is that IEEE 1588 – since it is primarily intended for the use in LAN and hardware assisted timestamping is well introduced – has a higher precision than NTP. The latter, on the other hand, is widely used in nearly any sort of device and requires less network resources (for a small number of client nodes) due to much lower polling intervals. Consequently, it makes sense to have a “bridge” between the two protocol worlds.

- The NTP daemon features a number of sophisticated methods for precise clock selection, clock steering, and input filtering (see chapter 4 of [Mil06a]). Therefore, it is very beneficial to provide an accurate timebase to the daemon and let it control the clock. This way the distribution of the time via PTP can make use of the very profound know-how, which comes from years of development in clock steering in NTP. Also, this is an easy way to bind the system timebase, which is kept in software, to the time of the CSC.

- Besides the well developed synchronization algorithm, the NTP reference implementation [ntp09] also provides a number of support tools to monitor and analyse the behaviour of the clock steering mechanism. Utilities like *ntpdc* can provide information about the kernel phase-lock loop operating parameters, loop filter variables, and a number of further statistic parameters useful to determine the synchronization quality, as well as the one of upstream clocks. In combination with the present reference driver, also the performance of the PTP can be evaluated with well known means from the community, therefore making the measurement values comparable to literature.

- For precise comparison, e. g. of PTP to NTP, there is the need for high similarity between the evaluated systems. Consequently, if the performance of the two protocols is to be compared, e. g. over long distance networks, it is recommended to use identical filtering and clock steering algorithms. This allows to circumvent the influence of different control loops on the measurement results. The tight integration of the IEEE 1588 enabled NIC allows to judge the network performance of two independent protocols with common means.

In general it has to be mentioned that the NTP reference daemon is not built to easily add new reference clock drivers. It has a defined API but it is sparsely documented and may change from one version to the next. Therefore, it is unavoidable to have deep understanding of the source code if not a very similar driver to one of the existing ones is required.

Although the NTP daemon runs as a user space application it has great dependencies on the Linux kernel, e. g. the way the system clock can be influenced. Further, the system internal timekeeping is not continuous. Therefore, the daemon implements special measures to interpolate between course clock readings. Jitter that originates from e. g., scheduling is filtered through outlier detection.

A more accurate solution is to directly use the CSC time as the system timebase. The Linux kernel uses any known and available hardware counter to generate IRQs in a regular interval to update and steer the system time in a feed-forward manner. The timebase for the whole system, which is relevant for all programs, is therefore kept in software. In case a synchronization protocol controls the CSC the accuracy of the system timebase can be enhanced by directly generating the required IRQs from there. This avoids the influence of a second oscillator on the system time, but requires the low-level driver to support the Linux kernel *clocksource* API as well.

6.2.3 PTP Stack

The IEEE 1588 protocol stack in the user space is the central instance that controls all synchronization activity in a system. It is responsible for appropriate message exchange with other nodes in the network, defining the correct hierarchy, as well as the evaluation of the gathered timestamps in order to steer the clock. For the present thesis, PTP was chosen, because it is a widely accepted standard that provides the necessary support for all type of devices (i. e., nodes and switches) in a LAN environment in order to perform high-accuracy clock synchronization. As mentioned in section 3.4 only a few implementations for NTP exist which mostly require special system architectures and setups. In any case, the device has to be designed for very close hardware and software interaction on a common timebase in order to benefit from hardware timestamping.

According to the protocol standard, which was discussed in chapter 3.2.2, the network stack is partitioned into various function blocks, which are required for the different node types, e. g. master, slave, boundary, or transparent clock. While some, like the clock steering algorithm are instantiated only once, other parts can be reused for different ports in case the stack runs on a node with multiple network interfaces. For this work only single port configurations were considered, since extensions to more ports are just a generalisation and do not have any influence on the synchronization quality compared to the basic problem.

Besides the reference implementation of the protocol stack, the application has to cooperate with the enhanced hardware of the network card in order to gather the required timestamps. Till the extension of the Linux kernel for hardware timestamping support (see chapter 6.2.1), out-of-band timestamp acquisition was mandatory. If an appropriate kernel version is used the original approach is still valid if timestamps with sub-nanosecond resolution are required.

During the start-up phase of the protocol stack, the clock steering mechanism has to work different from the general operation phase. At the master side, the stack commonly acquires the current system time and applies it to the timebase of the network card. This operation can be disabled in order to allow different synchronization mechanisms, e. g. GPS stack, to supply the reference time. Contrary, the slave side sets the time after the first synchronization message, in case the reference value differs significantly (e. g., more than one second) from the current value of the CSC. This way it is ensured that the control loop can reach the desired precision within a reasonable amount of time, even if the maximum rate for clock steering is limited (e. g., $\pm 10\,\%$).

Triggers are a limited resource in the implementation of the CSC. The PTP application requires at least one periodic trigger event in order to schedule the transmission of a Sync message and optionally a second one for the Delay_Req messages. Since the protocol stack is not the primary target for the use of synchronization cell resources, the stack should save them for user applications that depend on the provision of a correct timebase. To do so, the stack can also use the timers/triggers of the operating system. All operations of the protocol itself are not required to be precisely timed. In the case of Delay_Req message, the concurrent transmission by multiple nodes can even cause congestion in the network or the master node. To avoid negative effects due to message collisions on the network, the PTP itself uses a mechanism to randomly distribute the transmission interval within a certain range underlining the argument that no exact time source for issuing of these messages is required.

Besides the trigger functionality, also another feature of the synchronization cell can be used to enhance the operation of the PTP stack. The *applytime* feature allows to activate a previously defined set of parameters at an exact point in time with hardware support. The stack can benefit from this operation by setting the freshly calculated step-size at fixed supporting points in time. This allows a constant spacing between individual adjustments of the clock, easing the operation of the control loop.

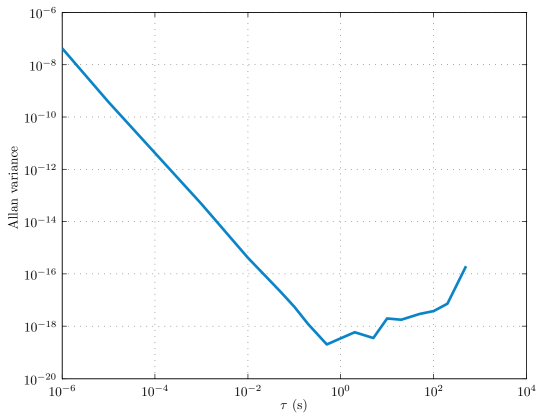
The most important operation that has to be performed in addition to the ones specified in the standard by the PTP stack is the matching of a timestamp to the packet it is required for. On reception of a packet via the event interface, the stack queries the receive timestamp FIFO of the CSC to acquire the matching data set. For the Precision Time Protocol the CSC stores the packet sequenceId and the sourceUuid together with the actual timestamp. The first two fields are then used by the stack to identify the matching timestamp by the extracted information from the frame data. Since previous packets could have been dropped by the intermediate network layers, the read out continues till the FIFO is reported to be empty or the correct timestamp has been found.

A crucial parameter that is influenced by the configuration of the protocol stack is the message exchange rate and hence the synchronization interval, T_{sI}, which defines the supporting points for the controller steering the clock. The quality of the source oscillator driving the CSC stands in close relation to the number of required timestamps per time interval. For a given node accuracy, a *hold-over* time can be defined that gives the maximum period between two consecutive corrections of the clock in order to keep the timebase within the given boundaries. Obviously, if the oscillator quality is higher also the synchronization interval can be chosen to be longer. In the following, the influence of the two influencing parameters on the overall accuracy achievable by the synchronization stack is discussed.

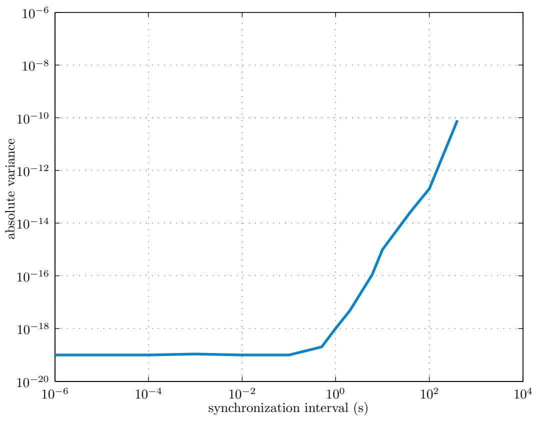
Synchronization Interval

The most interesting and easiest changeable parameter in a synchronization system is the equidistant interval T_{sI} for exchanging synchronization messages. For further investigation of the influence, it is assumed that both, the master and the slave, are equipped with an oscillator that has a drift $D(t)$. The presence of varying drift of the oscillators will increase the offset between the nodes with the progression of time. If it is assumed that two oscillators were perfectly synchronous at the begin of a synchronization interval, the offset α after T_{sI} is $\alpha = \int_{T_{\mathrm{sI}}}(D_{\mathrm{M}}(t) + D_{\mathrm{S}}(t))\mathrm{d}t$. [LENG08]

If the Allan variance of the oscillator is known, the standard variance of α can be estimated. The two-sample Allan variance can be interpreted as a variance of the



(a) Measured Allan variance of a typical crystal oscillator over the sampling interval



(b) Absolute variance σ^2_{abs} of the oscillator over the sampling/synchronization interval

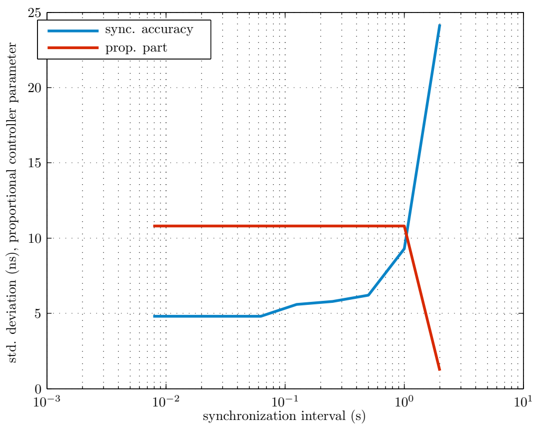
Figure 6.8: Clock source characteristics influencing the selection of the ideal synchronization interval.

frequency change rate. Therefore, the frequency variance is the integral of the Allan variance over T_{sI} plus a frequency offset, the standard variance σ^2_D for the drift. Another integration results in the time variance plus a time offset. If we assume that the frequency and time offsets are compensated in the synchronization system's servo, the variance of the time error is just the denormalised Allan variance (the normalized Allan variance multiplied by T^2_{sI}). Since the master and the slave oscillator are statistically independent, the resulting variance of α is the sum of the standard variances of both nodes.

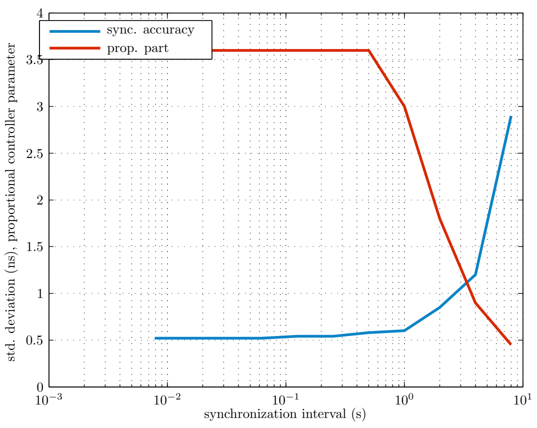
For a typical oscillator with an Allan variance as shown in figure 6.8(a), the absolute variance for short intervals remains constant (as depicted in figure 6.8(b)). Thus, just from the oscillator's point of view one gains no additional accuracy if the system synchronizes more often. For long synchronization intervals the standard deviation rises significantly, which is mainly due to long term effects like temperature changes. Consequently, the synchronization period should be selected as the maximum value that is available before the flicker floor of the Allan deviation is reached. This is equal to the end of the zero slope part of the absolute variance at about 0.2 s. From the oscillator's point of view there is no difference on which point of the zero slope part the synchronization period is chosen, which means that there is no further accuracy gain by exchanging more timing messages than necessary.

Obviously, long synchronization intervals are not recommended, because the variance of the sampled oscillator values significantly increases. The control loop of the synchronization stack has therefore the problem of unpredictable behaviour of the clock source. Precise corrections are hard to take which is due to the statistical values caused by the oscillator noise.

PTP version 1 [IEE02] has a default synchronization interval of 2 s and allows synchronization intervals between one and 16 s, incrementing by a factor of two. As pointed out



(a) Performance with a standard oscillator.



(b) Performance using an OCXO.

Figure 6.9: Control loop performance and the respective proportional parameter for the controller of the PTP stack in dependence of different synchronization intervals.

by figure 6.8, even the shortest possible period, according to the IEEE 1588 standard, of one second is rendered non-optimal for the tested oscillator resulting in a degradation of accuracy.

A closer look at the basic data for the diagrams reveals that a synchronization interval of 0.5 s or even better 0.2 s can significantly reduce the jitter introduced by the clock source sampling. If PTP version 2 [IEE08b] is used in order to allow for the mentioned values of T_{sI}, the control loop is supplied with a higher quality of timestamps in slightly shorter intervals, while the network load is still within a reasonable range of about five packets per second.

Oscillator Quality

Besides the general behaviour of oscillators in dependence of the sampling interval, also the overall quality has significant influence on the achievable synchronization accuracy. Figure 6.9 depicts the observed standard deviation of a point-to-point synchronization system with two nodes, both equipped with a 50 ppm XO and 0.3 ppm OCXO, respectively. The measured deviation of the two node clocks is plotted over different synchronization intervals. In parallel, the appropriate proportional factors for the proportional-integral controller adjusting the ABC's rate are shown as well.

For short synchronization intervals the control loop parameters can be set rather aggressive to compensate quickly for any change of the oscillator's frequency, but for long intervals (more than 1 s) such servo parameters lead to oscillations and instability of the control loop. These oscillation are partially caused by the large absolute offset, which can occur due to the drift during the long synchronization interval and the inability of a PI-controller to accurately predict the future behaviour of the oscillator. The combination of the relaxed control loop parameters and the high time error variance of the oscillator result in an increased standard deviation.

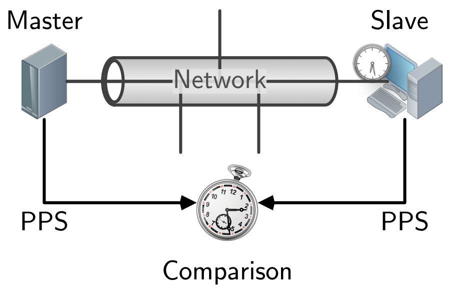
The actual measurements using a standard oscillator shown in figure 6.9(a) give the results expected from the remarks of the last section. Beginning at a synchronization interval of about 0.5 s the performance of the system significantly decreases. At a message exchange rate of 2 s, the overall accuracy already exceeds the single digit nanosecond range. Contrary, the quality of the OCXO allows the system to keep the standard deviation below 3 ns even at a message exchange rate of only eight packets per second. In addition to the generally better jitter behaviour of the oven-controlled oscillator, which results in a lower and broader noise floor than the one of the standard oscillator (see figure 6.8(a)), the OCXO is also more immune against external influences, especially temperature changes. Considering a typical thermal specification for uncompensated crystals of 1 ppm per Kelvin the influence can significantly effect the synchronization performance. A 1 K temperature rise over a synchronization interval of 2 s produces an error in the order of 2 µs. [IEE08b]

Extension for Master Group Implementation

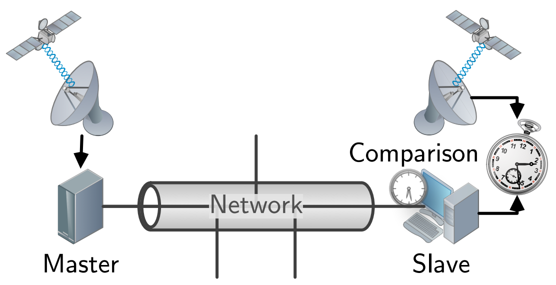
In order to enhance the fault tolerance of the protocol defined by the IEEE 1588 standard, a compatible extension with a group of masters has been described in [GLS10]. A group of PTP masters sends Sync messages as multicast to all members of the group. Therefore, a common notion of time with a fault-tolerant averaging algorithm can be calculated. The failure of one master cannot influence this ensemble time any more. A so-called *Master Group Speaker* – also member of the group – then communicates this time to the unmodified PTP nodes, which see the ensemble like a single master. In order to support the exchange of multiple Sync messages per synchronization period or to allow the implementation of the Master Group speaker, the PTP stack had to be slightly modified as described in the following.

As every member of the Master Group receives the Sync messages of all other members, the FIFO size of the CSC has to be able to cope with that amount of messages. The messages themselves can be treated in the same way as in the unique master mode, since the receive sequence always equals the one of the timestamps in the FIFO. The Sync messages are processed in the same way as in a normal slave node with the exception that a Master Group member always waits for the reception of all associated group members before the synchronization algorithm is called. In general the modification is therefore quite moderate.

In order to additionally support the functionality of the Master Group speaker, the FIFO handling functions of the PTP stack were modified. Instead of searching for the correct timestamp and dropping all non-matching successors, the latter are transferred to a temporary software buffer. On each call of the procedure, it is first checked, whether there is a matching timestamp available in the temporary FIFO and only in case of a miss, the hardware is queried. Since access is very fast (host system), on each query, the whole temporary buffer is scanned and entries just age out. That way it can be ensured that even without dedicated timestamp delivery – as described in chapter 6.2.1



(a) Evaluation setup for a low range network using two 1 PPS signals



(b) Performance evaluation setup for a wide range network using GPS

Figure 6.10: Evaluation of the accuracy of a synchronization system

on operating system support – the stack can handle the operation of a boundary clock using two different multicast domains on a single interface [Gad08].

6.2.4 GPS Stack

One important piece of software in order to evaluate the performance of synchronization protocols and possible dedicated hardware support is a software to relate the internal clock synchronization of a network to GPS. There are two main reasons, why someone wants to perform such a task. The most common matter is the introduction of a global timebase to the network by binding the master to the GPS time. That way, the timescale gets globally comparable. For example, NTP stratum 1 servers are usually bound to GPS or other atomic clocks. Another reason comes from the need to verify the precision of an installed clock synchronization system which spans greater distances. Figure 6.10 shows the principle of the measurement setup. In this case it is often no more feasible or applicable to draw dedicated lines (illustrated in figure 6.10(a)), transporting e. g. 1 PPS signals, from at least one node to the master in order to measure the synchronization offset. The solution therefore can be to compare both nodes (or the master) to a known reference which they have in common. The GPS is such a system, which is globally available and can be used as a common timebase in order to evaluate the performance of a clock synchronization system over greater distances. Either both nodes under test are compared to the global time or one (the master) is synchronized to GPS and the slave clock is then measured against the common reference as shown in figure 6.10(b).

Commercially available GPS receivers can provide a 1 PPS signals with an accuracy of $\sigma_{\text{GPS}} = 2 - 150\,\text{ns}$ [jt.08, Vv09, Tri01, Mei08]. The final accuracy depends on the correction information provided by the receiver. Jitter not only originates from decoding and calculation issues of the GPS signal, but also from the resolution of the 1 PPS, which can be corrected by information about the e. g. quantisation error. Many GPS receivers provide additional information about the 1 PPS signal via a specific protocol, e. g. the standardised NMEA protocol or Trimble Standard Interface Protocol (TSIP). That way, the absolute time and optional correction information (quantisation error or accuracy values) is transferred to the attached node.

There is also a second method, which allows the transfer of highly accurate timing information from the GPS receiver to the attached node. This technique uses a special signal – commonly the Request to Send (RTS) signal on the serial RS 232 [eia69] port – that is sampled by the GPS receiver. The device then responds with the absolute time, the rising edge was detected. That way, the answer can be compared to the timestamp which was drawn at the time the signal edge was generated. Since this method makes use of standard serial port interfaces, it is easier to implement, but suffers from software jitter. Due to the fact that the signal edge is issued by a software command, the exact execution time is not known. Even if the previous or next command is the drawing of a timestamp, there might be a preemption of the current task by the operating system. The latter causes unknown delay between the operation of drawing the timestamp and the issuing of the signal edge. The approach is therefore more generic, but might suffer from reduced precision.

Actually, also the comparison shown in figure 6.10(b) can be done within the node, using a timestamping network interface with additional event input. This has the advantage that only one processor is required to process the different software stacks involved. In fact, the network interface hosting the CSC – therefore the timebase – is used to timestamp the 1 PPS signal as well. When the timing information about the signal (absolute time and optional correction information) is received from GPS receiver, the calculated timestamp is compared to the one drawn by the card. Assuming that the master transmitted the correct global time, the comparison of the mentioned timestamps at the slave delivers the offset not corrected by the network synchronization protocol.

7 Validation

The performance of a clock synchronizing system depends on several individual components, i. e. implementation of the network layers, especially their jitter behaviour. A number of hardware and software components have to smoothly interact with each other in order to allow highly accurate clock synchronization over Ethernet networks. Besides the performance of each layer of the network stack (i. e., precision, resolution, stability, etc.) also the cooperation between the sub-modules is an important factor, e. g. the adaptation of the synchronization interval on the stability of the oscillator. Additionally, various external factors – like temperature, network load, or processing resources – influence the overall accuracy of the overall system. Although it is possible to analyse, evaluate, and simulate a number of individual parameters and the dependence of the performance on them, it is extremely difficult to predict the overall behaviour of a system that spans multiple network layers and is sensitive to many (external) factors.

Nevertheless, the simulation of clock steering algorithms and protocol specific issues makes sense, if large networks, i. e. high number of nodes, e. g. several hundred, are considered. Simulation is always preferable as soon as economic reasons hinder the implementation of a (test) network. For such cases, DES systems offer a suitable environment to evaluate the behaviour of such structures. The second part (section 7.2) of this chapter gives insights on special measures that have to be taken in order to gain more realistic models of the behaviour of the hardware.

7.1 Hardware Measurements

Due to the above mentioned issues concerning predictability of a complete synchronization system, a practical validation using prototypes is essential. The latter are used to verify the performance and effectiveness of the enhancements described in the previous chapters. The following sections describe the chosen validation methods and the gained results from the prototype or focused test implementations. Besides the evaluation of the complete system also the validity of individual measures, e. g. enhanced timestamp precision, to increase the performance are described.

7.1.1 Applicability of Phase Estimation to Enhance Precision

The precision of the timestamps drawn by the CSC is one of the limiting factors in a clock synchronization system. Phase estimation methods as explained in section 5.4.2 allow to significantly increase the precision while keeping the necessary processing frequency relatively low. In order to demonstrate the effectiveness of the mentioned techniques an FPGA implementation of the phase/frequency estimation method using an Altera Stratix II GX evaluation was developed [EL09].

Since the analysis should demonstrate the capabilities of the chosen method, both clocks are generated via two PLLs from a single 100 MHz oscillator. Obviously, in real applications the communication clock and the local clock do not have a fixed frequency relationship (at least for the receive side). Nevertheless, this has to be done in the test setup to study the maximal achievable performance without disturbing effects caused by oscillator drift. To simulate timestamping for a typical 100 Base-TX connection using MII to connect the PHY and the MAC, a communication clock of $1/T_\mathrm{c} = 25\,\mathrm{MHz}$ was chosen. The local clock $1/T_\mathrm{l}$ was selected to be 50.2272727 MHz, i. e. with an oversampling factor of $n = 2$ and $\beta = 0.00452489$ and $\epsilon = 0.00\dot{4}\dot{5}$ as given by the equations 5.4 to 5.7.

The hardware implementation for the test setup uses an ABC with fixed increment, featuring an 8 bit sub-nanosecond resolution as the time source for the timestamper. The latter generates 64 bit timestamps with enhanced precision using the phase estimation logic that are transferred to an analysis software. For evaluation the calculated ideal time is subtracted from the measured timestamps. Due to the long measurement time, also the very slight drift of the ABC caused by the limited granularity of the ABC's increment is compensated by the reporting software.

In order to generate a known scheme for the timestamp signal, $\mathcal{S}_\mathrm{TS}$, the signal is asserted every 250^th clock cycle (equals 10 µs, arbitrarily chosen) for a single period. $\mathcal{S}_\mathrm{TS}$ subsequently triggers the timestamper at a mathematically determinable instant of time to be compared to. Using such an ideal timestamp trigger source, the timestamps are expected to be equally distributed from $-T_\mathrm{l}/2$ to $+T_\mathrm{l}/2$, in case simple sampling of the signal (without taking the relative phase offset, δ_TS, into account) is utilised.

The measurements show that the implementation in fact exactly matches the expected equal distribution, obviously for the sampling implementation, but also for the modified method which additionally includes the phase offset. If δ_TS is taken into account, the standard deviation of the timestamps improves significantly from 5.761 ns to 26.029 ps. As shown in figure 7.1, which is derived from about 2.927 Msamples the timestamp error is almost equally distributed with slight imperfections around the centre.

The theoretical resolution for the given parameters is $\beta\,T_\mathrm{l} = 90.088\,\mathrm{ps}$ and therefore $\sigma_\mathrm{TS} = 26.006\,\mathrm{ps}$. Given the (cheap) layout of PLLs in FPGA devices, it is surprising that these PLLs can produce the required low-jitter clocks which are required to enable such accurate measurements. In fact this result was only possible with one out of four tested FPGAs, whereas the others generated timestamps with $\sigma_\mathrm{TS} = 52\,\mathrm{ps}$ instead of 26 ps. The higher standard deviation was caused by some timestamps about 130 ps away from the main equal distribution. The reason for this can be found in physical effects mainly the phase noise of the clock source. Random clock jitter causes cycle slips to be detected too early or too late which in consequence shifts the timestamps as well. In such cases, clocks with better stability are just one possible solution, but also combining phase/frequency estimation with multiple timestamps per frame can be considered to solve the issue.

As done in the described experiment, using the same clock source for $\mathcal{C}_\mathrm{l}$ and $\mathcal{C}_\mathrm{c}$ verifies the operability and accuracy of the timestamping scheme. Nevertheless, for practical

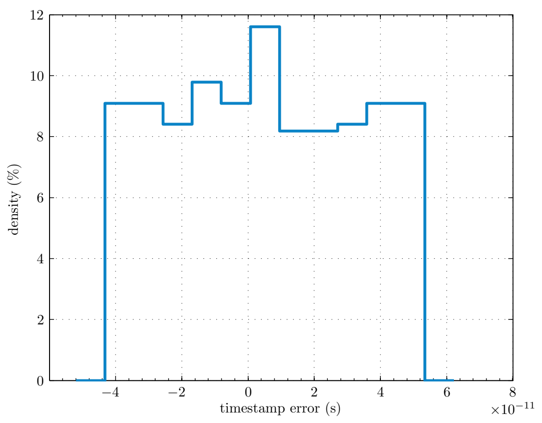


Figure 7.1: Histogram of the resulting timestamping error measured for the combined phase/frequency estimation method

applications the two clocks are normally sourced from different oscillators – for the ingress path this is always the case, for egress a fixed relation is possible. As the cycle slip detection relies on a single clock edge, a clock source with low phase noise is required. Since the clock output of Ethernet PHYs is just designed to drive digital logic, the quality of the clock outputs is limited (as measured for the SMSC LAN8700 PHY) and it might be required to feed the communication clocks of the PHY through a PLL to stabilise them. Another option is to estimate the cycle slip instant over several periods if the clock stability over multiple cycle slip periods is guaranteed.

If a highly stable reference oscillator is used as the clock source this accurate timestamping method can also provide data for stability measurements. If the timestamper is applied to an external input connected to a second oscillator (e. g., system or bus clock source) the frequency stability of this source can be measured and characteristic parameters like the Allan variance calculated. In order to demonstrate the feasibility of the method, a standard 100 MHz oscillator with an advertised stability of 50 ppm was tested against a 10 MHz Rubidium atomic clock [SRS05] which features an Allan variance of $\sigma^2_{\mathrm{Rb}} < 2 \cdot 10^{-22}$ @ $\tau = 1\,\mathrm{s}$. The result of the measurement is shown in figure 7.2.

In clock synchronizing networks such measurements can be used for oscillator classification in the nodes. Obviously, this can only be performed, if all of the nodes can be supplied with a common reference. In switched Ethernet networks this prerequisite can be fulfilled in most cases for multiple nodes that are directly connected to a single switch. Typically the latter drives all its PHYs by the same transmit clock and therefore the receiving PHYs in the nodes are able to recover the common switch's clock from the received data. The topology then allows all these nodes to use the incoming clock as the reference for stability measurement. Thus, all nodes can measure their frequency departure with respect to the switch's clock. The acquired data can then be utilised to calculate the frequency departure of every node with respect to the switch and in further consequence between the group of nodes. This information can be useful for the master election process, for a dynamic adjustment of the control servo parameters or just for the

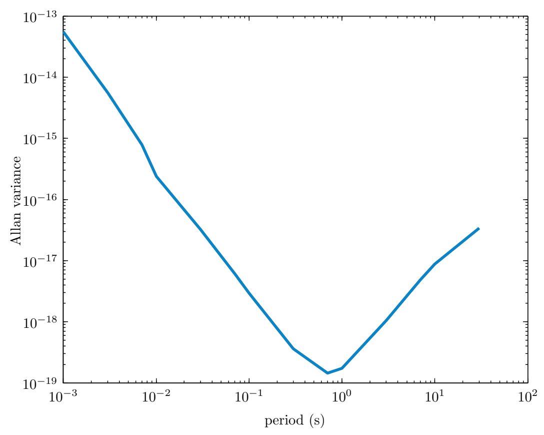


Figure 7.2: Measured Allan variance of a 100 MHz oscillator using a Rubidium atomic clock as the reference.

detection of faulty oscillators.

As illustrated in figure 6.9 the oscillator quality has major influence on the achievable synchronization precision. Given the availability of timestamp with a granularity in the picosecond range, the question may arise about the possible gain over traditional timestamping approaches with granularities in the range of about 10 ns. As mentioned in section 6.2.3 and discussed in [LENG08] it mainly depends on the oscillator and a properly chosen synchronization interval. Summing up the findings it can be said that the higher the stability of the oscillator and the longer the synchronization interval the more beneficial a very accurate timestamping method can render.

For cheap oscillators, which cause the clock in the slave node to drift away between the synchronization intervals in the order of several nanoseconds, a highly accurate timestamping method will result only in a marginal gain. Significant enhancements can be seen for highly stable oscillators or even for synchronous systems that use the described method to perform link delay measurements. The combination of a synchronization protocol together with Synchronous Ethernet allows all nodes to use the same frequency which simplifies the synchronization. Despite that, such a system still has to measure the transmission delay between master and slave. Using the proposed timestamping method this delay can be measured very precisely enabling synchronous systems to synchronize virtually offset-free given that the physical connection is symmetric in the case of round-trip measurements.

7.1.2 Synchronization Performance

To show the influence of the timestamp granularity on the overall system accuracy a test setup with variable timestamp granularity was designed. It is capable of drawing timestamps with 32 ns down to 2 ns and generating the 1 PPS signal with 1 ns granularity. The latter is important because the output precision also adds to the measurable deviation of the synchronization performance.

In order to be able to identify the influence of the timestamp granularity, a stable oscillator is required. Therefore, on both involved systems (master and slave), an OCXO (0.3 ppm) was used as the clock source for the CSC. All results concerning the overall system accuracy were obtained by monitoring the 1 PPS pulse on an oscilloscope with 20 GS/s. For the synchronization interval T_{sI} various values ranging from 7.8 ms up to 8 s were used. [LENG08]

Theoretical Boundary for the Experiment

Considering the jitter sources that cannot be cancelled out by the hardware support using timestamping on the MII, four remaining influences have to be taken into account. On both systems, the variance introduced by the oscillator, the PHY, the timestamping itself, and the output granularity are relevant. If it is assumed that all mentioned jitter sources are statically independent, the resulting variance of the jitter is the sum of all the individual variances. Consequently, the absolute (observed) variance that can be measured on the oscilloscope (not taking the measurement system with 50 ps granularity ($\sigma_{\mathrm{m}} = 14\,\mathrm{ps}$) into account, can be calculated as

$$\sigma^2 = 2\left[\sigma^2_{1\,\mathrm{PPS}} + \sigma^2_{\mathrm{osc}}(T_{\mathrm{sI}}) + \kappa(T_{\mathrm{sI}})(\sigma^2_{\mathrm{TS}} + \sigma^2_{\mathrm{PHY}})\right]. \tag{7.1}$$

Due to the 2 ns resolution of the timestamps the corresponding factor has a value of $\sigma_{\mathrm{TS}} = 577\,\mathrm{ps}$, whereas the 1 PPS signal adds $\sigma_{1\,\mathrm{PPS}} = 289\,\mathrm{ps}$. Both values are calculated under the assumption that the actual occurrence of the event is equally distributed over the interval between the two instances in time to generate/detect the event, which allows to use $\sigma^2_{\mathrm{x}} = T^2_{\mathrm{clock}}/12$.

For the SMSC LAN8700 physical layer device [SMS09] used in the experimental setup a standard deviation of $\sigma_{\mathrm{PHY}} = 100\,\mathrm{ps}$ for the OCXO was taken as shown in figure 4.2(b) illustrating the distribution of the delay between two PHYs measured from the assertion of the transmit enable (TX_EN) to the receive data valid (RX_DV) signal of the MII. It has to be mentioned that the value of σ_{PHY} has been chosen as a combination of jitter sources introduced by the physical layer. These include the oscillator supplying the transmitting PHY on the transmitting side as well as the chip internal PLL and the loop filter on the receiver side. The factor two in front of the brackets of equation 7.1 therefore indicates that σ_{PHY} actually is the mean value of $\sigma_{\mathrm{PHY,Tx}}$ plus $\sigma_{\mathrm{PHY,Rx}}$. Since they evidently have to appear pairwise, they have been combined together to one value.

Not only the oscillator's variance is dependent on the observed synchronization interval T_{sI}, but also the last term representing the quality of the timestamps with a factor κ. The latter describes the implication of the control loop on the standard deviation. Since the control loop parameters must match the behaviour of the oscillator, which depends on the synchronization interval, κ itself is a function of T_{sI} as previously illustrated in figure 6.9.

As modern synchronization protocols, like the mentioned PTP, feature delay compensation, the mean value of the distribution is compensated by evaluating the round trip time. This is of high importance, since every time a link is (newly) established, the PHY's internal receive and transmit filter parameters may be recalculated, which leads

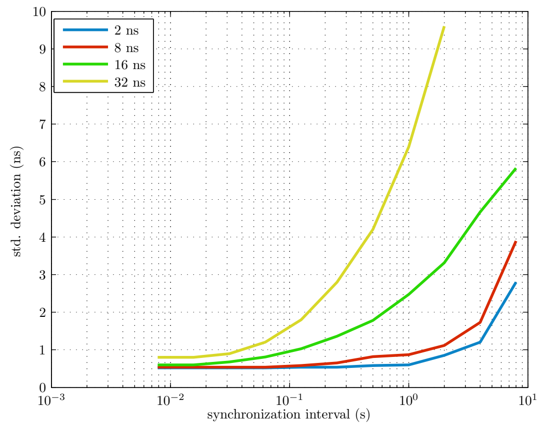


Figure 7.3: Measured standard deviation for identical synchronization system setups but different timestamp resolutions.

to changes in multiples of 8 ns in the absolute line delay due to the lock-in of the receiver PLL to the 125 MHz clock on the line for the case of 100 Base-T.

In order to achieve appropriate results, a frequency counter with 25 ps resolution [SRS06] supplied by a GPS disciplined Rubidium frequency standard [SRS05] was used. This ensures that the measurement setup features the required stability and precision to show the effects in the picosecond range.

Influence of the Timestamp Resolution

The granularity of the timestamps has a major influence on the accuracy, since it defines the resolution, with which a node is able to detect the point in time of the transmission or reception of a network packet. If the generation of the timestamps is neither correlated between sender and receiver nor to the local clock source of the ABC, then the absolute timestamp variance (relevant for the control loop and therefore also dependent on the synchronization interval) is $\sigma^2_{\text{abs}} = \sigma^2 \cdot T^2_{\text{sI}}$ using σ from equation 7.1.

For short synchronization intervals the granularity is of minor importance, since the absolute timestamp variance is low, because the control loop can average over several timestamps and therefore reduces the jitter by statistical means. Contrary, if highly stable oscillators or long synchronization intervals are used, low variance timestamps are getting increasingly important due to the relative higher effect on the system variance. This is due to the fact that the term $\kappa(T_{\text{sI}})(\sigma^2_{\text{TS}} + \sigma^2_{\text{PHY}})$ gets in the range of the variance of the oscillator $\sigma^2_{\text{osc}}(T_{\text{sI}})$. Thus, for long synchronization intervals the impact on the accuracy with different timestamp granularities becomes significant as pointed out by the measurements which are shown in figure 7.3.

The statistical independence required by equation 7.1 may not be satisfied in every case. For instance, if the master has a clock increment, which is a multiple of the timestamp granularity, the observed and transmitted timestamp is variance free (assuming that the PHY's send clock is supplied by the same clock source as the ABC). On the other hand, master and slave may be correlated, if both round up or down the timestamp at the same time due to an almost equal clock rate. The latter may happen in the steady state case and/or if the oscillators of master and slave show similar behaviour (quite likely for high-quality oscillators).

The performed measurements allow to select the most efficient method to achieve a certain time error variance, if required. There are several possible ways to reach a specified limit: a better oscillator, shorter synchronization interval, or a finer granularity of the timestamps produced. The obvious conclusion which can be drawn from the measurement results is that a good oscillator and a short synchronization interval are two universal options helping to improve the observed clock variance. However, fine granular timestamps can improve the variance for rather long synchronization intervals by a significant number, whereas for short intervals the observed deviations mainly origin from the oscillator noise itself. In the latter case, the improvements by enhanced timestamp granularity are not really noteworthy.

Summing up the above described measurement results, a really accurate mathematical model of the investigated problem is difficult to derive. This issue is mainly due to the behaviour of the oscillator and the analogue parts within the transmission chain (e. g., PHYs, PLLs). There are a lot of hidden internal parameters of the individual components, which are only known to the chip manufacturer. Thus, even for given system setup the only method is to build a behavioural model based on the observed measurement data.

For the chosen system the theoretical limit is $\sigma = 410\,\mathrm{ps}$, if $\kappa = 0$ and a perfect oscillator ($\sigma_{\mathrm{osc}} = 0$) are assumed. In other words, the measurable standard deviation solely origins from the granularity of the 1 PPS output pulses, while all other elements have the ideal characteristic. Compared to the lower boundary for the standard deviation, the minimal achievable value that could be measured in the above mentioned setup was 520 ps. This value was measured using an OCXO as the clock source for both nodes. A synchronization interval of $^1/_{128}\,\mathrm{s}$ is therefore very close to the theoretical boundary which assumes only ideal elements.

7.2 Simulation

A technique that is well suited for simulating clock synchronization systems is to use DES. Such environments are state-of-the-art tools for network simulation (for instance ns-2 [The09] or OMNeT++ [Var99]) or system and circuit design (like ModelSim® [Men09]). The approach is to have a timescale that is not divided into equal intervals, but into demand-driven virtually spaced sections where events are scheduled. The simulator assigns discrete points on a timescale to all events and processes them with the occurrence of this simulation time. This reduces the overall simulation effort while at the same time maintaining high precision.

The principles of simulating clock synchronization in computer networks are simple. All clocks situated on the distributed nodes must be steered in a way that a common notion of time is established with a given accuracy. The ABC, which increases the value of a time register with a freely configurable increment with every oscillator tick, has to be controlled appropriately to tie the local time value of the node to a given timescale.

One way to simulate such a system is to model the message exchange according to the synchronization protocol, the Adder-Based Clock, and the steering algorithm at the slave nodes. In a simple implementation all simulated nodes are driven by an oscillator with a modelled frequency according to a simplified version of equation 4.1:

$$f_n(t) = f_0 + \varepsilon_{f_{0,n}}. \tag{7.2}$$

The term describes the frequency of a node n with an ideal base frequency, f_0, and a frequency offset $\varepsilon_{f_{0,n}}$, which is different for every node. If such a model is used for simulating the performance of the simulation network it breaks down to controlling the offset between two (master and any slave) proportional integrating systems. This is a rather simple task for control loop design but cannot realistically model the real-world behaviour of clock synchronizing systems. Even if the equation is extended by adding an ageing term $a(t - t_0)$ and further an optional, relatively small (compared to f_0 and $\varepsilon_{f_{0,n}}$) random value to each sample, the resulting control loop deviation is almost completely determined by the latter and rather small compared to experiments.

For more realistic simulation results, the oscillator model must take into account stochastic frequency and phase variations. In addition, for DES systems, the model typically must provide the number of oscillator ticks elapsed from simulation start until any given arbitrary simulation time without actually running the simulation up to that point. That means the model has to operate with steps in simulation time and cannot rely on a fixed granularity for calls to update the calculation or continuous operation. Given the stochastic nature of the oscillator, this is a challenge as it has to be possible to incrementally draw samples for points which are close to each other without producing inconsistent output. The actual requirements for the simulation model can be summed up to:

- The behaviour of the simulator output has to be quantised. For an oscillator model, the generated ticks have to be assigned to a discrete simulation time. This is usually no problem as it is possible to have a much finer grained simulation timescale than the actual requirements of the result. For example, the least possible difference between two events in a DES system can be picoseconds when the simulation accuracy aims at values in the nanosecond range.

- The model has to support two basic request types: it must be possible to request the number of oscillator ticks elapsed until a given simulation time and also vice versa, i. e., obtain the simulation time reached by a given number of ticks.

- All data have to be consistent. Pairs of simulation time and oscillator ticks (t, n) must be monotonic, i. e. if $t_i > t_j \rightarrow n_i \geq n_j$, irrespective of the order in which they are generated. Moreover, identical requests must yield identical results, i. e. if $t_i = t_j \rightarrow n_i = n_j$, for a single simulation run. This issue is the most challenging one, as the scheduled events not necessarily appear in the same sequence as they have to be processed later on. If an event has to be scheduled before (according to the simulation time) any already existing one, the latter have to be rescheduled to keep the sequence monotonic even under the stochastic behaviour. Therefore, requests and their answers have to be stored.

In addition, the (in general) always limited resources of a simulation environment have to be considered. In this sense the most limiting factors are the available memory and the processing time. The above requirements could be easily satisfied by storing the exact time of every single oscillator tick. However, this is unfeasible due to the lack of memory. Thus, a caching strategy has to be developed where – for the sake of consistency – already calculated and scheduled events (instead of the ticks themselves) can be stored.

7.2.1 Oscillator Model Concept

Based on the considerations of the preceding section and the oscillator properties mentioned in section 4.2.1, the actual task of a good model is to reproduce the behaviour of the five different noise types. For this purpose the PSD of the effective noise can be used, since it is deterministic and independent of time (for these certain noise types). The approach to imitate realistic noise behaviour is the following:

If a random signal, $u(t)$, is filtered by a linear, time-invariant filter with the frequency response function $H(f)$, the power spectral density of the output signal, $v(t)$, can be calculated as [Hän01]

$$S_{y,v}(f) = S_{y,u}(f)\left|H(f)\right|^2, \tag{7.3}$$

where $S_{y,u}(f)$ is the PSD of $u(t)$ and $S_{y,v}$ refers to the PSD of the output signal. This fact can now be used in the following manner: in case $u(t)$ is white noise, $S_{y,u}(t)$ equals one. Therefore, the spectrum of the five types of noises can be defined which allows to define a filter cascade for white noise like in figure 7.4. If the transfer function of the filter, $H(f)$, has the same shape as the PSD of the desired oscillator, the resulting signal, i. e. the filtered white noise, equals the noise of the oscillator.

The key step to achieve a realistic model of an oscillator is therefore to acquire and define the PSD of the noises produced by the real oscillator. However, since it is easier to perform time-based measurements and then calculate the Allan variance, often the relation between the Allan variance and the PSD, given by equation 4.6, is used.

For a system with limited bandwidth this correspondence between the Allan variance and the PSD has to be adapted in order to consider the frequency limits. Thus, it is given by [AAH97]

$$\tilde{\sigma}_y^2(\tau) = 2\int_{f_{\mathrm{lc}}}^{f_{\mathrm{uc}}} S_y(f)\frac{\sin^4(\pi f \tau)}{(\pi f \tau)^2}\mathrm{d}f. \tag{7.4}$$

The modification concerns the always positive cut-off frequencies f_{lc} and f_{uc} of the measurement system. Thus, the corresponding PSD has to be limited to $[f_{\mathrm{lc}}, f_{\mathrm{uc}}]$, too. Since the different individual types of noise are proportional to a power of f, equation 4.3 allows to assemble (approximate) the resulting PSD of the overall noise out of the

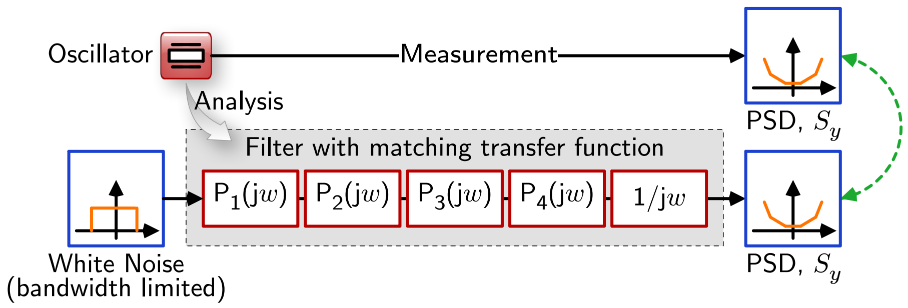


Figure 7.4: Concept of the oscillator model. A filter with a frequency response equal to the PSD of an oscillator is used to extract similar noise behaviour from a bandwidth-limited white noise. The elements of the dashed box are mathematically defined in the way that they as a whole form the spectrum according to the Allan variance of the oscillator.

five common noises using different weight factors h_α. In the case of a restriction to a bandwidth limited signal, the redefinition is simple, as only the contributing parts to $S_y(f)$ have to be cut off,

$$\tilde{S}_{y,\alpha}(f) = \begin{cases} h_\alpha f^\alpha & \text{if } f_{\mathrm{lc}} < f < f_{\mathrm{uc}} \\ 0 & \text{otherwise} \end{cases}. \tag{7.5}$$

Finally, the PSD can be summed up to

$$\tilde{S}_y(f) = \sum_{\alpha=-2}^{2} \tilde{S}_{y,\alpha}(f). \tag{7.6}$$

Using the above mentioned equation 7.6, the Allan variance on a limited bandwidth, $\tilde{\sigma}^2_{y,\alpha}$, is defined as

$$\tilde{\sigma}^2_{y,\alpha}(\tau, f_{\mathrm{lc}}, f_{\mathrm{uc}}) = 2\int_{f_{\mathrm{lc}}}^{f_{\mathrm{uc}}} \frac{\sin^4(\pi f \tau)}{(\pi f \tau)^2} \tilde{S}_{y,\alpha}(f) \mathrm{d}f \tag{7.7}$$

with the trivial consequence for the unlimited Allan variance that can be composed out of the individual parts,

$$\tilde{\sigma}^2_y = \sum_{\alpha=-2}^{2} \tilde{\sigma}^2_{y,\alpha}(\tau, f_{\mathrm{lc}}, f_{\mathrm{uc}}). \tag{7.8}$$

Power Spectral Density Forming

After the measurement of the Allan variance the h_α parameters and hence the PSD, $S_y(f)$, can be sufficiently determined. [Gad08]

- A straightforward approach would be to choose five values from the Allan variance plot and build a linear system of equations. This turns out not to be the optimal method. Due to the imperfections of the measurement system, the calculated values of the Allan variance do not fit the given curve perfectly and produce significant variations.
- To reduce the error of the model compared to the real oscillator, a second approach is to make an approximation of the Allan variance by calculating a fitting curve using the least square method [GLN+07].
- Another way to calculate the approximation is to use the infinity norm $\|\cdot\|$ of the error vector. The success of this method depends on the starting values of the approximation. If chosen properly, the technique turns out to be the best suited approach for the high dynamic of this double logarithmic scaled linear problem.

The next step is to bring the PSD function of the oscillator noise into a form that a causal transfer function $H(s)$ can be found. By substituting $\omega = 2\pi f$, S_y can be determined as

$$S_y(\omega, \omega_{\mathrm{lc}}, \omega_{\mathrm{uc}}) = \left(\frac{s_{-2} + s_{-1}\omega + s_0\omega^2 + s_1\omega^3 + s_2\omega^4}{\omega^2}\right) \cdot \Pi_{\omega_{\mathrm{lc}},\omega_{\mathrm{uc}}}(\omega). \tag{7.9}$$

In this equation the function $\Pi_{\omega_{\mathrm{lc}},\omega_{\mathrm{uc}}}(\cdot)$ is the windowing (*boxcar*) function [OS09], which defines a window satisfying

$$\Pi_{\omega_{\mathrm{lc}},\omega_{\mathrm{uc}}}(\omega) = \begin{cases} 1 & \text{if } \omega_{\mathrm{lc}} \le \omega \le \omega_{\mathrm{uc}} \\ 0 & \text{otherwise} \end{cases}. \tag{7.10}$$

Due to the fact that the input signal of the filter is white noise – the density function therefore equals one – the PSD of the output signal is given by

$$S_y(\omega, \omega_{\mathrm{lc}}, \omega_{\mathrm{uc}}) = |H(\mathrm{j}\omega)|^2 = H(\mathrm{j}\omega)H^*(\mathrm{j}\omega), \tag{7.11}$$

where $H(\mathrm{j}\omega)$ is the transfer function of the filter. According to the equations 7.9 and 7.11, the transfer function can now be defined as

$$|H(\mathrm{j}\omega)| = \frac{\sqrt{s_{-2} + s_{-1}\omega + s_0\omega^2 + s_1\omega^3 + s_2\omega^4}}{\omega} \cdot \Pi_{\omega_{\mathrm{lc}},\omega_{\mathrm{uc}}}(\omega). \tag{7.12}$$

In the following, the step from $|H(\mathrm{j}\omega)|$ to a factorized representation $|P(\mathrm{j}\omega)|$ is done under the assumption that the white noise input of the filter is bandwidth limited. Therefore, the limitation of the filter can be shifted to the noise source. The introduction of

$$|P(\mathrm{j}\omega)| = \frac{\sqrt{s_{-2} + s_{-1}\omega + s_0\omega^2 + s_1\omega^3 + s_2\omega^4}}{\omega} \tag{7.13}$$

and factorising the polynomial $s_{-2} + s_{-1}\omega + s_0\omega^2 + s_1\omega^3 + s_2\omega^4$ via real-valued constants $\omega_1 \dots \omega_4$ such as $(\omega_1 + \omega)(\omega_2 + \omega)(\omega_3 + \omega)(\omega_4 + \omega)$ allows to reach the representation,

$$|P(\mathrm{j}\omega)| = \sqrt{\omega_1\omega_2\omega_3\omega_4} \cdot \frac{|P_1(\mathrm{j}\omega)|\,|P_2(\mathrm{j}\omega)|\,|P_3(\mathrm{j}\omega)|\,|P_4(\mathrm{j}\omega)|}{\omega}, \tag{7.14}$$

where $|P_i(\mathrm{j}\omega)|$ is defined as

$$|P_i(\mathrm{j}\omega)| = \sqrt{(1 + \frac{\omega}{\omega_i})}. \tag{7.15}$$

Hence, one can consider $P(\mathrm{j}\omega)$ as a cascade of five transfer functions: $P_1(\mathrm{j}\omega)$, $P_2(\mathrm{j}\omega)$, $P_3(\mathrm{j}\omega)$, $P_4(\mathrm{j}\omega)$, and $1/\mathrm{j}\omega$. The last term represents an integrator, $P_{\mathrm{int}} = \sqrt{\omega_1\omega_2\omega_3\omega_4}/\mathrm{j}\omega$ that has a gain of $g = \sqrt{\omega_1\omega_2\omega_3\omega_4}$. Contrary, the first four parts, $|P_i(\mathrm{j}\omega)|$, represent a 10 dB/decade slope of the filter transfer function. Since it is not possible to build a filter using discrete components, which has this amplitude characteristics, an approximation is required. The latter can be achieved by utilising a lead-lag-filter,

$$P_{\mathrm{apr},i}(\mathrm{j}\omega) = \prod_{k=0}^{\infty} \frac{(1 + \frac{\omega}{\omega_{\mathrm{z},i,k}})}{(1 + \frac{\omega}{\omega_{\mathrm{p},i,k}})}, \tag{7.16}$$

where $k = 1, 2, 3, 4 \dots$ are natural numbers (for practical reasons implementations will limit the product to a distinct upper limit of k to allow for feasible calculation times). Furthermore, the zeros, $\omega_{\mathrm{z},i,k}$, and the poles, $\omega_{\mathrm{p},i,k}$, are chosen as

$$\omega_{\mathrm{z},i,k} = 2^{4k-5}\pi\omega_i, \tag{7.17}$$

$$\omega_{\mathrm{p},i,k} = 2^{4k-3}\pi\omega_i. \tag{7.18}$$

Following the approach explained above, the practical implementation of the oscillator model starts with the measurement of the Allan variance and the extraction of the spectrum defining h_α parameters.

An intermediate, however necessary step in the design is to switch from the continuous time to the discrete time domain. For the sake of simplicity this first step is done using MATLAB®, which involves the following two tasks:

- A digital filter with the same frequency response as the analogue $H(\mathrm{j}\omega)$ has to be created. The biggest problem in this step is the wide dynamic range that the digital filter cascade has to cover. Since the dynamic frequency range of digital filters is more limited than that of analogue filters, the higher frequencies are cut off to avoid stability problems.

- The required white noise source is usually modelled using random number generators providing a normal distribution. An identical seed to (internal state of) the random number generator allows repeatable experiments, which is one of the big advantages when simulating oscillators compared to (even highly controlled) experiments.

7.2.2 DES Model Description

As already mentioned, in the introduction to this section 7.2, the overall goal of the developed model is to have an oscillator model available for simulation in a DES environment. However, this is not needed for the sake of itself, but the model has to be embedded into a large scope of an efficient event triggered simulation (scenario). Owing to its high flexibility and particular suitability for network simulation, the OMNeT++ [Var99] simulator was chosen for a prototype implementation.

As a basic model, a typical clock synchronization problem is taken as test case. In this scenario the oscillator model is used as the time reference for the clock of a slave node. This clock consists of the ABC that increases the value with each tick of the oscillator. The node is then synchronized by a second, in the context of this experiment ideal master clock, which is per definition running perfectly aligned with the timescale of the simulation. At predefined points in time the master clock synchronizes the slave clock. The latter then requests from the oscillator model the number of ticks elapsed since the last clock update and calculates the actual local time of the node with respect to the current increment. Using this and a PI-controller structure a new increment is calculated in order to steer the slave clock according to the received timestamps from the master.

From an implementation viewpoint, the more complicated case is an extended one: for a full integration into a typical discrete event simulation, both directions for a conversion of the timescale have to be possible. On the one hand requesting the number of ticks at the current or future simulation time should be possible, but on the other hand also queries for the number of ticks at a given simulation time in the future (or vice versa the simulation time at a future number of ticks) should be answerable.

This prediction is necessary, e. g. to execute status changes of the simulated system in reaction to the triggering by the node internal clock (interrupts). This operation is relative simple for the discrete event framework, because it comes down to scheduling a message at a given point of the simulation time. However, it is not that easy if the oscillator model is introduced that translates the latter to the local time. In this case, it has to be ensured that the replies to transformation requests are consistent. As mentioned in the requirements list, this consistency is twofold: first, it has to be ensured that the number of ticks of request 1 is always smaller or equal to the ones of request 2, given that the simulation time in request 1 is smaller and second, a repeated request has to be answered in any case with the same value.

In order to fulfil these requirements a queue for all already calculated translations has to be established which stores the previously drawn value-tuples (simulation time and ticks). This queue has to be kept only for future values of the simulation time, as it is assumed that a simulated system does not schedule an event in the past. Additionally, the oscillator model can check all replies for consistency with this queue to avoid irregularities as described before.

From a statistical point of view, this problem renders rather severe, since it is in principle not allowed to draw another sample from the filtered random number generator, in case the previous did not meet the above mentioned requirements for the simulation.

This is due to the reason that using a second sample for the same request causes the statistical properties of the result (sample distribution) to be altered. Consequently, in such cases it would be required in principle that all drawn samples are repeated in a new simulation run using the same random number generator but with the knowledge of the sequence of the queries.

Nevertheless, a typical property of the aimed type of simulations can be taken into account: the probability for such cases is only reasonably high, if events for time differences have to be scheduled, which are significantly smaller than the time between two rising edges of the oscillator. Since even the simulated hardware runs at maximum with the oscillator frequency, the occurrence is very unlikely. Thus, the implemented model only checks for consistency and throws an exception, if the required conditions are not fulfilled. For the simulations done in this work, the mentioned criteria never applied, proving the low probability and therefore still the feasibility of the approach.

7.2.3 DES Model Implementation and Evaluation

As it will be illustrated later on in this section, the oscillator model exhibits realistic behaviour in terms of the Allan variance. However, it also has to be considered whether there is an efficient implementation for direct integration into a DES system.

One evident drawback of the model is the fact that the simulation of an oscillator produces an event with every status change (tick, period). Compared to relevant events in the simulation the clock source model would produce a significantly higher number of scheduled events in order to update the oscillator status. Hence, the simulation progress (i. e., global time) can only progress in single, small ticks.

This is not only resource consuming in case of a network containing a large number of clocks. Moreover, the actual advantage of the discrete event simulation – the reduced simulation effort due to the non-continuous calculation of status updates at only a distinct number of simulation events – cannot be used properly. Therefore, following this concept, the oscillator model has to be modified in a way that it is able to simulate long periods of time efficiently while still keeping the fine granularity when needed.

The approach described in the following was inspired by multi-rate filtering [Vai90, OS09], which replaces a single digital filter with a number of parallel ones. The latter structure uses different sample rates to achieve a higher processing efficiency in the system. For the DES model implementation, a parallel filter structure was designed that features different sampling rates. Each of the elements (cascade of filters as described in section 7.2.1) models the noise within a certain frequency band. The combination of the individual outputs than again shapes the PSD according to the one of a real-world oscillator. The advantage of this method is that longer periods of time can be calculated with the help of a filter modelling an oscillator with relatively low frequency, therefore reducing the number of events and calculation effort. In case finer granularity is needed, the intermediate values can be obtained by employing additional digital filters with higher sampling rates.

By combining the filters' outputs, arbitrary lengths of requested time intervals can be simulated in an optimised manner. While in multi-rate filtering intermediate values are

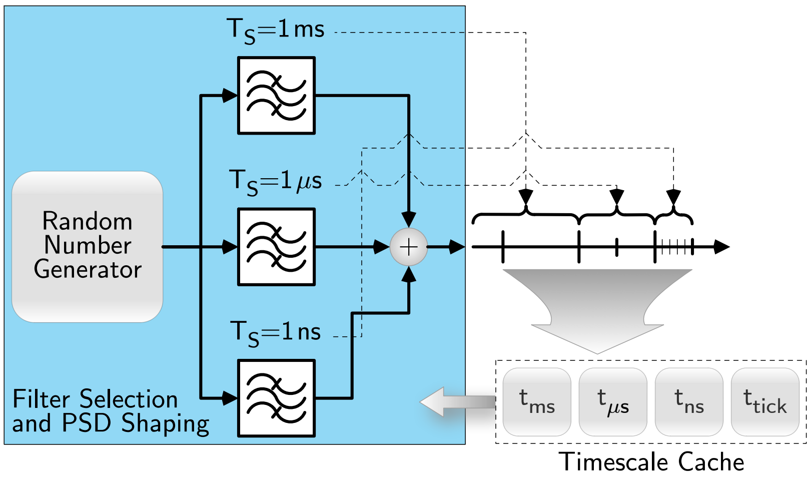


Figure 7.5: The structure of the OMNeT++ oscillator model

obtained by interpolation, in the presented approach the filters with higher sampling rates have the same PSD as the short term oscillator noise. This gives more realistic results for the oscillator noise on high frequencies compared to standard multi-rate filtering methods.

Further optimisations of the DES model can be done through adjusting the number of cascade elements, $P_i(\mathrm{j}w)$, for the effective transfer function of the individual digital filters. For example, since the long-term behaviour of the overall structure is determined by the transfer function of the filter with the lowest sampling rate, the remaining filters only need to cover a narrow frequency range. Hence, their transfer functions can be reduced by the long-term shaping parts and therefore be much simpler than the complete one defined by equation 7.14.

Finally, in the OMNeT++ implementation three parallel filter cascades (as illustrated in figure 7.4), with transfer functions $H_1(\mathrm{j}\omega)$, $H_2(\mathrm{j}\omega)$, and $H_3(\mathrm{j}\omega)$, have been designed. The structure of the OMNeT++ model is shown in figure 7.5. The sampling times of the filters are chosen as $T_1 = 10^{-3}\,\mathrm{s}$, $T_2 = 10^{-6}\,\mathrm{s}$, and $T_3 = 10^{-9}\,\mathrm{s}$. To additionally reduce the computational effort, as mentioned before, only the first filter provides the noise that has the same long-term behaviour as an oscillator. This is again connected to the mathematical model via the matched shape of the PSD. Using this set of sampling times it is possible to model an oscillator with the highest frequency equal to f_{max}=1 GHz and at the same time to calculate long time intervals (even in the range of multiple seconds) efficiently. As highlighted in section 7.2.2 the model serves two functionalities: it can calculate the simulation time for a requested number of oscillator ticks and additionally the elapsed number of ticks at a requested simulation time to schedule future events. Moreover, the requested times of oscillator ticks can be retrieved in an arbitrary order.

Even with all mentioned improvements, the model would be still ineffective if the node local time was calculated for every request from the oscillator start-up time on. Thus, the last calculated simulation time and number of ticks (together with the current filter

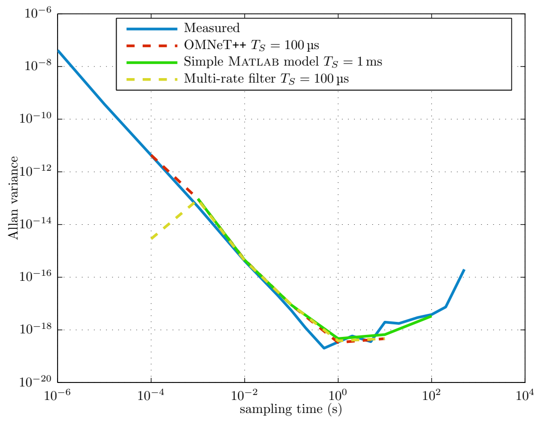


Figure 7.6: Comparison of Allan variance for various modelling approaches

values) are stored instead whenever an event is processed. These values are then used later on as reference points for calculating subsequent events.

A comparison of the Allan variance plots for different modelling approaches is shown in figure 7.6. The figure illustrates the variance of a real-world oscillator and the variance of an oscillator noise simulated using a filter with a sampling time of 1 ms. Besides these, the figure also provides the variance obtained using the described approach and by using multi-rate filters. In the last two cases the values for $T_s = 100\,\mu s$ are obtained by upsampling the values gained from the filter by a factor of 10. Since the multi-rate filter uses interpolation, the calculated samples are correlated and, hence, the value of the Allan variance is much lower than the measured value. On the other hand, the proposed approach utilising a white noise filter instead of interpolation provides more realistic results.

As the graphs indicate, oscillator noise can be simulated over several orders of magnitude with sufficient precision compared to the real-world oscillator. Thus, also the optimised multi-rate filter approach that gives higher simulation performance is applicable. The method clearly exceeds simpler models concerning real behaviour, which allows accurate simulation of high accuracy clock synchronization systems.

8 Outlook and Conclusion

During the course of this thesis the performance of packet-oriented clock synchronization over Ethernet was significantly improved using dedicated hardware support at layer 2. Still a number of issues remain open to be investigated for future improvements in the topic of clock synchronization and the achievable accuracy. One of the biggest topics to be handled is tight integration with existing architectures.

- In order to avoid non-deterministic influences of the physical layer (e. g., the PLL locking behaviour on link establishment) or additional clock transitions (e. g., for RMII), it would be in general reasonable to move future timestamping units from the xMII interface to the physical layer. There, timestamping can also benefit from continuously available carrier signals or sub-symbol timing calculations (e. g., 1000 Base-T).

- Integration of the protocol at the MAC layer allows to run clock synchronization without dependency on the higher layers, operating system, or node architecture. As outlined in section 8.1 there are already possible solutions. Since they themselves cannot enhance accuracy, this topic is not covered by the present thesis.

- The combination of network interface and timekeeping device, as indicated in the previous paragraph, also suggests better integration with the operating system. Recent developments in Linux (see section 6.2.1), allow to seamlessly transfer timestamps from the hardware to the application. Still the design is at an early stage and requires further enhancements. Additionally, the control of timing enhanced network cards is not yet standardised. First efforts, e. g. [CM10], seek for a similar API as it was implemented for access of the NTP daemon to the kernel timing infrastructure.

- Further improvements should have the goal that clock synchronization is a transparent service of the clock source of a node. Currently, in common (embedded) computer architectures, the available clock source to an operating system are oscillators and (high precision) counters. A topic for an upcoming dissertation could be a timebase that is inherently synchronized to a master, completely transparent to the operating system. Thus, there is no need for special adaptation, except for (possible) configuration issues.

Apart from integrative aspects for clock synchronization, also different system aspects for networks are not fully covered by research yet. The probably most important future step is the increased usage of synchronous lines, meaning that frequency is transferred over the transmission channel. In contrast to SyncE the system only requires a continuous carrier frequency in order to permanently estimate the phase relation to the upstream clock. This not necessarily requires a clock tree hierarchy, as downstream nodes may be supplied by a different (local) source, which is controlled but not strictly coupled to the upstream. In addition to the frequency transfer, a combination with time-of-day packet transmission is required to provide an absolutely synchronized system. An approach for the optical domain was presented in section 2.3.

Another big step for the technology is the application to the wireless domain. Since current developments for networking increasingly focus there – home, office, and even in the area of factory and building automation – future systems are expected to require clock synchronization also for wireless links. This estimation originates from the fact that most techniques used for wired Ethernet have been adapted to work for WLAN according to IEEE 802.11 as well. Thus, section 8.3 gives further aspects of future developments for wireless communication links.

8.1 Layer 2 Implementations

One step towards higher integration is the utilisation of the PTP at layer 2. The protocol was defined in annex F of version 2 [IEE08b]. It allows to send PTP frames directly over Ethernet without the need for higher layer protocols. Thus, it is also possible to implement the protocol engine directly above the MAC without the need for further protocol stacks.

While current products commonly still use a program in user space to handle the relatively complex protocol states, future devices will increasingly use layer 2 implementations. The goal will be to provide a global view of the timescale to all devices within a LAN without the need for additional software in the application layer. An emerging standard that makes use of a MAC layer implementation is IEEE 802.1as [GGT09]. Similar to the LXI standard, it specifies the protocol and procedures used for time sensitive applications to guarantee certain synchronization constraints. Possible application areas are audio and video transmissions, across bridged LANs having fixed and symmetrical transmission delays, e. g. IEEE 802.3 full duplex links. The tasks covered are the maintenance of synchronized time during normal operation and following addition, removal, or failure of network components and network reconfiguration.

Although, accuracy is not enhanced by moving the processing of the PTP to layer 2, there are significant advantages of this solution. First, the integration of PTP close to the hardware of the MAC eases the handling of timestamps together with their respective packets. Since special queues can keep packet and timestamp in parallel there is no issue with late association at the protocol level. Additionally, no support of the OS is required and the step of translating the timestamp format from the internal representation to the one of the OS and finally to PTP for comparison can be omitted, ensuring full resolution for the control algorithm. Second, the integration of the clock source together with the synchronization algorithm allows to provide higher layers a synchronized timebase. This can be done without any special enhancements or modifications, hence, the solution is fully transparent for all time dependent applications.

Obviously, the full integration requires the network interface to have processing capabilities in order to handle ingress synchronization packets, generate new and response frames, and calculate the control algorithm. For that purpose an integrated microcontroller or a sophisticated state machine is required. Since modern network interfaces support various features, e. g. checksum offloading, fragmentation offloading, or scatter/gather DMA, there are already capabilities for frame processing available. Thus, the increased effort required to integrate PTP directly into the network interface has to be seen in relation to the already implemented capabilities. An increased version of the already implemented processing power is therefore usually sufficient to cover the additional procedures.

8.2 Embedded Frequency Carrier for Wired Links

The currently available implementations for Ethernet-based clock synchronization are in general based on timestamping at layer 2 or use a synchronous approach on the link in order to transfer frequency. The first approach is limited to the timing properties of the physical layer, which were tackled in the course of this thesis, and therefore has its limit at a standard deviation of about 500 ps. Contrary, the second solution does not allow to distribute an accurate time of day to attached nodes.

As mentioned in chapter 2.3, for optical links further improvements to the accuracy of a system are possible through the combination of the transfer of frequency and time of day. Another logical step is the provision of clock synchronization techniques already at the physical layer. So far, the technology moved from pure software-based approaches to software assisted by hardware (timestamping).

With this step the clock synchronization task was split between hard- and software, which resulted in numerous problems like mapping the timestamps to the corresponding frames and control loop issues. The latter is especially critical for small synchronization intervals as the protocol stack running in the OS is unable to react in a deterministic way (unless equipped with real-time extensions). While smart algorithms have ruled out these problems, the next step is clearly visible: hardware only clock synchronization.

As indicated by the previous section, the goal is to have clock synchronization as encapsulated services of the network to the node. Consequently, a future goal has to be the integration of accurate synchronization at the physical layer, which then transparently provides the common timebase of the network to the node. An appropriate solution has to tackle the following issues:

- One of the biggest problems in current clock synchronization schemes is the asymmetry which affects the final achievable accuracy. Not only the PHY internal structure, but also the different lengths of the cable pairs contribute to this offset between the clocks. Thus, increased accuracy can be achieved if the targeted solution offers the measurement of the length of individual line pairs. It should be possible to separately determine the delay on receive and transmit line pairs. Especially, for the utilisation in localisation applications (see chapter 2.2) also the static offset significantly degrades the usability of the system.

- Another important feature for highest accuracy is the possibility to continuously measure the link delay. This is required to compensate for different delays caused by changes of environmental parameters that influence the propagation speed or length of the used transmission media, e. g. mainly temperature, humidity, or geometry (bends in a multi-mode fibre). Additionally, the permanent tracking of the delays should not affect the available data transmission bandwidth.

- Due to the above mentioned permanent tracking of the remote side, an appropriate message exchange would consume significant bandwidth that could not be used for data transmission. Therefore, an optimal solution should use out-of-band signalling to establish clock synchronization.

- The synchronization protocol utilised for the transfer of the time of day should run on layer 2 of the ISO/OSI protocol stack. The PTP is available in an appropriate standard, which can be implemented efficiently in hardware in order to provide clock synchronization as a service to higher layers. This is one step towards the goal of fully transparent integration with independence of the used platform or operating system.

- Finally, the targeted architecture should provide a timebase for operating systems without requiring the latter to care about the synchronization procedure. This includes the provisioning of timers, triggers, and timestamping of system events. The functions should be made available via the standard APIs used in current operating systems for clock sources. In an ideal world, one should be able to plugin a network card and have the node being synchronized to a networked remote clock source. That means, all components would have to interact in an integrated way and transparently for the operating system, which has access to the time like to any other local clock source.

- Current synchronization protocols require the exchange of messages with high rates, if the time of a master node should be tracked in a fine-grained and precise way. The logical optimum is to transfer the frequency to the slave node via the link and to continuously track the phase-/frequency offset between local node and the remote side. Thus, the synchronization messages are only required to transfer the current time of day, which then has to be done only at a much lower rate than required for accurate syntonization.

A possible solution is tackled in the Ætas project [EGK10]. In this project, clock synchronization is intended to be implemented as a hardware service. The advantage of this approach is significant: the system is independent of the operating system, bus structure, and processor platform. Further, it has no processing time constraints in the host OS. Similar to the Auto-Negotiation function (clause 28 of [IEE08a]) of standard Ethernet PHYs, which allows the exchange of capabilities and agree on a common mode of operation, the devices are designed to automatically synchronize without the need to run an operating system or any kind of software stack. This approach is similar to the implementation of PTP version 2 at layer 2, which also allows to benefit from the integration of clock synchronization at a low layer to provide a service to the node. Nevertheless, layer 2 operation still suffers from the same restrictions regarding accuracy, i. e. asymmetry issues or a high message exchange rate reducing the available data bandwidth. The project covers the issues described above using the following techniques:

Synchronous Clock Distribution. Syntonization is achieved by a constant carrier frequency. This continuous transfer of the master clock is either done using data, which is modulated to the passband or by clock recovery based on the signal transitions, if the signal is not modulated and transmitted in the baseband. Most important criteria for this technique is that the signal is constantly present and can be processed by the receiving node either directly or via permanent phase estimation.

Absolute Time of Day. Additionally to the frequency transfer, a protocol to exchange the time of day is run in order to align all nodes also absolutely. Therefore, the proposed system combines two approaches: the syntonization will be maintained by the physical layer as in SyncE (clock recovery from data), whereas the synchronization (the phase alignment) will be established by accurate link delay measurements and time of day distribution by data frames. If the frequency is shared among all slaves, bringing the absolute time to all nodes resolves in estimating the phase (theoretically only once) at power-up.

Given that the transmission delay does not change over time, all nodes have a constant but unknown delay to each other. If all transmission delays are measured, a synchronous clock output to the master can be generated by a PLL in the slaves with the ability to adjust the phase.

Separate Communication Channel. The key component of the proposed system is an separate logic that is responsible for all synchronization information (e. g., time, offset, or status info). The required circuit can be either implemented in a separate device in parallel to a COTS PHY or in a later step even integrated into one device. In order to have a separate communication channel for timing information, all synchronization related data will be transmitted using a modulation scheme, which is highly orthogonal to the Multilevel Transmission Encoding – 3 levels (MLT-3) encoding of Ethernet meaning the crosstalk between the two encoding schemes is low.

As possible solutions, Direct Sequence Spread Spectrum (DSSS) or passband modulation can be used. The first spreads the signal energy over a wide spectrum making it appear as white noise to other receivers. It uses the same spectrum of the MLT-3 encoding of Ethernet, but due to the spreading the required spectral power can be very low and therefore ensure reduced cross talk. The second solution uses the same encoding, but a different centre frequency. As Ethernet itself uses the frequency band up to 31.25 MHz (for 100 Base-T), the clock synchronization carrier is designed for 62.5 MHz of bandwidth which works flawlessly with category 5E cabling and therefore in parallel.

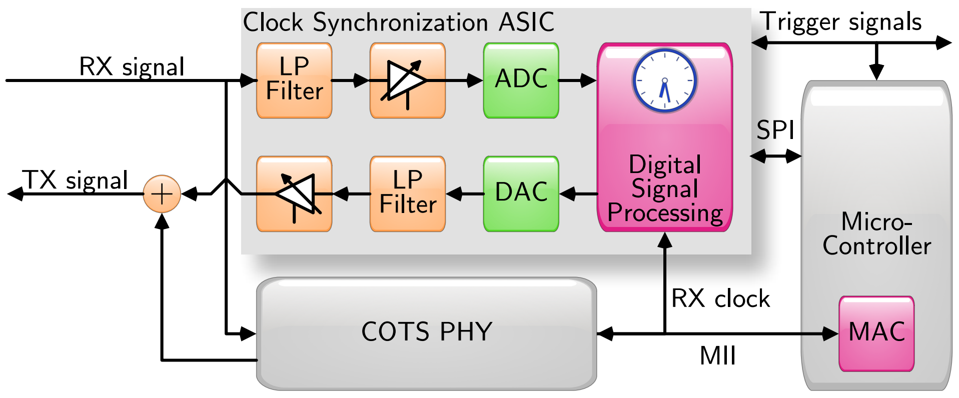


Figure 8.1: Physical layer architecture with synchronization extensions. For COTS products, the extensions can be integrated into the PHY.

Timestamping. The accuracy of hardware based timestamping systems is limited by the fact that the PHY uses a hybrid digital receiver structure which utilises matched filtering and clock recovery in the analogue domain. One proven option to increase the resolution is to employ a digital receiver. In contrast to hybrid receivers, the ADC is driven by the local (fixed) clock and the phase tracking is performed numerically by the symbol synchronizer. The exact phase relationship between local clock and received signal is therefore available as a numerical value. Using the phase estimation methods of chapter 5.4.2, the device is able to detect the relationship between local and remote clock.

As the proposed system shares a common clock among the nodes the timestamping resolution is only critical for the delay measurements. The delay measurement signal from the slave will arrive at any possible clock phase in the master node depending on the propagation delay on the line and therefore this signal will not be aligned to the local clock. The digital receiver can resolve the resolution issue by providing the phase as numerical estimation value.

Asymmetry Measurement. The measurement of line asymmetry can only be performed by an observer or link delay measurements on a per line basis. The latter approach uses the transmission of messages in periodic intervals over one channel back and forth. The principle is the same as round trip measurements in PTP with the difference that the same cable pair is used in both directions. As the timescale on both nodes is the same due to syntonization, it does not matter how fast the master replies to Delay_Req message. The fact that the same cable pair has to be used to determine the asymmetry requires an analogue switch to change cable pairs.

Figure 8.1 gives an overview of the design. The analogue/digital conversion subsystem with amplifiers is utilised to transmit and receive signals. It uses variable gain amplifiers to adjust the system to the cable attenuation and adopt the transmission power – the low-pass filters are required to fulfil Shannon's sampling theorem for the ADC and to suppress out-of-band distortions of the reverse direction. Both will be clocked synchronously to the receive clock of the PHY, whereas the actual required clock frequency depends on the modulation scheme.

All further signal processing as well as the clock handling is performed using digital logic. As the front-end must be able to measure asymmetry by detecting the round trip delay on a single wire pair, receive and transmit channel have to be swappable by a switch. Finally, the additional synchronization device performs similar functionality as the previously described CSC, but integrated into the physical layer of the networked device.

8.3 Wireless Links

As illustrated in section 2.2.2 timestamping for wireless Ethernet networks is feasible with sufficient accuracy [EGL10], even for high demanding applications like localisation. Although the basic mechanism for timestamping has been proven to work and can be easily extended to both communication directions, this is only the prerequisite for clock synchronization in wireless networks. Due to the clear separation in implementations of the first two network layers in IEEE 802.11 [IEE07], the implemented timestamping mechanism is located at the physical layer, using the baseband signal. Still, the transmission properties of the high frequency signal over the channel can cause significant distortions to clock synchronization mechanisms.

As networks according to IEEE 802.11 operate half-duplex on a shared medium there are some theoretical benefits. Unfortunately, several practical drawbacks, like changing path delays, reflections, attenuation, and noise outweigh the advantages of a common channel – thus identical properties – for all nodes. In order to allow future solutions in wireless networks to accurately synchronize nodes, not only precise timestamping is required, but also path delay estimations, which provide a precision in the same order of the timestamps.

Since path delays are commonly estimated using round-trip measurements, there are several factors of the physical layer including the wireless channel itself that can hinder appropriate results and therefore accurate clock synchronization.

- One of the most important influences is the *propagation delay through the high-frequency transceiver*. It would not matter, if constant, but unfortunately it is dependent on the temperature as well as the actual amplification factor. If reported by the devices, both factors can in theory be compensated with predetermined compensation functions. Still issues like dispersion of production and ageing have to be handled to render the method usable for COTS usage. All these issues are not yet fully investigated.

- Signal *attenuation* of objects that are within the operational area of the network causes varying properties of the channel. Besides the change of the Signal-to-Noise Ratio (SNR) and a linked required increase of the amplification – causing higher propagation delay – also loss of signal in one direction is possible. The latter can be caused by different effective transmit power of the nodes.

- Objects between or around the communication nodes can hinder direct *line-of-sight* and/or cause reflections of the transmitted signal. Since it is not guaranteed that the direct signal is the strongest, always the first arriving one – which is the direct signal – has to be chosen. Unfortunately, due to detection issues, the direct line-of-sight signal might not be visible to the node, thus changing paths can cause varying transmission delays.
- Generally speaking, changes due to *moving nodes* can cause variances in the path delay and change the asymmetry factor of the communication channel. The mobility is one of the key arguments for wireless transmission schemes and therefore important to be tackled when it comes to disturbances for clock synchronization. One issue that results from this is the fact that the path delay has to be continuously tracked for changes. In case of accelerated nodes, this is even required for keeping the syntonization.

While channel problems can relatively easy be tackled in localisation applications by using multiple receivers (three are required anyway) to build an overdetermined system, the overhead for a synchronization system would be rather high. Therefore, the difficulties introduced by the unstable properties of the channel weigh a lot more for the latter. Another big simplification the localisation system can benefit from is the fact that the path delay is not required at all. The wireless (mobile) node thus is not required to support timestamping at all allowing also COTS clients to use the localisation infrastructure provided by a network.

The independence from the absolute path delay and almost implicit redundancy allowed localisation implementations to progress over wireless synchronization applications. The overhead of multiple receivers is usually not feasible for synchronization only demands. Future solutions have to cover the problem of varying channel parameters in order to be successful.

Although the synchronization scheme of NTP, which uses four timestamps at each sample point, is not as efficient regarding bandwidth utilisation as the one of PTP it seams more feasible for wireless clock synchronization. The reason is the fact that with the two packets offset and path delay can be determined. Thus, there are as many offset values as delay measurements allowing to continuously track both in close relationship to each other. That way, the influence of changing properties of the channel, e. g. due to moving nodes, can be minimised at the cost of higher traffic. Optimised solutions should also investigate the possibility to attach timestamps to data packets in order to keep the overhead for clock synchronization low.

Another important issue for wireless clock synchronization will be filtering algorithms. Since the channel is prone to changes and variations, outliers and obviously wrong values should be filtered appropriately. While research in the wired domain started to seriously investigate these issues with the introduction of PTP version 2 [HM09], the wireless domain has to find filtering solutions from the beginning on to handle the unreliable channel. In addition to understanding the behaviour of the transmission delay, additional problems introduced by the possible movement of node have to be investigated for suitable algorithms.

8.4 Final Conclusions

The starting point of the work is the state of the art in timestamping, which assumes that drawing timestamps in hardware, e. g. between the physical and the media access layer, sufficiently solves the problem of accurate clock synchronization. Although the outcome is (significantly) better than pure software based solutions, the common approaches are not based on deep knowledge about the influencing factors on the achievable accuracy of a synchronizing system. Thus, this thesis characterises the components involved in order to give recommendations about the design of highly accurate systems.

The introduction leads into the world of clock synchronization, distributed systems, and why these two terms cannot be separated without losing applicability to useful applications. This forms the theoretical basis for the field of possible solutions. Supported by the evolution of networks for sensors and control, and instrumentation, respectively, given in the state of the art to packet-oriented networks, the quantity of technical approaches is narrowed down from synchronous ones. Due to the widespread use in modern communication systems the work is focused on Ethernet-based solutions.

Motivated by the fact that Ethernet originally was not designed for utilisation in real-time environments, adaptations have to be made in order to fulfil the requirements of the sample applications illustrated at the beginning of the thesis. The latter are covered by specialised hardware elements to enhance the technology with dedicated support for clock synchronization. In combination with the most popular protocols for clock synchronization, i. e. NTP and PTP, common real-time Ethernet solutions are analysed in a systematic way covering both, software and hardware components.

Although some PTP devices have hardware support for clock synchronization, mostly on the MII, a number of jitter sources remain, which can mitigate the overall performance. The thesis analyses which sources still affect the synchronization and to which extend. Among the most important ones, the physical layer (including the transmission standard), the involved oscillators, as well as several clock transitions, are characterised.

The detailed analysis of the achievable precision leads to design decisions concerning the Clock Synchronization Cell (CSC). The optimisation of the internal structure gives better performance regarding clock frequency as well as resources. Further, new techniques for gathering more accurate timestamps with enhanced versions of the MIIS are described and evaluated regarding the applicability.

The overall system architecture, the partitioning, and especially the interworking of hardware and software components is investigated. The solutions presented are an important factor for systems to ensure an efficient flow of timestamps together with their associated packets. As shown, a well chosen approach can significantly reduce the effort that has to be taken in hardware, while requiring only minimal changes in software. Furthermore, standardised interfaces for clock synchronization in the software layers

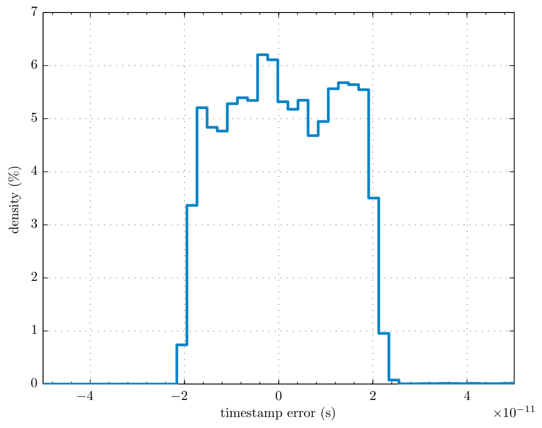


Figure 8.2: Timestamping performance for 50 ksamples of a combined phase/frequency estimation method using least squares regression

allow to use the information by many applications without the need for close interaction and therefore detailed knowledge of the actual hardware implementation.

All aforementioned aspects, which are covered throughout the thesis are primarily investigated by developing appropriate mathematical representations or simulation models, which are then compared to real-life measurements. The approach allows to validate the model, while the influence factors on the behaviour can be easily extracted and analysed to draw conclusions for enhanced synchronization approaches.

One of the most important outcomes, besides the characterisation of the influence factors on the achievable accuracy, is the identification of tradeoffs, e. g. accuracy in dependence of the oscillator (quality) as shown in figure 6.8) or the timestamp precision, see figure 7.3, versus the synchronization interval. Depending on the application and the available hardware options, an optimal solution can be selected using the developed guidelines. A high timestamp precision is only effective in combination with an oscillator that is sufficiently stable over the synchronization interval. On the other hand, from a certain rate onwards, increasing the accuracy does not further benefit from more frequent updates from the timing master. Furthermore, the selected transmission standard defines an important boundary condition for the achievable accuracy as examined in detail in chapter 4.3.

Figure 8.2 shows the performance of a combined method consisting of a phase/frequency estimation and least squares regression using 128 samples over a single frame. Since the phase estimation suffers from limited resolution for detecting the cycle slip, the much more accurate frequency estimation is used to generate all but the first timestamp. That way, it is possible to reduce the influence of cycle slip jitter and measurement noise. Actually, the least squares regression does benefit from a slight phase jitter since the algorithm can cope better with such a distortion than with a systematic error. The outstanding performance of only 12.036 ps standard deviations (for a reference frequency of 50.1 MHz) proves that the method used in the prototype clearly allows to reach the physical limitations of analogue signal transmission. As a reference, the typical phase

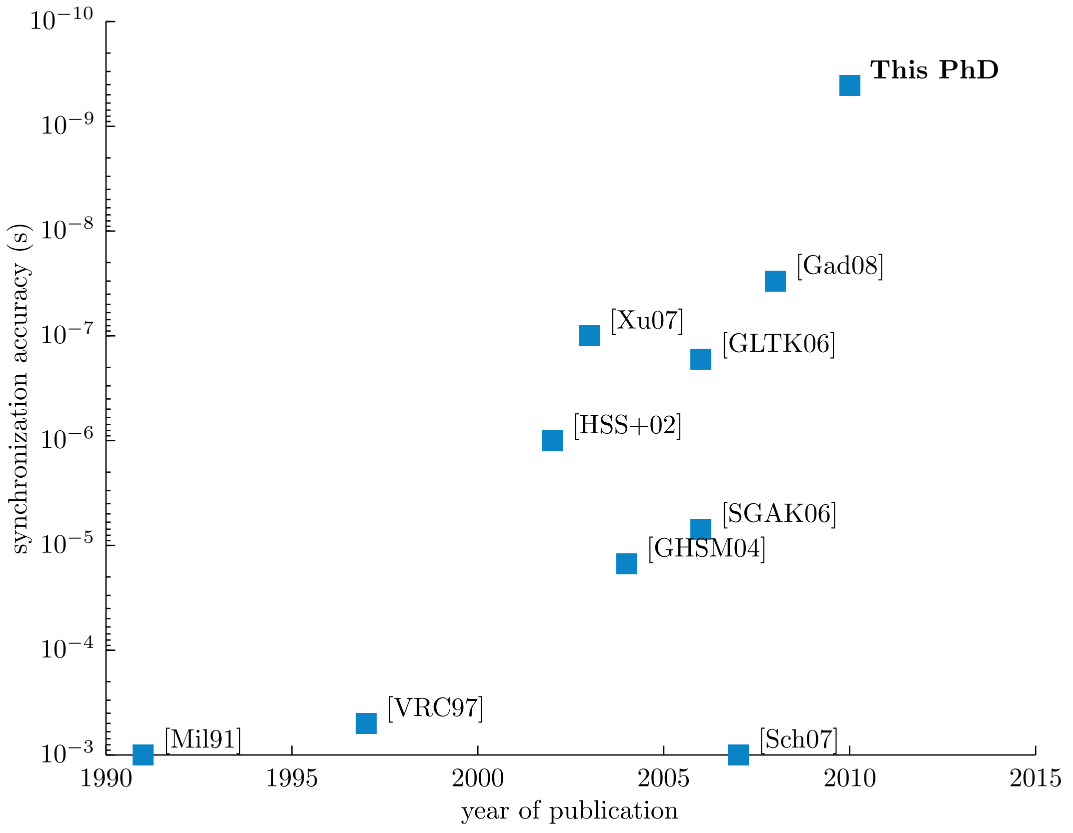


Figure 8.3: Comparison of selected publications with the presented achievements in terms of synchronization accuracy.

jitter of a recovered Ethernet clock is in the range of 300 ps if no special equipment is used.

The enhancement achieved by the phase/frequency estimating method compared to the starting point of the investigations, an oversampling packet detection approach with 10 ns resolution, is a factor of 240. As the precision improves also the overall synchronization accuracy improves. For the latter the factor is about 275 if all components are carefully adjusted according to the investigated characteristics and design rules. Figure 8.3 shows the improvement over the state of the art by the present work.

The integrated approach of looking at the system architecture allowed to optimise the adjustment between hardware and software components. Through the close integration the resource utilisation can be reduced while the overall performance could be significantly enhanced. Compared to the starting point of the work, the gained knowledge about the system characteristics allows to better estimate the achievable performance of a design. Additionally, it is used to significantly improve the maximum performance of an Ethernet-based clock synchronization system.

Acronyms

ABC	Adder-Based Clock	19
ADC	Analogue to Digital Converter	104
AHB	Advanced High-Performance Bus	97
AMBA	Advanced Microcontroller Bus Architecture	97
ANSI	American National Standards Institute	43
AWGN	Additive White Gaussian Noise	39
API	Application Programming Interface	62
ASIC	Application Specific Integrated Circuit	27
BMC	Best Master Clock	51
CERN	European Organisation for Nuclear Research (French: Conseil Européen pour la Recherche Nucléaire)	11
CSMA	Carrier Sense Multiple Access	52
CD	Collision Detection	60
CRC	Cyclic Redundancy Check	
CSC	Clock Synchronization Cell	27
COTS	Commercial Off-The-Shelf	31
CPU	Central Processing Unit	18
DARPA	Defense Advanced Research Projects Agency	47
DCNET	Distributed Computer Network	47
DDR	Double Data Rate	104
DES	Discrete Event Simulation	39
DMA	Direct Memory Access	112
DNS	Domain Name Service	48
DPP	Digital Phase Processing (phase alignment and/or measurement)	37
DSP	Digital Signal Processor	27
DSSS	Direct Sequence Spread Spectrum	158
DTSS	Digital Time Synchronization Service	47
ERTEC	Enhanced Real-Time Ethernet Controller	57

ETS	Equational Timing System	45
EUI	Extended Unique Identifier	112
FCS	Frame Check Sequence	17
FDM	Frequency-Division Multiplexing	42
FFM	Flicker Frequency Modulation	74
FPM	Flicker Phase Modulation	74
FFT	Fast Fourier Transform	105
FIFO	First In – First Out	21
FIR	Finite Impulse Response	106
FPGA	Field Programmable Gate Array	87
FSM	Finite State Machine	27
GMII	Gigabit Media Independent Interface	36
GPIB	General Purpose Interface Bus	11
GPS	Global Positioning System	20
HDL	Hardware Description Language	119
HID	Human Interface Device	29
ICMP	Internet Control Message Protocol	47
IEC	International Electrotechnical Commission	51
IEEE	Institute of Electrical and Electronics Engineers	27
IEN	Internet Engineering Note	
IIR	Infinite Impulse Response	107
IRT	Isochronous Real-Time	57
ISO	International Organisation for Standardisation	16
IP	Internet Protocol	8
IPG	Inter-Packet Gap	88
IRIG	Inter Range Instrumentation Group	20
IRQ	Interrupt ReQuest	20
ITU	International Telecommunication Union	43
LAN	Local Area Network	62
LHC	Large Hadron Collider	11
LORAN-C	LOng RAnge Navigation Revision C	29
LSB	Least Significant Bit	87
LT	Luby Transform	38

LVDS	Low Voltage Differential Signalling	62
LXI	LAN eXtensions for Instrumentation	11
MAC	Media Access Control	22
MD5	Message-Digest Algorithm 5	
MDIO	Management Data Input/Output	84
MII	Media Independent Interface	
MIIS	Media Independent Interface Scanner	66
NEXT	Near-End Cross Talk	79
NMEA	National Marine Electronics Association	96
MCXO	Microcontroller Compensated Crystal Oscillator	72
MLT-3	Multilevel Transmission Encoding – 3 levels	158
NIC	Network Interface Card	60
NTP	Network Time Protocol	47
OCXO	Oven Controlled Crystal Oscillator	69
OS	Operating System	67
OSI	Open Systems Interconnection	16
PC	Personal Computer	85
PCB	Printed Circuit Board	73
PCI	Peripheral Component Interconnect	63
PCM	Pulse Code Modulation	42
PHY	PHYsical layer entity	60
PI	ProfiBus and ProfiNet International	56
PLC	Power Line Communication	25
PLL	Phase-Locked Loop	44
PPS	Pulse Per Second	20
PSD	Power Spectral Density	74
PTCP	Precision Transparent Clock Protocol	57
PTP	Precision Time Protocol	27
PXI	PCI eXtensions for Instrumentation	62
QoS	Quality of Service	114
RAM	Random Access Memory	94
RFC	Request for Comments	47
RMII	Reduced Media independent Interface	60

RMS	Root Mean Square	48
RPC	Remote Procedure Call	48
RT	Real-Time	56
RTS	Returnable Timing System	45
RTSI	Real-Time System Integration	63
SCADA	Supervisory Control and Data Acquisition	25
SCM	(Linux) Socket Control Message	127
SDH	Synchronous Digital Hierarchy	43
SERCOS	SErial Realtime COmmunication System	34
SFD	Start-of-Frame Delimiter	54
SHA	Secure Hash Algorithm	
SI	international system of units (French: Système International d'unités)	13
SMA	Sub-Miniature type A	112
SNR	Signal-to-Noise Ratio	160
SNTP	Simple Network Time Protocol	47
SoC	System-on-a-Chip	118
SONET	Synchronous Optical NETwork	43
STR	Smart Timing Repeater	30
SyncE	Synchronous Ethernet	34
TAI	international atomic time (French: Temps Atomique International)	13
TCP	Transport Control Protocol	56
TCXO	Temperature Compensated Crystal Oscillator	63
TDL	Tapped Delay Line	104
TDM	Time-Division Multiplexing	42
TDMA	Time-Division Multiple Access	20
TDoA	Time Difference of Arrival	11
TLV	Type, Length, Value [according to IEEE Std. 802.1AB]	37
TSIP	Trimble Standard Interface Protocol	136
TTP	Time-Triggered Protocol	57
TVP	Time Variant Parameter	12
UDP	User Datagram Protocol	62
UTC	Coordinated Universal Time [ITU02]	13
UUID	Universally Unique IDentifier	52

VFO	Variable Frequency Oscillator	45
WAN	Wide Area Network	
WLAN	Wireless Local Area Network	28
WFM	White noise Frequency Modulation	74
WPM	White noise Phase Modulation	74
XO	Crystal Oscillator	69

Bibliography

[AAH97] David W. Allan, Neil Ashby, and Clifford C. Hodge. The Science of Timekeeping. Application Note 1289, Hewlett Packard, June 1997. Available from: `http://www.allanstime.com/Publications/DWA/Science_Timekeeping/`.

[All87] David W. Allan. Time and Frequency (Time-Domain) Characterization, Estimation, and Prediction of Precision Clocks and Oscillators. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 34(6):647–654, November 1987. `doi:10.1109/T-UFFC.1987.26997`.

[ANS95] ANSI. Synchronous Optical Network (SONET) - Data Communication Channel Protocol and Architectures. *Standard T1.105.04-1995 (R2005)*, pages 1–27, December 1995.

[ARM99] ARM Limited, 110 Fulbourn Road, Cambridge, Great Britain. *AMBA Specification*, 2.0 edition, May 1999.

[BC89] Ralph Ballart and Yau-Chau Ching. SONET: Now It's the Standard Optical Network. *IEEE Communications Magazine*, 27(3):8–15, March 1989. `doi:10.1109/35.20262`.

[Bec09] Beckhoff Automation GmbH, Eiserstraße 5, 33415 Verl, Germany. *ET1100 EtherCAT Slave Controller*, 1.6 edition, August 2009. Available from: `http://www.beckhoff.at/english/download/ethercat_development_products.htm`.

[Bel95] John C. Bellamy. Digital Network Synchronization. *IEEE Communications Magazine*, 33(4):70–83, April 1995. `doi:10.1109/35.372197`.

[BGNV09] Jeff Burch, Kenneth Green, John Nakulski, and Dieter Vook. Verifying the Performance of Transparent Clocks in PTP Systems. In *Proc. of the International IEEE Symposium on PrecisionClock Synchronization for Measurement, Control and Communication, ISPCS*, pages 1–6, Brescia, Italy, October 2009. `doi:10.1109/ISPCS.2009.5340195`.

[Bio06] Joseph Biondo. SERCOS III: Ethernet-based real time network for Industrial Automation. presentation, Bosch Rexroth Corporation, 5150 Prairie Stone Parkway, Hoffman Estates, IL 60192, USA, December 2006. Available from: `http://www.sercos.com/literature/literature.htm#10`.

[BNK89] Z. Bar-Nathan and G. J. Kühn. Mutual synchronization in digital networks. In *Proc. Southern African Conference on Communications and Signal Processing*, pages 37–42, June 1989. `doi:10.1109/COMSIG.1989.129013`.

[Boe90] Rodney J. Boehm. Progress in Standardization of SONET. *LCS, IEEE*, 1(2):8–16, May 1990. `doi:10.1109/73.80381`.

[Boe99] David W. Boerstler. A Low-Jitter PLL Clock Generator for Microprocessors with Lock Range of 340-612 MHz. *IEEE Journal of Solid-State Circuits*, 34(4):513–519, April 1999. `doi:10.1109/4.753684`.

[Bre98] Stefano Bregni. A Historical Perspective on Telecommunications Network Synchronization. *IEEE Communications Magazine*, 36(6):158–166, June 1998. `doi:10.1109/35.685385`.

[BRV09] Timothy Broomhead, Julien Ridoux, and Darryl Veitch. Counter Availability and Characteristics for Feed-forward based Synchronization. In *Proc. of the International IEEE Symposium on PrecisionClock Synchronization for Measurement, Control and Communication, ISPCS*, pages 1–6, Brescia, Italy, October 2009. `doi:10.1109/ISPCS.2009.5340210`.

[Buc88] John Buck. Synchronous rhythmic flashing of fireflies. ii. *The Quarterly Review of Biology*, 63(3):265–289, September 1988. `doi:10.1086/415929`.

[CF03] Flaviu Cristian and Christof Fetzer. Probabilistic Internal Clock Synchronization. In *Proc. of the 13th Symposium on Reliable Distributed Systems*, pages 22–31, Dana Point, California, USA, May 2003.

[CM10] Richard Cochran and Christian Marinescu. Design and Implementation of a PTP Clock Infrastructure for the Linux Kernel. In *Proc. of the International IEEE Symposium on PrecisionClock Synchronization for Measurement, Control and Communication, ISPCS*, pages 116–121, New Hampshire, Durham, USA, October 2010. OMICRON electronics GmbH. `doi:10.1109/ISPCS.2010.5609786`.

[Cos09] Lee Cosart. Packet Network Timing Measurement and Analysis Using an IEEE 1588 Probe and New Metrics. In *Proc. of the International IEEE Symposium on PrecisionClock Synchronization for Measurement, Control and Communication, ISPCS*, pages 1–6, Brescia, Italy, October 2009. Symmetricom, Inc. `doi:10.1109/ISPCS.2009.5340225`.

[CRKH05] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, USA, 3rd edition, February 2005.

[CSVV09] Gianluca Cena, Luci Seno, Adriano Valenzano, and Stefano Vitturi. Performance analysis of ethernet powerlink networks for distributed control and automation systems. *Computer Standards & Interfaces*, 31(3):566–572, March 2009. `doi:10.1016/j.csi.2008.03.022`.

[DG94] Xianzhong Dai and Ralf Gretsch. Quasi-synchronous sampling algorithm and its applications. *IEEE Transactions on Instrumentation and Measurement*, 43(2):204 – 209, April 1994. `doi:10.1109/19.293421`.

[Duc05] Peter Duchemin. Ethernet in Echtzeit. *Elektronik*, 4:42–46, March 2005. Available from: `http://www.elektroniknet.de/bauelemente/technik-know-how/halbleiter/article/128/0/Ethernet_in_Echtzeit`.

[DVRT08] Luca De Vito, Sergio Rapuano, and Laura Tomaciello. One-Way Delay Measurement: State of the Art. *IEEE Transactions on Instrumentation and Measurement*, 57(12):2742–2750, December 2008. `doi:10.1109/TIM.2008.926052`.

[EGK10] Reinhard Exel, Georg Gaderer, and Nikolaus Kerö. Layer 1 Ethernet Clock Synchronizaton. In *Proc. of the Precise Time and Time Interval (PTTI) Systems and Applications Meeting*, Reston, Virginia, USA, November 2010.

[EGL10] Reinhard Exel, Georg Gaderer, and Patrick Loschmidt. Localisation of Wireless LAN Nodes using Accurate TDoA Measurements. In *Proc. of the IEEE Wireless Communications and Networking Conference, WCNC*, pages 1–6, Sydney, Australia, April 2010. IEEE. `doi:10.1109/WCNC.2010.5506651`.

[eia69] EIA Standard RS-232-C Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Data Interchange, August 1969.

[Eid06] John C. Eidson. *Measurement, Control, and Communication Using IEEE 1588*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[EL09] Reinhard Exel and Patrick Loschmidt. High Accurate Timestamping by Phase and Frequency Estimation. In *Proc. of the International IEEE Symposium on PrecisionClock Synchronization for Measurement, Control and Communication, ISPCS*, pages 126–131, Brescia, Italy, October 2009. `doi:10.1109/ISPCS.2009.5340223`.

[eth09] The Ethernet Fieldbus: EtherCAT. Technical report, EtherCAT Technology Group, Ostendstraße 196, 90482 Nuremberg, Germany, October 2009. Available from: `http://www.ethercat.org/en/publications.html`.

[Fan90] Bertrand T. Fang. Simple Solutions for Hyperbolic and Related Position Fixes. *IEEE Transactions on Aerospace and Electronic Systems*, 26(5):748–753, September 1990. `doi:10.1109/7.102710`.

[FFM+08] Paolo Ferrari, Alessandra Flammini, Daniele Marioli, Stefano Rinaldi, Emiliano Sisinni, Andrea Taroni, and Francesco Venturini. Clock Synchronization of PTP-based Devices through PROFINET IO Networks. In *Proc. of the International IEEE Conference on Emerging Technologies and Factory Automation, ETFA*, pages 496–499, September 2008. `doi:10.1109/ETFA.2008.4638445`.

[FGJ+08] Jean-Loup Ferrant, Mike Gilson, Sebastien Jobert, Michael Mayer, Michel Ouellette, Laurent Montini, Silvana Rodrigues, and Stefano Ruffini. Synchronous Ethernet: A Method to Transport Synchronization. *IEEE Communications Magazine*, 46(9):126–134, September 2008. `doi:10.1109/MCOM.2008.4623717`.

[Fra83] Robert L. Frank. Current Developments in Loran-C. *Proceedings of the IEEE*, 71(10):1127–1139, October 1983. `doi:10.1109/PROC.1983.12742`.

[Gad08] Georg Gaderer. *Fault Tolerance Enhancements to Master/Slave Based Clock Synchronization.* PhD thesis, Vienna University of Technology, November 2008.

[Get93] Ivan A. Getting. Perspective/Navigation-The Global Positioning System. *IEEE Spectrum*, 30(12):36–38, 43–47, December 1993. `doi:10.1109/6.272176`.

[GGT09] Geoffrey M. Garner, Aaron Gelter, and Michael Johas Teener. New Simulation and Test Results for IEEE 802.1AS Timing Performance. In *Proc. of the International IEEE Symposium on PrecisionClock Synchronization for Measurement, Control and Communication, ISPCS*, pages 1–7, October 2009. `doi:10.1109/ISPCS.2009.5340214`.

[GHSM04] Gaderer Gaderer, Roland Höller, Thilo Sauter, and Hannes Muhr. Extending IEEE 1588 to Fault Tolerant Clock Synchronization. In *Proc. of the IEEE International Workshop on Factory CommunicationSystems, WFCS*, pages 353–357, September 2004. `doi:10.1109/WFCS.2004.1377745`.

[GLM08] Georg Gaderer, Patrick Loschmidt, and Aneeq Mahmood. A novel Approach for Flexible Wireless Automation in Real-Time Environments. In Gianluca Cena and Francoise Simonot-Lion, editors, *Proc. of the IEEE International Workshop on Factory CommunicationSystems, WFCS*, pages 81–84. IEEE, May 2008. `doi:10.1109/WFCS.2008.4638744`.

[GLN+07] Georg Gaderer, Patrick Loschmidt, Anetta Nagy, Roman Beibelbeck, Nikolaus Kerö, and Jörgen Mad. An oscillator model for high precision synchronization protocol discrete event simulation. In *Proc. of the Precise Time and Time Interval (PTTI) Systems and Applications Meeting*, pages 363–370, Long Beach, California, December 2007. Available from: `http://www.pttimeeting.org/archivemeetings/2007papers/paper33.pdf`.

[GLN$^+$09] Georg Gaderer, Patrick Loschmidt, Anetta Nagy, Reinhard Exel, and Thilo Sauter. Localisation in Wireless Sensor Networks. In Paddy J. French, editor, *Proc. of the IEEE Sensors Conference*, pages 1004–1009, October 2009. `doi:10.1109/ICSENS.2009.5398225`.

[GLS06] Georg Gaderer, Patrick Loschmidt, and Thilo Sauter. Quality Monitoring in Clock Synchronized Distributed Systems. In *Proc. of the IEEE International Workshop on Factory CommunicationSystems, WFCS*, pages 13–21, June 2006. `doi:10.1109/WFCS.2006.1704118`.

[GLS10] Georg Gaderer, Patrick Loschmidt, and Thilo Sauter. Improving Fault Tolerance in High-Precision Clock Synchronization. *IEEE Transactions on Industrial Electronics*, 6(2):206–215, May 2010. `doi:10.1109/TII.2010.2044580`.

[GLTK06] Georg Gaderer, Patrick Loschmidt, Albert Treytl, and Nikolaus Kerö. Internal and external clock synchronization in a power line network. In Lee A. Breakiron, editor, *Proc. of the Precise Time and Time Interval (PTTI) Systems and Applications Meeting*, pages 213–222. US Naval Observatory, December 2006. Available from: `http://www.pttimeeting.org/archivemeetings/2006papers/paper15.pdf`.

[GN06] Thomas Gleixner and Douglas Niehaus. Hrtimers and Beyond: Transforming the Linux Time Subsystems. In *Proc. of the Linux Symposium*, volume 1, pages 333–346, Ottawa, Ontario, Canada, July 2006.

[Gra10] Wolfgang Granzer. *Secure Communication in Home and Building Automation Systems.* PhD thesis, Vienna University of Technology, February 2010.

[Hän01] Eberhard Hänsler. *Statistische Signale.* Springer Verlag, Berlin, 3rd edition, 2001.

[HM09] Ilija Hadžić and Dennis R. Morgan. On Packet Selection Criteria for Clock Recovery. In *Proc. of the International IEEE Symposium on PrecisionClock Synchronization for Measurement, Control and Communication, ISPCS*, pages 1–6, Brescia, Italy, October 2009. `doi:10.1109/ISPCS.2009.5340211`.

[HSS$^+$02] Martin Horauer, Klaus Schossmaier, Ulrich Schmid, Roland Höller, and Nikolaus Kerö. PSynUTC – evaluation of a high-precision time synchronization prototype system for ethernet lans. In *Proc. of the Precise Time and Time Interval (PTTI) Systems and Applications Meeting*, Reston, Virginia, USA, December 2002. Available from: `http://www.pttimeeting.org/archivemeetings/2002papers/paper24.pdf`.

[hyp07] hyperstone / iAd, Line-Eid-Straße 3, 78467 Konstanz, Germany. *Hyperstone hyNet® XS*, 2f edition, July 2007.

[IEC04] Precision clock synchronization protocol for networked measurement and control systems. *IEC 61588 First edition 2004-09; IEEE 1588*, pages 0_1–156, September 2004.

[IEC07] Industrial communication networks — fieldbus specifications — part 1: Overview and guidance for the iec 61158 and iec 61784 series. *IEC 61158-1 Edition 2.0 2007-11*, November 2007.

[IEC08] Information technology - Generic cabling for customer premises. *IEC 11801 Edition 2.1 2008-05*, May 2008.

[IEE02] IEEE Std. 1588 - 2002 IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Std 1588-2002*, pages i–144, November 2002. Replaced by 61588-2004. `doi:10.1109/IEEESTD.2002.94144`.

[IEE04a] The Open Group Technical Standard Base Specifications. *IEEE Std 1003.1*, 6, 2004.

[IEE04b] Higher performance protocol for the standard digital interface for programmable instrumentation – Part 1: General. *IEEE Std 488*, pages 0_1–158, July 2004. `doi:10.1109/IEEESTD.2004.95749`.

[IEE07] IEEE Computer Society, New York, NY, USA. *IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, June 2007. `doi:10.1109/IEEESTD.2007.373646`.

[IEE08a] IEEE Computer Society, New York, NY, USA. *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*, December 2008. `doi:10.1109/IEEESTD.2008.4726971`.

[IEE08b] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pages c1–269, July 2008. `doi:10.1109/IEEESTD.2008.4579760`.

[Int05] Intel Corporation. Hardware-Assisted IEEE 1588 Implementation in the Intel IXP46X Product Line. White Paper 001, 2200 Mission College Blvd., Santa Clara, CA 95054-1549, USA, March 2005.

[ISO94] Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model. *ISO/IEC 7498-1*, 1:1–59, November 1994. Available from: `http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip`.

[ISO07] International vocabulary of metrology – Basic and general concepts and associated terms (VIM). *ISO/IEC Guide 99:2007*, 1:1–92, December 2007.

[ITU99] ITU-T Study Group 15. *Synchronization layer functions.* International Telecommunications Union, Geneva, Switzerland, June 1999. Available from: `http://www.itu.int/rec/T-REC-G.781-199907-S/en`.

[ITU00a] ITU-T Study Group 13. *Architecture of transport networks based on the synchronous digital hierarchy (SDH).* International Telecommunications Union, Geneva, Switzerland, March 2000. Available from: `http://www.itu.int/rec/T-REC-G.803-200003-I/en`.

[ITU00b] ITU-T Study Group 13. *Functional architecture of connectionless layer networks.* International Telecommunications Union, Geneva, Switzerland, March 2000. Available from: `http://www.itu.int/rec/T-REC-G.809-200303-I/en`.

[ITU00c] ITU-T Study Group 13. *Generic functional architecture of transport networks.* International Telecommunications Union, Geneva, Switzerland, March 2000. Available from: `http://www.itu.int/rec/T-REC-G.805-200003-I/en`.

[ITU02] ITU-R Study Group 7. *Standard-frequency and time-signal emissions.* International Telecommunications Union, Geneva, Switzerland, February 2002. Available from: `http://www.itu.int/rec/R-REC-TF.460-6-200202-I/en`.

[ITU03] ITU-T Study Group 15. *Timing characteristics of SDH equipment slave clocks (SEC).* International Telecommunications Union, Geneva, Switzerland, March 2003. Available from: `http://www.itu.int/rec/T-REC-G.813-200303-I/en`.

[ITU04a] ITU-T Study Group 15. *Architecture of Ethernet layer networks.* International Telecommunications Union, Geneva, Switzerland, February 2004. Available from: `http://www.itu.int/rec/T-REC-G.8010-200402-I/en`.

[ITU04b] ITU-T Study Group 15. *Timing requirements of slave clocks suitable for use as node clocks in synchronization networks.* International Telecommunications Union, Geneva, Switzerland, June 2004. Available from: `http://www.itu.int/rec/T-REC-G.812-200406-I/en`.

[ITU07a] ITU-T Study Group 15. *Network node interface for the synchronous digital hierarchy (SDH).* International Telecommunications Union, Geneva, Switzerland, January 2007. Available from: `http://www.itu.int/rec/T-REC-G.707-200701-I/en`.

[ITU07b] ITU-T Study Group 15. *Timing characteristics of synchronous Ethernet equipment slave clock (EEC).* International Telecommunications Union, Geneva, Switzerland, August 2007. Available from: `http://www.itu.int/rec/T-REC-G.8262-200708-I/en`.

[ITU08a] ITU-T Study Group 15. *Distribution of timing through packet networks.* International Telecommunications Union, Geneva, Switzerland, October 2008. Available from: `http://www.itu.int/rec/T-REC-G.8264/en`.

[ITU08b] ITU-T Study Group 15. *Timing and synchronization aspects in packet networks.* International Telecommunications Union, Geneva, Switzerland, April 2008. Available from: `http://www.itu.int/rec/T-REC-G.8261-200804-I/en`.

[Joh04] Svein Johannessen. Time synchronization in a local area network. *IEEE Communications Magazine*, 24(2):61–69, April 2004. `doi:10.1109/MCS.2004.1275432`.

[JSW04] Jürgen Jasperneite, K. Shehab, and K. Weber. Enhancements to the Time Synchronization Standard IEEE-1588 for a System of Cascaded Bridges. In *Proc. IEEE International Workshop on Factory Communication Systems*, pages 239–244, September 2004. `doi:10.1109/WFCS.2004.1377716`.

[jt.08] jt.goh. *M12M Timing Oncore Receiver Datasheet.* i-Lotus Corporation Pte.Ltd., 51 Ayer Rajah Crescent, #06-08. Singapore, 1.1 edition, February 2008.

[Kal60] Rudolf Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME – Journal of Basic Engineering*, 82(Series D):35–45, March 1960. Available from: `http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf`.

[Kal04] Jozef Kalisz. Review of methods for time interval measurements with picosecond resolution. *Metrologia*, 41:17–32, February 2004. Provided by the SAO/NASA Astrophysics Data System. `doi:10.1088/0026-1394/41/1/004`.

[Kir08a] Stephan Kirchmayer. *Ethernet POWERLINK Communication Profile Specification.* Ethernet POWERLINK Standardisation Group, Kurfuerstenstraße 112, 10787 Berlin, Germany, 1.1.0 edition, October 2008.

[Kir08b] Stephan Kirchmayer. *Ethernet POWERLINK Multiple-ASnd.* Ethernet POWERLINK Standardisation Group, Kurfuerstenstraße 112, 10787 Berlin, Germany, 1.0.0 edition, August 2008.

[Kle10] Leonard Kleinrock. An early history of the internet [history of communications]. *IEEE Communications Magazine*, 48(8):26–36, August 2010. `doi:10.1109/MCOM.2010.5534584`.

[KP02] Scott C. Karlin and Larry Peterson. Maximum packet rates for full-duplex ethernet. technical report TR–645–02, Princeton University, February 2002. Available from: `http://www.cs.princeton.edu/research/techreps/TR-645-02`.

[KSPP97] Jozef Kalisz, Ryszard Szplet, Jerzy Pasierbinski, and Andrzej Poniecki. Field-programmable-gate-array-based time-to-digital converter with 200-ps resolution. *IEEE Transactions on Instrumentation and Measurement*, 46(1):51–55, February 1997. `doi:10.1109/19.552156`.

[Lam78] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978. `doi:10.1145/359545.359563`.

[LAN08] LAN Access Division (LAD). *Intel 82576 Gigabit Ethernet Controller Datasheet*. Intel GmbH, 2200 Mission College Blvd., Santa Clara, CA 95054-1549, USA, 2.1 edition, July 2008.

[Lat08] Lattice Semiconductor Corporation, 5555 N.E. Moore Court, Hillsboro, Oregon. *LatticeXP2 Family Handbook*, 01.7 edition, April 2008.

[LBS+03] Julian Lewis, Jean-Claude Bau, Javier Serrano, David Dominguez, and Pablo Alvarez Sanchez. The Evolution of the CERN SPS Timing System for the LHC Era. In *Proceedings of ICALEPCS 2003*, pages 125–129, October 2003. Available from: `http://cdsweb.cern.ch/record/693149`.

[LE09] Robert Leidenfrost and Wilfried Elmenreich. Firefly Clock Synchronization in an 802.15.4 Wireless Network. *EURASIP Journal on Embedded Systems*, 2009. `doi:10.1155/2009/186406`.

[Lee08] Sungwon Lee. An Enhanced IEEE 1588 Time Synchronization Algorithm for Asymmetric Communication Link using Block Burst Transmission. *IEEE Communications Letters*, 12(9):687–689, September 2008. `doi:10.1109/LCOMM.2008.080824`.

[LENG08] Patrick Loschmidt, Reinhard Exel, Anetta Nagy, and Georg Gaderer. Limits of Synchronization Accuracy Using Hardware Support in IEEE 1588. In *Proc. of the International IEEE Symposium on PrecisionClock Synchronization for Measurement, Control and Communication, ISPCS*, pages 12–16, Ann Arbor, USA, September 2008. `doi:10.1109/ISPCS.2008.4659205`.

[LGHD85] Wiliam C. Lindsey, Frazad Ghazvinian, Walter C. Hagmann, and Khaled Dessouky. Network Synchronization. *Proceedings of the IEEE*, 73(10):1445–1467, October 1985. `doi:10.1109/PROC.1985.13317`.

[LGS06] Patrick Loschmidt, Georg Gaderer, and Thilo Sauter. Synchronized Access to Sensor Networks. In *Proc. of the IEEE Sensors Conference*, pages 785–788, Daegu, October 2006. `doi:10.1109/ICSENS.2007.355586`.

[LGS07] Patrick Loschmidt, Georg Gaderer, and Thilo Sauter. Clock Synchronization for Wireless Positioning of COTS Mobile Nodes. In *Proc. of the International IEEE Symposium on PrecisionClock Synchronization for Measurement, Control and Communication, ISPCS*, pages 64–69, Vienna, Austria, October 2007. doi:10.1109/ISPCS.2007.4383775.

[LGS+09] Patrick Loschmidt, Georg Gaderer, Natasa Simanic, Pedro Moreira, and Arshad Hussain. White Rabbit – Sensor/Actuator Protocol for the CERN LHC Particle Accelerator. In Paddy J. French, editor, *Proc. of the IEEE Sensors Conference*, pages 781–786, October 2009. doi:10.1109/ICSENS.2009.5398529.

[Lis93] Barbara Liskov. Practical Uses of Synchronized Clocks in Distributed Systems. *Distributed Computing*, 6:211–219, July 1993. doi:10.1007/BF02242709.

[LKD78] W. Lindsey, A. Kantak, and A. Dobrogowski. Network Synchronization by Means of a Returnable Timing System. *IEEE Transactions on Communications*, 26(6):892–896, June 1978. doi:10.1109/TCOM.1978.1094146.

[LSL84] Algie L. Lance, Wendell D. Seal, and Frederik Labaar. Phase Noise and AM-Noise Measurements in the Frequency Domain. In Kenneth J. Button, editor, *Infrared and Millimeter Waves*, volume 11, chapter 7, pages 239–289. Academic Press, New York, November 1984.

[Lub02] Michael Luby. LT codes. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 271–280, February 2002. doi:10.1109/SFCS.2002.1181950.

[lxi08] LAN eXtensions for Instrumentation (LXI). *LXI Standard*, October 2008. Available from: http://www.lxistandard.org.

[Mar84] Keith Marzullo. *Maintaining the Time in a Distributed System: An Example of a Loosely-Coupled Distributed Service.* PhD thesis, Stanford University, Department of Electrical Engineering, February 1984.

[Mei08] Meinberg Funkuhren GmbH & Co. KG, Lange Wand 9, D-31812 Bad Pyrmont. *Technische Daten Inbetriebname M600 / MRS xmr*, March 2008.

[Men09] Mentor Graphics Corporation, 8005 SW Boeckman Road, Wilsonville, OR 97070, USA. *ModelSim PE User's Manual*, December 2009. Available from: http://supportnet.mentor.com/.

[Mil81a] David L. Mills. DCNET Internet Clock Service. RFC 778, Internet Engineering Task Force, April 1981. Available from: http://www.ietf.org/rfc/rfc778.txt.

[Mil81b] David L. Mills. Time Synchronization in DCNET Hosts. IEN 173, Internet Engineering Task Force, February 1981. Available from: `http://www.networksorcery.com/enp/ien/ien173.txt`.

[Mil85] David L. Mills. Network Time Protocol (NTP). RFC 958, Internet Engineering Task Force, September 1985. Obsoleted by RFCs 1059, 1119, 1305. Available from: `http://www.ietf.org/rfc/rfc958.txt`.

[Mil88] David L. Mills. Network Time Protocol (Version 1) Specification and Implementation. Technical Report 1059, Internet Engineering Task Force, July 1988. Obsoleted by RFCs 1119, 1305. Available from: `http://www.ietf.org/rfc/rfc1059.txt`.

[Mil89] David L. Mills. Network Time Protocol (Version 2) Specification and Implementation. Technical Report 1119, Internet Engineering Task Force, September 1989. Obsoleted by RFC 1305. Available from: `http://www.ietf.org/rfc/rfc1119.txt`.

[Mil91] David L. Mills. Internet Time Synchronization: The Network Time Protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991. `doi:10.1109/26.103043`.

[Mil92a] David L. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. Technical Report 1305, Internet Engineering Task Force, March 1992. Available from: `http://www.ietf.org/rfc/rfc1305.txt`.

[Mil92b] David L. Mills. Simple Network Time Protocol (SNTP). Technical Report 1361, Internet Engineering Task Force, August 1992. Obsoleted by RFC 1769. Available from: `http://www.ietf.org/rfc/rfc1361.txt`.

[Mil06a] David L. Mills. *Computer Network Time Synchronization: The Network Time Protocol*. CRC Press, Inc., Boca Raton, FL, USA, 2006.

[Mil06b] David L. Mills. Network Time Protocol Version 4 Reference and Implementation Guide. Technical Report 06-6-1, University of Delaware, June 2006. Available from: `http://www.eecis.udel.edu/~mills/database/reports/ntp4/ntp4.pdf`.

[Mil06c] David L. Mills. Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI. Technical Report 4330, Internet Engineering Task Force, January 2006. Available from: `http://www.ietf.org/rfc/rfc4330.txt`.

[MK00] David L. Mills and Poul-Henning Kamp. The Nanokernel. In *Proc. of the Precise Time and Time Interval (PTTI) Systems and Applications Meeting*, pages 423–430, November 2000. Available from: `http://www.pttimeeting.org/archivemeetings/2000papers/paper33.pdf`.

[MO83] Keith Marzullo and Susan Owicki. Maintaining the Time in a Distributed System. In *Proc. of the second annual ACM symposium on Principles of distributed computing, PODC*, pages 295–305, New York, NY, USA, 1983. ACM. `doi:http://doi.acm.org/10.1145/800221.806730`.

[MSW$^+$09] Pedro Moreira, Javier Serrano, Tomasz Wlostowski, Patrick Loschmidt, and Georg Gaderer. White Rabbit: Sub-Nanosecond Timing Distribution over Ethernet. In *Proc. of the International IEEE Symposium on PrecisionClock Synchronization for Measurement, Control and Communication, ISPCS*, pages 58–62, Brescia, Italy, October 2009. `doi:10.1109/ISPCS.2009.5340196`.

[Nat05] National Instruments Corporation, 11500 North Mopac Expressway Austin, Texas, USA. *NI PCI-1588 User Manual*, October 2005.

[Nat07] National Instruments Corporation, 11500 North Mopac Expressway Austin, Texas, USA. *NI PXI-6682 User Manual*, October 2007.

[Nat08] National Semiconductor Corporation, 2900 Semiconductor Dr., P.O. Box 58090, Santa Clara, California, USA. *DP83640 Precision PHYTER - IEEE 1588 Precision Time Protocol Transceiver*, February 2008.

[ntp09] *The Network Time Protocol (NTP) Distribution*, November 2009. Available from: `http://www.eecis.udel.edu/~mills/ntp/html/index.html` [cited December 2009].

[OLS08] Patrick Ohly, David N. Lombard, and Kevin B. Stanton. Hardware Assisted Precision Time Protocol. Design and case study. In *Proc. of the 9th LCI International Conference on High-Performance Clustered Computing.* Intel GmbH, April 2008.

[OS09] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing.* Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, August 2009.

[Ple05] Dan Pleasant. LXI Triggering. White Paper 2, Agilent Technologies, December 2005.

[Pos80] Jonathan Postel. User Datagram Protocol. Technical Report 768, Internet Engineering Task Force, August 1980. Available from: `http://www.ietf.org/rfc/rfc768.txt`.

[pow08] POWERLINK Basics. brochure, Ethernet POWERLINK Standardisation Group, Kurfuerstenstraße 112, 10787 Berlin, Germany, December 2008.

[Pry08] Gunnar Prytz. A performance analysis of EtherCAT and PROFINET IRT. In *Proc. of the International IEEE Conference on Emerging Technologies and Factory Automation, ETFA*, pages 408–415. ABB AS Corporate Research Center, September 2008. `doi:10.1109/ETFA.2008.4638425`.

[PXI04] PXI Systems Alliance. Hardware Specification. *PXI*, Rev. 2.2, September 2004.

[QK06] Jürgen Quade and Eva-Katharina Kunst. *Linux-Treiber entwickeln*. Dpunkt.Verlag, Ringstraße 19 B, 69115 Heidelberg, 2 edition, May 2006.

[Red02] Reduced Gigabit Media Independent Interface (RGMII). Technical Report 2.0, Hewlett Packard, January 2002.

[RMI98] RMII specification. Technical Report 1.2, RMII consortium, March 1998.

[Ros06] David Rosselot. DP83848 and DP83849 100Mb Data Latency. Application Note 1507, National Semiconductor, August 2006.

[Rub05] Enrico Rubiola. The Leeson Effect – Phase Noise in Quasilinear Oscillators. *ArXiv Physics e-prints*, page 117, February 2005. Available from: `http://arxiv.org/abs/physics/0502143v1`.

[SAHW90] D. B. Sullivan, D. W. Allan, D. A. Howe, and F.L. Walls. Characterization of Clocks and Oscillators. Technical Note NIST/TN-1337, National Institute of Standards and Technology, March 1990.

[Sau10] Thilo Sauter. The Three Generations of Field-Level networks – Evolution and Compatibility Issues. *IEEE Transactions on Industrial Electronics*, 57(11):3585–3595, November 2010. `doi:10.1109/TIE.2010.2062473`.

[Sch72] Ralph O. Schmidt. A New Approach to Geometry of Range Difference Location. *IEEE Transactions on Aerospace and Electronic Systems*, AES-8(6):821–835, November 1972. `doi:10.1109/TAES.1972.309614`.

[Sch96] Ralph O. Schmidt. Least Squares Range Difference Location. *IEEE Transactions on Aerospace and Electronic Systems*, 32(1):234–242, January 1996. `doi:10.1109/7.481265`.

[Sch07] Kai Uwe Schmidt. IEEE 1588 on Windows XP Powered Measurement Devices – Mastering the Trigger Challenge. In *Proc. of the International IEEE Symposium on PrecisionClock Synchronization for Measurement, Control and Communication, ISPCS*, pages 92–95, October 2007. `doi:10.1109/ISPCS.2007.4383779`.

[Sch09] Stefan Schwalowsky. System Integration of the ε-WiFi indoor localization system based on IEEE 802.11 WLAN. Master's thesis, Ostwestfalen-Lippe University of Applied Sciences, Lemgo, Germany, November 2009.

[ser07] SERCOS III: Universal Real-Time Communication with Ethernet. presentation, SERCOS International e.V., Kueblerstr. 1, 73079 Suessen, Germany, November 2007. Available from: `http://www.sercos.com/literature/literature.htm#2`.

[SGAK06] Klaus Steinhammer, Petr Grillinger, Astrit Ademaj, and Hermann Kopetz. A Time-Triggered Ethernet (TTE) Switch. In *Proceedings of Design, Automation and Test in Europe, DATE*, volume 1, pages 1–6, March 2006. `doi:10.1109/DATE.2006.244145`.

[Sie08] Siemens, Würzburgerstr. 121, 90766 Fürth, Federal Republic of Germany. *ERTEC 200 Enhanced Real-Time Ethernet Controller*, 1.1.1 edition, November 2008.

[SJH01] Tor Skeie, Svein Johannessen, and Oyvind Holmeide. Highly Accurate Time Synchronization over Switched Ethernet. In *Proc. of the International IEEE Conference on Emerging Technologies and Factory Automation, ETFA*, volume 1, pages 195–204, 2001. `doi:10.1109/ETFA.2001.996369`.

[SKS00] Ryszard Szplet, Jozef Kalisz, and Rafal Szymanowski. Interpolating Time Counter with 100 ps Resolution on a Single FPGA Device. *IEEE Transactions on Instrumentation and Measurement*, 49(4):879–883, August 2000. `doi:10.1109/19.863942`.

[SMS09] SMSC, 80 Arkay drive, Hauppauge, NY 11788, USA. *SMSC LAN8700/LAN8700i Datasheet*, 2.1 edition, June 2009. Available from: `http://www.smsc.com/main/catalog/lan8700.html`.

[SRS05] SRS Stanford Research Systems, 1290-D Reamwood Avenue, Sunnyvale, California 94089, USA. *FS725 Rubidium Frequency Standard*, 1.1 edition, November 2005. Available from: `http://www.thinksrs.com/downloads/PDFs/Manuals/FS725m.pdf`.

[SRS06] SRS Stanford Research Systems, 1290-D Reamwood Avenue, Sunnyvale, California 94089, USA. *MODEL SR620 Universal Time Interval Counter*, 2.7 edition, February 2006. Available from: `http://www.thinksrs.com/downloads/PDFs/Manuals/SR620m.pdf`.

[SSC+09] Javier Serrano, Pablo Alvarez Sanches, M. Cattin, E. Garcia Cota, J.H.Lewis, P. M. Oliveira Fernandes Moreira, Tomasz Wlostowski, Georg Gaderer, Patrick Loschmidt, Joze Dedic, Ralph Baer, Tibor Fleck, Matthias Kreider, Cesar Prados, and S. Rauch. The White Rabbit Project. In *Proc. of the 2009 IEEE International Conference on Accelerator and Large Experimental Physics Control Systems, ICALEPCS*, Kobe, Japan, October 2009. Available from: `http://accelconf.web.cern.ch/AccelConf/icalepcs2009/papers/tuc004.pdf`.

[Ste85] Samuel R. Stein. Frequency and Time – Their Measurement and Characterization. In Eduard A. Gerber and Arthur Ballato, editors, *Precision Frequency Control*, volume 2, chapter 12, pages 191–232. Academic Press, New York, July 1985. Available from: `http://ts.nist.gov/MeasurementServices/Calibrations/upload/PFC-2.PDF`.

[Ste08] W. Steiner. *TTEthernet Specification.* TTTech Computertechnik AG, Schönbrunner Straße 7, 1040 Vienna, Austria, 0.9.1 edition, November 2008.

[STG^{+}09] Stefan Schwalowsky, Henning Trsek, Georg Gaderer, Reinhard Exel, and Nikolaus Kerö. Lokalisierung in IEEE 802.11 WLANs durch hochgenaue Uhrensynchronisation und TDoA Messungen. In Jörg F. Wollert, editor, *Wireless technologies: von der Technologie zur Anwendung; 11. Kongress*, pages 162–172, Düsseldorf, September 2009. VDI Verlag GmbH.

[Tel04] Telecommunications and Timing Group (TTG) of the Range Commanders Council (RCC). IRIG serial time code formats. *IRIG Standard 200-04 (update of IRIG Standard 200-98)*, September 2004. Available from: `https://wsmrc2vger.wsmr.army.mil/rcc/PUBS/pubs.htm`.

[The09] The VINT Project. *The ns Manual.* UC Berkeley, LBL, USC/ISI, and Xerox PARC, 1 Cyclotron Road, Berkeley, CA 94720, USA, May 2009. Available from: `http://www.isi.edu/nsnam/ns/ns-documentation.html`.

[TNYH06] R.Y.S. Tee, T.D. Nguyen, Lie-Lang Yang, and Lajos Hanzo. Serially Concatenated Luby Transform Coding and Bit-Interleaved Coded Modulation Using Iteratlive Decoding for the Wireless Internet. In *63rd IEEE Vehicular Technology Conference*, volume 1, pages 22–26, Melbourne, Australia, May 2006. `doi:10.1109/VETECS.2006.1682768`.

[Tri01] Trimble Navigation Limited, 645 North Mary Avenue, Post Office Box 3642, Sunnyvale, CA 94088-3642, USA. *Acutime 2000 Synchronization Kit User Guide*, a edition, April 2001.

[TTE09] TTEthernet – A Powerful Network Solution for All Purposes. Technical report, TTTech Computertechnik AG, Schönbrunner Straße 7, 1040 Vienna, Austria, May 2009. Available from: `http://www.tttech.com/fileadmin/content/white/TTEthernet/TTEthernet_Article.pdf`.

[Vai90] P.P. Vaidyanathan. Multirate digital filters, filter banks, polyphase networks, and applications: a tutorial. *Proceedings of the IEEE*, 78(1):56–93, January 1990. `doi:10.1109/5.52200`.

[Var99] Andras Varga. Using the OMNeT++ Discrete Event Simulation System in Education. *IEEE Transactions on Education*, 42:11, November 1999. `doi:10.1109/13.804564`.

[VB99] John R. Vig and Arthur Ballato. *Ultrasonic Instruments and Devices II*, volume XXIV of *Physical Acoustics*, chapter Frequency Control Devices, pages 209–269. Academic Press, Inc., 525 B Street, Suite 1900, San Diego, CA 92101-4495, USA, 1999.

[Vig92] John R. Vig. Introduction to Quartz Frequency Standards. Technical Report SLCET-TR-92-1 (Rev. 1), Army Research Laboratory, Electronics and Power Sources Directorate, Fort Monmouth, NJ 07703-5601, USA, October 1992. Available from: `http://handle.dtic.mil/100.2/ADA256373`.

[Vig04] John R. Vig. Quartz Crystal Resonators and Oscillators for Frequency Control and Timing Applications – A Tutorial. Technical Report SLCET-TR-88-1 (Rev.8.5.2), U.S. Army Communications-Electronics Command, Fort Monmouth, NJ 07703, USA, January 2004. Available from: `http://indeedzcr.googlepages.com/Vig-tutorial8.5.2.0.pdf`.

[VRC97] Paulo Veríssimo, Luís Rodrigues, and Antonio Casimiro. CesiumSpray: a Precise and Accurate Global Time Service for Large-scale Systems. *Real-Time Systems*, 12:243–294, March 1997. `doi:10.1023/A:1007949113722`.

[Vv09] Pavel Vyskočil and Jiří Šebesta. Relative timing characteristics of gps timing modules for time synchronization application. In *Proc. of the International Workshop on Satellite and Space Communications, IWSSC*, pages 230–234, September 2009. `doi:10.1109/IWSSC.2009.5286378`.

[WATP$^+$05] Geoffrey Werner-Allen, Geetika Tewari, Ankit Patel, Matt Welsh, and Radhika Nagpal. Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects. In *Proc. of the 3rd international conference on Embedded networked sensor systems, SenSys*, pages 142–153, New York, NY, USA, 2005. ACM. `doi:10.1145/1098918.1098934`.

[WL88] Jennifer Lundelius Welch and Nancy Lynch. A New Fault-Tolerant Algorithm for Clock Synchronization. *Information and Computation*, 77(1):1–36, 1988. `doi:10.1016/0890-5401(88)90043-0`.

[Xu07] Guochang Xu. *GPS – Theory, Algorthms and Applications*. Springer-Verlag, Berlin, 2nd edition, 2007.

[YOU68] Junji Yamato, Masamichi Ono, and Shogo Usuda. Synchronization of a PCM Integrated Telephone Network. *IEEE Transactions on Communication Technology*, 16(1):1–11, February 1968. `doi:10.1109/TCOM.1968.1089819`.

[YOU06] Yoshiaki Yamada, Satoru Ohta, and Hitoshi Uematsu. Hardware-Based Precise Time Synchronization on Gb/s Ethernet Enhanced with Preemptive Priority. In *IEICE Transactions on Communications*, volume E89-B, pages 683–689, March 2006.

[ZSYZ08] Xiangwei Zhu, Guangfu Sun, Shaowei Yong, and Zhaowen Zhuang. A High-Precision Time Interval Measurement Method Using Phase-Estimation Algorithm. *IEEE Transactions on Instrumentation and Measurement*, 57(11):2670–2676, November 2008. `doi:10.1109/TIM.2008.925025`.