UNIVERSITÀ DEGLI STUDI DI TRENTO

Faculty of Engineering

Master Thesis on

Telecommunication Engineering

Generating High Accuracy Timing Signals For Time Driven Switching From GPS Receivers Using an FPGA Discrete Filter

Advisor

Dr. Andrea Boni

Co-advisor

Dr. Giorgio Fontana

Author

Leonardo Gasparini

Academic Year 2005/2006

To my sweet grandma, Corinna

Alla mia dolce nonna, Corinna

Acknowledgements

Per lunghi giorni e lunghe notti, sono molte le persone che mi hanno sopportato in questi mesi di fatiche, ed è con piacere che a questo punto il mio pensiero si rivolge a loro.

Il mio primo ringraziamento, il più importante, lo dedico alla mia bella Teresa, per avermi pensato ogni giorno, ogni ora, ogni minuto e, chissà, ogni secondo della giornata in un periodo di ansie e schizofrenie, per aver accettato di vivere con me momenti difficili che presto scompariranno.

Un grazie a tutta la mia famiglia: il re Sandro e la regina Ester, il fratellone Bruno e la nonna Corinna, che mi hanno incoraggiato in tutti i modi ma soprattutto per avermi permesso di uscire dal tunnel dei panini e delle pizzette con piatti prelibati settimanalmente recapitati in via Borino.

Un abbraccio anche alla zia Grazia, agli zii Paola e Giancarlo, Costante e Paola, Federico ed Anna, a tutti i cugini ed in particolare a Iaio, guru della mia vita.

Grazie al mio cane Paco, per aver morsicato il cavo del trasformatore del mio portatile senza restarci secco.

Come vuole la tradizione, un particolare pensiero lo dedico alla mia seconda famiglia, la famiglia Peterson: Camicia di Forza per essere stato in grado di trovare qualsiasi cosa io perdessi, Gonzo per aver digerito a suon di risate tutti gli insulti che gli ho rivolto, Matrix e Rimba per aver subito i miei scherzi quasi senza batter ciglio (al massimo un po' di tachicardia), la zia Sam e lo zio Umbe per le ottime cene a base di zampone ed infine il gemello Collasso, emigrato lontano, ma che ogni tanto ritorna a farsi un pisolino. Grazie anche ad Aldo e alla Antonia, per avermi concesso di vivere in una casa così calda e spaziosa.

Un grazie a tutti i miei amici: 100, Burgy, Cerro, Enrico, Fulmine, Mattia e Lisa, Napo, Nicola, PieroBZ, Pilota, Tonino, Virile, i compagni del Buena e delle mitiche feste ai campi e tutti quelli dimenticati. Un abbraccio anche a Pierluca, che cotidianamente ritorna nei miei pensieri.

La mia esperienza in facoltà non sarebbe stata la stessa senza l'allegria di Burgy, Giambo, Mattia Broilo, la Vale, e l'incomprensibilità delle parole degli Erasmus dalle valli trentine, ovvero Chini, Dal Follo, Endrici, Gionta e Zadra.

Thanks also to ChessMaster Troy Goldschmidt, who never let me turn to the Dark Side of the Force, and all the people I've been living with during my Erasmus experience in Helsinki.

I would like to thank also Dr. Andrea Boni, Dr. Giorgio Fontana and Dr. Ofek for having given me the chance to partecipe to this project, Francesco, Franco, Luca and Michele for their help during the hard days of work in the laboratory of electronics.

In the end, thanks to David Pajo and the Dirty Three, whose celestial music defeated my sleeplessness.

Contents

List of Figures

List of Tables

"The Web infrastructure, and even Google's [infrastructure] doesn't scale. It's not going to offer the quality of service that consumers expect"

Vincent Dureau, Google's head of TV technology
Amsterdam, Cable Europe Congress
7th Febrary 2007

"Network synchronization plays a central role in digital telecommunications as it determines the quality of most services offered by the network operator. However, the importance of network synchronization is often underestimated [...]" [8]

Introduction

The Internet has been growing exponentially for several years and the demand for broadband services is going to find new reasons to increase from streaming media flows, requiring a lot of bandwidth in order to fulfill customers' request for high definition video. Moreover, real-time applications, such as video-conferencing, video-telephony and gaming, require small delay to allow them to be fully enjoyed by customers. But the actual infrastructures seem to be inadequate to satisfy the future requirements, so new technologies are needed to prevent the network from collapsing.

In 2004, Marie Curie Chair Professor Yoram Ofek started a project called **IP-FLOW**, founded by the European Union, developing a new architecture for optical networks based on *Fractional Lambda Switching*, an innovative technology for the realization of low complexity, high scalability switches invented by Ofek himself. In 2006, after less than two years of hard work involving people and structures from three italian universities (of Trento, Milan and Turin), a prototype was successfully developed and tested.

Pipeline forwarding is the method at the base of an ultra scalable switch presented as the solution to the problem. In this context, a major role is played by **synchronization**: packets flow over the optical network from the server to the client without the need of processing overhead, but the switching operation is driven by a timing signal based on *Coordinated Universal Time (UTC)* providing *common time reference (CTR).*

But the synchronization issue does not end here: the upcoming activation of the *Galileo positioning system* shows the way forward to new applications, especially in the telecommunication field, as shown by the growing interest on conferences like the *International Telecoms Synchronization Forum.* From this perspective, the higher is the precision of the timing signal, the higher is the upper-bound performance achievable by the system.

Improvements in the same field are also connected to the astonishing success of FPGAs, a type of powerful, small and inexpensive semiconductor device containing programmable logic components and programmable interconnects. According to In-Stat [10], a high-tech market research firm, the value of worldwide FPGA shipments will increase from $1.9 billion in 2005 to $2.75 billion by 2010; moreover, communications are one of the largest two end-use segments whose combined market share of the FPGA market will increase from 73.8% in 2005, to 76.8% by 2010.

In the synchronization field knowledge is still extremely meager while real-time Internet applications are increasing their catchment area without the certainty of having enough resources to satisfy the need for deterministic *Quality of Service.*

The purpose of the presented thesis project is to generate high accuracy timing signals from the *UTC-one pulse per second (UTC-1PPS)* signal, which nowadays can be easily derived from inexpensive GPS receivers, with a low-cost FPGA. The realized device will then advantageously replace the actual GPS board which is now supplying the switch prototype with low jitter *1PPS* and 10MHz *clock* signals used for switching operation.

The thesis is structured in the following way:

in the **first chapter** an overview on the **IP-FLOW** project is given, explaining its basic principles, focusing mainly on the synchronization issue, and describing the prototype's architecture;

the **second chapter** focuses on the thesis project objectives, defining the system constraints related to the timing aspect;

in the **third chapter** all the devices and instruments used for the development of the project will be described in details; thus, not only the ones constituting the final device will be treated, but also the ones used for debugging, testing and comparing, which have been very useful to identify the source of the various problems and find the corresponding solution;

the **forth chapter** is dedicated to the software used for programming the FPGA and evaluate the performance of the device, concentrating on the most useful tools used during the implementation, debug and test phases;

the **fifth chapter** focuses on the algorithm which the thesis project is based on; the first practical problems will be identified also, as well as their solutions;

in the **sixth chapter** the first VHDL code implementation phase will be discussed, in which the generation of a clock signal with high phase and frequency accuracy from a precise *1PPS* signal was carried out;

in the **seventh chapter** the testing of the implemented VHDL code will be described, focusing on the non idealities introduced by the several components and the related problems' solution;

the **eight chapter** focuses on the reduction of the jitter affecting the input reference signal; the proposed solutions will be explained concentrating on the system response to the filtering operation;

the **ninth chapter** is dedicated to the measurements that have been made; the performance of the designed product will be evaluated in order to understand the limitations the algorithm suffers from;

in the **tenth and last chapter** the thesis conclusions will be drawn; moreover, future developments will be suggested.

Chapter 1

Background and thesis objectives

1.1 the IP-FLOW project

IP-FLOW [16] is a three-year European project supported in full by Marie Curie Chair Excellent (EXC) Actions and has been awarded to Prof. Yoram Ofek. The focus of IP-FLOW project is on the flow control of IP packet over the Internet. Timing and flow control are critical since they directly affect how users perceive the quality of IP-based services, such as voice-over-IP or video-over-IP. Its final goal is to provide an efficient solution to the flow control problems that internet is going to experience when real-time applications like video-conferencing, video-telephony and gaming, whose *Quality of Service (QoS)* requirements (in particular regarding delay and jitter) can not be fulfilled by the actual infrastructure, will be widely spread around the world. Nowadays, circuit switching is the only known way to ensure it, but this technology suffers from some serious technical limitations; circuit switching networks are in fact bandwidth-inefficient and restrictive in supporting a diverse array of end-user services. On the other hand, packet switching and statistical multiplexing can provide a better network utilization but for these technologies it is currently impossible to provide deterministic QoS despite lots of sophisticated queuing algorithms have been deployed: some with good results, but none of them can actually ensure this kind of requirement [13].

IP-FLOW is an acronym for **I**nternet **P**rotocol **FL**ows over **O**ptical and **W**ireless: in fact, the underlying research and the project itself are focused on several

IP packets flow issues through IP networks, from the optical core to the wireless edges.

The IP-FLOW project develops in three research areas:

1. *UTC-based pipeline forwarding* for IP flows for solving switch and link bottlenecks;
2. *TrustedFlow*: a general method for run-time authentication that guarantees correct execution of service level agreements and window flow control;
3. delivering high fidelity content over wireless (Wi-Fi) with directional antennas.

In this thesis, the first area will be widely described, while the other two ones are still under preliminary research.

IP-FLOW development started in April 2004, involving people and structures from three italian universities: Università degli Studi di Trento, Politecnico di Torino and Politecnico di Milano; many activities have been proposed, planned and executed ever since, leading to the realization of a Terabit/s switch prototype based on the *Fractional Lambda Switching (FλS)* technology [7] featuring *QoS* guarantees (deterministic delay and jitter, no loss) for (UDP[1]-based) *constant bit rate (CBR)* and *variable bit rate (VBR)* streaming applications.

This thesis deals with **synchronization**, an issue of major importance in this context.

1.1.1 Principles of operation

Unlike whole λ switching, FλS dynamically switches fractions of a wavelength allocated with the proper size to satisfy the specific needs of the access networks to which a λ fraction is connected.

Pipeline forwarding [7], [3] is the fundamental mechanism that lays at the base of the switch: the idea is to translate in the optical networks environment the

[1]Acronym for User Datagram Protocol; it is an unreliable protocol belonging to the transport layer of the the TCP/IP model, used in applications such as VoIP.

assembly line, i.e. the manufacturing process of historical importance firstly successfully realized by Henry Ford in his car factories at the beginning of the 20th century.

This is done taking advantage of the *UTC*[2] *One Pulse Per Second (1PPS)* signal provided by a GPS receiver, thanks to which it is possible to have the necessary requirement of *Common Time Reference (CTR)*, used by the network nodes to get *phase synchronization* with *identical frequencies*.

Nowadays, GPS receivers can be easily found in the market at different prices, depending on the quality of the service provided. Moreover, the GALILEO project, the satellite positioning and navigation system born from the collaboration between the European Union and the European Space Agency and specifically designed for civil purposes, will be fully operational in the immediate future (according to the last estimates, everything will be ready before 2012), allowing higher precision than is currently available through GPS [19].

In this way, time can be universally divided into the basic elements called *Time Frames (TFs)* of duration T_f, which are grouped into *Time Cycles (TCs)*, which are in turn grouped into one UTC second long *Super Cycles (SCs)*. For example, as shown in figure 1.1, 100 *TFs* form one *TC* and 80 *TCs* are included in a *SC*, which is identified with one *UTC* second.

Therefore, an end-user demanding for a service experiences the advantages of circuit switching, since packets *flow* over the network through a pre-defined path, but without wasting bandwidth, since the same wavelength[3] is shared with other users by dividing time in the way above explained. The *TFs* are then associated to every single user depending on two factors: the *bandwidth* needed for the specific application and the *delay* it can tolerate. The traffic is shaped according to the *Time Driven Priority (TDP)* mechanism whose design basic objective is to satisfy real-time multimedia services by giving higher priority to real time traffic in a periodic fashion [12], [13].

[2]Acronym of *Universal Coordinated Time*, it is the international atomic time scale that serves as the basis for timekeeping for most of the world. UTC is a 24-hour timekeeping system. The hours, minutes, and seconds expressed by UTC represent the time-of-day at the Earth's prime meridian ($0°$ longitude) located near Greenwich, England.

[3]Actually data won't be carried on the same frequency all along the path, but at every node it will be generally moved from a wavelength to another

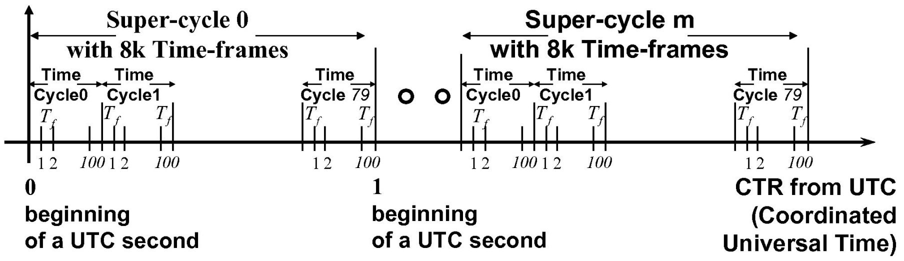


Figure 1.1: Common Time Reference structure.

In the proposed solution, packets flow over the optical network according to two rules:

1. all packets that must be sent in the t^{th} *TF* by a node must be in its output ports' buffers at the end of the $(t-1)^{th}$ *TF*;
2. a packet p transmitted in the t^{th} *TF* by a node n must be transmitted in the $(t+d_p)^{th}$ *TF* by node *n+1*, where d_p is an integer constant called *forwarding delay*; the t^{th} and the $(t+d_p)^{th}$ *TFs* are also referred to as the forwarding *TF* of packet p at node n and node *n+1*, respectively. The value of the forwarding delay is *determined at resource-reservation time*, i.e. it is known, and must be large enough to satisfy rule 1.

In figure 1.2 there's an example of how this can happen. A, B and C represent the network nodes the packet has to go through, T_{ij} stands for the delay introduced by the propagation from node i to node j while T_{ii} represents the processing delay due to node i. Hence, the forwarding delay can be calculated according to the following pseudo-code:

$$d_p = round\_up(d_{propagation} + d_{processing}); \tag{1.1}$$

where $d_{propagation}$ is the *propagation delay* and $d_{processing}$ is the time required by the node for *processing*. The basic unit to which the rounding operation refers to is, of course, the *TF*.

The predefined schedule for forwarding a pre-allocated amount of bytes during one or more TFs along a path of subsequent UTC-based switches is known as *Syn-*

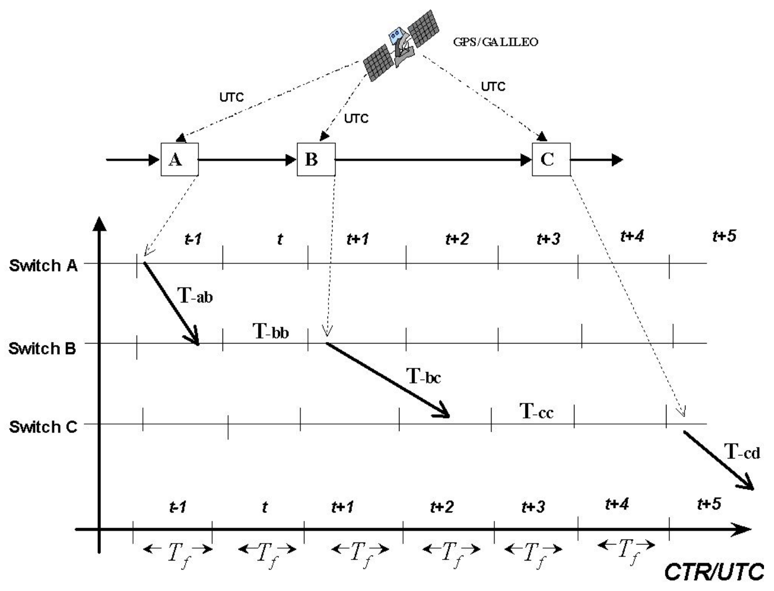


Figure 1.2: Example of packet flowing in a *UTC-based pipeline forwarding* architecture.

chronous Virtual Pipe (SVP) [3]. The *SVPs* are calculated and provided by a *TDP* router directly connected to the service provider, e.g. a video streaming source; the flow is then managed by the switches, which perform a series of operations that can be summarized by two steps, as described by figure 1.3:

1. *receiving and alignment*: the switch receives the data units belonging to a *TF* on each wavelength separated by a WDM de-multiplexer, but at the input port the *TFs* are usually not aligned because of the propagation delay, which is generally not a multiple of T_f; thus, some *CTR* aligning procedure is needed performed by an alignment system, which also provides input buffering of data units until they can be switched;

2. *switching and forwarding*: after step 1 has been accomplished, all the data units are transferred through the switching fabric to their corresponding output port, where they are transmitted on their selected λ through a WDM multiplexer.

There are two different approaches in doing this: one implementing *immediate forwarding*, in which the data units received in the t^{th} *TF* are moved to their output

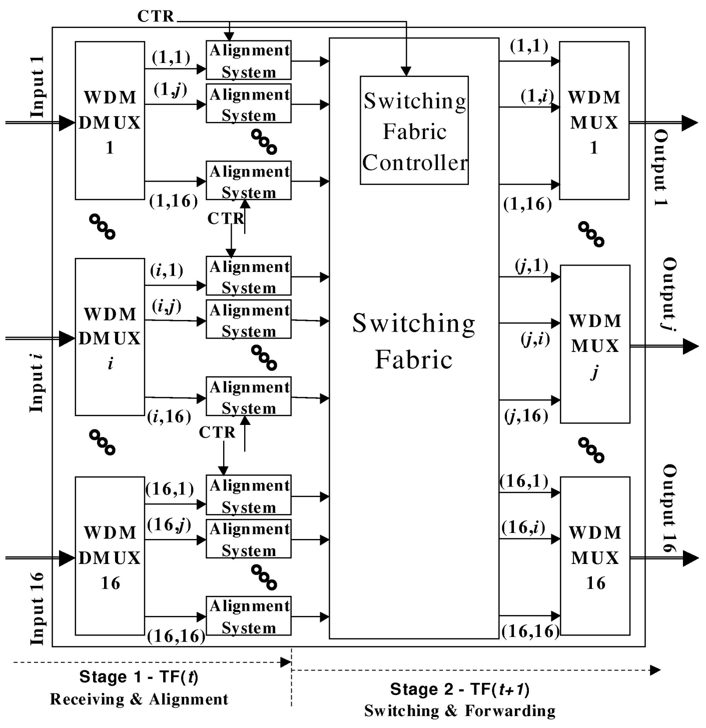


Figure 1.3: Fractional λ switch architecture and operating stages.

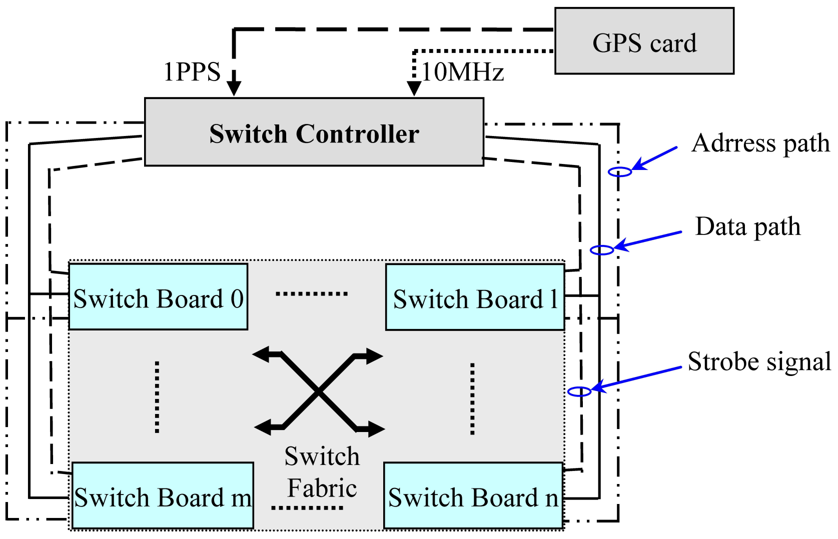


Figure 1.4: Functional diagram of a Time Driven Switch.

port and forwarded in the $(t+1)^{th}$ *TF*; the other implementing *non-immediate forwarding*, in which the two steps are not necessarily carried out in consecutive *TFs*.

1.1.2 Timing signals

It is now clear that synchronization is a fundamental aspect in the switching operation, which the whole system can not prescind from: for this reason, this technology is also known as *Time Driven Switching (TDS)*.

There two timing signals, which are provided to every single switch by GPS cards, as shown in figure 1.4. The first one is the *1PPS* signal, used to identify uniquely the first *TF* of every SC. Then, a *10MHz clock* is furnished to the FPGA that controls the switching fabric and that uses it to build the *TFs*. This clocking signal is synchronized in frequency and in phase with the *1PPS* signal, i.e. it is generated in such a way that there are exactly $10,000,000$ cycles between two consecutive rising edges of the *1PPS* and that the first and last cycles of every SC are aligned with the pulse. In such a way it is possible to achieve *CTR*, which the whole system is based on, since it is essential during for alignment phase of the switching procedure.

1.1.3 The prototype

The architecture described so far has been implemented on a testbed in the Electronic Laboratory of the Department of Information and Communication Technology at University of Trento. The prototype, whose functional diagram is shown in figure 1.5, is composed by the following components:

- a video server emulating heterogeneous and asynchronous streaming sources (audio, video and text) by transmitting two different movies to two distinct clients;
- TDP router that represents a TDP domain at the backbone edge; the TDP router acts both as edge router encompassing a SVP interface and as an interface to the TDS backbone [11], [12], [6];
- two TDS switches that are constructed with *Mindspeed* **M21151** switching boards, with capacity of 400 Gb/s;
- two distinct FPGA-based controllers for the two switches;
- three GPS boards, providing the needed timing signals;
- two streaming media receivers for separately playing the two video streams transmitted from the asynchronous streaming sources;
- 20 km single-mode optical fiber connecting the two switches;
- multi-mode optical fiber to connect the router with to the first switch and the second switch to the two clients.

The *TF* duration T_f and the size of the *TCs* have to be set in the *TDP* router and the two switches.

Two asynchronous video flows are generated by the streaming media source and transmitted to the SVP interface within the *TDP* router. The streaming video packets are then forwarded via an optical link by the *TDP* router through Gigabit Ethernet (GE) transceivers to the first *TD* switch during different predefined *TFs*. Specifically, in reference to 1.5, the 5^{th} *TF* and the 15^{th} *TF* belong to one *SVP* while

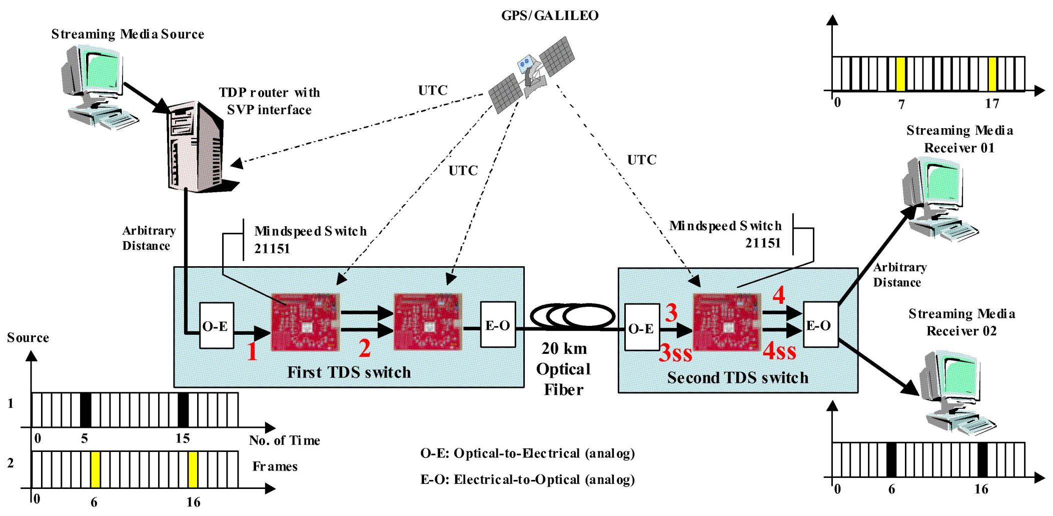


Figure 1.5: Testbed functional diagram.

the 6th *TF* and the 16th *TF* belong to another *SVP*. The optical signal entering the first *TD* switch is converted to an electrical one, switched by the first stage and forwarded to the second switching stage through an electrical connection. Then the video streams are transmitted as an optical signal, through a single mode optical fiber link of 20 km, to the second *TD* switch that routes each video stream to a different output. Then the separated video streams are forwarded to two receivers through optical links of an arbitrary length. Video flows are therefore multiplexed on the first and second link they traversed, but *TDS* ensures that video packets reach their corresponding destination with deterministic QoS, i.e., during the 6th *TF* and the 16th *TF* and during the 7th *TF* and the 17th *TF*respectively. Switching of all three switching boards and network interfaces are synchronized with the *1PPS* signal received from three different GPS receivers [3].

1.2 Objectives

Synchronization plays a major role in the entire project: without *well-defined timing uncertainty* in the alignment, the optical switches can not forward data units in the correct direction, since it is time that drives the switch. Thus, high accuracy timing signals are needed in order to ensure a correct flow control of the IP packets but the question is: how close to the ideal case does the alignment have to be?

1.2.1 Accuracy requirements

There are two issues that have to be considered: one connected to the jitter affecting the *TF* connected clock, the other,which is definitely more delicate and crucial, connected to the accuracy of the *1PPS* signal aligned to UTC.

The former is easily resolvable once we have high precision *1PPS* signal delimiting the time interval in which there has to be an *exact* number of *TFs*, i.e. between two consecutive input pulses no *TF* has to be missing and there has to be no time for additional *TFs* over the pre-defined quantity. The device dedicated to the supply of the timing signals, a low frequency one and a high frequency one, set to 1Hz and to 10MHz respectively, has to tune the frequency of the last mentioned clocking signal according to the first one in such a way that the number of output cycles contained in a *SC* respects the network design parameter related to that particular node. As a consequence, the accuracy issue is translated into a stability question regarding the oscillator internal to the device. Fortunately, the answer is already given, since, short of abrupt physical environment changes, the actual quartz oscillators achieve really high despite short term stability: a typical crystal oscillator's frequency vs. temperature (which is the parameter the oscillators are more sensitive to) stability may be $\pm 25 ppm$ for a temperature range of -55°C to $+85$°C [20].

The latter issue requires more dedication since different alternatives can be implemented, but all sharing the same basic mechanism: the *1PPS* signal is used for the identification of the first *TF* of the first *SC* included in a *UTC* second, obtaining in such a way synchronization among all the optical switches constituting the network. Depending on how the whole system is designed, the switching operation turns out to be *non-ambiguous* if the following accuracy requirements are fulfilled:

1. in case the *TFs* are not bounded by any delimiter, the timing uncertainty has to be much smaller (e.g. 5%) than T_f; moreover, a *safety margin*, i.e. an *idle time* in which no data is transmitted, between the *TFs* is needed in order to be sure not to forward fractions of *TFs* in the wrong direction; such

a safety margin is not intended to delimit the *TF*, since also inside a *TF* itself there could be some idle time, e.g. in case no one is asking for bandwidth. Obviously, the absence of delimiters does not allow any alignment operation: *TFs* can not be distinguished from each other and, as a consequence, the incoming signal is seen as a continuous stream of data; therefore, no buffering is supported, i.e. the *forwarding delay* is equal to the *propagation delay.* In order to allow correct packet forwarding this delay has to be an *integer multiple* of T_f, requiring precise calculations of the lengths of the optical fibers connecting the network nodes;

2. in case the system is provided with delimiters, things get much easier: *delimiters* can be implemented in various ways ranging from defining a control packet to be transmitted as a delimiter to setting a 1 bit field in the first packet transmitted during a *TF*; the requested accuracy to maintain correct mapping of *TFs* is therefore lowered to $\pm\frac{1}{2} \cdot T_f$ [7].

3. accuracy relaxation can be enhanced exploiting the re-occurring nature of the *TCs*: *TFs* can be counted for univocal identification within the *TC* whose half period consequently represents the modulus of the minimum accuracy required for correct forwarding;

4. alternatively, the accuracy requirement can be relaxed if the use of *Time Stamps* is introduced. A *Time Stamp (TS)* is a record of the time associated to a certain event. In this case, since time is divided into *TFs*, which represent the basic time unit for the system, the *TS* is identified with the *TF*'s index within the *TC*; again, the minimum accuracy required is lower, in modulus, than half the *TC* period. The mechanism could paradoxically be propagated to *TCs* into *SCs*, but obviously such a requirement goes against the idea of *real-time* application, besides the fact that even low cost GPS receivers accuracy is in the order of few microseconds. In the light of this last consideration, *TSs* can be useful for *network fault detection*: if the misalignment exceeds a pre-defined threshold value, it means that the timing signal generator is not working properly and the affected node has to be considered out of order. Thus, the system's fault tolerance capability could be extended

since it would be possible to isolate the faulty node by the execution of a recovery procedure with the aim of generating new routes that avoid that node.

The insertion of delimiters give flexibility to the switching operation but there's a major issue related to their complexity, while the *"delimiters-free"* approach is simple and supported by the precision of actual GPS receivers combined with the elaboration power of the existing programmable logic devices.

1.2.2 Previous configuration

In the actual prototype, *Tekelec*'s **Epsilon Board OEM II** is the device which has the task of providing both the timing signals to the switch controller. The board does not simply include a GPS receiver which provides the *UTC 1PPS*, but is made of a high quality and, as a consequence, high cost mixed analog/digital technology which allows to obtain really high accuracy timing signals: an oven stabilized crystal oscillator with *1PPS* output is synchronized to *UTC 1PPS* with a *Phase-Locked Loop (PLL)*.

A *PLL* is an electronic circuit implementing a closed-loop feedback control system designed to generate an output wave at a specific frequency, synchronized with another input signal (a.k.a. "reference signal") whose frequency is different. It is generally made up by the following components, as shown by figure 1.6:

- a *phase comparator*;
- a *voltage-controlled oscillator (VCO)*;
- a *frequency divider*.

Let's assume that the *VCO* generates an output wave whose frequency is closed to the desired one; the divider returns a signal whose frequency is a submultiple of the one generated by the *VCO* and compares its phase with the reference signal's one. The output of the comparator then drives the *VCO*, keeping its frequency strictly phase-locked to the input.

Tekelec's GPS board is designed in a more complex way: the phase shift is measured and commanded to zero with a digital PID controller that analogically

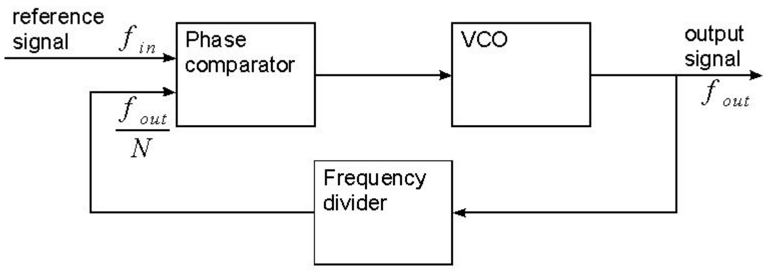


Figure 1.6: Phase-Locked Loop - Scheme of principle.

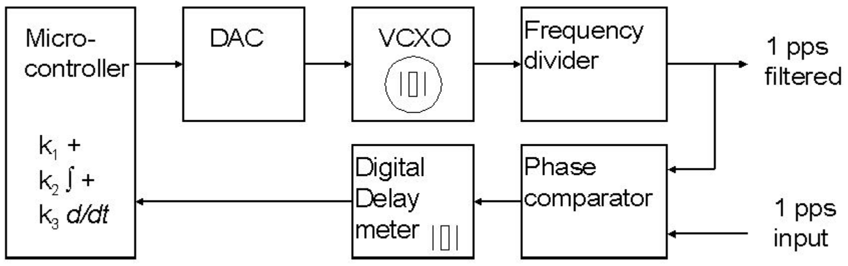


Figure 1.7: Block Scheme of the *Tekelec*'s **Epsilon Board II**.

drives a varactor diode in the crystal oscillator. The digital PID is required because the involved time constants are very large and may reach hours. The principle of functioning and the hardware setup of the evaluation kit in which the device is included are illustrated in figures 1.7 and 1.8.

The purpose of the whole thesis is to replace the **Epsilon Board OEM II**, providing the same timing signals, with a digital device implemented on an FPGA, trying to get as close as possible to its performance, which is known to be really good (this aspect will be discussed deeply in chapter 5, and at least providing the minimum accuracy requirements above highlighted.

The motivations that lead to the realization of such a component are:

- *Tekelec*'s **Epsilon Board OEM II** is an almost unique device in the market: there are no many companies providing products with similar purposes, but the whole testbed needs to be independent with respect to the market;

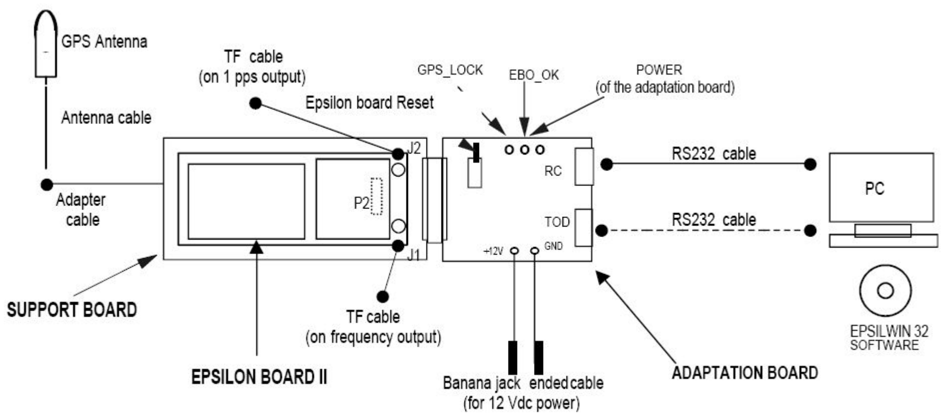


Figure 1.8: Hardware setup of the *Tekelec*'s **Epsilon Board OEM II**.

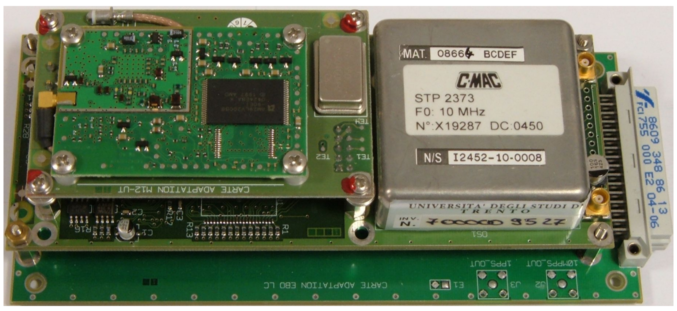


Figure 1.9: *Tekelec*'s **Epsilon Board OEM II**.

- *Tekelec*'s **Epsilon Board OEM II** is too expensive,
- *Tekelec*'s **Epsilon Board OEM II** is bulky, while an FPGA can contain the code corresponding to the timing signals provider and still have space for additional logic.

Chapter 2

Devices and instruments

In order to attain the pursued objectives stated in chapter 1.2, several devices and instruments were used. Here's a list of them, with their main features described and particular care given to the most useful aspects regarding this project. Moreover, the achievement of the objectives laid down at the beginning of the project required the capability to make many different devices interact. The creation of the VHDL code, its simulation in order to fix the bugs, the estimate of its efficiency, programming the FPGA and the measurements of the system performance are examples of such kind of activities that were lead by heterogeneous software. In this chapter the computer programs used in the whole development phase will be described too.

2.1 Main devices

The expression *"Main devices"* refers to all the elements that constitute the final sub-system[1], i.e. without all the instruments used for initial implementation, debugging and testing, as shown in figure 2.1. Thus, the GPS receiver and the CardBus FPGA Development Platform will now be described in details.

The *1PPS* signal is provided to the FPGA by the *U-BLOX*'s **LEA-4T** GPS module, shown in figure 2.2.

[1]With the term sub-system, the GPS board, matter of the thesis, is meant, while the term system refers to the whole $F\lambda$ switch prototype.

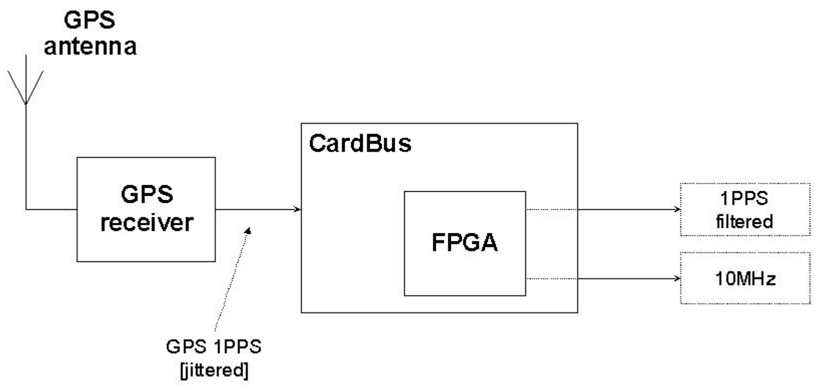


Figure 2.1: Final system.

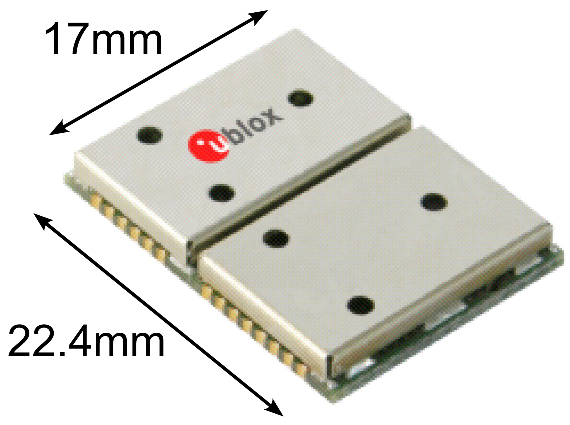


Figure 2.2: *U-BLOX*'s **LEA-4T** GPS receiver.

2.1.1 GPS receiver #1

It's main features, according to the data sheet, are:

- dimensions: $17 \times 22.4 \times 3mm$;
- configurable time pulse: $0.1Hz$ to $1kHz$;
- time pulse accuracy: $50ns$;
- digital I/O interface for the time pulse signal;
- Power Supply: $2.7 - 3.3V$;
- ultra low power consumption: typically $39mA$ @ $3.0V$ or $38mA$ @ $2.7V$;
- operating temperature range: -40 to $85\,°\mathrm{C}$;
- rise time : $800ns$ (see figure 2.6 for a comparison with *Tekelec*'s **Epsilon Board OEM II**)
- price: 145€.

The GPS module is provided with a software that allows to set the receiver's parameters, and gives informations about the signals captured by the antenna, such as the strength, the transmitting satellites sending the information and their position. The software then decodes the messages received, containing exact time and 3D space coordinates.

The work does not take into account any elaboration that could be operated by the receiver, since the purpose is to be able to generate high accuracy timing signals from every GPS receiver, thus taking into consideration worst-case scenarios (i.e. "raw", unprocessed signals). Anyhow, if better precision is needed, every option can be exploited.

2.1.2 FPGA #1

The FPGA in which the timing signal generator has been implemented is included in a PCMCIA/CardBus FPGA development platform model **COM-1300** by *ComBlock*, of which a picture is provided in figure 2.3.

Figure 2.3: *ComBlock*'s **COM-1300**.

Its main features are (in the CardBus configuration):

- typical sustained data throughput: 100/40 Mbits/s (transmit/receive);
- *Xilinx* **Spartan-3 XC3S400-4ft256 FPGA**: 400K gate Xilinx Spartan-3 FPGA with sixteen 18-bit multipliers, 288Kbits of block RAM, and up to 500MHz internal clock speeds (50MHz internal clock multipliable and divisible through DCMs[2];
- 32MB SDRAM for use as elastic buffer;
- GUI[3] called "*ComBlock Control Center*", used for remote monitoring and control over simple serial link;
- ComScope, a component which allows to store internal digital signals in real-time within the internal memory and then export them to a host computer for plotting, storage and further processing.

The GPS module is installed on a circuit board connected to the FPGA through the expansion port of the **COM-1300**; in such a way the **LEA-4T** is power supplied while providing the FPGA with the *GPS-1PPS* signal. The same board also serves the function of connecting the FPGA to the *Mindspeed* **M21151** switching

[2]Acronym for Digital Clock Manager, a component capable of generating clocks at whichever frequency, belonging to the output frequency range, through the use of a DLL unit. acronym for Delay Locked Loop

[3]Acronym for Graphical User Interface.

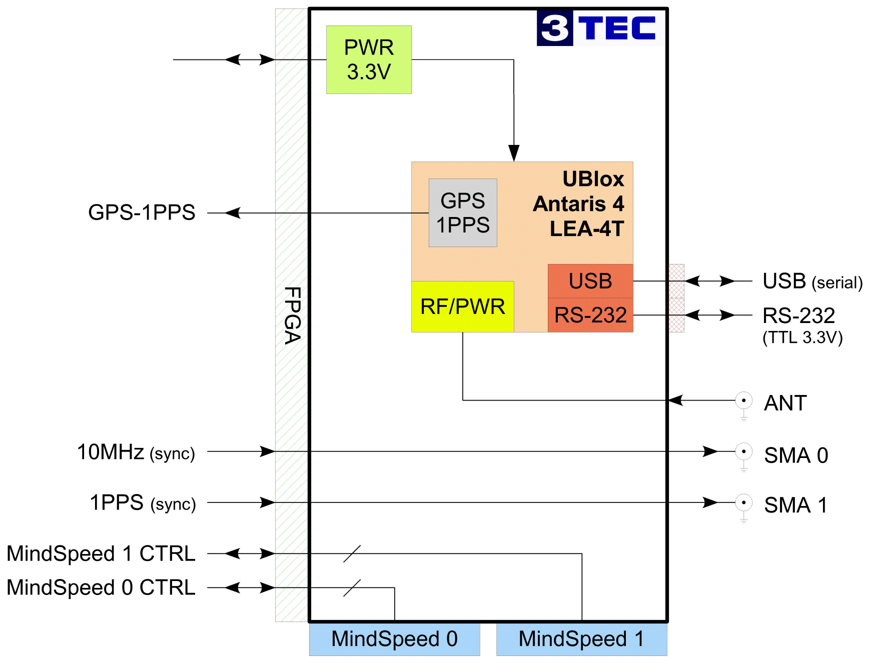


Figure 2.4: *UBlox*'s **LEA-4T** circuit board.

boards for control. Moreover it allows USB and RS-232 communication and output signals monitoring through *SMA*[4] connectors (refer to figure 2.4).

2.2 Other devices

Many other devices were used during the implementation, debug and test phases, used to simulate input signals, reproduce **COM-1300** behavior and measure output quality. Here is a list of them.

2.2.1 FPGA #2

During the development phase, the VHDL code was not implemented on the CardBus platform, but a different device was used for debugging and first testing: *Digilent*'s **Spartan-3 Starter Board**, whose main characteristics are:

[4]Acronym for SubMiniature version A connector, it is a a type of coaxial RF connector developed in the 1960's as a minimal connector interface for coaxial cable with a screw type coupling mechanism.

- *Xilinx* **Spartan-3 XC3S200-4ft256 FPGA**: 200K gate Xilinx Spartan-3 FPGA with twelve 18-bit multipliers, 216Kbits of block RAM, and up to 500MHz internal clock speeds (50 MHz internal clock multipliable and divisible through DCMs;
- on-board 2Mbit Platform Flash (XCF02S);
- 8 slide switches, 4 pushbuttons, 8 LEDs, and 4-digit seven-segment display;
- serial port, VGA port, and PS/2 mouse/keyboard port;
- three 40-pin expansion connectors;
- three high-current voltage regulators (3.3V, 2.5V, and 1.2V);
- works with JTAG3 programming cable.

The complementarity with the CardBus board was ensured by the belonging of the included FPGAs to the same family (*Xilinx Spartan-3*).

The Starter Board was really helpful thanks to the display, which allowed to monitor the content of any counter, selectable through the use of the switches, thanks to the leds, which allowed to monitor the system flags, such as resetting and enabling signals, thanks to the buttons, used to simulate external interfering elements, and finally thanks to the expansion ports which the interface with the other elements composing the whole system was realized through.

The **Starter Board** features an FPGA which is half sized in terms of logic space if compared to the one included in the CardBus. However, it has also to be considered that part of the *Spartan3* of the **COM-1300** is already filled with drivers for the CardBus interface.

2.2.2 Function generator

At the very beginning of the development phase, the 1PPS signal sent by the satellites and captured by the GPS receiver has been simulated by a function generator, which allows to build digital signals such as sinusoids, square waves, noise and pulse-like signals (as requested by the application) allowing to set the

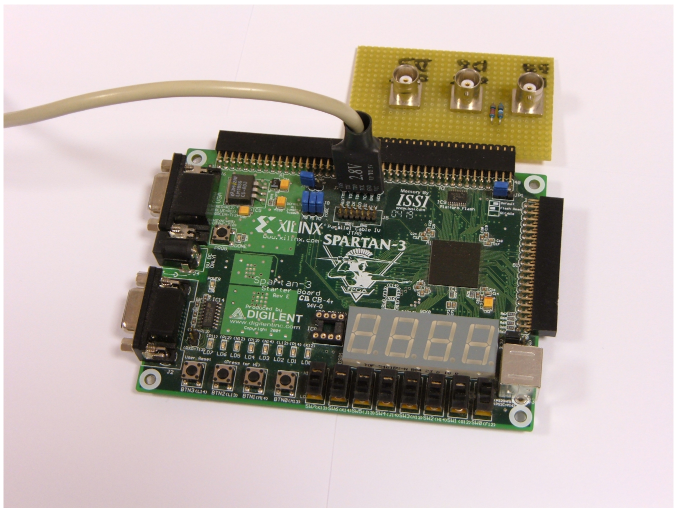


Figure 2.5: *Digilent*'s **Spartan-3 Starter Board**.

frequency in a dynamic way, i.e. whenever needed, also during the generation of the output, through the use of buttons and a knob.

2.2.3 Oscilloscope

The assessment of the system has been carried on mainly through the use of a digital real-time oscilloscope featuring two input channels, 9-bit vertical resolution, sample rates up to $1.25 GS/s$, automatic delay measurement and LAN communication port. In addition to the monitoring of the signals on the color LCD, a program developed in LabVIEW environment has been configured to communicate with the oscilloscope with the purpose to collect and analyze the delay acquisitions between the various 1PPS signals.

2.2.4 Universal counter

The accuracy of the $10 MHz$ square wave output has been estimated not only using the oscilloscope, but also through the use of a universal counter featuring two input channels, 10 digits per second of frequency/period resolution and a bandwidth of $225 MHz$ and capable of frequency ratio measurements between the input channels.

2.2.5 GPS receiver #2

The absence of an extremely high precision clock, e.g. an atomic clock, does not allow to quantify univocally the accuracy of a timing signal. Thus, the measurements have been performed comparing Tekelec's output with the generated one, but in order to understand the origins of the different behaviors that can be observed by the acquired data, another receiver has been used, allowing cross-comparisons: *Trimble*'s **Resolution T**. The results of the analysis will be discussed later on in section 4.2.

Another parameter that characterize the three receivers is the rise time, which is directly connected to the bandwidth of the circuit.

In figure 2.6 the time needed by the three GPS receivers to change from 10% of its final value to 90% of its final value is shown. As it can be seen, *UBlox*'s **LEA-4T** has a really long rise time: this is the price that has to be paid when reducing the device dimensions in such a drastic way.

2.3 The FPGA-related software

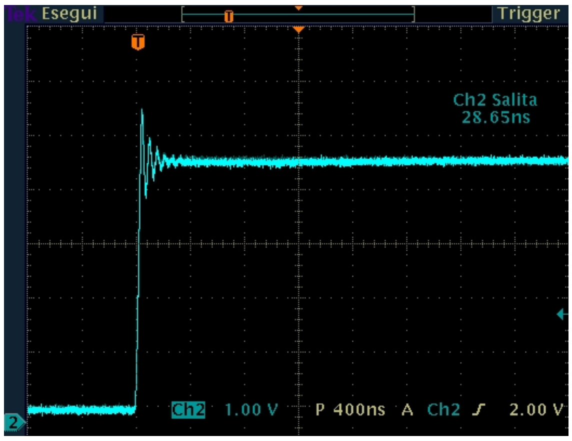
Xilinx is the worldwide leading company in the FPGA field. Together with its programmable logic products, it provides software for the implementation of the algorithm into the devices. In the case of this project, the following programs were used:

- *Xilinx*'s **ISE 8.1i**, providing a HDL synthesis and simulation, implementation, device fitting, and JTAG programming environment (see its graphical interface in figure 2.7);

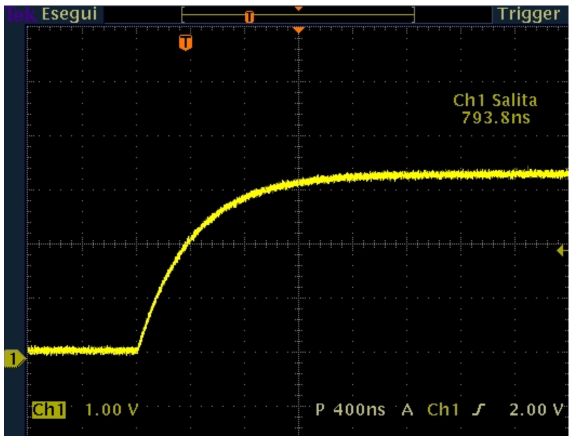
- *Xilinx*'s **ModelSim XE III v6.0d**, providing an HDL simulation environment that enables to verify the functional and timing models of the implemented design and the HDL source code (see an example of the generated output in figure 2.8).

In particular, **ISE 8.1i** features the *IP*[5] *CoreGen & Architecture Wizard* utility, which allows the designer to utilize many types of ready made intellectual property

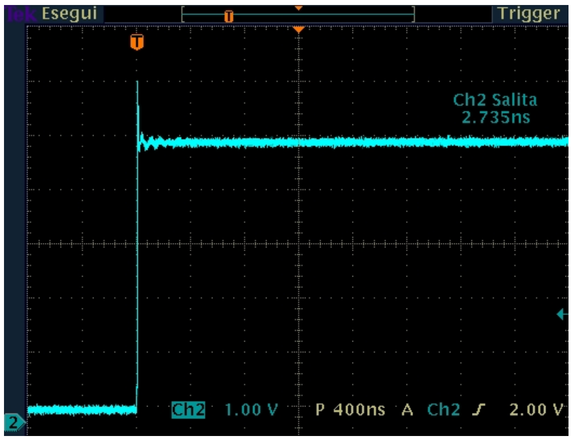
[5]Acronym for Intellectual Property.



(a) *Tekelec*'s **Epsilon Board OEM II** - Rise time.

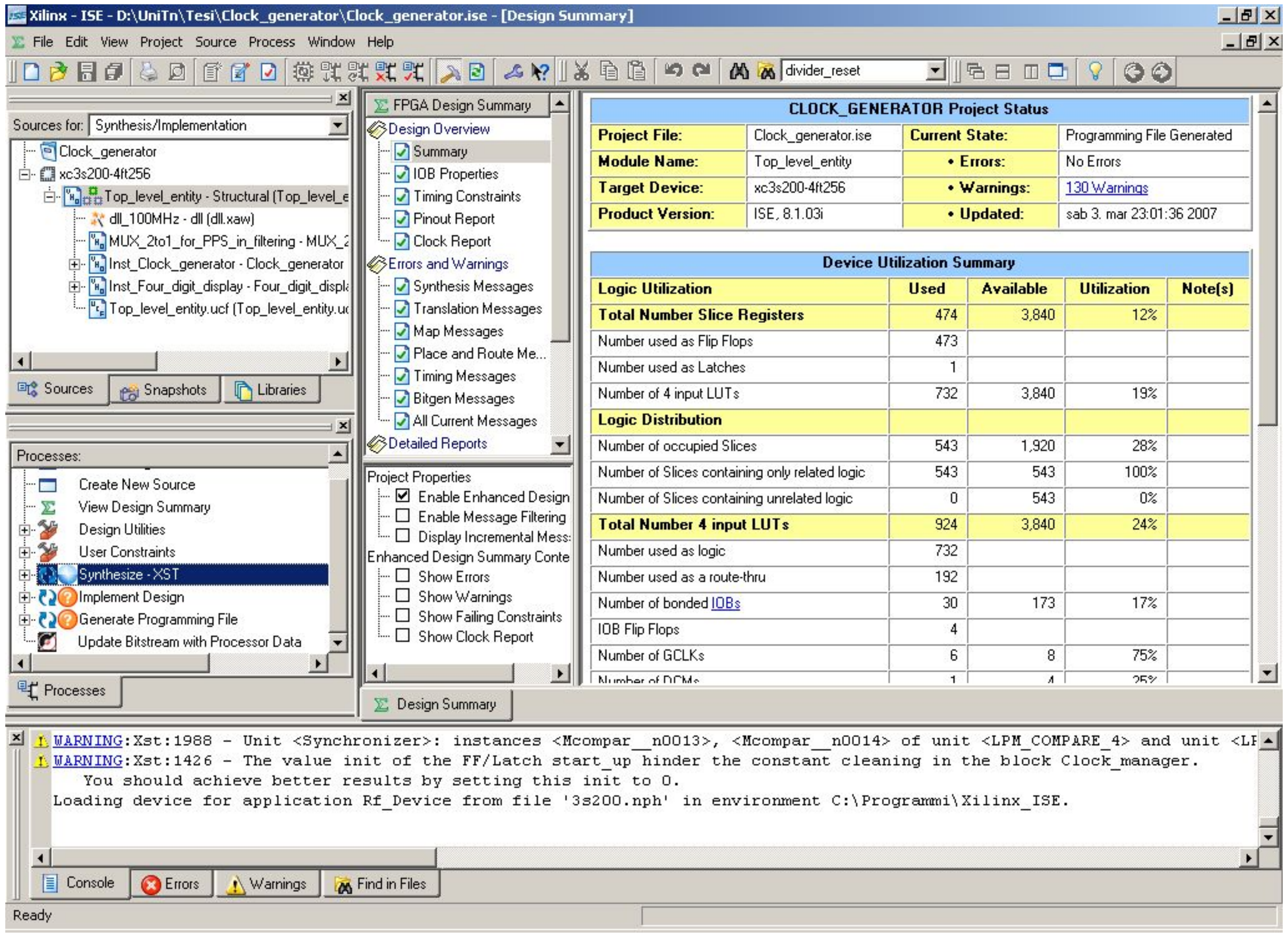


(b) *U-BLOX*'s **LEA-4T** - Rise time.



(c) *Trimble*'s **Resolution T** - Rise time.

Figure 2.6: Rise time comparison between the three receivers.



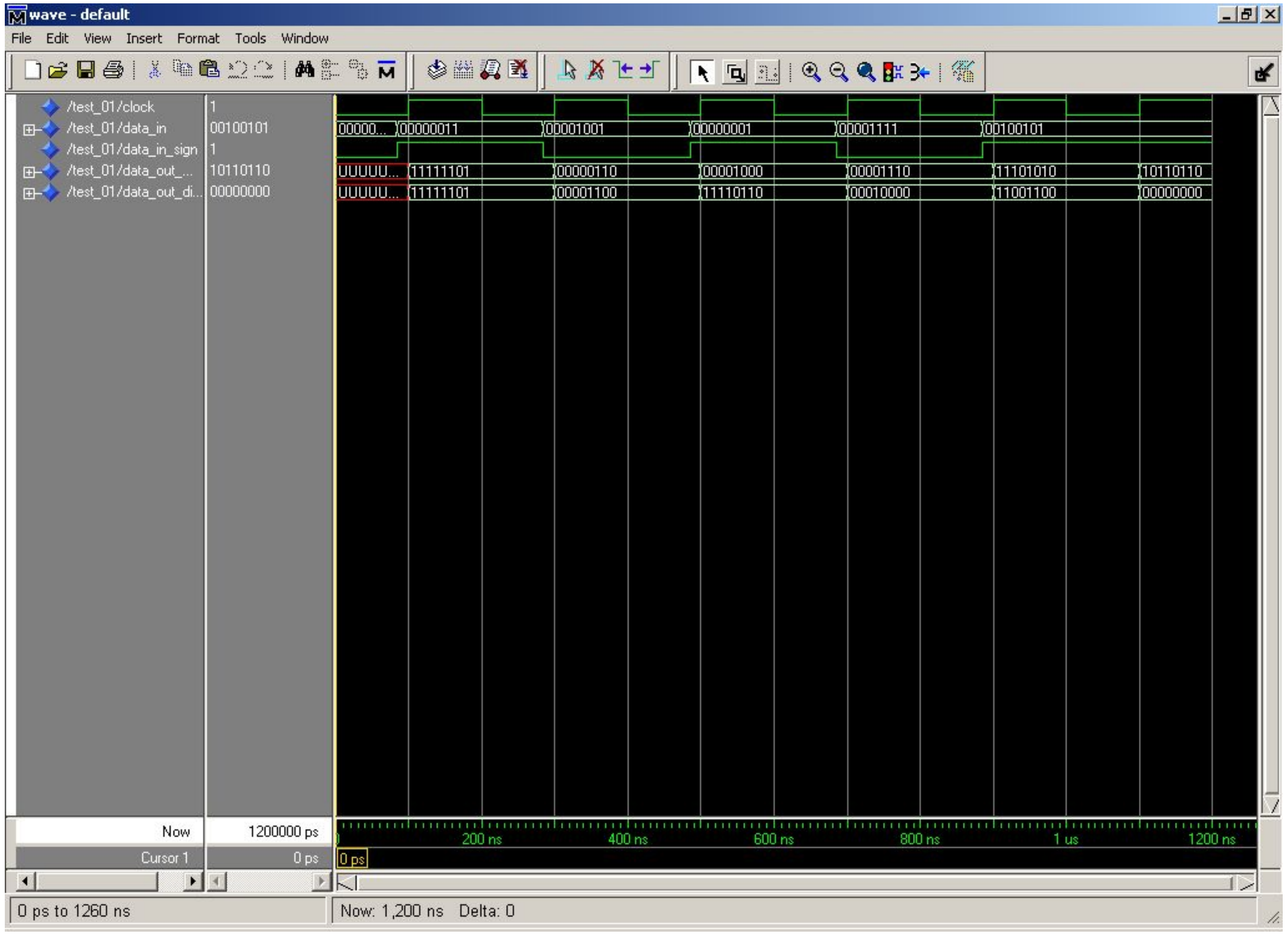
CLOCK_GENERATOR Project Status			
Project File:	Clock_generator.ise	**Current State:**	Programming File Generated
Module Name:	Top_level_entity	**• Errors:**	No Errors
Target Device:	xc3s200-4ft256	**• Warnings:**	130 Warnings
Product Version:	ISE, 8.1.03i	**• Updated:**	sab 3. mar 23:01:36 2007

Device Utilization Summary

Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	474	3,840	12%	
Number used as Flip Flops	473			
Number used as Latches	1			
Number of 4 input LUTs	732	3,840	19%	
Logic Distribution				
Number of occupied Slices	543	1,920	28%	
Number of Slices containing only related logic	543	543	100%	
Number of Slices containing unrelated logic	0	543	0%	
Total Number 4 input LUTs	924	3,840	24%	
Number used as logic	732			
Number used as a route-thru	192			
Number of bonded IOBs	30	173	17%	
IOB Flip Flops	4			
Number of GCLKs	6	8	75%	
Number of DCMs	1	4	25%	

```
WARNING:Xst:1988 - Unit <Synchronizer>: instances <Mcompar__n0013>, <Mcompar__n0014> of unit <LPM_COMPARE_4> and unit <LF
WARNING:Xst:1426 - The value init of the FF/Latch start_up hinder the constant cleaning in the block Clock_manager.
   You should achieve better results by setting this init to 0.
Loading device for application Rf_Device from file '3s200.nph' in environment C:\Programmi\Xilinx_ISE.
```

Figure 2.7: *Xilinx* **ISE 8.1** graphical interface.



```
/test_01/clock           1
/test_01/data_in         00100101      00000...  00000011   00001001   00000001   00001111   00100101
/test_01/data_in_sign    1
/test_01/data_out_...    10110110      UUUUU...  11111101   00000110   00001000   00001110   11101010   10110110
/test_01/data_out_di...  00000000      UUUUU...  11111101   00001100   11110110   00010000   11001100   00000000

Now       1200000 ps     200 ns   400 ns   600 ns   800 ns   1 us   1200 ns
Cursor 1  0 ps           0 ps

0 ps to 1260 ns          Now: 1,200 ns   Delta: 0
```

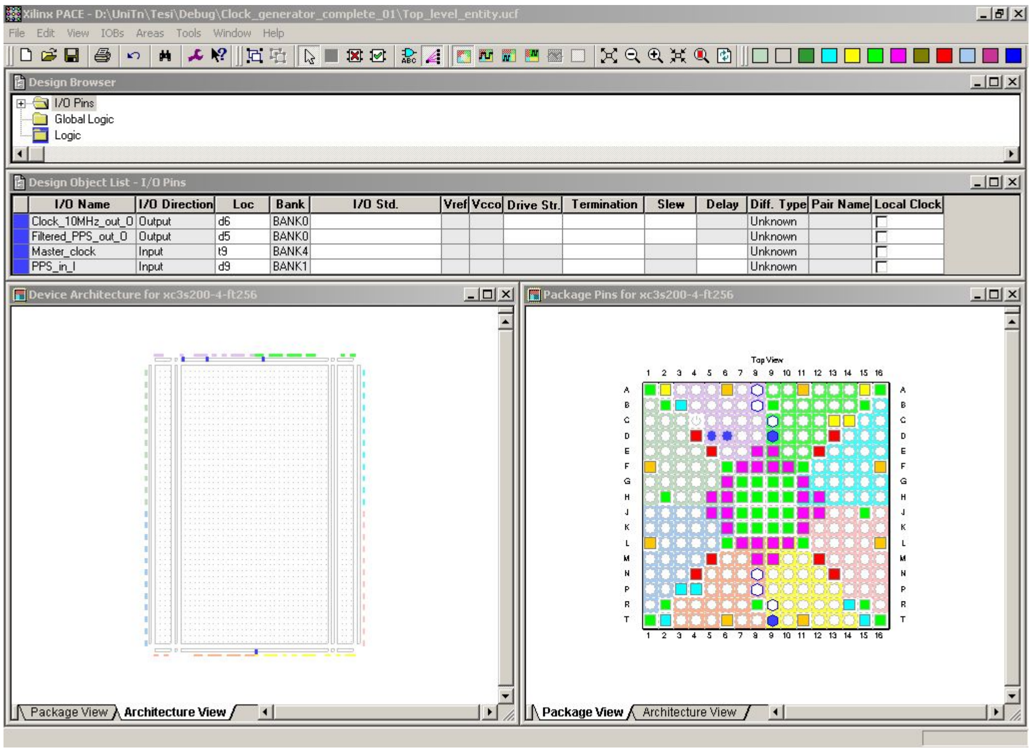
Figure 2.8: *Xilinx* **ModelSim** output example.

(IP) cores in the project. IP cores can range in complexity from simple arithmetic operators to complex system-level building blocks such as filters, transformers or memory.

In our case, the tool was used to create a Digital Clock Manager, a component that, among the offered services, can be used to increase the clock's frequency. In this way it is possible to limit the effects entailed by the oscillator frequency since it can be multiplied for whatever factor comprised in a wide set of values; in such a way, the problem related to the operating frequency is transfered to the code efficiency.

As a matter of fact, one of the big deals with FPGA is that an implemented algorithm can not work at whatever frequency, but there's a bound on the speed it can run at. This is due to gate and net delays introduced by the "compiled" code, which does not allow the signal to propagate correctly unless it respect the speed constraint imposed by the particular device the developer deals with. In this sphere, every time the top level entity is synthesized, **ISE 8.1i** provides a report of the elaboration in which, among the many informations, an estimate of the maximum frequency the designed entity can work at is provided, and which comes with a detailed description of the time needed by the signals in the different entities to propagate. In this way it is possible to understand where the code weaknesses are and as a consequence act to improve it already knowing the points that most require some optimization.

The so obtained code can then be simulated through the creation of *test bench* file, which can be realized with the ISE graphical tool that allows to modify the inputs in the shape of waveforms, or writing down directly the VHDL code defining the values assumed by input signals. This procedure brings to the creation of a file that has to be given in input to the **ModelSim**, whose task is to generate the code expected output. Of course, there is no guarantee about the provided results, since the tool does not take into account the effects introduced by the effective physical implementation of the designed entity, such as signal propagation problems, presence of asynchronous signals, fan out issues and many more; but at the same time the simulation allows to understand if the algorithm itself is developed correctly through the verification of the generated output correctness.


Xilinx PACE - D:\UniTn\Tesi\Debug\Clock_generator_complete_01\Top_level_entity.ucf
File Edit View IOBs Areas Tools Window Help
Design Browser
I/O Pins
Global Logic
Logic
Design Object List - I/O Pins
Device Architecture for xc3s200-4-ft256
Package Pins for xc3s200-4-ft256
Top View
Package View
Architecture View

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
Clock_10MHz_out_0	Output	d6	BANK0								Unknown		☐
Filtered_PPS_out_0	Output	d5	BANK0								Unknown		☐
Master_clock	Input	t9	BANK4								Unknown		☐
PPS_in_I	Input	d9	BANK1								Unknown		☐

Figure 2.9: *Xilinx* **PACE** graphical interface.

Once the **ModelSim**'s output corresponds to the desired one, it is possible to assign the input and output signals to the package pins, writing down the related ".ucf" extended file manually or using *ISE*'s *PACE* utility, whose interface is shown in figure 2.9, and create the programming file that is used to configure the FPGA, task that is still left to the **ISE**. After the generation of this file, it is possible to program the FPGA, task that is left to *ISE*'s *iMPACT* utility (shown in figures 2.10 and 2.10), via the JTAG port included in the *Digilent*'s **Spartan-3 Starter Board**, or providing the file with extension ".mcs" to the PROM which the FPGA included in *ComBlock*'s **COM-1300** is connected to.

2.4 Data acquisition related software

The assessment of the system accuracy is mainly based on the oscilloscope's delay measurements. In order to collect the acquisitions in some computer files and to

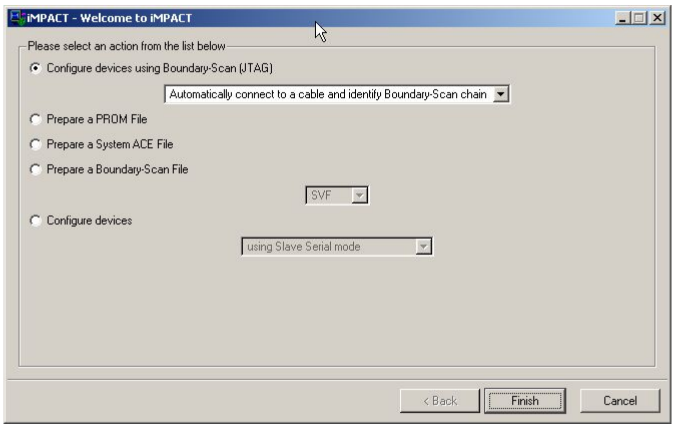
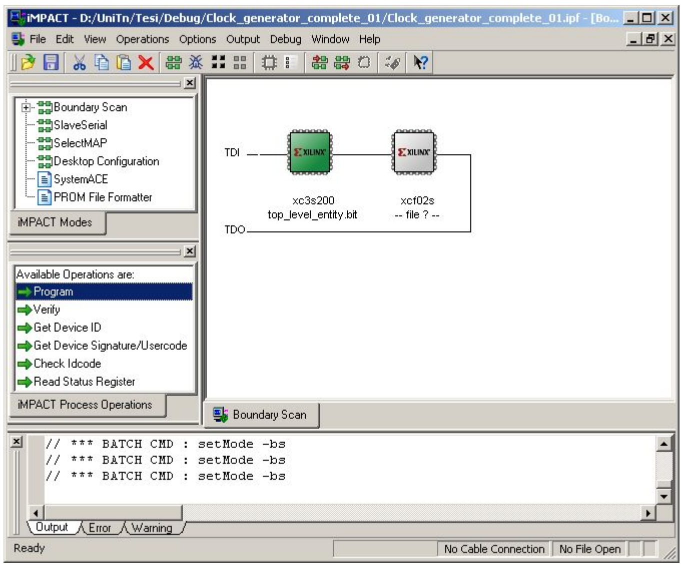
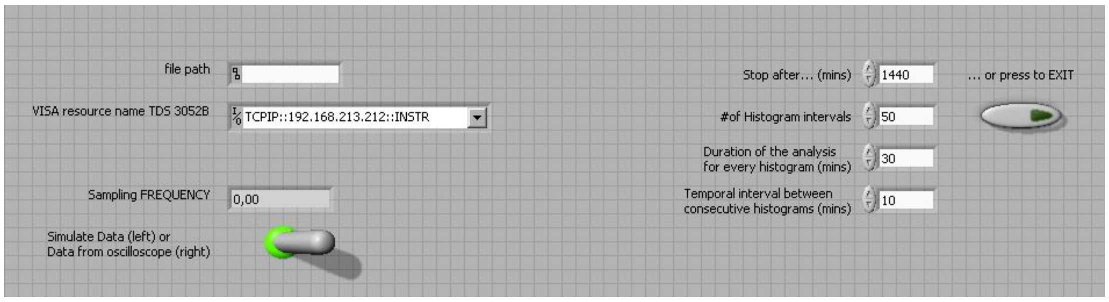

iMPACT - Welcome to iMPACT
Please select an action from the list below
Configure devices using Boundary-Scan (JTAG)
Automatically connect to a cable and identify Boundary-Scan chain
Prepare a PROM File
Prepare a System ACE File
Prepare a Boundary-Scan File
SVF
Configure devices
using Slave Serial mode
< Back
Finish
Cancel

Figure 2.10: *Xilinx* **iMPACT** - Programming type selection.



```
// *** BATCH CMD : setMode -bs
// *** BATCH CMD : setMode -bs
// *** BATCH CMD : setMode -bs
```

Figure 2.11: *Xilinx* **iMPACT** - JTAG programming window.



file path

VISA resource name TDS 3052B: TCPIP::192.168.213.212::INSTR

Sampling FREQUENCY: 0,00

Simulate Data (left) or Data from oscilloscope (right)

Stop after... (mins): 1440

... or press to EXIT

#of Histogram intervals: 50

Duration of the analysis for every histogram (mins): 30

Temporal interval between consecutive histograms (mins): 10

Figure 2.12: Oscilloscope's_data_acquisition.vi - Front panel: inputs.

carry out deep analysis on the generated output, a **LabVIEW** program called *"Oscilloscope's data acquisition.vi*[6]*"* was developed.

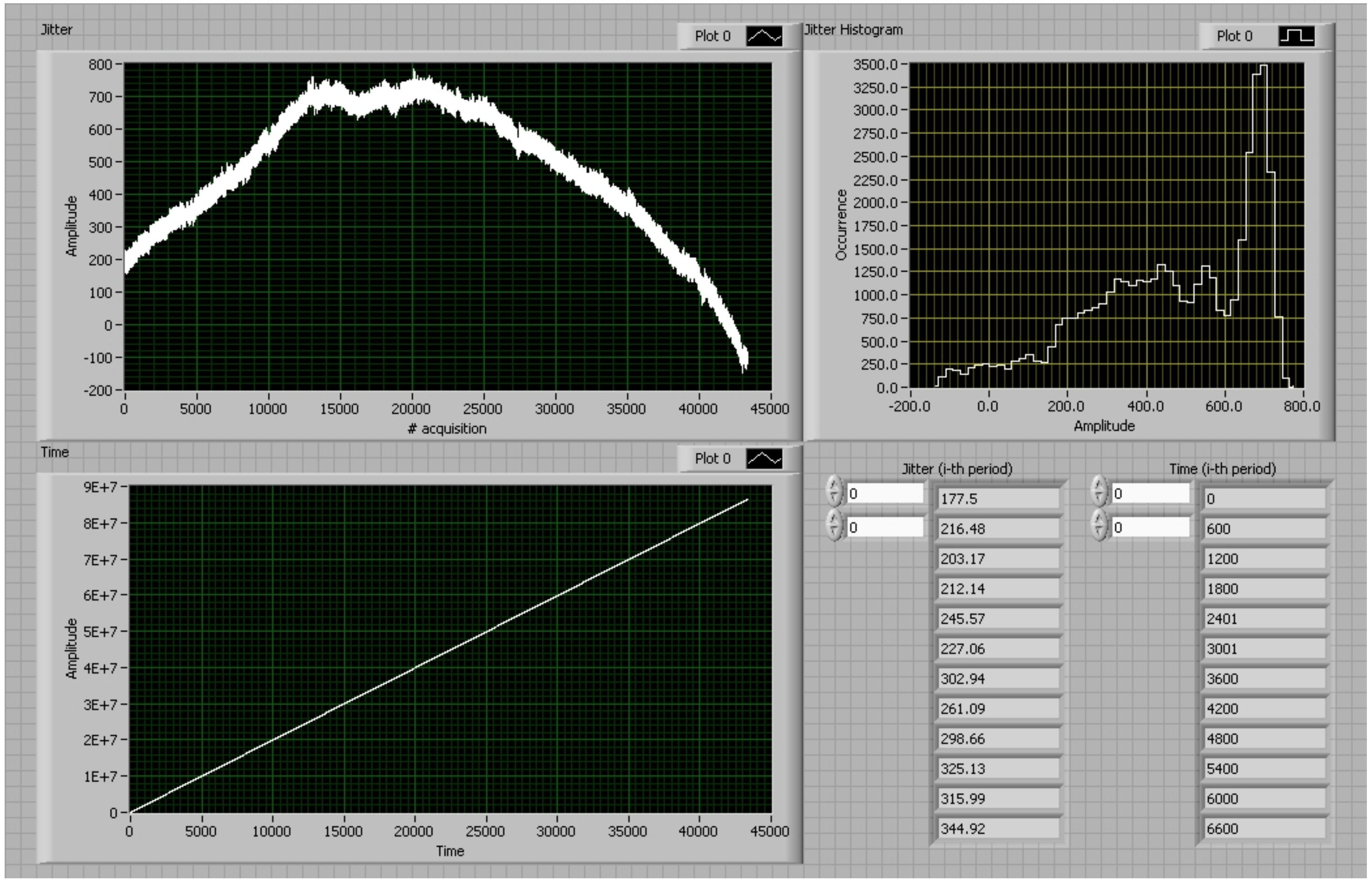
National Instrument's **LabVIEW** is a powerful graphical development environment, for signal acquisition, measurement analysis, and data presentation; it also provides the flexibility of a programming language without the complexity of traditional development tools.

The created VI features the front panel shown in figure 2.12 from which the user defines the output file in which the data will be saved, the duration of the whole acquisition process and, since the communication occurs over the internet, the IP address of the oscilloscope. Then, it provides real time graphs representing the time progress (in which it is possible to superimpose a polynomial curve of a given order that best represent the data) and the histogram of the gathered information (refer to figure 2.13); finally, after all the required measurements have been taken, a 3D time sliced histogram analogue to the one present in figure 2.14 can be analyzed, showing the evolution over time of the occurrences of the data in a temporal window whose width has to be defined by the user in the front panel.

The temporal window can be used in the following ways:

1. *sliding window*: its width remains the same and it moves forward with time, thus observing the samples from the i^{th} to the $(i+W)^{th}$, from the $(i+K)^{th}$ to the $(i+K+W)^{th}$, from the $(i+2K)^{th}$ to the $(i+2K+W)^{th}$, etc., where K stands for the time interval between two consecutive histograms and W

[6]the programs developed with LabVIEW are called Virtual Instruments, or, in a simpler way, VIs



Jitter (i-th period)	Time (i-th period)
177.5	0
216.48	600
203.17	1200
212.14	1800
245.57	2401
227.06	3001
302.94	3600
261.09	4200
298.66	4800
325.13	5400
315.99	6000
344.92	6600

Figure 2.13: Oscilloscope's_data_acquisition.vi - Front panel: output 1.

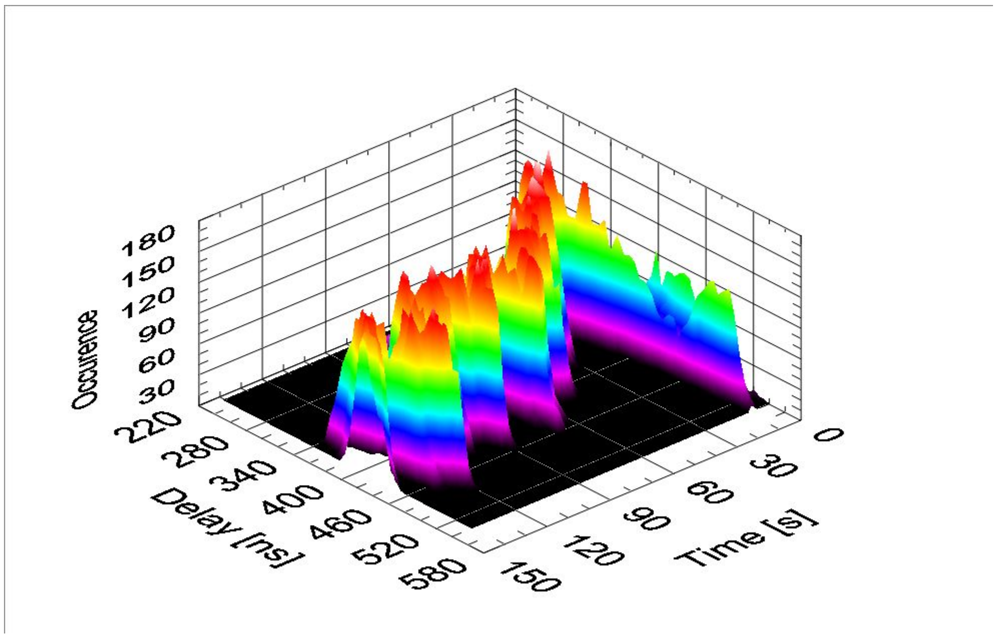


Figure 2.14: Oscilloscope's_data_acquisition.vi - Front panel: output 2.

defines the window width; in such a way it is possible to understand if the receiver's behavior changes during the day, e.g. if the receiver works better when the environment is warmer;

2. *enlarging window*: the starting point of the analysis is always the first acquisition, but the window width increases with time; the samples analyzed will then be the ones from the 1^{st} to the $(W)^{th}$, from the 1^{st} to the $(2W)^{th}$, from the 1^{st} to the $(3W)^{th}$, etc.; again, the period between two consecutive histograms is given by the parameter K (even though it does not appear in the formulas) while W now represents the window enlargement. This kind of study helps to understand if the histogram spreads with time.

The oscilloscope is controlled through a set of instructions and queries by a remote computer in which the program is running: in such a way it is possible to command the device automatically with a cycle in which every iteration starts with a request for a measure and finishes when the information is received and saved into the indicated file.

Another **LabVIEW** VI was then developed, called *"Data elaboration from file.vi"*, carrying out the same analysis of the previous one but the data is read from the saved files instead of being gathered from the oscilloscope. Moreover, in order to make the comparisons between different couples of signals easier, two input files are expected.

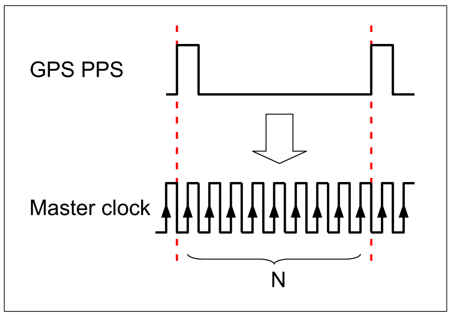
Another interesting and useful function used for the analysis is the *curve fitting*, which in our case was realized through polynomials of order dependent on the particular acquisition. In such a way it is possible to discriminate the effect of drifting with the one related to the jitter.

Chapter 3

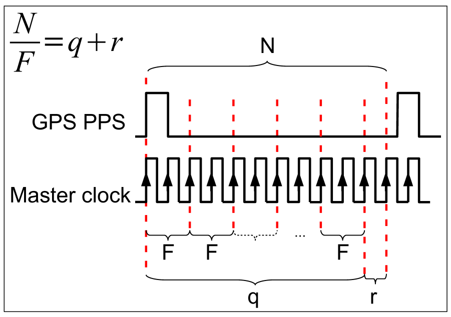
The principle of working

The idea that lays at the base of the whole system is really simple and can be summarized by the figure 3.1: from the GPS receiver the system obtains the basic timing reference from which it is possible to build the output: the high accuracy *1PPS* signal, whose precision we can't query. During the i^{th} cycle, delimited by the rising edge of two consecutive pulses, the clock internal to the device is used to measure the pulse length (so a counter is needed, counting the number of clock cycles present in a *1PPS* period) which should be close to the ratio between the internal clock's frequency and the reference signal's frequency. The obtained value is then divided by the ratio between the wanted output signal frequency (let it be a generic frequency $\boldsymbol{F}$) and the reference signal frequency (i.e. 1Hz) in order to get the output's signal length (measured in clock cycles). This value is used to build the $\boldsymbol{F}$[Hz] square wave during the $(i + 1)^{th}$ *1PPS* cycle, supposing the internal clock's frequency to be stable.

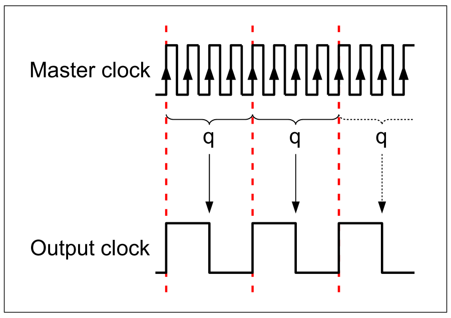
The stability assumption has to not to be taken for granted for long, so the procedure has to be repeated continuously, for every incoming pulse. In this way it is possible to to put bounds on the effect of the frequency drift affecting the quartz oscillator, an effect which is strongly correlated to instabilities including: aging, noise, temperature effects, warm up, acceleration effects, magnetic field effects, atmospheric pressure effects, radiation effects, and interactions among the various effects[20].



1. Count



2. Divide



3. Build

Figure 3.1: Basic algorithm.

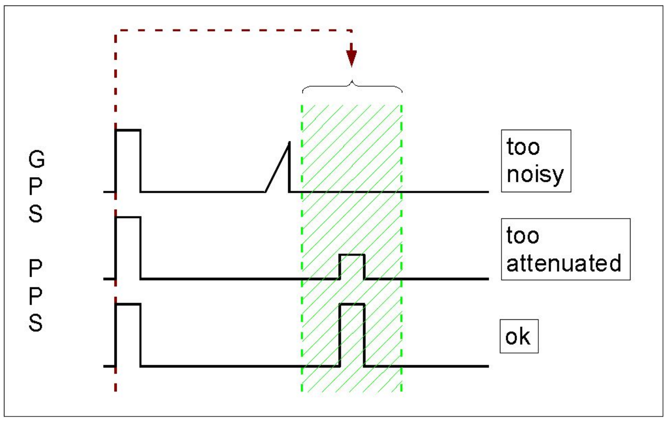


Figure 3.2: Temporal window.

3.1 Practical problems

When a developer gets in touch with reality, he/she often finds out that things are quite more complicated than the ideal case.

First of all it has to be considered the chance that the GPS receiver is not working properly, due to whatever faulty functioning (e.g. interference or bad antenna positioning), providing erroneous pulses or missing some of them. Thus, it is necessary to lay down limits in the shape of *temporal windows*, as shown in figure 3.2, during which correct pulses are supposed to happen. In case of pulses received out of these windows, or not received at all, the system has to be able to detect the failure and consequently act to generate the output omitting the faulty information gathered but storing the last supposed correct data as long as internal clock's stability can be considered uncorrupted (i.e. despite the information stored was correct, the generated output is wrong because the internal clock's frequency has changed, even if in a slight way; so, in the generation process, there's no connection between the information obtained during the i^{th} period and the output built in the $(i+n)^{th}$ period using the above-mentioned data which now is obsolete) or a new reliable input is provided.

Another aspect that needs to be considered for a proper functioning is the presence of the reminder in the division calculation. If the output is built without considering it, high bad synchronization risk has to be taken into account. Here's

Symbols	
F_{ref}	= reference signal frequency
F_{out}	= output signal frequency
F_{clock}	= theoretical internal clock frequency
L_{ref}^{ideal}	= theoretical reference signal length
L_{out}^{ideal}	= theoretical output signal length
L_{ref}^{real}	= real reference signal length
L_{out}^{real}	= real output signal length

Frequency ratio		
F_{ref}	$=$	$1Hz$
F_{out}	$=$	$100Hz$
	$\Downarrow$	
F_{out}/F_{ref}	$=$	100

Ideal behavior				
F_{clock}	$=$	$1KHz$		
	$\Downarrow$			
L_{ref}^{ideal}	$=$	F_{clock}	$=$	1000
L_{out}^{ideal}	$=$	$\frac{F_{clock}}{F_{out}/F_{ref}}$	$=$	10

Table 3.1: Algorithm: ideal behavior.

an example of how this could happen:

But since the division calculation returns an integer, it could happen that:

In this way, if the internal clock is stable, i.e. the result of the counting will not change in the following *1PPS* period, $999/9 = 111$ output square wave periods will be built, not 100. In other words, the output generation entails a distortion that could compromise the functioning of the system. This distortion, that can be seen in figure 3.3, is introduced by the presence of the reminder in the division calculation; however, the reminder can be recovered during the generation phase through the use of an accumulator module F_{out}/F_{ref}, which the reminder is added to at the end of every cycle built. Thus, the distortion can be compensated extending the square wave period for one additional internal clock cycle every time an accumulator overflow occurs, as shown in figure 3.4, where, for obvious space limitations, the frequencies have been changed as follows: the output signal frequency is set to 3Hz and theoretical internal clock frequency to 10Hz, while the reference signal

Possible real behavior		
L_{ref}^{real}	$\simeq$	F_{clock}
	$=$	999
	$\Downarrow$	
$\frac{L_{ref}^{real}}{F_{out}/F_{ref}}$	$=$	$9\ rem\ 99$
L_{out}^{real}	$=$	$\lfloor \frac{L_{ref}^{real}}{F_{out}/F_{ref}} \rfloor = 9$

Table 3.2: Algorithm: possible real behavior.

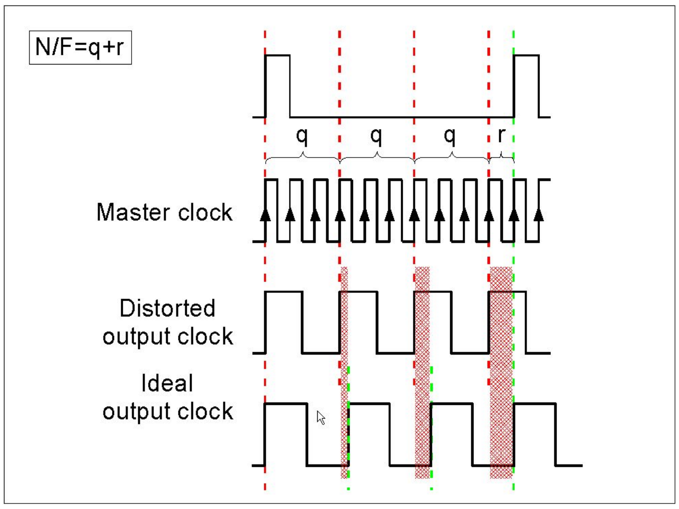


Figure 3.3: Reminder distortion.

frequency remains to 1Hz.

3.2 Parameters

After these considerations, it's possible to create a list of parameters that we need to provide for the correct working of the system:

- the ratio between the frequency of the output square wave (which in our case is 10MHz) and the frequency of the reference signal (which is 1Hz, but it could be any frequency; obviously, the higher it is, more often the data used to build the output is updated);

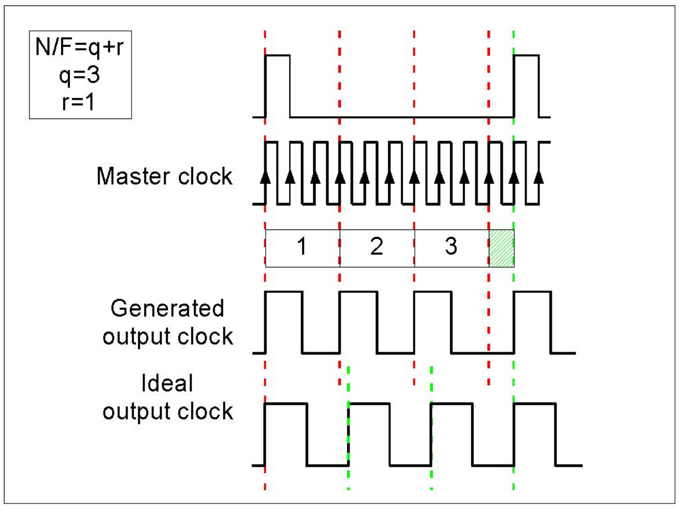


Figure 3.4: Reminder recovery.

- the theoretical reference signal length, i.e. the theoretical rate between the frequency of the internal clock and the frequency of the reference signal;
- a maximum value for the delta between the theoretical reference signal length and the real one.

The first parameter is used as divider in the calculation of the output's length, while the other two ones describe the *temporal window* used to check the input's correctness, defining the first it's center and the second it's width. Of course different strategies can be used to define the temporal window; for example the last valid PPS length measurement can be used as center, but the first option was implemented since a certain connection with the nominal frequency is desired.

Chapter 4

The implementation phase

The algorithm explained in chapter 3 was then translated in VHDL language. Usually the best approach for the realization of a whole hardware system, whose aspects are not considered in their totality yet because of the intrusion of unexpected problems, is to implement it in a modular way, connecting different components, having each of them a specific task to fulfill. In this manner, since a component sees only its own input signals without the need to know how they are managed, it is possible to modify a single component without caring about the others, as they perform correctly their own duties.

In order to understand if the algorithm is working properly or not, a whole system made of the GPS receiver and the clock generator needs to be simulated. At this point the *Digilent*'s **Spartan-3 Starter Board** offers much more useful instruments than the *ComBlock*'s **COM-1300**. The performance of the system can be assessed through the help of a function generator providing the *1PPS* signal, then buttons, switches and LEDs of the starter board can be used to control and monitor internal signals, a display to show the content of the counters (with the help of a switch to change the bytes displayed in case the digits are not enough) and an oscilloscope to depict the outputs. The functioning of all the components needs to be checked, so many steps are required to complete this phase. Of course, the fact that the system works properly using artificially generated signals does not directly imply that it will work with the real ones, since many factors of uncertainty are introduced by the external environment and by the devices themselves.

4.1 The VHDL components

As a consequence of all the considerations given in chapter 3, it is possible to define in a really clear way which are the main tasks that have to be fulfilled. Now, they will be described.

4.1.1 Reset generation

Every time a pulse is received, the system needs to be reset, i.e. the various counters need to be reset to zero after their last value had been memorized (if needed). The fact that this pulse is $100 \mu s$ long entails the fact that the signal supplied by the GPS receiver has to be filtered somehow, making the pulse shorter (not longer than a cycle of the internal clock), in order to allow the other components to correctly measure its period and properly build the output. The component this task is entrusted to is the **Reset generator**, initially realized through a *Mealy machine*[1] receiving in input the *1PPS* signal and a clock, and providing as output the reset signal for the other components of the system. In this way the reset signal is kept asynchronous, so that there's no loss of time between the instant the GPS pulse arrives and the instant the entities are reset.

Its functioning is really simple:

1. *Reset out* is set to "1" when *Reset in* arrives;
2. *Reset out* is reset to "0" at the immediately next rising edge of the *Master clock* (it remains high for a period shorter than or equal to a period of *Master clock*);

where *Reset in* is the signal provided by the GPS receiver and *Reset out* is the shortened pulse.

The debug of this entity has been realized with the help of the function generator, simulating the GPS signal, and the oscilloscope, monitoring the *1PPS* signal and the **Reset generator** output. The generated reset has to rise with the pulse and fall in less than a *Master Clock*'s period.

[1]A finite state machine which produces an output for each transition, i.e. the output generated by a Mealy Machine depends on both its state and its inputs.

4.1.2 *1PPS* signal period measurement and division calculation

The system algorithm is based on the measurement of the *1PPS* signal period and its result's division to obtain the length of the output cycles. As said in chapter 3, this is achieved by counting the number of rising edges of the *Master Clock* that happen between every two following pulses (considering their rising edges as temporal bounds for the counting).

When a pulse arrives, the consequent reset signal determines the saving of the counter before its value is canceled to allow the system to acquire the following input period length measurement. Since the informations needed to build the output have to be ready right after the input cycle is finished, the division is performed contemporaneously with the *1PPS* signal period measurement through the use of other two counters, one incremented at every rising edge of the *Master Clock* and counting from 0 to the value of the frequency ratio between the output square wave and the reference signal, the other incremented every time an overflow of the previous said counter occurs.

Here's an example of how the signals work, considering *init* and *end* as the the instants when a pulse and the following one respectively arrive. Being the pulse asynchronous with the clock, a small delay is considered between the pulse arrival and the first rising edge of the clock.

The values contained in the registers at the end, i.e. when the second pulse arrives, are then memorized in other registers allowing the construction of the output.

The debug of the divider can be realized with the help of the display for checking the result of the calculation (saved in the **Clock lengths memory**). In order to understand if the operation is computed correctly a low frequency clock signal is desirable to simplify the measurements. Through the function generator it is possible to set the period of the input signal (so the result has to change), allowing to check if the algorithm works correctly in a dynamic way. Some results are reported in table 4.2.

Symbols		
F_{ref}	=	reference signal frequency
F_{out}	=	output signal frequency
F_{clock}	=	theoretical internal clock frequency
Dividend	=	*Master clock* counter, reset every time a pulse arrives;
DIVISOR	=	frequency ratio between the output square wave and the reference signal;
Reminder	=	*Master clock*-module *DIVISOR* counter;
Quotient	=	overflow counter of the *Master clock*-module *DIVISOR* counter.

Parameters		
F_{ref}	=	$1Hz$
F_{out}	=	$3Hz$
F_{clock}	=	$10Hz$
	$\Downarrow$	
$DIVIDEND$	=	3

Behavior							
Registers	*Master clock*'s rising edges						
	init	01	02	03	04	05	...
Dividend	0	1	2	3	4	5	...
Reminder	0	1	2	0	1	2	...
Quotient	0	0	0	1	1	1	...
	000	05	15	25	35	45	...
	Time [ms]						

Behavior							
Master clock's rising edges							Registers
...	06	07	08	09	10	end	
...	6	7	8	9	10	**10**	*Dividend*
...	0	1	2	0	1	**1**	*Reminder*
...	2	2	2	3	3	**3**	*Quotient*
...	55	65	75	85	95	1000	
Time [ms]							

Table 4.1: Divider's functioning.

Clock	1PPS	d	D	Q	r	D	Q	r
Frequency [Hz]			Ideal			Experimental		
100	1,3	10	76	7	6	75/76	7	5/6
100	1,2	10	83	8	3	82/83	8	2/3
100	1,2	10	90	9	0	89/90	8/9	9/0
100	1,0	10	100	10	0	99	9	9
100	0,9	10	111	11	1	110/111	11	0/1
100	0,8	10	125	12	5	124	12	4
100	0,7	10	142	14	2	141/142	14	1/2
100	0,6	10	166	16	6	165/166	16	5/6
100	0,5	10	200	20	0	199	19	9
100	0,4	10	250	25	0	249	24	9
100	0,3	10	333	33	3	332/333	33	2/3
100	0,2	10	500	50	0	499	49	9
100	0,1	10	1000	100	0	999	99	9

Table 4.2: Divider's results. D/d=Q+r.

4.1.3 *1PPS* signal period correctness check

As said before in chapter 3, it could be possible that the receiver is not working properly, thus some pulses must not to be considered in the measurement or some others are lost causing the counting to be unconnected to reality. It is then necessary to check in some way the correctness of the *1PPS* pulse received. This is done simply allowing the saving of values provided by the **Divider** only inside *temporal windows* which define the correctness of the pulses.

This is achieved through the combination of a temporal bounds manager, called **Clock length bounds memory**, which defines the temporal window, and of a **Synchronizer** of the counts and the saves with the input signal. The **Clock length bounds memory** supplies as output the minimum and the maximum values that the main *Master clock* counter of the **Divider** can assume in order to be considered valid and let it be saved. It can work in two different modes, called *static bounds mode* or *dynamic bounds mode*. The former indicates that the temporal bounds are decided prior to FPGA programming and are never changed, the latter that they are set during the algorithm execution depending on the last *1PPS* correct period, creating a window centered in the last measured value and with a Delta equal to $1/2^N$ of the window center, where N is fixed during the development phase. Obviously, at start-up, a default value has to be used.

The **Synchronizer** receives as input the temporal window, the actual value of the main *Master clock* counter and the reset signal which indicates that a *1PPS* pulse have just been received. In order to avoid problems caused by possible counter overflows, it disables the increment of the counters when the main one exceeds the upper temporal bound through a flag supplied as output; it also enables counter saving when a correct pulse is received and it determines whether the connection with the satellite works properly or not,i.e. if an erroneous pulse is received, or in case of a missing one, it waits for a minimum pre-defined number of consecutive correct *1PPS* pulses before updating the last valid measurement acquired.

The measure of the *1PPS* period needs to be available for the output generation. This task is assigned to a memory element, called **Clock lengths memory**, which saves the given input data on the rising edge of its clock. The clock of this component is not the *Master clock* of the system, but it is obtained by the reset provided by the **Reset generator** activated through the enabling signal provided by the **Synchronizer**.

The **Divider**, the **Clock length bounds memory**, the **Synchronizer** and the **Clock lengths memory** are then grouped into a unique component called **Clock manager**.

Its tasks are:

- to provide the length of a cycle of the *1PPS* period signal checking its validity through the use of a temporal window which a *1PPS* pulse has to happen into in order to be considered valid;

- to provide the length of a cycle of the output signal performing the division between the number of cycles included in a *1PPS* cycle and the ratio wanted;

- to enable the generation of internal clocks when the system locks to the satellite for the first time.

  The debug of these components can be verified using a button to simulate external noise or stopping the the input signal supply, fact that has to lead to ignore the corrupted acquired data.

4.1.4 Output generation

Once valid data have been stored, the outputs generation is enabled based on the lengths provided by the **Clock manager**. The task is assigned to the **Clock builders**.

Two outputs are built: the first one is the internal *1PPS* signal, whose length simply corresponds to the last measured period of the input signal, the other one is the one at higher frequency, that is build using the information obtained by the division calculation.

Both the **Clock builders** receive as inputs an enabling signal allowing the output construction only when the information held in the internal registers is reliable, a resetting signal defining the initial instant of the generation process and the output's length (measured in *Master clock*'s cycles) giving information about the output's period. The higher frequency signal builder gets also the reminder of the division calculation. In fact its presence suggests, during the generation phase, the use of the fraction recovery procedure explained in chapter 3 in order to achieve a better result. In other words, the reminder represents the fraction of *Master clock* lost while carrying out the integer division, which introduces a distortion in the output. This can be avoided accumulating the reminder in a base-(F_{out}/F_{ref}) counter during the generation process and perpetuating for an additional *Master clock*'s cycle the output's period every time an overflow occurs, i.e. every time the accumulated fractions reach the value of a unit. Two other control signals are then used: the first one to allow the generation of both pulse-like signals and signals with uniform duty-cycle, the second one to set a maximum number of generated output cycles since the last reset has happened. In this way, it is possible to stop the construction of the internal 1PPS in case the signal from the satellite is lost and to keep the other output, providing a high accuracy high frequency clock, low once the desired number of cycles are built, avoiding the construction of spurious periods due to the remaining unrecovered accumulated fractions of the *Master clock* or to sudden oscillator frequency change.

The correctness of this component can be easily checked with the help of the oscilloscope, capable of measuring the output frequency. Moreover, it is possible to monitor the behavior of the higher frequency output right before the input *1PPS* rises, so that the reminder recovery procedure and the avoidance of spurious periods construction can be verified , and compare the generated frequency with a known correctly working signal using the universal counter.

4.2 Test phase and corrections

The third part of the the development of the thesis project, after the implementation and debug phases, was constituted by the testing of the VHDL code with real, asynchronous and noise affected inputs.

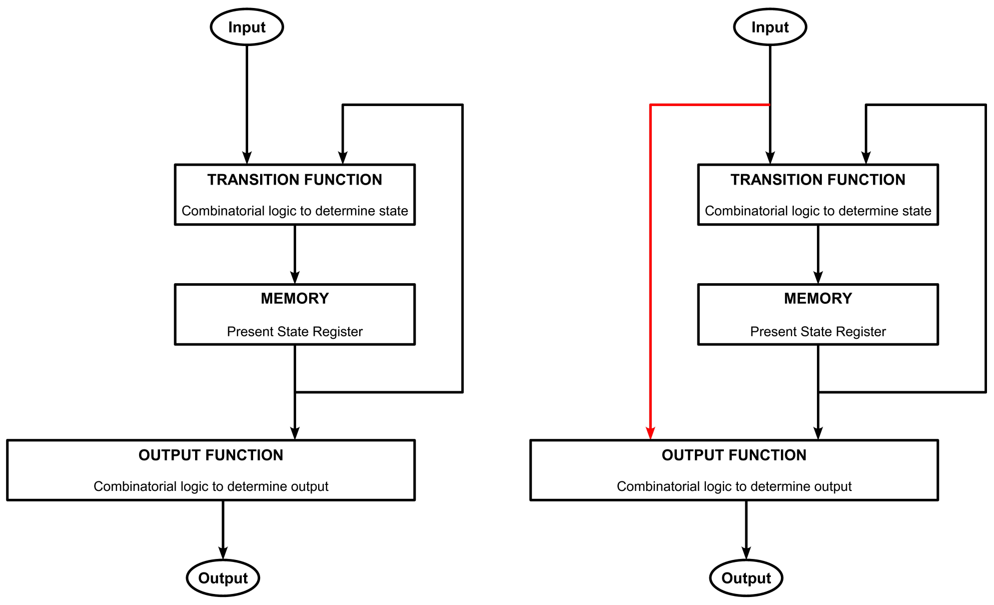
4.2.1 The asynchronous input issue

At the very beginning, the high frequency output behavior was verified setting the following configuration: the high frequency output clock of the FPGA was compared with the one coming from *Tekelec*'s **Epsilon Board II** using its *1PPS* signal as input to the **Spartan-3**, in order to eliminate possible uncertainties introduced by the use of different receivers. Unfortunately, the fact that the signal used to reset the system, i.e. the *1PPS* signal supplied by the GPS receiver, is asynchronous with the clock led to problems: once every some minutes the reset could not be seen by all the components, entailing unacceptable “holes” in the generated output; therefore, an input signal sampler was included, despite this operation unavoidably introduced a first element of loss of precision, which intensity is inversely proportional to the internal clock frequency. In practice, the *Mealy machine* was replaced by a *Moore machine*[2] and the problem was resolved. The price to pay for the introduction of the sampler is a loss of resolution equal to the *master clock* frequency, fact that, in any case, does not create problems since the accuracy requirements are quite relaxed.

4.2.2 The input rise time issue

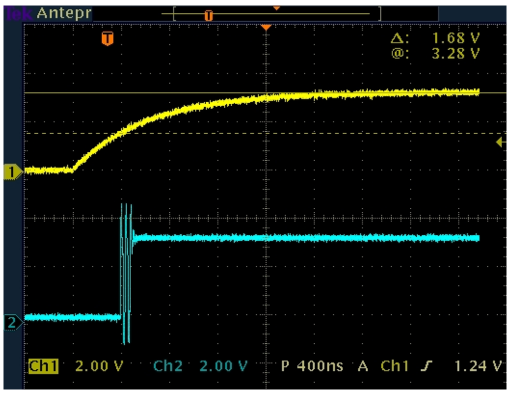
Then, the test was carried on replacing the input to the FPGA: *Tekelec*'s **Epsilon Board II** was replaced by *U-Blox*'s **LEA-4T**, whose rise time is really high, as

[2]In the theory of computation, a Moore machine is a finite state machine in which the outputs are determined by the current state alone and do not depend directly on the inputs.

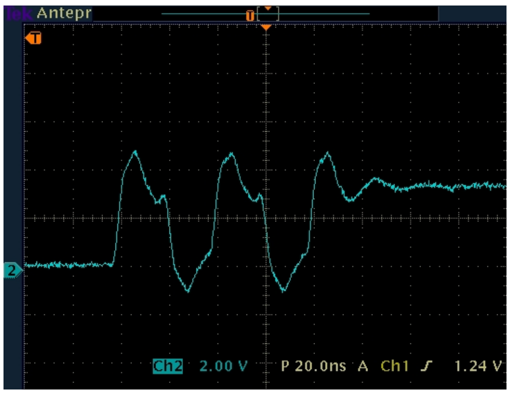

Input
TRANSITION FUNCTION
Combinatorial logic to determine state
MEMORY
Present State Register
OUTPUT FUNCTION
Combinatorial logic to determine output
Output
Input
TRANSITION FUNCTION
Combinatorial logic to determine state
MEMORY
Present State Register
OUTPUT FUNCTION
Combinatorial logic to determine output
Output

(a) Moore machine (b) Mealy machine

Figure 4.1: Moore vs. Mealy machine.



(a) Pure *1PPS* vs. Sampled *1PPS* signals.



(b) Sampled *1PPS* signal: detail.

Figure 4.2: Pure *1PPS* vs. Sampled *1PPS* (enlarged detail on the right).

can be seen in figure 2.6. The added noise does not allow a correct sampling of the *1PPS* input signal, because it determines the output fluctuations presented in figure 4.2 due to unwanted threshold crossing. Moreover, this effect is amplified when the operating frequency is boosted. The sampler was then changed into a more complex *Finite State Machine (FSM)* implementing a sort of *hysteresis*[3] cycle: it constitutes a sampler that, when it detects a threshold transition, interrupts the sampling operation for some time (i.e. some nanoseconds) waiting for the transient to be over. The duration of the needed suspension period depends on the rise time and on the noise amplitude. Since this amount of time has to be translated for the FPGA into a number of cycles, the operating frequency is another parameter that has to be taken into consideration.

4.2.3 Jitter and drift

After that, the system was tested comparing the *1PPS* signals and the higher frequency ones provided on the one hand by *Tekelec*'s **Epsilon Board II** and on the other hand by the FPGA. Again, the response was not as good as desired, even though the accuracy is already sufficient for allowing the implementation of the device into the *Fractional* λ switch prototype: in fact, signal jumps, offset and

[3]Hysteresis is a property of systems that do not instantly follow the forces applied to them, but react slowly, or do not return completely to their original state: that is, systems whose states depend on their immediate history, not only on the present stimulus

jitter seemed to be affecting one of the two signals. Unfortunately, no atomic clock was available for high precision acquisitions; hence, in order to understand which of the receiver was the source of the problems, the third GPS receiver, *Trimble*'s *Resolution T*, was used to perform cross-analysis of the three devices.

The acquisitions were realized by the oscilloscope measuring the delays between the three receivers, analyzed separately in pairs. The acquired data were transmitted to a PC and collected into files for further processing with the *LabVIEW* software described in chapter 2. In figures 4.3, 4.4 and 4.5 the delay vs. time graphs are reported, showing also the lowest order polynomial curves that achieved good fitting at first glance. It seems that the signal jumps and the offset are mostly connected to some filtering that takes place in the **Epsilon Board II**, while the jitter affects mainly the **Ublox**'s and **Trimble**'s receivers.

Moreover, since the measurement carried out in the i^{th} cycle is used to build the output in the $(i+1)^{th}$ cycle, the effect of the jitter is doubled: let's assume for example that the implemented VDHL code is running at 50MHz (the counting resolution is therefore $20ns$) and that the jitter affecting the received *UTC 1PPS* at the i^{th} cycle is $+45ns$ over the ideal pulse length; thus, the counter will experience a $+2$ ticks count. Then suppose that in the following second, the system misses 2 ticks because of a $-45ns$ overlapping noise. The overall count will then be affected by a -4 ticks error (-2 due to the late arrival of the first pulse which resets the counter, and -2 due to the early arrival of the second pulse), that will cause a $-80ns$ difference over the ideal case, entailing the generation of a distorted output.

This fact introduces the need of some kind of filtering on the signal provided by the receiver, since the precision requested by the entire switching system does not allow such a distortion in the generated output. Since this operation turned out to be quite complicated, the whole section 4.3 is dedicated to the implementation of the filter.

4.3 Input signal filtering

Once the test phase revealed that the main problem can be identified with the jitter affecting the signal, the project evolved focusing on the reconstruction of the *1PPS* signal with the purpose of eliminating or at least reducing the effect of the noise

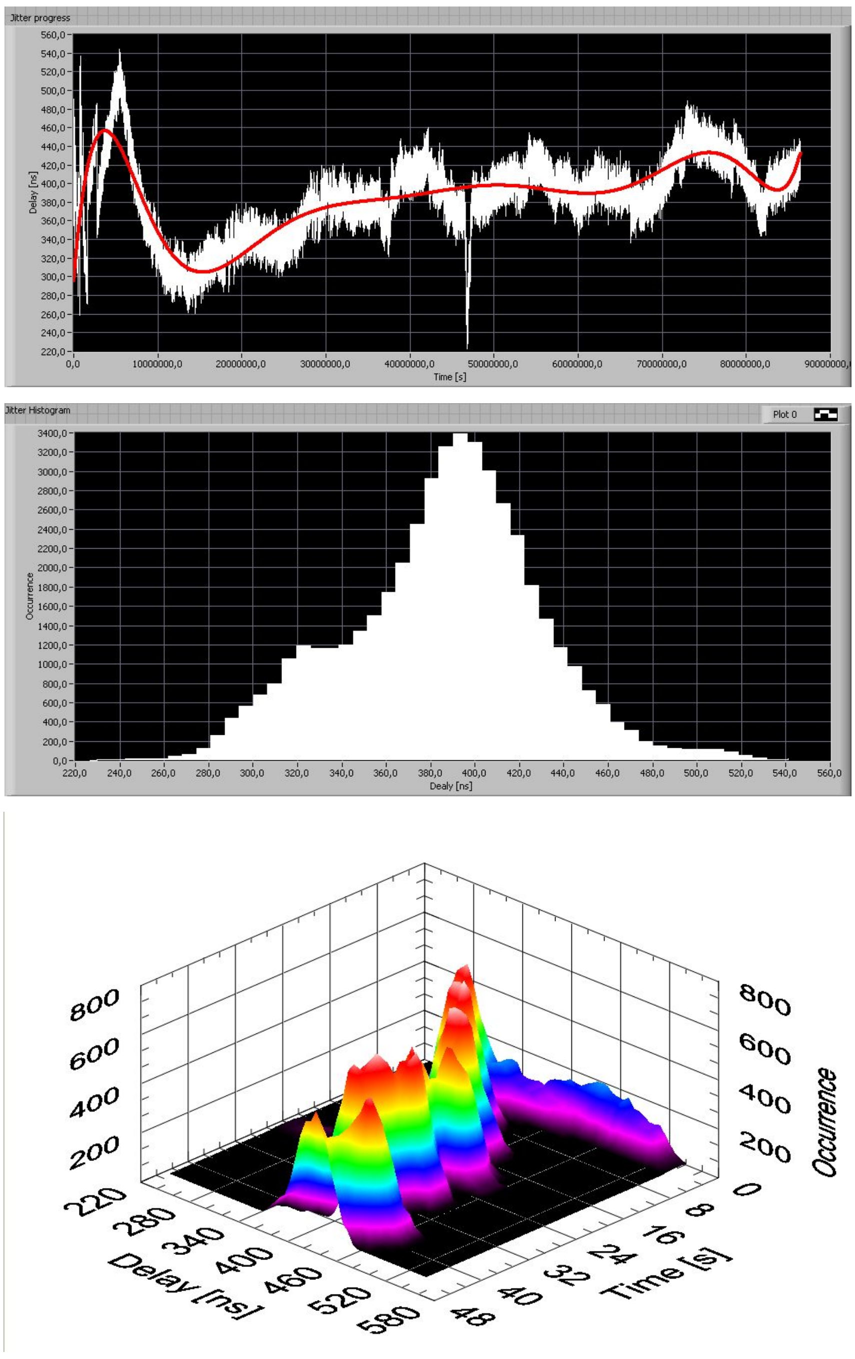


Figure 4.3: *Tekelec*'s **Epsilon Board II** vs *U-Blox*'s **LEA-4T**: delay time progress, histogram and time sliced histogram using a sliding window.

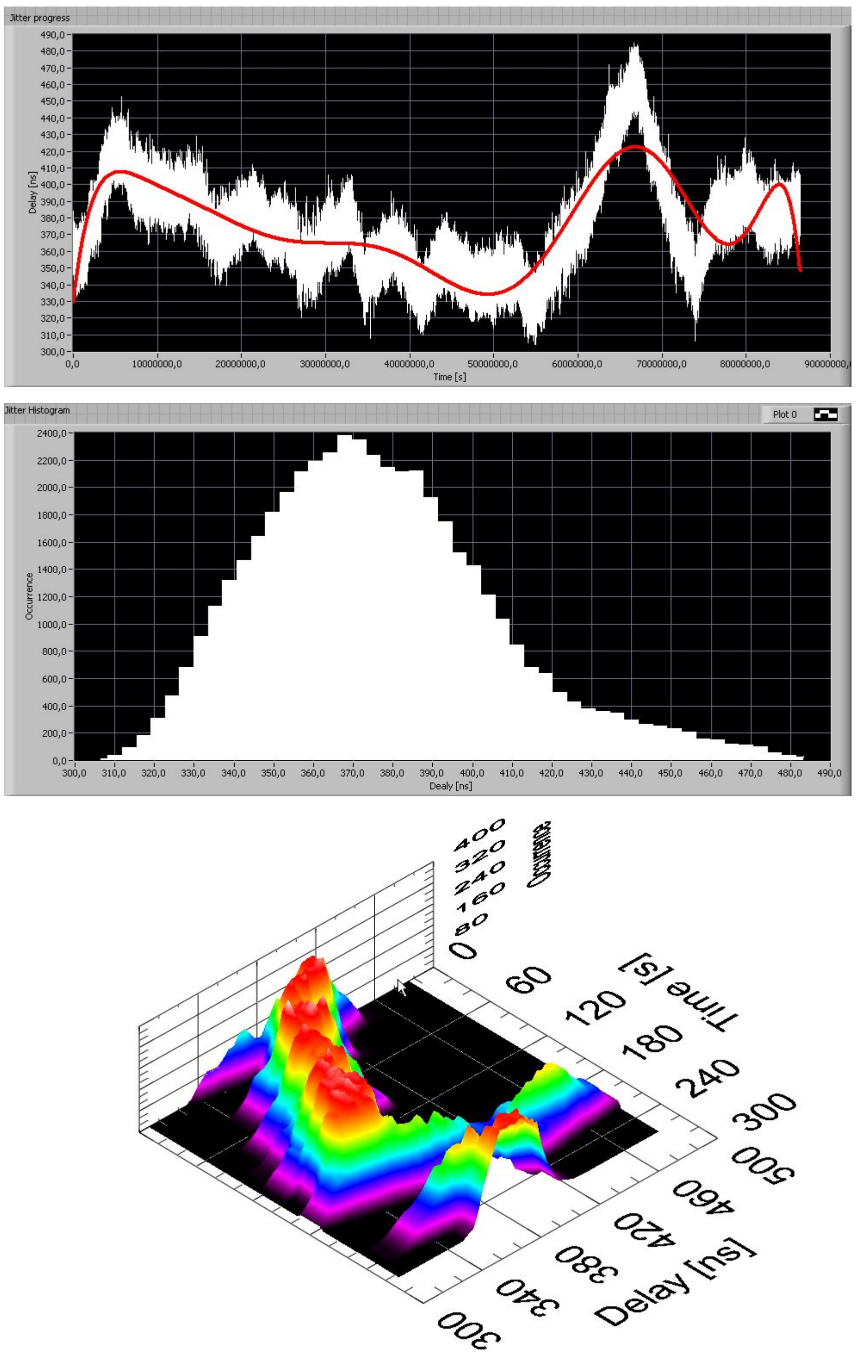


Figure 4.4: *Tekelec*'s **Epsilon Board II** vs *Trimble*'s **Resolution T**: delay time progress, histogram and time sliced histogram using a sliding window.

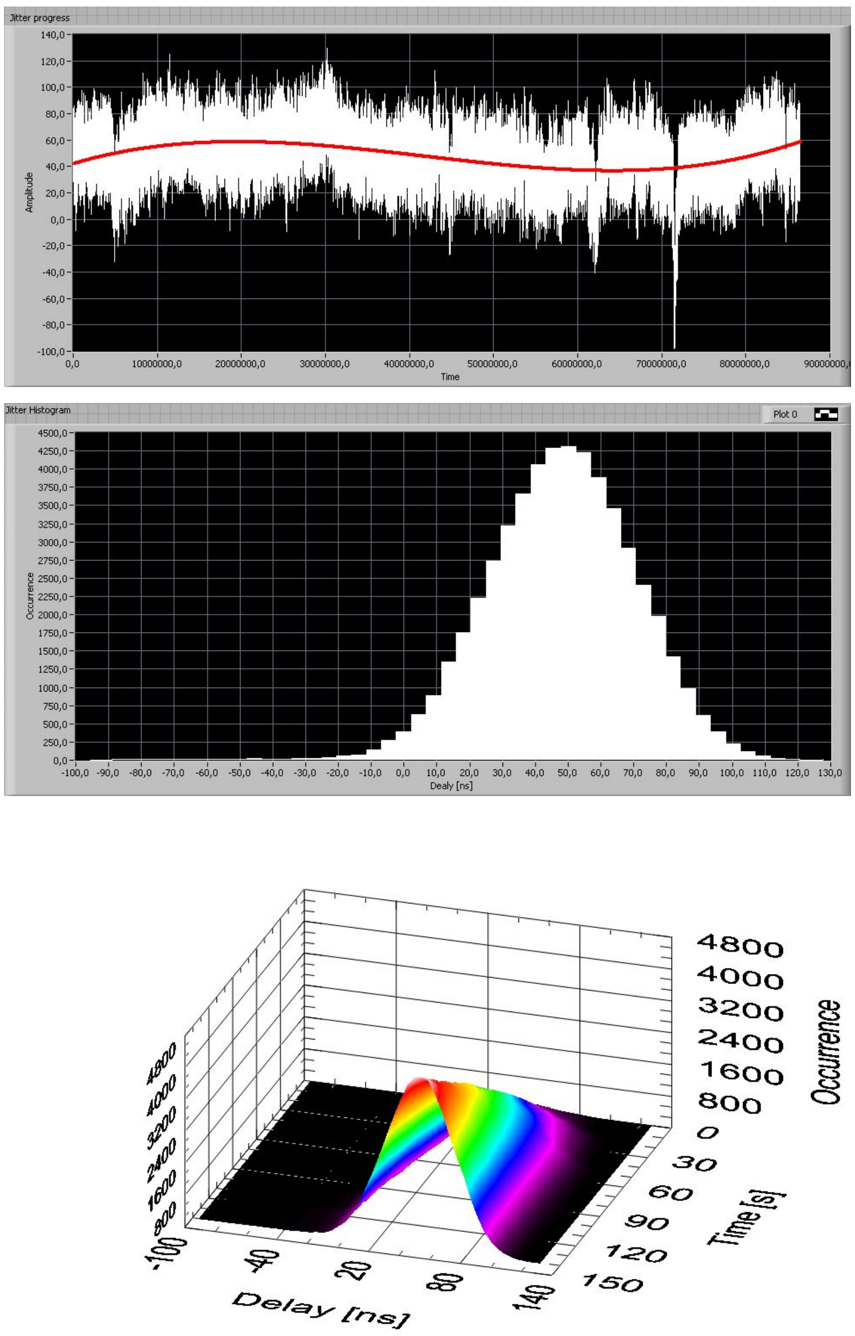


Figure 4.5: *U-Blox*'s **LEA-4T** vs *Trimble*'s **Resolution T**: delay time progress, histogram and time sliced histogram using an enlarging window.

introduced by the physical environment, i.e. temperature and antenna positioning, and by the data processing internal to the receiver. The way this could be done was obviously recognized with some filtering operation on the signal provided by the GPS receiver.

The issue is quite complicated: our sub-system is characterized by two independent clocks: on the one hand the signal derived from the satellites by the GPS receiver, whose purpose is to provide a precise reference used to eliminate the inherent low frequency instability of the quartz oscillator connected to the FPGA, and on the other hand the clock internal to the FPGA-based GPS board, which purpose is to eliminate the high frequency noise that overlaps the *1PPS* signal obtained by the satellites. The filter has to be designed in such a way that the filtering at a certain frequency id delegated to the correct clock.

4.4 FIR filter

4.4.1 FIR filter

In order to understand in a clearer way which kind of operation is needed, another acquisition has been taken with the following configuration: *Tekelec*'s **Epsilon Board II** was allowed to lock for 24 hours, in order to achieve an accurate *1PPS* signal. Then, the antenna was disconnected from the receiver to avoid further adjustments obtaining a stable signal, i.e. low jittered and not jumping, that was compared with *U-BLOX*'s **LEA-4T** output. The histogram over a 24 hours analysis revealed a gaussian distribution; assuming the jitter not to have a line spectral density (power per frequency interval) which means that it does not behave in any particular way in the frequency domain (in which case a specific band pass filter could be used), a low pass filter was firstly considered. But since the filtering operation has to be executed continuously, for the reasons explained in chapter 3, a sort of sliding window is needed; for this reason, instead of a simple averaging filter, a *FIR filter* was designed, which, moreover, permits the attribution of different weights for different input elements.

A *FIR filter*, where the acronym stands for "Finite Impulse Response", is a digital filter which consists of a finite number of sample values. The system is

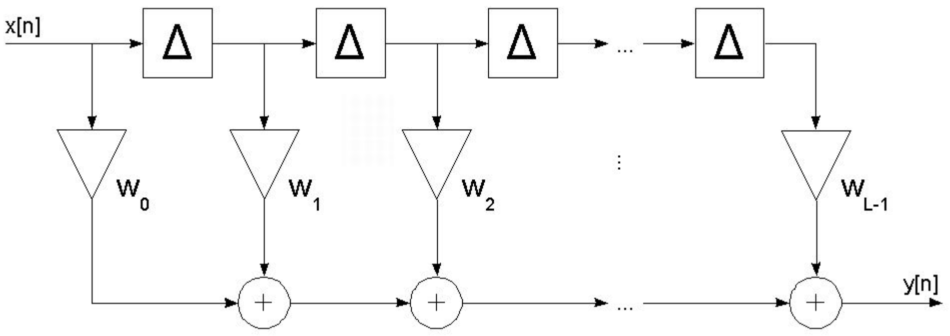


Figure 4.6: FIR filter in the direct structure.

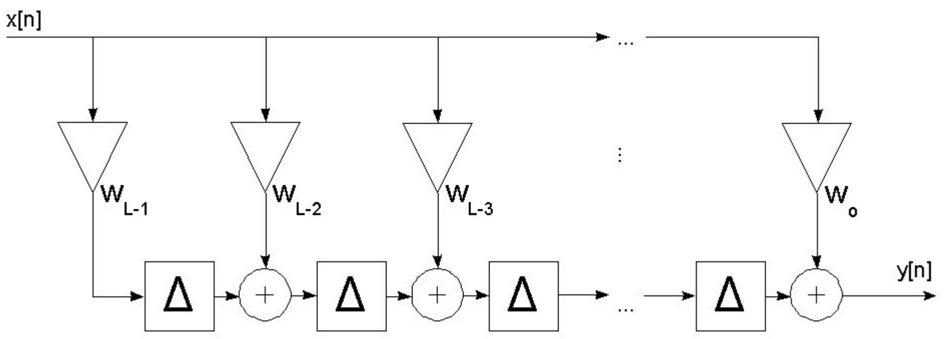


Figure 4.7: FIR filter in the transposed structure.

described by the following equation:

$$y\left[n\right]=\sum_{i=0}^{L-1}w_i x\left[n-i\right]; \tag{4.1}$$

where w_i are the coefficients that characterize the filter. It can be designed as the tapped delay line presented in figure 4.6 or the one illustrated in figure 4.7.

In order to design the wanted moving average filter, the weights have to be set according to the following equation:

$$w_i=\frac{1}{L} \qquad \text{for i}=0,1,2,\ldots,L-1; \tag{4.2}$$

where L represents the length of the line.

The VHDL implementation of the filter led to several problems, which are strongly connected to the nature of the FPGA and that help to understand the difference between software and hardware programming.

The input to the filter is the result of the last *1PPS* signal period measurement. In order to simplify the operations the FPGA has to perform, no division is performed, which would require a lot of hardware resources, but only shifts, which correspond to division by powers of 2. Thus, the length of the delay line has to be itself a power of 2; only in this way it is possible to have correct coefficients. Moreover, since all the weights are equal, there is no need for a tap weight for every sample; thus, as suggested by the following equation, just one coefficient was applied to the final accumulation:

$$\begin{cases} y\left[n\right] = \sum_{i=0}^{L-1} w_i x\left[n-i\right] \\ w_i = \frac{1}{L} \qquad \text{for i} = 0, 1, 2, \ldots, L-1 \end{cases} \tag{4.3}$$

$$\Rightarrow y\left[n\right] = \frac{1}{L} \cdot \sum_{i=0}^{L-1} x\left[n-i\right]; \tag{4.4}$$

Equation 4.4 was implemented in VHDL language as shown in figure 4.8. Compiling and simulation were successful, but when the layout generated by the place and route tool was physically programmed into the FPGA, errors in the calculations occurred. After deep debugging, the source of the problem seemed to be identified with the too high number of flip-flops connected to the same signal, i.e. the number of components the signal was given as input to, was higher than the fan out[4] of the component generating that signal. The solution was found in the transformation of the parallel accumulation typical of the FIR filter in a mix of parallel and serial configuration, made possible by the fact that no immediate output refresh is needed, but there's the opportunity of splitting a complex operation in several smaller duties (remember that the filtered measurement generated in the i^{th} *1PPS* cycle will be used to build the output signal in the $(i+1)^{th}$ one, so one second later). The central point of the solution is to accomplish the accumulation

[4]Fanout is a measure of the ability of a logic gate output, implemented electronically, to drive a number of inputs of other logic gates of the same type.

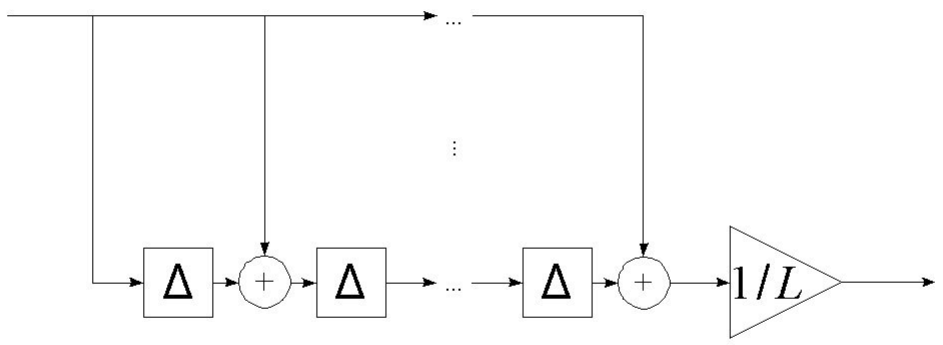


Figure 4.8: The implemented FIR filter.

of the input in a parallel way for a number of flip-flops lower than the fan out of the component providing the measurement; the operation is then brought on serially by blocks of the previous said parallel accumulations until all the sums are performed.

At the beginning a 32 input signal periods long sliding window was used, and everything was working fine, despite unwanted output fluctuations due to the small dimensions of the filter. But when its width was incremented, an unresolvable problem came to the surface: the number of slices characterizing the FPGA was not enough to fulfill the code's space requirements.

Thus, the idea of an FIR filter was replaced by a more simple filter: instead of using a number of accumulators equal to the length of the window, just one accumulator was created, which the most recent measure was added to and the most old one was subtracted from. But also this led to the same problem, because in order to perform the subtraction, the system needed to keep in memory a number of values equal to the window width.

Furthermore, even accepting the fluctuations compromising the output, the internally built *1PPS* signal was affected by strong drifting if compared with the *Tekelec*'s **Epsilon Board II**'s one. Obviously, this condition could not be acceptable.

4.4.2 PID controller

The above described situation strongly required the implementation of some sort of feedback in order to eliminate the drift. This was achieved with a PID controller, whose mechanism is here following described: it takes the last measured value of the *GPS-1PPS* signal and compares it with the length stored in memory; the difference (called *"error"* signal) is then used to adjust the same *1PPS* output length with three contributions, whose initials form the name of the controller:

P - a proportional one, obtained multiplying the last error by the correspondent weight K_P: conceptually, its task is to handle the intrinsic drift affecting the digital system;

I - an integral one, obtained accumulating the error multiplied by the correspondent weight K_I: its purpose is to adjust the output *1PPS* length in case the oscillator's frequency changes;

D - a derivative one, obtained multiplying the difference between the last delay and the second to last one by the correspondent weight K_D: it takes action to inhibit rapid changes of the delay trying to reduce the output oscillations and overshoot;

These terms concur in the output calculation as described by the following equation:

$$\begin{aligned} y\,[n] = &\quad K_P \cdot e[n] \; + \\ &+ \; K_I \cdot \textstyle\sum_{i=0}^{+\infty} e\,[n-i] \; + \\ &+ \; K_D \cdot \left(e[n] - e[n-1]\right); \end{aligned} \tag{4.5}$$

where $e[n]$ represents the *signed* delay, measured in *master clock* periods, from the internally reconstructed *1PPS* signal to the one provided by the GPS receiver (considering their rising edges as start and end points).

Equation 4.5 was translated in VHDL language in the way presented by figure 4.9.

The delay measurement was accomplished through a *Finite State Machine (FSM)* enabling a counter during the interval delimited by the rising edges of the two *1PPS* signals.

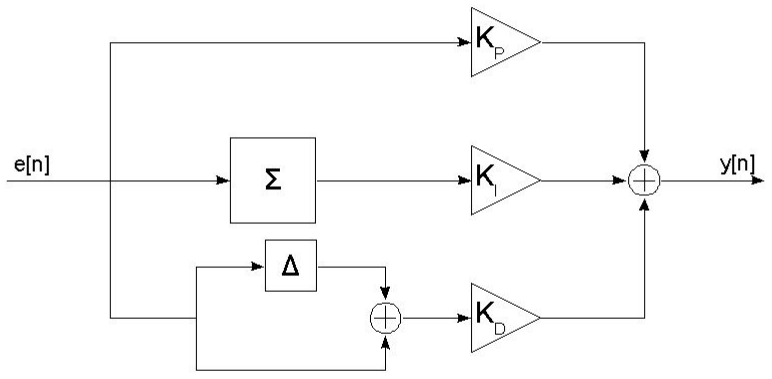


Figure 4.9: The implemented PID controller.

The recovery of the reminder described in chapter 3 was substituted with a rounding operation in order to eliminate the interference that seemed to be compromising the effects of both the PID controller and the reminder recovery procedure. This effect was due to the fact that the system did not take into account the accumulating fractional part when measuring the delay. The length of the *1PPS* signal, before being provided to the square wave generator, which of course has to receive an integer number of clock periods, is rounded in the following way:

- if the fractional part is smaller than 0.5, then the length is rounded to the immediately inferior integer;
- if the fractional part is larger than 0.5, then the length is rounded to the immediately superior integer;
- if the fractional part is equal to 0.5, then the length is rounded to the immediately inferior integer if the LSB of the bit string representing the integer part is '0', viceversa it is rounded to the immediately superior integer if the LSB of the bit string representing the integer part is '1'.

Such an algorithm was implemented in order to avoid redundancies which are know to cause numerical problems in digital environments: we can consider that, in case of fractional part equal to 0.5, there is equiprobability in the value assumed by the LSB of the integer part.

Another aspect that was considered was that the implementation of dividers in the FPGA consumes a lot of resources, so the applied weights were all set equal to powers of 2, allowing the division to be performed through simple, fast and space saving right shift operations[5]. Of course, in this way it was not possible to analytically determine the weights that have to be applied; therefore, an exhaustive research has been done by introducing in the code a disturbance in the accumulator which led to an extension of the output's length of $1\mu s$. Using an operating frequency equal to 50MHz, the response to the stimulus did not oscillate around zero but tended to it in a negative exponential way (refer to figure 4.10) with the following weights:

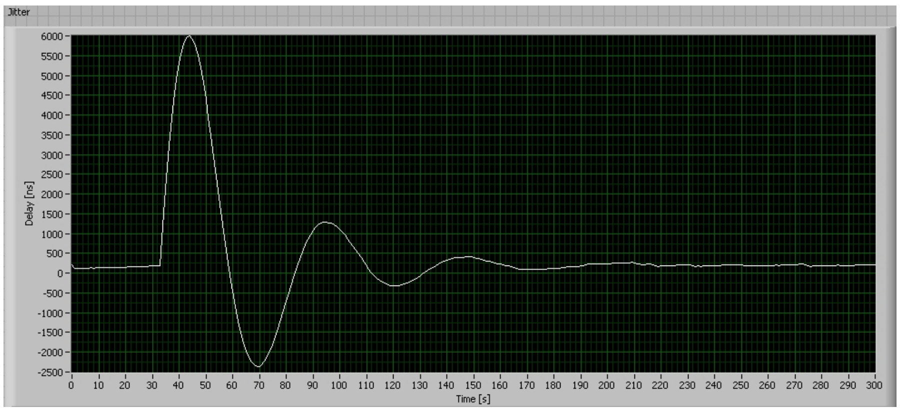
- $P = 2^{-3} = \frac{1}{8}$;
- $I = 2^{-5} = \frac{1}{32}$;
- $D = 2^{-\infty} = 0$;

Moreover, with these weights, the time needed for stabilization was among the lowest ones.

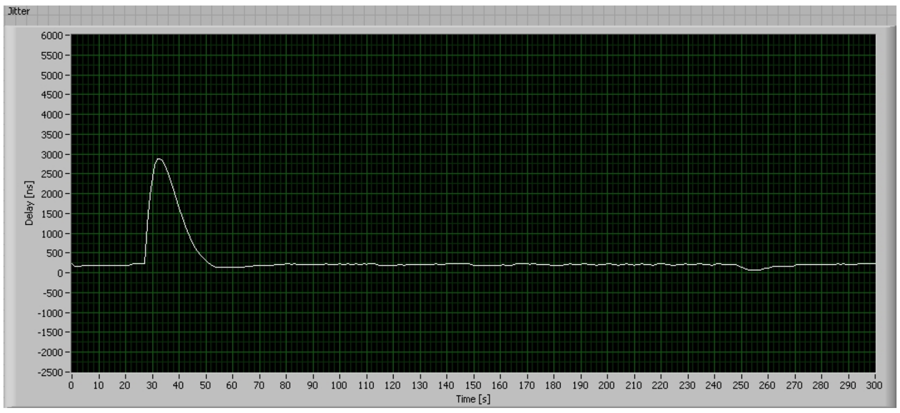
In a PID controller, the derivative part should partecipate to the control, but in our case its contibution is not helpful: the reason could be identified with the fact that jitter affects the reference signal, and the use of a derivative dependent component leads to unwanted fluctuations.

In this way, the system seemed to be working correctly since no drift affected the FPGA output; however the operating frequency seems not to be sufficient to drastically reduce the jitter, as will be described in the measurements exhibited in chapter 5.

[5]When using N-based representation, the result of a left or a right shift applied to a number corresponds respectively to a multiplication or division of that number by the base N.



(a) “Bad” step response.



(b) “Good” step response..

Figure 4.10: Step responses of the PID controller.

Chapter 5

Final results

The system so far described has been tested with the following configuration: the oscilloscope connected to the internet via Ethernet; *Tekelec*'s **Epsilon Board OEM II** is applied to one input, while the other one is dedicated to the signal that has to be analyzed. In order to achieve the highest precision in the measurement, the **Epsilon Board OEM II** is left connected to the antenna for 24 hours before starting the acquisition, allowing it to generate a high accuracy output, whose frequency and phase are locked to the *UTC 1PPS* signal provided by the receiver internal to the board. Then, the antenna cable is disconnected to avoid further adjustments by the closed-loop feedback control system: in such a way the provided output is characterized by really stable frequency, despite this results in a drifting behavior (with respect to the signal coming from the satellites) that can be easily eliminated in the post-acquisition phase through data filtering. The oscilloscope is then triggered by the rising edge of one of the two inputs (usually the one of the **Epsilon Board OEM II** was used) and the delay between the two pulses is measured. The so obtained value is then sent to a laptop in which the *"Oscilloscope's data acquisition.vi"* software was running. In such a way, data monitoring and analysis was performed, through the help of jitter vs. time graphs and histograms. The whole process is managed by the *LabVIEW VI*, which moreover saves the data into a text file for eventual future analysis.

Several measurements have been carried out, in order to understand the behavior of the system when changing its configuration, i.e. the PID coefficients.

Input statistics		
Max [ns]	Min [ns]	σ^2 $[ns^2]$
73.4	-67.7	416.5

Table 5.1: Acquisitions results - *U-BLOX*'s **LEA-4T** *GPS-1PPS*

With the FPGA working at 50MHz and the PID coefficients set according to the analysis explained in chapter 4.3, the PID controller seems to filter the jitter as wanted, despite the fact that firstly $20ns$ of resolution are lost in the sampling operation, secondly the filtering operation entails a $^{0}_{+1}$ difference, thus it can be possible that the difference between the incoming *1PPS* and the output one length is up to $20ns$ per second. Moreover the controller does not react instantaneously, but requires some time to update the output's length. The speed with which the system responds is higher with bigger coefficients, but this implies also a higher sensitivity to the jitter: there's the need to reach a compromise between stability and reactivity. In fact, the whole system is characterized by two factors: the first one is connected to the crystal oscillator connected to the FPGA, which features short-term stability (i.e. it is not affected by any jitter) but tends to drift; the second one is connected to the *GPS-1PPS* signal, featuring long-term stability (since its generated by the combination of the signals provided by atomic clocks in the satellites) but affected by jitter. Thus, the PID controller's purpose is to delegate the control of the long-term stability to the *GPS-1PPS* signal and the low-term one to the crystal oscillator. We can imagine there exist a threshold in the frequency domain which separates the operating ranges of the two elements and that can be moved forwards or backwards by setting the PID coefficients in the proper way.

Moreover, the effect of filtering result can be improved by increasing the operating frequency: the VHDL code implemented in the *Spartan3 FPGA*, featuring a -4 speed grade, works fine at this speed and as confirmed by the results shown in figure

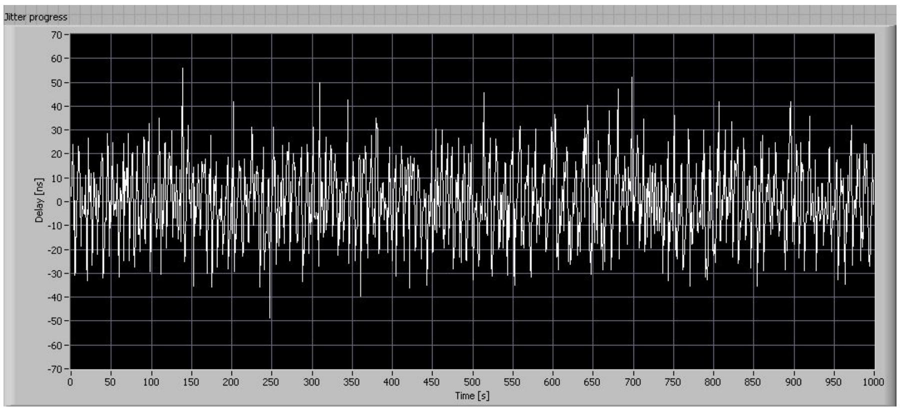
In tables 5.1 5.2 and 5.3 there's a summary of the results obtained from the acquisitions, while in figure 5.1 there's a comparison between the *1PPS* signal provided by the GPS receiver and the ones provided by the FPGA, working at 50MHz and at 70MHz respectively.

PID coefficients		Output statistics		
P	I	Max [ns]	Min [ns]	σ^2 [ns^2]
-2	-5	43.7	-38.4	158.0
-3	-5	69.7	-65.2	347.3
-4	-4	63.8	-51.9	215.1
-4	-5	54.8	-53.6	213.6
-4	-6	61.2	-76.6	281.9
-5	-4	70.2	-55-5	299.9

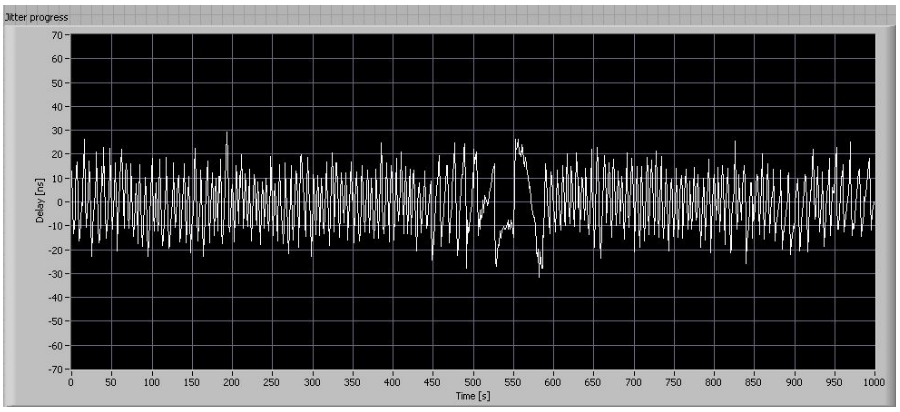
Table 5.2: Acquisitions results - Operating frequency: 50MHz

PID coefficients		Output statistics		
P	I	Max [ns]	Min [ns]	σ^2 [ns^2]
-3	-5	61.2	-59.8	227.4
-3	-6	53.7	-43.9	160.7
-3	-7	58.5	-51.4	168.9
-3	-8	85.7	-77.2	245.7
-4	-4	88.3	-62.4	375.2
-4	-5	53.2	-47.7	142.6
-4	-6	50.7	-60.9	159.9
-4	-7	74.8	-54.8	149.7
-4	-8	83.9	-83.0	261.1
-5	-7	85.2	-98.4	225.7

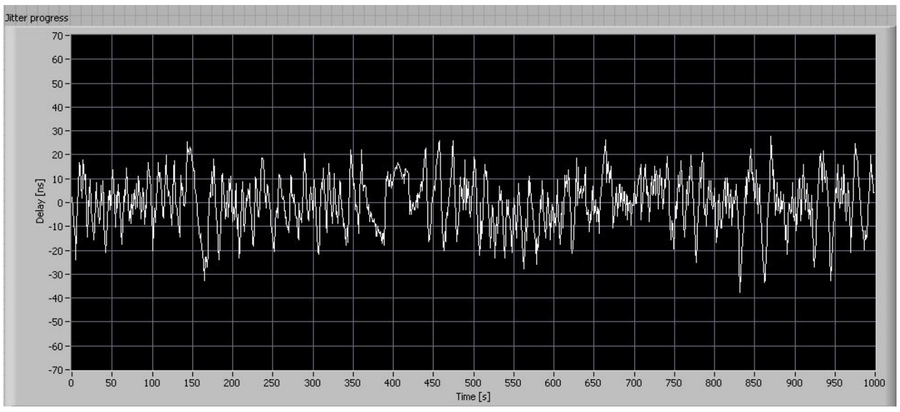
Table 5.3: Acquisitions results - Operating frequency: 70MHz



(a) *GPS-1PPS*



(b) PID controlled *1PPS* with FPGA operating at 50MHz



(c) PID controlled *1PPS* with FPGA operating at 50MHz

Figure 5.1: 1000 seconds long acquisitions of *1PPS* signals delay.

Chapter 6

Conclusions

This work provides a description of the implementation of a GPS board, whose purpose is to provide high accuracy timing signals to an optical *Multi Terabit/s switch* prototype, part of the *IP-FLOW* project. One of this project's aims is to propose a way to solve the switch and link bottlenecks problem that the internet network is going to experience as a consequence of the high definition media streaming diffusion over the common internet user population. The technology which the whole system relies on is the so called *Fractional Lambda Switching (FλS)*, based on the *Pipeline Forwarding* mechanism, which allows to solve the problem along with *QoS* guarantees. This is achieved through the exploitation of *Common Time Reference (CTR)*, which is extracted from a GPS module supplying an FPGA with *Universal Coordinated Time (UTC) one pulse per second (1PPS)* signal in order to generate high precision timing signals used to drive the switch. In such a way it is possible to reach both frequency and phase synchronization within few nanoseconds all throughout the network. The fact that the GPS board is FPGA-based entails low cost realization and integration with other functions, but with limited characteristics in relation to operating frequency and space for the related logic.

6.1 Final remarks

The thesis project required the implementation of software programs to automate data acquisition and data processing: a communication via Ethernet with an oscilloscope was established to collect the measured delay between signals provided by different GPS receivers and the realized GPS board. The requested output signals were a stable and accurate *1PPS* signal and a 10MHz clock aligned with the *1PPS* one. Unfortunately, the low resolution that characterize the FPGA (in the order of tens of nanoseconds) and the noise affecting the GPS signal required the implementation of a filter to be applied to the input signal. Since such kind of operation implies the use of sophisticated and as a consequence resource expensive algorithms which can not fit into an FPGA; thus, divisions were replaced with right shifts (thus forcing divisions by powers of two) to save space into the device. The filtering issue has been the most difficult to resolve since the realization of an FIR filter revealed itself to be space wasting and drift affected, despite the high stability of the output. The solution was found in the integration of a PID controller, which allowed to reduce the jitter power as the FIR did, but without being affected by drift.

Several measurements have been taken, trying to find the best configuration for the coefficients that characterize the controller. Better improvements were obtained increasing the operating frequency from 50MHz to 70MHz, but in both cases even slight tuning on the PID controller's coefficients led to completely different performance.

However, the minimum accuracy requirements needed for by the switch prototype were fulfilled and the board was successfully integrated into the testbed. With the best PID coefficients tuning the system managed to almost halve the jitter amplitude and to riduce its variance by a factor of $13 over 5$.

6.2 Further developments

Several other steps can be done to improve the generated output quality.

Further research activity such as estimating the spectral densities of the fluctuations of local clock and GPS receiver could be planned in order to minimize the fluctuations of the output *1PPS* signal by changing the coefficients of the PI controller determined in an analytical way. Furthermore, some other improvements related to the VHDL code optimization will be implemented in order to reach a higher maximum operating clock frequency. In fact, such a clock frequency increase is expected to improve also the accuracy of the synchronization circuit, leading to a further reduction of the jitter.

Other tests that can be carried out are connected to the realization of a dynamic controller, i.e. capable of changing the PID coefficients in a dynamic way according to the input behavior. Moreover, other approaches are possible: the FPGA used during the realization of this thesis is not the top of the range in terms of both the available space for logic and speed grade; thus, the use of a bigger and faster device could open new doors for the implementation of different types of controllers, notwithstanding a higher cost has to be paid.

Another interesting aspect that should be considered is how the designed device reacts if the jitter affecting the GPS receiver's output is much stronger. In fact, the actual cost of the whole GPS board is now dominated by the receiver; therefore, the use of a cheaper despite less performing should be taken into account.

Bibliography

[1] D. Agrawal, M. Baldi, M. Corra, G. Fontana, G. M. T. H. Truong, V. T. Nguyen, Y. Ofek, D. Severina, and O. Zadedyurina. Multi-terabit per second scalable switch prototype. Technical report, University of Trento (Italy), 2006.

[2] D. Agrawal, M. Baldi, M. Corra, G. Fontana, T. H. Truong, G. Marchetto, V. T. Nguyen, Y. Ofek, D. Severina, and O. Zadedyurina. A scalable approach for supporting streaming media: Design, implementation and experiments. Submitted for Pubblication at INFOCOM 2007.

[3] D. Agrawal, M. Baldi, M. Corra, G. Fontana, T. H. Truong, G. Marchetto, V. T. Nguyen, Y. Ofek, D. Severina, and O. Zadedyurina. Ultra scalable utc-based pipeline forwarding switch for streaming ip traffic. IEEE International Conference on Computer Communications (Infocom), 2006.

[4] D. Agrawal, M. Baldi, M. Corra, G. Fontana, T. H. Truong, G. Marchetto, V. T. Nguyen, Y. Ofek, D. Severina, and O. Zadedyurina. Scalable switching testbed not "stopping" the serial bit stream based on off-the-shelf components. 2007. Submitted for Pubblication at ICC 2007 Optical Networks and Systems Symposium.

[5] D. Agrawal, M. Corra, V. T. Nguyen, Y. Ofek, D. Severina, and O. Zadedyurina. Utc based controller for scalable time driven switching. IEEE Globecom 2006.

[6] M. Baldi, G. Marchetto, F. Risso, G. Galante, R. Scopigno, F. Stirano, V. T. Nguyen, Y. Ofek, D. Severina, and O. Zadedyurina. Time driven priority

router implementation and first experiments. International Conference on Communications (ICC), 2006.

[7] M. Baldi and Y. Ofek. Fractional lambda switching - principles of operation and performance issues. *“TRANSACTIONS of The Society for Modeling and Simulation International”*, 80(10):527–544, 2004.

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

[9] L. Gasparini, O. Zadedyurina, G. Fontana, D. Macii, A. Boni, and Y. Ofek. An effective digital circuit for jitter reduction of gps synchronization signals. Submitted for presentation to the IEEE Interbational workshop on Advanced Methods for Uncertainty Estimation in Measurement (AMUEM), Trento, July 2007.

[10] In-Stat. URL: www.instat.com.

[11] C.-S. Li, Y. Ofek, A. Segall, and K. Sohraby. Pseudo-isochronous cell switching in atm networks. In *INFOCOM*, pages 428–437, 1994.

[12] C.-S. Li, Y. Ofek, and M. Yung. Time-driven priority flow control for real-time heterogeneous internetworking. In *INFOCOM*, pages 189–197, 1996.

[13] G. Marchetto. Prototypal implementation of a time-driven priority router. Master’s thesis, Politecnico di Torino, Apr 2004.

[14] G. M. Mario Baldi. Pipeline forwarding of packets with a network-distributed common time reference. Submitted for Pubblication at INFOCOM 2007.

[15] U. Meyer-Baese and U. M. Baese. *Digital Signal Processing with Field Programmable Gate Arrays with Cdrom.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[16] Y. Ofek. The IP-FLOW project. URL: http://dit.unitn.it/ip-flow.

[17] Y. Ofek. Generating a fault tolerant global clock using high-speed control signals for the metanet architecture. *IEEE Transactions on Communications*, 42(5):2179–2188, 1994.

[18] Tektronix. *Tektronix TDS3000 & TDS3000B Series Digital Phosphor Oscilloscopes - Programmer Manual*. URL: www.tek.com.

[19] E. Union and E. S. Agency. Galileo - European Satellite Navigation System. URL: http://ec.europa.eu/dgs/energy_transport/galileo/index_en.htm.

[20] J. R. Vig. Introduction to quartz frequency standards. Technical Report SLCET-TR-92-1 (Rev. 1), Army Research Laboratory, Oct. 1992.