UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Video Transport over Internet Protocol

Timing Synchronization

Dissertation presented by
Yolan FERY

for obtaining the Master's degree in
Computer Science and Engineering

Supervisors
Jean-Didier LEGAT, Pascal PELLEGRIN

Readers
Jean-Didier LEGAT, Pascal PELLEGRIN , Benoît MACQ

Academic year 2017-2018

Abstract

What about all the clocks in the world giving the same time ? Nobody would be late ? Maybe not. But the only thing that is sure is the following : it would considerably simplify the life of professional Video broadcasters. In fact, they are currently moving from SDI synchronous infrastructure towards asynchronous IP based networks. The benefits of IP transport are appealing. But a synchronization system should be provided.

This master thesis aims at providing a synchronization system for intoPIX, a professional in the video industry. The synchronization system takes the form of an Embedded system running on a FPGA. A Precision Time Protocol instance is running on a Real-Time Operating System and is interfaced to some hardware components providing clock tuning capabilities. A more in-depth study of the controller stability under hard constraints is given. The system achieves synchronization performances lower than 100 microseconds.

I would like to express my deep gratitude to Professor Jean-Didier Legat for his invaluable advice and accessibility. Thank you for the confidence you granted to me. Words are not sufficient to thank Pascal Pellegrin. His permanent enthusiasm, infinite patience and excellent pedagogy were the ingredients of my thirst to learn. My thanks are extended to the intoPIX team. Thank you for your warm welcome and your kindness. It was a pleasure to be part of such a smart and attentive family. Thanks to my –real– family and my friends for their support. Finally, I would also like to thank the Professor Benoît Macq to have accepted to evaluate that work.

Thanks again,

Yolan

Contents

Glossary

BMC Best Master Clock

CPU Central processing unit

FIFO First In, First Out

FPGA Field-Programmable Gate Array

HDMI High Definition Multimedia Interface

IP Internet Protocol

NTP Network Time Protocol

PI Proportional-Integral

PICXO Phase Interpolator Controlled crystal or Xtal Oscillator

PTP Precision Time Protocol

RTC Real-Time Counter

RTOS Real-Time Operating System

SDI Serial digital interface

UDP User Datagram Protocol

VCXO Voltage Controlled Crystal Oscillators

List of Figures

List of Tables

Chapter 1

Introduction

Abstract

Through this chapter, the general context of the subject is presented. The objectives raised by that context are listed and described. The followed methodology to reach the determined goals is exposed. After that, a list of the main results is given and lastly, an outline is opening the reading of the subsequent chapters.

1.1 Context

Here comes the Internet revolution ! The Internet Protocol (IP) has changed our lives and is currently metamorphosing those of professional Video broadcasters [24]. Actual distribution infrastructures, based on the Serial Digital Interface (SDI), are providing phenomenal image quality, low latency and jitter as well as enabling unidirectional protocol very simple to deploy [19]. But why should we leave such an exceptional solution ?

The advantages of IP are even more numerous and appealing :

- **Flexibility** and **Agility** : the data is split into packets and separated to its timing signal. Packets can travel the networks through different paths and be recombined at certain points.
- **Development** and **Investments** : Commercially Off-The-Shelf (COTS) IT-based infrastructure pushed forward by the IT industry can be leveraged.

But the migration from SDI to IP is giving rise to new challenges : higher latency and jitter must be mitigated. Moreover, packets can be lost and the asymmetrical path delays are introducing differences in upstream and downstream transport.

More importantly, the transition from the synchronous nature of SDI to the inherent asynchronous nature of IP networks requires a huge effort in time caring. Unlike SDI, the timing signal can not be transported along with the data stream, therefore the common solution is to see the data as a series of timestamped Events (as defined by the Definition 1).

Definition 1 (Timestamping)**.** *Timetsamping* is the process to assign to something a value sampled from a clock.

That solution is currently used by the Real Time Protocol (RTP) as a basis. RTP is comparable to UDP but carries a timestamp and a sequence number header allowing to detect if some packets were lost. That open opportunities like the one shown at Figure 1.1 : data like Video and Audio can be acquired from different sources, timestamped and recombined at the destination.

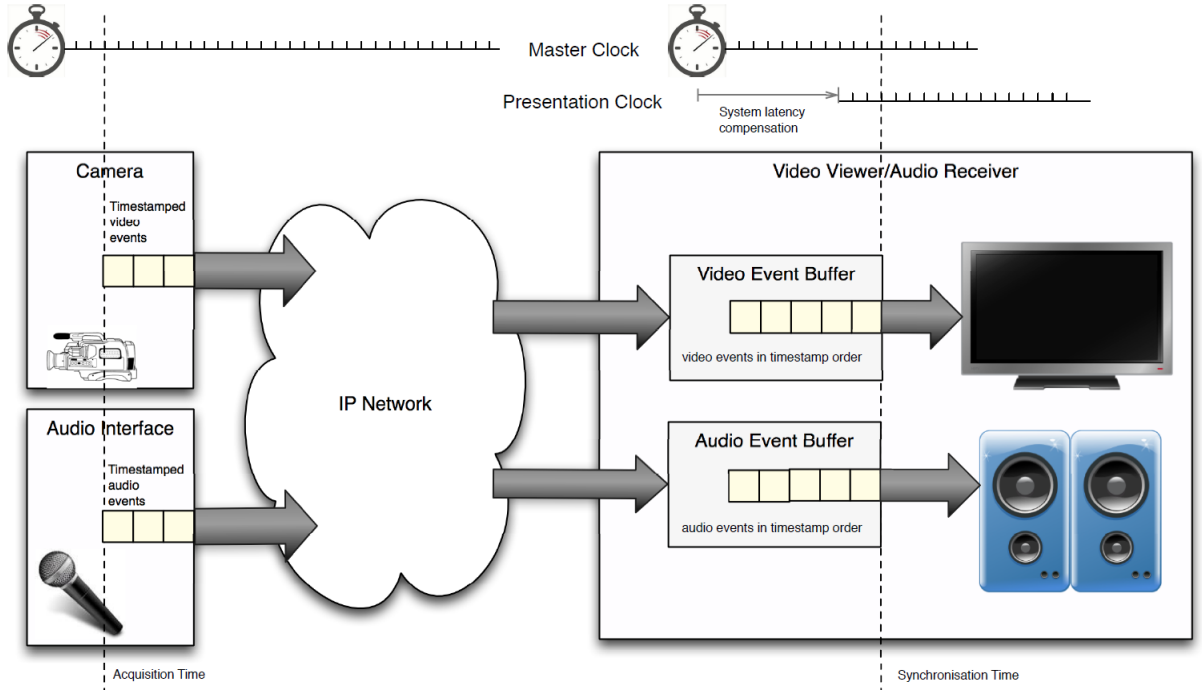


Figure 1.1: Data acquisition by different sources and synchronized reconstitution in an other node of an IP network [20]

But to allow consistent exchanges and utilization of such timestamped packets, the sampled clock should have a high-resolution and be distributed over every nodes communicating. That constraint is a big challenge and as a leader in the transport and compression of Video, intoPIX is currently facing that kind of challenge.

They have implemented a demonstration of the TICO-core, one of their flagship products, to compress a Video stream from one FPGA, send it via Ethernet to an other FPGA which is decoding and showing the result on a HDMI monitor. The TICO-core is a compression algorithm offering simultaneously low complexity, low latency and visually lossless quality at low compression ratio [11]. But the final demonstration is suffering from a flow rate problem. The clocks in the FPGA's are not synchronized and potentially running at frequencies slightly different from their nominal frequencies. This is due to the inherent error of crystal oscillators, defined by the manufacturer as the Frequency Tolerance (see Definition 2).

Definition 2 (Frequency Tolerance)**.** The *Frequency Tolerance* of an oscillator is its initial deviation frequency from its nominal frequency when measured at 25°C. Often expressed in *ppm* (parts per milion) it should not be confused to the Stability over Temperature defined at Definition 3.

Definition 3 (Stability over temperature)**.** The *Stability over temperature* over a defined operating temperature range of an oscillator is defined as the Frequency Deviation compared to the measured frequency at 25°C. It is also often expressed as *ppm*.

These errors cause the FPGA's to have a different notion of time and to send and consume packets at different rates and at different times. As shown at Figure 1.2, a frame buffer has been deployed in order to avoid packets to be discarded in the decoder. But this is only a temporary solution since buffer overflow or even underflow can still occur. An obvious need for a synchronization process is observed. Since the HDMI clock is determining the rate of consuming of the packets in the frame buffer, that should be that clock synchronized with the sender's one.

To tackle these common problems and embrace the transition from SDI to IP, a few standards have been published to give a prevalent solution for all actors in the broadcast industry.

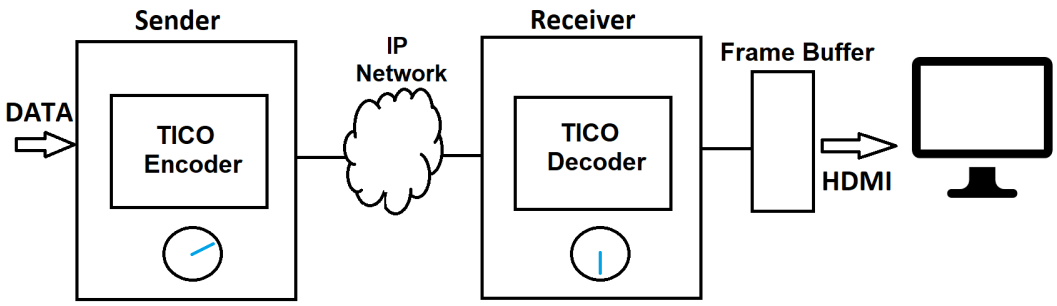


Figure 1.2: Abstract representation of the TICO-core over IP demonstration

The SMPTE ST2022 Standard is designed to make the transition between SDI and IP by defining the transport of uncompressed SDI video, audio and metadata over RTP/UDP [19].

The Standard IEEE 1588 defines a synchronization protocol.

The SMPTE ST2059 defines a profile for the IEEE 1588 in order to guarantee a synchronization of each entrant element in the network within 5 seconds and a synchronization range under the μs.

1.2 Objectives

To define the objectives of the thesis, we first have to define the Syntonization and the Synchronization (see Definitions 4 and 5).

Definition 4 (Syntonization)**.** In the context of oscillators, the *syntonization* is the process to make oscillators syntonized. Oscillators are syntonized if they oscillate at the same frequency.

Definition 5 (Synchronization)**.** In the context of clocks, the *synchronization* is the process to make clocks synchronized. Clocks are synchronized at time t if sampled at time t they are returning the same value.

Remark 1. If we want clocks to be synchronized at any time, syntonization is required.

Firstly, an analysis of the cutting-edge tools available to achieve timing synchronization should be presented.

The main goal of the thesis is to implement a stable system of clocks synchronization in the sender and receiver of the TICO demonstration in order to avoid buffer overflow and underflow of the frame buffer. The only allowed mode of communication between the two is the Ethernet interface. The resource utilization by the solution should be minimized in area and network load. The convergence time of the system is not a primary concern but a discussion on how to reduce it should be provided.

Since the frame buffer is expensive in term in of memory usage, the synchronization accuracy should be reduced at maximum to allow to reduce its size. A sizing of the buffer is then required.

1.3 Methodology

We started the project by reading a few invaluable books ([17, 6]) providing a good immersion in the closed world of Video broadcast industry and timing synchronization. The presence in the office of intoPIX company was maximized by, for instance, the realization of an internship coupled to the master thesis. That presence allowed to gain the vocabulary of the domain and understand the day-to-day challenges the company is tackling.

Once the immersion had been done, a study of some more specific elements has been required. It has been performed by reading papers. At first, the papers were very general (like [24]) but

progressively these papers were making arise additional questions that could only be answered by reading more specific papers (like [15]). At the end of this step, a draft of a first architecture for the system accomplishing the goals has been performed.

After that, some models, simulations and emulations (like the one presented at Appendix C) have been performed to rapidly evaluate if the potential solutions are sustainable and could satisfy intoPIX requirements and therefore reduce the risks.

Since the tasks were relatively new for us, using an Agile methodology for the development was hard since we did not know our velocity at all. We still tried to follow such a methodology by for instance taking part to Scrum meetings each day in the company at noon. We also used a version control tool (git) for the software part.

Since the software is resulting from interfaces modifications of an open-source project, it has not been tested intensively. The adjustable clock has been made from scratch and therefore tested using simulations on each main part of the system and then on the global block.

1.4 Main results

The main numerical and technical obtained results are listed hereafter.

- An innovative study of the stability of a control loop under hard constraints has been performed.
- An adaptation of a PTP stack has been made to make it run on a MicroBlaze System interface to Ethernet running the FreeRTOS Real-Time Operating System. That includes network management modifications and a fully interface realization with the hardware components like an adjustable clock.It results in a fully functional synchronization system with low CPU utilization, frequency tracking and synchronization range of $\pm$ 100 μs.
- A sizing of the decoder frame buffer has been proposed. A buffer of size one can be used under the condition of delaying the decoder by at least 100 μs.

1.5 Outline

The thesis is organized as following.

The State-of-the-art Chapter presents the current available tools to measure, filter, command and reduce the synchronization error.

The Chapter 3 gives the details of the implementation by starting with a global architecture of the system. Then it dives into design choices for the hardware and software to achieve the fixed goals.

The Chapter 4 assesses the performances of the implemented System and proposes a sizing of the frame buffer in consequence of the performances.

Finally, the Chapter 5 synthesizes the obtained results and proposes some perspectives.

Chapter 2

State-of-the-art

Abstract

In the following Chapter a study of the cutting-edge tools to achieve timing synchronization is given. The first step to synchronize systems is to measure the error that should be corrected. This is the goal of the two protocols presented. Measures are intrinsically noisy, therefore, we discuss the filtering methods. There is also an obvious need for a mechanism to correct the error in the most efficient way. Since the error is about time, clock servos are studied. Finally, different timestampers are analyzed and evaluated in their ability to provide a precise sample of the current time.

2.1 Protocols

The project requires a clock synchronisation with a sub-microsecond precision because if the error between the encoder and decoder is less than 17 ms (duration of a video frame at 60FPS) a buffer of size one can be used. And only one channel is available for the communication between devices : Ethernet. Hence, a time synchronisation protocol over Ethernet has to be defined or chosen.

It is pointless to define a new protocol : multiple protocols tending to perform that function have been standardized and deployed. This Section gives a condensed overview of 2 well-established protocols : the Precision Time Protocol and the Network Time Protocol. After that, in order to pick the more appropriate one, a comparison, based on relevant criteria for the project, is given.

2.1.1 Precision Time Protocol

The Precision Time Protocol has been normalized under the name IEEE 1588 [5]. A first version (PTPv1) of the protocol has been published in 2001 and is now superseded by a more complete version (PTPv2) since 2008. Unfortunately, due to changes in message formats, the new version is not compatible with the old one [21]. Thus, caution must be taken while deploying PTP, to use the same — and preferably the newest — version of the protocol on the nodes requiring synchronisation. For the sake of simplicity, the following description will only focus on the last version of the protocol.

The protocol enables synchronisation of clocks of several inherent stability, resolution and precision with a Grandmaster clock dynamically elected. The message exchanging mechanism is designed to work for systems communicating by local area networks including, but not limited to, Ethernet [5]. Messages can be exchanged using the UDP/IP layer, the DeviceNet layer or even, directly the Ethernet layer.

The standard defines 2 modes : End-to-End (E2E) and Peer-to-Peer (P2P). The first is the default mode and only requires that ending nodes of the network are IEEE 1588 compliant. The last is more accurate by computing network link delays but requires that each node of the network, including routers and switches, is IEEE 1588 compliant [1].

From a precision point of view, some PTP implementations have permitted to reach sub-nanosecond synchronisation [8, 10].

Best Master Clock Algorithm

The IEEE 1588 is designed as a Master-Slave architecture. A Grandmaster is elected and shares its time with the slaves. The election is done via the Best Master Clock Algorithm by taking into account the clock quality of each device in the network. Basically, the election process is done as follow :

- The Grandmaster regularly publishes its clock quality.
- Each device compares the received clock quality with its own. If the device clock quality is better than the received one, the device starts publishing its own clock quality in the network.
- If the Grandmaster receives a better clock quality, it immediately leaves the Grandmaster state and starts the slave mode.
- If after a certain amount of time, a slave has not received a message announcing a better clock quality than its own, it takes the Grandmaster role.

That dynamic configuration allows the network to be administration free while ensuring to maximize the clock quality of the Grandmaster.

PTP device types

In PTP networks, there are 5 types of device. These types are described in the Table 2.1.

Type	Description
Ordinary Clock	That clock tries to get synchronized accurately to the PTP clock in order to use it for another job.
Boundary Clock	Network element including a PTP clock (e.g. : bridge, router or repeater). The goal of that Clock is to create sub-networks and limit the PTP packets exchanged.
End-to-end Transparent Clock	Pass-through updating the residence time correction field in PTP packets. It optionally contains a PTP clock since residence time is relative.
Peer-to-peer Transparent Clock	Pass-through updating the residence time correction field in PTP packets. It also computes and transmits the path delay.
Management Node	Source of management messages. It do not need to contain a PTP clock.

Table 2.1: PTP device types

PTP message classes

Messages exchanged by PTP devices can be Event or General messages. Event messages are timed at both sending and reception. They are received on port 319. The Table 2.2 is listing the

set of Event messages as well as their role and the sender's state. General messages do not have to be timestamped and are sent to port number 320. A list of the set of General messages along with their role and their sender's state is given at the Table 2.3.

Name	Sender's state	Function
Sync	Master	Starts a synchronization process, it should be followed by a **Follow_Up** message giving a more accurate timestamp of when the message left the device. It allows the computation of the delay Master to Slave.
Delay_Req	Slave	Starts a request for the delay Salve to Master.
Pdelay_Req	Master/Slave (P2P only)	Starts a request for the link delay.
Pdelay_Resp	Master/Slave (P2P only)	Responds to a request for the link delay. It should be followed by a **Pdelay_Resp_Follow_Up** message giving a more accurate timestamp of when the message left the device.

Table 2.2: Set of Event messages

Name	Sender's state	Function
Announce	Master/Slave	Establishes synchronisation hierarchy.
Follow_Up	Master	Gives a more accurate timestamp of when a **Sync** message left the device.
Delay_Resp	Master	Responds to a request for the delay Slave to Master.
Pdelay_Resp_Follow_Up	Master/Slave (P2P only)	Gives a more accurate timestamp of when a **Pdelay_Resp** message left the device.
Management	Any	Manage the PTP clocks data sets.
Signaling	Master/Slave	Communication and negotiation between PTP clocks.

Table 2.3: Set of General messages

Synchronisation process

The goal of the protocol message exchange process is to allow slaves to gather enough data to synchronize and syntonize their clock with the Grandmaster one. The abstract network topology that the BMC gives to the network is shown at the Figure 2.1.

Recurrent (around 1 per second by default) **Sync** and **Follow_Up** messages sent by the Grandmaster can be used to syntonize the clocks. By using timestamps generated at sending and receiving of the messages as shown at the Figure 2.2a, we can compare intervals of time that should be equal in both the Master and the Slave. By using these intervals, we can compute the

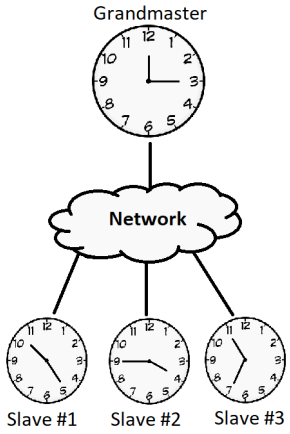
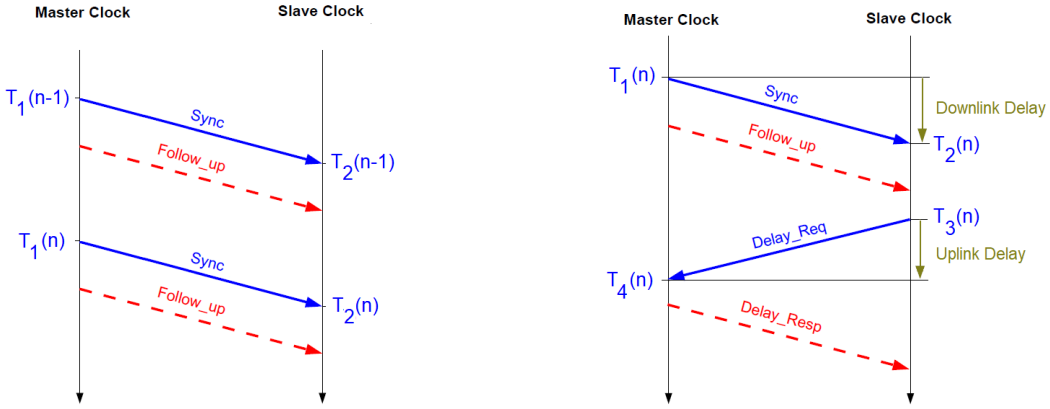


Figure 2.1: Abstract PTP network topology

frequency offset the Master has comparing to the Slave :

$$\text{Frequency offset (ppm)} = \left(\frac{T_1(n) - T_1(n-k)}{T_2(n) - T_2(n-k)} - 1\right) \cdot 10^6 \tag{2.1}$$



(a) Syntonization

(b) Synchronisation

Figure 2.2: PTP Syntonization and Synchronization principles

We can also compute the delay between the Master and the Slave :

$$\text{Downlink Delay} = T_2(n) - T_1(n) \tag{2.2}$$

To compute the delay between the Slave and the Master we need the **Delay_Req** and **Delay_rep** messages. The complete message exchange process is shown at the Figure 2.2b.

$$\text{Uplink delay} = T_4(n) - T_3(n) \tag{2.3}$$

By definition, the Mean Path Delay is equal to the average of the Downlink Delay (2.2) and the Uplink Delay (2.3) :

$$\text{Mean Path Delay} = \frac{(T_2(n) - T_1(n)) + (T_4(n) - T_3(n))}{2} \tag{2.4}$$

The difference in phase between the Slave and the Master is :

$$\text{Offset from Master} = \text{Downlink Delay} - \text{Mean Path Delay} \tag{2.5}$$

By using (2.2) and (2.4), we finally obtain :

$$\text{Offset from Master} = \frac{(T_2(n) - T_1(n)) - (T_4(n) - T_3(n))}{2} \tag{2.6}$$

To achieve the syntonization and the synchronisation of the clocks of the PTP network, by following the Definitions 4 and 5, we want the frequency offset (2.1) and the offset from Master (2.6) to be equal to zero for each Slave in the network.

2.1.2 Network Time Protocol

Published during the eighties by David L. Mills, the Network Time Protocol is currently one of the oldest network protocol still in use. Trusted-time server are used as reference to synchronize the clocks of the network. Messages are exchanged on the UDP/IP layer. Typical achievable synchronization accuracy is about a few milliseconds but highly depends on the number of network elements between trusted server and ending nodes [14, 13].

The architecture of the NTP network is organized in levels called *Stratum*. *Strata* are composed of multiple servers serving their time to lower levels. The root *Stratum* is typically composed of national standards or on-site atomic standard clocks [14].

Since delays between the first *Stratum* and ending nodes are determinant in the final accuracy, a shortest-path spanning tree is built for all the clients by using a Bellman-Ford algorithm [12].

NTP message classes

Main messages exchanged by NTP devices are timed at both sending and reception. They are received on port 123. The Table 2.4 is listing the main set of NTP messages as well as their function.

Name	Function
NTP_Req	Starts a request for the delay Salve to Master and gives the delay Master to Slave.
NTP_Resp	Responds for the delay Salve to Master.

Table 2.4: Main set of NTP messages

Synchronisation process

The synchronisation process (shown at Figure 2.3) of the NTP is almost identical to the PTP one. Thus, Equations from (2.1) to (2.6) are still relevant here.

We can highlight the main difference with PTP: the **Follow_Up** message mechanism is not implemented for the NTP. It undeniably affects the accuracy of the resulting system by reducing the precision of the timestamping operation. This concern is more thoroughly studied in the Section 2.4.

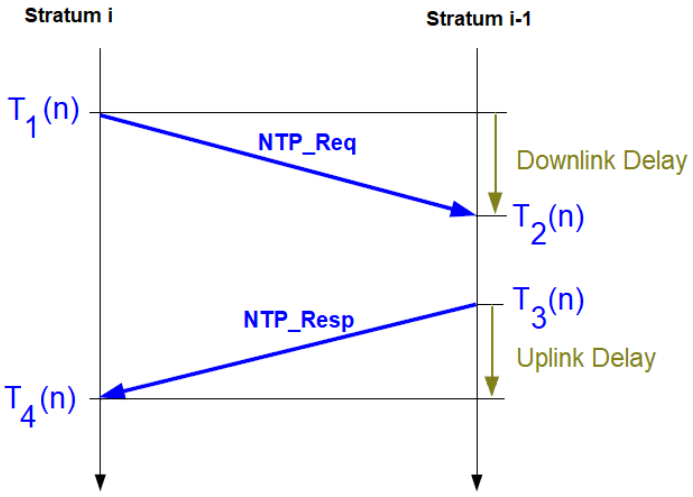


Figure 2.3: NTP Synchronisation principle

2.1.3 Conclusion

The Table 2.5 is synthesizing previous results and gives a comparison between PTP and NTP.

The Precision Time Protocol can achieve nanosecond precision synchronisation by using hardware timestamping and by requiring a PTP clock on each device on the path of the clocks of interest (e.g. : routers, bridges and switches). These opportunities, in addition with the fact that PTP is recent and still not well-established, increase the total cost of deployment. Finally, the PTP is administration free and can be isolated from any server.

The Network Time Protocol can achieve millisecond precision synchronisation without hardware timestamping. This protocol is currently well-established but need to be statically instantiated, configured and maintained.

For the current project, we have chosen to pick the PTP since the accuracy and architecture correspond to the requirements and it is the advised protocol for time distribution by the standard of the Video broadcasters industry.

Protocol	Accuracy	Architecture	Hardware Timestamping	Cost
PTP	$\sim$ ns	Dynamic Master-Slave	Highly recommended	High
NTP	$\sim$ ms	Static Stratous	Authorized	Low

Table 2.5: Comparison of PTP and NTP

2.2 Filters

Focus during the implementation was not given to the filtering part since it does not avoid the system to work. Therefore it was not studied in depth here. But most reported implementations of filters for PTP and NTP are in fact Kalman filters [2, 3, 4].

2.3 Clock Servo

A classical way to compute a command in a feedback system is to use a Proportional and Integral (PI) controller. The Proportional term correct the current measured error and the goal of the Integral term is to guarantees that there is no steady-state error due to control mechanism.

The command have then to feed a clock controllable in frequency. Two main methods can be highlighted [6]:

- Pulse addition and swallowing
- Voltage-Controlled oscillator

The first can be described with the help of the Figure 2.4. It is based on a mechanism of "invisible" fractional part of the actual counter. In this example, the final counter, giving the elapsed time is in fact the MSB of the of a large number composed of the "counter" and "adder" blocks on the Figure. The counter LSB is 16 ns whereas the adder LSB is 1 ns. The adjustment control block selects 15, 16 or 17 if the final clock should be decelerated,stay stable or accelerated. That can inject or retrieve in the fractional part an additional nanosecond. The problem of that kind of circuitry is that, it can happen that at certain cycles, the counter does not change. For instance, if the actual value of the adder is 15, if 15 is added again, there is no overflow visible by the counter since $15 + 15 < 32$.

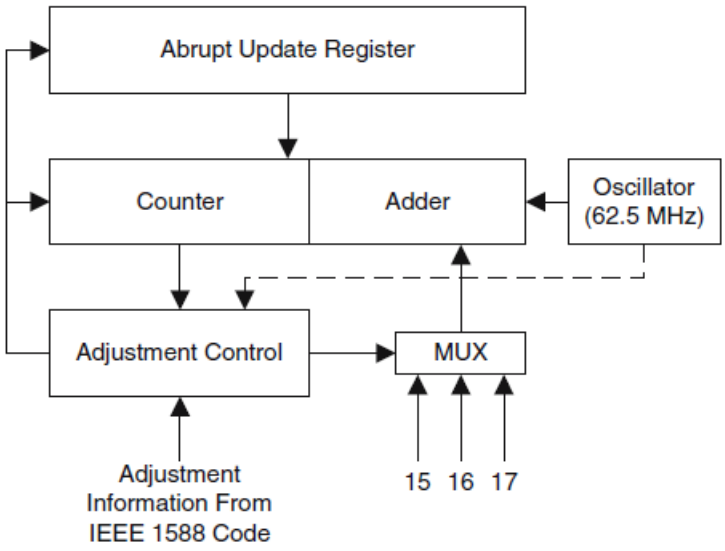


Figure 2.4: Clock adjustable in frequency by pulse addition and swallowing [6]

The second consist of an analog circuit where supplied voltage determines the frequency of the oscillator. It is a sustainable solution and offers better performances than the second [6]. But VCXO are generally external components added to the FPGA, they carry a power and cost penalty [22]. Moreover it can introduce more interference and cross talk at the board level [22].

2.4 Timestamper

The two protocols presented earlier need to associate a time value to some events like for instance the departure or reception of a protocol packet. Since the protocols are trying to measure the network delay as accurately as possible, the delay fluctuations should be reduced. As shown at the Figure 2.5, multiple options are open to capture timestamps. Software timestamping is subject to a lot of errors due to the protocol stack delay fluctuations, ranging from hundreds of microseconds to milliseconds [9]. Interrupt level timestamping is more precise : tens of microseconds [9]. Finally hardware timestamping, as close as possible to the physical layer, generates only timestamps with a few nanoseconds of errors [9]. The Table 2.6 is summarizing the presented previous results. As a matter of fact, the accuracy of the whole system is affected

by the accuracy of the timestamping. Patrick Ohly *et. al.* [15] have demonstrated that PTP can have a precision 50 times better with hardware timestamping.

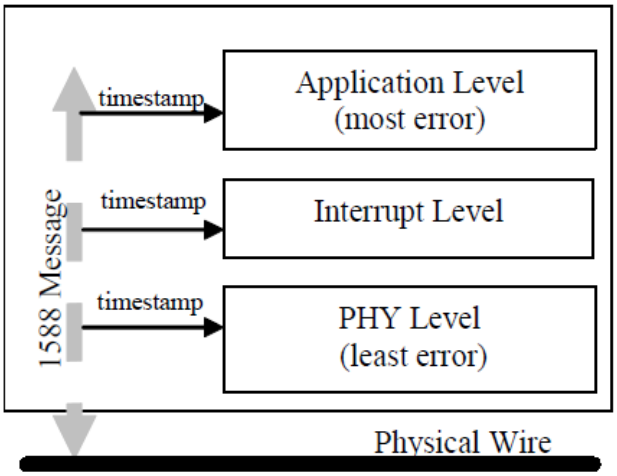


Figure 2.5: Timestamping at different layers [9]

Layer	Error
Application	0.1 - 10 ms
Interrupt	10 μs
PHY level	1-10 ns

Table 2.6: Timestamp errors in function of the timestamping layer

2.5 Summary

From these studies of different available tools to deal with time measurements, filtering and control. An ideal general system could be defined if no constraint (on cost) is specified : it would be a PTP instance using a Kalman filter, a PI controller driving a VCXO and with a hardware timestamp unit.

Chapter 3

Implementation

Abstract

This Chapter presents the system that has been implemented in order to achieve the defined goals. An overview of the whole system architecture is given. The system consists in a Hardware and a Software. A description and a discussion of the design choices is presented for both.

3.1 System Architecture

The State-of-the-art chapter allowed us to list and compare the best tools and perspectives we have to provide a system achieving the defined goals. In short, the ideal system, represented at the Figure 3.1, would be composed of :

- A **PTP instance** running along with the TICO encoder/decoder. The instance measures the offset and drift between all TICO encoders/decoders HDMI clocks in the network.
- A **Kalman Filter** cleaning the noisy measures taken by the PTP instance.
- A **Clock Servo** composed of an Adjustable Clock and its associated Control System. Ideally the Adjustable Clock should be adjustable in time but also fed by an oscillator controllable in frequency. The oscillator is also feeding the HDMI contained in the TICO. The Control System would be a stable PI controller.
- A precise **Timestamp Unit** for the PTP packets.

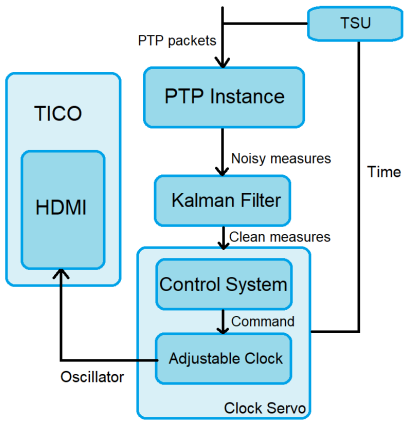


Figure 3.1: Ideal schema block of the System

Since we are targeting a FPGA, the ideal system described above can either be implemented as pure hardware or as a software running on a softcore (optionally interfaced to hardware components). Even if rare authors [16] have accomplished the laborious task to write a pure hardware PTP system, this is not advised and not the trend we are observing in the literature. In fact , pure hardware implementation suffers from the following issues :

- Long and tedious **development**. A lot of cases have to be considered in the protocol process and needless to say that low-level implementations are longer than high-level developments.
- Hard **maintainability**. PTP is already at its second version which is not compatible with the first one. The protocol should follow the technology improvements and is therefore subject to frequent updates. But Hardware modification can be expensive, even more if the circuit has been printed...
- Huge **logic and area utilization**. The protocol is not computationally intensive and is more about control and states transitions. Performances and speed is not a concern : even a low-frequency clocked CPU can achieve PTP processing and throughput. Consequently, the need for specific dedicated circuitry is badly justified.

Luckily, the second alternative is more promising. Even more if, as in intoPIX systems, a softcore is already deployed. In fact, intoPIX have deployed a Microblaze (a Xilinx softcore) along with their Hardware TICO-core to initialize, configure and control the core but also the Ethernet and HDMI interfaces. Hence, the smart move, here, would be to leverage that softcore to minimize the number of additional FPGA resources used.

Therefore the global system would look like a Softcore system running in parallel the TICO software and the PTP instance. It would be connected, in addition to the the configuration ports of the TICO-core, the HDMI and the Ethernet interfaces, to a Clock adjustable in time and frequency. Since performance in terms of computation speed is not a concern, the control system can be implemented in pure software. The Softcore would be also capable of sending and receiving packets on an Ethernet port. This is achieved by connecting the Softcore to an Ethernet interface.

The following sections are describing the design steps, issues met and final implemented results for each of the ideal parts presented above. To accelerate the development process, we decided to put, for the moment, integration into intoPIX product, filtering and hardware timestamping aside to focus on the synchronisation system.

3.2 Hardware

In this section, the elements composing the Hardware system are described.

3.2.1 System on Programmable Chip

As already mentioned earlier, a Softcore is needed to provide the ability to run software programs. Since intoPIX TICO demonstration contains a Microblaze, we decided to work on that kind of Xilinx Softcore. Obviously, a MicroBlaze can not work alone, it needs specific attached components to form a working system. For instance, if we need to run a program, we need some memory to store the program but also to store the data. The system that has been implemented for the project is depicted at the Figure 3.2. It is a typical Microblaze system capable of sending and receiving data via Ethernet. Each part of this system is more thoroughly described hereafter.

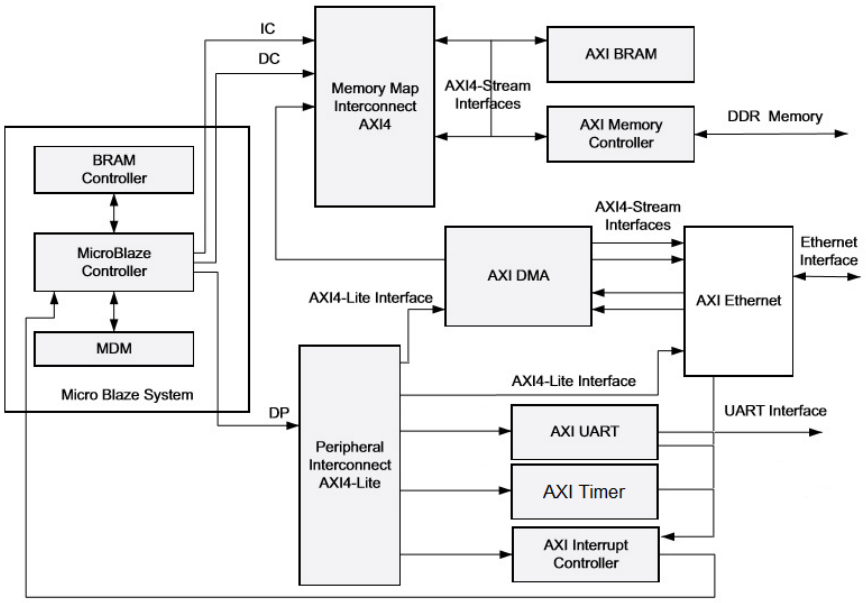


Figure 3.2: Microblaze system with Ethernet interface

MicroBlaze System

Composed of the MicroBlaze Processor, a MicroBlaze Debug Module and a BRAM Controller, it is the core of system. It allows to execute Instructions from the BRAM on the processor. Memory for the program is allocated and accessed in the BRAM thanks to the BRAM Controller. The Debug Module enable Debug mode in the Software Development Kit including Step-by-Step execution and registers value information. The MicroBlaze can be customized to fit the needs of the software running on it.

Since performances of the PTP does not rely mainly on computation speed, not that much focus was given to the tuning of the processor. However, we still took care to enable hardware operators like multiplier and divider since they are common operations in offset, drift and control computation.

AXI Protocol

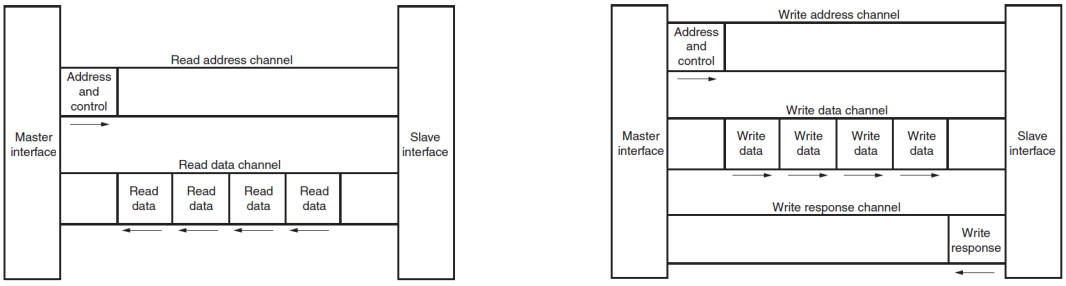
The communication between the Microblaze and other IP (Intellectual Property) cores is done via a protocol called AXI. By taking advantage of the abstraction principle, it is not needed to understand and master that protocol to develop and connect the MicroBlaze system. However, if we want to implement some cutom IP connected to the MicroBlaze, we should master the protocol in order to develop an AXI Interface in the IP and connect it to the logic.

AXI4 is –counter-intuitively– the second version of the AXI protocol. There are three kinds of AXI4 interfaces [23]:

- **AXI4** : memory-mapped interface with high-performances. Mainly used for raw data transfers.
- **AXI4-Lite** : memory-mapped interface with simple and low-performances. Essentially used to write and read control and status registers.
- **AXI4-Stream** data streaming interface. Most suitable for inline large bunches of data transfer.

AXI4 interfaces are connected and communicating in a static Master-Slave fashion. Read and write operations are available and illustrated at Figures 3.3a and 3.3b. Read and write

operations can occur in both directions simultaneously. Dedicated channels allow to control the flow and the address actually read/written in the memory-mapped interfaces.



(a) Read channels Architecture

(b) Write channels Architecture

Figure 3.3: AXI4 Read and Write Channels [23]

AXI Interconnects

AXI Interconnects allow to map one or more AXI Master to one or more AXI Slaves. It is only intended for AXI4 and AXI-Lite : AXI4-Stream are commonly connected one-to-one or to DMA.

AXI BRAM

Memory containing the Instructions and Data of the software accessible via AXI protocol.

AXI Memory Controller

This component provides AXI access to the external DDR4 SDRAM. That memory location is used to store Ethernet packets received.

AXI Ethernet

This IP provides an implementation of an Ethernet MAC allowing to send and receive Ethernet packets to the external PHY (physical layer).

AXI DMA

The AXI Direct Memory Access is bridging the Ethernet AXI-Streams to and from the external DDR3 SDRAM (via the AXI Memory Controller).

AXI UART

The AXI UART is an interface to send/receive data via UART. It is a suitable way to print debug messages in the console of the Software Development Kit or to allow configuration of the FPGA from the console without compiling, programming and launching again the Software.

AXI Timer

This component is required by some Operating System running on the MicroBlaze. The name is explicit, that IP provides basic and intuitive timer capabilities.

AXI Interrupt Controller

Interrupts are for some Operating System (e.g. Real-Time OS) the essence of responsiveness. That is why an AXI Interrupt Controller has been deployed in order to manage interrupts coming from multiple sources. Three main sources have to be identified :

- **AXI Ethernet** : signaling reception of Ethernet packets.
- **AXI UART** : signaling reception of data.
- **AXI Timer** : signaling the expiration of a timer.

3.2.2 Adjustable clock

All the work is about synchronization of clocks. Therefore a precise and adjustable clock should be provided. The clock should be able to :

1. Track the time elapsed since a certain point in time (often the Unix Epoch, in January 1, 1970) in seconds and nanoseconds. The time should be tracked by the PTP clock.
2. Give the time to the hardware and the software.
3. Accept offset commands in time (seconds and nanoseconds).
4. Accept offset commands in frequency (ppm).

In our context, the PTP clock is the HDMI clock going to the HDMI transceiver. The time can be given to the hardware by direct wires but should be connected to the MicroBlaze via AXI protocol to give it to the software. Since the measures of the offset and the drift as well as the commands are computed in the MicroBlaze, the clock should also be able to accept commands via AXI protocol. A problem appears since the commands are in the AXI clock domain and the counter is evidently in the HDMI clock domain. We observe a clock domain crossing problem that we solved by using FIFO buffers. All that requirements and issues led us to define the Architecture presented at the Figure 3.4.

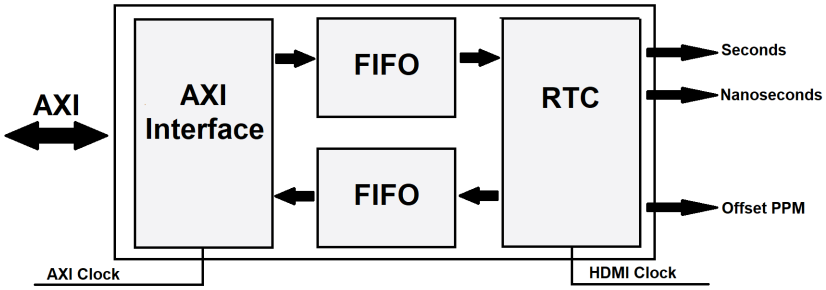


Figure 3.4: Adjustable Clock Architecture

Finally, the oscillator feeding the HDMI clock should be able to be offset in frequency. It as accomplished by taking advantage of some HDMI transceiver capabilities and a PICXO exploiting these capabilities. The Figure 3.5 represents the connections between these three blocks. We describe more in details each part of that Architecture in the following sections.

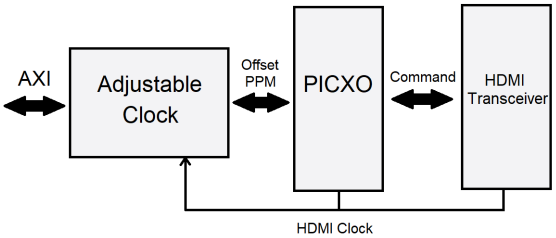


Figure 3.5: Adjustable Clock Connection with PICXO and HDMI transceiver.

AXI Interface

The goal of the AXI communications for the adjustable clock is to update the value of a bunch of registers and read values of some others. The AXI4-Lite is therefore the best suited AXI protocol for that kind of operations. To complete each one of the clock objectives, a set of 7 registers is defined. The list of the registers along with their address and function is given at the Table 3.1.

Name	Address	Function
Offset seconds	0x0	Offset the register containing the seconds.
Offset nanoseconds	0x1	Offset the register containing the nanoseconds.
Update PPM	0x2	Update the frequency offset of the HDMI Clock.
Sample Time	0x3	Sample the RTC and update registers 4, 5 and 6.
MSB Seconds	0x4	Value of the 16 most significant bits of the seconds counter.
LSB Seconds	0x5	Value of the 32 least significant bits of the seconds counter.
Nanoseconds	0x6	Value of the nanoseconds counter.

Table 3.1: Adjustable Clock AXI registers

Since the AXI4 transactions are on 32 bits data width, it is not possible to read the time in one transaction (48 bits for the seconds counter and 32 bits for the nanoseconds). Therefore, the registers 3, 4, 5 and 6 have been set up. Thus, to read the current time, Sample Time should be accessed, that samples the RTC and updates the 80 bits of time split among registers 4, 5 and 6. Finally these last 3 registers can be read to get the sampled value of the time.

Clock domain crossing

When dealing with multiple clocks, clock domain crossing problems can appear. Such kind of issues happens when data is transferred from a register fed by a clock A to a register fed a different clock B. Data can be not acquired by register of clock B if for instance the frequency of B is lower than frequency of A. Data can even be corrupted if B acquire data from A during a change of value since all bits of the value can, physically, not change at the same time. We observed that kind of problems in our adjustable clock but also in Appendix C. There exist multiple ways to solve or to avoid as much as possible that complication, we give here the 3 most common :

- **Two-stages shift-register** : it consists of 2 consecutive registers in the domain of B. It is designed to reduce the complications but never annihilates them.
- **Grey encoding** : if the transferred data is monotonically increasing, it is possible to convert the data into a domain where each incrementation changes only one bit of the data. Hence, corrupted acquisition can not occur. But data can still be missed or repeated if the difference in frequency between the 2 clocks is too large. That solution was used for solving problems in Appendix C since we were dealing with monotonic incremental counters fed by clock with almost the same frequency.

- **Asynchronous FIFO buffer** : Data is written in a FIFO buffer in the domain of clock A and is read from it in the domain of clock B. A counter is keeping track of the number of elements present in the FIFO buffer. As the counter is grey encoded it can be shared between the two clock domains. It provides perfect protection against data corruption, repetition and loss during data crossing clock domains. That solution was used for our adjustable clock since it ensures correct functionality and is trivial to implement since Xilinx is providing well-made Wizards for generating FIFO buffers.

Real-Time Counter

The goal of that component is to keep track of the time (in terms of seconds and nanoseconds) elapsed since a certain point in the time. It obviously takes the form of a nanoseconds counter and a seconds counter. When reaching 10^9 the nanoseconds counter should roll over. Overflow is detected by subtracting 10^9 to the value that the register will take at the next HDMI clock cycle. If overflow is detected, the seconds counter increments and the nanoseconds register takes the value obtained from the subtraction. Some multiplexers and logic gates are deployed in order to enable offset processes. Since the clock to be synchronized is the HDMI clock, counter registers are fed by this last. The circuitry for all these mechanisms is shown at the Figure 3.6.

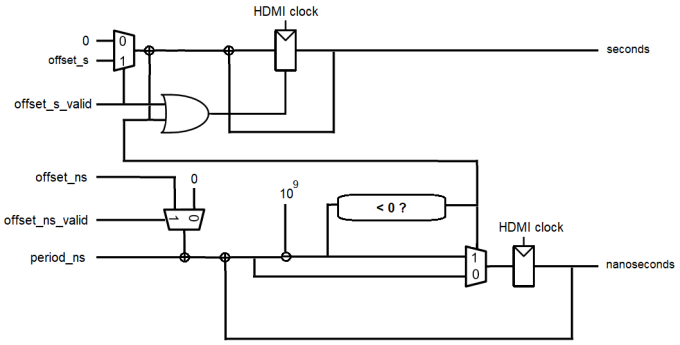


Figure 3.6: RTC circuit architecture

PICXO

As determined in the Section 2.3, VCXO modules are the best solutions to provide controllable oscillators. The problem with that kind of solutions is that they are expensive in fabrication cost, power consumption and in space [22].

The Phase Interpolator Controlled crystal or Xtal Oscillator (PICXO) is a solution developed by Xilinx with the intend to replace VCXO components by using unique feature of transceivers composing some of their FPGA. Transceivers have one Phase Interpolator giving the ability to, fully digitally, phase and frequency modulate the transmit clock operating the transceiver [22]. PICXO connects the transceiver (as shown at Figure 3.7) and exploits that functionality to provide seamless controllable oscillator.

The PICXO alternative has undeniable advantages : it is almost inexpensive since transceivers are printed silicon and the HDMI clock, our clock of interest, can directly be tuned in the HDMI transceiver.

The main interfaces that the PICXO is offering is the OFFSET_PPM and its associated enable, OFFSET_EN. If the enable signal is set to 1, OFFSET_PPM is a direct command to control the HDMI clock frequency offset. The relation between the two is given by Equations 3.1 and 3.2.

$$\text{Offset (Hz)} = \text{OFFSET\_PPM} * \frac{\text{ACC\_STEP} * \text{CEpi}}{64 * \text{TXOUT\_DIV} * 2^{21}} \tag{3.1}$$

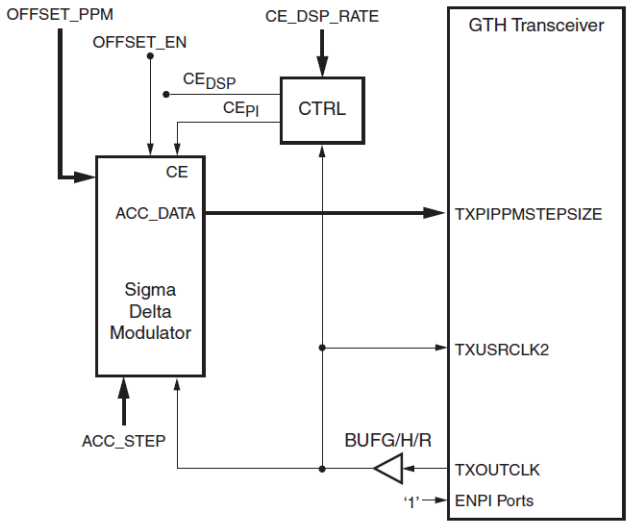


Figure 3.7: PICXO and Transceiver Connection [22]

$$\text{Offset (ppm)} = \text{Offset (Hz)} * \frac{10^6}{\text{bitrate}} \tag{3.2}$$

With :

OFFSET_PPM = 22-bits signed integer command
ACC_STEP = 4-bits unsigned integer selected by user
CEpi = half the frequency of the HDMI clock
TXOUT_DIV = setting for operating rate range for the transceiver
birate = final serial interface line rate and 32 times the frequency of the HDMI clock

It can be observed from Equation 3.1 that ACC_STEP is controlling a trade-off between the resolution and the range of the offset control. If ACC_STEP is low we have high-resolution, if ACC_STEP is high we have a large range of control. Let us illustrate that observation by taking a practical example.

The resolution of the control is the difference in control of two consecutive commands. The range of the control is the absolute difference between extreme control values. By choosing the frequency of the HDMI clock to be equal to 125 MHz and TXOUT_DIV = 1, we have CEpi = 62.5MHz and bitrate = 4Gbps. By using Equation 3.1, for ACC_STEP equal to 1 we have :

$$\text{Resolution(Hz)} = \frac{62.5 * 10^6}{64 * 2^{21}} = 0.47 \text{ Hz}$$

$$\text{Range(Hz)} = |-2^{21} * \frac{62.5 * 10^6}{64 * 2^{21}} - 2^{21} * \frac{62.5 * 10^6}{64 * 2^{21}}| = 1953125 \text{ Hz}$$

By using Equation 3.2, we find :

$$\text{Resolution(ppm)} = 0.0001164 \text{ ppm}$$

$$\text{Range(ppm)} = 488.28125 \text{ ppm}$$

By following the same reasoning, we can find the resolution and range for ACC_STEP equal to 15 (the maximum value of a 4-bits unsigned integer), that we compare to the values found for ACC_STEP in Table 3.2. Such adjustment capabilities of the PICXO open opportunities to dynamically define ACC_STEP to accelerate convergence towards a target time by using a larger range and thus larger offset. And then, refine the search once the error is closer to zero by

ACC_STEP	Resolution(ppm)	Range(ppm)
0x1	0.0001164	488.28125
0xF	0.0017462	7324.21875

Table 3.2: Trade-off between resolution and range of the PICXO in function of ACC_STEP parameter

using a better resolution. We did not explore such mechanism since speed convergence of the system was not a priority concern for intoPIX.

In conclusion, we have shown that the PICXO is a perfectly suitable solution covering all the needs of the adjustable Clock. Moreover, it is highly configurable in resolution and range of control.

3.3 Software

In this section, the software running on the Softcore is presented.

3.3.1 Operating System

Once the MicroBlaze system is in place, we can consider to execute lines of codes. That code can be executed either as bare-metal or in an Operating System. To choose the appropriate solution, it is important to highlight some of the required needs of the final application :

1. The code should fit in a system with a relatively restrained memory.
2. The application should be able to switch between the TICO software and the PTP application.
3. The application should be responsive to some external events like packets received on the Ethernet interface.

Bare-metal alternative has the advantage that no overhead in term of memory is added to the application respecting the first constraint defined previously. However, the application can not switch between applications since no context saving routines are implemented. And can not react to external events since no interrupt handling method are provided. Obviously we need something more suitable for our application even if we will undeniably pay it in term of memory space.

Operating systems are offering the ability to switch between tasks by implementing context switch routines. They also offer the ability to react to some external events. But memory sizes necessary to run classical Operating Systems like Linux or Windows are too large. It is mainly because some features implemented by such OS like file-systems and User Interface support are not needed in our application.

There exist alternatives to classical Operating Systems, for instance, Real-Time Operating Systems. They provides ability to switch between tasks and handle external events in a responsive and deterministic manner. Most Real-Time Operating Systems are using a low amount of memory.

A lot of RTOS are on the market, but Xilinx is providing a Plug-and-Play way to generate a FreeRTOS project running on a MicroBlaze. As it sounds, FreeRTOS is a free RTOS under the GPL license (any change to the kernel should remain public). All that elements pushed us to use FreeRTOS for the project.

A side effect of that choice is that we can now design the application such that a list of real-time constraints is respected, that is the main feature of RTOS. We do not have any hard or soft constraints in our application. However, priority of some operations has to be respected in order to provide the best achievable performances :

- Since we are doing software timestamping, that operation should be the task with highest priority in order to reduce variable delays between reception and timestamping. That design consideration is discussed in the Section 3.3.2.
- After that the error has been measured and the synchronization and syntonization commands computed, commands should be sent as fast as possible to the adjustable clock. Non-respect of that requirement is studied in the Section 3.3.4.

3.3.2 IP/TCP Stack

We have chosen to implement the UDP/IP PTP since it is the more general one. Therefore, we need an UDP/IP stack. We do not have to reinvent the wheel, an open-source TCP/IP (and UDP) stack exists and is widely used in embedded systems : lwIP (lightweight IP). The deployment and configuration are easy and documented, we faced no problem during these phases.

A design problem occurs when we take into account that the priority of the timestamping of the received packet should be the highest. Since packet arrival is dealt by lwIP, timestamping can not be done until lwIP as processed the obtained packet. To solve that problem, there is 3 solutions :

- Modify the lwIP library in such a way that timestamping is the first operation of the receiving functions.
- Override the handler of the signal warning packets arrival. The new handler would timestamp and then call the former handler (lwIP).

These solutions are tricky, we decided to not consider them and to still timestamp after the lwIP operations. Our choice was motivated by the fact that INTOPIX wants, after the current project, to investigate hardware timestamping capabilities. Hence, optimizing the software timestamping is a loss of time since it will disappear.

3.3.3 PTP Stack

This is the core part of the project. The PTP stack has the role to exchange with other PTP instances in the network, measure the drift and offset in time, compute a command and send it to an adjustable clock. No open-source PTP stack for FreeRTOS and lwIP exists but multiple projects exists for other platforms like Embedded Linux or even bare-metal. Hence, instead of implementing the PTP from scratch, we decided to port an existing implementation to our environment. Mainly two projects were appealing candidates :

- **PTPd** : 7-years old open-source Linux daemon under a BSD license (almost in public domain).
- **PPSi** (PTP Ported to Silicon) : 5-years old CERN PTP stack based on PTPd under a GNU LGPL (changes to the program do not have to be public). It aims to facilitate porting to embedded systems.

The first thing that we can observe from that is : thanks to licensing, if INTOPIX wants to keep secret the modified version of one of the two previous projects they can. Although more recent, not finished and badly documented, we have chosen to port the PPSi project. PTPd is a nice project but sadly, as a Linux daemon, is too much linked to Linux libraries and Linux System Calls.

A great part of the porting task has already been made by the CERN. They have isolated, in an interface file, all the System Calls. These Calls are communication and commands with and to external components like timer, Ethernet interface, adjustable clock,... So, all we have to do

in order to completely port the project in our environment, is to implement these function by using our hardware components. The signatures of the functions are listed in Listing 3.1.

Listing 3.1: PPSi System Calls signatures

```
/* syscall.h */

/* Stop call */
extern void exit(int exitval);

/* Network stack calls */
extern int select(struct sel_arg_struct *as);
extern int write(int fd, const void *buf, int count);
extern int ioctl(int fd, int cmd, void *arg);
extern int socket(int domain, int type, int proto);
extern int bind(int fd, const struct bare_sockaddr *addr, int addrlen);
extern int recv(int fd, void *pkt, int plen, int flags);
extern int send(int fd, void *pkt, int plen, int flags);
extern int setsockopt(int fd, int level, int name, const void *val,int len);
extern int shutdown(int fd, int flags);
extern int close(int fd);

/* Time calls */
extern int gettimeofday(void *tv);
extern int settimeofday(void *tv);
extern int adjtimex(void *tv);

extern int clock_gettime();
```

The problem is that they are not specified at all. Since there are similitude with the native Linux System Calls, we assumed that the specifications are the same.

We have implemented the Stop call by using the exit function provided by FreeRTOS. Network stack calls can directly be routed to our IP/TCP Stack interface by using lwIP functions. Finally, Time calls are implemented by using the AXI4 interface of the adjustable clock. The *sys_settimeofday* function is used to change abruptly the clock time in order to rapidly recover from large error. The *sys_adjtimex* function is called to either abruptly offset the value of nanoseconds or apply a frequency offset. As described in the Appendix B, determination by the PTP stack if an abrupt change or a frequency offset is required, is determined by a dedicated parameter. The *clock_gettime* function requires that the read clock is monotonically increasing. Hence, our adjustable clock can not be used. Instead, we used the FreeRTOS internal clock tick counter. Everything except trivial routing of Network stack calls, is shown at Listing 3.2.

Listing 3.2: PPSi System Calls implementation

```
/* syscall.c */

/* Stop call */
void exit(int exitval){
   vTaskDelete(NULL);
}

/* Time calls */
int sys_gettimeofday(struct bare_timeval *tv){
   /* Sample the time */
   RTC_mWriteReg(XPAR_RTC_0_S00_AXI_BASEADDR,RTC_S00_AXI_SLV_REG3_OFFSET,1);
   /* Read the time */
   int seconds_MSB = RTC_mReadReg(XPAR_RTC_0_S00_AXI_BASEADDR,
       RTC_S00_AXI_SLV_REG4_OFFSET);
   int seconds_LSB = RTC_mReadReg(XPAR_RTC_0_S00_AXI_BASEADDR,
       RTC_S00_AXI_SLV_REG5_OFFSET);
   int nanoseconds = RTC_mReadReg(XPAR_RTC_0_S00_AXI_BASEADDR,
       RTC_S00_AXI_SLV_REG6_OFFSET);
```

```
16    /* Merge MSB and LSB */
17    uint64_t seconds = (((uint64_t)seconds_MSB) << 32 | ((uint64_t)seconds_LSB));
18    tv->tv_sec = seconds;
19    tv->tv_nsec = nanoseconds;
20    return 1;
21 }
22 int sys_settimeofday(struct bare_timeval *tv){
23    /* Read actual time */
24    struct bare_timeval current;
25    sys_gettimeofday(&current);
26    /* Compute offset to apply */
27    int os = tv->tv_sec - current.tv_sec;
28    int ons = tv->tv_nsec - current.tv_nsec;
29    /* Write offset */
30    RTC_mWriteReg(XPAR_RTC_0_S00_AXI_BASEADDR,RTC_S00_AXI_SLV_REG0_OFFSET,os);
31    RTC_mWriteReg(XPAR_RTC_0_S00_AXI_BASEADDR,RTC_S00_AXI_SLV_REG1_OFFSET,ons);
32    return 1;
33 }
34
35 #define inv_G_tot 8589.9346 /* Inv Gain of PICXO, see PICXO Excel file */
36 #define freq_ppm_max 240 /* Range(ppm)/2 */
37 #define freq_MHz 125 /* Frequency(MHz) of the HDMI clock */
38
39 int sys_adjtimex(struct bare_timex *tv){
40    /* Get commands */
41    int offset_ns = tv->offset;
42    double freq_ppm = ((double)tv->freq)/freq_MHz;
43    /* Anti wind-up */
44    if(freq_ppm > freq_ppm_max)
45       freq_ppm = freq_ppm_max;
46    else if(freq_ppm < -freq_ppm_max)
47       freq_ppm = -freq_ppm_max;
48    /* Send commands if required */
49    if(offset_ns != 0){/* Offset the nanoseconds */
50       RTC_mWriteReg(XPAR_RTC_0_S00_AXI_BASEADDR,RTC_S00_AXI_SLV_REG1_OFFSET,
          offset_ns);
51    }
52    if(tv->freq != 0){ /* Offset the ppm */
53       RTC_mWriteReg(XPAR_RTC_0_S00_AXI_BASEADDR,RTC_S00_AXI_SLV_REG2_OFFSET,
          inv_G_tot*freq_ppm);
54    }
55    return 1;
56 }
57
58
59 int clock_gettime(){
60    return xTaskGetTickCount()*portTICK_PERIOD_MS;
61 }
```

3.3.4 Control Loop

PPSi contains an implementation of a PI controller. They used a Proportional gain P of 0.1 and an Integral gain I of 0.001. But they never justified the choice of such values, we do not even know if the system is stable under any circumstance. Thus, the goal of this section is to model the controller and give a region for the pair P and I where stability is guaranteed under the worst circumstances.

Model

The model of the control loop of a slave clock is given at Figure 3.8. The system is a composed of a PI controller C, a sample and hold S and a plant P representing the adjustable clock. The

signal $m(t)$ is the time of the PTP Master, $s(t)$ is the time of the slave and thus the error signal, $o(t)$, is representing the offset from Master value computed by the PTP. The sample and hold block is there to represent the Synchronization Interval T of the PTP, the system is therefore a discrete system. Some time, represented by $d(t)$, is needed to make measures, compute and send a commands to the controller. We decided to define $d(t)$ as a fractional part (between]0,1[) of the Synchronization interval T.

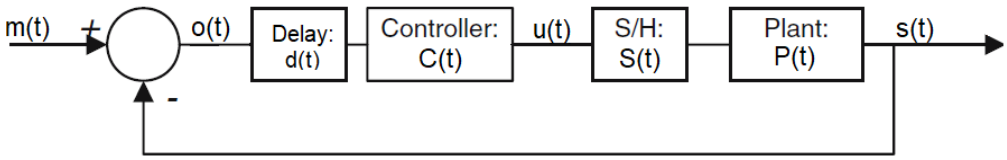


Figure 3.8: Model of the control loop of a slave clock

Stability inspection

The Appendix A computes a valid region for the pair P and I to ensure stability. But in Appendix A, strong assumption is made on $d(t)$, the computational delay of the system. Originally in the range]0,1[, $d(t)$, the time between the offset measure and the actual control sent to the clock servo, has been approximated to 0. That is because most of PTP implementations tend to minimize that value. But minimization and stability can not be guaranteed due to non-determinism in the PTP process (packets arrival time, scheduling,...). Hence, we can observe that $d(t)$ can move up to 1 but also be different at each synchronization interval (that is why d is a function of t). These variations can invalidate results obtained at Appendix A and generate instability in the PI controller. A more thorough study of the impact of $d(t)$ on the system stability is therefore needed.

We are led to ask ourselves if it is possible to tune the controller in such a way that stability is ensured for every value of $d(t)$. Due to the complexity of the computational study of the system, it is almost impossible to express the Proportional and Integral gains (P and I) in function of $d(t)$. It would need to compute the poles of the controller transfer function and derive conditions in function of P, I and $d(t)$ such that the poles are within the unit circle.

By using the major result A.11 of Appendix A, the transfer function of the controller is equal to :

$$T_{CL}(z) = \frac{z^2(1-d(t))(P+I) + z\{d(t)(P+I) + d(t)P\} - d(t)P}{z^3 + z^2\{-2 + (1-d(t))(P+I)\} + z(1 + 2d(t)P + d(t)I - P) - d(t)P} \tag{3.3}$$

Finding the expression of the roots of the denominator in all cases is a real challenge. Hopefully, there is an other way to ensure the stability of the system without even computing the roots of the denominator. By using the Jury stability criterion [18], we can ensure the stability if a set of constraints is respected. For a system with the denominator of the transfer function of order 3 :

$$D(z) = b_3 z^3 + b_2 z^2 + b_1 z + b_0 \tag{3.4}$$

We ensure stability if :

$$\begin{cases} b_3 + b_2 + b_1 + b_0 & > & 0 & (1) \\ -b_3 + b_2 - b_1 + b_0 & < & 0 & (2) \\ b_0 - b_3 & < & 0 & (3) \\ b_0 b_2 - b_1 b_3 - b_0^2 + b_3^2 & > & 0 & (4) \end{cases}$$

By identification between Equations 3.3 and 3.4, the system becomes :

$$\begin{cases} 1-2+(1-d(t))(P+I)+1+2d(t)P+d(t)I-P-d(t)P & > & 0 & (1) \\ -1-2+(1-d(t))(P+I)-1-2d(t)P-d(t)I+P-d(t)P & < & 0 & (2) \\ -d(t)P-1 & < & 0 & (3) \\ -d(t)p(-2+(1-d(t))(P+I))-1-2d(t)P-d(t)I+P)-(-d(t)P)^2+1 & > & 0 & (4) \end{cases}$$

A bit of Algebra gymnastic lead us to :

$$\begin{cases} I & > & 0 & (1) \\ (1-2d(t))(2P+1) & < & 4 & (2) \\ -d(t)P & < & 1 & (3) \\ (d(t))^2PI-d(t)(P^2+PI+I)+P & > & 0 & (4) \end{cases}$$

We want to find conditions on P ($\neq 0$) and I ($\neq 0$) such that this system is satisfied for all $d(t)$ in the range]0,1[. Let us consider 2 cases : $P < 0$ and $P > 0$.

- $\underline{P < 0}$:

  By using Equation (3), we find that P $\in$]-1,0[.

  In this case Equation (4) is never satisfied :

  * $(d(t))^2$ is positive and PI is negative :

    $$(d(t))^2PI \leq 0$$

  * $-d(t)$ is negative, $P^2 \in$ [0,1] and $PI + I = (P+1)I > 0$:

    $$-d(t)(P^2+PI+I) \geq 0$$

  *

    $$P < 0$$

  The sum of 3 negative terms can not be strictly positive. Therefore Equation (4) is never satisfied and the case $P < 0$ does not lead to relevant values for P and I.

- $\underline{P > 0}$:

  In this case, Equations (2) and (3) can be ignored since there are always satisfied.

  We observe that Equation (4) is in fact a quadratic equation in function of m. We want the equation to be satisfied for every $d(t)$ in the interval $[0, 1]$. Since $PI > 0$, there are 3 possible simple and favorable cases :

  (a) Equation(4) has no roots.

  (b) Root(s) of Equation(4) are in the negative and non-zero domain.

  (c) Root(s) of Equations(4) are strictly greater than 1.

  Mathematically, we have :

  $$\begin{aligned} \Delta &= (-(P^2+PI+I))^2-4P^2I \\ &= P^4+P^2I^2+I^2+2P^3I+2PI^2-2P^2I \end{aligned} \tag{3.5}$$

  (a)

  $$\Delta < 0$$

(b)

$$\frac{P^2 + PI + I \pm \sqrt{\Delta}}{2PI} < 0$$

(c)

$$\frac{P^2 + PI + I \pm \sqrt{\Delta}}{2PI} > 1$$

Let us verify, in each case, if relevant values for I and P can be extracted :

(a)

$$P^4 + P^2I^2 + I^2 + 2P^3I + 2PI^2 - 2P^2I < 0$$

By rewriting under the form of a quadratic equation in function of I :

$$(P^2 + 2P + 1)I^2 + (2P^3 - 2P^2)I + P^4 < 0$$

The discriminant is :

$$\Delta' = (2P^3 - 2P^2)^2 - 4P^4(P^2 + 2P + 1) = -16P^5$$

Since the discriminant is negative and the coefficient of I^2 is positive, Δ is never negative.

(b)

$$\frac{P^2 + PI + I \pm \sqrt{\Delta}}{2PI} < 0$$

Since $\frac{P^2+PI+I+\sqrt{\Delta}}{2PI}$ is always positive, this condition is never satisfied.

(c)

$$\frac{P^2 + PI + I \pm \sqrt{\Delta}}{2PI} > 1$$

Since $\frac{P^2+PI+I+\sqrt{\Delta}}{2PI} \geq \frac{P^2+PI+I-\sqrt{\Delta}}{2PI}$, we only need to verify that :

$$\frac{P^2 + PI + I - \sqrt{\Delta}}{2PI} > 1$$

After simplification, we have :

$$I < P - P^2$$

This result is relevant and reduces the range of P and I constants : since $I > 0$, we have $P < 1$. And since $P < 1$, we have $I < 1$.

As a sanity check, the root is represented for each values of P and I in the grid [0,1]x[0,1] at the Figure 3.9.

The red plan separates valid values to non-relevant ones. We can also highlight that the red plan is, in fact, representing the value of $d(t)$. We observe that if the value of $d(t)$ is reduced, the number of possible pairs P and I increases. That could be intuitively expected since we reduce the constraints, the number of solutions can only increase.

In conclusion, we have found a set of conditions on I and P to ensure the stability of the PI controller for any value of $d(t)$:

$$\begin{cases} 0 < I < 1 & (1) \\ 0 < P < 1 & (2) \\ I < P - P^2 & (3) \end{cases}$$

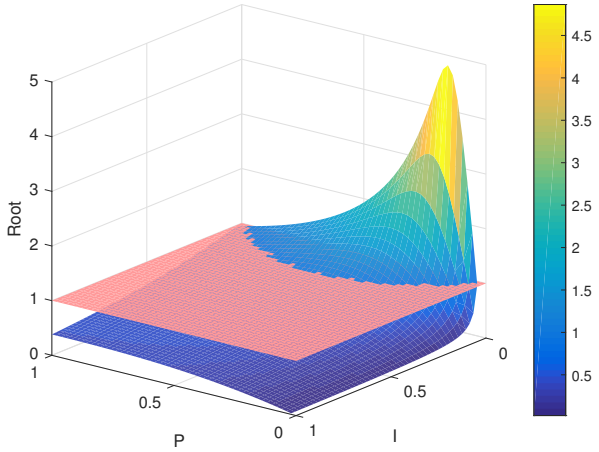


Figure 3.9: Root of one of the Jury stability criterion in function of P and I

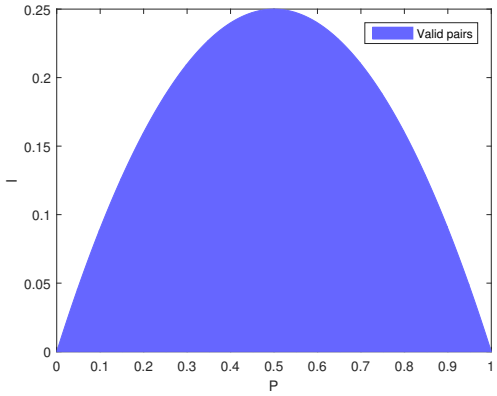


Figure 3.10: Region of valid P and I values ensuring stability of the PI controller

The region of valid values for I and P is represented at the Figure 3.10.

That is a lot of computation, we therefore decided to make a quick sanity check to ensure that our final conditions have sense. If we fix $P = 0.1$, we should fix $I < 0.1 - 0.1^2 = 0.09$.

If we compute the magnitude of the poles of the Equation 3.3 for $d(t) = 1$, $P = 0.1$ and $I = 0.091$ (slightly larger than the stability boundary) we obtain 1.0006, 1.0006 and 0.0999. Two poles are not in the unit circle, therefore the system is not stable as predicted.

If we compute the magnitude of the poles of the Equation 3.3 for $d(t) = 1$, $P = 0.1$ and $I = 0.089$ (slightly lower than the stability boundary) we obtain 0.9994, 0.9994 and 0.1001. All the poles are within the unit circle, therefore the system is stable as predicted.

Final selection of P and I

The fruit of chance, values defined in the PPSi are within the valid region ensuring stability of the system that we have determined at Figure 3.10. Stability is not the only concern when talking about controllers. In fact, the convergence speed is also a common concern. But in our context, it is not a priority and we assume that the CERN has fixed the default values of P and I in function of running performances. So, finally, we are using $I = 0.001$ and $P = 0.1$ as controller parameters ensuring at least stability.

3.4 Summary

Final obtained architecture is shown at Figure 3.11.

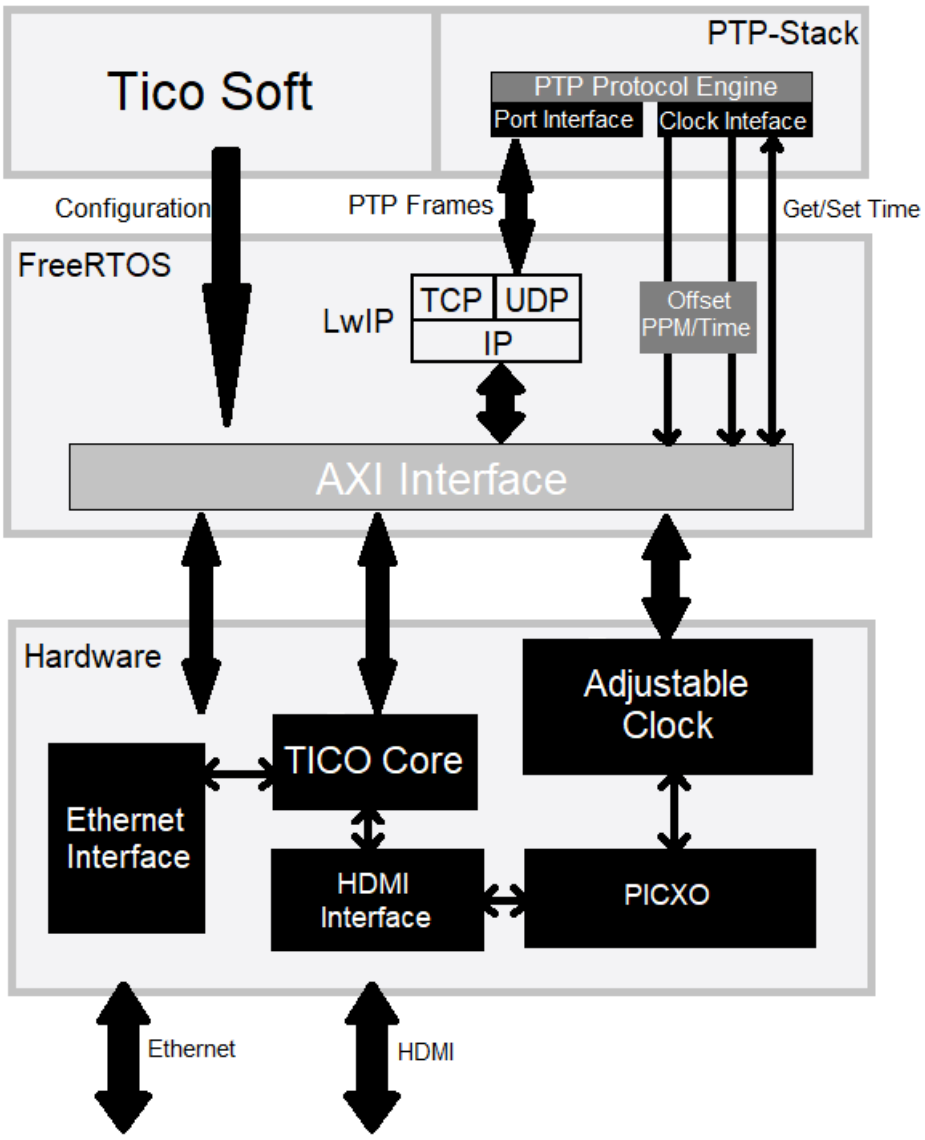


Figure 3.11: System Architecture Overview

A hardware MicroBlaze system has been deployed in order to enable the ability to run software. The MicroBlaze is connected to the TICO-core, Ethernet interface and an adjustable clock via the AXI4 protocol. Adjustable clock is implements an AXI4 interface, a Real-Time Counter a mechanism to allow data to cross clock domains. It is connected to a PICXO instance adjusting the frequency of the HDMI clock by dynamically taking advantage of frequency options in the HDMI transceiver.

The software is running on FreeRTOS enhanced with the lwIP IP/TCP stack. The application is composed of a PTP stack and the TICO Soft. The PTP stack is in fact the PPSi CERN project ported to run on our OS and with our IP/TCP stack. The stack has been connected to our hardware adjustable clock via the implementation of a list of System Calls. The stability of the software control loop has been studied to safely fix the Proportional and Integral constants of the PI controller under hard constraints including large computational delays.

Chapter 4

Results

Abstract

In this chapter, obtained results are described and discussed. Measurements conditions are given to allow a better reproducibility of the experiments. Functional performances serve as a sanity check of the porting process of the PPSi. CPU utilization by the PTP stack is studied to verify the ability of the Microblaze to run the TICO soft aside. Finally, synchronization performances are explored to ensure that the system achieve the primary requirement : timing synchronization.

4.1 Measurement Conditions

As illustrated at the Figure 4.1, the system under test is composed of two KCU105 FPGA's connected in a point-to-point manner via Ethernet. The FPGA are both programmed with the System described in the Implementation chapter with PTP Edge to Edge mode. The internal Si570 oscillator is used as a generator of the HDMI clock in the HDMI transceiver. That clock has a Frequency Tolerance of ± 50 ppm. One FPGA is configured as the Master only and with 192.168.1.100 as IP address. The remaining is Slave only and with 192.168.1.10 as IP address. FPGA's stand in the same room heated at 25°C and are connected using a short Ethernet cable (0.8 m). A surrounding PC is connected via 2 USB Jtag cables for each FPGA: one for downloading the bitstream representing the System, the other to get data from the MicroBlaze (i.e. debug messages).

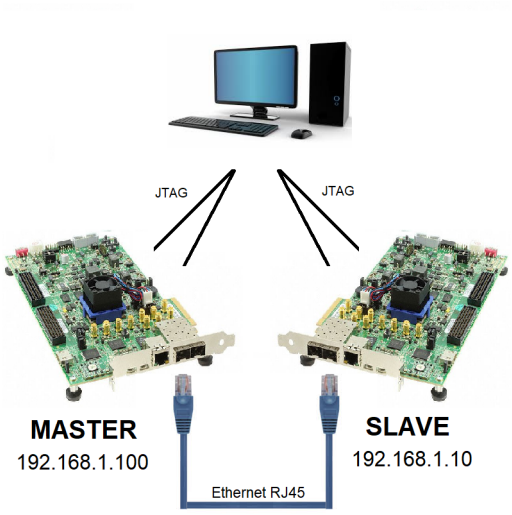


Figure 4.1: System under test representation

For measurements requiring Wireshark capabilities, a PC with Debian is connected in the same manner to one FPGA. The PC is running the unmodified version of the PPSi compiled for Linux kernel or the PTPd daemon since both are interoperable with the implemented System.

If not specified, the parameters of the System are set to the values described in the Appendix B.

It is important to stress the fact that timing synchronization performances have been measured using the value computed by the PTP stack, the observed Offset from Master. We have already discussed the multiple sources of noise affecting that measure. Using it as a performance metric is therefore not ideal but is clearly sufficient for the targeted range of precision required by intoPIX. For information purpose only, we provide, here, an example of better measurement system. FPGA's are generating a PPS (pule per second) signal each time the counter fed by the PTP clock increments the seconds. These signals are then carried (using a link with identical delay) to an analog device allowing to compare them. For instance, an oscilloscope can be used to compare the signals by looking at their phase and frequency differences. The noise introduced by such a measurement system is almost null : carriage to an external port of the FPGA board, transport by the link and data acquisition by the oscilloscope are the only possible sources of variable delays.

4.2 Functional performances

To assess the basic functionality of the implemented System, we have used Wireshark to analyze the message exchanges during the synchronization process. A Wireshark dump of a typical message exchange can be found at the Figure 4.2.

No.	Time	Source	Destination	DstPort	Protocol	Length	Info
8	1.578573	192.168.1.100	192.168.1.10	319	PTPv2	86	Sync Message
9	1.579277	192.168.1.100	192.168.1.10	320	PTPv2	86	Follow_Up Message
11	2.429004	192.168.1.100	192.168.1.10	319	PTPv2	86	Sync Message
12	2.429729	192.168.1.100	192.168.1.10	320	PTPv2	86	Follow_Up Message
13	2.581075	192.168.1.10	192.168.1.100	319	PTPv2	86	Delay_Req Message
14	2.581777	192.168.1.100	192.168.1.10	320	PTPv2	96	Delay_Resp Message
15	3.069575	192.168.1.100	192.168.1.10	320	PTPv2	106	Announce Message
16	3.249012	192.168.1.100	192.168.1.10	319	PTPv2	86	Sync Message
17	3.249630	192.168.1.100	192.168.1.10	320	PTPv2	86	Follow_Up Message
18	3.751079	192.168.1.10	192.168.1.100	319	PTPv2	86	Delay_Req Message
19	3.751744	192.168.1.100	192.168.1.10	320	PTPv2	96	Delay_Resp Message

Figure 4.2: Wireshark dump of the synchronization process

Here is what we can mainly observe from that dump :

- The message exchange process as defined in our State-of-the-art chapter is well respected.
- The Sync Interval (the time between **Sync** messages) is conform as what we have defined in our parameters.
- The destination port for Event and General messages respect the PTP standard.
- The messages are recognized as part of the PTP protocol by Wireshark giving a satisfactory verification that messages are well-formatted.

To explore more in depth the Functional performances, we have decided to also study the details of a packet. The details composing a **Sync** message is shown at Figure 4.3.

From that Figure, we can essentially observe three things :

- All PTP fields composing a **Sync** are present and set.
- The correction field is set to 0. That could be excepted since it is reserved for Transparent PTP clocks updating that field by the residence time of the packet in their buffer. Since we

```
> Frame 8: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)
> Ethernet II, Src: Critical_9d:b0:3f (00:90:9e:9d:b0:3f), Dst: Xilinx_00:01:02 (00:0a:35:00:01:02)
> Internet Protocol Version 4, Src: 192.168.1.100, Dst: 192.168.1.10
> User Datagram Protocol, Src Port: 319, Dst Port: 319
v Precision Time Protocol (IEEE1588)
  > 0000 .... = transportSpecific: 0x0
    .... 0000 = messageId: Sync Message (0x0)
    .... 0010 = versionPTP: 2
    messageLength: 44
    subdomainNumber: 0
  > flags: 0x0200
  > correction: 0.000000 nanoseconds
    ClockIdentity: 0x00909efffe9db03f
    SourcePortID: 1
    sequenceId: 1
    control: Sync Message (0)
    logMessagePeriod: 0
    originTimestamp (seconds): 1528378844
    originTimestamp (nanoseconds): 738925482
```

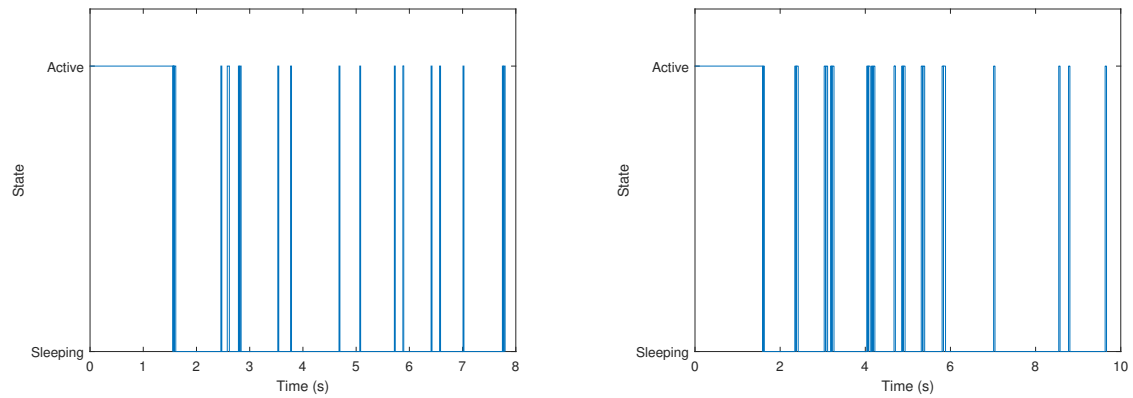
Figure 4.3: Detailed Wireshark dump of a **Sync** packet

have setup a point-to-point direct network, the packets are not going trough any network element except the Sender and the Receiver.

- The packet is well timestamped with a value for both the seconds and the nanoseconds, representing the Epoch time.

4.3 CPU utilization

As a major design choice of the System was to run the PTP stack along with the TICO-Soft on the same MicroBlaze, we wanted to verify that the CPU utilization by the PTP stack is reasonable. A graphical representation of the CPU utilization is given at the Figure 4.4 for both the Master and the Slave. The start-up time along with the inline CPU utilization proportion are given at the Table 4.1.



(a) Master (b) Slave

Figure 4.4: MicroBlaze utilization by the PTP Master and Slave instances

We first observe from these data that the start-up time for the Master and the Slave are almost identical. We can explain that by the fact that they both have to initialize the same data structures mainly required for network connection and data analysis. Secondly, although relatively reasonable, inline CPU utilization of the Slave is two times greater than the Master one. This observation can be justified with the multiple expensive computations (e.g. multiplications

State	Start-up CPU utilization time (s)	Inline CPU utilization proportion (%)
Master	1.6004	5.3856
Slave	1.5962	10.0410

Table 4.1: MicroBlaze utilization by the PTP instance

and divisions) required by the Slave to obtain the error measures, an updated state of the PI controller and the actual command sent to the adjustable clock.

Finally, we can expect that the Master CPU utilization increases linearly with the number of Slaves in the network. But not proportionally since a huge part of the messages sent by the Master can be emitted in broadcast or multicast.

4.4 Synchronization Performances

The main goal of the System is to provide timing synchronization. This section aims at evaluating the synchronization performances as well as discussing the benefits that can be given to the intoPIX demonstration.

The Figure 4.5 gives the evolution of the error of synchronization for multiple executions of the synchronization process with different values of the Sync Interval. To preserve a readable scale we have discarded the abrupt changes of the adjustable clocks occurring at start-up to recover from large Offset. We observe that each experiment is converging towards zero after having suffered from the typical overshoot inherent to PI controllers. Some peaks can be observed on the graph because no filtering technique was actually implemented to smooth or discard noisy measures.

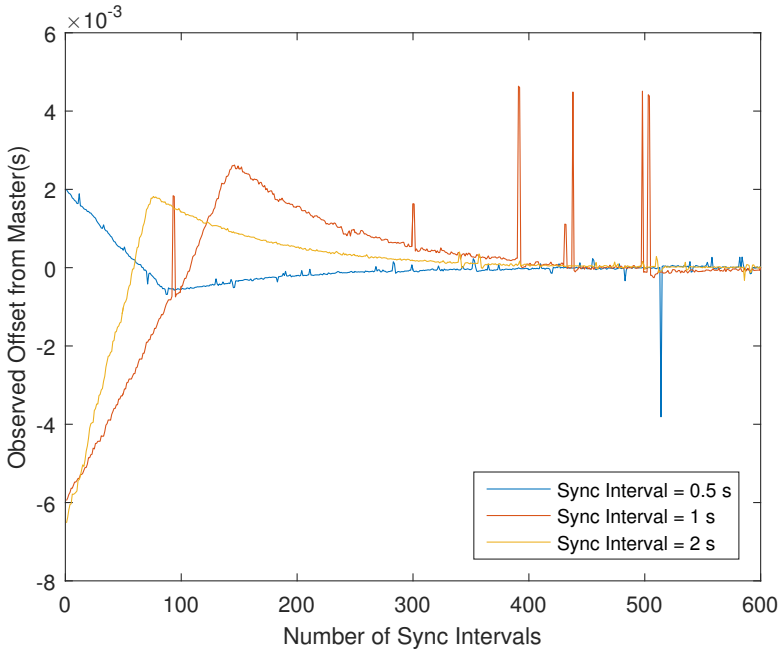


Figure 4.5: Evolution of the observed Offset from Master for different values of Sync Interval

We have decided to represent the error in function of the number of Sync Intervals and not the time to highlight the fact that each experiment achieve the same speed convergence in term of packets exchanged. An obvious trade-off has therefore to be discussed between the duration and intensity of the overhead caused on the network and the required convergence speed. The convergence speed is not a primary concern but some tracks could be followed to increase it :

- The already discussed reduction of the Sync Interval duration.
- Abrupt changes are enabled for error greater than 1 seconds by default, but this value could be reduced.
- The PICXO Resolution could be reduced for the benefit of its Range in order to allow for more powerful commands during the convergence phase. After having converged, the value could be readjusted.
- The Proportional and Integral Gains of the controller could be tuned.

A zoomed view of the Figure 4.5 is given at Figure 4.6. That Figure gives a focus on the phase after the convergence. Along with the Table 4.2 giving a statistical analysis of a larger sample of data after convergence, we can make one essential observation : the Offset from Master is stabilizing under around 100 μs in absolute value. This result is very important and promising, because it ensures that INTOPIX can use in its demonstration a frame buffer of size one without fearing a buffer overflow since the error range is smaller than the duration of a frame ($\pm$ 17 ms since the Video is at a 60 FPS rate). Buffer underflow can be mitigated by adding a small delay (greater than guaranteed synchronization, i.e., 0.1 ms) to the TICO decoder to ensure that there is always a frame available when the current will finish.

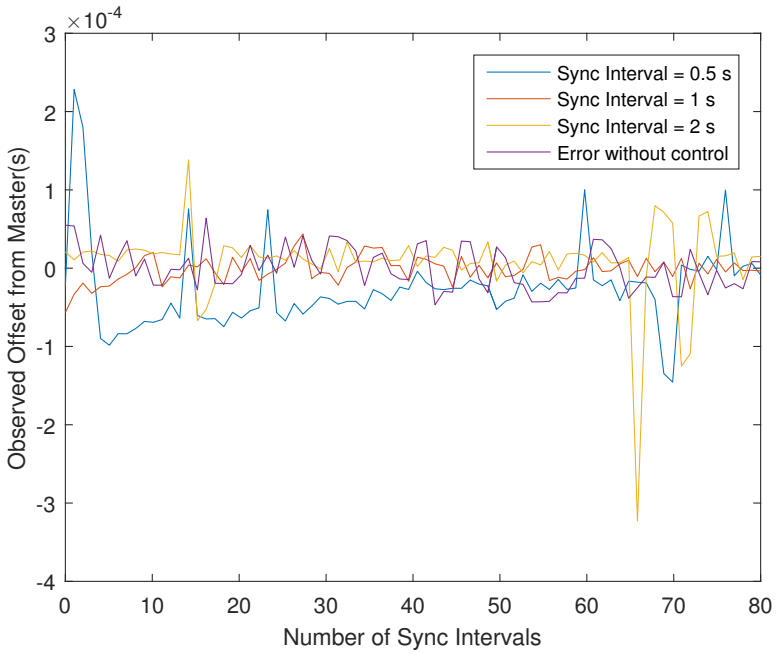


Figure 4.6: Evolution of the observed Offset from Master for different values of Sync Interval after convergence

Sync Interval	Mean (μs)	Standard deviation (μs)
0.5	11.2	56.4
1	-1.5	60.3
2	8.6	49.9

Table 4.2: Statistical analysis of the observed Offset from Master for different values of Sync Interval after convergence

We also have drawn on the Figure 4.6 the measure of the Error if no control is applied. It is not representative of long term evolution since clock could drift way more if for instance they

are placed in rooms with different temperature. But that gives us information on the quality of the provided System : the error converged towards and stays within the range of oscillations of the measures. The source of these measures oscillations can be two-fold : clocks drift and/or measure errors. Both are possible since the clocks have a Frequency Tolerance of $\pm$ 50 ppm and can therefore accumulate 100 μs of error on a Sync Interval of 1 s, or even 200 μ s on a Sync Interval of 2 s. But our State-of-the-art and the fact that more frequent measures does not affect performances push ourselves to designate the quality of the measures as the cause. If this is the case, the System response has reached the measure precision and we can not expect way more from it. Investigation should then be taken to see if measures can be improved and by consequent performances, that includes for instance hardware timestamping and/or Kalman filter. One way to verify that the measures are in fact bounding the precision range, is to use the System with perfect measures. In the Appendix C, we explore that opportunity by implementing an hardware "emulation" of the protocol coupled with the same control system. Results are largely better under the same control and adjustment conditions and proves that improvement of the measure process has to be explored.

Finally, we wanted to inspect the value of the actual command sent to the adjustable clock of the Slave. Figure 4.7 is showing the response of the Slave against an artificial drift supplied to the Master with a Sync Interval of 0.25 s (to accelerate measurement process). If the clock of the Master is artificially offset by 50 ppm, we observe that, after short convergence phase, the Slave also applies an average offset of 50 ppm to its adjustable clock and the error converges towards 0.

This result is not sufficient alone. In fact, it is not warranted that the clocks are running at the same speed without any control. That is why we decided to also represent the evolution of the error if no control is applied but the Master is still artificially biased. We observe that, after 50 seconds (200 Sync Intervals of 0.25 s), the Slave has accumulated 0.0025 s of delay. The Slave has thus accumulated $0.0024515/50 = 49 * 10^{-6} \approx 50 * 10^{-6}$ s of delay per second which corresponds to 50 ppm. That proves that during this period of time, the clocks of the Master and the Slave were running at the same speed except for the artificial bias of 50 ppm supplied to the Master. This last result proves that the system is working perfectly since it is finely tracking the change in frequency of the Master.

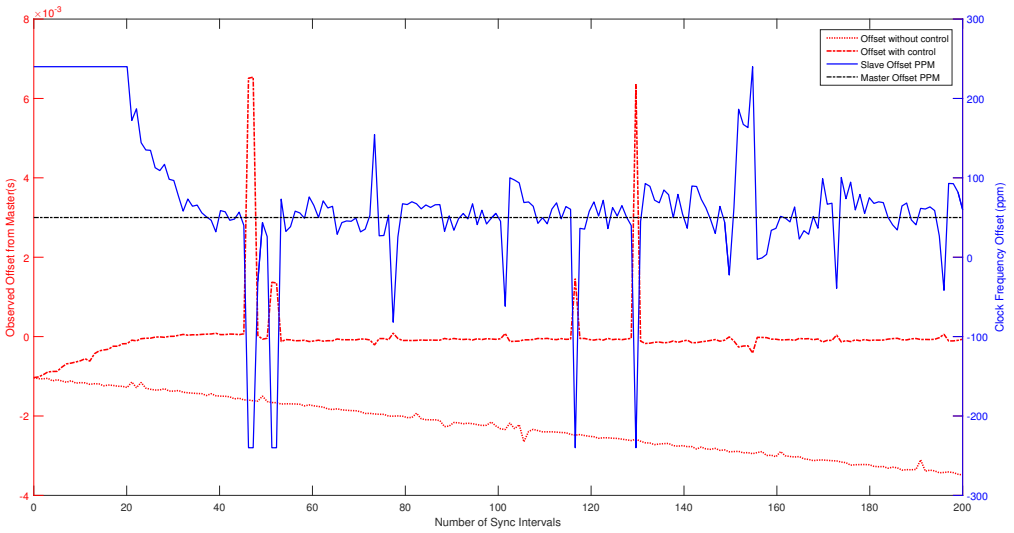


Figure 4.7: Evolution of the observed Offset from Master and Clock Frequency Offset of a Slave connected to a Master clock artificially offset by 50 ppm

4.5 Summary

Functional performances of the protocol have been demonstrated in a direct point-to-point network with one Master and one Slave.

The Master and Slave requires around 1.6 s to start-up. The CPU utilization by the Master is about 5% and by the Slave 10%. This difference is assumed to be due to required expensive computations for the error measurements, PI controller updates and command elaborations.

The System achieves a synchronization within the range around $\pm$ 100 μs in a fixed amount of time. Hints to help the reduction of that time are given. The PI controller stability has been shown. The synchronization performances ensure that intoPIX can use a frame buffer of size one without fearing a buffer overflow. Buffer underflow can be avoided by setting a small delay to the TICO decoder, i. e., 0.1 ms. That delay should be larger than the synchronization range. Finally, frequency tracking of the Master by the Slave has been demonstrated although the need for more precise measurements leveraged for instance by using hardware timestamping and/or a Kalman filter.

Chapter 5

Conclusion

Abstract

This Chapter synthesizes the results obtained for the the entire thesis. Perspectives and a Summary are then given.

5.1 Synthesis of Results

The synthesis of the contributions and main results is given hereafter.

- A study of cutting-edge tools available for timing synchronization has been performed.
- Fully functional MicroBlaze system interfaced to the Ethernet has been deployed, opening opportunities for intoPIX to exploit network capabilities in their demonstrations like web servers,...
- A hardware clock adjustable in time and frequency interfaced to a MicroBlaze has been developed. The Range and the Resolution of the frequency adjustment is fixable in function of a parameter.
- An adaptation of a PTP stack has been made to make it run on a MicroBlaze running the FreeRTOS Real-Time Operating System. That includes network management modifications and a fully interface realization with the hardware components like the adjustable clock.
- An innovative study of the stability of a control loop under hard constraints has been performed.
- By combining precedent results, a fully functional synchronization system with low CPU utilization, frequency tracking and synchronization range of $\pm$ 100 μs has been deployed.
- By using last result, a sizing of the decoder frame buffer has been proposed. A buffer of size one can be used under the condition of delaying the decoder by at least 100 μs.
- A list of perspectives for the improvement of the implemented system is given.
- A hardware System emulating a PTP instance with perfect measures and under ideal conditions has been developed and assessed. The synchronization range is within $\pm$ 8 ns.

5.2 Perspectives

Throughout the Results Chapter, we have highlighted the fact that the measures computed by the PTP are not enough precise and are noisy. They are actually bounding the performances obtained by the current Implemented System. We can highlight two ways to overcome this issue :

- Implementation of a filtering technique like a Kalman Filter as presented in the State-of-the-art Chapter.
- Deployment of hardware timestamping as described in the State-of-the-art Chapter.

Also, if the performances are increased, a more precise test framework should be investigated like the one described in Section 4.1.

Finally, the integration in intoPIX final product should be performed in order to verify that the constraints defined over the buffer can guarantee a working system without any overflows.

5.3 Summary

A working and stable (under hard constraints) basis of a synchronization System has been realized. Performances within the range of $\pm$ 100 μs allows to define the size and constraints on the decoder frame buffer contained in the TICO demonstration. The frame buffer can be fixed to a size of 1 if the decoder has an artificial delay slightly larger than 0.1 ms. The final obtained system is not ideal and suffers from unfiltered measures and inaccurate timestamping. To hope achieve synchronization performances under the μs range, these issues should be considered.

Appendix A

PI controller stability conditions

This Appendix aims to provide theoretical analysis of the stability of the system presented at the Figure 3.8. We first try to put in equation the system and then analyze the stability in function of the Proportional and Integral gains of the controller.

The development proposed here is largely based on the invaluable work done by John C. Eidson [6]. Some mathematical steps and concepts are also based on the excellent book of Franklin *et al.* [7].

The Equation describing the PI controller is :

$$u(t) = K_P o(t - d(t)T) + \frac{K_I}{T} \int_{-\infty}^{t-d(t)T} o(\tau) d\tau \tag{A.1}$$

By assuming that o(t) is slowly varying, we can make the following linear interpolation helping in the future Z-transform:

$$o(t - d(t)T) = o(t) + d(t) \times [o(t - T) - o(t)] \tag{A.2}$$

We obtain the Z-transform :

$$\sum_{k=-\infty}^{\infty} z^k o(kT - d(t)T) = \sum_{k=-\infty}^{\infty} z^k (1 - d(t) + d(t) z^{-1} o(kT) \tag{A.3}$$

By using that last obtained approximation, we can derive the Z-transform of Equation A.1 :

$$\begin{aligned} u(z) &= \sum_{k=-\infty}^{\infty} z^k u(kT) \\ &= K_P \sum_{k=-\infty}^{\infty} z^k (1 - + d(t) z^{-1}) o(kT) + \frac{K_I}{T} \sum_{-\infty}^{\infty} z^k \int_{-\infty}^{t-d(t)T} o(\tau) d\tau \end{aligned} \tag{A.4}$$

We can work on the integral term :

$$\begin{aligned} \sum_{k=-\infty}^{\infty} z^k \int_{-\infty}^{kT-d(t)T} e(\tau) d\tau &= \frac{1 - z^{-1}}{1 - z^{-1}} \sum_{k=-\infty}^{\infty} z^k \int_{-\infty}^{kT-d(t)T} e(\tau) d\tau \\ &= \frac{1}{z-1} \sum_{k=-\infty}^{\infty} z^{k+1} \{ \int_{-\infty}^{kT-d(t)T} e(\tau) d\tau - \int_{-\infty}^{kT-T-d(t)T} e(\tau) d\tau \} \\ &= \frac{1}{z-1} \sum_{k=-\infty}^{\infty} z^{k+1} \int_{kT-T-d(t)T}^{kT-d(t)T} e(\tau) d\tau \end{aligned} \tag{A.5}$$

We can approximate the integral by using a backward estimation (the integral is approximated, for each step, by the upper limit value):

$$\begin{aligned}\sum_{k=-\infty}^{\infty} z^{k+1}\{\int_{kT-T-d(t)T}^{kT-d(t)T} o(\tau)d\tau\} &= \sum_{k=-\infty}^{\infty} z^{k+1}To(kT-d(t)T) \\ &= \sum_{k=-\infty}^{\infty} z^{k+1}T(1-d(t)+d(t)z^{-1})o(kT)\end{aligned} \tag{A.6}$$

Merging precedent result with Equation A.4 lead to :

$$C(z) = \frac{u(kT)}{o(kT)} = (1-d(t)+d(t)z^{-1})\{K_P + K_I\frac{z}{z-1}\} \tag{A.7}$$

Since the plant outputs a time but receives as input a rate, it is modeled as an integrator with a gain K_c. We find the following Laplace transform for the combined plant and sample and hold blocks :

$$SP(s) = \frac{s(s)}{u(s)} = \frac{1-e^{-sT}}{s}K_c\frac{1}{s} \tag{A.8}$$

The Z-transform is derived :

$$\begin{aligned}SP(s) &= \frac{e^{sT}-1}{e^{sT}}K_c\frac{1}{s^2} \\ SP(z) &= \frac{z-1}{z}K_c\{Z(\mathcal{L}^{-1}\frac{1}{s^2})\} \\ &= \frac{z-1}{z}K_c\frac{Tz}{(z-1)^2} = K_c\frac{T}{z-1}\end{aligned} \tag{A.9}$$

We finally obtain the open-loop transfer function by using previous result and Equation A.7 :

$$H(z) = \frac{s(kT)}{o(kT)} = (1-d(t)+d(t)z^{-1})\{K_P + K_I\frac{z}{z-1}\}K_c\frac{T}{z-1} \tag{A.10}$$

We find the closed-loop transfer function :

$$\begin{aligned}T_{CL}(z) &= \frac{s(kT)}{m(kT)} \\ &= \frac{H(z)}{1+H(z)} \\ &= \frac{z^2(1-d(t))(P+I)+z\{d(t)(P+I)+d(t)P\}-d(t)P}{z^3+z^2\{-2+(1-d(t))(P+I)\}+z(1+2d(t)P+d(t)I-P)-d(t)P}\end{aligned} \tag{A.11}$$

To reduce the complexity, we can assume that $d(t)$ is stable and null over time.

$$\begin{aligned}T_{CL}(z) &= \frac{s(kT)}{m(kT)} \\ &= \frac{H(z)}{1+H(z)} \\ &= \frac{K_cT\{(z-1)K_P+zK_I\}}{(z-1)^2+K_cT\{(z-1)K_P+zK_I}\end{aligned} \tag{A.12}$$

The denominator is the characteristic equation of the system :

$$z^2 + z\{-2+K_cT(K_P+K_I)\} + \{1-K_cK_PT\} = 0 \tag{A.13}$$

By renaming scaled values $P = K_P K_c T$ and $I = K_I K_c T$, we have :

$$z^2 + z(-2 + P + I) + (1 - P) = 0 \tag{A.14}$$

To ensure stability of the system, roots must be in the unit circle. We compute the roots :

$$\begin{aligned} z &= \frac{-(-2 + P + I) \pm \sqrt{(-2 + P + I)^2 - 4(1 - P)}}{2} \\ &= \frac{-(-2 + P + I) \pm \sqrt{(P + I)^2 - 4I}}{2} \end{aligned} \tag{A.15}$$

Depending on the value of the discriminant in the square root, we have multiple cases to consider :

- $(P + I)^2 - 4I < 0$: roots are complex. The square of the magnitude of the roots should lie in the unit circle :

  $$0 \leq |z|^2 = \frac{(-2 + P + I)^2 + (4I - (P + I)^2)}{4} \leq 1$$

  By simplification we find :

  $$0 \leq 1 - P \leq 1$$

  We finally obtain a set of conditions for stability :

  $$(P + I)^2 < 4I$$

  $$0 \leq I \leq 4$$

  $$0 \leq P \leq 1$$

- $(P + I)^2 - 4I = 0$: roots are equal and real. We have :

  $$-1 \leq z = \frac{-(-2 + P + I)}{2} \leq 1$$

  By simplification we have the set of conditions :

  $$(P + I)^2 = 4I$$

  $$0 \leq I \leq 4$$

  $$0 \leq P \leq 1$$

- $(P + I)^2 - 4I > 0$: roots are different and real. We observe that, in that case, the line $2P = 4 - I$ is in fact a boundary between stable and unstable regions. On that line, the roots are :

  $$z = \frac{-(2 - P) \pm P}{2}$$

  We have $z = -1$ for the negative case, which is within the unit circle.

  And we have root in the unit circle if $0 \leq P \leq 2$ for the positive case.

  Thus, in that range, the line defines a boundary between stable and unstable regions. A quick test allow us to check which region is stable and which one is not.

The Figure A.1 is summarizing previous results.

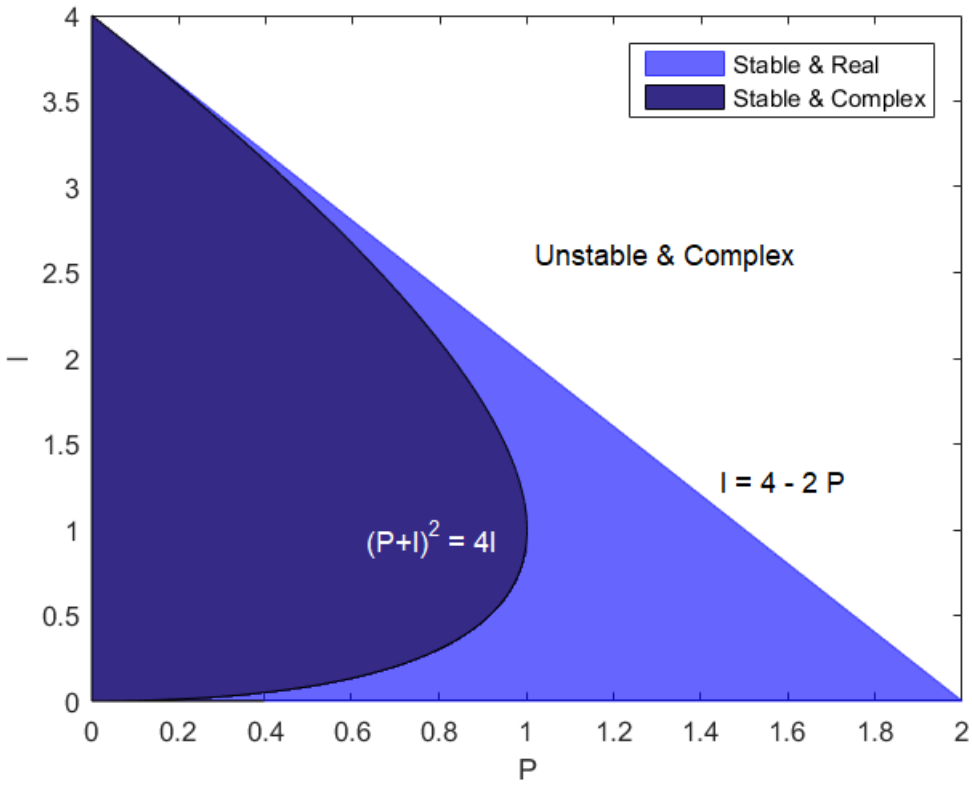


Figure A.1: Regions of valid P and I values ensuring stability of the PI controller for $d(t) = 0$

Appendix B

System parameters

This Appendix lists and describes the main parameters of the implemented system at Table B.1.

Name	Default value	Description
Sync Interval	1	Time (in seconds) between two **Sync** messages.
P	0.1	Proportional Gain of the PI controller.
I	0.001	Integral Gain of the PI controller.
ACC_STEP	1	Tuner for the PICXO Range and Resolution. Can be fixed in one of the hardware VIO core.
Abrupt change threshold	1	Value (in seconds) determining if the control on the adjustable clock should be done abruptly or by adjusting the clock frequency.

Table B.1: Main System parameters list, default values and descriptions

Appendix C

Hardware PTP emulation

The goal of this Appendix is to describe the hardware PTP emulation that we have implemented in order to verify that under ideal constraints, accurate synchronization can be achieved. By "emulation" we mean an imitation of the real protocol but by reducing or even annihilate sources of noise.

The Figure C.1 is showing the abstract Architecture of the implemented System. Two Gray counters are representing the Master and Slave time. One is fed by an external reference clock and the second is fed by the HDMI clock. Counters are in Gray format to allow clock domain crossing without any trouble as described in our Implementation Section. Values of the counters are passed into an adder whose role is to emulate the Equation 2.6 by giving a perfect value.

The computed value is sampled one time per second to represent the default PTP Sync Interval. A small adder with a register are forming a PI control loop with a Proportional Gain of 1 and an Integral Gain of 1 (for ease of implementation). The Gains are within the stability region defined at Figure A.1 (we can assume here that $d(t) = 0$ since the delay is actually a few cycles). Finally a block formatting the result for the PICXO is implemented.

The command is sent to the PICXO connected to the HDMI, to allow the control of the frequency offset of the HDMI clock.

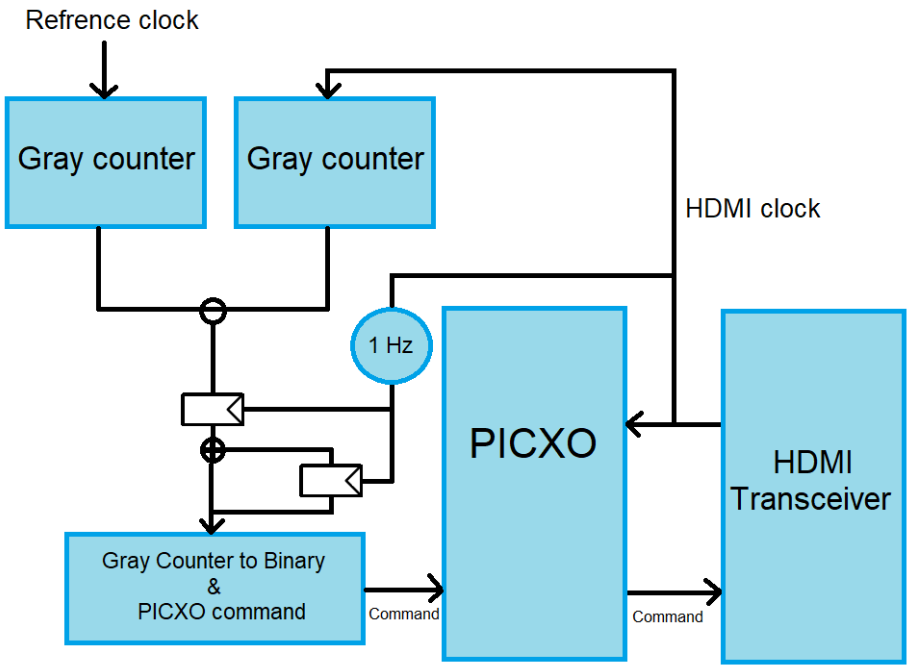


Figure C.1: Architecture of the PTP hardware emulation System

Synchronization performance of that circuit is shown at Figure C.2. We observe that the synchronization error is oscillating between -1 and 1, meaning that the two clocks have, after tuning, only one cycle of error. If clocks of 125 MHz of nominal frequency are used, that means an error of ± 8 ns.

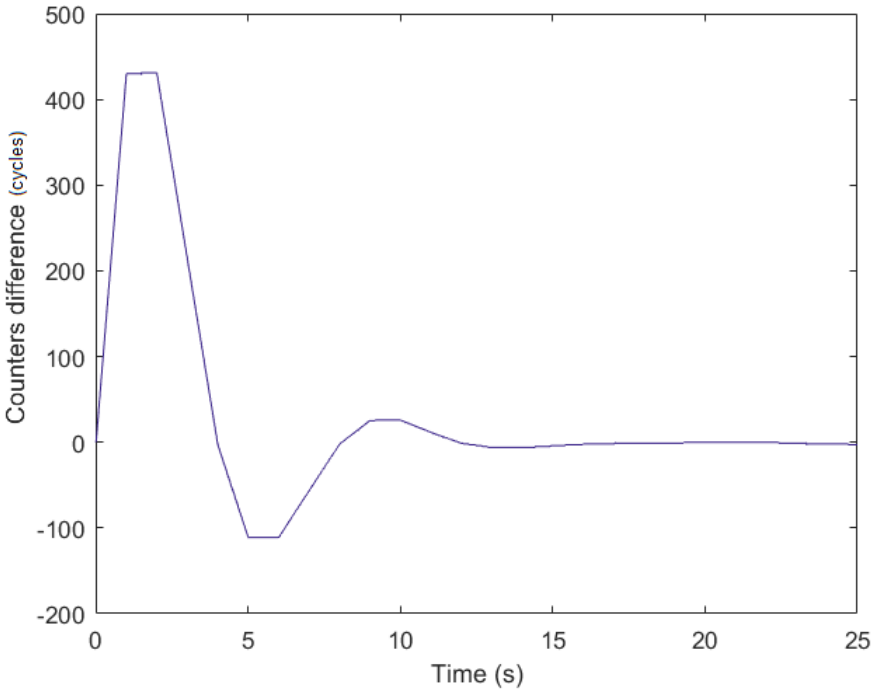


Figure C.2: Difference between two counters synchronized by a PTP hardware emulation combined with a PICXO

Bibliography

[1] Doug Arnold. End-to-End Versus Peer-to-Peer. `https://blog.meinbergglobal.com/2013/09/19/end-end-versus-peer-peer/`, 2013. [Online; accessed 22-05-18].

[2] Aggelos Bletsas. Evaluation of kalman filtering for network time keeping. *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, 52(9):1452–1460, 2005.

[3] Zdenek Chaloupka, Nayef Alsindi, and James Aweya. Clock skew estimation using kalman filter and ieee 1588v2 ptp for telecom networks. *IEEE Communications Letters*, 19(7):1181–1184, 2015.

[4] Wu Chen, Jianhua Sun, Lu Zhang, Xiang Liu, and Liang Hong. An implementation of ieee 1588 protocol for ieee 802.11 wlan. *Wireless networks*, 21(6):2069–2085, 2015.

[5] IEEE Standards Committee et al. Precision clock synchronization protocol for networked measurement and control systems. *IEEE Std*, 1588, 2004.

[6] John C Eidson. *Measurement, control, and communication using IEEE 1588*. Springer Science & Business Media, 2006.

[7] Gene F Franklin, J David Powell, and Michael L Workman. *Digital control of dynamic systems*, volume 3. Addison-wesley Menlo Park, CA, 1998.

[8] Guanghua Gong, Shaomin Chen, Qiang Du, Jianming Li, Yinong Liu, and Huihai He. Sub-nanosecond timing system design and development for lhaaso project. *Proceedings of ICALEPCS2011, Grenoble, France*, 2011.

[9] Intel. *Utilizing IEEE 1588 with Intel® EP80579 Integrated Processor Product Line*.

[10] Maciej Lipiński, Tomasz Włostowski, Javier Serrano, and Pablo Alvarez. White rabbit: A ptp application for robust sub-nanosecond synchronization. In *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on*, pages 25–30. IEEE, 2011.

[11] Jean Baptiste Lorent and Rue Emile Francqui. Tico lightweight codec used in ip networked or in sdi infrastructures. 2016.

[12] David L Mills. *Computer network time synchronization: the network time protocol on earth and in space*. CRC Press, 2016.

[13] David L Mills, Ajit Thyagarjan, and Brian C Huffman. Internet timekeeping around the globe. Technical report, Delaware Univ Newark Dept. Of Electrical and Computer Engineering, 1997.

[14] Teodor Neagoe, Valentin Cristea, and Logica Banica. Ntp versus ptp in computer networks clock synchronization. In *Industrial Electronics, 2006 IEEE International Symposium on*, volume 1, pages 317–362. IEEE, 2006.

[15] Patrick Ohly, David N Lombard, and Kevin B Stanton. Hardware assisted precision time protocol. design and case study. In *LCI Intl. Conf. on High-Perf. Clustered Comp*, 2008.

[16] Gerald Remsak. *IEEE1588 PTP Hardware Implementation in VHDL: IEEE1588 VHDL HW Implementation*. VDM Verlag Dr. Müller, 2010).

[17] Wes Simpson. *Video over IP: IPTV, internet video, H. 264, P2P, web TV, and streaming: A complete guide to understanding the technology*. Taylor & Francis, 2013.

[18] Juhng-Perng Su. *Digital control systems*. SRL Publishing Co., Champaign, IL, 1977.

[19] Tektronix. An Introduction to IP Video and Precision Time Protocol. Technical report, 2015.

[20] Robert Wadge. Media Synchronisation in the IP Studio. Technical report, 2015.

[21] Hans Weibel. The second edition of the high precision clock synchronization protocol. *Zurich University of Applied Sciences: Technology Update on IEEE*, 1588:3–5, 2009.

[22] Xilinx. *All Digital VCXO Replacement for Gigabit Transceiver Applications (7 Series/Zynq-7000)*.

[23] Xilinx. *AXI Reference Guide*.

[24] Hiroshi Yamauchi and Alen Luštica. Audio and video over ip technology. In *ELMAR (ELMAR), 2015 57th International Symposium*, pages 125–128. IEEE, 2015.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**