
Universidade de Aveiro
2011
Departamento de Electrónica, Telecomunicações e Informática

José Daniel Costa Varela

Sniffer Gigabit Ethernet em Hardware para sistemas de Tempo-Real

Gigabit Ethernet Hardware Sniffer for Real-Time Systems


Universidade de Aveiro
2011
Departamento de Electrónica, Telecomunicações e Informática

José Daniel Costa Varela

Sniffer Gigabit Ethernet em Hardware para sistemas de Tempo-Real

Gigabit Ethernet Hardware Sniffer for Real-Time Systems

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e de Telecomunicações, realizada sob a orientação científica do Dr. Arnaldo Silva Rodrigues de Oliveira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Dr. Paulo Bacelar Reis Pedreiras, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri

presidente Prof. Dr. João Nuno Pimentel da Silva Matos
Professor Associado, Departamento de Eletrónica, Telecomunicações e Informática, UA

Prof. Dr. Arnaldo Silva Rodrigues de Oliveira
Professor Auxiliar, Departamento de Eletrónica, Telecomunicações e Informática, UA

Prof. Dr. Paulo Bacelar Reis Pedreiras
Professor Auxiliar, Departamento de Eletrónica, Telecomunicações e Informática, UA

Prof. Dr. Luis Miguel Pinho de Almeida
Professor Associado, Departamento de Engenharia Eletrotécnica e de Computadores, FEUP

agradecimentos

Agradeço aos meus orientadores pela colaboração e apoio incansável. Agradeço aos meus amigos, principalmente aos membros da Erasmus Student Network. Agradeço à minha namorada e à minha família, especialmente à minha mãe, que sempre acredita em mim.

Este trabalho é financiado por Fundos FEDER através do Programa Operacional Factores de Competitividade – COMPETE e por Fundos Nacionais através da FCT – Fundação para a Ciência e a Tecnologia no âmbito do projecto "HaRTES: Hard Real-Time Ethernet Switching" (PTDC/EEA-ACR/73307/2006);

palavras-chave Tempo-Real, Ethernet, FPGA

Resumo

As ferramentas habituais de análise do comportamento lógico e temporal de uma rede de comunicações, conhecidas popularmente por Sniffers, são satisfatórias para as redes de uso geral. No entanto, não correspondem aos requisitos concretos de alguns protocolos de tempo-real, nomeadamente no que concerne à resolução e precisão das medições dos instantes de transmissão e recepção de mensagens. Esta incapacidade tem a sua origem no facto de estas ferramentas serem aplicações em software, a correr em computadores comuns. Nestes, as suas características multitarefa e o próprio mecanismo de "time-stamping" das mensagens não são apropriados para requisitos de tempo-real.

Como resposta a esta limitação, desenvolveu-se um Sniffer Ethernet em Hardware, recorrendo-se a FPGAs e a núcleos sintetizáveis de propriedade intelectual.

A ferramenta desenvolvida é capaz de capturar tráfego Gigabit num segmento Ethernet realizando o time-stamping das mensagens em hardware. Os dados são depois transferidos para um computador novamente pela via Ethernet.

Do lado do PC os dados são primeiro reconhecidos pelo popular software analisador de dados, Wireshark. Seguidamente, com recurso a ferramentas de software desenvolvidas, os dados são exportados e convertidos para um formato mais conveniente para serem analisados em ferramentas de cálculo.

A ferramenta mostrou ser capaz de capturar todo o tráfego procedente de uma porta Ethernet com uma precisão temporal de 8ns e um jitter de 16ns.

Keywords Tempo-Real, Ethernet, FPGA

Abstract

The standard tools for analysis of the logical and temporal behavior of a communication network, commonly known as Sniffers, are satisfactory for general purpose networks. However, they are insufficient for the specific requisites of some real-time protocols, namely in what concerns the resolution and temporal precision associated with the time-stamping of the arriving messages. This incapacity has its source in the fact that these tools are software based, running in common computers. The way time-stamping ins performed on these machines, as well as the multitask features associated with them are not appropriate for the requisites of real-time systems.

As an answer to this limitation, a Gigabit Ethernet hardware based was developed on an FPGA and making use of intellectual Property Cores.

The tool developed is capable if capturing Gigabit Ethernet traffic on an Ethernet Link, measuring the time-stamping on hardware. The data is then transferred again through an Ethernet Port.

On the PC side, all data is first captured by the popular software data analyzer, Wireshark. Next, making use of software tools developed, the data is exported to a convenient format, in order to be analyzed by math tools.

The tool proved to be capable of capturing all the traffic coming from an Ethernet port with an 8ns resolution and 16ns jitter.

ÍNDICE

LISTA DE FIGURAS

CHAPTER 1. INTRODUCTION

1.1. FRAMEWORK

The need for real-time communication protocols is everyday more visible. That comes from the specific requirements of some communication networks, where is crucial to achieve not only logical correction but also punctuality. Examples of such types of network are the air traffic control systems or an online ticket selling service. In the first example, a time failure can have disastrous consequences, sometimes even putting human lives in danger. On the second one, failures may be acceptable, but they will always cause loss of quality. It is then vital to use real-time protocols on these systems. The demand for higher bandwidth on these protocols has put Ethernet on the spot, with several efforts made to make this networking technology real-time oriented.

On the other hand, the development of more and more accurate protocols, demands also the development of network analyzers capable of reaching the precision achieved by these protocols. These network analyzers are commonly called Sniffers and their function is to evaluate the network for proper operation, capturing the messages flowing on the transmission lines and the time information associated with these messages. However, the majority of these tools are software applications running over an operating system. These last ones, deal with: processor assignment multiplexing, multiprogramming issues, interrupts, access denial to shared resources or even direct memory accesses. All of these factors originate unacceptable time-stamping errors and temporal uncertainty for real-time systems.

To overcome the shortcomings of the vulgar network analyzers, there has been a focus on the development of network sniffers hardware based.This project presents a solution for a network analyzer for real-time Ethernet networks operating at gigabit mode, aiming to be capable of responding to the temporal precision and resolution demands of the real-time protocols. For that to be achievable, all the reception, time-stamping and data process were made using dedicated hardware, immune to the mentioned issues of the software applications. The hardware was built on an FPGA (Field Programmable Gate Array), thus being possible to reprogram it. In addition, synthesizable intellectual property cores were used to implement the Ethernet MAC layer and to build the FIFOs, useful for temporary data storage of the captured messages. The transmission of the data captured is performed also at the expense of the Ethernet technology, making use of the various Ethernet PHYs available on the NetFPGA developing board (NetFPGA, 2011).

1.2. MOTIVATION

Previous approaches to the subject have revealed that hardware tools can achieve much more satisfying results than similar software applications.

An example of such tools is a Fast Ethernet Hardware Sniffer developed at the University of Aveiro entitled *Sniffer para Redes Ethernet de Tempo-Real Baseado em FPGA*(Puga, 2008). The Sniffer was compared with a popular software application known as Wireshark (Wireshark Developer's Guide, 2004-2010). From the comparison it could be concluded that the hardware approach was far better than the software tool in every aspect. First, with a software application there is always a risk for loss of packets and the user may not even be aware of their existence. Contrary, on the hardware approach all packets flowing on the transmission line were captured. The sniffer previously developed achieved a temporal resolution of 10 ns, and a maximum jitter of

100 ns, both values clearly smaller than the tens of µs of temporal resolution and a maximum error of some milliseconds associated with the software tools.

This new work intends to be a solution for an analyzer for Gigabit Ethernet traffic, than can be able to capture the Ethernet messages flowing on an Ethernet Segment at Gigabit rate and register with high resolution and accuracy the time of their arrival.

1.3. OBJECTIVE

This new approach aims on building a tool capable of sniffing Gigabit Ethernet Data, and transfer the information captured to a PC. The Sniffer must be able to capture the traffic on an Ethernet segment, and collecting time-stamping information with the precision demanded by Real-Time Systems. The data must then be analyzed on the PC, by means of data processing applications, and give relevant measurements of the data captured such as jitter, instant of arrival, packet size, number of messages exchanged and packet content.

1.4. STRUCTURE

Besides the introductory chapter, this document is structured in the following way:

Chapter 2 – Theoretical Background - This chapter provides relevant theoretical context for this thesis, providing information on topics such as Real-Time Systems and FPGAs. It also explores the different communication technologies considered for this work and the choice for Gigabit Ethernet. Finally, there is an insight about Ethernet, the communication technology used on the device designed.

Chapter 3 – State of the Art – Focuses on related work developed on the subject: existent real-time Ethernet protocols developed and the networking analysing solutions available both in software and hardware. It is given special attention to a Fast Ethernet Hardware Sniffer designed at the University of Aveiro.

Chapter 4 – Tool Development: capturing, storing and processing data. Information transference to the PC – It explains all hardware components developed: the physical interface components, and the components responsible for sniffing the Ethernet packets and to store them in IP Core FIFOs along with time-stamping and control information. It also explains the process of sending out the information collected to a PC.

Chapter 5 – Software Tools for Data Analysis – It is presented the way data is received on the PC side, the problems in reading data at a convenient format to be analyzed and the software tools used or created to overcome them.

Chapter 6 – Tests and Results – All the tests performed on the Sniffer and its results.

Chapter 7– Conclusions – A summary on the work developed on the whole project, and tips for future work.

CHAPTER 2. THEORETICAL BACKGROUND

2.1. REAL-TIME SYSTEMS

A real-time system is a system that is dependent in relation to time, meaning that it must obey to temporal constraints imposed by the surrounding environment. Failing to fulfil the time requirements in such systems will result in consequences that can go from a simple loss of quality in the system's function to unacceptable disasters like big economic damages, or even the loss of human lives. A DVD player is an example of a real-time system that, in case of delay in task performing, will result in a loss of quality, which may irritate the user but will not cause any harder consequences. However, if an air traffic control system task – another real-time system - fails to meet its deadline, it can result in the loss of hundreds of human lives. Either way, in both cases is evident that the surrounding environment of the system imposes temporal restrictions according to its own dynamic. Thus, a real time-system has to guarantee not only logical correction in its operation, but also temporal one. It must act on time!

2.1.1. ITS ABOUT PREDICTABILITY, NOT SPEED

When speaking of real-time systems, it is important to clearly distinguish among the different times associated with the system. Time variations can come from delays in getting the current state of the system, from delays on acting accordingly to the state or from variations in both last delays, phenomenon known as jitter. Figure 2.1 depicts these differences.

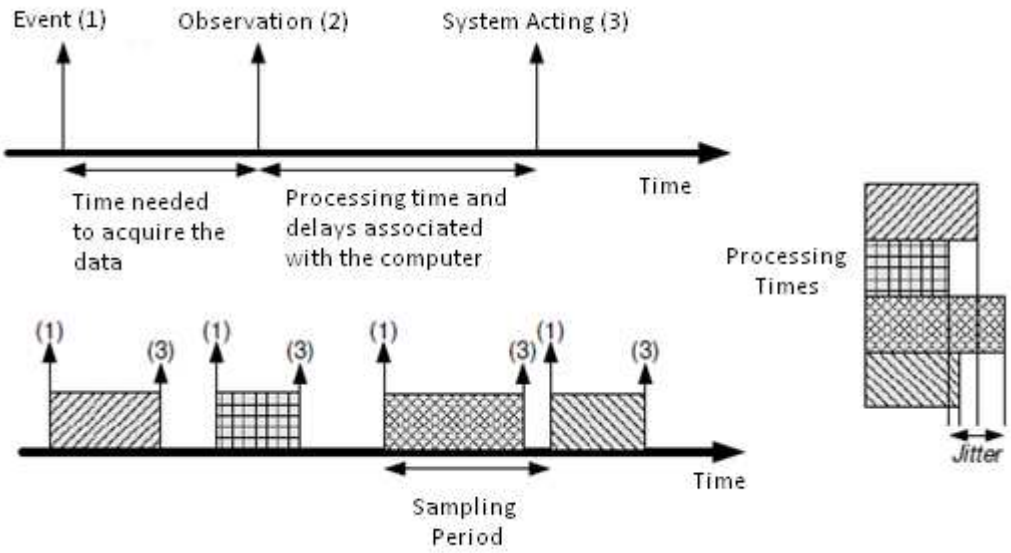


Figure 2.1 - The different times in a real-time system (adapted from (Puga, 2008))

It is a popular misconception to think that the temporal constraints associated with real-time systems are solved by increasing the speed of execution. The speed does increase the performance of a system, minimizing the average response time of a set of tasks. But more important than the average performance, in a real-time system, is the need to meet the time constraints associated with each particular task. In real-time systems, the concept of predictability is way more important than the speed of execution.

A system is predictable if its evolution and behaviour can be foreseen and if it offers guarantees – within limits of course – that it can fulfil its logical and temporal requirements. This is not an easy task in a lot of systems. A variety of different factors can influence the behaviour of a computational system during execution: the system load, hardware variations (i.e. clock deviations), the structure of the programming code, DMA accesses and pipelining, interrupts, multi-tasking, access to shared resources, etc. In a lot of systems a failure can origin catastrophic consequences. In real-time systems is then important: to study very well the time of execution of tasks as well as the worst case scenarios and possible failures; to have stable interfaces between subsystems to avoid error propagation; and that these subsystems can be separate from each other for independent verification purposes. The requisites towards assuring that a system is reliable are often called *dependability* requisites.

2.1.2. Real-time systems classification

Real-time systems can be categorized in respect to safety as: *Soft Real-Time Systems*, also known as non-critical, when the consequences in case of failure are in the same level of the benefits that the system offers. A DVD player system and the ATM Network system are examples of non-critical systems because failures are acceptable, despite the possibility of loss of quality. On the other side, systems are known as *Hard Real-Time Systems*, or critical systems, when the consequences in case of failure largely exceed the benefits offered by the system. This would be the case of an air traffic control system or of a traffic light system, as failures in such systems can cause exceptionable disasters (Todt, 2011).

Additionally, real-time systems can be classified as *Safe Systems* or as Operational *Systems*. A system is said to be safe when there is a variety of different alternative plans that guarantee good safety in case of failure. On the other hand, a system is considered operational when despite of losing performance in case of failure, is able to at least guarantee the functioning of minimum services.

Real-Time systems can also be distinguished in respect to the kind of response they offer. If there are enough resources to face even the worst case scenarios, the system is said to be a Guaranteed Response System. Alternatively, the Best Effort strategy controls the rhythm of the system, delaying responses in case of too many requests, and serving tasks or processes as it is possible.

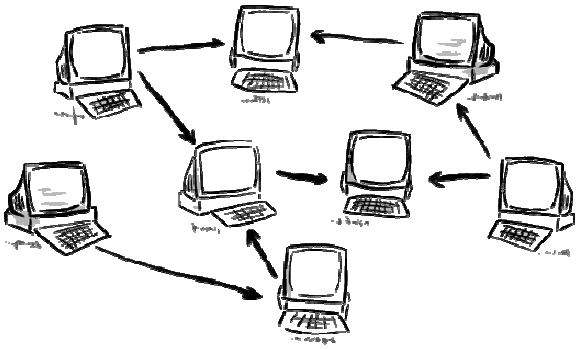
There is also a distinction to the way the real-time systems perform temporal control: Time-Triggered, when the system action is done periodically and caused by a clock signal. Alternatively, the action can be caused by an asynchronous signal due to a change in the system (e.g. an external interrupt), in which case the system is said to be Event-Triggered (Buttazzo G. C.).

2.1.3. Real-time Image and Database

The control system of a real-time system has to permanently update the value of the internal that represent the state of the system. The current value of these internal variables is called the *Image* of the system. As the real-time systems are very often related to physical processes, the value of the internal variables may change at any time, depending on the physical process itself. So the image of the system is valid only for a limited period of time and new data has to be acquired every time a variable changes. The set of successive images forms the *Real-time Database*.

2.1.4. DISTRIBUTED ARCHITECTURES

An architecture is said to be "distributed" when the system components are autonomous and physically distributed over an area. However, as the components communicate with each other through message passing, the architecture appears to the user as single coherent system.



Architectures

Distributed Architectures are very popular in Real-time Systems. The increase of complexity demands a capacity for local and more independent processing, where each component can take care of a specific task. The great advantage of this approach in Real-Time Systems is that hypothetical failures tend to be independent as well, so error propagation is more difficult in such architectures if adequate design methodologies are used.

2.1.5. SCHEDULING

Scheduling is the sorting of the resources of the system towards the fulfilment of the temporal constraints existent. Frequently on real-time systems, tasks have to share resources such us memory and the processor and on communication lines the messages share the transmission line itself. As there are different temporal requirements associated with the tasks or the messages in a system such as the period, the initial phase or the deadline, a need arises for algorithms that can attribute the different resources of the system to a specific task or message, in order to satisfy the temporal requirements associated with them.

There are two major types of scheduling: *Static Scheduling* and *Dynamic Scheduling*.

On the *Static Scheduling* there is a complete planning sequence of all scheduling decisions before the execution of the system tasks. This type of scheduling assumes that all parameters are immutable and known beforehand. Thus, it is very rigid! Any change will force a complete new mapping of the system tasks.

On the other hand, the *Dynamic Scheduling* is constantly executed and at any time choosing the new task to be executed. The only information available is regarding the tasks already triggered and all the decisions are taken based on the pending requests. This type of scheduling is less predictable than the static one and generates more overhead, but it is also more flexible and adapts easily to the occurrence of new tasks.

Some of the most popular scheduling algorithms are the algorithms with fixed priority. In such algorithms, the priority is a fixed parameter, and the tasks are ordered according to that parameter. Examples of this kind of algorithms are: the *Rate Monotonic* (Liu & Layland, 1973), in

which the assignment of priority is inversely proportional to the period of the tasks and the *Deadline Monotonic*, with the attribution of priority being inversely proportional to the deadline of the tasks. It is worth to remind that the algorithms just presented are a good solution when dealing with tasks on real-time operating systems, due to the centralism that exists. But in communication networks those algorithms do not present themselves necessarily as a suitable solution, since they require complete information about the system state to schedule the messages. Such level of information is not available and may imply a significant amount of overhead to obtain. Some real-time ethernet protocols (e.g. FTT-SE and Ethernet Powerlink) employ a centralized scheduling architecture that allows implementing this kind of scheduling.

2.2. FPGA

2.2.1. WHAT IS?

In a nutshell, an FPGA is a semiconductor device that can be programmed after manufacturing. FPGAs can be programmed to perform any logic function desired, but differently from ASICs (application-specific integrated circuits), FPGAs have the capability to replace that same function for another one.

The acronym FPGA stands for Field Programmable Gate Array. Field programmable means that it is the user's responsibility, and not the manufacturer's, to give a logic function to the FPGA. The Gate Array refers to the way FPGAs are designed to be reprogrammable. An FPGA is composed of thousands of identical logic cells. Those cells can be programmed independently from each other and they perform simple standard logic functions. The cells can then be interconnected through a matrix of wires and programmable switches. The FPGA's function is achieved by programming each logic cell with a simple function and then closing the appropriate switches.

Figure 2.3 gives an insight of the internal structure of a Virtex FPGA. It can be seen that it is composed of configurable logic blocks, surrounded by I/O blocks and interconnected by the wires matrix. It is important to make a distinction between a configurable logic block and a logic cell. Although the definition may vary accordingly to the FPGA, in the case of Virtex-II Pro (the one used in this work) the logic hierarchy is as follows:

Logic cell – a 4-input lookup table, a flip flop, interconnection to other cells and arithmetic logic to compute a 4-input expression;

Logic slice – 2 logic cells, although on Xilinx there are 2.25 logic cells per slice because they can do more per configurable block (CLB) than other architectures;

Configurable logic block – consists of 4 slices.

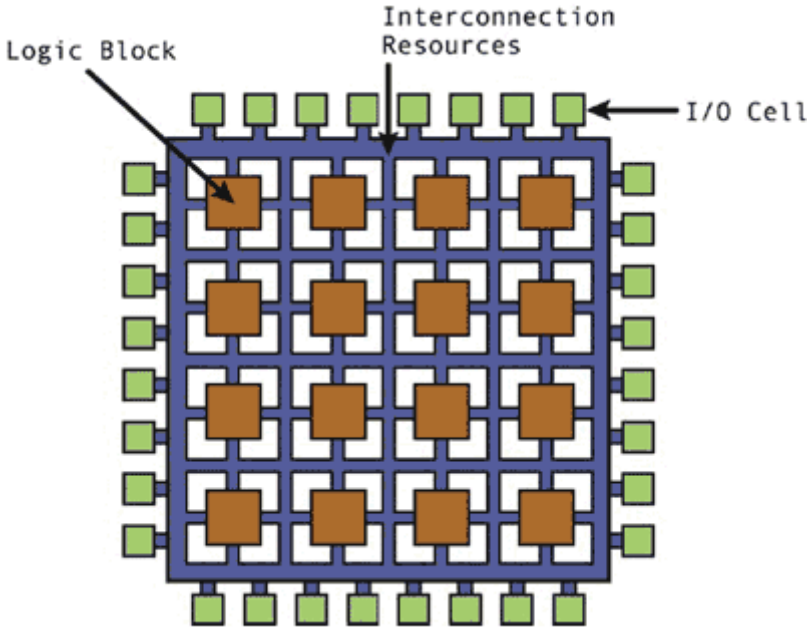


Figure 2.3 – Internal Structure of a Virtex FPGA

2.2.2. FPGA DESIGN FLOW

In (Xilinx, 2011) there is a complete description of the design flow process and the Xilinx tools helpful in every phase. The project flow starts with the *Design Entry Phase*. This phase consists of building up the design, that is, the circuit to be implemented. Usually Hardware Description Languages (HDLs) are used, but it is also possible to build the schematic of a circuit through graphical tools. These last ones are more suitable for designers who want to deal more with the hardware. But when the design is complex and the designer wants to think of the design algorithmically then HDLs are preferable. The Xilinx Integrated Software Environment (ISE) was the software tool used to support the Virtex-II in this work, and VHLD the hardware description language used to build the design circuit. A set of pre-existing libraries with predefined basic blocks is available to the programmer, avoiding each user to build the same standard blocks. Additionally, software tools allow the generation of configurable components, as it is the case of the CORE Generator from Xilinx, which was used to generate not only the Tri-Mode Ethernet Media Access Controller (TEMAC) but also the FIFOs used in this work.

The second phase of the design flow is the *Synthesis Phase*. The synthesis consists of translating the VHDL code into a circuit with logical elements (registers, AND-ports, etc.) by stating what kind of elements exist in the design and what is connected to what. This is called a netlist: a file that conveys information about connectivity and provides nothing more than instances, nets and maybe some attributes. The Xilinx software used at this stage is the XST (Xilinx Synthesis Technology) that also checks for any errors on the code syntax and analyses the hierarchy of the design to make sure that the design is optimized for the chosen architecture. The output is a .ngd (Native Generic Circuit) file.

The following phase is the *Implementation Phase*. This phase is constituted of three steps: translate, map and route. The translate process collects the information on the netlist together with the constraints saved on a .ucf (User Constraints File) and builds an NGD (Native Generic

Database) file. At this stage, ports are assigned to physical elements (ex. pins, switches, buttons, etc.) of the intended device and time requirements of the design are specified. The mapping process matches the logic defined in the NGD file with the FPGA components (such as CBLs, IOBs, etc.), generating an NCD (Native Circuit Description) file. This file represents how the logic elements of the design are distributed among the FPGA resources. Finally, the place and route process physically places the logic into FPGA components according to the NCD file and makes the necessary connections. It may also make judgments on the best physical assignments if there is a constraints conflict, in order to get the best performance out of the design. The output file is a completely routed NCD file.

The final phase is the Device Programming Phase. As the information needs to be transmitted to the FPGA device in a format that the FPGA can read, the routed NCD file is transformed into a bit stream to configure the FPGA device. The loading can be done through iMPACT, a configuration tool that takes care of all process between bit stream generations to the device download. This file contains the necessary information to configure the logic cells with the intended logic function and to selectively close the switches of the interconnection matrix.

Additionally, it is possible to verify and debug the design to ensure logical correction throughout the whole project flow.

Firstly, even before implementing the design, it is possible to simulate the behaviour of the circuit using software design simulators, as it is the case of Xilinx ISE Simulator. The purpose is to confirm that the design logic is functioning as intended. This is called *Behaviour Simulation*.

During *Implementation Phase*, *Functional Simulation* gives information about the logic operation of the circuit. It is possible to verify the functionality of the design right after the translate step, and in case the functionality is not the expected, to correct the code. After the MAP and PAR steps, timing reports listing signal path delays allow for a *Static Timing Analysis*.

Finally, after the FPGA has been configured the verification is possible through the use of software based logic analyzers that monitor the status of selected signals, making possible the detection of errors. On this case, blocks of memory inside the FPGA are used to store the value of the signals, which are later transferred to the PC through a JTAG interface. The Xilinx Chipscope Pro is an example of such software. Alternatively, the physical signals can be routed to the FPGA pins, and then verified with logical analyzers like an oscilloscope. All the phases are represented on Figure 2.4.

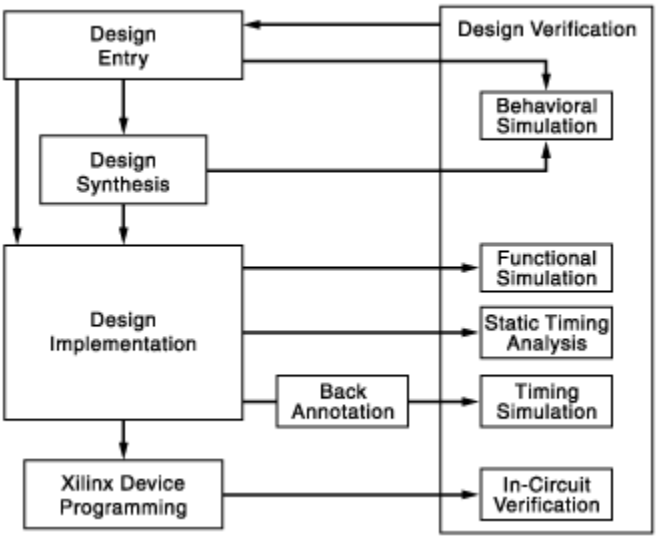


Figure 2.4 – FPGA Xilinx Design Flow (taken from (Xilinx, 2011))

2.2.3. WHY ARE FPGAS NEEDED?

Nowadays, building hardware has become a choice between flexibility and speed. When designing hardware based on flexibility, different applications may be executed. However, when performance is the main aim, fewer operations will be available, but the ones available will be faster.

It is possible to build general purpose chips, that is, chips able to execute a variety of different functions. Application-specific devices, on the other hand, can execute only a small limited number of instructions. However, for these same instructions, these devices are way faster than general purpose ones.

Examples of general-purpose devices would be the microprocessors like INTEL, Pentium or Motorola Power PC. All of these are commonly found on people's computers, and they can execute virtually any algorithm desired. On the opposite side, the dedicated-hardware circuits like *ASICs (Application Specific Integrated Circuits)* can only perform the concrete functions for which they have been designed, but are smaller, faster and consume less energy.

The real-time issues already explained on this thesis clearly indicate that there is a great need of performance for the Sniffer, this way justifying the creation of specific hardware to better accomplish the Sniffer's tasks. What are then the advantages of using an FPGA over an ASIC? The answer relies on the definitive characteristic of the ASIC hardware: a slight modification or the inclusion of a new functionality requires the development of a new component. Differently, the FPGA can be reconfigured at any time, allowing for constant hardware improvement, without the need for extra manufacturing. The huge cost of ASICs is also something to bear in mind, since they require mass production in order to be profitable.

In a way, it can be said that an FPGA gets the best out of both the general purpose and the dedicated hardware systems. It can be considered general purpose in the sense that it can become any desired circuit and thus is able to perform a variety of different functions, but as soon as it becomes the new circuit, it operates towards the specific function intended for that same circuit.

2.3. COMMUNICATION TECHNOLOGIES AND THE CHOICE FOR GIGABIT ETHERNET FOR TRANSMISSION

One of the first technologies thought right away to be suitable for out flowing large amounts of data per time unit was the Serial ATA (SerialATA Workgroup, 2003). The Serial ATA is a technology for transferring data between a computer and large storage devices such as hard disks. This technology succeeds the homologous technology known as Parallel ATA, but the serial transference of data carried out by the Serial ATA results in the usage of much thinner cables. This allows for operation at frequencies that were not possible to achieve in the previous technology, permitting higher throughput.

By the time that this technology was a possibility for this project, there was an intellectual property CORE that could make the FPGA to operate as a SATA device. This CORE supported the 3.0 SATA specification, thus theoretically allowing the communication to achieve speeds of 3.0 Gb/s. With this solution, some driver updates or changes could still need to be done on the host side, as a lot of device drivers labeled as SATA are often running in "IDE emulation" mode.

However, the choice for SATA technology seemed to be the one offering more guarantees in terms of speed and implementation. Unfortunately, the IP CORE was only available at a high price, so this option was unsuitable economically.

After dropping SATA, the Parallel ATA was taken into consideration. The 133 MB/s data rate maximum transfer supported by this standard was not very exciting. However, the choice for PATA would have the advantage of being compatible with SATA technology, and the sniffer could be left working on PATA technology until SATA IP Cores would be available cheaply, or developed by someone else on another project. Nevertheless, the only IP Cores found on Internet were oriented so that the FPGA would behave as a SATA host device, meaning that the information would have to be sent to a hard disk, and then switch plugs with the final host destination. On top of all these, building a device on a technology that will be soon wiped out of existence was not pleasant at all. All these disadvantages also took the Parallel ATA out of the way.

Other technologies, such as the USB 3.0 standard and FIREWIRE were also disregarded for lack of ports in the boards. Additionally, in all cases the boards available with such ports had only one Gigabit Ethernet PHY. There would still exist the need to build another Ethernet Gigabit PHY.

The obvious choice was then the Gigabit Ethernet technology. The data rate would be enough for sniffing Fast Ethernet Networks and the NETFPGA (NetFPGA, 2011), used in this work, was available and has sufficient Gigabit Ethernet ports for both sniffing and sending at the same time. Besides, the IP Core necessary to make the FPGA communicate through this standard was available from Xilinx as a .ngc (Native Generic Circuit) file. Of course one Gigabit Ethernet port cannot leak data from two other Gigabit Ethernet ports if both channels transmit data at a high rate, but the technology for sniffing Gigabit Ethernet networks could be built, and in the future the transmission technology can be changed, even to 10 Gb technology if there are physical resources to accomplish such a task.

2.4. THE ETHERNET

Ethernet is a network communication standard for data transmission among devices that are relatively closed to each other, that is, for Local Area Networks (LANs). This almost-forty year's old technology has been standardized as IEEE 802.3, and works at both the Physical (defining cabling and electrical signalling) and Data Link (packet frame format definition) Layers of the OSI Model.

Initially oriented for networks in which all the devices would be connected to the same transmission line, the growth of network dimensions has made the technology evolve to the concept of Switched Ethernet, which allows network segmentation.

Over the years, Ethernet has become the most famous and used network technology in the world.

2.4.1. ORIGIN AND EVOLUTION

The birth of the Ethernet goes back to the year of 1973, and has Robert Metcalfe, a Xerox researcher, as the father. His mission was to connect hundreds of computers on the same building among each other and to the first laser printer ever made. Working towards that aim, he developed the physical requirements to interconnect devices in proximity within each other as

well as the rules for their communication. That was the process of building the technology that is still nowadays known as Ethernet.

The Ethernet physical layer has developed a lot throughout time, with data rates constantly increasing.

The first version of Ethernet used the coaxial cable as the physical medium, operating at a speed of 2.94Mbit/s. The following early implementations of Ethernet remained coaxial, reaching speeds from 1Mbit/s to 10 Mbit/s.

The coaxial cable has been replaced by the UTP, Unshielded Twisted Pair, mainly for economic reasons. The UTP is a cable consisting of 4 twisted pairs of wire (meaning 8 wires per cable). The aim of twisting the wires is to eliminate electromagnetic interference. The cables then connect to devices through RJ45 connector, like the one we see on Figure 2.5. The standards designed for this cable allow speeds up to 100 Mbit/s, known as Fast Ethernet and 1000 Mbit/s, known as Gigabit Ethernet. The UTP is the most popular Ethernet cabling and it was the one used on this work.

Nowadays, the optical fiber is also becoming popular, because although they are more expensive, they also offer better performance than coaxial cables or UTP cables.

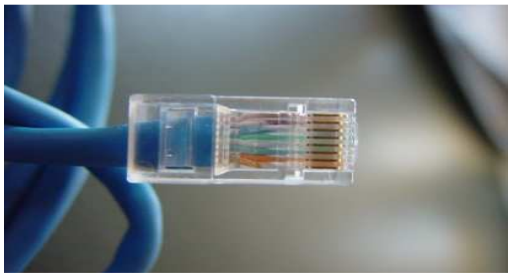


Figure 2.5 – UTP Cable

Initially Ethernet was designed only for devices sharing the same medium. When such a situation happens, all the devices in the network are connected to the same transmission line and the rules for communication are very similar to the ones used by human beings at a dinner table, as described in (Pidgeon, 2000). How?

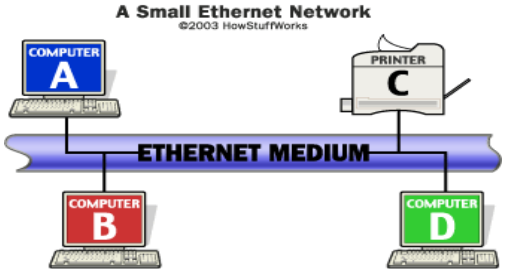


Figure 2.6 –Ethernet Shared Medium

Well, the same way people sharing the same dinner table can listen to everything that is being said there, so the devices using the same medium can hear all the messages in the transmission line. This way, all devices in the network must have a unique physical address that identifies them. This address can be used to convey a message to a specific device in a way similar to the one human beings reach a specific person by means of this person's name: "Peter, can you pass me the salt, please?" On Ethernet networks this physical address is known as the MAC (Medium Access Control) address.

Also, the rules of communication applied to human beings teach people to be polite and to speak one at a time. Analogously, devices in the network can hear the current conversations, and will only attempt to transmit a new message when there is silence at the transmission line. What happens when two devices try to break the silence at the same time? In that case there is a collision. If two people at a dinner table would attempt to speak approximately at the same time after a moment of quietness, one of them could just start talking louder and louder till the other one would shut up. However, if both are polite people, they would both shut up immediately. The earlier one then attempting to restart talking would then be the one communicating. In the same way, devices attempting to communicate at the same time stop their transmission, wait for a random amount of time and finally attempt to transmit again if there is silence at the transmission line.

Figure 2.7 – CSMA/CD Dinner Table Analogy (taken from (Pidgeon, 2000))

On this dinner table analogy, it was being assumed that only a few people were at the dinner table. As the number of people eating at the same table would increase, restricting the right to speak to only one person at a time would make a lot of people frustrated, because they would have to wait a lot of time before being allowed to speak. Of course, at real life this analogy is broken, because human beings are able to have multiple conversations going on at the same table. The human ear is selective enough to pick the messages that a person is interested in, and people engage in the topics that they feel addressed to automatically. Additionally, people inherently choose other people in proximity to have a chat with because if a lot of people are talking, odds are that only close voices can be heard, if not yelled. When dealing with electric signals, all this natural portioning does not occur, because signals propagate very fast for long distances, and having multiple devices connected to the same transmission line would end in a lot of collisions, generating network congestion. The solution to this problem is to divide the single segment into multiple segments. This way, collisions can occur only among devices connected to the same segment, or in other words, the same collision domain.

The concept of segmentation has evolved in a way that on today's Ethernet Networks, each device has its own dedicated segment. They can communicate with other stations through switches. A switch is a device that receives all packets from all segments connected to it, and then forwards each Ethernet packet only to the intended station. This way, it is possible to have multiple communications happening in parallel among the devices in the network. This concept is known as Switched Ethernet.

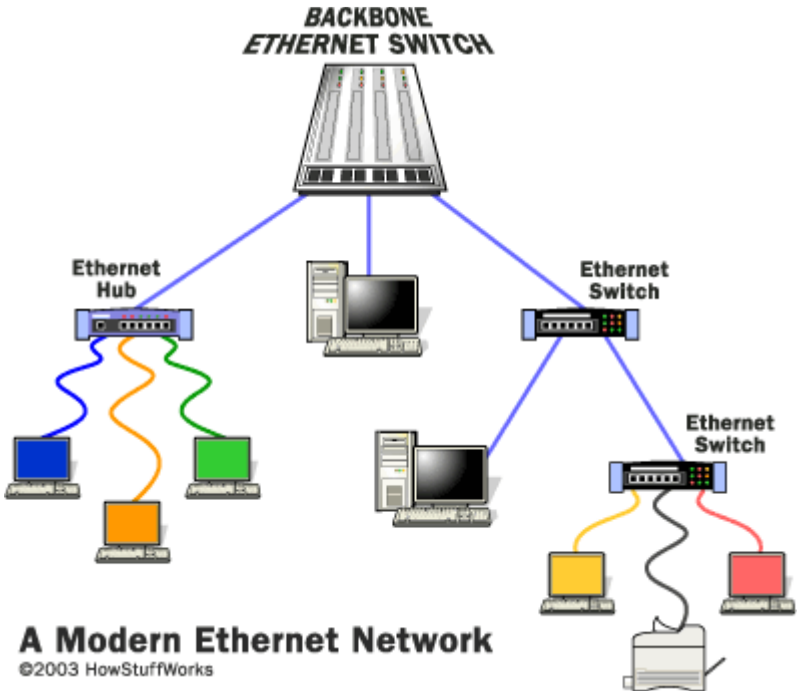


Figure2.8 – How an Ethernet Network looks nowadays

2.4.2. REDUCED GIGABIT MEDIA INDEPENDENT INTERFACE

The Ethernet physical layer is responsible for connecting an Ethernet device to the transmission medium. That is accomplished by means of a circuit commonly known as Ethernet PHY. The PHY has the function of enabling analog access to the link, coding and decoding the data sent over the twisted pair cables.

To the upper side of the communication chain, the PHY encounters the MAC (Medium Access Control) device. In order to make possible the connection of different types of PHY devices operating at gigabit speeds to different media (i.e. twisted pair, optic fiber, etc) without redesigning or substituting the MAC hardware, an interface called Gigabit Medium Independent Interface (GMII) was created. This interface makes the bridge between an Ethernet PHY and a MAC device, and can operate on speeds up to 1000 Mb/s. However, as this interface was based and it is compatible with the former Medium Independent Interface (MII), slower speeds like 10 or 100 Mb/s are also allowed. In order to achieve the maximum speed of 1000 Mb/s the GMII Interface uses an eight bit data interface clocked at 125 MHz. However, the interface used in this project was the Reduced Gigabit Medium Independent Interface (RGMII)(RMII Consortium, 1998), which is able to achieve the same data rate with an only four-bit data bus from the PHY. This reduction is achieved by clocking data on both the rising and falling edges of the clock and by eliminating non-essential signals when operating at 1000 Mb/s.

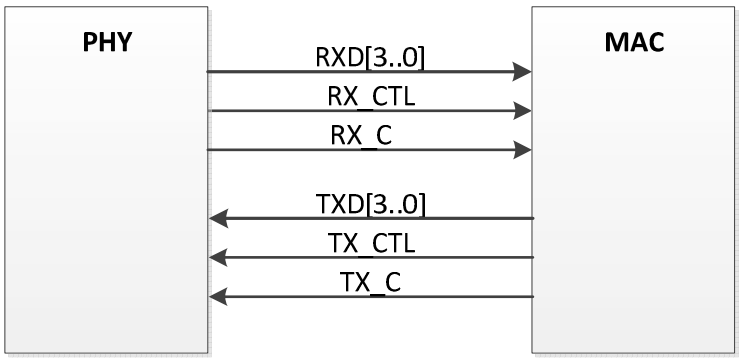


Figure 2.9 - The RGMII Interface: an interface between the PHY and the MAC

2.4.3. THE ETHERNET FRAME

The Ethernet data is transmitted in frames, and the frames are composed of several different fields. The fields of the Ethernet Frame (see Figure 2.10) are transmitted from left to right, and the bits in the fields are transmitted from the most significant to the least significant. All Ethernet devices follow the order and the size of the fields when transmitting, and expect incoming frames to have such a pattern. This is of course what makes possible for the Ethernet devices to understand each other, by means of "speaking" the same "language".

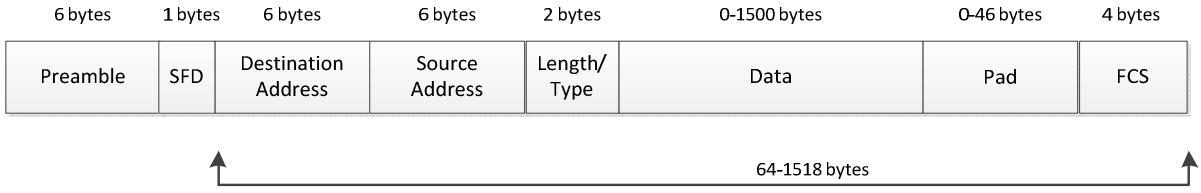


Figure 2.10- The Ethernet Frame

The purpose of each field is as follows:

Preamble – it is a group consisting of a byte pattern that repeats itself 7 times. This field started out being used for bit synchronization, although nowadays Ethernet MACs are able to receive frames without this field.

SFD - SFD stands for *Start of Frame Delimiter* and it marks the beginning of the frame. It is composed of only one and the same byte: 0xD5.

Destination Address– it is the MAC address of the intended recipient to where the frame is going to be sent.

Source Address – it is the MAC address of the sender recipient.

Length/Type–this field can either be interpreted as length field, if the value of this field is equal to or less than 0x05D5, or as a type field if above that number. If used as Length field it indicates the number of bytes in the Data field(Novell, 2003).

Data – the data is inserted here and must be between 0 and 1500 bytes.

Pad – this field ensures that the minimum length of the Ethernet frame is 64 bytes, adding zeros to the frame. This field can vary from 0 to 46 bytes depending on the length of the Data field, as the remaining ones have always the same size.

FSC – it contains the *Frame Check Sequence*, which is calculated using Cyclic Redundancy Check (CRC). This field is used for error detection and rejection of spoiled frames.

CHAPTER 3. STATE OF THE ART

3.1. INTRODUCTION

In this section it is provided a general context of the purpose of the project developed. The chapter represents a brief description on the existent knowledge on the ways to communicate on Real-Time Networks and the role of Ethernet as a suitable technology for those systems. There is also an inspection on the different types of Network Analyzers and their accuracy on timing measurements.

3.2. MAKING ETHERNET REAL-TIME

Real-Time electronic distributed systems were an important step in technological evolution. In such systems, the communication between the different nodes obeys to specific constraints in time. These real-time systems become more prevalent everyday with applications that can go from simple multimedia tools to critical industrial control systems. Accompanying this growth is the intent in building real-time protocols that can suit such specific-requisite systems (Doyle, 2004).

On the last decades, special networks commonly known as Fieldbuses were developed, aiming to suit the needs of the distributed control systems, such as handling a lot of small data messages with strict time constraints or being able to order tasks based on priority. However, the growth in application complexity and in the amount of data exchanged increased over time, and eventually exceeded the capacity of traditional Fieldbuses. So attention drew to higher bandwidth networks. Among those, despite its non-deterministic feature, Ethernet has been frequently seen as a potential solution to these problems because of its:

- Cost – it is produced at a large scale, making it cheap.
- Bandwidth – data rate has been increasing repeatedly over time, and it is expected to continue to grow.
- Popularity – it is the most popular media in use, which means wide availability not only of experts familiar with this protocol but also of test equipment.
- Compatibility – it's easy to integrate with higher layer protocols and with Internet, allowing for remote control or monitoring. (P. Pedreiras, 2005)

However, the Ethernet as defined in the 802.3 is not real-time suited *per se*. The non-deterministic bus access mechanism, CSMA/CD, allows unpredictable message loss and timing, when dealing with collisions.

In(P. Pedreiras, 2005), in an effort to compile everything that has been tried on this subject, several approaches on how to adapt Ethernet to Real-Time systems are presented. They are categorized in the following groups:

- CSMA/CD based protocols;
- Modified CSMA protocols;
- Token Passing;
- Time-Division Multiple Access [TDMA];
- master/slave techniques;

and Switched Ethernet.

Using small sized messages and keeping the bus utilization factor low, it is actually possible to use Ethernet on Real-Time applications. The probability of collision is very low in such conditions, offering a high level of certainty that messages will not be lost norwill they miss their deadlines. The NDDS and ORTE are examples of such protocols that rely only on those factors and that can work with the CSMA/CD on real-time applications, though because there is no absolute guarantee of meeting the temporal constraints of the system, these protocols suit better Soft Real-Time applications.

As opposed to this last method, some protocols do change the properties in the regular arbitration mechanism, either by delaying transmissions to reduce collisions or by controlling the collisions when they occur. However, these algorithms have Internet integration and expansion capability drawbacks, as new nodes would also need to be changed in a similar fashion.

A very simple approach to make Ethernet Real-Time oriented is by means of a token. On this kind of approach, there is only one token for the entire network and so it can only be used by one node at a time. The node that has the token is thus the only one that can transmit messages. There are different ways to handle the token, though. For example, with RETHER protocol the token rotates periodically, but with RT-EP protocol the token is first rotated through all the nodes just to determine the one with higher priority, to which the token should be given back to.

The TDMA protocols consist in having an exclusive time window attributed to only a node or a device, which completely removes the possibility of happening collisions. This way, these protocols are ideal for systems in which safety is crucial.

The Master/Slave approach is a technique in which one of the nodes, the Master, controls the remaining nodes, the slaves. These last ones can only access the medium with permission from the Master. This will of course have the direct consequence of setting the medium free of collisions, but will also increase the overhead: for each data message coming from a node, a control message needs to be sent before to that same slave. An attempt to overcome this problem is the FTT protocol, in which there is flexibility to the amount of control data sent over time. This protocol divides time in cycles, each of them containing only a control message that schedules all data messages over that period of time.

Switched Ethernet also provides a medium with no collisions and reduces the effect of the non-deterministic feature of the original CSMA/CD arbitration mechanism, but it doesn't eliminate it completely. Buffers can still get full, if the rate at which messages arrive is higher than the rate at which they departure. This could result in loss of messages or deadline missing. As a way of overcoming these last problems, switches built with several queues of different priority have been proposed.

3.3. NETWORK ANALYZERS

Network Analyzers are devices used to inspect the data flowing on communication networks. Their two main goals are:

1. To capture the messages on the transmission line.
2. To register the arrival time of the messages captured.

This last action is commonly known as time-stamping. Roughly, there are two methods of performing the time-stamping of the messages: Software Time-Stamping and Hardware Time-Stamping. However, the accuracy of the measurements depends a lot on how and where in the

communication chain they are taken, as it can be seen in (Weibel & Béchaz, 2004), an article where a comparison between different types of measuring time-stamping took place.

At the Application Layer, for instance, time-stamping is not very precise. The reason for this is associated with the multiprogramming characteristics of the general purpose operating systems, in which the CPU is busy with a variety of different applications and thus is not checking all the time the messages arriving. Furthermore, the PCI bus is used to establish the communication between the CPU and not only the network interfaces, but other peripherals such as USB adapters or video or sound cards. So the time that takes for the NIC to get access to the PCI bus will introduce error at the time-stamping values and jitter (Puga, 2008). Last but not least, the network card itself has the ability to store some and upload them to the main memory in a single operation to save overhead. This has, however, the direct consequence that all messages in the buffer will be time stamped with basically the same value.

Some examples of software application layer network analyzers are the recent Capsa Network Analyzer (Colasoft, 2011), PRTG Network Monitor (Paessler) or the most popular Software Network Analyzer, Wireshark (Wireshark Developer's Guide, 2004-2010).

There are a lot of ways to improve time-stamp accuracy. One of them is to remove the time-stamping from the Application Layer to the NIC driver level, as proposed by Li Wenwei (Zhou, Cong, Lu, Deng, & Li, 2010). Some network cards even have an onboard time-stamping register to save the time of arrival of the messages, replacing the system clock timestamp.

Nevertheless, the conclusion of (Weibel & Béchaz, 2004) was assertive in respect to this subject: hardware time-stamping is the only way to go for high accuracy. Measurements on hardware can be done right after the MAC signalizes the arrival of a new message, or even between the MAC and the PHY, at the MII bus.

There are several hardware solutions available on the market, but they have the disadvantage of being very expensive. Several of these tools can be found in (Puga, 2008), a dissertation at the University of Aveiro about a hardware tool designed to sniff Fast Ethernet Data on a transmission line and that it was the starting point for the current subject developed on this document.

3.3.1. Ethernet Hardware Sniffer

In order to develop an answer to the specific demands of Real-Time Systems in communication networks, it was developed at the University of Aveiro an Ethernet Network Protocol Analyzer based on hardware. This tool was intended to analyze, in particular, the FTT-SE (Marau, Almeida, & Pedreiras, 2006) protocol also developed at this university.

The device built was a Sniffer for Fast Ethernet traffic on a transmission line. The Sniffer was able to capture data in full-duplex connections, absorbing frames in both directions of the transmission flow.

The Fast Ethernet Sniffer was built on dedicated hardware, making it immune to the multiprogramming characteristics of the operating systems, and so being better suited to precise time-stamping measurements. This work was carried out on a development board containing an FPGA programmed, which was programmed using VHDL language.

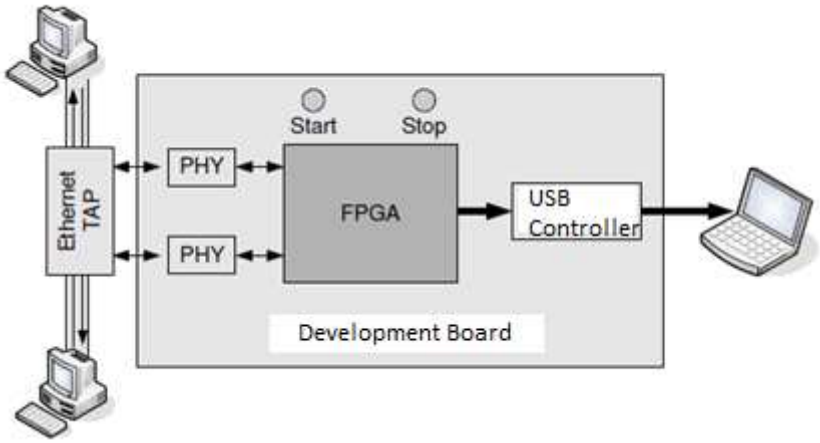


Figure 3.1 - Fast Ethernet Sniffer Architecture

The device captures the Ethernet information and stores it on Intellectual Property FIFOs. Two types of FIFOs are present on this tool: a FIFO to store the raw frames collected and another one to save temporal information. After processing all data, this is sent to the PC for analysis.

The interfaces to this machine are two Ethernet PHYs from which the Sniffer receives data, and a USB port, used to send the data to the PC side. For the reception of data on this side, a software application was developed, which must be running before the Sniffer is initiated. It is responsible for the reception of data coming from the USB port and transference to a binary file compatible with Wireshark. With this last tool, data is filtered and ordered as designed, to finally be analyzed on Octave, thanks to a script that collects the specific data desired to be analyzed.

The FEHS is a tool that allows measuring the time-stamp of messages with a 100 ns precision and a 10 ns resolution. The results obtained with this tool are far better than the ones obtained with software applications. The major shortcoming of this tool was the bandwidth: the USB connection is insufficient to leak the data captured by both Ethernet PHYs when the transmission line is heavily loaded(Puga, 2008).

CHAPTER 4. TOOL DEVELOPMENT: CAPTURING, STORING AND PROCESSING DATA. INFORMATION TRANSFERENCE TO THE PC.

4.1. INTRODUCTION

This chapter describes all the hardware of the designed tool. The approach taken is based on firstly describing the system in general and later detailing every module that composes the Sniffer. The sequence of modules described follows the trajectory of an Ethernet Message, since the reception, passing through its storage and until its transmission. The Sniffer is depicted making use of block diagrams, to show the structure of the system, timing diagrams to describe the system behavior through time, and state charts to interpret the logical behavior of the state-machines used to build some of the blocks. In this chapter are also given details about the use of the FPGA and its resources and how to properly design the hardware to comply with the FPGA restrictions.

4.2. GENERAL DESCRIPTION

In Figure 4.1, the block diagram helps to depict how the Sniffer was built. The Sniffer ultimate objective is to capture data on an Ethernet Segment at Gigabit operation on both directions of the transmission line. However, to simplify the Sniffer, this was begun to be built sniffing data only from one channel of the transmission line. Later the hardware can be duplicated, so that the Sniffer can capture data from both directions. The data is captured from a device sending out bytes from an Ethernet port and then stored and processed on a Virtex-II Pro FPGA. The connection of the FPGA with the outside is done through two Ethernet PHYs, one for the reception of data, and a second one to transfer the information from the FPGA to a PC to collect the data. Here is where the data is later analysed.

In order to keep it simple, each Ethernet frame sent to the PC by the transmitter PHY contains data from only an Ethernet frame captured. There is a unique correspondence between an Ethernet frame sent and an Ethernet frame received.

The Ethernet PHYs connect a link layer device (TEMAC) to the physical medium. The MAC

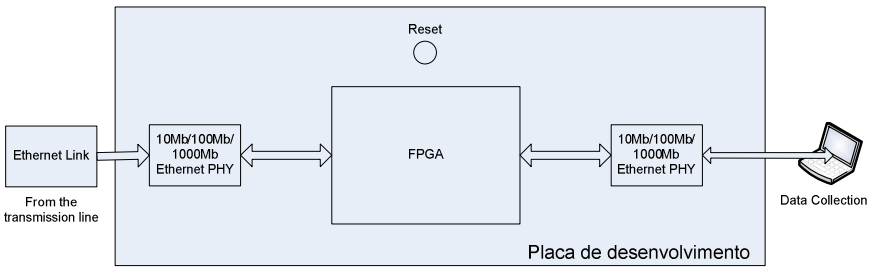


Figure 4.1 - Sniffer Architecture

device is already incorporated inside the FPGA by means of an IP Core integrated with the remaining logic. The MAC is programmed by a configuration module at the beginning to work with specific properties, including speed of operation and activation/deactivation of

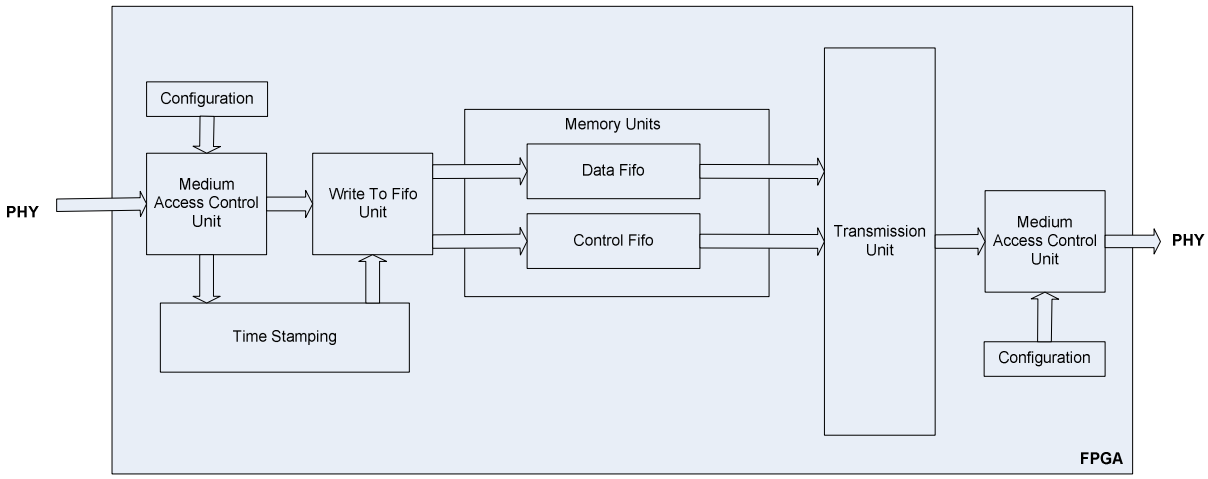


Figure 4.2 - Sniffer Internal Structure

reception/transmission. Temporary storage is done at the expense of built-in IP Core Generated FIFOs. Those are used not only to save data, but also to isolate the reception and the transmission clock domains. Two FIFOs are used on this project: a Data FIFO and a Control FIFO. The Data FIFO is responsible for saving the raw frame as captured from the transmission line, but there is also the need for saving information regarding the time stamping and the size of the received messages. That sort of information belongs to the Control FIFO. The storage process is controlled by specific modules designed to write information on the FIFOs, as the data is available. The different types of information are written in parallel to each FIFO. As soon as a message arrives it starts to be saved on the Data FIFO, and on the Control FIFO is written the instant of time in which the message arrived. The time-stamping information comes from a module designed specifically for that task, acting as a clock for the system. The fact that all process and storage information occurs at basically the same time, saves precious clock cycles that a machine relying so much on speed could not afford to lose. The basic behaviour of the system is described on Figure 4.2.

4.3. MEDIUM ACCESS CONTROL AT RECEPTION

4.3.1. RGMII INTERFACE

The medium access is performed by implementing an RGMII Interface. Like mentioned in section 2.4.2 this is a standard for communication between the Ethernet PHYs and the TEMAC devices. In an Ethernet communication, the TEMAC is responsible for high level tasks such as framing and error detection, while the PHY handles low level issues such as decoding of information and deserialization. The RGMII is the interface between the two.

Clocking at reception is achieved by means of a Clock Generator Module (see Figure 4.3). This component receives as inputs a global 125 MHz clock and signals indicating the speed at which the MAC is configured to operate. The resulting output signal is a clock with the proper frequency, which is then used on the RGMII Interface components as well as on the TEMAC.

The RGMII interface reduces the number of signals necessary to connect a PHY to a TEMAC from 16 to between 6 and 10. The data bus is a 4-bit bus, but for Gigabit mode of

operation it is capable of transmitting 1 byte every clock cycle, at the expense of using both the rising and the falling edges of the clock. Before the information is sent to the FPGA, the signals are registered in device IOBs. The complete list of signals used follows:

rgmii_rxd [3..0] – the receiver data bus.

rgmii_rx_ctl – control signal from the PHY that generates the error control signal phyemacrxer and the data valid signal for the PHY, phyemacrxdv.

rgmii_rxc – clock from PHY.

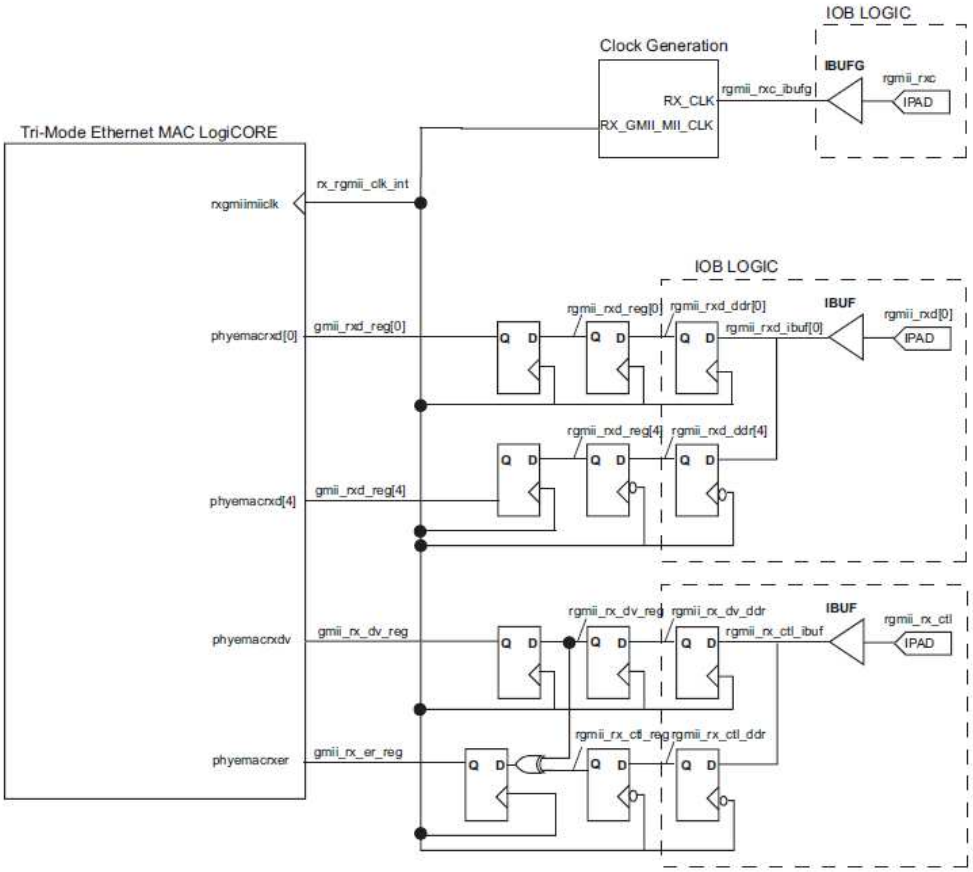


Figure 4.3 - RGMII Interface

4.3.2. CLIENT INTERFACE

Before the TEMAC and the PHY start to work, they have to be configured. In respect to the PHY, this was left with its standard configuration, as by default the PHY is already ready to operate at 1 Gbps, as intended. As for the TEMAC, a module named Configuration Unit was built with the purpose of configuring the TEMAC. This module is a state machine that operates on the rising edge of the hostclk signal. Each clock cycle, the machine writes the value on the signal hostwrdata into the register indicated by the address contained in the signal hostaddr. The write operations on the registers had the following objectives: inhibit the transmission, activate the reception and set the speed to 1 Gbps.

The signals made available by the TEMAC to the Client are the ones which allow the treatment of data. The signal emacclientrxdvld indicates that a message is being received and the corresponding data is available on an eight signal bus named emacclientrxd. The signal emacclientrxstats is also a bus, but wider, and among other things, it contains the size of the message received. The signal emacclientrxstatsvld is a control signal that indicates when the information on the bus emacclientrxstats is valid. Finally the signals emacclientrxgoodframe and emacclientrxbadframe indicate if the frame was received correctly or if the TEMAC should discard it. However, as there is an interest in knowing all the traffic that flows on the Ethernet link, these signals are never used and all data captured is processed. All signals described are synchronous with the receiver clock from the PHY, rxgmiimiiclk that operates at 125 MHz. The signal emacclientrxenable is used only when operating at lower speeds, and serves as clock enable for the receiver clock.

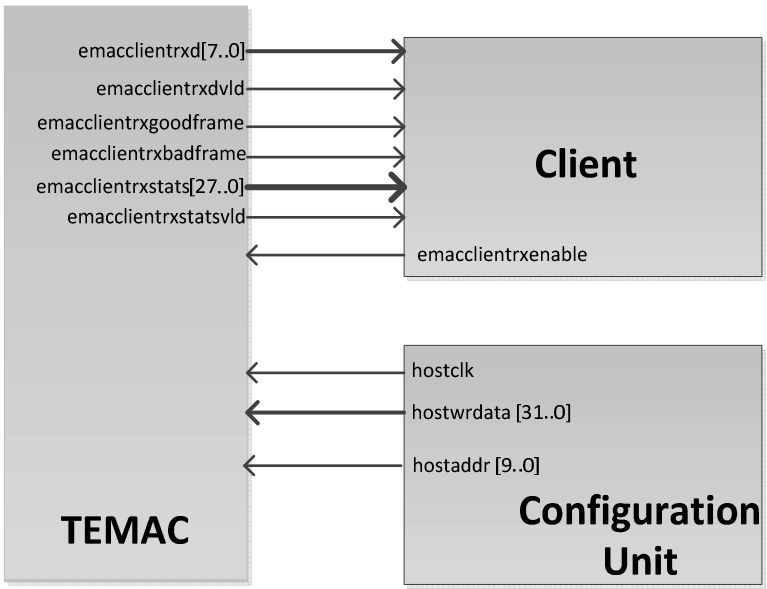


Figure 4.4 - Client Interface

4.4. TIME-STAMPING

This module was adapted from the time-stamping module designed in (Puga, 2008), with small changes on it to operate at Gigabit mode.

Two tasks are assigned to this module:

1. To act as clock for the system, by means of an incremental counter.
2. To record the arrival time of the messages in seconds and in nanoseconds (time-stamping).

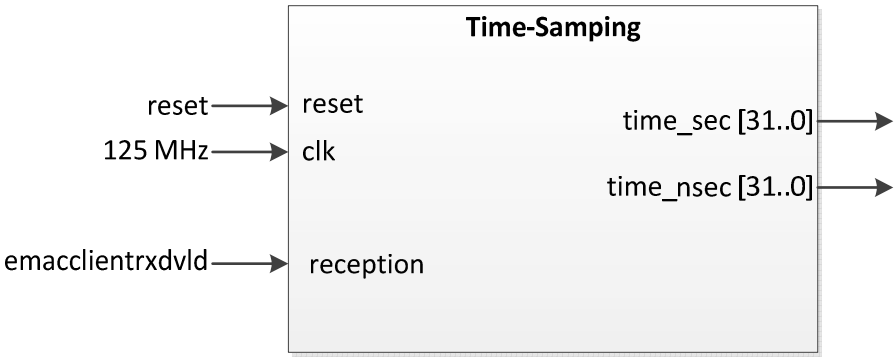


Figure 4.5 - Time-stamping Module

Implementation

The clock for this module is a 125 MHz clock, which provides a resolution of 8ns. The procedure is as follows:

- On the rising edge: a counter is incremented. This counter is composed of two bus signals: one to store the amount of nanoseconds and the other the amount of seconds, both since the beginning of the capture. The time is registered if a new message has been detected since the last rising edge.

- On the falling edge: the module checks if a new message has arrived, by means of detecting a transition from 0 to 1 in the signal emacclientrxdvld.

The worst case scenario is depicted in the Figure 4.6, and it happens when the signal emacclientrxdvld indicates the beginning of a new message right after a falling edge of the clock (1). In that case the signal is only detected by the module on the next falling edge (2) and the time stamp is going to occur only on the subsequent rising edge of the clock (3), resulting in a maximum error of 12 ns. On the other hand, though, if the message arrives right before the first falling edge, then the error is of 4ns, being this the best case scenario. The jitter introduced solely by this module is then 8ns.

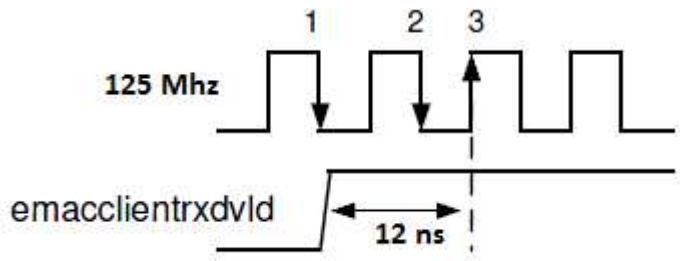


Figure 4.6 - Worst case scenario for time-stamping

4.5. FIFOs

There are two different memories on the Sniffer.

The first kind of memory is the Data Memory, to which the complete Ethernet Frame captured is sent. In the case of lack of space in the memory, the storage is interrupted, and the data stored is a truncate version of the original frame. On the limit, if the memory is completely full there is no data capture at all. However, there is always interest in capture some basic data that can identify the message captured. So instead of capturing the entire frame for all the arriving messages, the designer has the possibility to limit the capture to a specific number of bytes, increasing the number of frames for which there is data stored. If problems of Data memory space occur, the choice for the maximum amount of bytes will always be a trade of between the amount of data stored per message, and the number of messages captured.

As for the Control Memory, it exists to save information details about the Ethernet Frame captured, rather than the actual content of the frame. For each message captured, the Sniffer saves always 12 bytes of information divided on the following fields:

- Time stamping – the moment of arrival of the message, with 4 bytes to save the number of seconds, plus 4 bytes to save the number of nanoseconds.

- The total amount of bytes captured – 2 bytes to save the number of bytes captured by the Sniffer;

- The size of the message – 2 bytes to save the number of bytes of the original frame received by the TEMAC. This value can be different from the one above in case the Data memory has reached its limit or when the designer limits the amount of bytes per message captured to a specific limit;

If the Control Memory gets full, the Sniffer ends the capture completely, even if there is space available on the Data Memory. The reason for this is that the most important information that a Sniffer can provide is the arrival of a message and the instant of time in which that occurred. When the Sniffer can no longer register this kind of information, is pointless to store the raw Ethernet frame, because there would be no way to interpret the data on the PC's side. The Control Memory could get full when the rate at which data is captured at the reception is higher than the rate at which is drained to the PC. If the Control Memory becomes full, the capture is not reinitiated, because the user would not have knowledge of lost frames.

Implementation

The memories chosen for this project were block RAM FIFOs (First-in-first-out) IP Cores generated by Xilinx Core Generator (Xilinx, 2008). The FIFOs were generated with the option for independent clock domains, so there are two clocks, the rd_clk and the wr_clk, each one to control read and write actions to the FIFO. This eliminates any synchronization problem as the read and write operations become completely independent from each other.

The FIFOs also allow having different sizes for the input and output data buses, which comes in hand for the specific case of the Control FIFO, where the information that needs to be

sent there goes in words of 32-bits, but the TEMAC input data bus is an 8-bit bus. In the case of the Data FIFO this does not occur, since the FIFO is in between two TEMACs, so both the input and output buses have an 8-bit size. Both FIFOs have the capacity to store 16384 bytes of information. This corresponds approximately to 256 messages of minimum size and to 10 messages of maximum size, on the Data FIFO. As for the Control FIFO, it allows writing information regarding 1365 messages into the FIFO.

As it will later be described on this document, the transmission of a frame occurs only when there is at least one complete Ethernet Frame captured stored on the Control FIFO. For this reason, the EMPTY FLAG from the FIFO is not used when deciding on a read operation. Rather, it was added a 14-bit bus called Read Data Count that indicates the number of bytes available on the FIFO to be read at a given time. This allows checking if the number of bytes is enough or not to include an entire Ethernet Frame.

Last but not least, the FIFOs were generated with the option First-Word-Fall-Through (FWFT). This option makes the next word to be read from the FIFO to be already on the output data bus, without issuing a read operation. As it will be seen later in this document, this characteristic of the FIFOs will prove to be very important for the performance of the Sniffer. All options taken when generating the FIFOs can be seen in Chapter 5 at the end of this document.

4.6. WRITING TO THE FIFOS

The figure below is a block diagram that shows how the Ethernet messages are stored. In it, it can be seen: the Time-Stamping Unit signals and the FIFOs, a multiplexer that is controlling which information is being written into the Control FIFO, and two blocks that correspond to state machines which control the writings on the FIFOs:

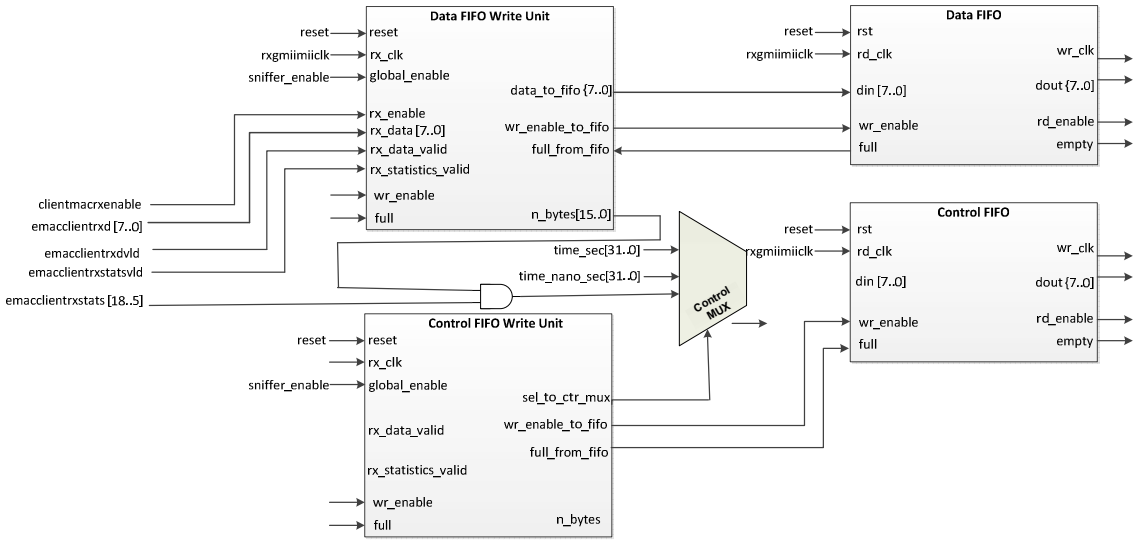


Figure 4.7 - Block Diagram for the reception and storage of the messages

4.6.1. WRITING TO DATA FIFO

The Data Fifo Write Unit is the block responsible for placing the Ethernet frames received from the TEMAC into the Data FIFO. The Sniffer was designed to operate at 1Gbps, but the state-machines were conceived to handle all speeds, so that it can be easier to adapt the Sniffer for whoever needs it to capture data at different rates. The TEMAC provides the data to the client in groups of 8-bits, but if working at lower speeds than 1 Gbps, the TEMAC output data bus is updated with valid data only every two clock cycles. As it is possible to limit the amount of information captured by Ethernet Frame, this module is also responsible to count the number of bytes captured, storing it on the signal n_bytes. This signal will then be forwarded to the Control FIFO Write Unit, for control information storage.

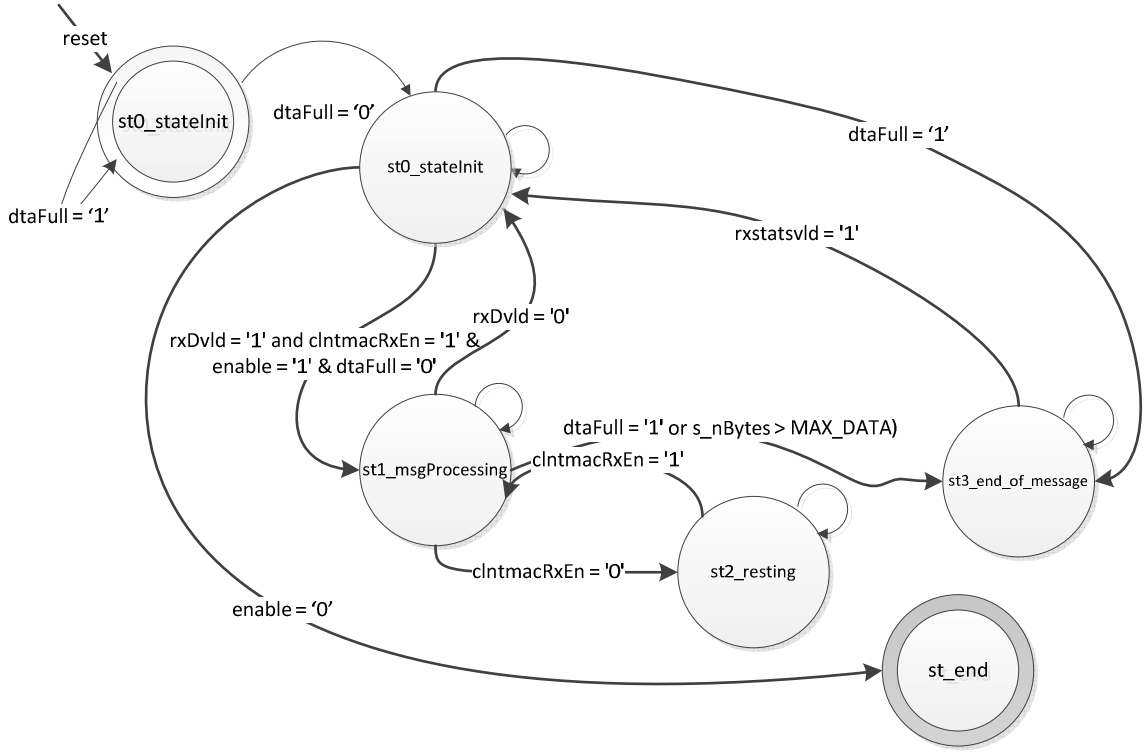


Figure 4.8 - Data Fifo Control Unit state chart diagram

Implementation

The Unit is a Moore State Machine composed of 6 states: st_pre_initial, st0_stateInit, st1_msgProcessing, st2_resting, st3_end_of_message and st4_end. State changes happen on the falling edge of the signal rxgmiimiiclk and actual writings on the FIFO occur on the subsequent rising edge.

When the machine is at the initial state, st0_stateInit, the unit is waiting for a new message to arrive, event that is triggered by the signal rx_data_valid (originally clientmacrxdvld from TEMAC). When this occurs, the machine goes to the state st1_msgProcessing. Here, the data coming from the TEMAC is copied to the output bus of the unit and writing on the Data FIFO is enabled. To bear operations at slower speeds, the signal clntmacRxEn (same signal as emacclientrxen from Section 4.3.2) serves as an enabling signal that must be checked before accepting the data on the TEMAC client output data bus. In such cases, the machine can't be permanently writing into the Data FIFO, and will then alternate st1_msgProcessing with an idle state, st2_resting, depending on the value of clntmacRxEn (data is updated when this signal is HIGH).Also, it is possible to store incomplete messages into the Data FIFO, whether it's because the FIFO is full (dtaFull = '1'), or it's

the user's wish to truncate the message to save space on the FIFO. In either case the state machine will transit to the state st3_end_of_message, from which can only get out when the TEMAC has stopped receiving data (that situation is certain when the signal rx_statistics_valid is set to HIGH). In order for the client not to lose track of the amount of bytes captured, a signal is incremented once for every byte written into the FIFO. Finally, if the capture is stopped, the machine will go to an end state, st4_end, but a transition to this state is only possible from the initial state, meaning that if the capture is at any time finished, the unit will still write the message being currently received, until the TEMAC is no longer receiving data regarding that message. Only a reset can make the machine to leave this state.

When there is a reset pulse, the FIFO generated by the *Xilinx Core Generator* enters its own reset state from which can only leave after three clock cycles. During this time, the FIFO FULL flag is asserted, to ensure no writing operations occur during the FIFO reset state. All states from the Data Fifo Write Unit, including the state st0_statelnit, check if this flag is asserted, to end the capture in case the FIFO is already full. This would result in the state-machine to always transition to the st_end_of_message state upon a reset, failing to capture any data after the first reset. Rather, an st_pre_initial_state was built to ensure that the transmission of a message after a reset is done only when the FIFO is actually ready to receive bytes.

4.6.2. WRITING TO THE CONTROL FIFO

The *Control FIFO Write Unit* is the block responsible for writing information into the Control FIFO. The objective of this block is to write the information concerning the time stamping and size of the message received. This is accomplished controlling the three entries of the Control Multiplexer: time in seconds, time in nanoseconds and a bus signal containing the real size of the message (on the first 16 bits) and the number of bytes captured by the Sniffer (on the last 16 bits).

Implementation

The unit is built as a Moore State Machine composed of seven different states: st_pre_initial,st_initial, st_time_seconds, st_time_nano_seconds, st_idle, st_size and st_end. State changes happen on the falling edge of the rxgmiimiiclk and actual writings on the FIFO occur on the subsequent rising edge.

The machine leaves the initial state, st_initial when the signal rx_data_valid (originally emacclientrxdvld from TEMAC) triggers the beginning of the arrival of a new message and the consequent writing of another control message in the Control FIFO, if capture hasn't stopped yet, that is, if global_enable = '1' (control signal managed by the user), and if the FIFO is not full. On states st_time_seconds and st_time_nano_seconds stamping information, in seconds and nanoseconds respectively, is sent into the FIFO. On state st_size the information written concerns the size of the message, information that is only available when the message has been fully captured at the reception, event triggered by the signal rx_statistics_valid. While this does not happen, the state machine remains on state st_idle, where writing into the FIFO is not enabled. Overwriting in a full FIFO has to be avoided, so it is possible to jump to the st_end state from any other state in case full FIFO condition is detected (full_from_fifo = '1'). When st_end is reached, writing on the Control FIFO is no longer possible and the only way to leave that state is if the user resets the Sniffer.

Similarly to what happened with the Data FIFO Write Unit, when there is a reset pulse, the FIFO generated by the *Xilinx Core Generator* enters its own reset state from which can only leave after three clock cycles. During this time, the FIFO FULL flag is asserted, to ensure no writing operations occur during the FIFO reset state. As mentioned above, all states from the *Control Fifo*

Write Unit module check if this flag is asserted, to end the capture in case the FIFO is already full. This would result in the state-machine to always transition to the st_end state upon a reset, failing

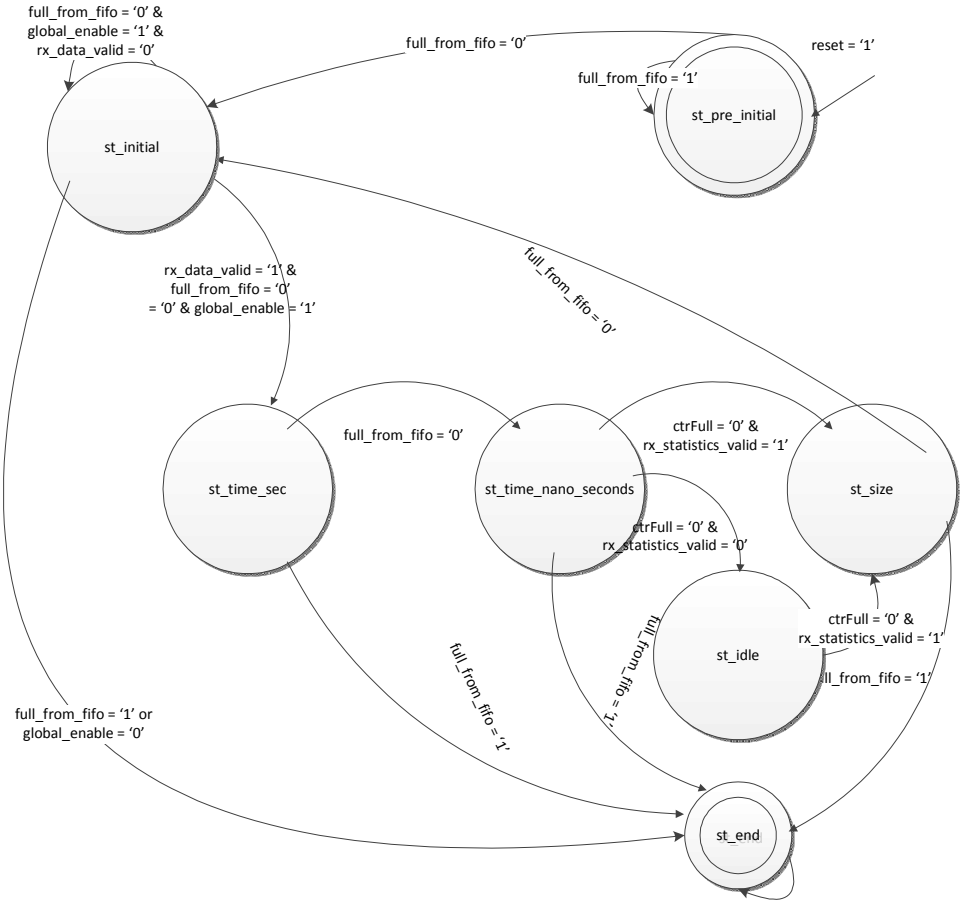


Figure 4.9 – Control Fifo Write Unit State-Chart Diagram

to capture any message after the first reset. Rather, an st_pre_initial_state was built to ensure that the transmission of a message after a reset is done only when the FIFO is actually ready to receive bytes.

4.7. FPGA TO PC INFORMATION TRANSFERENCE

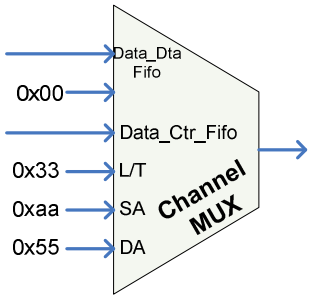
After the information has been received, processed and stored at the FIFOs, it is then necessary to transfer it to the PC, so that data can be analysed.

As mentioned before, the communication technology chosen for extracting the information out of the FPGA was the Gigabit Ethernet. This means that it is the Tri-Mac Ethernet MAC Core from Xilinx again the tool used to serve as the interface between the Sniffer logic and the outside.

4.7.1. READING FROM THE FIFOS AND SENDING THE INFORMATION COLLECTED

The Channel Multiplexer and the Ethernet Frame Fields

As the entry names indicate, the first three entries of the Channel Multiplexer are directly related with specific fields of the Ethernet frame. Attached to each of these three entries is a specific 8-bit word, chosen by the designer. The data on these Ethernet frame fields will then be a repetition of the respective byte associated to each of the entries. Each of the words is repeated for as many times necessary to fulfil each field size. For instance, if on the Channel Multiplexer we would have the following words at the mux inputs:



On the output, the first bytes of every frame sent would be:

0x 55 55 55 55 55 55 aaaaaaaaaaaa 33 33

The first six bytes would correspond to a six times (Destination Address field has a size of six bytes) repetition of the Word 0x 55, the following six bytes would correspond to a six times repetition of the word 0x aa (Source Address field) and the remaining two bytes would correspond to a repetition of two times of the word 0x 33 (Type/Length field has a size of two bytes).

The reason why this scheme was chosen was because of simplicity. There is no need for extra logic, nor extra processing, and because the FPGA PHY is directly connected to the PC NIC, it doesn't matter which DA or SA are selected. The Ethernet Frame will always be detected on the computer's side.

Figure 4.10 - The Channel Multiplexer

The information transference from the FPGA to the PC consists in taking the data from the Data and Control FIFOs, wrap it on an Ethernet Frame and send it to the PC via an Ethernet cable.

There are many sources of data stored and only one way out (the TEMAC) so it is necessary to select the proper source of data from which to take information at a given time to be sent to the TEMAC transmitter. This is done by the *Channel Multiplexer*, a multiplexer containing five entries from which it can be transferred data concerning, respectively, the *Destination Address*[DA], the *Source Address*[SA], the *Type/Length*[T/L], data from Control Fifo, two bytes containing zeros and data from Data Fifo (see Figure 4.10 for more details).

When transmitting frames, the TEMAC makes use of an enabling signal, tx_data_valid, to delimit the window of time in which the TEMAC is transferring whatever information is present on its data input bus to the outside. The beginning and ending of an Ethernet Frame is actually denoted by the assertion or deassertion of this signal. That means that the signal tx_data_valid must be hold high during the entire frame transmission and as a consequence every single frame byte must be ready to be transferred continuously. This would suggest that the data multiplexing should be done first to a third FIFO, existent only to get data in proper order and easier to transfer. However, as speed is a key element of this project, this solution was skipped and the

transmission is achieved without an extra FIFO and relying on a state machine that performs all transmission tasks without a middle step.

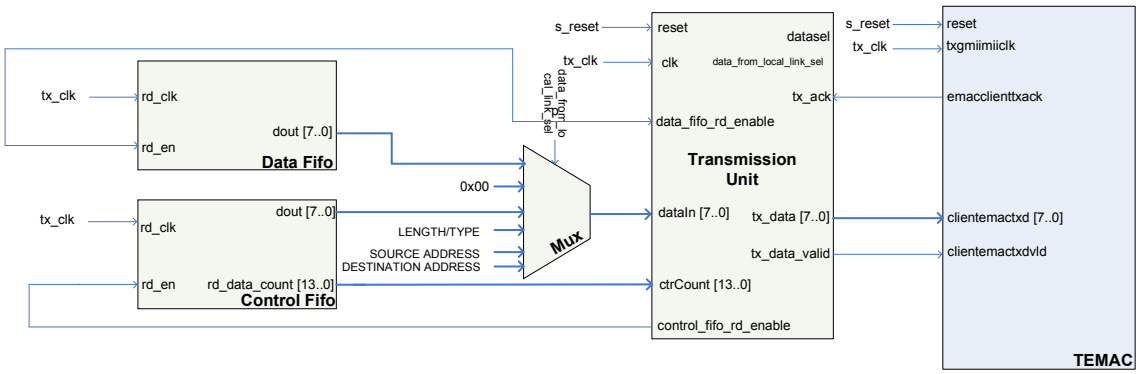


Figure 4.11 - Information transference to the PC Block Diagram

4.7.2. TRANSMISSION UNIT

The Transmission Unit is the module responsible for controlling all the transmission data flow. It is a finite-state machine responsible for controlling the Channel Multiplexer Selector, and for the control signals from both the FIFOs and the TEMAC.

As the Ethernet technology is being used on both the reception and the transmission, what will be seen on the computer's side is the ethernet frame captured at the reception embedded in a bigger ethernet frame, along with time-stamping information (see Figure 5.1). To avoid errors and because the ethernet packets transmitted need to be sent continuously, the bigger ethernet frame will only be sent when the ethernet frame at the reception is fully captured. This way, transmission will start only when there are 12 or more bytes at the Control FIFO. Why? Because 12 bytes is the amount of bytes of control data stored for a complete ethernet message, and the last information to be stored on a FIFO regarding a receiving frame is the size of the frame, which is stored on the control FIFO. So when that FIFO has 12 bytes, it is certain that there is a complete ethernet frame captured.

The message begins to be transmitted by assertion of the tx_data_valid signal, as mentioned before, but the simple assertion of this signal does not mean that the TEMAC is ready to accept data. Contrarily, the Transmission Unit has to wait for the handshaking signal from the TEMAC, emacclienttxack before transmitting the next bytes. That signal indicates that the TEMAC has received the first frame byte, and that is ready, on the immediate subsequent clock cycle, to receive the very next byte of data.

After transmitting the field bytes correspondent to the Destination Address, Source Address and Type/Length frame fields, the Transmission Unit then proceeds to transmit information stored on the Control and Data FIFOs. From the control FIFO is read: the timing information, that is, the time-stamping in seconds (4 bytes) and nanoseconds (4 bytes); the

number of bytes captured (2 bytes) and finally the size of the message (2 bytes). From the Data FIFO are transferred all the bytes captured associated with that message. For that to be possible, it is necessary to save the value of the number of bytes captured to a register, during the reading of that field from the Control FIFO.

All data concerning the ethernet message captured is sent in the format in which it could be saved in a wireshark file, because it may be useful to analyze the data on this network analyzer and what is seen by Wireshark when data is captured is the bigger ethernet frame (see section **Error! Reference source not found.** - **Error! Reference source not found.** of this document). The only differences with respect to the original frame captured are the *number of bytes captured* and the *size* of the message. For this reason, two bytes filled with zeros are sent just before each of these fields. That is the reason for a 0x00 entry on the Channel FIFO Multiplexer.

The different tasks carried out by the Transmission Unit must occur in different instants on time, in order to ensure proper operation of the Sniffer. Two different key moments are needed on transmission:

- A moment to evaluate the current state of the system and to act on the MUX selectors, the FIFO reading control signals, and the TEMAC transmitting control signals;
- A moment for the TEMAC to collect the data to transmit;

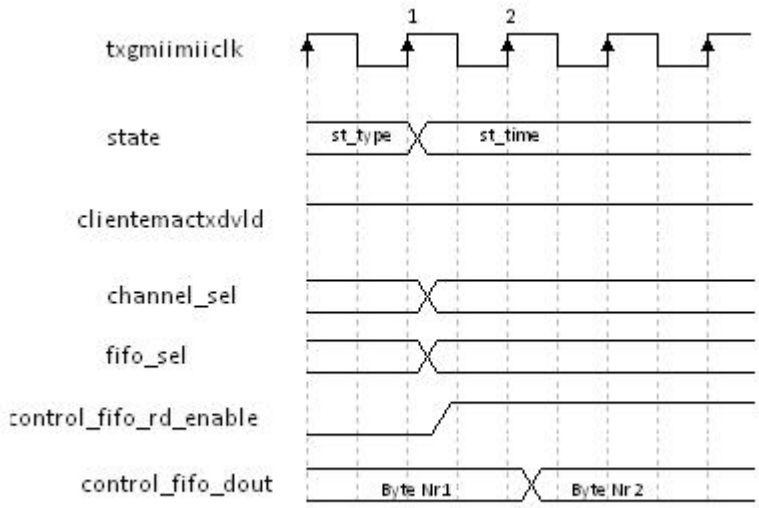


Figure 4.12 - Transmission Unit Timing Diagram

At first sight, the above could indicate the usage of different clocks or edges of the same clock for the different logic, but in fact the same clock is used throughout all different components used for transmission. So the FIFOs read clocks, the Transmission Unit clock and the TEMAC transmission clock are all the same and they are synchronous with the rising edge of that clock, tx_clk, coming from a Clock Generator Module. The reason for using the same clock and the same clock edge for all components is that when using the fastest clock available on the FPGA (125 MHz), half clock cycle is not enough to perform some of the tasks that the Transmission Unit needs to carry out. In order for the logic to meet the timing constraints specified in the design, the

changes in the control signals, as well as a reading operation on the FIFO must have at least an entire clock cycle of margin to properly update the data buses.

In order for the system to work on the timing restrictions above mentioned, the FIFOs had to be configured to operate in the First-Word-Fall-Through (FWFT) mode. This feature provides the next word to be read from the FIFO to be already available on the output data bus, allowing catching the word before issuing a read operation. Without this mode of operation, unnecessary complexity would be required for the state-machine to work properly: the data would be valid only one clock cycle later, but there is no way to pause the TEMAC while transmitting a frame.

The mode of operation is then described on the Figure 4.12. The figure represents the moment in which the Moore State Machine transitions from the state st_type, a state in which the bytes correspondent to the type/length are being transmitted, to the state st_time, when the bytes concerning the time-stamping information are transferred. On the rising edge 1 the state transition happens, and as the timing information is stored on the Control FIFO, a reading operation has to occur. For that reason, the control signals from the multiplexers are changed accordingly and the enable signal to read from the FIFO, control_fifo_rd_enable, is asserted. Also on the rising edge 1, and because on that very moment the state is still st_type, none of the signals changes have occurred yet, so the TEMAC (that is also synchronous with the tx_clk) reads the byte on the input data bus correspondent to last state, st_type. On the next clock cycle, the byte number 1 on the Control FIFO is already on the FIFO output data bus as the FIFO is configured on the FWFT mode of operation. Because the Control FIFO is synchronous with the signal tx_clk too, it will take time also to change the FIFO output data bus to the next byte, and so the data on rising edge 2 will be correct. As it can be seen from the timing diagram, when the machine changes state, the correspondent byte is only written on the next rising edge, just before the state changes again. This phenomenon is drag from when the MAC sends the emacclienttxack (Figure 4.13.) This signal can only be seen by the Transmission Unit on the next rising edge. Then and only then the Transmission Unit can perform the proper changes, but the TEMAC is already ready to accept a new byte of data. This situation results in that the machine will have to stay in the st_dest_address state on purpose one less time than common sense would point, originating that the write operations subsequent to this state will always seem advanced in respect to the state.

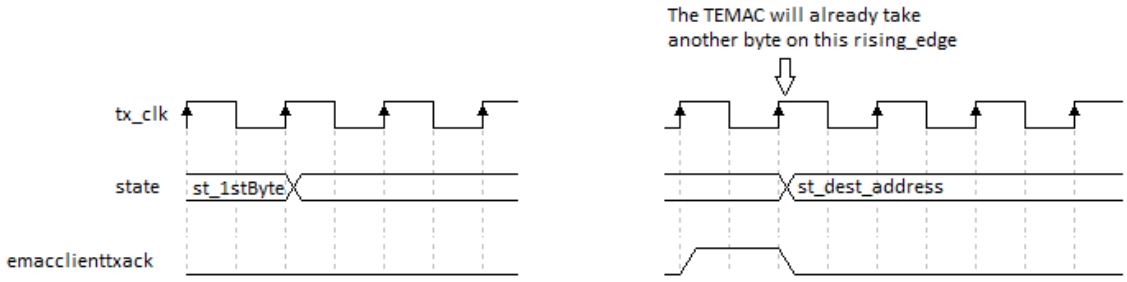


Figure 4.13 - Acknowledge Signal from the TEMAC

CHAPTER 5. SOFTWARE TOOLS FOR DATA ANALYSIS

5.1. RECEIVING DATA ON THE PC SIDE

As the interface to the computer is also a PHY, data is sent also recurring to a TEMAC. As seen in the previous chapter, this means that the expecting data on the PC side is also in the form of Ethernet Frames. As mentioned in the Section **Error! Reference source not found.** of this document, Wireshark is the most popular software available to analyze network traffic. Wireshark possesses several features, from which are relevant to this project:

- Being able to decode hundreds of protocols, including 802.3, grouping the data received on recognizable fields according to this standard.
- Capture is performed live, but it is possible to analyze data offline. All captures made with Wireshark can be saved into a file for later analysis.
- Capture in promiscuous mode is available, meaning that all data on the Ethernet Link can be sniffed, not just traffic addressed to that interface.
- It allows the application of filters, displaying or hiding messages according to a variety of parameters such as protocol or Ethernet Source or Destination Addresses. This comes in hand, as the Sniffer does not perform any filtering on hardware level. It could be relevant to do so only in what bandwidth saving says respect, but Wireshark is enough for showing only the data sent by the FPGA, and hiding handshaking and protocol network messages.
- It is possible to sort the messages according to the time-stamp.
- It provides the user the ability of exporting the data to various formats, including CSV format and plain text for analysis on math tools.

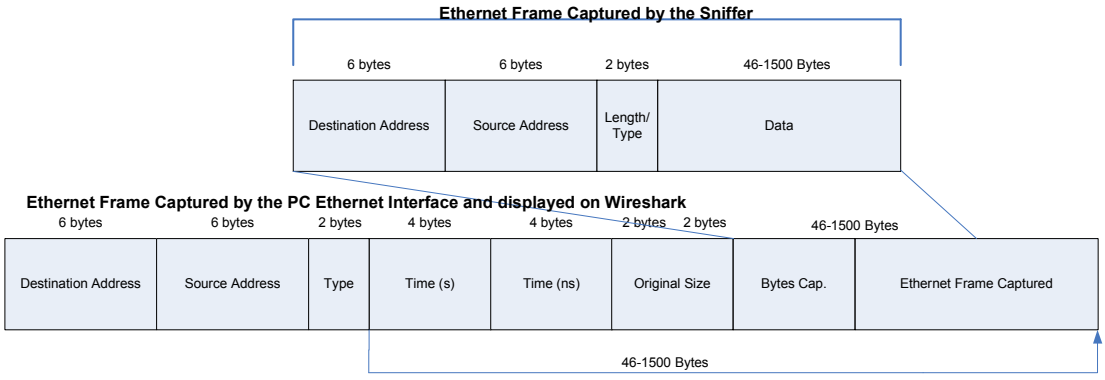


Figure 5.1 - Ethernet Frame received on the PC side

Figure 5.2 shows how a capture looks like in Wireshark with a filter set to sniff only the messages with a Destination Address equal to the one the messages coming from the FPGA have.

However, recall from the previous Chapter that the data received on the PC side is an Ethernet Frame encapsulated on a bigger Ethernet Frame (see Figure 5.1), and this last one is not the one we wish to analyze. Figure 5.2 also shows the packet as received on the PC side.

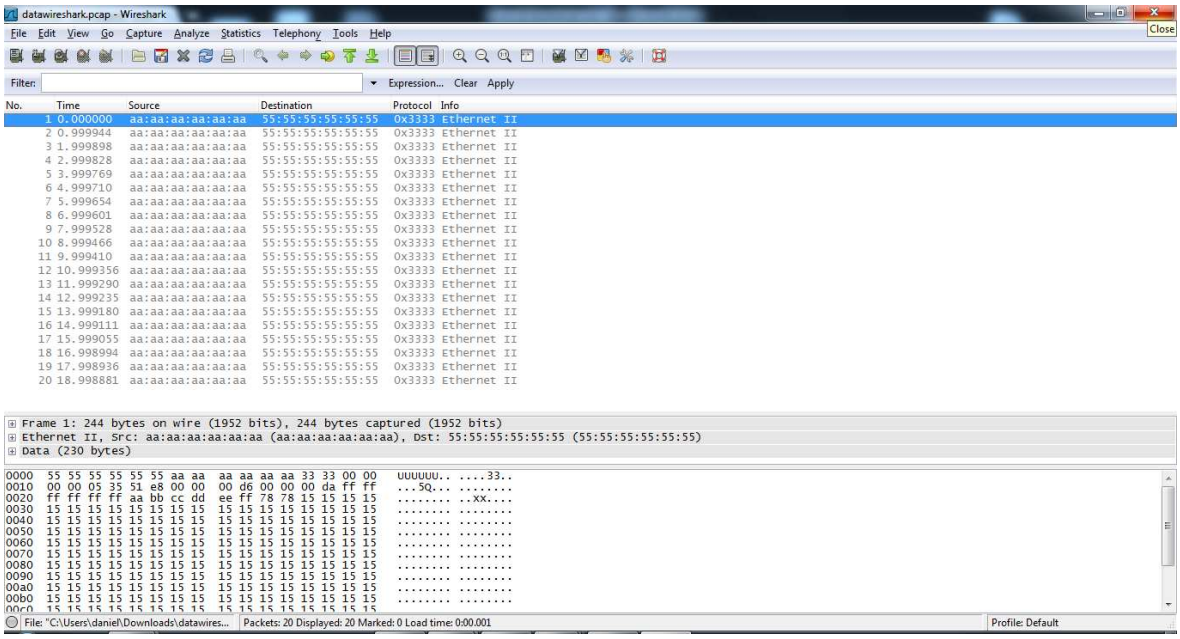


Figure 5.2 - Wireshark Capture

It is of no interest, for instance, to know the software registered time-stamping of the Ethernet Message sent by the transmitter TEMAC on the FPGA, but rather the smaller Ethernet Frame that was captured and processed by the Sniffer. This creates a problem, since Wireshark doesn't recognize this format, and so it interprets the smaller Ethernet Frame as a single field: the data field of the actual Ethernet received. To overcome this situation, two software applications were developed to extract the relevant portions of the smaller ethernet frame captured.

5.2. DATA PROCESSING

Even if Wireshark was able to decode the Ethernet Frame special format in which the data packets are received on the PC side, this tool would not be enough to perform the kind of analysis that is intended for the Sniffer. Although Wireshark is a very useful tool for a general visualization of what happens in the network, it does not have the features of a math application. It is of interest to perform mathematical operations to the data and to analyze it graphically, making space for probability and statistics, as well as worst case scenario analysis.

As mentioned before, Wireshark has the ability to export data to a variety of different file formats, including CSV that can be read on mathematical tools. However, it is obvious that if Wireshark interprets the entire encapsulated frame as a single field corresponding to the data field of the actual Ethernet Frame received, when exporting to CSV file, the different fields of the smaller frame are going to be concatenated on a single value. So to overcome this problem, the data is first exported to a plain text file. Wireshark gives the possibility to export only the packet content, without the packet summary or any details. This way, this is how the information appears after being exported:

```
0000   55 55 55 55 55 55 aa aa aa aa aa aa 33 33 00 00   UUUUUU......33..
0010   00 00 05 35 51 e8 00 00 00 d6 00 00 00 da ff ff   ...5Q...........
0020   ff ff ff ff aa bb cc dd ee ff 78 78 15 15 15 15   ..........xx....
0030   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
0040   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
0050   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
0060   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
0070   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
0080   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
0090   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
00a0   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
00b0   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
00c0   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
00d0   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
00e0   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
00f0   15 15 15 15                                       ....

0000   55 55 55 55 55 55 aa aa aa aa aa aa 33 33 00 00   UUUUUU......33..
0010   00 00 05 34 61 68 00 00 00 d6 00 00 00 da ff ff   ...4ah..........
0020   ff ff ff ff aa bb cc dd ee ff 78 78 15 15 15 15   ..........xx....
0030   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
0040   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
0050   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
0060   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
0070   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
0080   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
0090   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
00a0   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
00b0   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
00c0   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
00d0   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
00e0   15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15   ................
00f0   15 15 15 15                                       ....

0000   55 55 55 55 55 55 aa aa aa aa aa aa 33 33 00 00   UUUUUU......33..
```

(...)

Figure 5.3 - Information exported from Wireshark to a plain text

Then a C application called convert2csv.c was developed. This application has as input a plain text file as the one from Figure 5.3 and takes the string that contains the byte with the time-stamping information. The data read from the file are ASCII characters that represent 2 hexadecimal values, the instant of arrival of the message in seconds and nanoseconds, respectively. So the program needs to then convert those values into integers. Finally the application writes the timing values collected and writes them into an output file in the CSV format, as it can be seen from Figure 5.4. In this format, data can be easily analyzed by math tools like Microsoft Excel or Matlab.

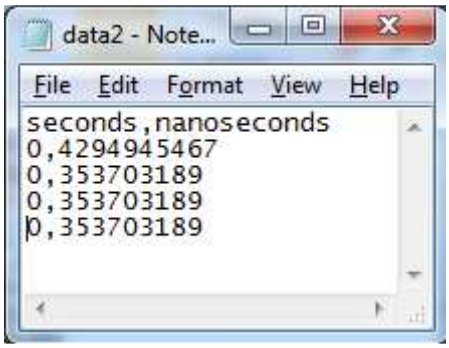


Figure 5.4 - Data in CSV format

CHAPTER 6. TESTS AND RESULTS

6.1. INTRODUCTION

In order to evaluate the performance of the Sniffer, the tool was put under specific tests. These had the objective of verifying the Sniffer's two desired abilities: to be able to capture the Ethernet frames on the Ethernet link, and to register the time-stampings correctly and accurately.

6.2. WORKING DIAGRAM

In order to create the conditions to test the Sniffer, there was the need to have a device inserting Ethernet traffic on an Ethernet link. As explained in section 3.3 if a general purpose computer would be used as the source of such traffic, the multiprogramming and resource sharing issues associated with these devices would insert errors and thus would irreversibly corrupt the measurements taken. The accuracy of the results would be certainly low, but there would be no way of knowing if the source of that would be the computer or both the computer and the Sniffer. To overcome such problem, another tool was developed in VHDL language, to serve as a packet generator. With this device, the user is able to set the size of the messages, as well as the period. As this is another hardware dedicated tool, it is immune to the issues of the general purpose computers just mentioned, and thus can be trusted as a reliable source of packet generation without time deviations. To run this tool, another NetFPGA was used and connected to the brother board were the Sniffer was running. Figure 6.1 depicts the situation just described.

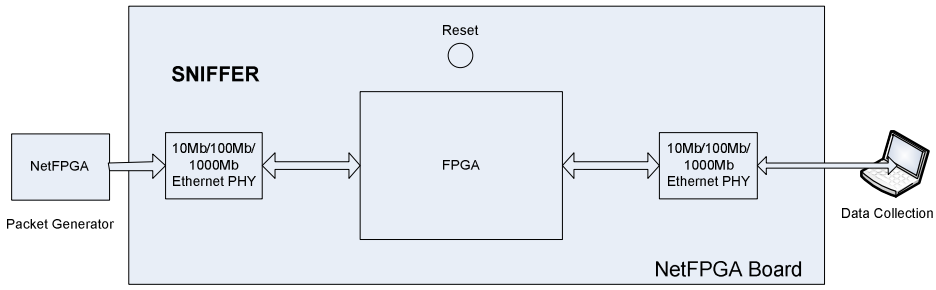


Figure 6.1 - Sniffer Testing Working Diagram

Period (ns)	Size (Bytes)	Maximum (ns)	Minimum (ns)	Mean (ns)	Standard Deviation	Jitter (ns)
10000	60	10192	10176	10184	1.22	16
10000	100	10188	10072	10080	1.22	16
50000	60	50088	50072	50079	4.04	16
50000	100	50088	50072	50079	4.41	16

Table 6.1 - Results

Several measurements were taken to evaluate the performance of the Sniffer.

Table 6.1 contains some of these measurements. Each row of the table represents the results taken for 20000 packets from the packet generator, with two parameters varying: the period and the size. For all measurements taken all the 20000 packets were successfully captured. In what concerns the time-stamping, recall from section 4.4 that the Sniffer resolution is 8 ns. However, the jitter measured was 16 ns, the double. This is due to the lack of synchronization of the clocks used for reception and time-stamping that causes sometimes the time-stamping being taken two clock cycles later. However, the Sniffer reveals great accuracy on the measurements, as the mean value and the standard deviations indicate. The figures below are the graphic representation of the value of the delay between messages for the different measurements. As it can be seen the messages converge firmly to the period value, regardless of the measurement variables.

Figure 6.1 - Results: Period = 10000; Size: 60 bytes

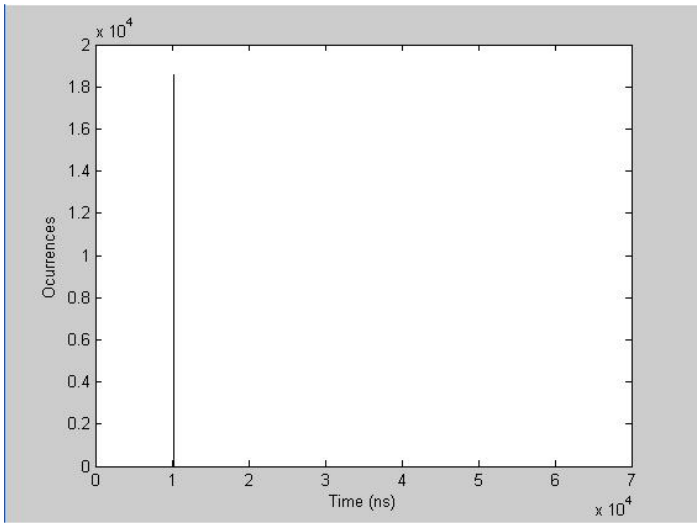


Figure 6.2 - Results: Period = 10000; Size: 100 bytes

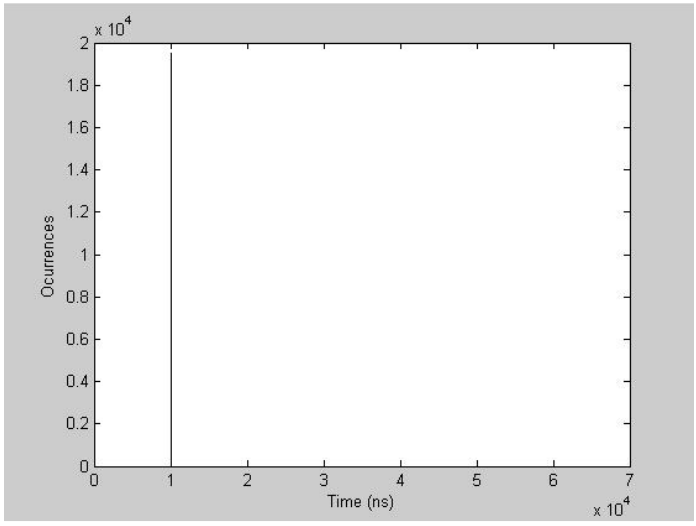


Figure 6.3 - Results: Period = 50000; Size: 60 bytes

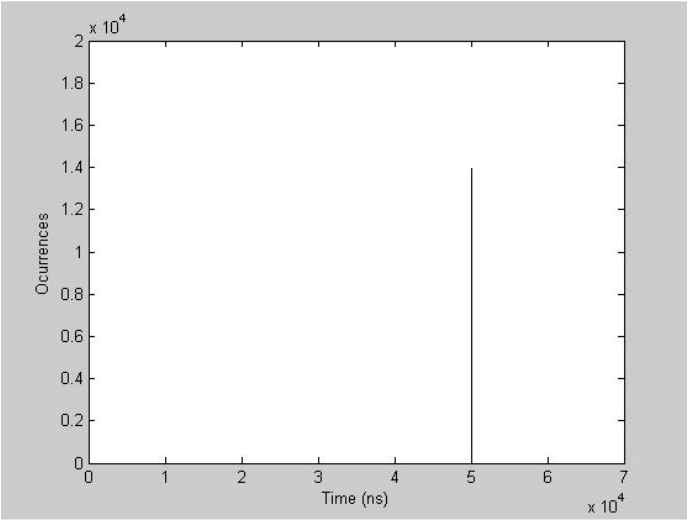
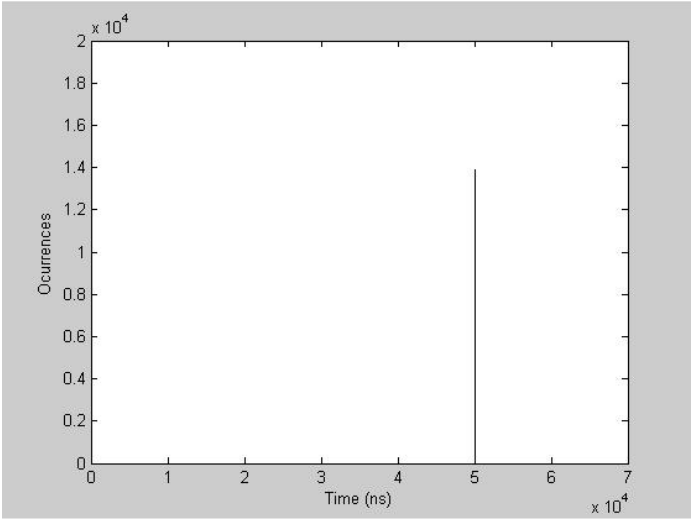


Figure 6.4 - Results: Period = 50000; Size: 60 bytes



CHAPTER 7. CONCLUSIONS

7.1. SUMMARY

In this work it was analysed the insufficiency of vulgar software network analyzers in the face of strict temporal restrictions imposed by Real-Time systems on a distributed architecture. Additionally, the need for higher bandwidth protocols on the industrial area and the consequent attention to Ethernet as a suitable solution was also focused on this project. As an answer to these issues, it was developed capable of sniffing Gigabit Ethernet traffic flowing in an Ethernet Link. The device built is capable of capturing the Ethernet Messages flowing on the transmission line with an 8ns resolution and a jitter of 16ns.

The Gigabit Ethernet Sniffer connects with the physical transmission line at the expense of two Gigabit Ethernet PHYs. One of these PHYs captures the traffic flowing on the Ethernet Link inspected, and the second one connects the device to a PC. The rest of the messages route happens inside an FPGA that communicates with the Ethernet PHYs by means of a Xilinx Intellectual Property Core MAC device, the TEMAC. Message contents and control information data are then stored into FIFOs - IP Cores as well – thanks to finite state machines that perform controlled writing operations on the FIFOs. Finally the Ethernet Message finds her way out of the FPGA again through a TEMAC, being this process supervised by another state machine, which performs reading operations on the FIFOs and multiplexes the information sent to the TEMAC device.

As the Ethernet is the technology used both to receive and to transmit the information, this last one arrives at the PC side on the form of small Ethernet frames embedded in bigger ones. The popular network analyzer Wireshark is good to filter the relevant packets for analysis, but it can't decode the protocol used to send the information to the PC, neither can perform graphical and mathematical analysis. So a software tool was designed to extract the relevant fields of information coming on the Wireshark files and to put it on a CSV format. The information is then treated by math tools which perform a complete analysis of the traffic flowing on the transmission line.

7.2. FUTURE WORK

7.2.1. TWO CHANNELS

For the Sniffer to be complete, it must capture data on both sides of the communication line. The intended operation for the Sniffer is to interface it with a transmission line through a device named TAP, commonly known as a splitter. This way the Sniffer would be able to capture all data on an Ethernet Segment without interfering with the normal flow of communication. Right now it would be possible to duplicate the Sniffer and capture data on both directions, but this would demand the use of two NetFPGAs and two computers to receive the data, or at least a computer with two Ethernet Gigabit Ports. But even on that situation, the channels would not have the same temporal basis, so it would not be possible to compare the timing information collected.

Adapting the Sniffer for two channels is not complicated, as all there is to be done is to duplicate the reception logic and then multiplex the information at the transmission side. This has already started to be implemented on this work, but still without a working version.

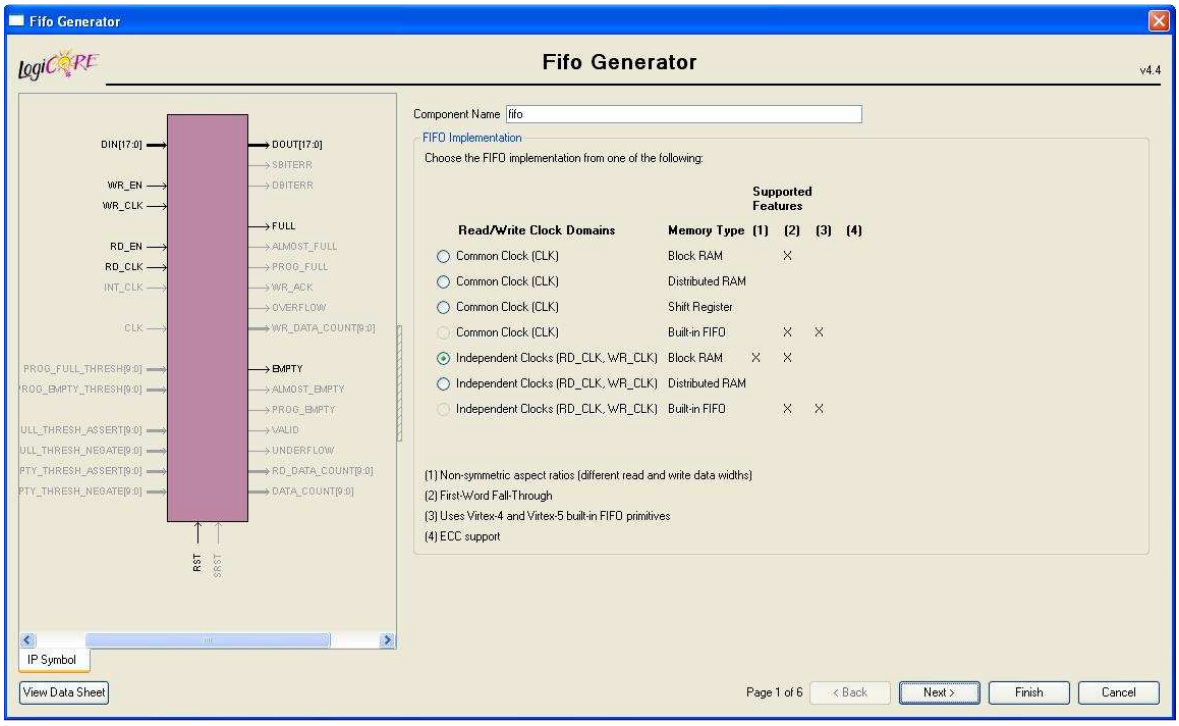
When adapting the Sniffer to capture data on both directions, a bandwidth limitation will certainly arrive, and so will the need to change the communication technology for transmission into a higher bandwidth solution. The Sniffer would be capturing traffic from two Ethernet PHYs but sending information only through one, meaning that the occupied bandwidth on reception could be roughly twice as the one available for transmission, if the system was heavily loaded. Correcting the problem by changing the technology for transmission would mean a new inspection to the available free technologies compatible with the FPGA. The world of technology is constantly evolving and IP Cores made very expensive in the past can now be available at affordable prices. Recommended solutions would be SATA or 10 Gigabit Ethernet. Alternatively, on a next project at the University of Aveiro, these protocols could be developed from scratch to make an FPGA "talk" these communication technologies.

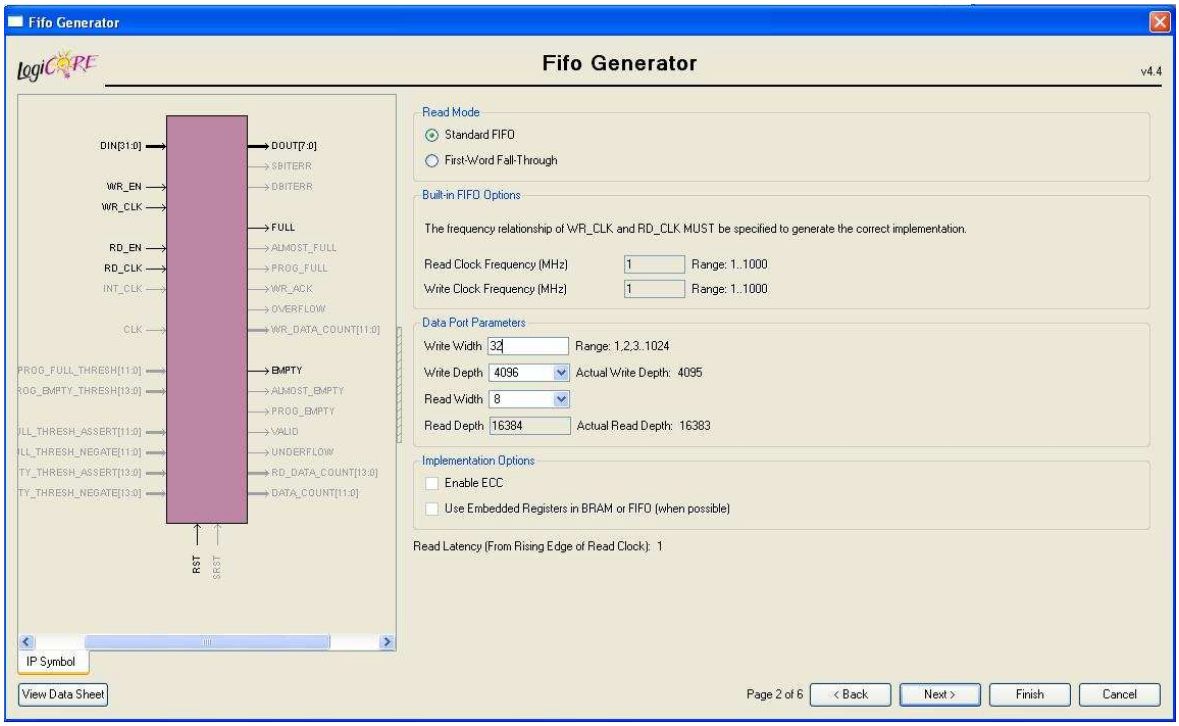
Another possibility and simpler one for a full duplex Sniffer could be to take out the multiplexing and using the last Ethernet PHY of the NetFPGA developing board. This would erase the need for a higher bandwidth technology, but would still demand the use of two computers or a computer with two Ethernet Ports.

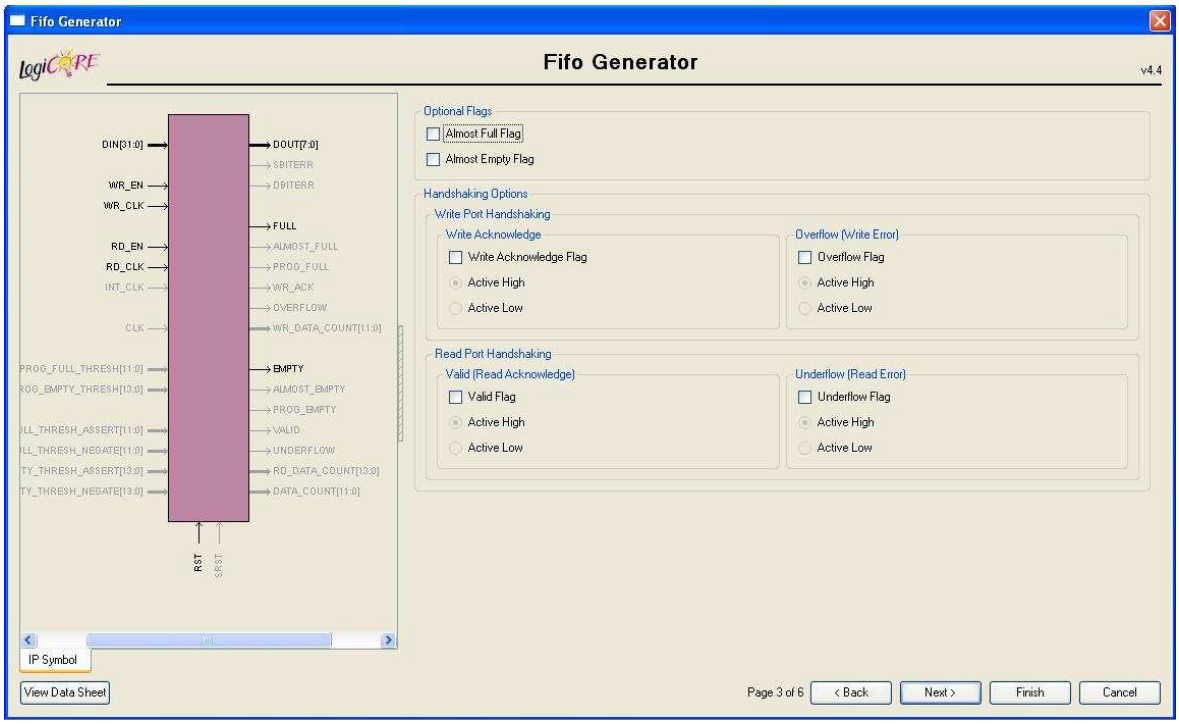
7.2.2. Temporal Precision

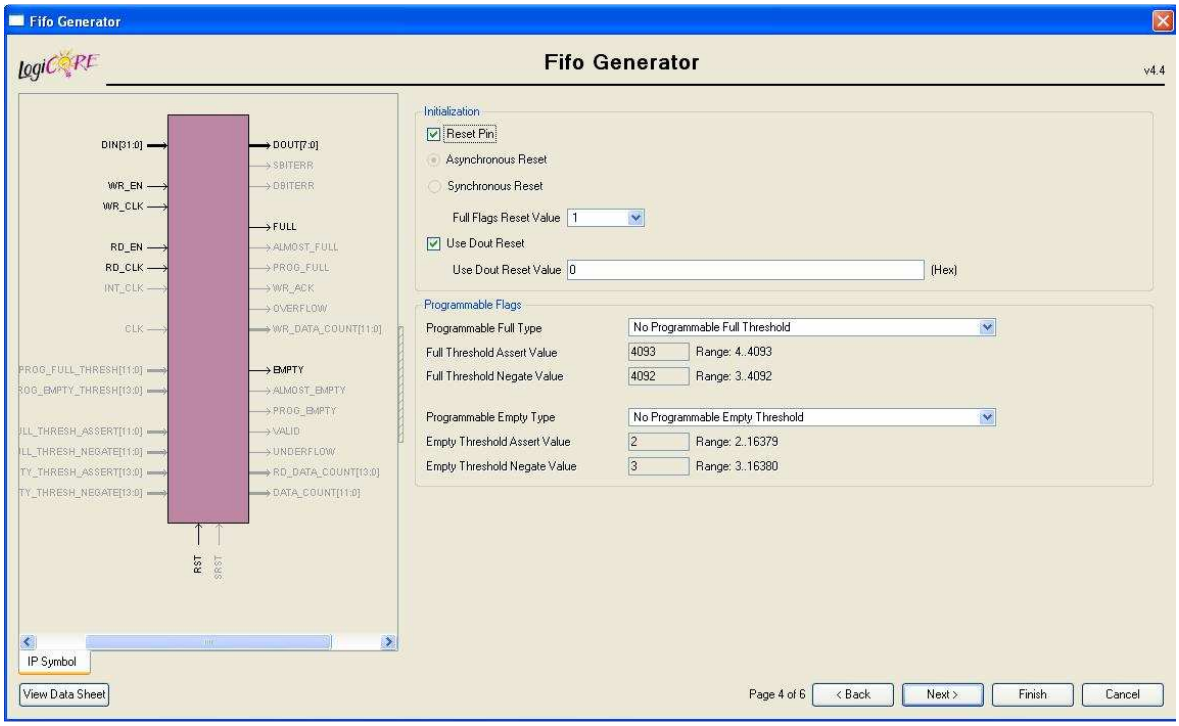
The TEMAC has a variation on the latency that affects the temporal precision of the Sniffer. A way to overcome this would be the construction of a path parallel in relation to the TEMAC, to increase the precision of the time-stamping. However, to accomplishing this would require an inspection to the signals coming from the RGMII bus, in order to detect when the message actually arrives.

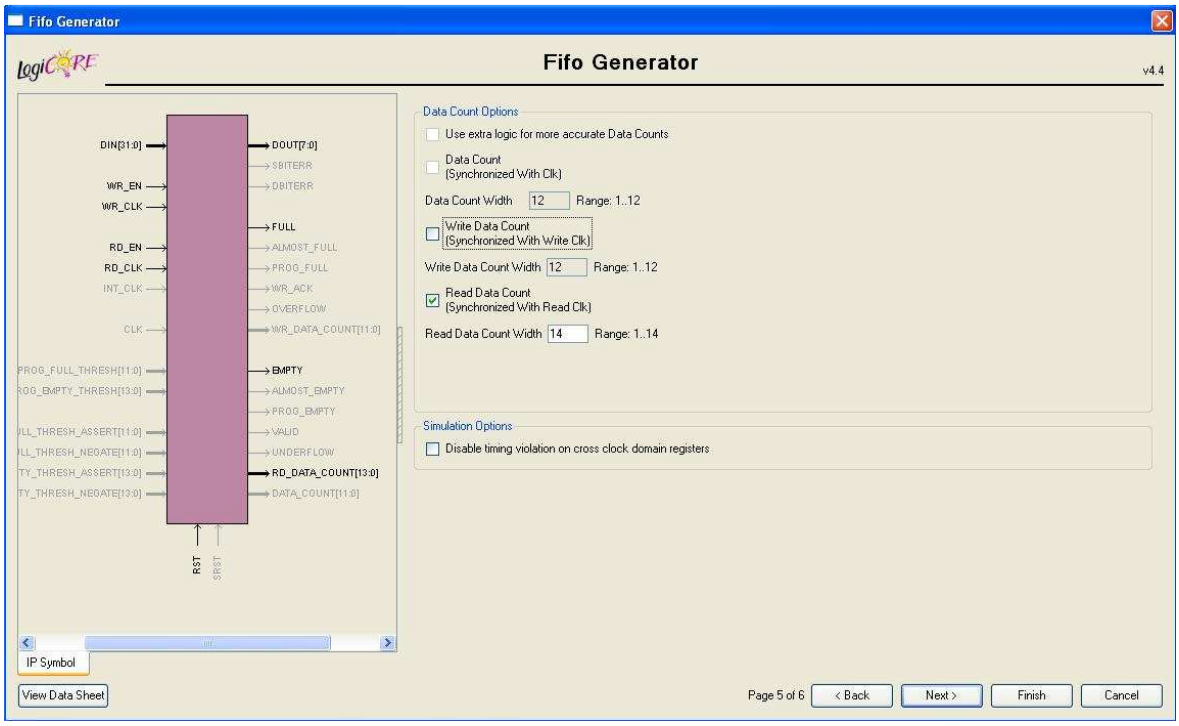
APPENDIX A) FIFOS INTELECTUAL PROPERTY CORE GENERATION

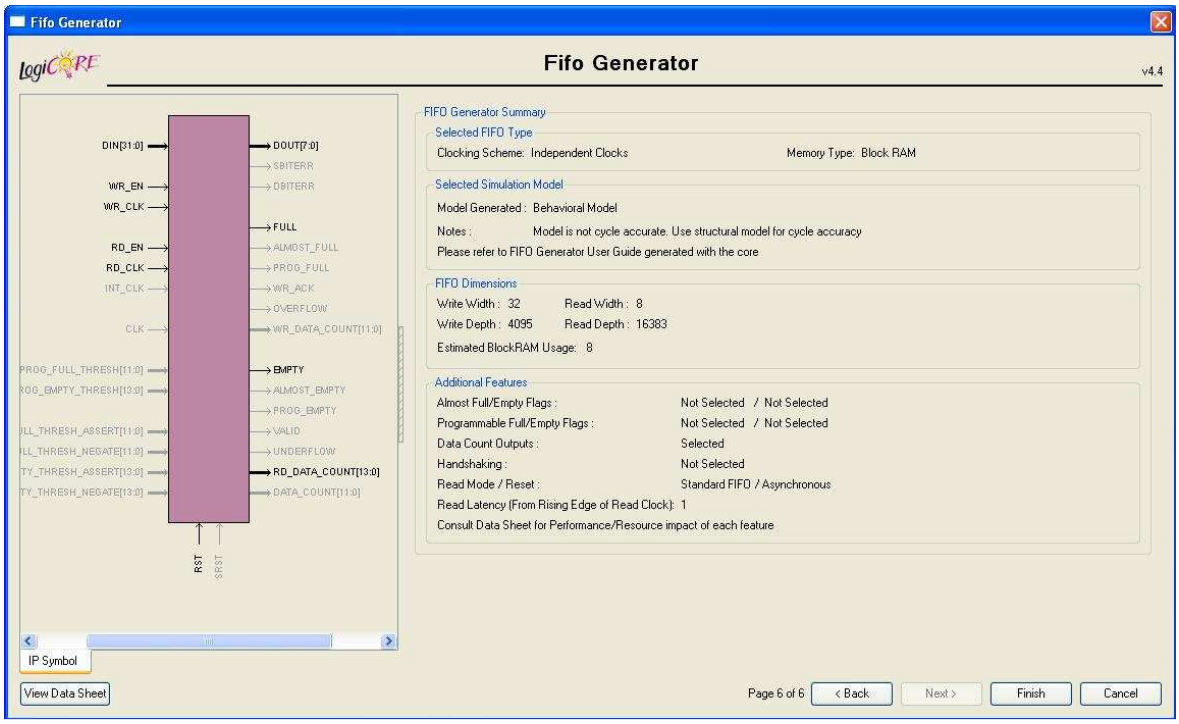




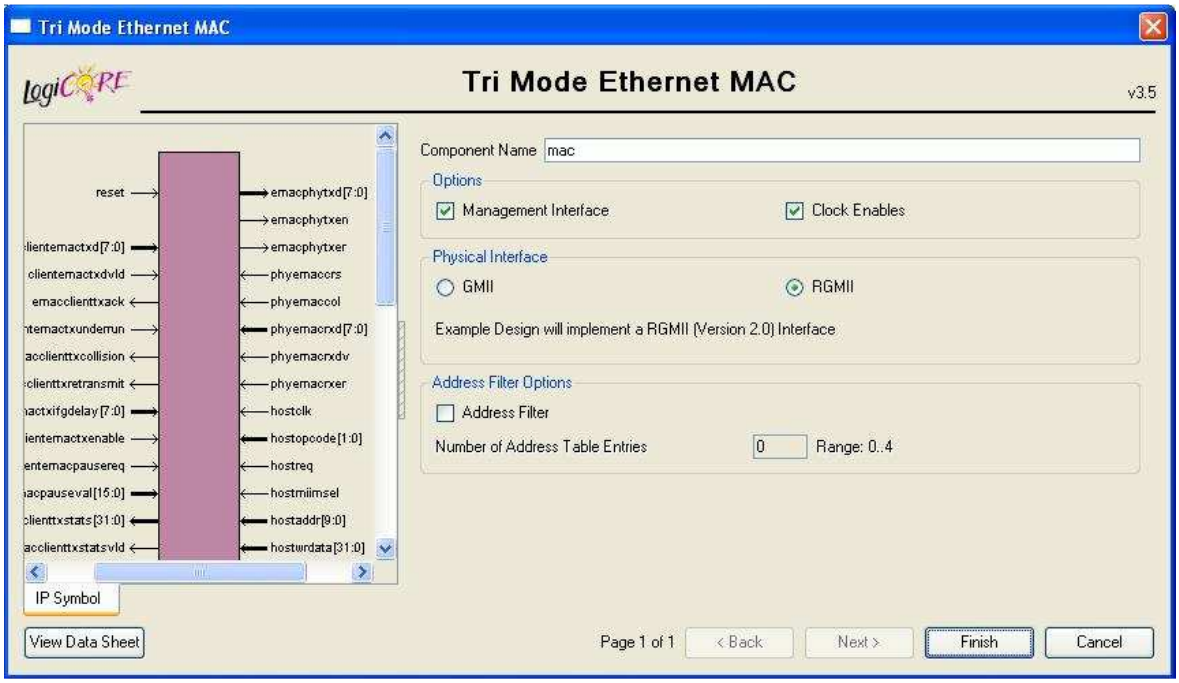

Fifo Generator
Fifo Generator
v4.4
DIN[31:0]
WR_EN
WR_CLK
RD_EN
RD_CLK
INT_CLK
CLK
DOUT[7:0]
SBITERR
DBITERR
FULL
ALMOST_FULL
PROG_FULL
WR_ACK
OVERFLOW
WR_DATA_COUNT[11:0]
EMPTY
ALMOST_EMPTY
PROG_EMPTY
VALID
UNDERFLOW
RD_DATA_COUNT[13:0]
DATA_COUNT[11:0]
RST
SRST
IP Symbol
View Data Sheet
Optional Flags
Almost Full Flag
Almost Empty Flag
Handshaking Options
Write Port Handshaking
Write Acknowledge
Write Acknowledge Flag
Active High
Active Low
Overflow (Write Error)
Overflow Flag
Active High
Active Low
Read Port Handshaking
Valid (Read Acknowledge)
Valid Flag
Active High
Active Low
Underflow (Read Error)
Underflow Flag
Active High
Active Low
Page 3 of 6
< Back
Next >
Finish
Cancel


Fifo Generator
Fifo Generator
v4.4
DIN[31:0]
WR_EN
WR_CLK
RD_EN
RD_CLK
INT_CLK
CLK
DOUT[7:0]
SBITERR
DBITERR
FULL
ALMOST_FULL
PROG_FULL
WR_ACK
OVERFLOW
WR_DATA_COUNT[11:0]
EMPTY
ALMOST_EMPTY
PROG_EMPTY
VALID
UNDERFLOW
RD_DATA_COUNT[13:0]
DATA_COUNT[11:0]
RST
SRST
IP Symbol
View Data Sheet
Initialization
Reset Pin
Asynchronous Reset
Synchronous Reset
Full Flags Reset Value
1
Use Dout Reset
Use Dout Reset Value
0
(Hex)
Programmable Flags
Programmable Full Type
No Programmable Full Threshold
Full Threshold Assert Value
4093
Range: 4..4093
Full Threshold Negate Value
4092
Range: 3..4092
Programmable Empty Type
No Programmable Empty Threshold
Empty Threshold Assert Value
2
Range: 2..16379
Empty Threshold Negate Value
3
Range: 3..16380
Page 4 of 6
< Back
Next >
Finish
Cancel


Fifo Generator
Fifo Generator
v4.4
DIN[31:0]
WR_EN
WR_CLK
RD_EN
RD_CLK
INT_CLK
CLK
PROG_FULL_THRESH[11:0]
ROG_EMPTY_THRESH[13:0]
LL_THRESH_ASSERT[11:0]
LL_THRESH_NEGATE[11:0]
TY_THRESH_ASSERT[13:0]
TY_THRESH_NEGATE[13:0]
RST
SRST
DOUT[7:0]
SBITERR
DBITERR
FULL
ALMOST_FULL
PROG_FULL
WR_ACK
OVERFLOW
WR_DATA_COUNT[11:0]
EMPTY
ALMOST_EMPTY
PROG_EMPTY
VALID
UNDERFLOW
RD_DATA_COUNT[13:0]
DATA_COUNT[11:0]
IP Symbol
View Data Sheet
Data Count Options
Use extra logic for more accurate Data Counts
Data Count (Synchronized With Clk)
Data Count Width 12 Range: 1..12
Write Data Count (Synchronized With Write Clk)
Write Data Count Width 12 Range: 1..12
Read Data Count (Synchronized With Read Clk)
Read Data Count Width 14 Range: 1..14
Simulation Options
Disable timing violation on cross clock domain registers
Page 5 of 6
< Back
Next >
Finish
Cancel


Fifo Generator
Fifo Generator
v4.4
DIN[31:0]
WR_EN
WR_CLK
RD_EN
RD_CLK
INT_CLK
CLK
PROG_FULL_THRESH[11:0]
ROG_EMPTY_THRESH[13:0]
LL_THRESH_ASSERT[11:0]
LL_THRESH_NEGATE[11:0]
TY_THRESH_ASSERT[13:0]
TY_THRESH_NEGATE[13:0]
RST
SRST
DOUT[7:0]
SBITERR
DBITERR
FULL
ALMOST_FULL
PROG_FULL
WR_ACK
OVERFLOW
WR_DATA_COUNT[11:0]
EMPTY
ALMOST_EMPTY
PROG_EMPTY
VALID
UNDERFLOW
RD_DATA_COUNT[13:0]
DATA_COUNT[11:0]
IP Symbol
View Data Sheet
FIFO Generator Summary
Selected FIFO Type
Clocking Scheme: Independent Clocks
Memory Type: Block RAM
Selected Simulation Model
Model Generated : Behavioral Model
Notes : Model is not cycle accurate. Use structural model for cycle accuracy
Please refer to FIFO Generator User Guide generated with the core
FIFO Dimensions
Write Width : 32
Read Width : 8
Write Depth : 4095
Read Depth : 16383
Estimated BlockRAM Usage: 8
Additional Features
Almost Full/Empty Flags : Not Selected / Not Selected
Programmable Full/Empty Flags : Not Selected / Not Selected
Data Count Outputs : Selected
Handshaking : Not Selected
Read Mode / Reset : Standard FIFO / Asynchronous
Read Latency (From Rising Edge of Read Clock): 1
Consult Data Sheet for Performance/Resource impact of each feature
Page 6 of 6
< Back
Next >
Finish
Cancel

APPENDIX B) TRI MODE ETHERNET MEDIUM ACCESS CONTROLLER INTELLECTUAL PROPERTY GENERATION



BIBLIOGRAPHY

Almeida, L., & Pedreiras, P. (2005). *Acetatos Aulas Teóricas STR.* Retrieved from STR - Sistemas de Tempo-Real: http://sweet.ua.pt/~lda/str/str.htm

Buttazzo, G. C. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.*

Buttazzo, G. (2005). Rate Monotonic vs. EDF: Judgment Day.

Colasoft. (2011). *Capsa Enterprise White Paper.*

Doyle, P. (2004). Introduction to Real-Time Ethernet I. *The Extension* .

Liu, L. C., & Layland, W. J. (1973). Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM* .

Marau, R., Almeida, L., & Pedreiras, P. (2006). Enhancing RT Communication over cots Ethernet Switches. *6th IEEE Workshop on Real-Time Networks.* Dresden, Germany.

NetFPGA. (2011). *NetFPGA*. Retrieved November 2011, from About: www.netfpga.org

Novell. (2003). *Ethernet Frame Types.* Retrieved from Novell Frequently Asked Questions list.

P. Pedreiras, L. A. (2005). Approaches to enforce real-time behavior in Ethernet.

Paessler. (n.d.). *Monitor Your Network with PRTG - It's So Easy!* Retrieved September 2011, from Paessler Webpage: http://www.paessler.com/prtg

Pedreiras, P., Gai, P., Almeida, L., & Buttazzo, G. (2005, August). FTT-Ethernet: a flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems. *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS* .

Pidgeon, N. (2000, April 01). *How Ethernet Works.* Retrieved September 2011, from How Stuff Works: http://computer.howstuffworks.com/ethernet.htm

Puga, J. (2008). *Sniffer para Redes Ethernet de Tempo-Real Baseado em FPGA.* Aveiro: Universidade de Aveiro.

RMII Consortium. (1998, March 20). RMII Specification.

SerialATA Workgroup. (2003). *High Speed Serialized AT Attachment.*

Todt, E. (2011). *Introdução Sobre o Tempo-Real.* Retrieved September 2011, from Departamento de Informática da Universidade Federal do Paraná: http://www.inf.ufpr.br/todt

Weibel, H., & Béchaz, D. (2004). Implementation and Performance of Time Stamping Techniques. *IEEE* .

Wireshark Developer's Guide. (2004-2010). Retrieved from http://www.wireshark.org.

Xilinx. (2008). *LogiCORE™ IP FIFO Generator v4.4.*

Xilinx. (2011). *FPGA Design Flow Overview.* Retrieved from Xilinx Website: http://www.xilinx.com/itp/xilinx10/isehelp/ise_c_fpga_design_flow_overview.htm

Zhou, Z., Cong, L., Lu, G., Deng, B., & Li, X. (2010). High Accuracy Timestamping System Based on NetFPGA. *International Journal of Future Generation Communication and Networking* .