# In-band Time Synchronization in Real-Time systems

by

V.J.F.R. Waegenaere

to obtain the degree of Master of Science

in

Computer Engineering

at the Delft University of Technology,

to be defended publicly on Wednesday June 15, 2022 at 11:00 AM.

| | | |
|---|---|---|
| Student number: | 4384504 | |
| Project duration: | February 1, 2021 – June 15 , 2022 | |
| Thesis committee: | | |
| Advisor: | Dr. S.D. Cotofana, | TU Delft (Computer Engineering Laboratory) |
| Chairperson | Dr. S.D. Cotofana, | TU Delft (Computer Engineering Laboratory) |
| Member | Prof. dr. P.J. French, | TU Delft (Bioelectronics Laboratory) |
| Member | A. van Herk, M.Sc., | Prodrive Technologies |

*This thesis is confidential and cannot be made public until June 15, 2025.*



TU Delft Delft University of Technology

# Abstract

Real-time (RT) systems are widespread over different industries, e.g., healthcare, robotics, manufacturing, machine vision, etc. These systems consist of a hardware and software part that execute an RT application. These systems require a bounded and predictable time response on incoming events and execute all input and output (IO) tasks simultaneously. To ensure this concurrent behavior, the different IO devices are synchronized to achieve a common time notion. The implementation of time notion in systems can differ and therefore various types of synchronization exist, e.g. in-band and out-of-band. In-band synchronization utilizes the general communication channel to distribute time information, contrary to out-of-band synchronization which uses an external wire to transfer the time signals. As the in-band type implies the synchronization transactions flow together with the general traffic, the synchronization mechanisms are often implemented in the interconnect protocol. This integration limits the choice for a feasible synchronization protocol as this choice is dependent of that of the interconnect due to restricting communication requirements of RT applications. Current synchronization protocols are often limited to sub-microsecond (µs) latency variations and/or partially rely on an out-of-band principle. The goal of this master thesis is to find a solution which is able to achieve in-band synchronization with nanosecond (ns) range jitter. The proposed design is optimized towards nanosecond-scale jitter, by taking implementation challenges of Precision Time Protocol (PTP) into account, and is separate from the choice of interconnect. PTP is an existing synchronization mechanism which retrieves the differences in time notion (offsets) across multiple devices while accounting for transmission latency to each individual device. The implementation of PTP can result in limited performance in terms of jitter. The design focuses on minimizing this jitter with increasing the accuracy and robustness of the PTP synchronization algorithm by improving the precision of timestamps and filtering the calculated offsets for outliers. The synchronization mechanism was evaluated through simulation and validation in hardware. This master thesis presents a Proof of Concept (PoC) that can be implemented into real-world RT systems. It consists of two devices synchronizing to one reference device using the proposed synchronization mechanism. The PoC achieves down to 7 ns jitter, which was not reached by feasible existing in-band synchronization yet.

# Contents

# List of Figures

# List of Tables

# Acknowledgements

There are a number of people that I would like to thank for their help during my graduation project. First of all my family and friends that always supported me, not only during this final part of my master but throughout my entire study. In the course of this project I had great colleagues which helped bring this project to a good end by providing me with feedback, support, knowledge. Lastly I would thank both my TU Delft advisor, Dr. Sorin Cotofana, and advisor from Prodrive Technologies, André van Herk M.SC., for their help and guidance during my journey of my graduation project.

*V.J.F.R. Waegenaere*
*Delft, The Netherlands*
*June 2022*

# 1 Introduction

Real-time (RT) systems are implemented worldwide in many different application fields build on a combination of software and hardware. Examples of RT applications are amongst others in healthcare [6, 7], high-end control networks [8, 9] or signal processing devices [10, 11].

Their functioning consists of acting on incoming events with an appropriate response within a given period of time. This response is first computed and secondly executed by output devices. However, the desired behavior of RT systems is not only dependent on the correctness of its computational results but also on the time in which these results are available [12]. In other words, each system has a restricted time interval in which it needs to act on incoming data, such that the system properly acts within the same environment that it sampled [13]. This requirement is called *timeliness* and is defined as the bounded and predictable time response of an RT system. The second RT property is *time synchronization*, which is the ability of distributed entities having their own clocks to simultaneously execute tasks [14].

The different devices keep a local time to execute their tasks at a predefined moment. However, because the actual frequency of all the local oscillators (components used in keeping time) slightly differ, the devices drift apart in terms of time. Similarly, fluctuations in the rotational speed of the Earth result in the Coordinated Universal Time (UTC) and Universal Time (UT1) standards to drift apart. To ensure that this drift is kept within limits, a leap second is introduced when the time difference reaches 0.9 second [15, 16]. This action is called *synchronization* and it is meant to limit the time drift to less than one second. Figure 1.1 depicts the synchronization of UTC to keep its drift in relation to UT1 within bounds.

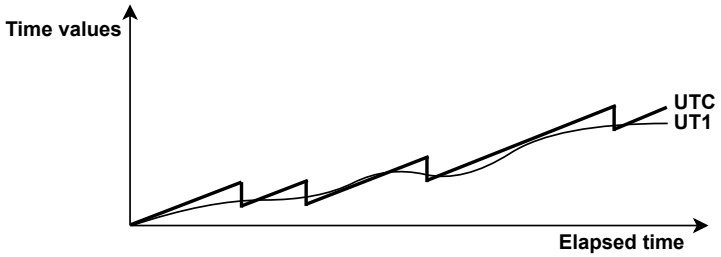


*Figure 1.1: UTC vs UT1 [1]*

Two entities are defined as synchronized if they have the same elapsed time notion with a bounded skew [2]. This relation between different devices is needed to achieve coordinated actions as small time differences can eventually lead to large time deviations. Therefore synchronization is needed to keep the time differences bounded. This is achieved with the use of different mechanisms, which can be divided into two categories: out-of-band and in-band.

The out-of-band approach requires an additional wire which is connected to all devices. This wire is used to distribute periodic pulses, i.e., a global clock , which are used as a time reference throughout the whole network. Although the implementation is simple and straightforward, it adds hardware with the cabling. This is a disadvantage due to cost and lack of flexibility, certainly for large systems. Additionally, a clock skew is introduced due to the physical distance between its source and destinations.

In contrast to out-of-band methods, in-band techniques use the existing communication infrastructure to transmit synchronization information. Consequentially, these synchronization transactions are inserted between normal data transfers. While not requiring any extra hardware resources such methods face an extra challenge as the synchronization data arrival time can vary due to regular traffic causing congestion. Each in-band mechanism copes with this differently, which results in different accuracies [2, 17, 18]. While this type of implementation adds complexity in the device designs, it brings big advantages in terms of scaling, freedom in network design, and cost.

Because in-band synchronization utilizes shared communication channels, its functionality is often incorporated in the network interconnect, which is used to link devices together in networks [19–23]. Because of this incorporation, the communication protocol has influence on the application performance. Some systems use multiple interconnects with varying properties. This adds extra complications, such as compatibility and transmission latency. Therefore it is beneficial to design the synchronization mechanism independent of the network interconnection.

## 1.1. Problem Statement

The research scope of this thesis is to achieve low-nanosecond (ns range) jitter for in-band time synchronization. Although out-of-band synchronization is a more straightforward and easy implementable solution, the goal of this research is to address and challenge the disadvantages of an in-band approach. This makes it possible to be applied in constrained embedded RT systems that cannot support out-of-band synchronization. Although there are a number of in-band synchronization mechanisms, these have a limited accuracy of sub-microsecond scale, are incorporated in the interconnect protocol or are a hybrid between in-band and out-of-band. Currently, the time requirements within RT systems increase, which causes decreased budget for jitter and skew, therefore new solutions have to be found [2, 19, 24]. All requirements set for this research are described in Section 1.2. Based on these requirements, the main research question is formulated and divided into sub-questions:

- **How much jitter reduction can be achieved with a redesigned in-band synchronization mechanism?**
  - Can the principles of existing synchronization protocols be used?
  - What limits the performance of existing mechanisms?
  - How can we improve these limitations to reduce the jitter and skew?
  - What are the limitations of the proposed mechanism?

## 1.2. Requirements

This section presents the requirements set for the research into in-band synchronization in real-time systems. These requirements are determined based on the current and future RT application fields. Although the exact values are an indication for the upper bound of the performance, they correspond with the target requirements set by the stakeholders.

- **System Requirements**
  - The synchronization shall be in-band.
  - The synchronization should be independent of the choice of interconnect.
- **Performance Requirements**
  - The absolute synchronization jitter shall be less than 20 ns per endpoint.
  - The cycle-to-cycle synchronization jitter shall be less than 10 ns per endpoint.

## 1.3. Contributions

This thesis introduces a novel way to achieve low-nanosecond jitter in in-band time synchronization by providing a solution to a frequently occurring weakness of the widely used Precision Time Protocol (PTP) [2, 17], namely asymmetric transmission latency. Compared to earlier solutions, this design only uses in-band synchronization. This brings big advantages in terms of scaling, freedom in network design, and cost in contrast to out-of-band solutions [25].

Based on literature research, a basic PTP mechanism design was implemented, as this presents the best potential in terms of accuracy. The goal of this first design step was to have a baseline mechanism where the behavior could be observed. This was done with the use of a testbench that could be adapted iteratively to mimic the system environment. The results of these simulations presented the weaknesses that were identified during the literature study, namely the errors in the offset calculations due the inaccurate timestamp generation and the lack in robustness against asymmetric transaction latencies.

To address these main weaknesses and improve the performance of the synchronization, two specific functionalities are added to the basic design, a dedicated trigger interface and an offset filter. Firstly, the goal of the dedicated interface is to improve the precision in which the time can be latched in addition to increasing the symmetry of the transmission latency. This is accomplished by this interface being able to request high-priority messages to be sent and to receive the trigger of the departure and arrival of such messages. The second improvement proposed in this design is the addition of a filter block after the offset calculation. Although the accuracy of the timestamps is increased, the synchronization still assumes an symmetric transmission latency. To minimize the effect of incorrect offsets due to this assumption, the filter block is added. In this thesis two types of filters, the division-by-constant and the sliding window averaging, are evaluated to see which of them is most suitable for being implemented in the proposed synchronization design.

The design is implemented in a *Proof of Concept* (PoC), consisting of one device which keeps the reference time and two adjacent devices which need to synchronize with the reference. The performance of the proposed protocol in this PoC set-up is down to 7 ns jitter. We note that no state-of-the-art solution can provide such an accuracy by means of solely in-band synchronization protocol.

The process/work-flow of this thesis is built iteratively. The complete work-flow is visualized in Figure 1.2. It starts with the main research question steering the literature research towards the existing solutions and uncovering their advantages and weaknesses. Based on this research a potential solution is designed which goes through different steps, i.e., conceptualization, implementation, verification and validation. The behavior and requirements are checked in different design phases with a path back to an earlier phase when a check fails. The thesis process finishes with the proposed design meeting the requirements and the research questions are answered.

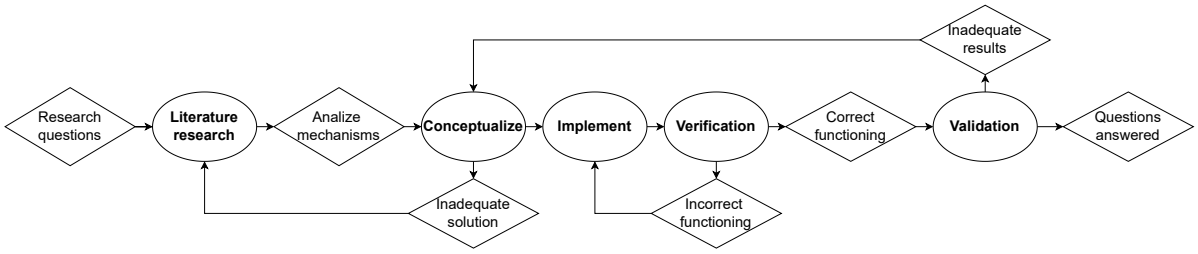


*Figure 1.2: Methodological workflow*

## 1.4. Thesis organization

Chapter 2 gives the basic fundamentals of real-time systems and synchronization. Chapter 3 discusses current literature covering different existing in-band synchronization methods. Chapter 4 explains the new mechanism by elaborating on the principle this design is based on. In addition, the main components of the synchronization system are further elaborated. Chapter 5 presents the simulations of an important functional block, the filter, and of the full setup, followed by the results of both simulations. Finally, Chapter 6 concludes the thesis contribution and discusses potential future work. Appendices show background details.

# 2

# Background

As discussed in the introduction, RT systems are defined as systems with requirements regarding timeliness and time synchronization. This chapter first elaborates on the network level and presents the different types of devices in RT systems. After that, extra background is provided on the mathematical inside of synchronization, which allows to better understand the different types of mechanisms.

## 2.1. Basic fundamentals of real-time systems

RT systems consist of computational (e.g., CPUs, GPUs, DSPs), Input/Output (IO) (e.g., cameras, sensors, actuators), and network devices (e.g., break-out boxes, routers, switches). Depending on the requirements, some RT networks allow communication only in a specific way (e.g., from compute to IO and vice versa), whereas others do not have any restriction and allow traffic in all directions (e.g., from IO to IO). An generic example of an RT network is depicted in Figure 2.1. This set-up consists of two compute units, which can communicate to different IO devices through one or more network devices. The IO devices execute specific tasks to provide the appropriate RT system behavior.

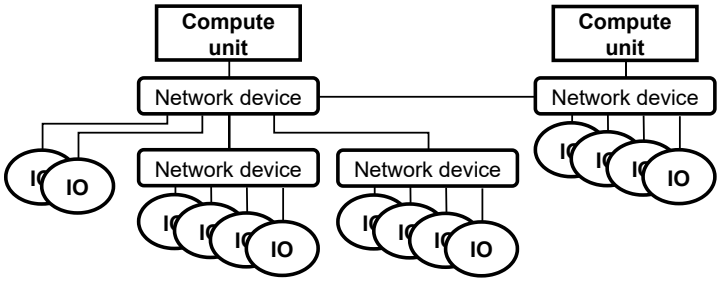


*Figure 2.1: RT system example*

The different tasks that are executed during the operation of RT systems can be divided into an order of actions. The exact set of actions can differ based on the RT application. An example sequence of an RT hardware implementation is illustrated in Figure 2.2 and described in the following steps:

- The sensors sample their environment;
- The measured data are sent to (the) dedicated compute device(s);
- The compute device(s) calculate(s) the setpoints toward the desired state;
- Each setpoint is transfered to its corresponding actuator;
- The actuators execute their action.

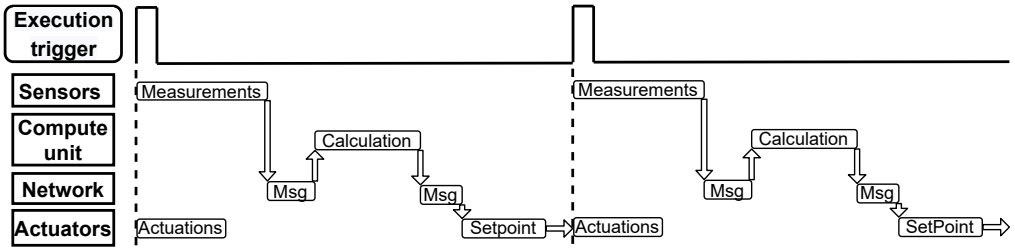


*Figure 2.2: Example of an RT cycle*

## 2.2. Basic fundamentals of synchronization

To ensure all the RT tasks are executed simultaneously, periodic triggers are used, called *heartbeats*. These are produced by each device based on its own notion of time. Although in many applications the clock signal has a wave form (with a duty cycle of 50%), in its more general form it can be described as a pulse train with variable duty cycles. Just like all periodic functions, the pulse train can be mathematically represented as a Fourier series [26] which describes any periodic functions as a summation of $cos()$ and $sin()$ functions as indicated in Equation (2.1):

$$y(t) = a_0 + \sum_{n=1}^{\infty} a_n * cos[\omega * n * (t+\theta)], \tag{2.1}$$

where:

$\omega = 2 * \pi * f$ = Angular frequency [rad/s]
$\theta$ = Phase shift [rad]

The coefficients $a_0$ and $a_n$ are presented in Equation (2.2) and Equation (2.3), respectively. Both are expressed using the duty cycle ($d$) and the amplitude ($A$). While the amplitude is one of the main properties of the pulses, it does not have any purpose during clock synchronization. For the duty cycle, the importance depends on the interpretation of the signal in terms of edge triggering.

$$a_0 = d * A, \tag{2.2}$$

$$a_n = \frac{2 * A}{n * \pi} * sin(d * n * \pi), \tag{2.3}$$

where:

$A$ = Amplitude
$d$ = Duty cycle, $0 \leqslant d \leqslant 1$

The clock synchronization process consists of three categories, namely frequency, phase, and time synchronization. These categories are related to each other, each step adding information and providing guarantees [24].

- **Frequency synchronization** is presented in Figure 2.3 In this case, source B should get the same frequency $f$ as source A, while a certain phase shift is acceptable. This phenomenon is also called *syntonization* [2].

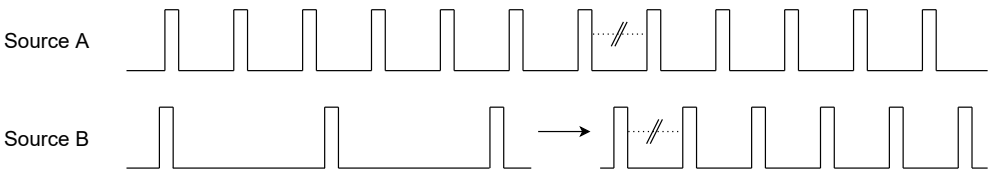


*Figure 2.3: Frequency synchronization*

- **Phase synchronization** has as purpose to adjust the phase $\theta$ of source B to match with the phase of the signal of source A such that all signal events occur simultaneously. This type of synchronization is usually done between clocks that have an integer multiple frequency with respect to each other. Figure 2.4 shows the sources already have the same frequency and after synchronization the pulses align with each other.

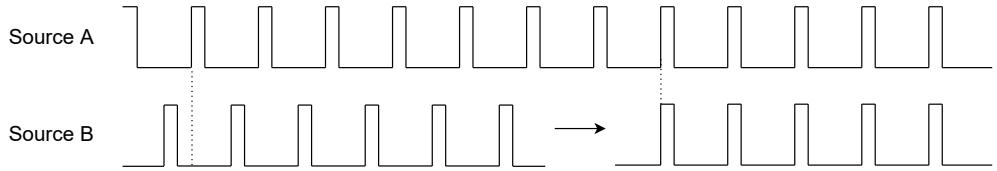


*Figure 2.4: Phase synchronization*

- **Time synchronization** synchronizes the time notion instead of matching the characteristics of the clock signals like the previous two types do. It utilizes a local counter which keeps count of the elapsed clock cycles and indicates not only time events (incrementation of the count), but also a point in time with its value, called the *count*. In time synchronization the aim is to have the same count throughout the network. As a result, this corresponds to matching the moment the time started, called the *epoch* [2]. For this method the different clocks, which drive the counters, are assumed to have a limited difference in frequency. In other words, all frequencies are set to be the same, however the production and environmental variations need to be rectified.

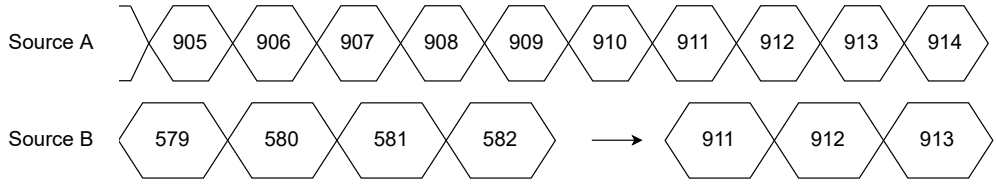


*Figure 2.5: Time synchronization*

## 2.3. Conclusion

Real-time systems require strict synchronization to ensure all tasks are executed simultaneously. This synchronization can be applied on certain parameters of the heartbeat signal (i.e., frequency, phase and time). With this background information in mind, state-of-the-art in-band synchronization is presented in the next chapter.

# 3

# State of the art in synchronization

This chapter elaborates on state-of-the-art RT system synchronization mechanisms. First a general selection is given of the current field of existing protocols. Next, the PTP standard is discussed in more detail as it serves as the basis for this thesis' proposed design.

## 3.1. Synchronization mechanisms

The approach of synchronizing devices in a network is called a synchronization mechanism. These strategies can be divided into event-based or time-based synchronization. These two categories are elaborated further and examples are provided.

Event-based synchronization is a specific type of synchronization protocols in which the time is only kept by a central device and triggering the rest of the network with dedicated messages to synchronize them. The arrival of these packets is used as a reference for local functionalities. For this type of protocols to work and achieve their optimal performance, the communication should be deterministic, meaning transactions have a predictable duration. In addition to this, the latency should also be low. These properties ensure the events keep their time relevance, resulting in simultaneous execution of tasks throughout the network. In the next paragraphs, different event-based procedures are presented which range in complexity and performance.

The simplest of these implementations uses the arrival event of a dedicated message directly, without additional computation. Because of its simplicity, it is vulnerable to variance in transmission latency and different path lengths due to the network topology. It is important to keep the relationship between the moment of the dedicated message departure and arrival correlated. To achieve this association, multiple mitigation strategies can be applied. For example, prioritizing the event transfers or adding a constant delay before the execution of the task to compensate for traffic load and link length, respectively. Although these precautions improve the performance significantly, this mechanism is not robust and reliable.

Clock recovery [27, 28] is a more advanced event-based method, which attempts to retrieve the reference frequency of a central device which corresponds with the frequency in which the tasks need to be executed. There are multiple approaches to recover the intended frequency. *Jitter timestamp* (JTS) [29] is one of these, aimed at removing the jitter of the arriving packages in *Variable bit-rate* (VBR) transmissions. The source sends time indications and based on these, in combination with the variance between the actual and expected arrival of the packets, the reference clock is recovered. Although it seems to recover the reference frequency, it actually recovers the source rate. This gives the number of packets per second.

More information can be added to the transaction to increase the precision. By incorporating the departure timestamp into the message, the receiver is able to extrapolate the ratio of its own clock to that of the sender. In addition to the static network topology, the dynamic traffic results in variable arrival of these packets. This problem can be formulated in a *regression through the origin* representation. Two solutions are proposed by K. S. Kim in the paper *"Asynchronous Source Clock Frequency Recovery through Aperiodic Packet Streams"* [30], namely *least square* and *cumulative ratio*. In contrast to JTS, the source frequency is recovered instead of the number of packets sent per second.

Contrary to event-based synchronization, time-based synchronization uses the data from the dedicated messages to establish a synchronized network [3]. Although this difference adds complexity in terms of interpreting data, it increases the accuracy.

A widely used protocol to share time through a network is the Network Time Protocol (NTP). NTP divides the network in a hierarchy of accuracy levels, *stratums*. These levels get their time from a higher level and distribute it to the lower ones [18]. The exchange is based on Cristian's algorithm [31]. This algorithm is executed by a central entity which checks the time of all lower level entities and compares the time of these devices to that of its own. It sends each of the lower entities the correction for their time. As the protocol does not take transmission latency into account, this adds weakness to it.

To deal with this weakness, Precision Time Protocol (PTP) [2, 17] can be used. It is mainly implemented over Ethernet to synchronize the local clocks of each device. A handshake type of message exchange is used, which removes the transmission latency from the calculation. In this way each device determines its actions based on its local clock instead of waiting for an event-message. The protocol uses a number of transactions to find the offset between the local time and a reference time. The protocol takes the transmission latency into account but assumes it to be symmetric, i.e., the delay from device A to B is the same as from B to A. As mentioned before, PTP is mostly used over Ethernet, which comes with reliability concerns for RT systems. The way packets can be dropped, and retransmitted, does not meet the requirements of most RT systems [3].

## 3.2. Industrial implementations

In this section, different popular RT interconnect protocols are analyzed in terms of synchronization method. It focuses on the way the protocols incorporates the synchronization functionality and not on other properties. Each communication protocol has a different strategy of achieving synchronization.

RapidIO [32] is a frequently used interconnect protocol. One of the synchronization methods of this protocol the utilization of Multicast-Event Control Symbols (MECS). These symbols have the highest priority and can be incorporated in other packages to minimize the transmission latency. They can be used directly as triggers, assuming the transmission latency is constant, or for a more advanced mechanism such as PTP [33].

Peripheral Component Interconnect Express (PCIe) [22] is a high-speed, packet-based, serial communication standard. It is commonly used for communication between CPU, GPU, memory, etc. Although this interconnect is often used between components in a single device, it is also possible to implement it as inter-device communication in larger networks.

In early 2013, an engineering change note (ECN) presented Precision Time Measurement (PTM) [23], as an optional synchronization feature within PCIe. It was only described in the protocol's third generation revision, i.e., Gen 3 base specification document [22]. The mechanism offers similar exchange of timestamps as PTP. This results into robustness against asymmetric network latency. In contrast to PTP, this protocol starts its exchange at the receiver's side, requesting a synchronization [22]. Although this feature is described in the protocol standard, it is not widely implemented in current PCIe devices and therefore not used in the field.

Synchronous Ethernet (SyncE) [21, 34, 35] is an augmented Ethernet version, which provides in-band synchronization. A local port clock is shared between two devices transmitting data to each other to avoid congestion. SyncE connects all these local clocks to each other to achieve synchronization over a network. To extract a clean clock signal out of the transmitted clock, a digital phase locked loop (DPLL) is used. Instead of using the clock from one of the devices, two dedicated timing cards can be added to deliver their clock on out-of-band buses. These cards introduce firstly redundancy with the second clock, but also externalize the reference clock which makes it possible to control its operation properties (i.e., frequency and duty cycle) from dedicated devices.

White rabbit (WR) [19] is a project developed at CERN, an Ethernet-based protocol. It combines SyncE and an adapted version of PTP, reaching sub-nanosecond accuracy. The idea is the WR devices first recover the reference clock with SyncE and synchronizes these local clocks by applying PTP. The White rabbit architecture can accommodate up to thousands of devices and distances of 10km and the system is commercially available by multiple vendors [20]. Although WR achieves a great performance in terms of accuracy, the SyncE part of this solution means it utilizes an Ethernet based interconnect protocol with an out-of-bound clock distribution to achieve this performance.

## 3.3. Precision Time Protocol, IEEE 1588

This section elaborates on the Precision Time Protocol (PTP, IEEE 1588 standard), covering its specifications, mechanism, and basic functionality but also analyzes its weaknesses. Based on the literature research presented in the previous section it seems that the PTP approach can be a potential basis for the mechanism proposed in this thesis. Although there are weaknesses that affect the performance, this protocol can be augmented to reach the requirements and achieve a strict time synchronization.

Because this section elaborates on an existing protocol, we make use of its already established terminology. This results in the use of the hierarchical terms “master and slave”. This is a general naming used in different protocols to describe the device responsibilities. Due to the controversial connotation attached to both words, the use of them is being phased out and alternatives are introduced. Examples of these new expressions are “leader and follower”, “primary and secondary”, “conductor and follower”, or “source and sink” [36]. However, for this section the original names are used to keep the consistency to the protocol as it is stated in the standard [2].

Table 3.1 presents the different versions of PTP and a short overview of the implemented clocks in addition to compatibility with the previous revisions. The first version was introduced in 2002 as a solution to deliver high accuracy in systems for which NTP and Global Positioning System (GPS) were not suitable. The NTP approach cannot achieve the required time resolution. On the other hand, GPS can exchange time very precisely, but is very expensive and bulky for small devices [17]. A new version was released in 2008, which introduced among other things a new type of clock, the transparent clock. Although the performance of IEEE 1588-2008 improved significantly, it was not backward compatible with the initial edition. The latest revision, $PTP_{v2.1}$ from 2019, has enhanced the previous standard with new features while being kept compatible with $PTP_{v2.0}$ [2].

*Table 3.1: PTP versions*

| Standard | Version | Features |
|---|---|---|
| IEEE 1588-2002 | $PTP_{v1.0}$ | • Clocks<br>◦ Boundary clock<br>◦ Ordinary clock |
| IEEE 1588-2008 | $PTP_{v2.0}$ | • Clocks<br>◦ Boundary clock<br>◦ Ordinary clock<br>◦ Transparent clock<br>• Not backward compatible with $PTP_{v1.0}$ |
| IEEE 1588-2019 | $PTP_{v2.1}$ | • Improved version of $PTP_{v2.0}$<br>• Backward compatible with $PTP_{v2.0}$ |

In PTP networks, the clocks are divided into different categories. The first one is the boundary clock, which is located at the edge of a subnetwork and functions as the reference time for this part. Secondly, there is the ordinary clock that includes all device timers of such a section. These are the ones which synchronize to the boundary clock. The third category is the transparent clock. These are not used to synchronize other clocks, but to measure the time a message is stalled in a network device so that this delay can be taken into account during the offset calculation [2].

The concept of PTP is to retrieve a reference time by correcting the slave’s clock with an offset which corresponds to the difference between the two counters. To achieve this, packets are exchanged with the timestamps of their departures and arrivals. The count difference can be approximated with a single message, but a second exchange is required to eliminate the transmission latency. As a result, the mechanism consists of a handshake in which triggers are sent between the master and the slave in addition to the master communicating the latched egress and ingress timestamps [2].

Figure 3.1 depicts the PTP handshake, with the four transactions occurring between the master and the slave. This represents an ideal situation because all transaction latencies are the same. Furthermore, the acquisition of each timestamp by the slave is shown clearly, which is important for the offset calculation.

The sequence in which these messages are exchanged is initiated by the master with the *sync*-trigger followed by the *follow_up* to transfer the timestamp $t_1$. In response, the slave replies with a *delay_req* when receiving the initial packet. The master responds to the trigger from the slave with the timestamp of its arrival in the *delay_resp* transaction.

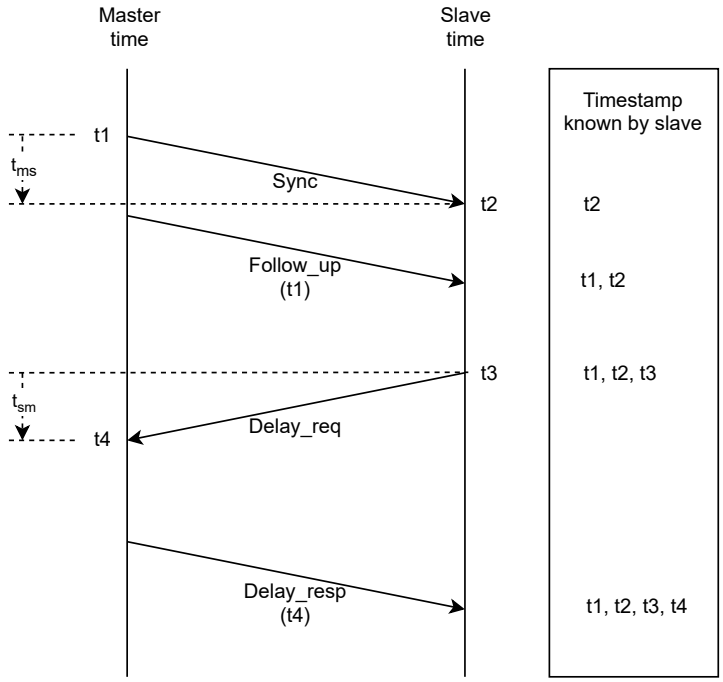


*Figure 3.1: PTP handshake [2]*

Table 3.2 presents the different messages used in the PTP protocol. These can be categorized into two types based on their functionality, namely event triggers and timestamp transfers. Additionally, the start and destination are given. Although the PTP standard also describes others, these are not fundamental and are out of this thesis scope [2].

*Table 3.2: Different PTP messages*

| Message | Type | From | To |
|---|---|---|---|
| Sync | Event trigger | Master | Slave |
| Follow_up | Timestamp transfer | Master | Slave |
| Delay_req | Event trigger | Slave | Master |
| Delay_resp | Timestamp transfer | Master | Slave |

The implementation of these two types of messages can affect the design. The reason for this is that the functionality differs. Trigger packets are used to register time on both transmission and arrival. Therefore, the timing of this type of packages is of interest rather than the data. This is in contrast with the data transfer messages, which are general messages where the data content is relevant, and the time constrains are less strict. Consequently, the requirements for these classes differ, which presents some design opportunities that are discussed in Section 4.2.

The PTP mechanism aims to retrieve the time error between two clocks. Therefore, the timestamps are compared which correspond to the departure and arrival of a trigger. Because this information is acquired at both sides, this needs to be transfered to the slave side. The offset calculation based on the information of one trigger, does not only contain the time difference between the two clocks but also an additional term. This term can be seen in Equation (3.1) and is the delay the message experiences going from the master to the slave. It is called the transmission latency and varies in time. The mathematical principle of this protocol is the use of a second trigger which goes from the slave side to that of the master. This results in a new expression of the offset, Equation (3.2), except the delay term has the inversed sign in respect to the same reference. When both equations are combined to express the offset, Equation (3.3) is obtained. PTP makes the assumption that the communication channel is symmetric and therefore the magnitude of both delays is equal. This symmetry states that the time from the master to the slave is the same as from the slave to the master. By adding both equations together, the result is two times the offset but both transmission latency terms are canceled out. The final expression of the offset is given in Equation (3.4).

$$t_1 - t_2 = \Delta t - l_{ms} \tag{3.1}$$

$$t_4 - t_3 = \Delta t + l_{sm} \tag{3.2}$$

$$\Delta t = \frac{t_1 - t_2 + l_{ms} + t_4 - t_3 - l_{sm}}{2} \tag{3.3}$$

$$\Delta t = \frac{(t_1 - t_2) + (t_4 - t_3)}{2} \tag{3.4}$$

where: $\Delta$ t The time offset between the reference count and the local count.
$l_{ms}$ transmission latency from master to slave.
$l_{sm}$ transmission latency from slave to master.

The theoretical assumption used to eliminate the unknown transmission latency from the offset calculation cannot always be guaranteed. There are different assumptions and sources of inaccuracy, which can introduce errors in the calculated offset. Firstly, the obtained formula is based on the assumption that the transmission is symmetric. However, this is not always the case with the high variance in communication traffic in RT systems. This leads to fluctuations in the latency experienced by trigger messages. Secondly, the correctness of calculation is dependent on the layer of the communication stack the timestamps are latched. Figure 3.2 present three different levels where this acquisition can be performed. These points correspond to the physical (i.e., "A"), operating system (i.e., "B") or the application layer (i.e., "C"). Although the offset should be independent of the transmission length if it is symmetric, each layer adds some variability to the latency. As a result, implementing the timestamp latch close to the actual network improves its precision [2].

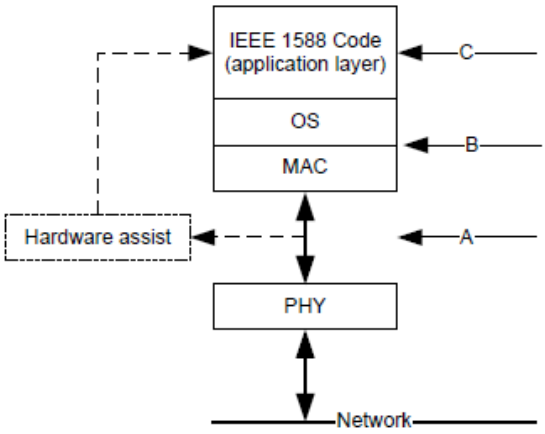


*Figure 3.2: Different layers to acquire timestamps [2]*

Figure 3.3 depicts a situation with different transaction latencies for the *delay_req* message to illustrate the effects of asymmetry in the calculation. Both counts have a difference of ten clock cycles. Only the trigger messages are shown to keep the figure clear. For each scenario, the offset is calculated with Equation (3.3), and the results are presented in Table 3.3.

As the offset calculation is the result of an averaging of two individual offset calculations, the error is also averaged, which results in the increase of the offset error being half of the latency difference [3]. However, if the asymmetry increases further, it can lead to large divergence in offset. This weaknesses introduces self-induced jitter if the count is compensated with this offset.

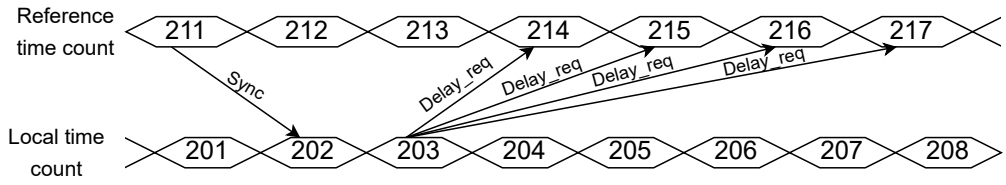


*Figure 3.3: Offset error due to asymmetric transmission latency*

*Table 3.3: Results of asymmetric offset calculations*

| $t_1$ | $t_2$ | $t_3$ | $t_4$ | Calculated offset | Actual offset | Offset difference |
|---|---|---|---|---|---|---|
| 211 | 202 | 203 | 214 | 10 | 10 | 0 |
| 211 | 202 | 203 | 215 | 10 | 10 | 0 |
| 211 | 202 | 203 | 216 | 11 | 10 | 1 |
| 211 | 202 | 203 | 217 | 11 | 10 | 1 |

Even a time synchronization with an ideal offset calculation is limited to ±1 period of the slave's local clock, due to the difference in frequency and phase between the two entities. This is illustrated in Figure 3.4. In other words, this local clock can be seen as the resolution in which the time can be adjusted. Hence, increasing this frequency leads to higher precision [3]. However, the local clock used for the remaining part of the system cannot be increased because this would introduce timing violations. As a result, separated clocks are required to increase the time resolution and assure the correct system behavior.

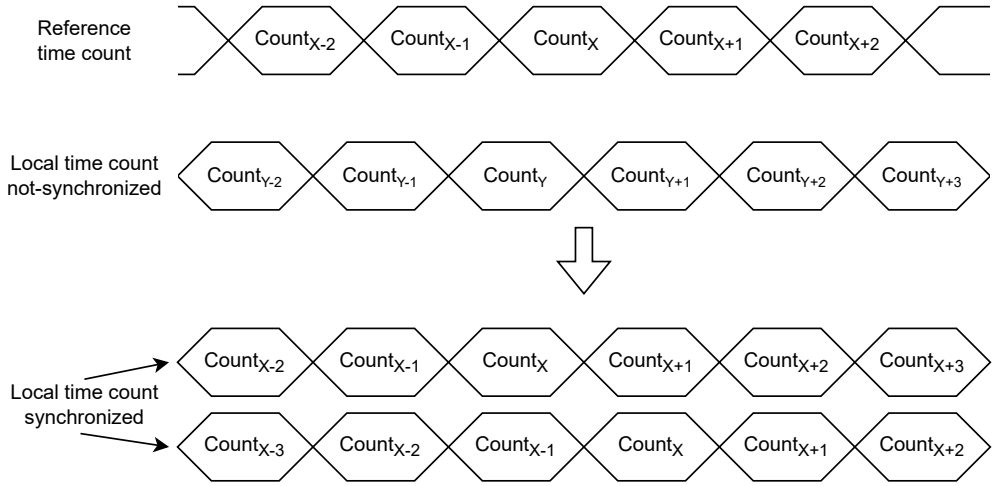


*Figure 3.4: Resolution limitations of synchronization [3]*

## 3.4. Conclusion

The different synchronization mechanisms can be divided into event-based or time-based protocols. Beside an overview of the different mechanisms, this chapter investigated how they are embedded in popular interconnects, such as RapidIO and PCIe. Research shows most of interconnect protocols use a type of PTP, as this offers robust synchronization. However, performance is highly dependent on the implementation. For example the SyncE part has a large contribution to the great results of White Rabbit. In conclusion, there are quite some implementation challenges that need to be addressed to achieve similar performance in a complete in-band mechanism. These will be discussed in the following chapter.

# 4

# Proposed synchronization mechanism

This chapter specifies and elaborates on the details of the proposed synchronization concept. This mechanism is an improved version of PTP, which provides solutions to address its weaknesses and provides better performance.

It starts with a specification section in which the topology, the general utilized mechanism and the specific functionalities are introduced. The following section gives a more detailed system view, where the interfaces between the synchronization design and that of the rest of the device are discussed. Next, the time representation in the synchronization components is elaborated on, followed by a section regarding the dedicated interfaces for trigger messages. To finalize this chapter, the filter block is discussed, which is utilized to increase robustness against incorrect offsets.

## 4.1. Specification

The goal of this section is to present the reader a high level understanding of the proposed synchronization mechanism. The design of this mechanism is based on the PTP principle, as it enables to retrieve the time difference between two time-holding entities while handling the transmission latency. When one entity adjusts its time with this offset, it synchronizes to the other entity's time. However, incorrect offsets lead to unnecessary adjustments and even time oscillations, in other words: it results in self-induced jitter. As seen in the previous chapter, PTP has some weaknesses which can lead to incorrect offset calculations, namely asymmetric transmission latency and lack in precision of the timestamps. Therefore these weaknesses are addressed in the proposed synchronization mechanism.

The network topology of the proposed mechanism, similar to PTP, consists of a centralized network layout which contains a primary device that holds the reference clock. The rest of the devices in the network communicate directly or indirectly with this reference node to retrieve its time. In contrast to a distributed design, which negotiates the correct time based on multiple devices, this category of networks makes it possible for the user to control and adjust the source clock.

Figure 4.1 presents an example network with one device that is set to have the reference time; all the other network elements synchronize to that source. For ease of reasoning, a tree topology is chosen. With the synchronization protocol time information is spread through the network beginning from the primary reference holder. Note that each device in the network can be set as the reference time, which offers flexibility to the designer.

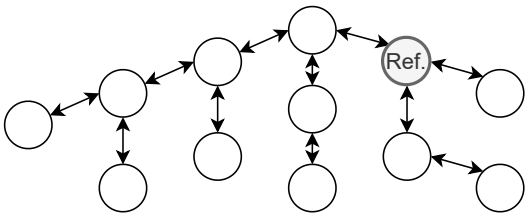


*Figure 4.1: Topology example*

The proposed system offers the possibility to add this synchronization mechanism to all sorts of different designs implemented in devices. It is separated from the interconnect protocol that connects all the devices in the network. Additionally, this make it possible to have different interconnect protocols connecting the different devices. However, the synchronization system assumes a high speed interface to connect all the devices as this is common for RT systems. The mechanism uses a hierarchical structure consisting of one component which is regarded as the reference and one which needs to be synchronized to that reference. These two components are called the SyncMaster and the SyncReceiver. Figure 4.2 gives an overview of both components with their time counter.


*Figure 4.2: The two synchronization components*

Both components interact with each other, similar to PTP, where the SyncMaster initiates the synchronization interaction with all the SyncReceivers. At arrival of the first trigger (i.e, *Sync*-message), a trigger is sent back by each SyncReceiver (i.e, *Delay_req*-message). Of the four timestamps required, two are acquired at the SyncReceiver side by receiving the first trigger and responding with its own. The other two are received from the SyncMaster. After the exchange of timestamps, the SyncReceiver computes the offset. Because the time of the SyncMaster is seen as the reference, its design does not consist of functional blocks to calculate or adjust the offset. Yet, the SyncReceiver needs to be able to calculate the offset and correct its local time. In the proposed design, a filter is added to minimize the effects of incorrect offsets; this is discussed later in more detail.

The proposed synchronization mechanism uses, similar to PTP, an exchange of timestamps to determine the offset. In addition, it implements design improvements to deal with the weaknesses and achieve more robust and precise performance. The two main parts that improve the basic design are a high priority communication channel aimed at transmitting triggers and a filtering block that reduces the effects of incorrect offsets. Both concepts are explained below, and afterwards discussed in detail in Section 4.2

As discussed in Section 3.3, the precision with which timestamps are acquired is highly important to bring the synchronization errors down. In the proposed design, all the synchronization transactions are taking place on the *Data Link Layer* (DLL) of the OSI model [37, 38]. To minimize the offset errors as a consequence of going through different levels, it is preferred to latch the timestamps as close to the network as possible [2]. However, the proposed mechanism differentiates the different categories of messages. The trigger event type is responsible for latching the timestamps and is characterized by its functionality as a trigger which does not necessarily carry any other information such as data. Therefore, these messages can be implemented as small high priority packages that can be inserted easily between the rest of the traffic without delaying it much. This is because a dedicated interface is designed which is able to send a trigger message directly to the transceiver of the interconnect. At both sides of the communication channel the stream of traffic is checked for such messages. As a consequence, the exact arrival and departure time of these packets is deduced.

As the trigger messages experience less jitter in their transmission because of the direct interface, fluctuations in the transmission latency are reduced, therefore the latency is more symmetric. Nevertheless, asymmetric behavior can occur which results in incorrect offset calculations. To avoid adjustments of the time with these correction values, a filter is added. This block ensures that abrupt offset changes and outliers are filtered out. As a result, the time is kept more stable and the jitter is decreased.

## 4.2. Device design

This section elaborates on the detailed design and functionality of the proposed synchronization solution. First, it covers the functioning at component level and gives an overview of the designs of both components. A component is defined as an independent subsystem responsible for a functionality of the complete system, e.g., SyncMaster or SyncReceiver. Afterwards, the time representation is discussed next to zooming in on the main functionalities that improves the performance of the proposed synchronization mechanism.

The SyncMaster and the SyncReceiver have the same interface. Although both components have some synchronization specific functional blocks, most of the IP blocks are also the same. This interface consists of multiple clock signals and two sorts of communication channels. These are discussed more extensively in Section 4.4. Most of the parts are made to suit the design of both components. This makes it possible to reuse them in both components and simplify the process of testing.

An overview of the interface grouping is shown in Figure 4.3. These connections are divided into system input-, timing specific-, data message-, and trigger message signals. The latter two are communication buses which contain multiple links, the first one is responsible for the data transactions, for example the timestamps but also the configuration of the components. The latter one for the communication of the dedicated interface for the triggers. Similar as for the data, the channel is bi-directional, in other words it is used to send and receive these messages.

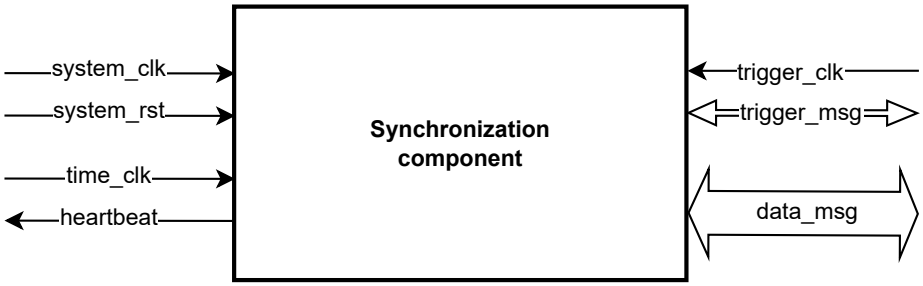


*Figure 4.3: General interface for the synchronization components*

Figure 4.3 also presents three different clock signals used in the components, namely *system_clk, time_clk,* and *trigger_clk*. Figure 4.4 shows where the clock domains are located and routed to in the device design. System_clk is the clock driving all Intellectual Properties (IPs), i.e., functional blocks, in the device and therefore also most of the functional blocks in the synchronization components. Time_clk drives the functional block, the counter, which holds the time. This way, its frequency is higher than the others as will be discussed later. Trigger_clk is used for the dedicated trigger message, which is also used in the receiver end of the interconnect transceiver. The transceiver consists of a transmitter and a receiver which are connected to their counter-parts in the transceiver of the adjacent device. Each transmitter is driven by the local clock of the device it is in. As a result, the receiver part needs to retrieve the clock of the transmitter it is connected to and run on this clock. Therefore all incoming traffic has to pass through a *clock domain crossing* (CDC).

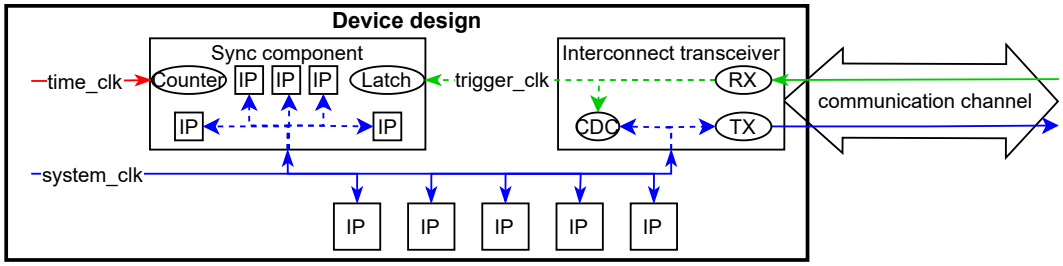


*Figure 4.4: The clock domains in a device design*

A CDC ensures the correct transfer of data between two clock domains. The situation of the transmitter (TX) and receiver (RX), where the the clock of RX experiences a slight difference to that of the TX, is shown in Figure 4.5. It can be seen that the transmitted signal changes close to the rising edge of the RX clock, which results in the violation of timing requirements of the internal registers. This is called meta-stability and leads to the signal taking randomly its old or new value [39]. These violations can lead to significant errors in the data transfer as this affects all changing bits. To ensure the correct transfer, a *synchronizer* can be used and is presented in Figure 4.5. This implementation adds an extra register which is also driven by the RX clock and guarantees a well defined state [5]. However, this solution comes with an additional clock cycle delay.

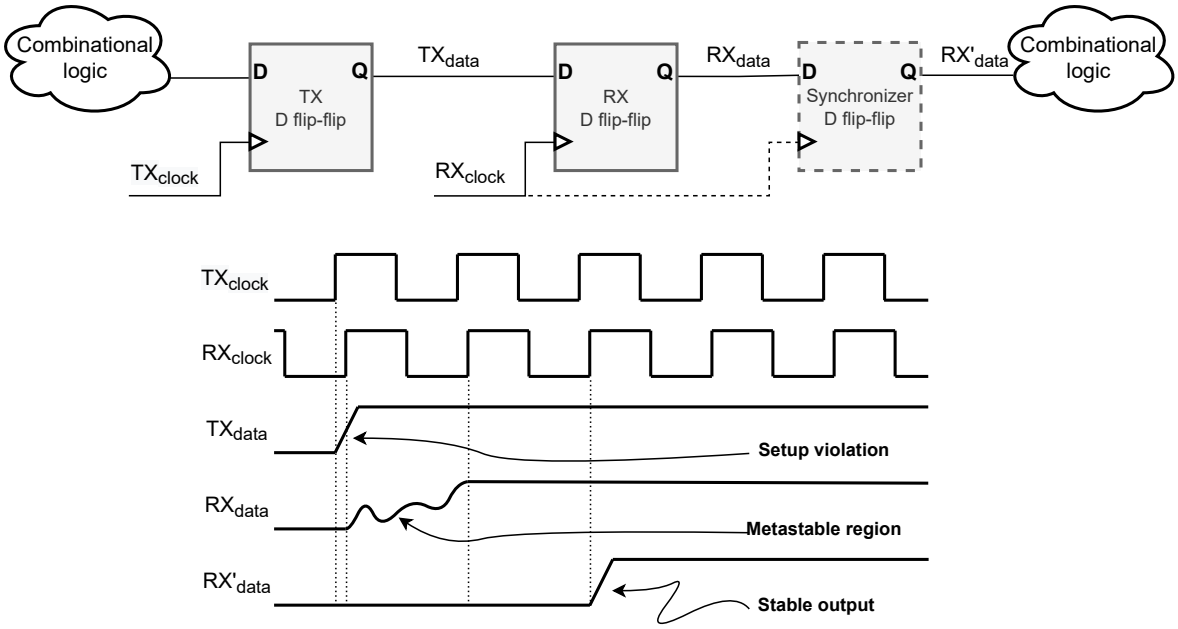


*Figure 4.5: CDC meta-stability [4, 5]*

## 4.3. Time representation

Each device keeps a local time notion on which a periodic trigger is produced. This trigger is called a heartbeat and indicates a predefined moment in time. As the synchronization mechanism seeks to ensure that the notion of time is equalized in the network, all heartbeats should be simultaneous. This event is similar to a church bell ringing each hour and is expected to be independent of location. These heartbeat events indicate the moment of execution for internal tasks. Additionally, these events can be compared between devices in the network in terms of synchronization. The frequency in which the heartbeats occur, $f_h$, is specific for the overall system of devices. The minimal period is determined based on the longest duration of task [39]. The relation between frequency and period is presented in Equation (4.1).

$$T = 1/f \tag{4.1}$$

where:

$T$ = Period
$f$ = Frequency

The period between two consecutive pulses can be divided in equal parts by a faster clock with frequency $f_c$. This is similar to regular time where each hour consists of 60 minutes. The relation between these two clocks is described as the number of clock cycles in the heartbeat period and presented in Equation (4.2). As a result, the phase of both clocks are synchronized while the frequencies are an integer multiple of each other. This can be seen as the resolution of the heartbeat. In other words, it is the precision in which the time can be adjusted to synchronize it.

$$N = T_h/T_c \tag{4.2}$$

where:

$N$ = Resolution value
$T_h$ = Period of the heartbeat
$T_c$ = Period of the counter

The *time_clk* input is used for this faster clock and can be designed independently of the rest of the design. As explained in the previous paragraph, its frequency corresponds to the precision of the synchronization. Because period $T_h$ is dictated by the system, $T_c$ needs to be used to achieve the required resolution. An increase in this frequency is advantageous as this results in a higher number of counts per heartbeat. Figure 4.6 depicts the heartbeat and sample signals with different frequencies. Table 4.1 presents the corresponding resolutions based on Equation (4.2). Although the resolution of time increases, the precision of the timestamps do not increase if the synchronization triggers are delivered via a CDC with a slow clock. To utilize the advantage of the faster time clock, this proposed synchronization protocol has an asynchronous latch mechanism implemented which will be discussed in Section 4.4.

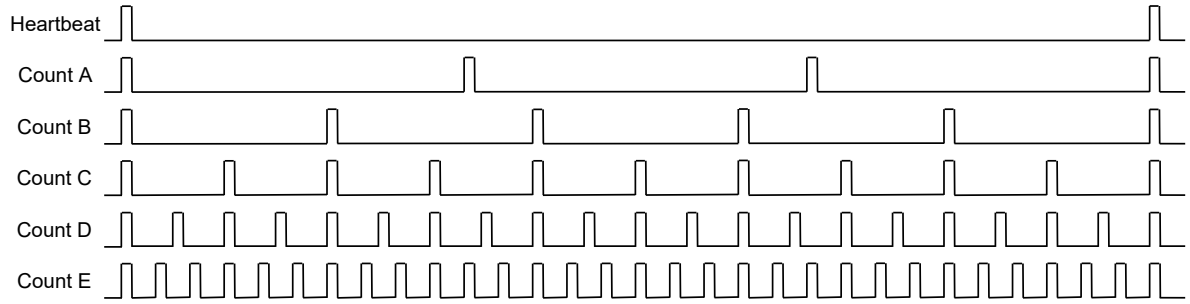


*Figure 4.6: Heartbeat signal with different different sample periods $T_c$*

*Table 4.1: Heartbeat resolutions corresponding to Figure 4.6*

| | Count A | Count B | Count C | Count D | Count E |
|---|---|---|---|---|---|
| $T_h / T_c$ | 3 | 5 | 10 | 20 | 30 |

This faster clock is sampled in a functional block called *the counter*. This entity stores a local count and increases it each rising edge of its local clock. Each device has its own oscillator which is responsible for the input signal of the counter. Because of variations in manufacturing and the operation environment, there are small differences in clock frequency. As a result, the triggers of the clocks diverge from each other. This is called drift.

To synchronize between devices in a network, the difference in counter value is determined and is compensated by an offset-correction. This means the counter IP also needs to be able to increase its current count with a dedicated offset value next to the periodic increment of one. These offsets are caused by the drift between the clocks, however, there is also a large offset before the first synchronization due to different starting moments.

As mentioned previously, a heartbeat is produced periodically, based on its local notion of time and attempts to be simultaneous with the rest of the network. There are different ways to implement the heartbeat. There is the option to use a second counter which counts the clock cycles since the last heartbeat. It would increase and synchronize its value the same way as the global counter. However, this block is reset to zero when it reaches the resolution value N and therefore a heartbeat is produced. This design choice adds an extra counter and complexity to synchronize to counters.

In this design, the heartbeat is implemented by scheduling it. Instead of a second counter, the count value for the next heartbeat is determined in terms of the general time counter. This requires that each time a heartbeat occurs, the following moment is calculated. Each clock cycle, the current count is compared to the time of the next heartbeat. This approach also offers the advantage in distributed systems to plan tasks simultaneously. Additionally, this enables the heartbeat to be configured such that it is generated with a user defined offset.

## 4.4. Dedicated trigger interface

This section goes into the design details and reasoning of the choices made in this proposed synchronization mechanism, regarding communication channels. Especially the dedicated interface for trigger messages is elaborated on. This functionality enables to directly send and receive these type of messages which improves the timestamp acquisition.

Both synchronization components, the SyncMaster and SyncReceiver, have the same communication channels. These channels are divided into two types based on the two synchronization transactions categories, namely data and trigger messages. This division makes it possible to design the channels based on the specific requirements instead of using a general communication channel for both types. The main communication channel is a memory mapped one which uses register writes and reads to transmit the data. This interface is used for all configuration and data transactions. This type of transfer requires mainly reliability, while transmission latency is less important. In our case, the AXI4-Lite protocol [40] is used.

For the other message type, a different approach is proposed. Because the trigger messages do not need to contain any data and the value of these messages lies in their departure and arrival, a dedicated interface is implemented. This interface is directly connected to the interconnect transceiver, as can be seen in Figure 4.7. Via this connection, the synchronization component, as both the interfaces of the SyncMaster and the SyncReceiver are similar, can request a trigger to be send. As presented earlier, these trigger messages are implemented as the smallest size packets with higher priority than the general data. This ensures that these messages are transfered with the least delays and jitter as this implementation is less susceptible for back-pressure. Additionally to the direct-request signal, this dedicated interface is able to receive triggers which indicate the departure or arrival of such a message. These triggers come directly from the transceiver and are used to latch the time asynchronously.

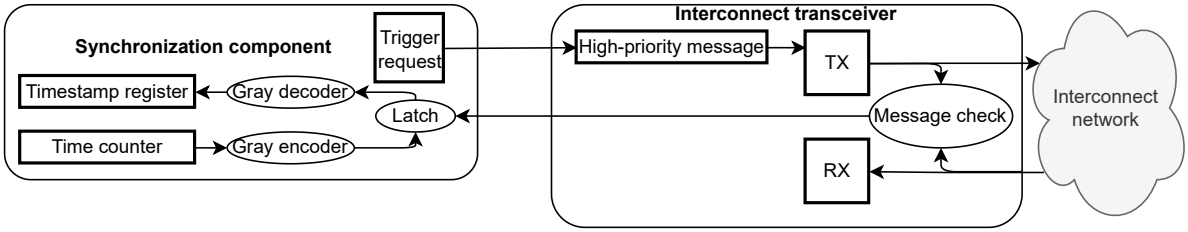


*Figure 4.7: Overview of the direct interface between the synchronization components and the interconnect transceiver*

The asynchronous acquisition of the timestamps can lead to meta-stability and therefore large errors in the timestamps. This is because the time is kept in a 64-bit counter which increases each clock cycle. During this incrementation of the count, multiple bits can change in value. Due to the meta-stability of this part of the system, each changing bit can go randomly to its old or new value. As a result, the latched value could be completely off. To solve this problem, the 64-bit time signal is first converted to *gray-code*, which is a representation where each incrementation only changes one bit, and therefore the meta-stability can lead to a maximum error of one [41].

## 4.5. Offset filters

The following part discusses the offset computation subsystem, which is used to achieve a robust and stable time correction. This subsystem is only implemented in the SyncReceiver as the time needs to be corrected in this component. This part of the SyncReceiver design can be divided into two stages, namely calculation and filtering. Figure 4.8 presents the two stages, together with the naming of the corresponding variables. In the proposed design, the filtering stage is added to reduce outliers and large jitter in the offset calculation caused by asymmetry in the trigger transmission. Two types of filters are discussed and compared to find the desired behavior of filtering the outliers.

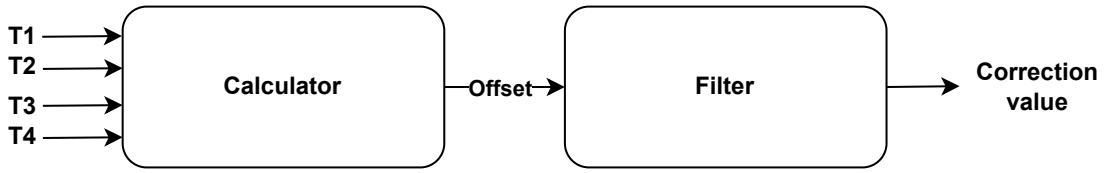


*Figure 4.8: Overview of the offset computation*

Offset is the time difference between the local clock of the SyncReceiver and the global reference of the SyncMaster. Figure 4.9 illustrates this time offset and shows the local clock can both lag (i.e., local time A) or lead (i.e., local time B) the reference which is indicated by the sign. The required data to determine the offset is acquired during the synchronization handshake by receiving and latching the triggers on its own side. As a result, all four timestamps are available at the SyncReceiver side and the offset calculations can be calculated.

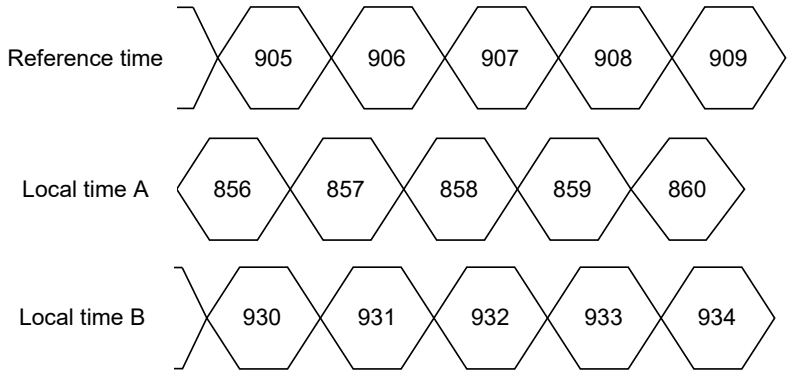


*Figure 4.9: Time offsets*

The first computational stage is the offset calculation. This is the same as that of PTP, presented in Equation (3.4). This formula consists of three additions and one division by two which are arithmetic steps that are easily implemented.

Although not all timestamps arrive at the same moment, this does not pose a problem as the calculations can start each time one of them is available. However, the result is only valid after all timestamps are available. The variables which have arrived previously stay unchanged until the next synchronization moment which is much later than the arrival of $t_4$.

The last calculation step is the division by two, which can be implemented by an *arithmetic right shift*. Because the offset can both be positive and negative, this operation handles the sign well. However, this operation brings some imprecision as the division result is rounded down. In other words the offset is underestimated in case it is positive and overestimated when it is negative.

Another solution to deal with the division is by not implementing it and treat the result as one with a fractional bit. This implies that the least-significant bit (LSB) indicates the presence of half in the result. So instead of losing information, it is kept in the offset and therefore increases the precision. Nevertheless, the result needs to be extended with a new most-significant bit (MSB) to keep the same range. This bit has the same value as the previous MSB.

The calculator IP delivers an offset based on the latest acquired timestamps. The correctness of the calculation is based on the assumption the synchronization handshake has experienced a symmetric transmission latency. If this assumption is not met, this introduces inaccuracy in the offset. This results in major corrections when the offset is used directly by the counter.

This lack in robustness introduces jitter and the desynchronization of the heartbeats. This can be limited by the use of a filter which decreases large differences in consecutive offsets. As a result the correction values are smoothened such that the adjustments are less aggressive. However, as a consequence large offsets which are actually present in the system need more time to be corrected. This needs to be taken into account during the design of such a filter.

Before the different possible filter types are discussed, the course of synchronization is elaborated to understand the design choices. When a device is added and enabled in the network, its counter starts from zero. As a result this time lags that of the rest of the network. In this case, a big adjustment is needed to catch-up with the reference clock. Once this is done, the required corrections are smaller as they need to compensate the drift between the two clocks. So, at the start it is expected for the offset to be large but after some corrections the assumption is that there only should be small rectifications of the time.

The first filter divides each offset by a constant and therefore decreases the magnitude of the correction value. To implement it efficiently, this constant value should be an exponent of two, as this corresponds to a bit-shift. This approach ensures a less aggressive time adjustment but only takes the current offset into account. This results in a constant damping, where actual larger offsets take more time to be corrected.

In contrast to only using the current offset, the filter can also take previous values into account. The sliding-window averaging filter is such a filter. It calculates the correction value as the average of previous offsets by adding all the values and dividing this by the window size. The window size is a parameter that dictates the number of samples this IP takes its average on. As it is a sliding window implementation, it takes the latest offset and discards the oldest one. Similar to previous division implementations, the window size is best to be an exponent of two. Because this filter uses the previous offsets, these old values also need to be corrected with the filtered value. This is needed as its local time-frame is adjusted with this correction. To avoid addition of values of large bit sizes, an assumption is made to only calculate with values smaller than 64. When an offset is larger than this threshold, it is directly used as correction value. The choice of this threshold value is based on limiting the size of offset representations to 6 bits. Additionally, this choice assumes that calculated offsets higher than 64 are the actual time difference and not wrongfully calculated due to network interference such as jitter or asymmetric transaction latencies.

For this synchronization mechanism, both filters are implemented to analyze their performance and compare these to the required behavior presented earlier. To make it possible to find the right constant for the first one and the window size for the second, both of these are parametrized. In other words, these are set during the initialization of the SyncReceiver.

## 4.6. Conclusion

A new synchronization mechanism is proposed which is developed based on PTP. Similarly, it uses a mathematical principle to cancel out the transmission latency of the trigger messages. However, this principle is based on the assumptions that these latencies are symmetric. To minimize the effects of incorrect offsets due to asymmetric and non-deterministic behavior, a filter design is presented that reduces large jumps and outliers. Beside the mitigation of incorrect offsets, this mechanism includes an implementation of a dedicated interface for trigger messages. This enables these messages to be directly send and received by the synchronization IP, and therefore increase the precision of the timestamp acquisition. Additionally, an extra high frequency clock is used to sample the time, which increases the resolution in which the timestamps are latched. In next chapter, the simulations of the different filter types and a full PoC with this proposed design are presented.

# 5

# Experimental results

The synchronization design and its functionalities discussed in previous chapter are simulated and validated in this chapter. First part focuses on the behavior of the filter IP by simulating the different types and window sizes. After that, experimental results are acquired from the complete system and shows the performance of proposed synchronization principle. Finally, the design is validated in a real hardware setup.

## 5.1. Filter simulation

Because the filter block has such an important role in the correct offset adjustment, this functional block is simulated to check its behavior. As discussed in the previous chapter, there are two types of filter implemented, namely division-by-constant and sliding-window average. Therefore their performance can be examined and compared. Beside the choice between these two types, a size parameter can be tuned. This parameter acts as the constant and window size for the division-by-constant and sliding-window average filters, respectively.

The test system for the filter IP is presented in Figure 5.1. It is a simple setup where the filter block can be configured with two signals dedicated to the type and window size. The test consists of a sequence of offsets which is delivered to the filter. Its response is then monitored and visualized by a monitor entity.

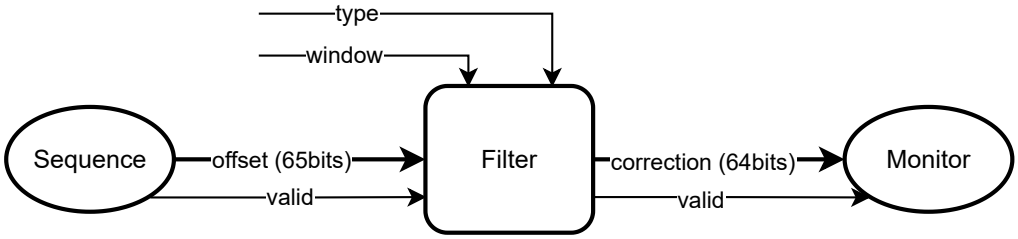


*Figure 5.1: Setup of filter simulation*

The goal of this simulation is not only to validate the correct functionality but also to see what filter configuration (i.e., type and window size) presents the desired performance. This behavior should be a balance between aggressive response and robustness against outliers. In other words, the filtered correction should follow the magnitude of the correct input but at the same time not react on a single incorrect deviation.

The tests are run with a specific sequence of offsets to mimic the synchronization environment. In other words, the inputs are bound to certain constraints when they are randomized in a way that the sequence resembles real offsets. These conditions are described as a distribution of different intervals with corresponding probability. The sequence is a stream of offsets with mostly small magnitudes, although there are some occasional outliers. The first sequence covers all types of offsets, where the largest are capped arbitrarily at an absolute value of 127. Secondly, another sequence is used which is similar to the first one except the largest offsets are kept beneath the direct-jump threshold of the averaging filter (i.e., 64). As a result, the simulation zooms in on the behavior of the filters in these circumstances. Both distributions of the input sequences are presented in Figure 5.2.

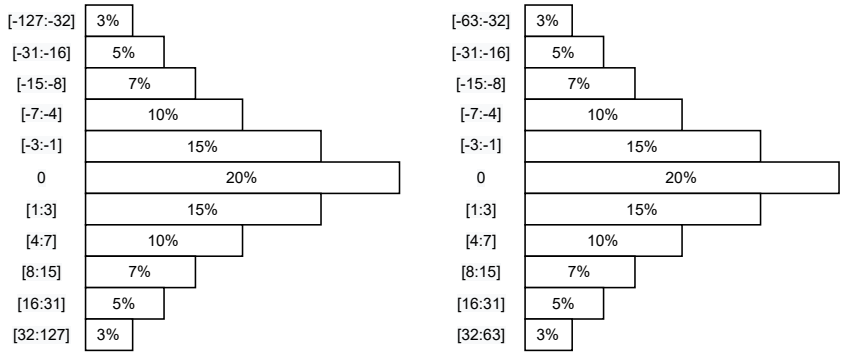


*Figure 5.2: Distribution of both input sequences*

Figure 5.3 plots the two offset sequences which only differ in the large values. These exact inputs are kept the same for each test which makes it possible to compare the different responses. It needs to be noticed that there is no correlation between an offset and its predecessor, nor in magnitude, nor in sign. This is different from the real-life offsets as these offsets are dependent on the previous corrections. Therefore these simulations have a more extreme behavior than the calculated offsets during synchronization.

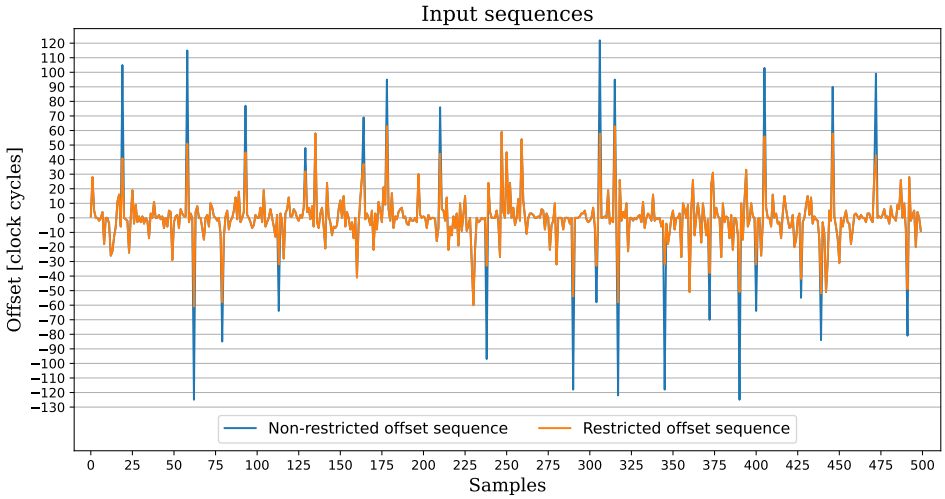


*Figure 5.3: Plots of both input sequences*

Firstly, the behavior of the division-by-constant is evaluated based on Figure 5.4 and Figure 5.5. The results are as expected a damping of the offset value towards zero; the increase in value of the constant corresponds with a halving of the output. Next, this simulation is compared with that of the sliding window averaging filter, depicted in Figure 5.6 and Figure 5.7, a big difference can be observed regarding the large offsets. These values surpass the threshold and are therefore not filtered but pass-though directly. Additionally, all previous offsets which are kept to calculate the average are reset to zero. This results in the filter not being able to proof its value as large offsets reset its calculations. To zoom in on its general behavior without these reset values, the second sequence is used.

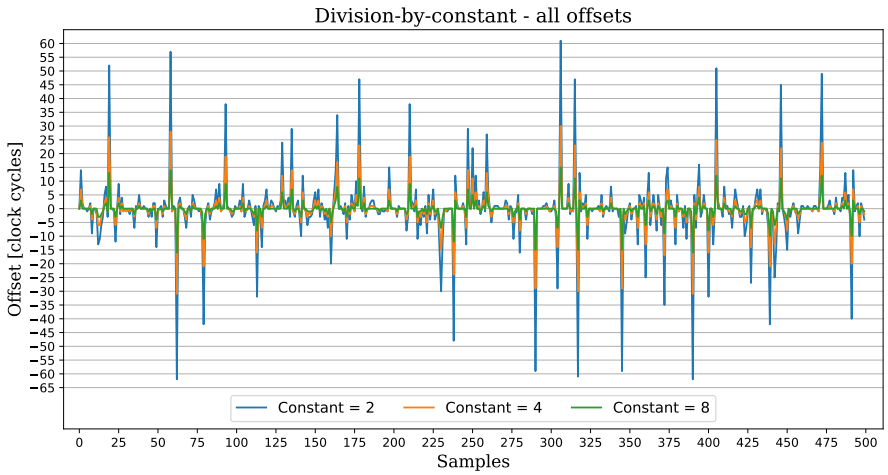


*Figure 5.4: Results of the division-by-constant filter with constants 2, 4, 8 - all offsets*

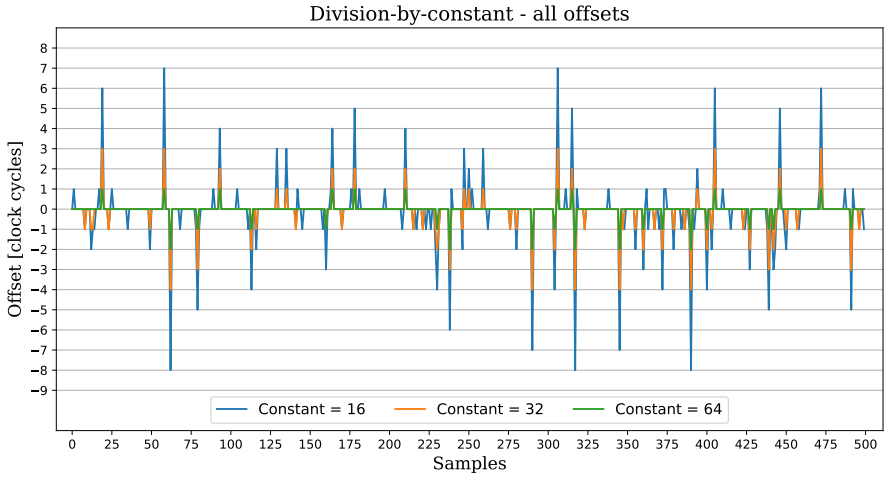


*Figure 5.5: Results of the division-by-constant filter with constants 16, 32, 64 - all offsets*


*Figure 5.6: Results of the sliding window averaging filter with window size 2, 4, 8 - all offsets*

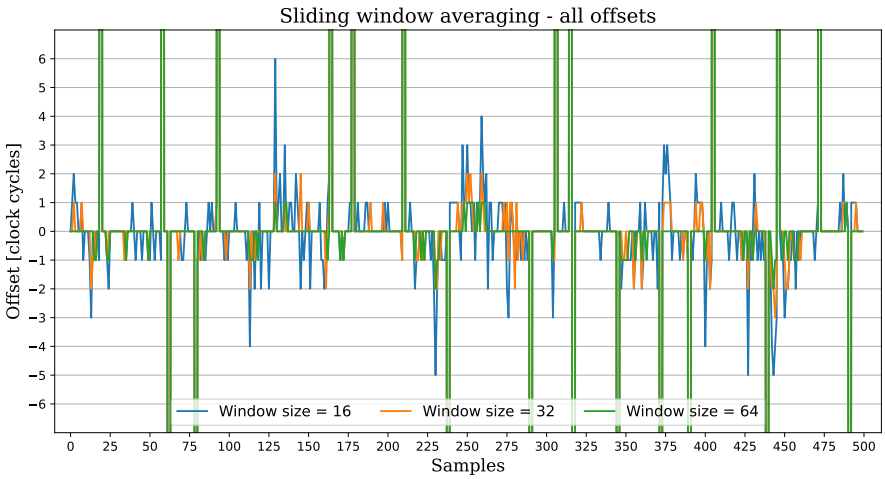


*Figure 5.7: Results of the sliding window averaging filter with window size 16, 32, 64 - all offsets*

For the second simulation where the large values are restricted beneath the threshold, not all results are discussed. Only the upper three window sizes of both filters are evaluated as the behavior of these are most of interest based on the desired damping. The results of both types are presented in Figure 5.8, Figure 5.9, Figure 5.10 and Figure 5.11. The selected outcome from the division-by-constant filter shows that all inputs with a smaller magnitude than the constant become zero, whereas all other offsets are reduced to small fractions of the input. This is in contrast with the filtered corrections of the sliding window averaging filter where the output is based on a trend of previous inputs. Both filters have presented some desired behavior. The division-by-constant damps the large values whereas the sliding window averaging is able to detect a trend. This trend in offsets can be assumed to be the drift and therefore the correct offset.

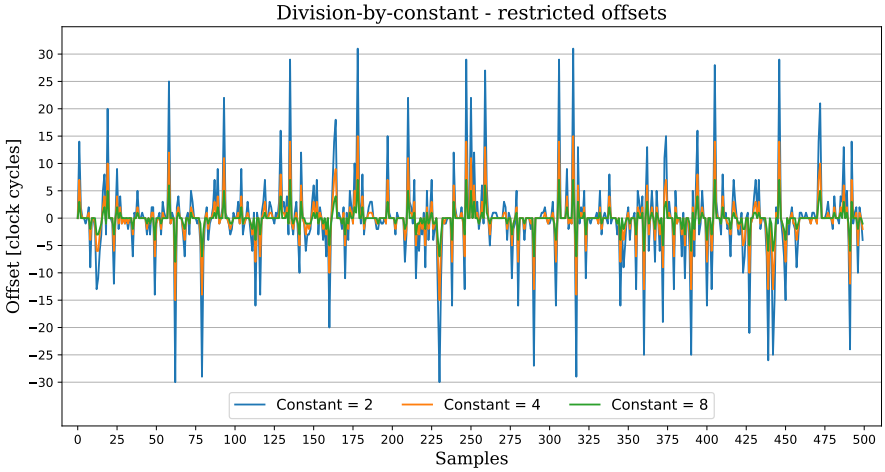


*Figure 5.8: Results of the division-by-constant filter with constants 2, 4, 8 - restricted offsets*

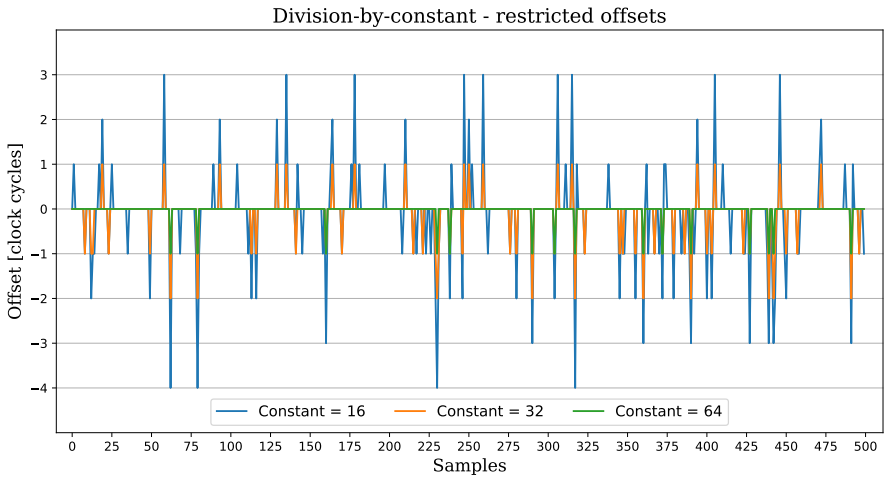


*Figure 5.9: Results of the division-by-constant filter with constants 16, 32, 64 - restricted offsets*


*Figure 5.10: Results of the sliding window averaging filter with window size 2, 4, 8 - restricted offsets*

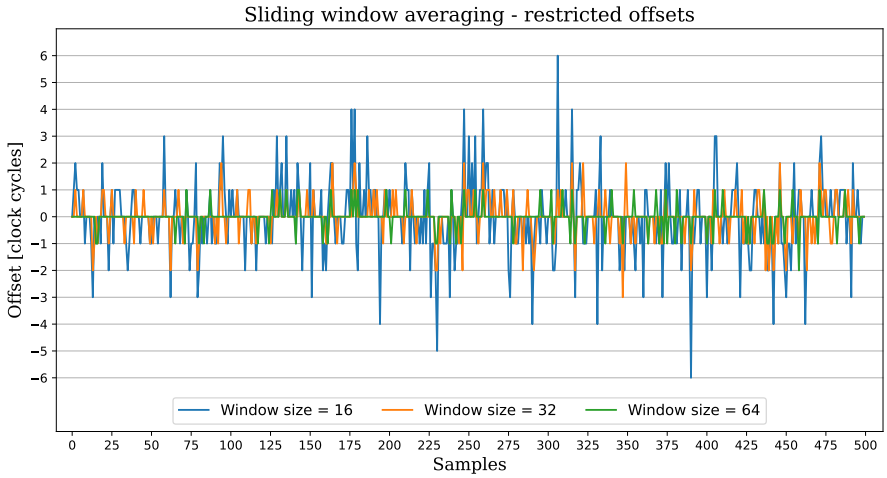


*Figure 5.11: Results of the sliding window averaging filter with window size 16, 32, 64 - restricted offsets*

## 5.2. Full system simulation

After focusing on the behavior of the filter IP and its different configurations, this section describes the complete system consisting of three devices. This test setup has one device, the host, which acts as the reference time keeper and two others that function as endpoints. An overview of this network is depicted in Figure 5.12 where it is also shown how the synchronization components are connected to each other. The three heartbeat signals of these components are wired to a monitor to visualize these events in reference to each other.

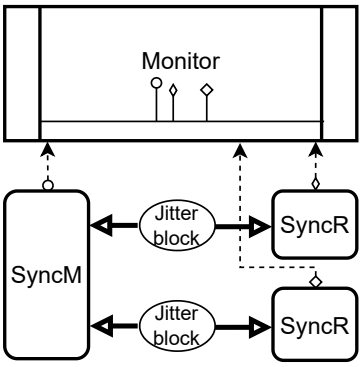


*Figure 5.12: Setup of full system simulation*

Regarding communication, the components are connected to some simple network IP's via an AXI4-lite interface. These ensure that the messages are directed to the right addresses at the other side. The inter-device communication would be a multi-bit serial interface which includes the different message priorities needed for the dedicated interface of the synchronization components. However, for this simulation the high speed link is abstracted and therefore the communication channel behaves as a perfect transfer medium without any delay or jitter. To introduce this behavior into the simulation, a jitter block is inserted in the channel.

To introduce jitter in the communication channel, the jitter block delays each message by holding the transactions for a random number of clock cycles before forwarding it to the other side. The random latency is based on statistical distributions. Different distribution parameters are used for the two types of messages, namely trigger and data. Because of the high priority of trigger packets, the delay distribution for these packets have a mean of zero but to test the robustness the synchronization, the standard deviation is set on five. For the normal data transmission, the deviation is defined by a mean of two and a standard deviation of ten. All the parameters are given in an overview in Table 5.1.

*Table 5.1: Parameter overview of the delay distributions*

| Parameters | Mean | Standard deviation |
|---|---|---|
| Trigger messages | 0 | 5 |
| Data messages | 2 | 10 |

This transfer jitter results in asymmetric transmission delays and variance in the timestamp exchange to the SyncReceiver. The mechanism should be robust against these imbalances. Another imperfection is the drift of the clocks due to small unwanted differences in clock frequency. The *system_clk*, which is used by most of the device design, is set to 200MHz and the specific *time_clk* at 400MHz. In this simulation, a drift is set of 10$\mu$s/s. This is representative for what would be present in real hardware setup.

For the sequence, the test starts with the configuration of all components which includes setting the heartbeat period and all addresses in addition to the filter configuration. In these tests, 1MHz is chosen as the heartbeat frequency. This is much higher than what many systems use. This has as consequences that the period in which the complete synchronization needs to occur is shorter although the drift also is more limited. Alternatively, if the periods are longer, the synchronization moment needs to be set closer to the heartbeat. This way the drift does not affect the corrected time and the heartbeat moment is kept simultaneous with that of the reference. After this setting, the host is enabled and its counter starts keeping the time. After 20$\mu$s, both endpoint components are enabled which introduces start offset. Now the synchronization mechanism begins and the behavior can be observed.

The performance can be monitored directly via the three heartbeats and the difference between them. This difference changes each heartbeat as the synchronization mechanism corrects this skew. In addition, the time of the heartbeat is recorded. Based on this information, the jitter is computed and plotted. Similar to the previous simulation, these tests are run with different filter configurations.

This simulation is run for 500$\mu$s which results in 493 heartbeat samples per device. For each configurations, the different skew and absolute jitter are calculated. First of all, the synchronization without filter, presented in Figure 5.13, is used as a baseline for the rest of the measurements.

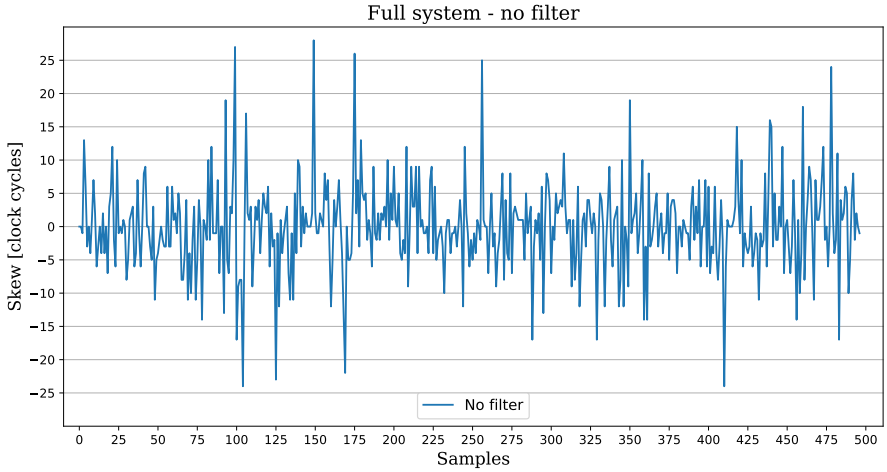


*Figure 5.13: Full system simulation results with no filter*

The results of the division-by-constant filter are quantified in Table 5.2 and show an immense increase in skew, much worse than the baseline measurements. This can be explained based on two properties of this filter. The first is the lack of a direct-jump for large offsets and secondly its general damping. Both properties can be observed in the simulation of the division-by-constant filter, which is depicted in Figure 5.14. At the start, the time difference between the local count and the reference is very large due to the different moments both components were enabled. This difference is retrieved by the synchronization mechanism, except due to the filter the count is only corrected with the calculated offset divided by the constant. As a result, the local count converges slowly towards the reference count. To better observe the behavior at this point, Figure 5.15 gives a zoomed-in view of the results. It can be seen that once the local counter has caught up, the performance is better than the baseline as the filter damps the jitter. For configurations with a higher constant, this behavior is more present. In other words, it takes more time to correct the count from the initial difference, next to the property that all offsets smaller than the constant are set to zero.

*Table 5.2: Results of skew and absolute jitter of division-by-constant filter*

| Filter type | No filter | Division-by-constant | | | | | |
|---|---|---|---|---|---|---|---|
| Window size/constant | | 2 | 4 | 8 | 16 | 32 | 64 |
| Median skew | 0 | 0 | -1 | -2 | -5 | -19 | -56 |
| Max skew | 28 | 16 | 7 | 3 | -2 | -13 | -47 |
| Min skew | -24 | -184 | -549 | -584 | -805 | -889 | -932 |
| Absolute jitter | 52 | 200 | 556 | 587 | 803 | 876 | 885 |

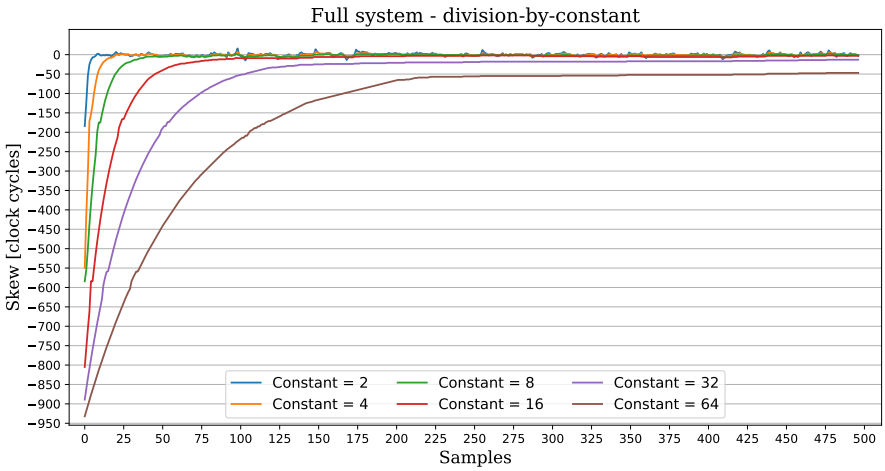


*Figure 5.14: Full system simulation results of division-by-constant filter*

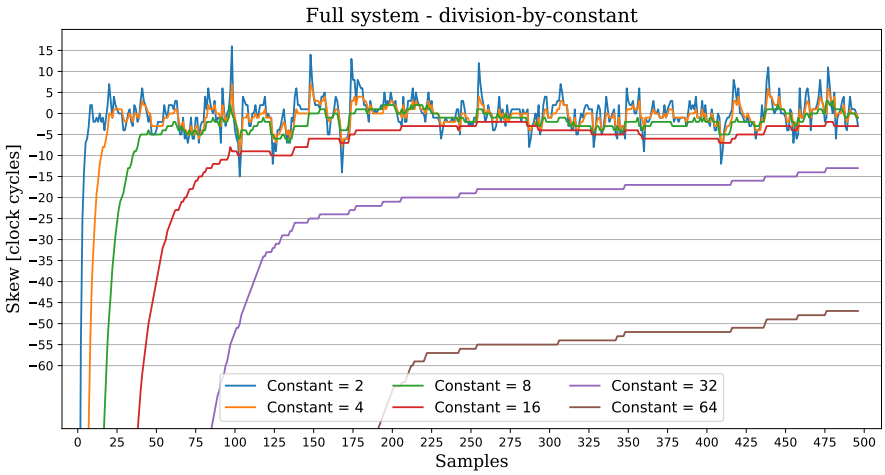


*Figure 5.15: Full system simulation results of division-by-constant filter, zoomed in*

To avoid the filtering of the initial offset, a division-by-constant filter is designed with a direct-jump, similar to the sliding window averaging. The results are depicted in Figure 5.16 and the performance parameters presented in Table 5.3. It can be seen that all the configurations perform better than the baseline as it directly jumps at the start and then continues to adjust the count filtered corrections. However, when the count is set to 64, there is no adjustments as all calculated offsets are filtered away. As a result, the skew is kept constant with the value the counter reached after the direct-jump, in this case three.

*Table 5.3: Results of skew and absolute jitter of division-by-constant filter with direct-jump*

| Filter type | No filter | Division-by-constant | | | | | |
|---|---|---|---|---|---|---|---|
| Window size/constant | | 2 | 4 | 8 | 16 | 32 | 64 |
| Median skew | 0 | 0 | 0 | -1 | -3 | -2 | 3 |
| Max skew | 28 | 16 | 7 | 4 | 3 | 3 | 3 |
| Min skew | -24 | -15 | -9 | -7 | -7 | -3 | 3 |
| Absolute jitter | 52 | 31 | 16 | 11 | 10 | 6 | 3 |

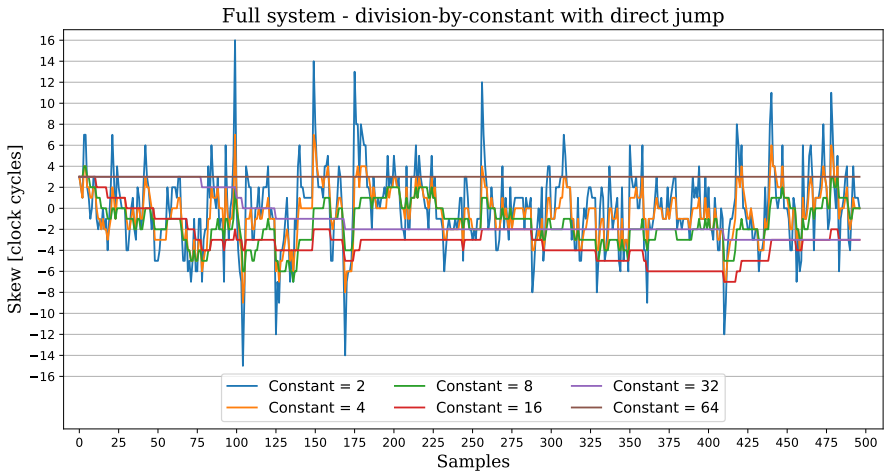


*Figure 5.16: Full system simulation results of division-by-constant filter with direct-jump*

Finally, the results of the full system simulation with the sliding window averaging filter are visualized in Figure 5.17 and Figure 5.18. The overview of the parameters is given in Table 5.4. It can be clearly seen that all parameters decrease as the window size increases. The filter with a window size 64 achieves a stable stable performance where the skew fluctuates around the zero and a max deviation of minus two. When comparing the results of this filter with that of the the division-by-constant with direct-jump, the performance is very similar. However, the sliding window averaging filter better keeps the ability to correct the count as the devision-by-constant filter damps the offset. For this reason, the sliding window average filter is used for the PoC.

*Table 5.4: Results of skew and absolute jitter of sliding window averaging filter*

| Filter type | No filter | Sliding window averaging | | | | | |
|---|---|---|---|---|---|---|---|
| Window size/constant | | 2 | 4 | 8 | 16 | 32 | 64 |
| Median skew | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Max skew | 28 | 18 | 10 | 7 | 3 | 3 | 1 |
| Min skew | -24 | -18 | -12 | -7 | -5 | -3 | -2 |
| Absolute jitter | 52 | 36 | 22 | 14 | 8 | 6 | 3 |

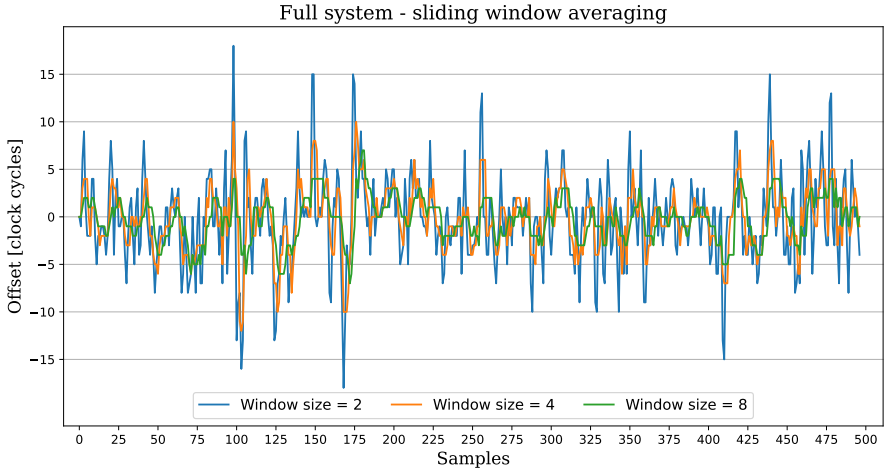


*Figure 5.17: Full system simulation results of sliding window averaging with window size 2, 4, 8*

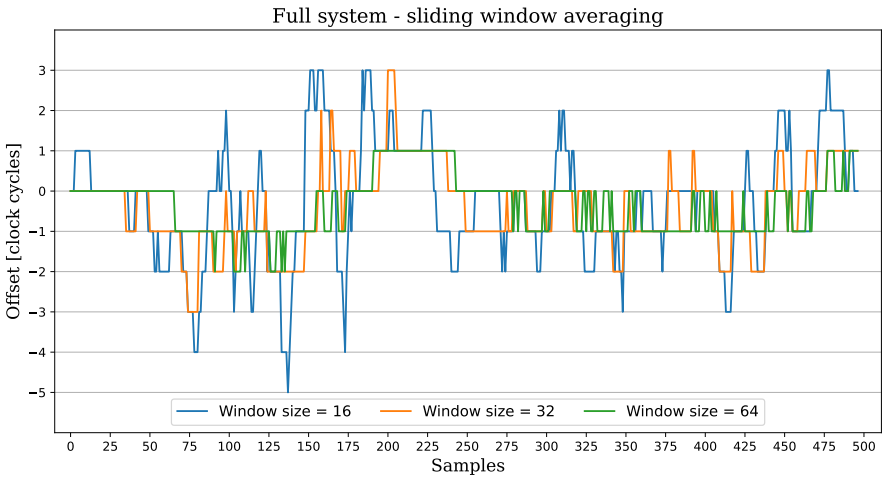


*Figure 5.18: Full system simulation results of sliding window averaging with window size 16, 32, 64*

## 5.3. FPGA Implementation and Evaluation

After simulating the synchronization design to validate and observe its behavior in a test environment, the mechanism was implemented in hardware to validate performance in real-life. The setup is similar to that of the full system simulation, one master containing the SyncMaster component attached to two endpoint devices, each with a SyncReceiver component. Each of these devices consist of a FPGA board with the same design as in the simulation. To monitor the heartbeats, an oscilloscope is used by which all three signals are visualized.

Figure 5.19 shows the result of the FPGA validation where the filters are configured to the sliding window averaging. It is a screenshot of the oscilloscope and depicts the rising edges of the three heartbeat signals, with the yellow one being the reference. Configurations of the endpoints are different. The first endpoint is set to the averaging filter with a window size of 64. The second endpoint is also configured to use the averaging filter, however here the window size is 32 which results in bigger absolute jitter.

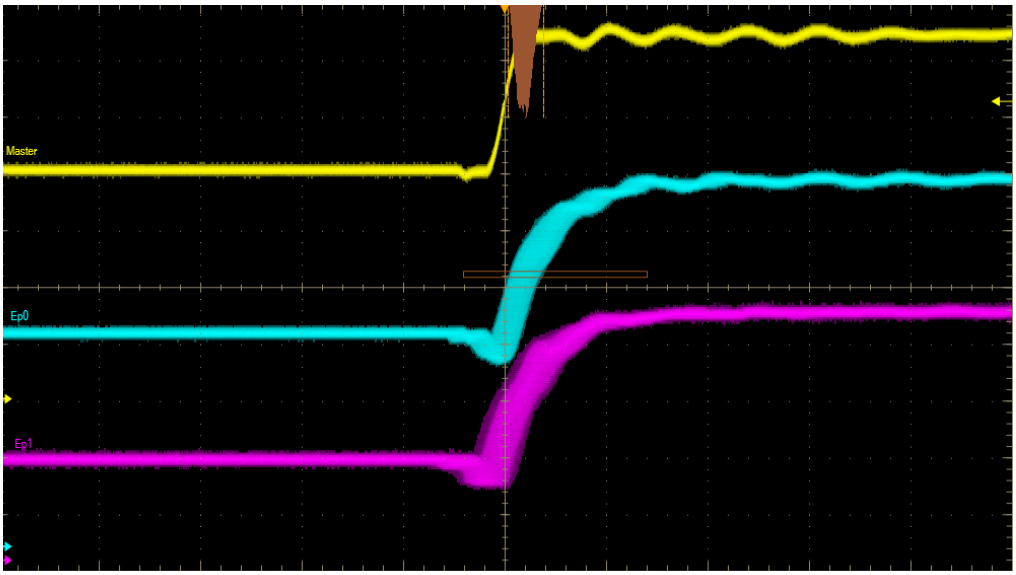


*Figure 5.19: Results of FPGA PoC implementation*

The figure shows the heartbeats of both endpoints drift away from the reference signal. Because of the synchronization mechanism, the drift is limited to a bounded range which is defined by the window size. In other words the mechanism ensures the heartbeat occurs within this range. Additionally both signals have a small skew in respect to the reference signal. Because this difference is static, it can be minimized by configuring a heartbeat offset such that the median of the skew is equal to zero. This configurable offset shifts the heartbeats by changing the scheduled time of these heartbeats.

The *time_clk* in this PoC is set to 390.625 MHz which corresponds to a time resolution of 2.56 ns. The results of the first endpoint, i.e., sliding window averaging filter with window size 64, is presented in Table 5.5. The absolute jitter measured during this PoC is 7 ns which corresponds with 2.73 clock cycles. This result is similar to that of the simulation where an absolute jitter was achieved of 3 clock cycles. The difference in these results are due to divergence between the simulation and the real-life situation.

*Table 5.5: Hardware results of the PoC*

| Value | Mean | Min | Max | Std Dev | Count |
|---|---|---|---|---|---|
| 7.0ns | 7.0n | 0.0 | 7.0n | 13.55y | 7.894k |

The PoC validates the performance of the proposed mechanism implemented into hardware (i.e., FPGA) by achieving even better results than in simulation. The behavior of the synchronized device is a stable heartbeat which only experiences a small bounded drift. Although there still are opportunities to improve the performance further, this achieved result has proven the potential of the mechanism in addition to meeting all requirements.

# 6

# Conclusions

In this thesis document the research and design of an in-band synchronization mechanism was presented, which goal was to achieve low nanosecond jitter to incorporate in RT systems. This document introduced the different state-of-the-art in-band methods. The current literature showed the interest in the field of synchronization, presenting numerous solutions. Still none of them met the requirements set for this project in terms of timing. The only available mechanism which achieved the desired robustness was the White Rabbit protocol, developed by CERN. However, this is a combination of PTP and SyncE which means this solution is a hybrid between in-band and out-of-band. Additionally, the White Rabbit protocol is implemented in an Ethernet based interconnect, resulting in some undesired properties for RT systems such as uncertain packet delivery. Nevertheless, the White Rabbit protocol showed the potential of the PTP principle.

The goal of this principle is to retrieve the offset between two time counters with the use of multiple timestamps. To ensure that the offset calculation is independent of the transmission latency, the triggers for the timestamps are sent in both directions. However, this only applies when this latency is symmetric, otherwise this violation results in an offset error. Additionally, the accuracy in which this offset can be retrieved depends on the precision in which the timestamps can be latched. In other words, the time should be represented in a high resolution in addition to having the timestamps acquisition as close as possible to the real departure and arrival of the trigger messages.

## 6.1. Thesis contribution

In the synchronization mechanism proposed in this thesis, weaknesses are addressed to improve the robustness and accuracy of the synchronization. First of all, a distinction is made between the two types of messages, namely data and trigger, used in the synchronization process. That made it possible to implement these with their own specific requirements in mind. For the trigger messages, this meant the implementation of a dedicated interface between the synchronization component and the interconnect transceiver. This direct connection is used to request the transfer of a trigger, but also the direct latching of departure and arrival of such a message. This resulted in a more accurate acquisition of timestamps. However, this strong-point is also a hard requirement which poses a limiting factor on the system it is implemented in. This is something that needs to be taken into the design of the device. Additionally, this also weights on the interconnect which has to be able to transmit these packages with high-priority. Although this sounds like a disadvantage to have such constraints, most of the communication protocols have such features and therefore can meet these synchronization requirements.

The second specific functionality of this design is a filter block to remove outliers in the offset calculations. The goal is to minimize the effects of incorrect offsets which are introduced due to jitter or asymmetric transaction latencies. By adding this filter block to the synchronization mechanism, the time corrections are smoother due to the removal of outliers and reduction of the adjustments. The addition of a filter block to the synchronization design increases the logical footprint. However, this is limited and would only be an issue in small applications where the logical area is restricted. In this design, two filter implementation types, the division-by-constant and the sliding window averaging, are presented and evaluated in simulation to find the desired behavior. The division-by-constant shows the damping off all inputs where the offsets smaller than the

constant become zero. Although this is desired for outliers, this behavior results in absence of adjustments for correct offsets. The sliding window averaging filter filters less although it spreads the larger offsets as it follows the input trend in terms of magnitude.

Performance was observed in simulations and hardware validations. Results show a reduction in both total and cycle-to-cycle jitter. The total jitter is brought under 7ns, which is lower than the requirements which where set and is a performance no other strictly in-band mechanism has achieved. In terms of cycle-to-cycle jitter, this is limited to two clocks. Although this could be improved even more down to one cycle, such robustness is a good result. The fact that the heartbeat can be configured and tuned with a resolution of nanoseconds offers great potential for RT systems and other applications.

## 6.2. Future work

Although the presented work meets the requirements set and achieved good performance, there are different parts of the system that can be improved. This section elaborates on some of these potential enhancements. The ones discussed in the following paragraphs are chosen as they could refine the functioning of the mechanism significantly, however they are not the only ones.

The synchronization mechanism as presented in this document showed good and reliable results in a setup with one reference clock device which synchronizes two others that are connected directly to it. However, the next step is to extend this to multiple hops, where the synchronized devices need to act as the reference for their connected devices. To do so, the counter of both synchronization components needs to be merged into one.

One of the augmentations of the new mechanism is the use of filters. Both filters that were implemented and tested had good results. Nevertheless, in a follow-up of the design, other types of filter could be investigated such as a finite-impulse response (FIR) filter. The advantage of this specific type is that each previous offset can be weighted individually in the formula depending on how old they are. This could make it possible to achieve the desired behavior more precisely.

This time synchronization corrected the counts of each SyncReceiver to ensure a global time throughout the network, even if each counter was driven by a clock with a slightly different frequency and phase. In other words, the mechanism corrected each time for the drift without adjusting the drift itself. However, there are mechanisms that do so such as the *White Rabbit* protocol where the clock is exchanged throughout the network by the *SyncE* part of the mechanism. The process of deriving the clock from the communication channel such that all device share a same clock is called *synthonization*. This is achieved by locking its local clock to the phase of the bit signal on the physical layer of the OSI model [2].

# A

# Definitions and Abbreviations

## A.1. Abbreviations

| | |
|---|---|
| AXI | Advanced eXtensible Interface |
| DSP | Digital Signal Processor |
| CDC | Clock Domain Crossing |
| CPU | Central Processing Unit |
| ECN | Engineering Change Notes |
| EMI | Electro Magnetic Interference |
| FIR | Finite Impulse Response |
| FPGA | Field Programmable Gate Array |
| GPU | Graphics Processing Unit |
| GPS | Global Positioning System |
| IO | Input/Output |
| IP | Intellectual Property |
| JTS | Jitter Timestamp |
| LSB | Least-Significant Bit |
| MECS | Multicast-event Control Symbols |
| MSB | Most-Significant Bit |
| NTP | Network Time Protocol |
| OSI | Open System Interconnect |
| PCI | Peripheral Component Interconnect |
| PCIe | PCI Express |
| PoC | Proof-of-Concept |
| PTP | Precision Time Protocol |
| RT | Real-Time |
| SyncMaster | Synchronization Master |
| SyncReceiver | Synchronization Receiver |
| TAI | International Atomic Time |
| UT1 | Universal Time |
| UTC | Coordinated Universal Time |
| VBR | Variable Bit-Rate |

## A.2. Technical definitions

| | |
|---|---|
| Clock | Periodic trigger. |
| Count | Integer which indicates the amount of pulses in a counter. |
| Counter | The IP block which keeps the count. |
| Drift | The change in skew due to (small) difference in the clock frequencies. |
| Event | A significant occurrence in hardware or software. |
| Frequency | Rate at which a event occurs. |
| Heartbeat | The execution clock used to indicate the instance to execute tasks. |
| Jitter | Change in skew. |
| *Absolute jitter* | The worst deviation in skew over time; in other words the sum of the largest skew in both directions [39]. |
| *Cycle-to-cycle* jitter | Change in skew between current and previous clock cycle[39]. |
| Latching | Lock on the current state (e.g., to latch the current time count). |
| Latency | Time between two events. |
| *Computational* latency | The time between the moment all data have arrived in the memory (or cache) and the calculated outputs are ready to be transmitted. |
| *Round-trip* latency | The time between the input data entering the network and the computed output leaving it. |
| *Transmission* latency | The time a message is inserted into the network until it arrives at its target receiver. |
| Operation frequency | Frequency at which the endpoints need to execute their task. |
| Period | The duration between two consecutive triggers. |
| Phase | Certain point during a clock period in reference to the trigger. |
| Real-time system | A system which correct operation depends on logical results of computation within a prerequisite time. [13] |
| Skew | Time difference between two triggers that ideally would occur simultaneously. |
| Synchronization | Precise equalization of a time component (i.e. frequency, phase or count). |
| Trigger | An event which starts an action. |

# Bibliography


[1] ESA, "Conventions: Time." https://sentinels.copernicus.eu/web/sentinel/user-guides/sentinel-3-altimetry/definitions/conventions.

[2] IEEE, "Ieee standard for a precision clock synchronization protocol for networked measurement and control systems," *IEEE Std 1588-2019 (Revision ofIEEE Std 1588-2008)*, pp. 1–499, 2020.

[3] D. Pedretti, M. Bellato, R. Isocrate, A. Bergnoli, R. Brugnera, D. Corti, F. Dal Corso, G. Galet, A. Garfagnini, A. Giaz, I. Lippi, F. Marini, G. Andronico, V. Antonelli, M. Baldoncini, E. Bernieri, A. Brigatti, A. Budano, M. Buscemi, S. Bussino, R. Caruso, D. Chiesa, C. Clementi, X. F. Ding, S. Dusini, A. Fabbri, R. Ford, A. Formozov, M. Giammarchi, M. Grassi, A. Insolia, P. Lombardi, F. Mantovani, S. M. Mari, C. Martellini, A. Martini, E. Meroni, L. Miramonti, S. Monforte, P. Montini, M. Montuschi, M. Nastasi, F. Ortica, A. Paoloni, E. Previtali, G. Ranucci, A. C. Re, B. Ricci, A. Romani, G. Salamanna, F. H. Sawy, G. Settanta, M. Sisti, C. Sirignano, L. Stanco, V. Strati, and G. Verde, "Nanoseconds timing system based on ieee 1588 fpga implementation," *IEEE Transactions on Nuclear Science*, vol. 66, no. 7, pp. 1151–1158, 2019.

[4] A. B. Mehta, *Clock Domain Crossing (CDC) Verification*, pp. 149–166. Cham: Springer International Publishing, 2018. https://doi.org/10.1007/978-3-319-59418-7_8.

[5] K. B. Pandit and S. R. Karbari, "Clock domain crossing—design, verification and sign-off," in *Proceeding of Fifth International Conference on Microelectronics, Computing and Communication Systems* (V. Nath and J. K. Mandal, eds.), (Singapore), pp. 627–639, Springer Singapore, 2021.

[6] R. Krohn, "The real-time health system: Adapting healthcare to the new normal," 30-11-2020. https://www.healthcareitnews.com/blog/real-time-health-system-adapting-healthcare-new-normal.

[7] O. Albahri, A. Zaidan, B. Zaidan, M. Hashim, A. Albahri, and M. Alsalem, "Real-time remote health-monitoring systems in a medical centre: A review of the provision of healthcare services-based body sensor information, open challenges and methodological aspects," *Journal of Medical Systems*, vol. 42, no. 164, p. 219–250, 2018. https://doi.org/10.1007/s10916-018-1006-6.

[8] M. Törngren, "Fundamentals of implementing real-time control applications in distributed computer systems," *Real-Time Systems*, vol. 14, p. 219–250, 1998. https://doi.org/10.1023/A:1007964222989.

[9] D. Lee, J. Allan, and S. Bennett, "Distributed real-time control systems using can," *Fieldbus Technology*, pp. 357–385, 2003. https://doi.org/10.1007/978-3-662-07219-6_15.

[10] S. Kuo, B. Lee, and W. Tian, *Real-Time Digital Signal Processing: Fundamentals, Implementations and Applications*. Wiley, 2013.

[11] A. Ibrahim, P. Gastaldo, H. Chible, and M. Valle, "Real-time digital signal processing based on fpgas for electronic skin implementation †," *Sensors*, vol. 17, no. 3, 2017. https://www.mdpi.com/1424-8220/17/3/558.

[12] F. Panzieri and R. Davoli, "Real time systems: A tutorial," in *Performance Evaluation of Computer and Communication Systems* (L. Donatiello and R. Nelson, eds.), (Berlin, Heidelberg), pp. 435–462, Springer Berlin Heidelberg, 1993.

[13] J. Martin, *Programming Real Time Computer Systems*. Old Tappan, NJ: Prentice Hall, 1965.

[14] Intel, "Real-time systems overview and examples." https://www.intel.com/content/www/us/en/robotics/real-time-systems.html.

[15] K. Bikos and G. Jones, "What is universal time?." https://www.timeanddate.com/time/universal-time.html.

[16] D. McCarthy and P. Seidelmann, *Time: From Earth Rotation to Atomic Physics*. Cambridge University Press, 2018.

[17] J. Eidson, *Measurement, Control, and Communication Using IEEE 1588*. Advances in Industrial Control, Springer London, 2006.

[18] D. L. Mills, "Network time protocol (version 3) specification, implementation and analysis," *RFC*, vol. 1305, pp. 1–109, 1992.

[19] M. Lipinski, "The white rabbit project," 2022. https://white-rabbit.web.cern.ch/.

[20] W. rabbit team, "The cern white rabbit project," *WSTS*, 2012.

[21] J.-L. Ferrant, M. Gilson, S. Jobert, M. Mayer, M. Ouellette, L. Montini, S. Rodrigues, and S. Ruffini, "Synchronous ethernet: a method to transport synchronization," *IEEE Communications Magazine*, vol. 46, no. 9, pp. 126–134, 2008.

[22] PCI-SIG, *PCI Express Base Specification 3.1a*, 07-12-2015.

[23] PCI-SIG, *ECN: Precision Time Measurement (PTM) 1.0a*, 31-03-2013.

[24] Z. Idrees, J. Granados, Y. Sun, S. Latif, L. Gong, Z. Zou, and L. Zheng, "Ieee 1588 for clock synchronization in industrial iot and related applications: A review on contributing technologies, protocols and enhancement methodologies," *IEEE Access*, vol. 8, pp. 155660–155678, 2020.

[25] T. Bigler and R. Exel, "Out-of-band asymmetry removal for high accuracy clock synchronization in 100base-tx," *2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings*, pp. 29–34, 2013.

[26] D. Poole, *Linear Algebra: A Modern Introduction*. Cengage Learning, 2014.

[27] B. Razavi, *Monolithic phase-locked loops and clock recovery circuits: theory and design*. John Wiley & Sons, 1996.

[28] T. von Lerber, S. Honkanen, A. Tervonen, H. Ludvigsen, and F. Küppers, "Optical clock recovery methods: Review," *Optical fiber technology*, vol. 15, no. 4, pp. 363–372, 2009.

[29] W. Su and I. Akyildiz, "The jitter time-stamp approach for clock recovery of real-time variable bit-rate traffic," *IEEE/ACM Transactions on Networking*, vol. 9, no. 6, pp. 746–754, 2001.

[30] K. S. Kim, "Asynchronous source clock frequency recovery through aperiodic packet streams," *IEEE Communications Letters*, vol. 17, no. 7, pp. 1455–1458, 2013.

[31] F. Cristian, "A probabilistic approach to distributed clock synchronization," *Proceedings. The 9th International Conference on Distributed Computing Systems*, pp. 288–296, 1989.

[32] RapidIO, *RapidIO™ Interconnect Specification Part 1: Input/Output Logical Specification*, 6 2017. Rev. 4.1.

[33] S. Fuller, *RapidIO: The Embedded System Interconnect*. Wiley, 2005.

[34] S. Rodrigues, "Ieee-1588 and synchronous ethernet in telecom," *2007 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pp. 138–142, 2007.

[35] E. Staff, "An introduction to synchronized ethernet." https://www.embedded.com/an-introduction-to-synchronized-ethernet/.

[36] T. Charboneau, "How "master" and "slave" terminology is being reexamined in electrical engineering." https://www.allaboutcircuits.com/news/how-master-slave-terminology-reexamined-in-electrical-engineering/.

[37] H. Zimmermann, "Osi reference model - the iso model of architecture for open systems interconnection," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, 1980.

[38] M. Smith, “The iso model of open systems interconnection (osi),” *Communications Standards*, p. 315–328, 1986.

[39] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital integrated circuits- A design perspective*. Prentice Hall, 2ed ed., 2004.

[40] ARM, *AMBA® AXI™ and ACE™ Protocol Specification Version E*, 22-02-2013.

[41] P. Ashenden, *The Student's Guide to VHDL*. Systems on Silicon, Elsevier Science, 2008.