u Ottawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Ahmad Abdo

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Hardware Implementation of a Packet Switch with an Optical Core

TITRE DE LA THÈSE / TITLE OF THESIS

T. Hall

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

V. Groza

L. MacEachern

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Hardware Implementation of a Packet Switch with an Optical Core

Ahmad Abdo

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the MASc degree in Electrical Engineering

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-25733-3
Our file Notre référence
ISBN: 978-0-494-25733-3

Abstract

With the rapid development of optical communications, transport of data over fiber channel can now reach rates as high as Gigabits per second. In order to take advantage of this tremendous bandwidth, fast packet switches that can operate at the same bit rates must be developed; as these modules currently represent the bottleneck of high speed telecommunication networks.

An optoelectronic packet switch architecture using reconfigurable optics in the central stage will be described. The central stage is surrounded by two electronic buffering stages partitioned into sectors to ease memory contention and to limit the size of the memory required for each sector. A Flexible Bandwidth Provision algorithm (FBP), is used to change the configuration of the central stage; essentially creating various numbers of internal paths that allocate variable bandwidth between sector pairs depending on traffic demands.

This thesis will present a scaled down *demonstrator* version of the packet switch, implemented on programmable logic devices (Stratix FPGA by Altera Corp.) forming electronic "islands" that are interconnected by a reconfigurable central stage crossbar. The FBP algorithm was implemented using VHDL at RTL level of abstraction. The design has 8 inputs ports, 8 output ports and 2 sectors in each buffering stage. Communication among modules within the demonstrator is performed through a custom high speed 10 Mbps interface. We will present our implementation of the central stage using first an electronic crossbar switch and then an optical switch (a Wavelength Selective Switch by Metconnex Inc.). Finally, we will also describe our configurable FPGA-based traffic generator built using the Nios II development board, Stratix Edition.

The generator consists of a Hardware packet generator, a Nios II processor playing the role of the "Maestro", and a custom graphical user interface to provide a friendly way to set traffic parameters. Our traffic generator is capable of supplying the switch under test with 2 different traffic models: Self Similar and Markov Modulated Bernoulli Process.

Keywords: Optical Networks, Packet Switching, Field Programmable Gate Array (FPGA), Very High Speed Integrated Circuit Hardware Description Language (VHDL), Traffic Generation.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transmission Mode
BER	Bit Error Rate
DMA	Direct Memory Access
E-O-E	Electrical to Optical to Electrical Converter
EMI	Electromagnetic Interference
ESD	Electrostatic Discharge
HAL	Hardware Abstraction Layer
GC	Global Controller
GUI	Graphical User Interface
FBP	Flexible Bandwidth Provision
FIFO	First In First Out
FCFS	First Come First Serve
FPGA	Field Programmable Gate Array
I-Sector	Input Sector
IP	Internet Protocol
MEMS	Micro-Electro-Mechanical Systems
MOEMS	Micro Opto-Electro Mechanical Systems
MPLS	Multi-Protocol Label Switching
MMBP	Markov Modulated Bernoulli Traffic
O-Sector	Output Sector
RAM	Random Access Memory
ROADM	Reconfigurable Optical Add/Drop Multiplexer
SFP	Small Factor Print
LE	Logic Element
LFSR	Linear Feedback Shift Register
LVTTL	Low Voltage Transistor - Transistor Logic
UART	Universal Asynchronous Receiver Transmitter

VCSEL	Vertical-Cavity Surface-Emitting Laser
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VOQ	Virtual Output Queue
VOSQ	Virtual Output Sector Queue
WSS	Wavelength Selective Switch

Dedication

To my family:

My mother Rola
My father Mohammad
My sister Julia
My brother Mahmoud

Acknowledgments

First and foremost, I would like to acknowledge and thank my thesis supervisor, Dr. Trevor J. Hall who has helped me and guided me throughout the last 2 years. I feel very grateful and privileged to work with someone that has contributed so much to the development of research in the field of optical networking.

I would also like to give my sincere thanks and gratitude to Dr. Sofia Paredes for her suggestions, interesting discussions throughout the past 2 years and for enormous help editing this work. Sofia, you are the pioneer of our group.

I would like to express my thanks to all my great friends and colleagues at the University of Ottawa.

Finally, I would like to express my greatest thanks to my lovely family for all their help, love, and encouragement through my entire life.

1 Chapter 1: Introduction

The rise of the Internet traffic over the last decade has allowed information transport to reach an unprecedented level. Multimedia information in particular has reached gigantic amounts with new video, images, and audio being generated, stored, and transmitted daily. This large amount of data has raised the challenge for efficient and high-speed switching and routing of data packets encapsulating the information.

With the rapid development of optical communications, there are more and more possibilities to transport these large amounts of internet data over optical fibre channels at Gigabits/second line rates. However, the switching modules at the nodes of the network currently represent the bottleneck of high speed telecommunication networks since there are few commercial devices capable of switching data packets at these rates without congestion. In order to be able to make use of the tremendous bandwidth provided by the optical links, extensive research on new routers / switches architectures is taking place in world class universities and companies.

One such research effort takes place at the Photonic Network Technology Laboratory (pntl.site.uottawa.ca) at the University of Ottawa [Abdo2 2005] [Paredes1 2005] [Abdo 2005] [Abdo1 2005]. An opto-electronic Clos-like packet switch has been designed that consists of two electronic stages surrounding an optical central stage and which plays to the strengths of electronics as a memory technology and to photonics as a communications technology whilst accommodating the relatively slow reconfiguration of current transparent photonic switch technology. The electronic buffering stages are partitioned into sectors to ease memory contention and limit the size of the memory required for each sector. A Flexible Bandwidth Provision (FBP) algorithm is used to

change the configuration of the optical central stage; essentially varying the numbers of internal paths between sector pairs in order to provide variable bandwidth to serve different types of traffic demands. The architecture and method have been extensively evaluated by simulation [Paredes 2005] [Taebi 2004] [Paredes2 2005] with results indicating that the simple FBP scheduling method successfully adapts the internal bandwidth of the switch to the incoming traffic; consequently reducing the overall delays of the packets traversing the switch, even at high loads of bursty traffic.

The natural next step for this research effort is to demonstrate that the concept of FBP in the sectored architecture is feasible in a hardware implementation; i.e., to build a *hardware demonstrator.* A hardware demonstrator focuses on control and architecture while using low data rates and port counts, which relieves the design of the difficulties related to high-speed electrical interfacing and large development costs while fully validating the FBP method. The development of a hardware implementation of the simulation work on a readily-available reprogrammable hardware platform demonstrates the function and reveals the control/synchronization issues of realization, leading to a better understanding of the requirements for scalability to higher port-counts and data rates.

The hardware platform chosen for the implementation of the demonstrator is the Field Programmable Gate Array (FPGA) and the FPGA development boards produced by Altera. Many reasons were behind the choice of the mentioned board; mainly:

- Its capability to integrate programmable logic and soft-processors onto a single device, which allows the partitioning of the design into software and hardware.

- The availability of various interfaces (2 serial ports, one RJ-45 and many programmable I/O at 3.3 V and 5 V).

The packet switch is being implemented on separate programmable logic devices forming electronic "islands" which are interconnected by a reconfigurable crossbar central stage with the capability of emulating optical link delay. The current design has 8 inputs and 8 outputs divided between 4 sectors, each of them being a "4x4" sub-switch implemented on a single Field-Programmable Gate Array (FPGA) with shared buffer memory. Communication among modules within the demonstrator is performed through a custom high speed 10 Mbps interface.

In this thesis we will present and describe several modules developed for the hardware demonstrator; namely the output sectors, the global controller (which includes FBP), the communication protocols between modules of the switch itself and an electronic version of the central stage. The functions and procedures have been debugged / simulated using specific software tools (Quartus II and Modelsim [Mentor 2005]). A testbed that will be used to measure various performance parameters of the switching modules has also been designed and implemented.

1.1 Objectives

The ultimate goal of the team project is to design and implement a scaled-down version of the architecture that will allow the emulation of the results obtained for the simulation of a larger switch.

The objectives for the present work are:

- To use the capabilities of the used FPGAs and development boards for the implementation of various packet switching functions; as well as find the limitations of these hardware platforms.
- To design the specific functionalities of the mentioned switch modules and implement them in the appropriate low level programming languages.
- To evaluate the modules and testbed developed using specific simulation tools for FPGA development boards.
- To download the implemented codes on to the FPGA and test the performance of the modules.

1.2 Contributions

The main contributions of this thesis include:

- Overall design of the packet switch demonstrator in collaboration with other members of the PNTLab, including the introduction of wavelength selective switches as the optical crossbar switches in the central stage.

- Design and development of low level software codes to implement the functionality of the following modules of the packet switch demonstrator with FPGA development boards:
  - Output sector (with simplified memory management)
  - Global controller, including the FBP method implemented using VHDL at RTL level of abstraction
  - Electronic version of the central stage

- Communication protocols between the input sectors and the global controller, as well as between the global controller and central stage

- Enhancement of the Flexible Bandwidth Provisioning algorithm by introducing comparison function before applying a new configuration in order to avoid unnecessary reconfigurations of the optical switches.
- Novel FPGA implementation of a traffic generator that will be used to generate packets with various stochastic distributions to test the packet switch demonstrator.

1.3 Thesis Organization

The remaining of the thesis is divided into five chapters. Chapter 2 provides an overview of networking world, where packet switches reside, describes and discusses the state of the art of the world of packet switching as well the various scheduling algorithms. Chapter 3 talks about the sectored three-stage optoelectronic packet switch architecture and the implementation of the Electronic Crossbar, Optical Crossbar and the Output Sector. Chapter 4 is dedicated for the description of the Flexible Bandwidth Provision algorithm, as well both our Software and Hardware implementations. Chapter 5 describes the design and the implementation of a programmable traffic generator. The last chapter concludes the thesis with the undergoing work and the future steps of this work.

1.4 Used Tools

- Modelsim v6.0 from Mentor Graphics
- Quartus II v4.0, including SOPC builder and DSP builder from Altera
- Nios II Development kit from Altera

- Nios II Integrated Development Environment
- BlueJ Integrated Development Environment
- Microsoft Visual C++ v6.0
- Matlab 7.0 from Mathwork
- WSS 5400 and WSS GUI v1.0.6 from Metconnex

2 Chapter 2: Background

2.1 Networking

2.1.1 The OSI Model

In computer communications, the tasks required to get two entities, that are connected either directly or via other networks talking is a complex task. It involves a wealth of technical complexity from the source transmitter to the destination's receiver [Stallings 2000]. Some of these tasks include: interfacing, signal generation, synchronization, exchange management, error detection and correction, flow control, addressing, routing, recovery, message formatting, security, network management.... Given this extended list, it becomes obvious that it is far easier to implement all this functionality in contiguous layered protocols rather than in a single monolithic architecture.

From the discussion above, it became clear that a single large monolithic protocol was not a viable solution. And at the time where personal computers were starting to emerge and the industry was being flooded by proprietary networking protocols, the International Organization for Standardization (ISO) set out to develop a complete protocol framework for computer communications. The result was the Open Systems Interconnection (OSI) model presented in Figure 1.

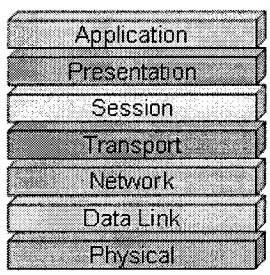


Figure 1: OSI seven layer model [Stallings 2000]

This model separates computer communications into seven hierarchical layers, each implementing a part of the needed functionality. It is important to note that the seven layers do not automatically equate to seven protocols. In fact each layer can be mirrored by one or more protocols.

2.1.2 Switching

Early in the development of communications networks, two types of frameworks emerged:

1. Switching networks
2. Broadcast networks

While the first became equivalent to Local Area Networks (LAN), the second now dictates much of the operation of Wide Area Networks (WAN). Switching networks are themselves categorized as:

1. Circuit Switching Networks
2. Packet Switching Networks

2.1.2.1 Circuit Switching

Circuit switching is a technique that utilizes a dedicated communication path between two stations. Circuit switching was developed for and mostly services voice traffic. It has been deployed since the early 1920's in the public telephone network and later in the Private Branch Exchange blocks (PBX). Circuit switching operation can be divided into three phases:

1. Circuit establishment
2. Data transfer
3. Circuit disconnection.

In circuit switching, (even when the conversation is animated) the channel utilization remains relatively low. This "inefficient" aspect of circuit switching is even more accrued in the case of jittery computer communication. The result is a dedicated channel that is idle most of the time. Despite this weakness, circuit-switching networks were deployed to handle voice and data traffic mostly because of a widespread installed base. This trend is now fading and packet switching is becoming the norm even in the voice segment of the market.

2.1.2.2 Packet Switching

Packet switching was designed to provide a more efficient facility than circuit switching for bursty data traffic. A voice phone call creates enough channel utilization to justify the use of circuit switching. By contrast data traffic is jittery and inconstant. It leaves a circuit switching connection idle and unutilized most of the time. To solve this problem, packet switching introduces a new philosophy where the information is chopped off into small packets. Each packet will receive a header containing control information. This appended control information differs from one protocol to another but the following are generally consistently found:

1. Originating entity: A unique identifier describing the sending entity.
2. Destination Address: A unique identifier describing the destination entity.
3. Packet length: A field describing the size of the packet.
4. Error detection bit(s): One or more bits allowing the flagging of a transmission error. In the case of a parity-checking algorithm, one bit is needed. More evolved cyclic redundancy checking algorithm will use more than one bit (eight being a standard).

Within packet switching, we can distinguish between two fundamentally different modes of operation:

1. Virtual Circuit
2. Datagram

An example of a packet switched network is shown in the picture below. The nodes in the network cloud can be switches or routers. As they arrive at each node, packets sent from host A to host B get switched onto a path which leads them to host B.

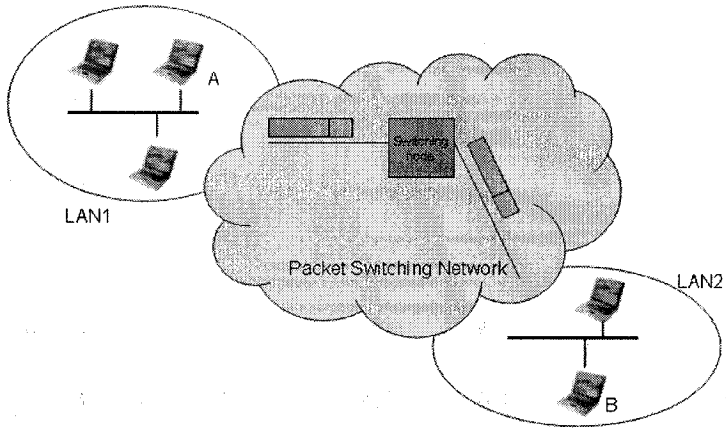


Figure 2: Packet switching network with two LAN

2.1.3 Internetworking

Internetworking Protocols are required to connect different or similar packet switching networks. They are located on the top of the network layer (Layer 3) of the OSI model. They subsume the packet switching protocol. An internetworking protocol must provide the following functions:

- **Link Provision:** This requirement is concerned with the creation and maintenance of a link between two similar or dissimilar packet switching networks.

- **Routing**: is the driving mechanism responsible for orienting packets in the network to their proper destination. It is the determination of a path that a data unit (frame, packet, and message) will traverse from source to destination. Over the years different types of routing schemes have been developed. Some of these schemes require information about the surrounding nodes, others do not.
- **Network Management:** An internetworking protocol must provide for networking management functions. Under the umbrella of network management resides such key aspects as billing, path status and traffic load.
- **IP: A Connectionless Internetworking Protocol:** The Internet Protocol IP is part of the TCP/IP suite and is the most widely used internetworking protocol. Within IP, one can gather around a multitude of support protocols that are critical to its proper functioning. These include the signaling protocols required to gather around the link-state information from the surrounding nodes, generically referred to as Interior Gateway Protocols (IGP) or Exterior Gateway Protocols (EGP). An IP Protocol Data Unit (PDU) is made up of a header and data received from the higher layers. A closer look into the IP header will provide more of an insight as to the operational characteristics of IP.

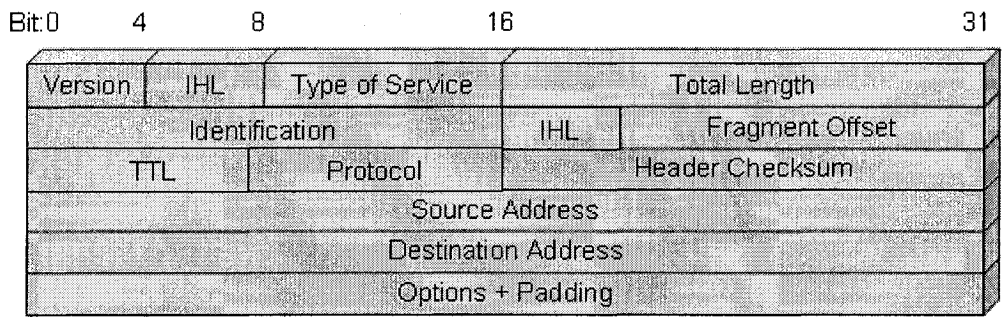


Figure 3: IP version 4 header [Stallings 2000]

Here are some highlights of the fields included in the version 4 header:

1. Version: indicates the IP version number. Currently two versions are available 4 and 6.
2. Header length: indicates the value of the header as a multiple of 32 bit words. The minimum value is 20 octets
3. Type of service: This field is often avoided and left out when discussing the version 4 header. It clearly shows that IP version 4 had from the start, the potential to support some kind of limited quality of service. Unfortunately the IETF was late pushing for standardization for this part of the header. The result was that router manufacturers either implemented their own custom QoS or ignored these 8 bits altogether providing best effort routers.
4. Time To Live (TTL): A hop count indicator avoiding infinite looping in the network.
5. Source address: 32-bit field uniquely identifying the datagram sender.
6. Destination address: 32-bit field uniquely identifying the datagram destination.

In the OSI communications model presented in Figure 1, a switch performs the layer 2 or Data-Link layer functionality such as an ATM cell switch.

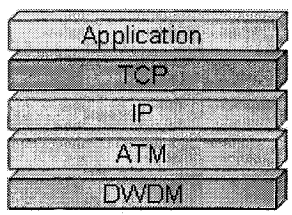


Figure 4: Example TCP/IP five layer model [Stallings 2000]

A packet switch can be embedded inside a router, where the ingress line cards divide the variable length packets (IP, MPLS or others) into fixed length packets or cells.

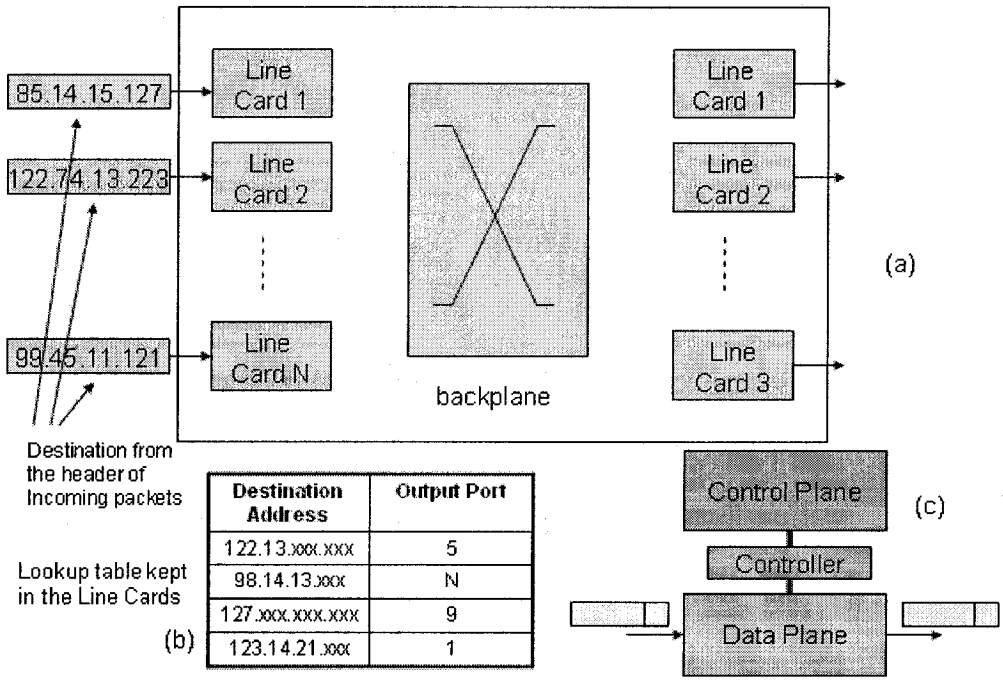


Figure 5: (a) Generic router architecture, (b) look-up table and (c) Planes division [Lekkas 2003]

2.2 Packet Switches

We can define a packet switch as: "a device that channels incoming packets from any input port to any output port" [SearchNet 2006]. Packet switching can be used within a single ASIC to connect multiple processors or in a network to forward data to the corresponding destination; packets can be transferred to multiple outputs in case of multicasting packets.

Modern packet switches consists of a number of Line Cards related by switch fabrics. The contents of a packet depend on the domain of the network. In the case of IP networks, the packet is of variable length with an estimated distribution shown in Table 1. The 40-byte length represents the TCP acknowledgement packet, which occurs the most. In the case of ATM networks, a packet has a fixed length of 53 bytes.

Packet size [byte]	Probability of occurrence
40	0.4373
512	0.1015
600	0.0234
700	0.0312
1500	0.3906

Table 1: IP packet size distribution [Tagami 2002]

The basic functionalities of a packet switch within the network are:

- The ingress line cards receive packets, perform header recognition in order to know the output port destination, and store them (or transmit them if possible) until the path to the egress line card is clear.
- The switch fabric, which can be optical or electronic, connects the ingress line cards to the egress line cards.
- The egress line card receives the packets and transmits them. The scheduling within the line cards is mostly done on FCFS basis.

In the case of routers, the centralized CPU will perform tasks other than reconfiguring the central core; such as keeping the routing tables updated. In addition, the router's capabilities include those of a switch, such as: support for quality of service, which is becoming a "must" feature in today's switching modules.

2.3 Evolution of Packet Switches

Initially, researchers considered designing and implementing packet switches / routers, based on the conventional computer architecture design. Thus, it was expected

that first generation routers (Figure 6) be built around a CPU with a shared backplane bus, shared pool of memory, and some line cards connected to the media.

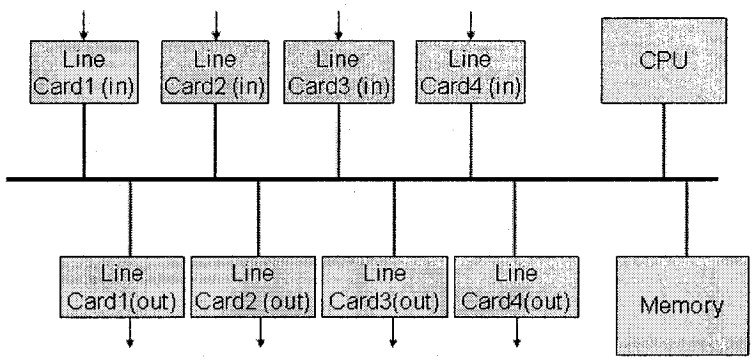


Figure 6: First generation packet switch [Katevenis 2005]

The operation of the router follows the subsequent sequence: when packets arrive at any line card, they are transferred to the CPU through a shared bus. All processing and forwarding decisions are made in the CPU, and the packets are buffered in a central shared memory until the outbound link becomes free. Finally, the packets are transferred through the shared bus to an outbound line card, and transmitted to the media connected on the next hop. While it is clear that this architecture is simple, especially that the computer architecture domain is quite mature, this design has a lot of drawbacks especially the use of a shared bus, which degrades the performance due to the following reasons:

- **Arbitration delay**: Time needed to transmit control data transmitted by the CPU in order to determine the next transmitter.
- **Turn-around overhead**: Idle time between two transmissions due to the bi-directionality of the medium.
- **Lack of parallelism**: Only one transmission at a time.

The second generation of switches (Figure 7) overcame the shortcomings of the first one by eliminating the main memory, and introducing local memory for each input line card as shown in the picture below. This allowed for better performance since the bus is used more efficiently.

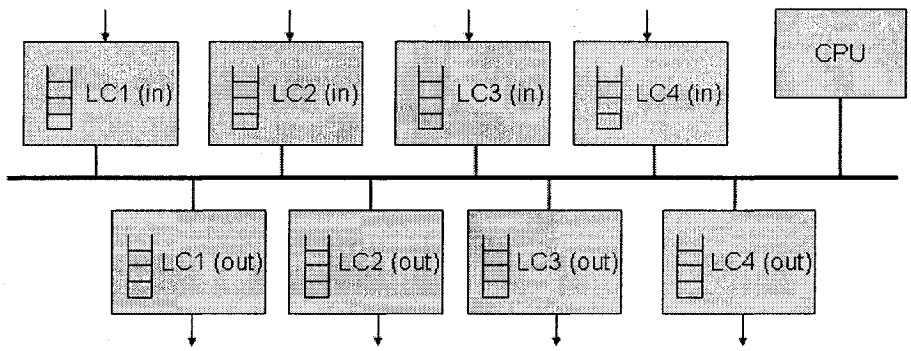


Figure 7: Second generation packet switch [Katevenis 2005]

One of the designs that eliminated the use of a bus is presented in Figure 8. In this architecture, the ports are connected in a ring. Cells, fixed length packets, are put into empty time slots and taken from filled and matching time slots. RI is the ring component of the switch. As shown, queuing in these switches is done mostly at output ports [Turner 1998].

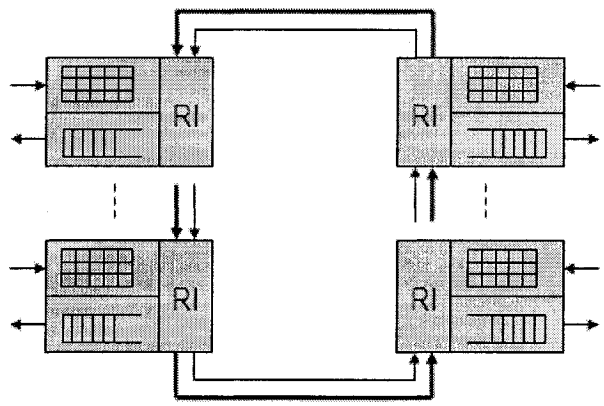


Figure 8: Router with line cards connected in a ring topology [Turner 1998].

The third generation packet switch / router used the crossbar for interconnecting the Egress/Ingress line cards. While this permitted parallelism, and decreased latency significantly in the operation of the router, it includes more complexity in the design of the switch fabric.

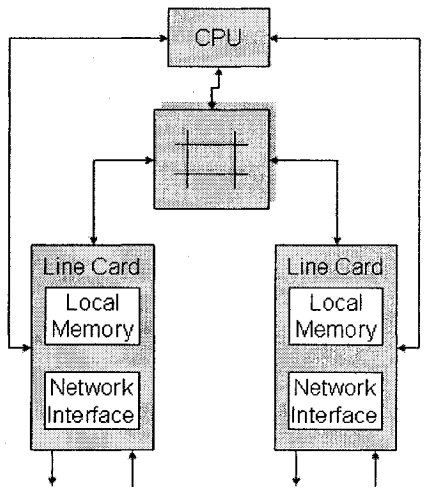


Figure 9: Third generation packet switch/router [McKeown 2005]

The fourth generation packet switches (Figure 10) introduced the deployment of photonic technology in the core. This became more possible and reliable with the exploitation of high speed SFP transceivers, since a mal-functional optical transceiver can be replaced without the need to remove the line card and perform changes in the printed computer board. The use of optics in the core instead of electronics has many advantages, some of which are:

- Minimization of power consumption
- Avoiding EMI due to high speed electrical links
- Allow scalability and update of the core without change in the design of the line cards.

While the disadvantages are:

- **Extra cost:** By introducing the expensive E-O-E. However, it is expected that the development of high density optical interconnect technology (for example based on VCSEL technology) will drive cost down.
- **Configuration time:** Optical switches are slow to configure, which make them unsuitable for switching on a per-packet basis.

In this generation of packet switches, clustering was introduced where the line cards are separated from the switch core. This made it easy to package, cool, and most importantly allow a larger number of line cards to be interconnected in a single packet-switch [pmc-sierra 1999].

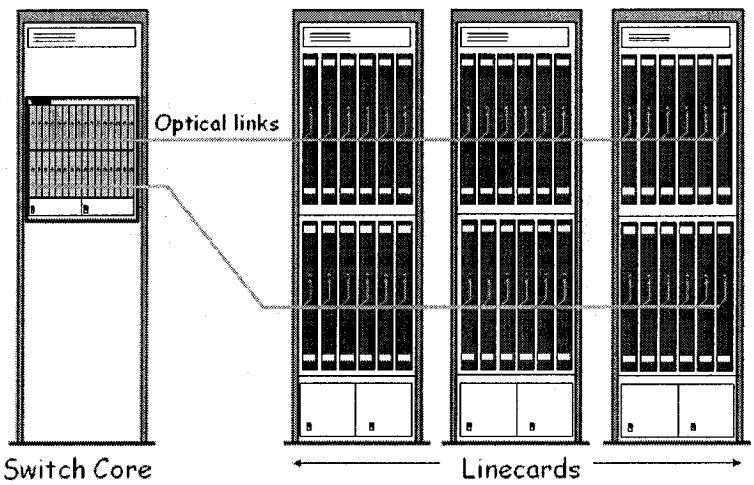


Figure 10: Fourth generation packet switch [pmc-sierra 1999]

2.4 Buffering Methods in Packet Switches

As mentioned earlier, one of the packet switch's functions is storing packets depending on their destination before forwarding them; and guaranteeing the path to the output line cards. Therefore, buffering strategies play a vital role in improving the performance of the switch, especially since memory access speed is a bottleneck in

today's systems such as servers. It is thus crucial for memory management and buffering to be well designed.

Input queued switches, presented in Figure 11, consist of N line cards, where N is the number of inputs to the switch (N=4 in Figure 11), a crossbar, and a scheduler. Scheduling is done per time slot, which corresponds to the transmission time of an internal packet/cell. An input buffered crossbar switch fabric is sometimes a more attractive solution due to its highly scalable, low cost, and non-blocking switching capabilities; however, it was proved by [Karol 1987] that the throughput of the input queue switch using standard FIFO is limited to $(2 - \sqrt{2}) \approx 58.8$ % when the number of ports increases under uniformly distributed and independent traffic. This is mainly due to "Head of Line" blocking, where a packet going to a non-busy output port is kept in the FIFO because another packet at the head of the FIFO is waiting for its destination to be free.

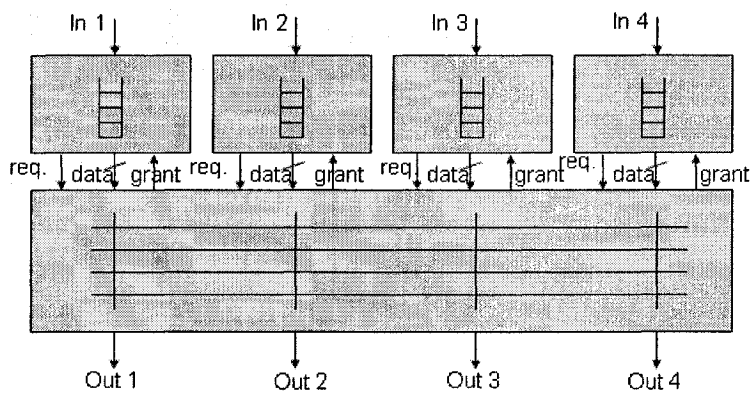


Figure 11: 4x4 input queue switches [Karol 1987]

A solution to this problem was introduced by [Anderson 1993] and [Karol 1992]. They presented the concept of VOQ (Figure 12), which requires the management of a

separate FIFO per output line at the input line card. [McKeown 1999] introduced two suitable scheduling algorithms, where it was shown that a 100 % throughput is possible.

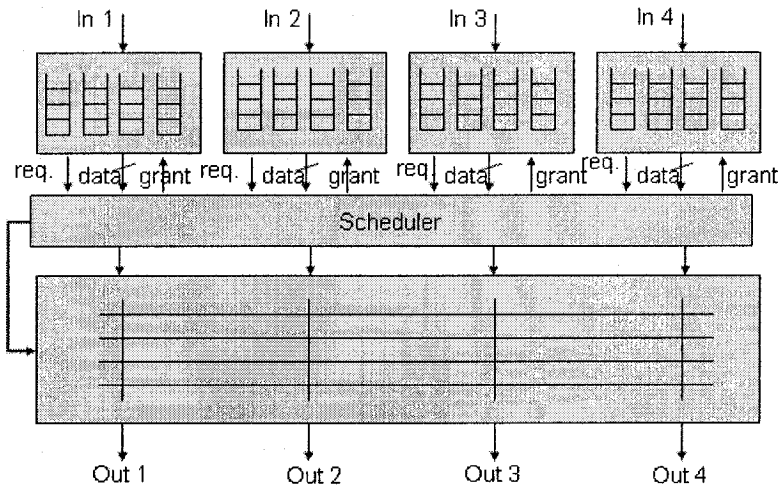


Figure 12: Advanced input queuing: virtual output queues [Karol 1992]

An input smoothing architecture was presented in [Karol 1988], and it is illustrated in Figure 13. The packets arriving are merged into a frame, within which they are stored at inputs buffer of each input and then enter simultaneously to input ports of switch (de-multiplexed) for *b* time slots [Gospodinova 2004]. While the probability of dropping a packet inside the switch can be decreased by increasing b, where b is the factor by which the size of the crossbar is scaled, there is a major disadvantage by having a large switch fabric; the bigger the switch fabric is, the harder the possibility to perform high-speed configuration of the crossbar. This is especially true since we are limited by the amount of current that can be used within the switch in order to change the connections in the crossbars.

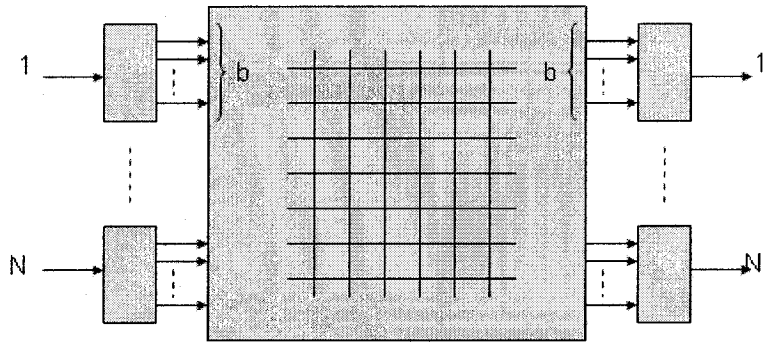


Figure 13: NxN input smoothing architecture [Karol 1988]

The paramount buffering strategy in packet switches is the output queue, where the input line rates run *N* times slower than the crossbar including the header recognition (i.e. the crossbar runs N times faster than the input/output line rates). In this case, even when all packets coming in a certain time slot are going to the same output, the crossbar is able to serve and store all of them in the corresponding buffer. However this is not practical since it is hard to build a crossbar with configuration speed faster than the interface. And it gets harder when the number of inputs/outputs increases.

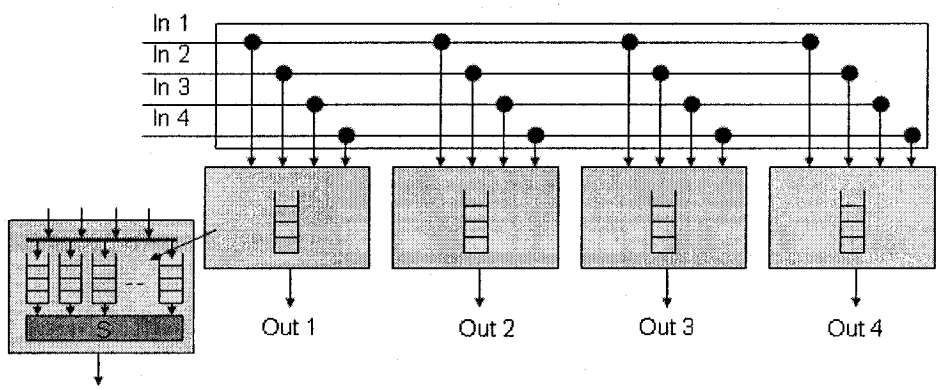


Figure 14: 4x4 output queue switch [karol1987]

2.5 Switch Architectures

In this section we will give a quick summary of an existing switch architecture that is similar to ours. It was described in [Agelis 2005], in which all packets from the inputs are transferred to the output buffer (output port queue on the board) with the correct routing to an output switchboard and with the least number of packets in the queue. The system uses MOEMS devices to achieve re-configurability in the backplane network, which provides the opportunity to adapt the shuffle exchange network to asymmetric application characteristics.

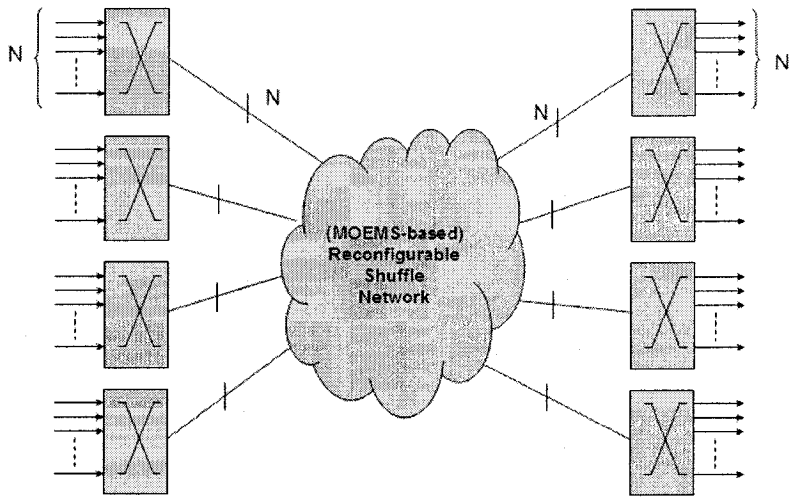


Figure 15: Agelis' proposed router architecture

2.6 Scheduling Algorithms

Scheduling algorithms play the role of a calendar in a packet switch, where an input line card sends a request with the packet destination to the CPU running the algorithm. Many scheduling algorithms are based on a bipartite graph representation of the input and output ports; The CPU performs a bipartite graph matching between the source line cards and destination line cards. Once the calculation is done, configuration of

the crossbars takes place, and a transmit permission is granted to input line cards. Since the bipartite graph is an important concept when talking about scheduling algorithms in packet switches, we will give its definition: A graph is an ordered pair (V, E), where V is called the vertex set and E is called the edge set. With each e $\in$ E, we associate two vertices v_i and v_j, which we call the ends of e.

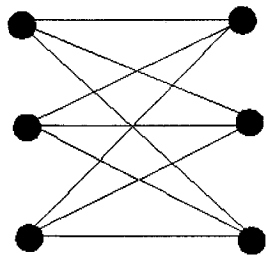


Figure 16: Complete 3x3 bipartite graph

Maximum size matching was presented in [McKeown 1995]. At each time slot, the scheduler looks at each FIFO (Figure 12) in the input stage, and performs a matching taking into consideration maximizing the number of edges. If the traffic is not uniformly distributed, the traffic will be headed to certain outputs of the switch. So, the scheduler will have a limited number of configuration choices. The best known maximum size matching algorithm converges in $O(N^{5/2})$ time [McKeown 1999]. The drawback of this algorithm is that it does not consider the length of the queues or the waiting time of the packets at the head of the input FIFOs. In order to overcome the latter problem, [McKeown 1996] presented the Maximum Weight Matching. A weight coefficient is assigned for each queue at the input line card. The bigger the FIFO storage is, a higher weight is assigned. The matching algorithm finds an input-output match that has the highest sum of weights. The advantage of this algorithm is its stability for uniform and

non-uniform type of traffic. However, the disadvantage of this algorithm is its high complexity $O(N^3 \log N)$, which makes it hard to implement in hardware.

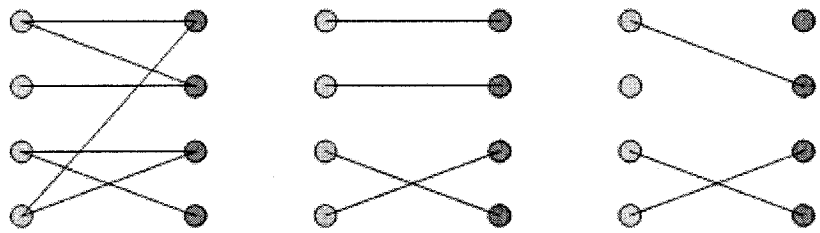


Figure 17: Bipartite graph, Maximum match and Maximal match

[Mekkittikul 1996] demonstrates that the Oldest Cells First scheduling technique results in 100% throughput. It uses the waiting time of cells on the head of each VOQ as weights and performs the matching in order to maximize the addition of all waiting times. The drawback of this algorithm is its complexity. Another algorithm, Longest Port First, is based on the Oldest Cells First and was introduced in [Mekkittikul 1998] with a complexity of $O(N^{5/2})$. The weight of each queue is the sum of aggregate input and output queue occupancies. This algorithm finds the match that is both maximum size and maximum weight. iSLIP was presented in [McKeown 1999], it is intended to schedule packets in input queued switches corresponding to a maximal size match. It was shown with simulation that it guarantees 100% throughput for uniform traffic with a single iteration. Several iterations are needed for non-uniform traffic in order to get 100% throughput [Paredes 2004].

2.7 Crossbars Overview

Crossbars are symmetric parallel physical paths between input ports and output ports. They have configuration ports (serial port, parallel port, Ethernet port ...) used to arrange the connection points shown in (Figure 18).

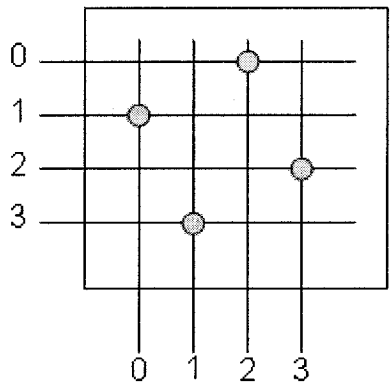


Figure 18: 4x4 crossbar switch with physical connections (0,2), (1,0), (2,3), (3,1)

[Keyvani 2001] introduced a new crossbar configuration, where each output is the result of a logic equation combining all the inputs with the control information. The control bits required are (# of inputs)^2, so the system is more susceptible to disturbances by EMI and ESD. We used this parallel line implementation in our case; however, another possibility would be sending the control information serially from the scheduler in order to eliminate deploying parallel lines. It is worth noting that the latter method creates additional delay in the setup.

3 Chapter 3: Sectored Packet Switch: Architecture and Implementation

In this chapter we will explore in detail our Sectored Packet Switch and we will present our hardware implementation of the O-Sector, as well as the electronic and optical crossbars. We will introduce the use of a ROADM as the optical interconnect within packet switches, as well as three optical performance measurements carried out on the Metconnex WSS 5400. This chapter is based on parts from [Abdo 2004]. Most of the modules simulations are presented in Appendix A.

3.1 Switch Architecture

Our hybrid switch architecture consists of three stages (Figure 19) The first stage contains l centralized shared memory packet switch modules, I-Sectors (Figure 20), that switch and buffer packets between n ingress and m egress ports. The third stage contains l O-Sectors (Figure 21) that switches and buffers packets between m ingress and n egress ports. The buffering stages are suited for electronic implementation. The overall switch dimension is $N = n \times l$. The central stage consists of m crossbar switches of dimension $l \times l$ with no buffering, therefore suitable for implementation in photonic technology. There are kN paths available in the central stage, where k is the *speed-up*. In our implementation, we are dealing with k equal to 1. The packets are assumed to be fixed-sized and a *time slot* is equal to the duration of a packet. All links transport one packet per time slot (i.e., all of them operate at the same line rate); therefore speed-up is provided spatially and it is equal to $k = m/n$.

A hardware demonstrator reproduces theoretical assumptions regarding component features such as dual port memory with bandwidth relative to the ingress data rate and photonic crossbar technology, as stated in [Paredes 2005] [Paredes1 2005]. However, due to component limitations and cost when scaled to practical line rates, parts of this architecture are challenging to implement. In particular, the memory assumed to be a fast, large dual-port RAM is expensive and management of multiple queues in a single RAM is a control intensive task that lowers the effective throughput of the memory.

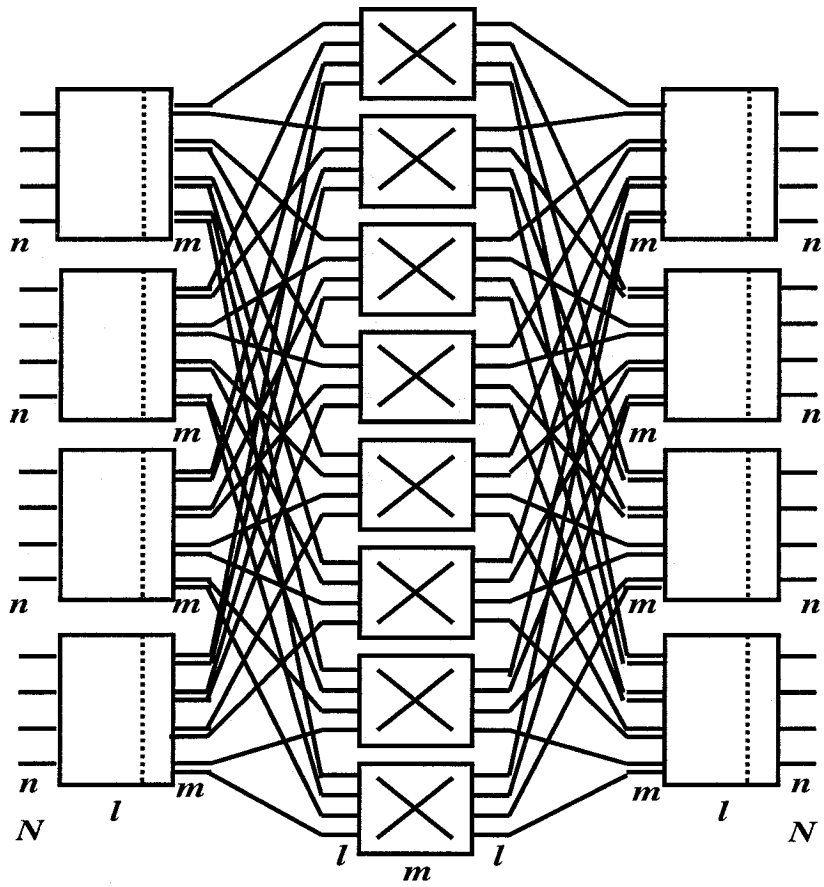


Figure 19: Sectored packet switch architecture with an optical core

Since the packet switch design contains scalable blocks; that is, the switch can be configured with a variable number of input sectors, output sectors and cross-bars; our hardware demonstrator is a scaled down version of the 64×64 port switch: it consists instead of a 8×8 port design with two 4×4 input sectors, two 4×4 output sectors and four 2×2 electronics crossbars. The limitation to the development of a larger design was strictly a result of component availability.

In this chapter, we will only introduce the architecture of the Input Sector. A full description of the Input Sector design and implementation is presented in [Bishtein 2004]. The input sectors maintain *Virtual Output Sector Queues* (VOSQ); i.e., one queue for each output sector within each input sector. The output sectors maintain *Output Queues* (OQ); i.e., one queue for each of its output ports within every output sector. The total number of queues to be managed in the switch is $l^2 + N$.

3.2 Input Sector

The I-Sector module (Figure 20) performs coarse switching: it segregates packets destined to a group of output ports (O-Sector) by placing them in VOSQs and serves these queues according to a *service matrix.* In our hardware implementation, the Input Sector is a controllable Shared Memory Packet Switch with control and buffering implemented on an FPGA, using SDRAM for queuing. Since memory is part of the data path, it imposes a limit on the sector's throughput. Careful rationing of memory-to-line bandwidth will prevent internal congestion. The functional role of the Input Sector and the Output Sector can be broken down into several smaller entities. These entities are classified as: *Input Module*, *Output Module* and *Memory Manager.* The Input Module

contains clock and data recovery receivers with SERDES (serializer-deserializers) to de-serialize incoming data; it extracts destination sector information from packet headers and appends each packet to the appropriate pre-store buffer (on-chip memory). The Output Module in the Input Sector executes the *service matrix* by copying packets from pre-fetch buffers (on-chip memory) into the memory of the transmitters that serialize the data and forward them to the central stage crossbars. The Memory Manager strives to maintain both the pre-store buffers empty and the pre-fetch buffers full by shifting packets to/from the SDRAM where they are stored in linked-lists managed by custom hardware. An *Arbiter* is used to create a fair access discipline (round-robin) to the single-port memory for write process and depending on the service matrix for the read process. We chose to make the realisation of the sectors with FPGAs because they allow multiple hardware designs to be investigated without the cost implications of ASICs.

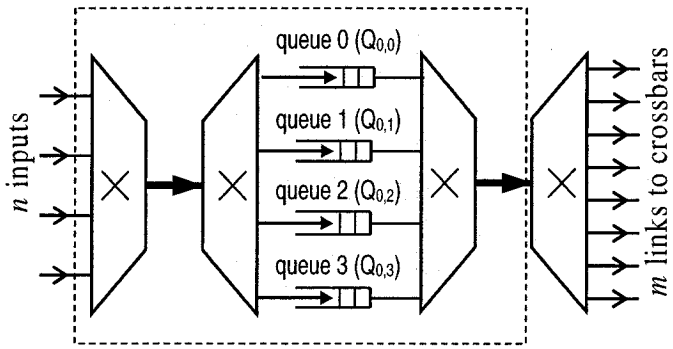


Figure 20: Input Sector architecture

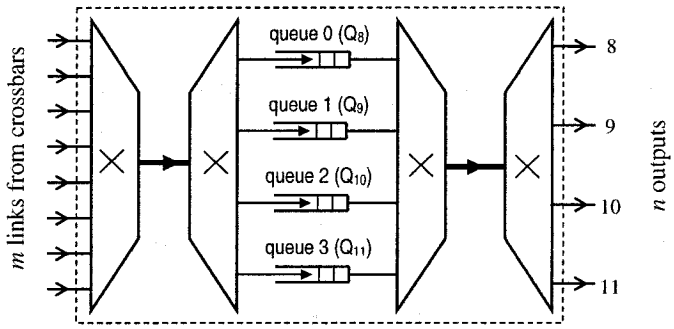


Figure 21: Output Sector architecture

3.3 Output Sector

The O-Sector (Figure 21) performs fine switching. Looked upon as a "neighbourhood" of ports by the preceding stages, it identifies the exact port to where packets are destined and switches packets by placing them in the appropriate queues. Functionally, the O-Sector is a simple 4×4 Shared Memory Switch with output queuing. Packets received from the crossbar stage are pre-buffered (on-chip) and stored in the appropriate output queue (implemented as a linked list) within the on-chip RAM. Whenever the queues become populated, a transmit process serves them in a round-robin fashion by connecting one queue to an output port.

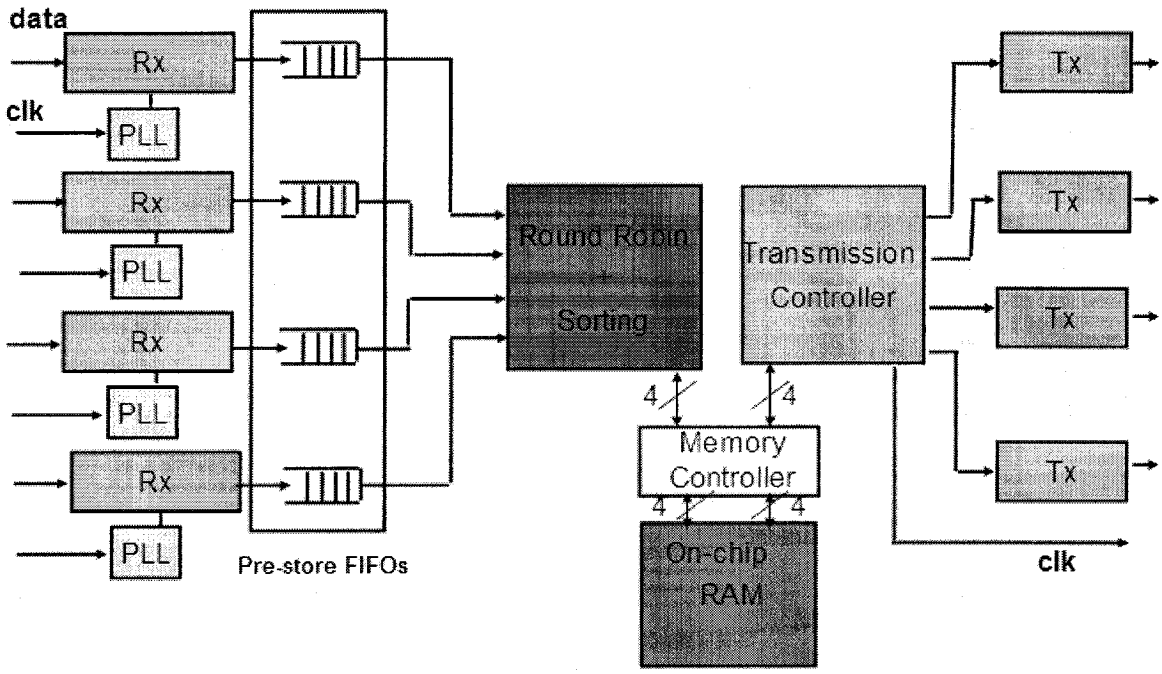


Figure 22: Block architecture of the Output Sector

Communicating across various clock domains was one of the challenges that we faced in the design of the output sector; however, the use of FIFO buffers with different read/write clocks solved it. A detailed explanation of the custom transmitter (Tx) and received (Rx) was presented in the next chapter.

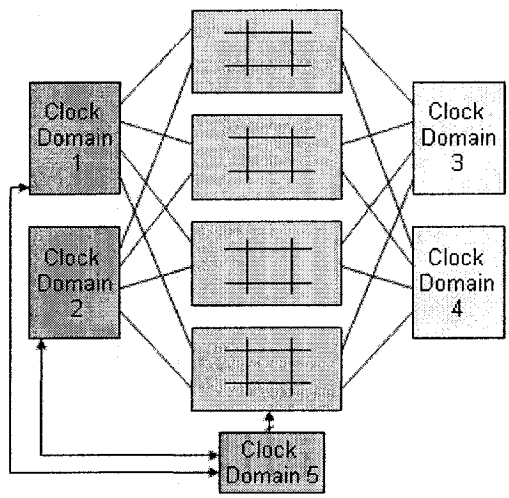


Figure 23: Implemented switch architecture with clock domain indication

The header recognition module in the output sector looks at the 2nd and 3rd bits of the header in order to take the appropriate decision. On the rising edge of the clock it will check the incoming packet; when reading "01", for example, it will assert the write signal of "port 1 FIFO" and put the packet on the data bus. This process, as mentioned earlier, is performed in a round robin fashion as long as the FIFOs of the receivers contain packets. Table 2 shows all possible headers of incoming packets and their correspondent destination output sector and the specific port of the sector.

Header	Destination: Sector – Port
000	1 – 1
001	1 – 2
010	1 – 3
011	1 – 4
100	2 – 1
101	2 – 2
110	2 – 3
111	2 – 4

Table 2: Possible headers of packets

Due to the complexity of accessing off-chip memory, in order to ensure that we are able to access all the memories available on the Nios development board: SDRAM, SRAM, flash and compact flash; we developed a couple of software programs. We included as well a time variable so we can measure the access time to read/write packets.

- *Program 1* implements an algorithm capable of making any single port addressed memory behave as a FIFO buffer. The modifications required to adapt it to different environments, will be to replace the pointer of the test-table by the base address of the memory and the index of the table by the correspondent offset (depending on the data size). Since it is a single port memory, only one input signal (rd/wr) is permitted to request a read operation or a write operation in a clock cycle. A controller will choose which operation he would like to perform at the current clock cycle. In Figure 24, the rd_ptr is the read pointer and wr_ptr is the write pointer. As well, we illustrate 3 examples of pointers position in the circular array implementation of FIFO queue:
    - The first (a) is the empty case where the wr_ptr and rd_ptr point to the same position and the empty signal (empt_fifo in Figure 25 is set).
    - The second case (b) is when the rd_ptr and wr_ptr point to two consecutive addresses respectively.
    - The third case (c) shows a general case with four free addresses.

As shown in Figure 25, when the controller wants to read a packet, the we_en is set to "11", and the address bus will correspond to the rd_ptr value.

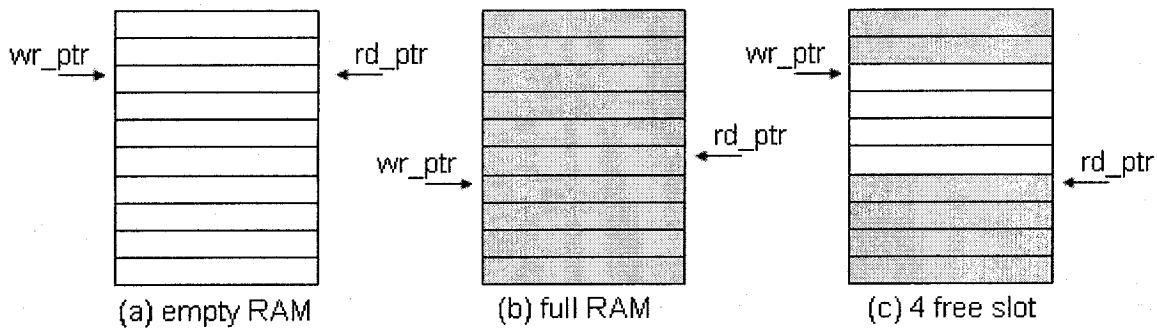


Figure 24: Circular array implementation of FIFO queue

- *Program 2* provides an implementation which is capable of making any dual-port addressed memory behave as a FIFO buffer. Being a dual-port memory, it is capable of reading and writing data in the same clock cycle.

- *Program 3* uses Nios II processor specific functions and .h files in order to access the Micron MT48LC4M32B2 SDRAM initialized in an Altera SOPC design. Using the on-chip timer, initialized in the same SOPC design, we were able to measure the delay required to perform the read/write operations. We used a PLL feed the SDRAM with a phase shift-less clock caused by the on-board routing.

Here's the list of variables used in the implementation of the ASM algorithm presented in Figure 25:

wr_en: it takes four different values.

11: write at the current write address.

00: read from the current read address.

01 or 10: do not perform any operation.

rd_add: is the read address.

wr_add: is the write address.

f_cyc: enable the controller to keep track, whether the rd_add and wr_add offset is more than one cycle.

count: represent the number of packets currently available in the memory.

max: is the maximum number of packets that can be contained in the memory.

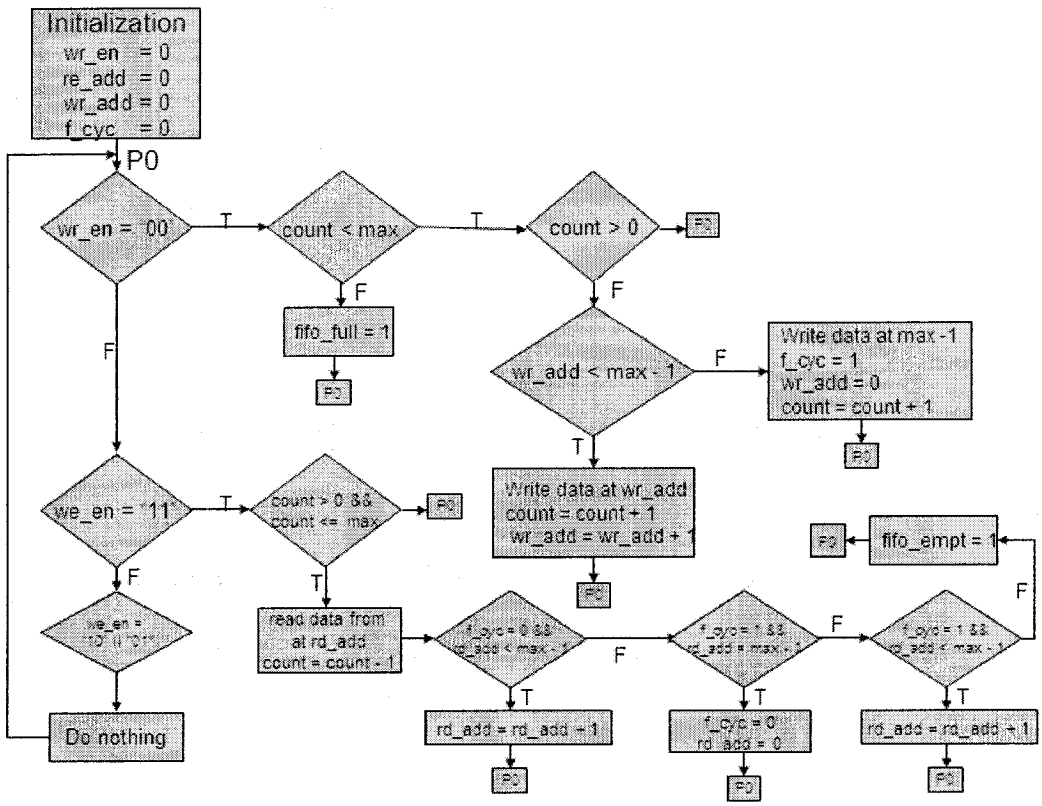


Figure 25: ASM chart of memory controller

Table 3 shows various timing information related to the Micron MT48LC4M32B2 SDRAM available on the Nios II development board. The most important parameter is the access time, which is 5.5 ns in order to perform a single operation, either read or write.

Operation	Operation Time
Initialization refresh cycles	2
Issue one refresh command every	15.625 (us)
Delay after power-up, before initialization	100 (us)
Duration of refresh command	70 (ns)
Duration of pre-charge command	20 (ns)
Active to read or write delay	20 (ns)
Access time	5.5 (ns)
Write recovery time	14 (ns)

Table 3: Access time information for read/write from SDRAM

Table 4 lists all the inputs and outputs pins of the output sector, with a brief description of each one of them. In Figure 27, we present a snapshot of the output sector timing and functional simulation. It is shown that packets are coming to each of the input line, and they are directed to right output port.

Pins (I for Input/O for Output)	Description
clk_inx[1] (I)	Clocks of incoming data
data_inx (I)	Input data lines
a_clr (I)	Global reset for the receiver, including the FIFOs
serial_outx (O)	Output data lines
pll_enx (I)	Data in serially with a leading 1
clk (O)	Clock of the output data

Table 4: Output Sector Input/Outputs pins

Table 5 presents the forming blocks of the output sector. While Table 6 list the Stratix chip resource needed during the synthesis of the output sector. Adding the amount of on-chip memory is possible, and it will result in decreasing the probability of packet loss within the switch.

[1] x takes all these values: 0,1,2 and 3

Blocks	Description
Trans_v0x	Transmitters
top_level_rx_gc2x	Receivers
test_round_rob_w_hr	Round robin serving, header recognizer and write data to the appropriate storing FIFO
memory_controller	Make on-chip RAM look like simple FIFO
Transmission_controller	Read packets from on-chip RAM and write it to FIFO of the transmitters

Table 5: Output Sector Blocks

Resource	Number of elements occupied (percentage)
Logic elements	621 (5 %)
Total pins	167 (39 %)
Memory bits	4096 (< 1 %)
DSP blocks 9-bit elements	0 (0 %)
PLLs	4 (66 %)

Table 6: Chip resource utilization of Output Sector

Figure 26 shows the LEs and memory occupied by the O-Sector, as well the maximum frequency on which it can operate.

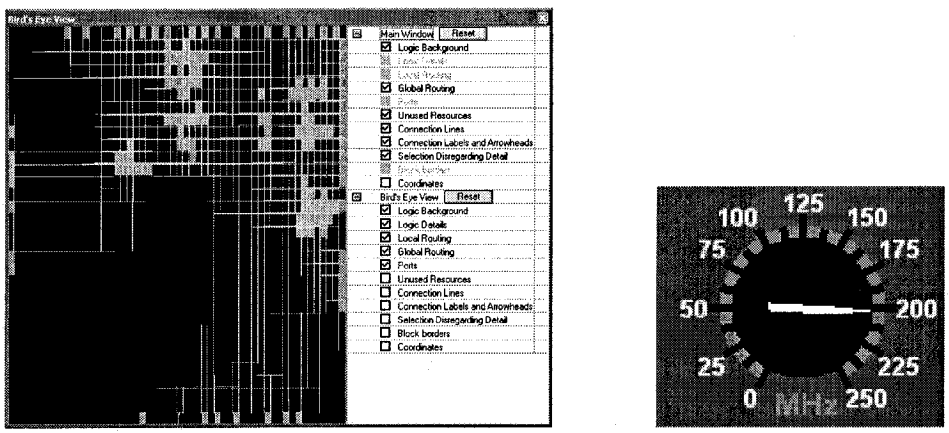


Figure 26: Eye bird' view and fmax illustration of the output sector

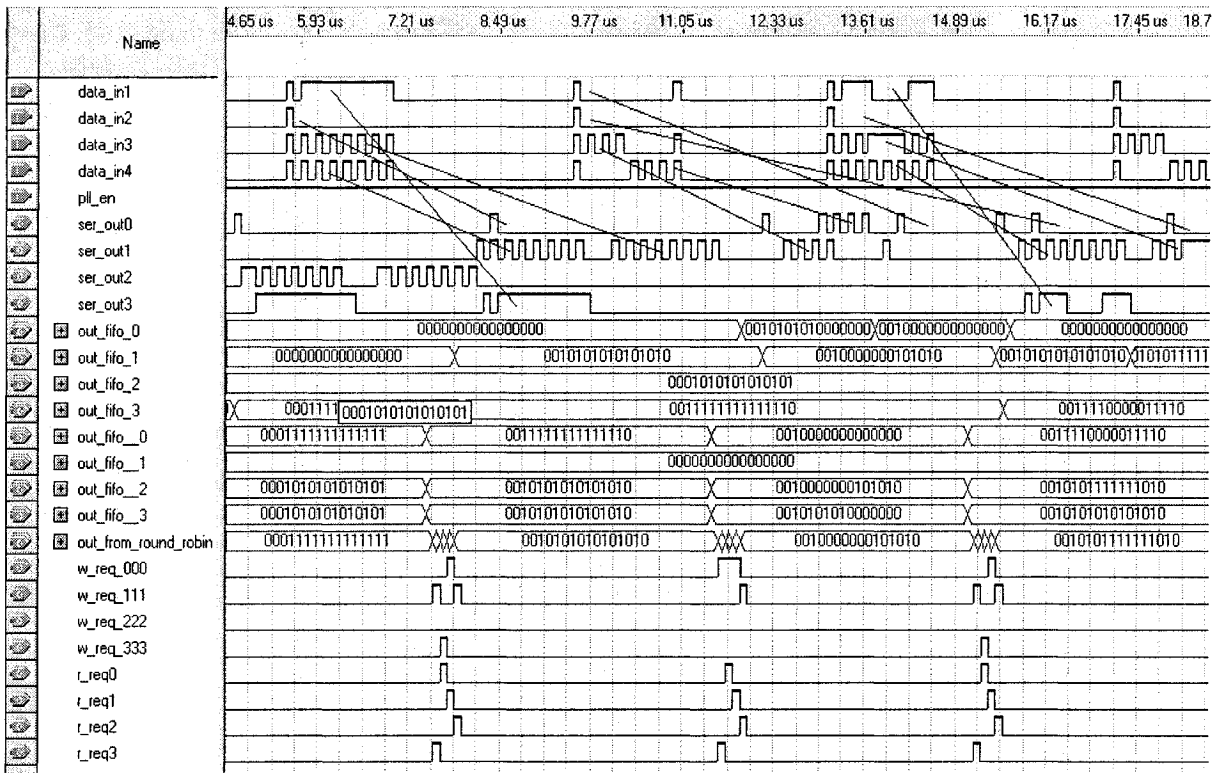


Figure 27: Simulation of 4x4 output sector

3.4 Electronic Central Stage

The central-stage is implemented as an asynchronous configurable entity. The input ports are assigned to their appropriate output ports via the configuration information communicated by the *GC* through a dedicated parallel bus. The 2×2 crossbar switches are implemented electronically with two multiplexers as shown in (Figure 28).

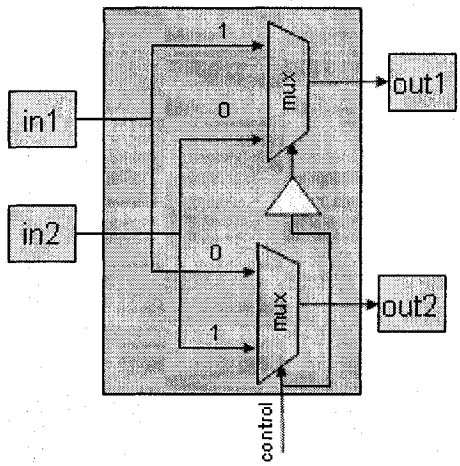


Figure 28: Crossbar implementation using multiplexers and an inverter

A w-bit wide shift register can be used to delay a serial stream in order to emulate a propagation delay between the sectors and the crossbar switch, which can be used to test a distributed version of our demonstrator (Figure 29). The de-serialization factor J and the depth of the shift register determine the amount of delay. Connecting these shift registers in series with a tap placed at the outputs of each section allows for the accumulation of any number of incremental delays, limited only by FPGA hardware. Selection of the desired delay is obtained by connecting the outputs of each shift-register to a w-bit, y-input multiplexer. With a data rate R of 8.9Mbps the shift-register clock rate will be $(8.9/J)$ MHz. The size of on-chip RAM-based memory of 920,448 bits limits the maximum achievable delay per transmission line between the input sectors and the input ports to the crossbars. The maximum delay is then calculated as:

$$\Delta t = \left\lfloor \frac{920448}{m \times 2l} \right\rfloor \times \left\lfloor \frac{1}{w} \right\rfloor \times \left\lceil \frac{1}{(R / J)} \right\rceil$$

$$\Delta t = \left\lfloor \frac{920448}{4 \times 4} \right\rfloor \times \left\lfloor \frac{1}{8} \right\rfloor \times \left\lceil \frac{1}{(8.9 \times 10^{6}/8)} \right\rceil$$

$$\Delta t = 6.5\,ms$$

The resolution (i.e. the smallest delay) is then the amount of time required to shift one w-bit wide word through a single shift register block of depth 1 (a Flip-Flop) and is a fraction of the data clock given by the de-serialization factor: $\Delta t = 0.898\mu\text{sec}$. For optical fibre link, this equals a fibre length of 185[2] meters.

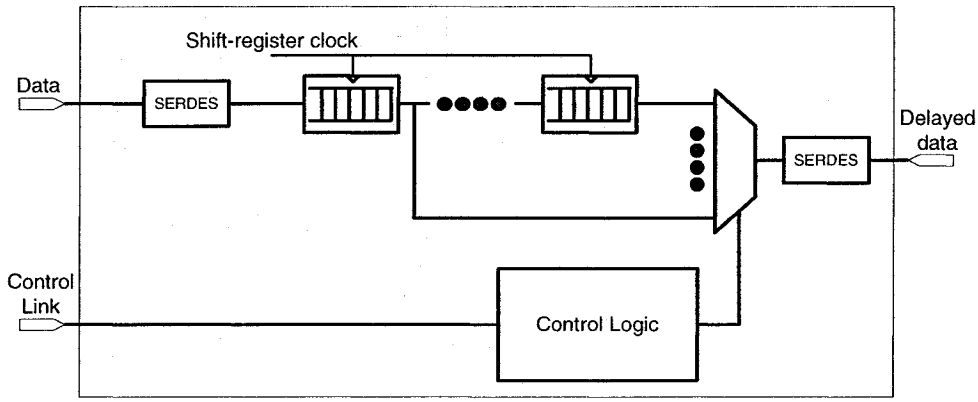


Figure 29: Link delay implementation

Because the communication protocol requires transmitting the clock in parallel with the data, we implemented the crossbars in 2 levels where the clock and the data are switched together depending on the control data. As it is shown in Figure 30, the clocks of both of the input sectors are inputs to the x_bar module.

[2] The speed of light in glass is c/nref, where c is the vacuum light speed and nref is the fibre core refractive index which is about 1.45 at 1550 nm.

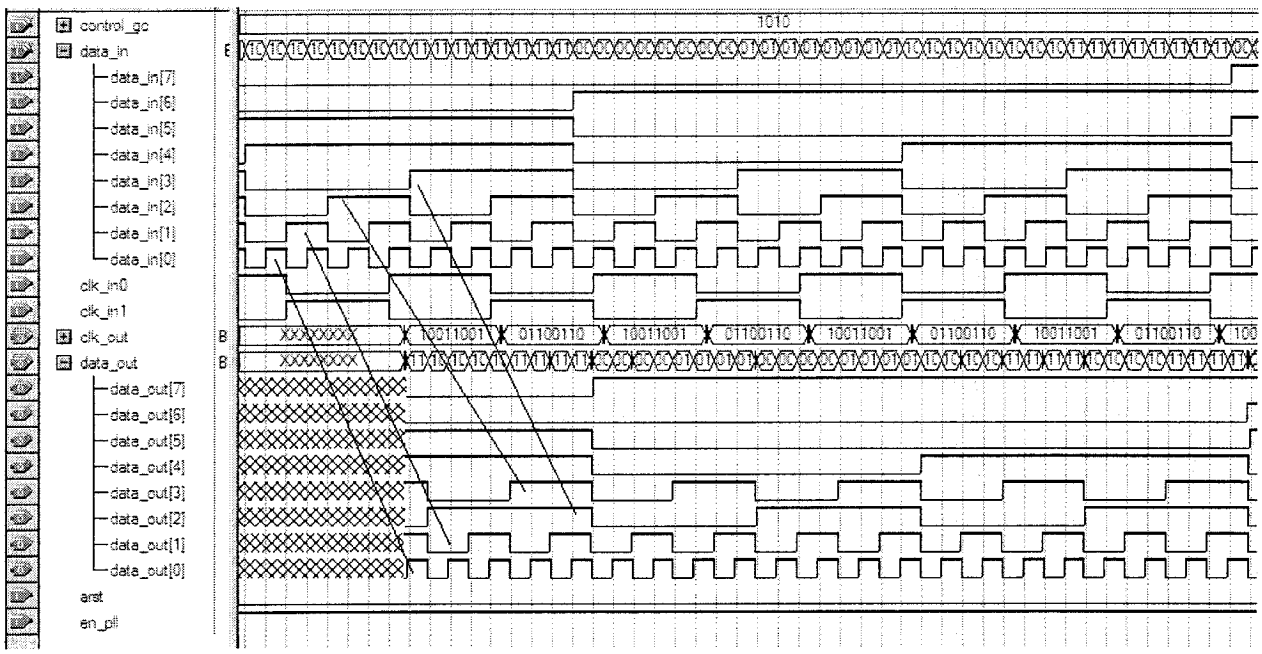


Figure 30: Simulation of the crossbars showing data switching

3.5 Optical Central Stage

A ROADM is a multi-input, multi-output optical switch that routes a number of wavelength division multiplexed (WDM) channels in and out of a network. In a ROADM, the signals remain in the optical mode, without the need for O-E-O transitions, and the number and wavelength of 'add' and 'drop' channels can be controlled and changed in real time. One of the technologies for implementing the ROADM functionality is the wavelength selective switch (WSS). The WSS 5400, shown in Figure 31, is part of a family of WSS products based on the Metconnex HI-PLC platform. It consists of a hybrid integration of PLC and MEMS. It is considered only half of a ROADM since it can add or drop wavelength at the same time. This device has multiple functionalities [Metconnex 2005]:

- De-multiplexing
- Any to any, hitless switching

- Wavelength blocking
- Per wavelength or per port power attenuation
- Multiplexing

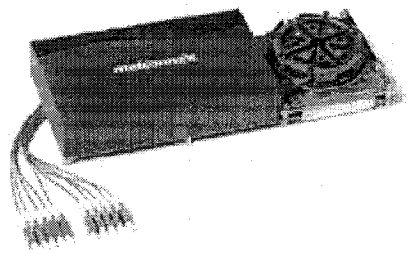


Figure 31: WSS 5400 an outside and inside look

Interface	Bit Rate
RS232	115.2 Kbps (max.)
I2C	400 Kbps (max.)
SPI	Variable

Table 7: Available interfaces on the WSS

To use the WSS devices in optical crossbar switches, we set them up in the configuration presented in Figure 32. We will use both of the JB 9 interfaces available on the Nios development board in order to change the configuration of the WSS 5400 upon finishing the FBP calculation carried by the Global Controller.

The diagram in Figure 32 shows the different parts of our opto-electronic packet switch. The input and output sectors will be implemented on Stratix GX chips mounted on Altera High-speed development boards. The simulation and timing analysis of the output sector were verified on the GX architecture. The development board was fully characterized; the optical output will be modulated using Eudyna' SFP.

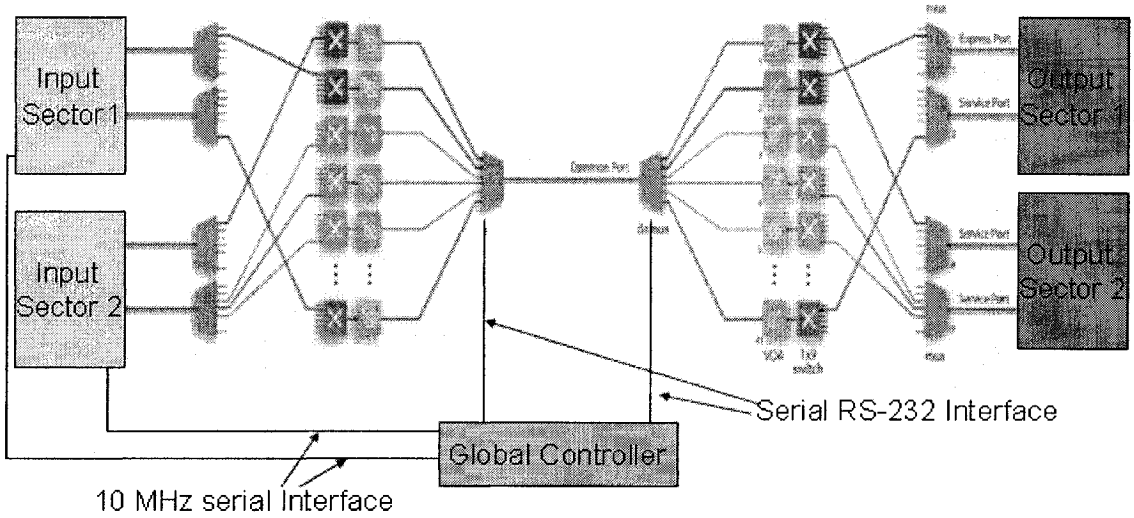


Figure 32: Optical core using WSS

3.5.1 Optical Measurements

When designing optical systems in general and DWDM systems in particular, it is important to know the characteristics of the modules used, such as Insertion Loss, Return Loss, and Switching Time. For example, the power of the laser can be adjusted to minimize power consumption and other techniques can be used to mitigate signal degradation caused by non-ideal performance of the optical components. In our application, the switching time is the most important parameter since it is vital for the GC to know how much time it will take the core to perform the new configuration so it will alert the input sector to start transmission. In this section, we show the result of various optical tests performed on the WSS 5400.

Test Equipment Used:

- HP 83440D Lightwave Detector 1300/1500 nm,
- JDS Uniphase Multiple Application Platform:
    - Tunable C+L band Laser,

- Power Meter,

- Tektronix 2465 Oscilloscope,
- Multi-wavelength Laser System, L-Band, Peleton-Photonic Systems,
- Tektronix PS280 DC Power Supply,
- Agilent, 86142B Optical Spectrum Analyser,
- Circulator,
- Fibre Optics Patch Cord.

Return Loss

Light reflection or scattering is an undesirable phenomenon when transmitting light through an optical component due to the effect it may cause in the system, such as causing inline amplifiers to oscillate (i.e., become lasers). It is important to measure the amount of reflections for the components in a DWDM system. The reflection factor for a component is a measure of how much light the component reflects. It is the ratio of the power reflected by the device to the power incident on the device. The return loss (RL) has units of dB. The return loss of the WSS 5400 was measured using the test setup shown in Figure 33. It is defined as follow:

$$RL = -10 * \log 10^{\Pr eflected / \Pr eference}$$

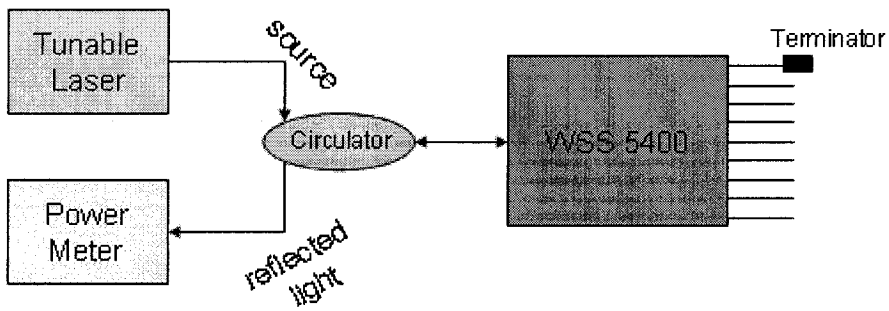


Figure 33: Test setup for RL measurements of WSS 5400

The value stated in the data sheet is 40 dB. As shown in Table 8, it is clear that our measurements are close to the specifications for most of the ports, expect ports 5 and 6, where the difference is be caused by fiber connection.

Port Number	Return Loss (dB)
1	35.81
2	35.80
3	37.00
4	33.99
5	28.20
6	29.84
7	36.50
8	36.40
9	36.40

Table 8: Measured Return Loss

Insertion Loss

Insertion loss (IL) is the amount of loss that the light experiences when it is passing through the device under test. It is important to measure Insertion Loss in order to make sure that the output power is enough so that the optical receiver will sense the incoming data with an acceptable BER. IL is defined as follows:

$$IL = 10 * \log \Pr eference / Pout$$

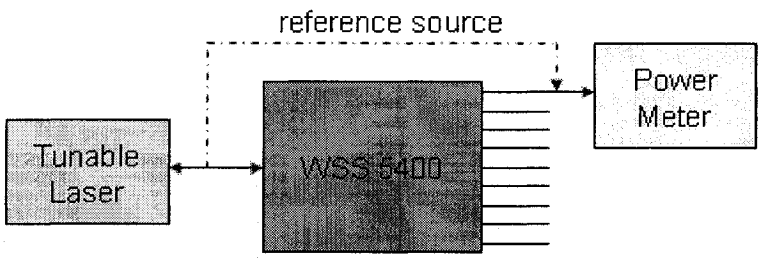


Figure 34: schematic of Insertion Loss testing

Port Number	Insertion Loss (dB)
1	5.18
2	5.18
3	5.10
4	5.02
5	5.05
6	5.02
7	4.94
8	5.02
9	4.18

Table 9: Measured Insertion Loss

Figure 34 shows the test setup used to measure the Insertion Loss of the WSS 5400, and Table 9 shows the values measured. The maximum value stated in the product data sheet is 6.5 dB.

Switching Time Test-bed

In order to measure the switching time of an optical switch that is being controlled using a RS-232 interface, we developed an FPGA-based test-bed. It consists of a GUI, shown in Figure 35, developed in Java to display the switching time and send the request, as well an FPGA programmed with a custom state machine. We developed the controller using VHDL.

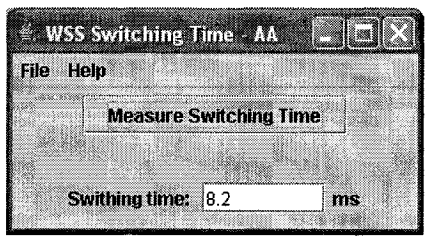


Figure 35: Custom GUI used to measure the switching time

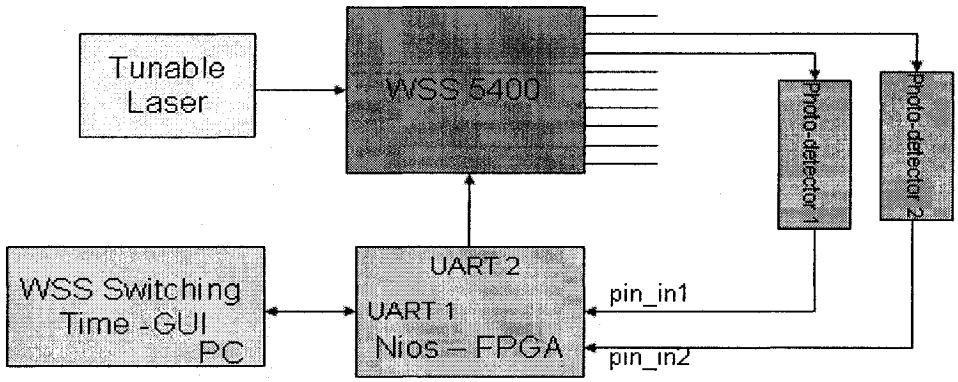


Figure 36: Test-bed used to measure the switching time

The principle of operation of the test-bed is as follows: if the UART1, receives the binary correspondent to ASCII "h", it will send the command sop[25][8] (all commands are available in [Metconnex 2005]) through UART2 in ASCII format, and will keep checking for pin_in1 to rise to "1". After the switching takes place, wavelength 1550.10 nm should appear on port 8 of the WSS. The photo-detector will convert the optical signal to electrical DC voltage. Since the output of the photo-detector is LVTTL standard, we expect a 3.3 V output from the photo-detector. When the pin_in1 is logical "1", it will enable the counter and will wait to pin_in2 to become "0". When this happens, the counter is disabled and sends its value to the serial port of the PC through UART1. The GUI will then display the value. The value will also be displayed on the 7-segment BCD display of the Nios II development board. The controller equipped with 2 UARTs occupied 363 logic elements and 512 bits of on-chip memory. Figure 37 shows the state machine of the VHDL controlling the test-bed, where "cnt" is the counter enabling signal and Idle is the initial state. Due to lack of equipment, mainly a second photo-detector, we could not examine this test-bed fully in hardware. So we tested the interface of one port with a photo-detector, we simulated the whole design in Quartus II and tested the interaction of the state machine with the custom GUI.

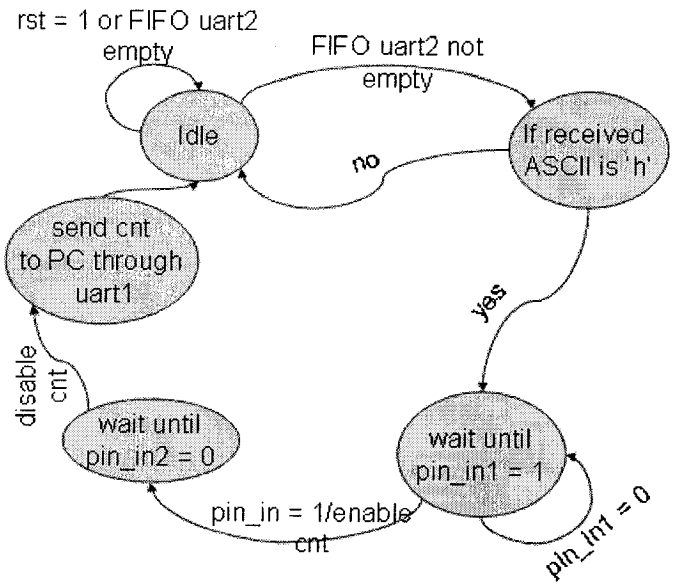


Figure 37: State machine of the switching time test-bed controller

Custom RS-232 Interface

In order to communicate with the WSS 5400, we need to send the commands as ASCII characters over the RS-232 interface. It would be better to develop a custom I2C interface because of its higher speed; however, the development board on which the WSS is mounted has only RS-232 interfaces. The UART packet, shown in Figure 38, consists of a start bit, 8 bits of data, parity bit and a stop bit. The data line in this protocol is always at high voltage unless a packet is being transmitted.

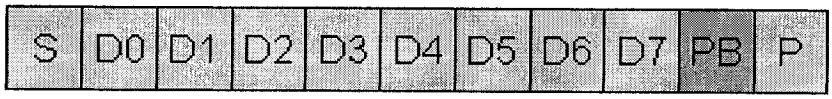


Figure 38: UART packet format

Figure 40 and Figure 42 present the state machine of the UART transmitter and receiver respectively. Since the UART is asynchronous, it relies on the transmitter and

receiver having the same frequency, requiring only the receiver's clock phase to be adjusted.

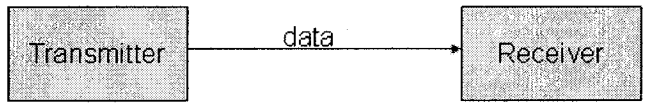


Figure 39: Asynchronous data transfer

Figure 40 shows the various states and conditions that control our custom UART transmitter, where Idle is the initial state.

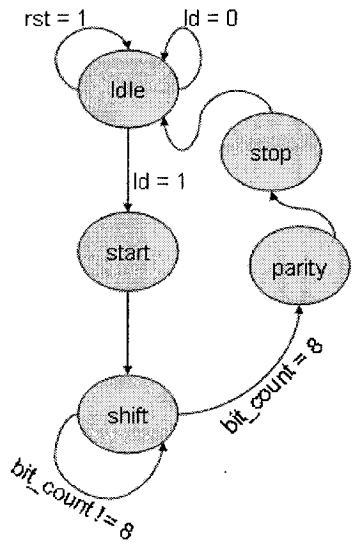


Figure 40: State machine diagram of UART transmitter

The output line is at high voltage until a packet is loaded in order to be sent out, as it is shown in Figure 41. When this happens, the multiplexer will be configured to permit the departure of the packet.

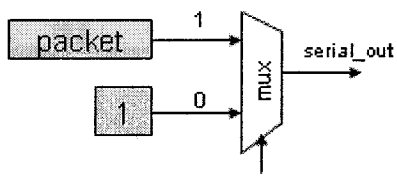


Figure 41: Multiplexer at the UART output

The receiver state machine presented in Figure 42 consists of 5 states. The status will stay in the idle state as long as the ser_in, the data input signal, is high logic, 3.3 V in this case. Once it senses a start bit, a counter will be enabled in order to count 8 data bits at 115200 Hz. A custom clock divider was developed in order to convert the on-board 50 MHz clock to the appropriate UART clock value. After checking the parity of the data, in the case of matching between the sent parity bit and the calculated one, the state machine will return to idle, otherwise, it will report an error before returning to idle.

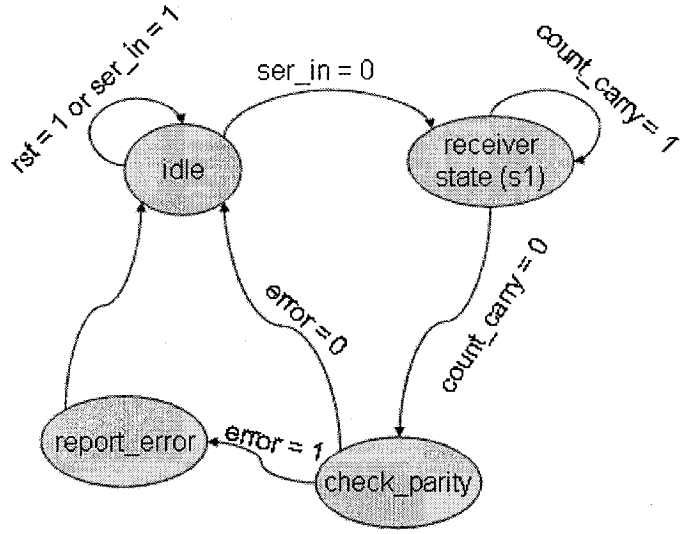


Figure 42: State machine diagram of UART receiver

In the following text, we show part of the VHDL implementation code of the algorithm for generating/checking the parity in the UART transmitter and receiver.

```
if (enable = '1') then
        temp := '0';

for i in packet_in'RANGE loop
        if packet_in(i) = '1' then
          temp := not temp;
        end if;
end loop;

   parity_out <= temp;
else
   parity_out <= 'Z';
end if;
```

The developed hardware UARTs may be used for pure HDL implementation of the FBP method, in order to control the optical core formed by the two WSS 5400. Beside the hardware implementation, we initialized a Nios II SOPC system with 2 "soft-UARTs" in order to run the software version of FBP, which will execute the configuration of the optical crossbars. The communication over UART in the Nios II is a relatively simple task, since it is programmed as in Unix where all the hardware modules of the system are treated as files. So when we require a write to the serial port we use *fwrite* and for reading from the serial port we use *fread*, with a file pointer generated by *fopen* and the data as arguments. If the Nios I was to be used, this task would require using specific Nios functions such as *nr_uart_rxchar*, *nr_uart_txchar* ...

4 Chapter 4: Description and implementation of FBP

The Flexible Bandwidth Provision algorithm for the distribution of internal bandwidth in the packet switch used in this work was presented in detail in [Paredes 2005]. Here, we will present an introduction to explore FBP, followed by both the Hardware implementation using VHDL and the Hardware/Software co-design using VHDL and C programming languages running on the Nios II embedded soft-processor. Both implementations were optimized and tested on Altera's Stratix FPGA. More detailed information of the FBP algorithm is presented in Appendix D.

4.1 Description of the FBP Algorithm:

To formulate the Flexible Bandwidth Provision algorithm, the switch architecture is represented by a regular bipartite graph; i.e., a bipartite graph in which all vertices has the same number of adjacent edges. In our representation, the input and output sectors correspond to the vertices and the edges correspond to the interconnections/paths between the two sets of sectors. The graph is built from an *inter-sector service rate matrix*, S, whose integer entries specify the bandwidth that will be provided between input/output sector pairs, measured in number of paths between them. Since the input sectors are represented by the rows of the matrix and the output sectors by the columns; the sum of each row and column must equal the number of paths in the central stage available for each sector; but the individual entries s_{ij} may vary to provide a different number of paths (and hence variable bandwidth) between sector pairs.

The configuration of paths in the central stage is found by decomposing this bipartite graph (which corresponds to solving an edge-colouring problem). The matrix *S*

is decomposed into *m* permutations; each one of them defines the state of a crossbar switch. Using these permutations, the m switches in the central stage and the read operations in the input sectors are configured. Every τ time slots (the inter-configuration period), the inter-sector service rate matrix is calculated from the estimated *inter-sector traffic matrix*, whose elements correspond to the number of packets to be transported between input sector i and output sector j. The traffic matrix may be derived from rate requests or estimated traffic arrivals. To achieve 100% throughput, one must ensure that the service rate of the VOSQs (the matrix elements s_{ij}) exceeds the arrival rate (set by the traffic statistics). In this context, most of the switching of packets occurs within the sectors, while the central stage mainly transports packets across the switch.

We will start with a simple example showing the steps of the FBP algorithm; this will help in introducing the idea and understanding the modules developed. Our demonstrator architecture (Figure 43) consists of two I-Sectors, two O-Sectors at the edges and four crossbars in the core.

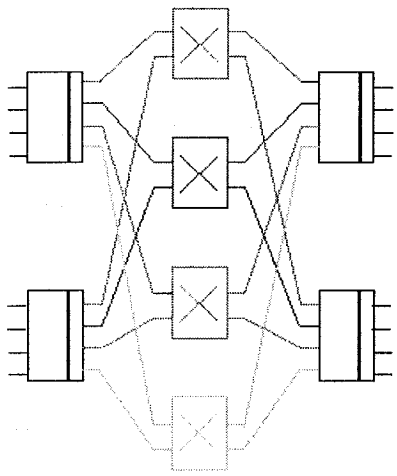


Figure 43: Dimensions of the switch demonstrator (N=8, I=2, n=4, m=4)

$$\Lambda_e = \begin{bmatrix} 14 & 35 \\ 43 & 5 \end{bmatrix} \longrightarrow S = \begin{bmatrix} 1 & 3 \\ 3 & 1 \end{bmatrix}$$

Figure 44: Example of traffic matrix and service matrix

The first matrix in Figure 44 is the traffic matrix, whose elements correspond to the measured/estimated traffic arrivals. It is constructed using the "Measure Incoming Traffic" function (shown in Figure 46) and is used as input to the "Service Matrix Calculation" function (shown in Figure 46) whose output is the service matrix ***S***. Its integer entries specify the number of paths to be set up between sector pairs. The decomposition of ***S*** and the correspondent switch configuration are shown in Figure 45:

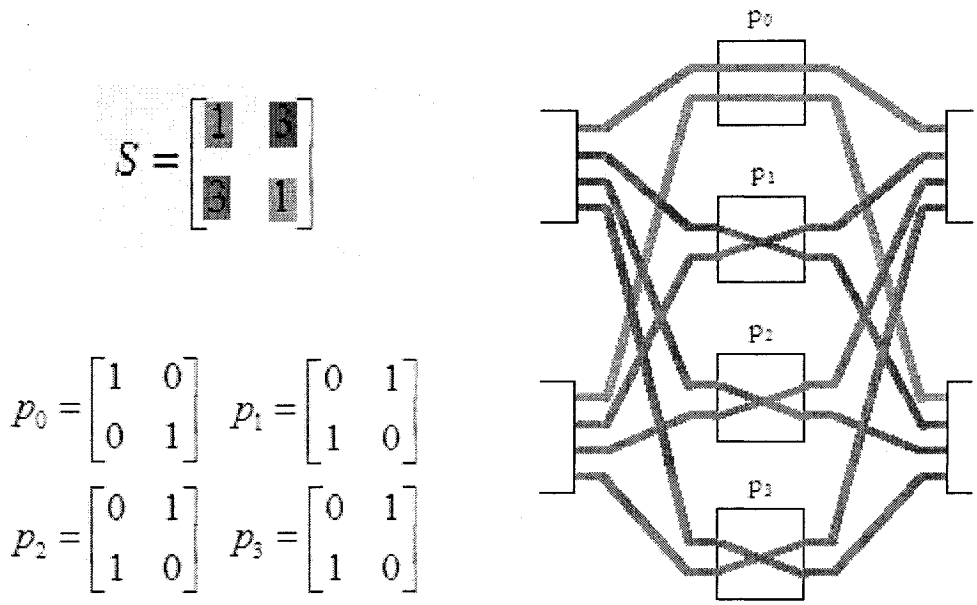


Figure 45: Example of permutation matrices and correspondent switch configuration

The intelligence of the switch is centered around the functionality of the Global Controller. Its sole function is to execute the FBP algorithm, and respond with the

appropriate configuration parameters for both the Input Sectors and the Crossbar switches. The process of acquiring traffic information (queue lengths) from the input sectors occurs at regular time intervals defined as the *inter-configuration period* of the switch. The Global Controller will notify the input sectors by sending a special control word containing the value, in time-slots, for the inter-configuration period.

Two implementations of the FBP algorithm were designed and tested. The first one was made in C language running on a NIOS™ II processor. The communication between the Global Controller and the I-Sectors is performed over dedicated 10 MHz service links. The transmitters and receivers are interconnected to the Nios II through the Avalon Bus and their functions are implemented in VHDL code. Given today's FPGA technologies, the entire design can be easily implemented on a single FPGA chip exploiting its embedded RAM for instruction and data memory. The second implementation was in VHDL. Figure 46 shows the block diagram of the functions coded to implement the global controller (GC). It measures the traffic demands, performs the FBP algorithm and configures the central stage of the switch.

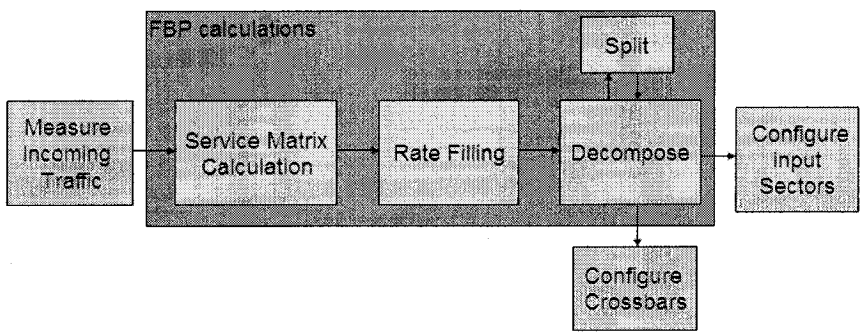


Figure 46: Global Controller (GC) function blocks

4.2 Definition of the Communication Protocol between the Global Controller and the Input Sectors

Definition of control packets: There are three types of 16-bit packets that the I-sector can receive:

1) inter-configuration period packet (Time interval before sending the Traffic-Matrix, τ)

2) reconfiguration-time packet (Central Stage's configuration time, also known as *switch overhead*)

3) service-matrix packet

Upon power-on, the input sector waits to receive the packet with the value for inter-configuration period. After receiving this packet, the sector can then receive the reconfiguration-time packet and then the Service-Matrix packet once the switch is in full operation. Configuring the I-Sectors consists of setting the appropriate read operations from memory [Bishtein 2004].

Header Bits	Command Interpretation
00	Service Matrix
01	inter-configuration period
10	reconfiguration-time
11	Unused

Table 10: Header protocol for control packets going from the global controller to the input sectors

The packet from I-Sector to the GC does not contain header bits, since the only information need to be sent is the traffic information. In Figure 47, we present the format of the message.

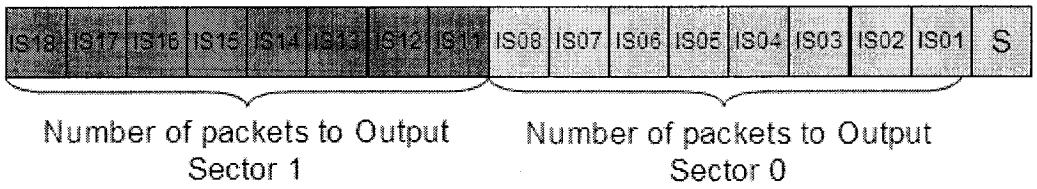


Figure 47: Message format sent from an I-Sector to the global controller

4.3 Hardware Development of the Customized FBP

4.3.1 Transmitter / Receiver Design and Implementation

There are four well known methods to synchronize data transfers (one listed in chapter 2, one in this chapter and the other two are listed in Appendix B). We chose the one in Figure 48 due to the availability of PLLs on the Stratix chip. In order to receive the data correctly, the PLL at the receiver side should be locked in terms of speed and phase with the clock signal sent by the transmitter. This synchronization method is applied in the communication in both directions (from I-Sectors to GC and from GC to I-Sector).

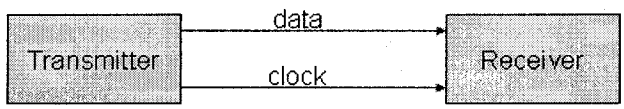


Figure 48: Type of synchronization used in control data transfer

The schematic of the modules below are presented in appendix C.

10 Mbps Transmitter (trans_v0): Figure 49 shows the state machine of the transmitter controller. The custom transmitter is in the idle state (which is also the initial state) as long as the FIFO transmission queue is empty or the reset is '1'. As soon as a packet arrives at the transmission queue, which is equivalent to the FIFO empty signal being equal to 0, the read request is enabled and a packet is loaded on to the parallel-to-serial module. A start bit is included in the packet and the shifting will take place until 17 bits

are counted. When count_carry is set to '1' it means that transmission of the packet has been completed and the process then goes back to the idle state. In Table 11, the interface signals are presented along with their description. Table 11 shows the modules that form the transmission functionality.

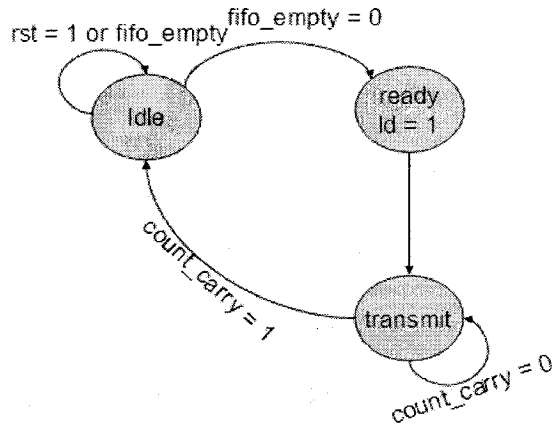


Figure 49: State machine of transmitter controller

Pins (I = Input,O = Output)	Description
w_req (I)	Request to enable writing on the FIFO
input_fifo[15..0] (I)	Data to be transmitted
rst (I)	Global reset for the transmitter, including the FIFO
clk (I)	Clock to feed the modules
serial_out (O)	Data out serially with a leading 1
clk (O)	Clock used to feed the next device

Table 11: Transmitter Input/Output pins

Blocks	Description
mega_fifo_receiver1	Storing FIFO before transmission
trans_controller	Maestro of the transmitter
cnt_16_trans	Counter set by trans_controller and used to alarm when transmission is done
parallel_to_serial	Parallel to serial converter explained below
ff_lpm_v0	Flip-flop used to enable parallel_to_serial block

Table 12: Forming blocks of the Transmitter

Parallel to serial converter module: Figure 50 shows a small version of the parallel to serial converter used in the transmitter. The implemented one contains 17 registers. This module operates as following: when ld, the load signal, is 1, the multiplexer will enable the path of the data to be loaded into the flip-flop. So when ld is back to 0, the path between flip-flops is enabled and the shift process will take the bits, at the rising edge of the driving clock, from one to another until all the whole packet are out.

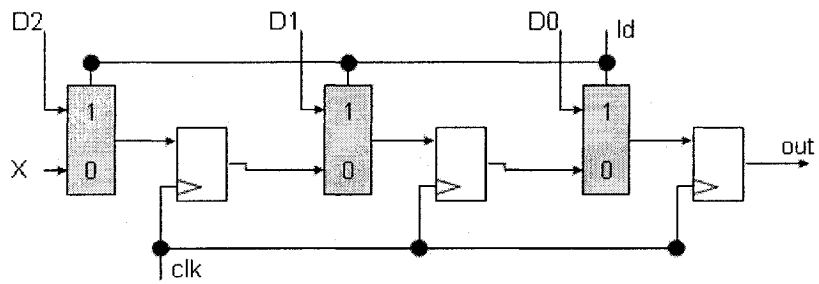


Figure 50: Three-bit version of a parallel to serial converter

10 Mbps Receiver (top_level_rx_gc2): Using the synchronization method described in Figure 48, we implemented our state machine presented below. For as long as the line at the input port, ser_in, is 0 we stay at the ready state (ready to receive packet). Once we receive a starting bit of 1, we enable the shift register, which performs the serial to parallel operation, and we start a counter that will notify us when the count reaches 16 (count_carry equal to 1), which corresponds to the length of the control packets sent from the GC to the input sectors. After a count carry, we return to the "ready" state, and wait for a new packet. Table 13 presents the interface signals and their descriptions. Table 14 shows the modules that form the receiver functionality.

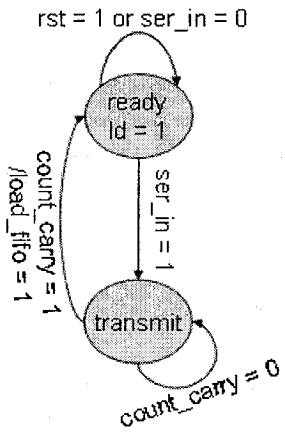


Figure 51: State machine of the receiver controller

Pins (I = Input, O = Output)	Description
pll_en (I)	Enable signal to activate the PLL
out_fromfifo[15..0] (O)	Data received in 16 bits format
a_clear (I)	Global reset for the receiver, including the FIFO
clk_in (I)	Clock of the received data
in_ser (I)	Data-in serially with a leading 1
full (O)	To determine if the FIFO is full or not
Empty (O)	To determine if the FIFO is empty or not

Table 13: Receiver Input/Outputs pins

Blocks	Description
mega_fifo_receiver1	Storing FIFO while reception
receiver2_ctr	Maestro of the receiver
mega_counter3	Counter set by receiver2_ctr and used to alarm when reception of one packet is finished
pll_mega2	Phase Locked Loop used to synchronize the receiver with data clock
mega_shift_3	Shift right module controlled to behave like serial-to-parallel converter

Table 14: Forming blocks of the Receiver

In order to be able to simulate the whole packet switch in VHDL simulation software such as Modelsim, and to explore the possibility of developing a custom ASIC of the switch controller, we implemented a simplified version of the FBP algorithm.

4.3.2 FBP Main State Machine

The implementation of FBP in hardware requires two transmitters, two receivers, four parallel interfaces to configure the crossbars, a FBP controller, and a computing module. In Figure 52, we illustrate the state machine of the Global Controller. Here's a description of the states:

Init: is the state of the switch at power up. Note that at any time when *g_rst* is set, the configuration of the switch is reset.

inter_s: the inter-configuration period is sent to the input sectors.

conf_s: the crossbar configuration is sent to the input sectors.

s_0: is the initial state of the FBP algorithm, where the process stays until it receives the traffic information from the I-sectors.

s_1: after receiving the VOSQs lengths, the process performs the computation of the service matrix and its decomposition in this state.

s_2: after computing the switch configuration, in this state the GC transmits the appropriate configurations to the input sectors and the central stage. The process will then loop back to s_0 in order to wait for the new traffic information.

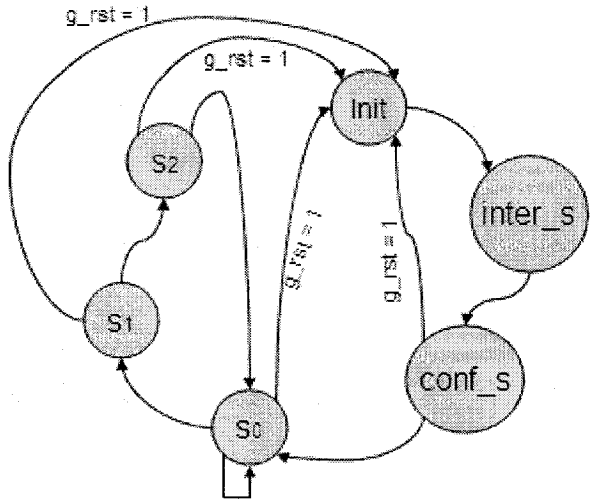


Figure 52: Global Controller state machine

Here's the description of the implemented function blocks:

stage1_fbp: This hardware module is the first stage in the FBP calculation. It takes the 16-bit data received, and splits it into four bit arrays, each representing the number of packets at each I-Sector destined to each O-Sector. These numbers correspond to the elements of the Traffic Matrix. The module then obtains the row sums and column sums of the Traffic Matrix. Finally, it calculates the maximum of these sums and sends this value to the "max" output pin. Table 15 indicates the Input and Output pins of "stage1_fbp" block.

Pins (I = Input,O = Output)	Description
Data_rx0[15..0] (I)	Received queue lengths from sector 0
Data_rx1[15..0] (I)	Received queue lengths from sector 1
a_clr (I)	Global reset for the registers and DSP blocks
clk_en (I)	Clock enable
clk (I)	Global clock for the whole stage1_fbp
w_req0 (I)	Write request for register assigned to store the data received from sector 0
w_req1 (I)	Write request for register assigned to store the data received from sector 1
vosq00[7..0] (O)	Number of packets at I-Sector0 destined to O-Sector0
vosq01[7..0] (O)	Number of packets at I-Sector0 destined to O-Sector1
vosq10[7..0] (O)	Number of packets at I-Sector1 destined to O-Sector0
vosq11[7..0] (O)	Number of packets at I-Sector1 destined to O-Sector1
max[8..0] (O)	Maximum of the column wise or row wise additions

Table 15: Input/Outputs pins of module "stage1_fbp"

stage2: In order normalize the traffic matrix formed by the "stage1_fbp" module; we need to divide all the traffic information (VOSQ lengths) by the maximum row/column sum obtained, which will result in floating point numbers. Since the DSP mega-functions of Quartus work only with integers, we shifted all the matrix elements four positions, which is equivalent to multiply it by 16. Then we performed a comparison between each of the results with eight, which is the equivalent of 0.5 multiplied by 16. If the result of the division is bigger than 8, generate a one, if it is smaller then generate a zero. The interface signals of "stage2" block are presented in Table 16. min_serxx represents the position of the result in the service matrix.

Pins (I for Input/O for Output)	Description
min_ser00 (O)	Minimum service matrix at position 00
min_ser01 (O)	Minimum service matrix at position 01
min_ser10 (O)	Minimum service matrix at position 10
min_ser11 (O)	Minimum service matrix at position 11
a_clr (I)	Global reset for the registers and DSP blocks
clk_en (I)	Clock enable
clk (I)	Global clock for the whole stage1_fbp
w_req0 (I)	Write request for register assigned to sector 0
w_req1 (I)	Write request for register assigned to sector 1
vosq00[7..0] (I)	Number of packets at I-Sector0 destined to O-Sector0
vosq01[7..0] (I)	Number of packets at I-Sector0 destined to O-Sector1
vosq10[7..0] (I)	Number of packets at I-Sector1 destined to O-Sector0
vosq11[7..0] (I)	Number of packets at I-Sector1 destined to O-Sector1
max[8..0] (I)	Maximum of the column wise or row wise addition

Table 16: Input/Outputs pins of module "stage2"

rf_sm: This VHDL module implements a look-up table, which takes into consideration all possible outcomes from the stage2 phase. This has been done in order to avoid performing all the calculations every time and hence make the operation of the global controller faster. "min_sm[3..0]" is a collection of the "min_serxx[3]" outputs in stage2. For each possible case, we made the calculations for the correspondent configuration of the crossbars and I-Sectors and made an entry of the look-up table. The return value when the look-up table is accesed corresponds to the control commands sent to the input sectors and the central stage. Table 17 illustrates the interfaces of the "rf_sm" block; the outputs of rf_sm module are the configuration of the switch. The I-Sectors configuration are sent serially using the transmitter described above, as for the crossbars configuration four output wires connecting the GC to the crossbars are used.

[3] xx, will take 00, 01, 10 and 11.

Pins (I for Input/O for Output)	Description
en (I)	Enable signal
min_sm[3..0] (I)	Minimum service matrix calculated by stage2
rst (I)	Global reset for the module
clk (I)	Clock of the received data
sm_is0[3..0] (O)	Service matrix (control command) to be sent to I-Sector0
sm_is1[3..0] (O)	Service matrix (control command) to be sent to I-Sector1
x_congif[3..0] (O)	Crossbars configuration bit stream

Table 17: Input/Outputs pins of module rf_sm

Test_4_f: This is the final stage in the FBP hardware implementation. This module makes a comparison between the new configurations and the old ones. The comparison is implemented in "comp_gc_out.vhd", with the state machine presented in Figure 53. The process remains in state s0, where we do not take any action, if the new configuration is the same as the previous one. If the new configuration is different, then the process moves to state s1, where we overwrite the old configuration with the new one and the corresponding control commands are sent to the input sectors and the central stage. The previous configurations are saved in a generic register, implemented in "reg_8.vhd".

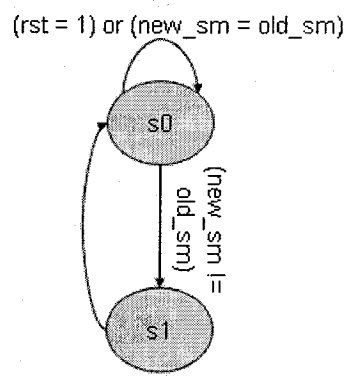


Figure 53: State machine of the configurations comparison method

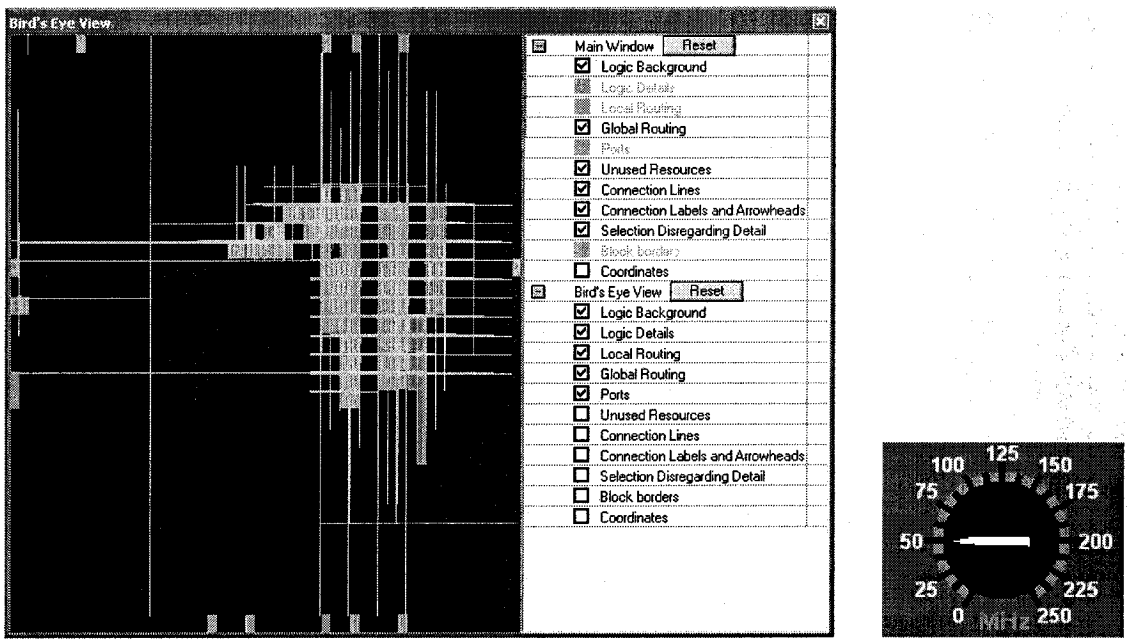


Figure 54: Bird's Eye View snapshot and fmax of FBP Hardware

Figure 54 shows the area, composed of LEs and on-chip memory, occupied by the FBP implementation on the Stratix chip. The exact percentage and number of resources used are presented in Table 18. The final schematic and simulations of the FBP and the sub-modules are presented in Appendix A.

Resource	Number of elements occupied (percentage)
Logic elements	757 (7%)
Total pins	25 (7%)
Memory bits	512 (< 1%)
DSP blocks 9-bit elements	4 (8%)
PLLs	2 (33 %)

Table 18: Chip resource utilization of hardware FBP

4.4 Mixed Hardware/Software Design and Implementation of FBP

Since the FBP algorithm contains a lot of complex and integer computation phases, it is hard to make a generic or scalable implementation in ASIC. In order to develop a generic FBP controller that provides the option to vary the number of crossbars and input sectors; we had to implement all the functionality in C, running on Nios II processor. In terms of computational speed, it is well known that a soft-processor such as Nios II is not the best choice. However, the capabilities such as integrating custom hardware and adding custom instruction sets are advantages of soft-core processors over hard-core processors (such as PowerPC 405 or Motorola HC12).

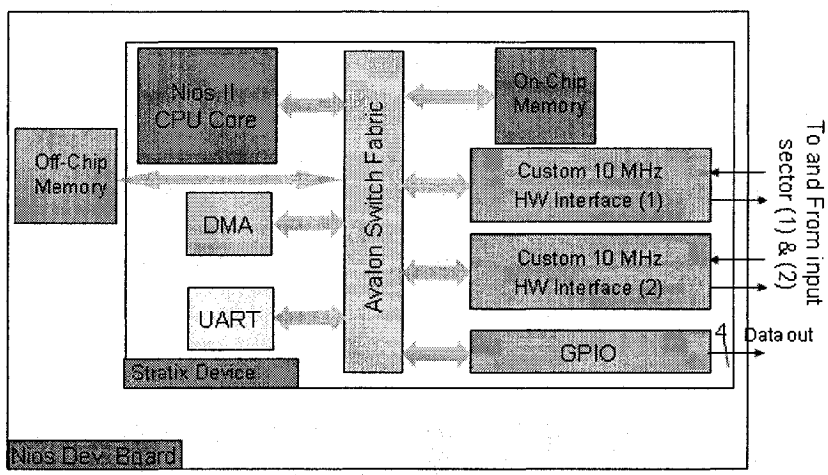


Figure 55: Global controller system-on-chip architecture

The functions, pointed out in Figure 46, were implemented in C with some custom Nios II functions and device drivers listed in Table 20 and Table 21. The architecture of our system-on-chip, used to execute the generated binary code, consists of a Nios II CPU core; a UART used to communicate with a PC; an on-chip memory RAM where the data set and the instruction set are saved; off-chip memory; a DMA which was not used in this implementation; and 3 custom hardware interfaces that are used to

configure the switch once the FBP computation is finished, as well as read the traffic information from the I-Sectors. Here we will list the functions coded for this implementation.

Function	Description
Main	Contains the main program, which implements the main FBP algorithm showed in Appendix 4. It calculates the service matrix from the traffic matrix, compares the outcome to the current service matrix used in the switch and obtains the next configuration of the switch (if needed) by decomposing the service matrix.
max_line_col	Finds the maximum among lines and columns of the traffic matrix.
find_min_mat	Locate the position (indices) of the smallest element in the matrix.
test_excess	Distribute the remaining interconnections, which consist of summing all the columns to verify that all the crossbars connections are used, if any. It calls test_rate_filling() function in an iterative manner.
test_rate_filling	Complete the rate filling algorithm.
test_compare	Compare the new service matrix to the previous one. If both are the same, do not perform any operation. If different, continue computation until finishing the configuration.
decomposition_rout	Computes the decomposition of the service matrix. This function is the last to be called before transmitting the configuration data to the hardware, by applying the interface subroutines defined in altera_avalon_cust_inter_routines.c.

Table 19: List of functions implemented to complete the FBP process

The custom transmitter and receiver, described in chapter 2, are put together in one module in order to provide a full-duplex interface between the GC and the I-Sectors. The interface to the crossbars is a bus of 4 wires, each one used to perform the configuration of each of the 4 crossbars. In order to integrate our custom 10 MHz interfaces within a Nios II-based system-on-chip, we had to describe the register file defining the interface. The register file is required by SOPC builder to enable the

communication between Nios II processor and any custom added hardware module. Table 20 shows the list of the required files.

File	Description
cust_inter_avalon_interface.v	Instantiates task logic and register file, and provides an Avalon slave interface. This file contains the top-level module.
cust_inter_register_file.v	Contains logic for reading and writing 10 Mbps custom interface registers.
cust_inter_task_logic.v	Contains the core of the 10 Mbps custom interface functionality, which is a combination of the transmitter and the receiver explained in the previous section.

Table 20: Hardware files of the custom 10 MHz interfaces

Table 20 shows the list of software files needed so Nios II can communicate to the registers associated with the custom 10MHz interface.

File	Description
Altera_avalon_cust_inter_routines.h	Declares function prototypes for accessing the 10 Mbps custom interface.
Altera_avalon_cust_inter_routines.c	Defines functions for accessing the 10 Mbps custom interface.
Altera_avalon_cust_inter_regs.h	Defines macros to access registers in the 10 Mbps custom interface component.

Table 21: Software files of the custom 10 MHz interfaces

4.5 Comparison of the Two Designs of FBP Computation

Comparing the performances; the hardware method first implemented for our 8x8 demonstrator required 18 cycles at 10 MHz clock in order to perform the computation, which corresponds to 1.79 micro-seconds. Using the on-chip timer in the SOPC system, and the "gettime( )" function provided by the HAL library, the time required to perform the calculation using the mixed hardware/software design was equal to 1.5 ms. The gap in the execution time is expected; here are some of the reasons:

- In the hardware implementation there is no use of a common bus; while in the mixed design the Avalon bus introduces a delay especially because there is a Turn-around overhead.
- All the operations are in parallel (addition, normalization…) using custom DSP blocks, while in the software implementation it is all sequential.
- In the software implementation it is required to read/write from memory since the number of variables is bigger than the number of registers available in the Nios II.

5 Chapter 5: Traffic Generator/Analyzer: Architecture and Implementation

Our main objective in this chapter is to present our novel FPGA-based traffic generator. It is built around a dedicated soft-processor within a configurable device (Stratix© from Altera). The traffic consists of fixed-size packets at a bit rate of 50 Mbps and two different patterns of network traffic in order to emulate the non-stationarity and bursty nature of real network traffic. The design makes use of the features offered by the Nios II board, Stratix Edition, chosen as development platform, and the advantages of System on a Programmable Chip (SOPC) builder provided by Altera. This chapter is based in parts on [Abdo 2005].

5.1 Introduction and Background

A traffic generator could be used as a module in a test-bed facility for performance benchmarking of layer 2 (e.g. switch) or layer 3 (e.g. router) switching components. It is used as an alternative solution to recording real traffic, which is an expensive task since it requires extensive storage. Many traffic generators have been implemented in software with different stochastic distributions, but these generators are slow and are not suitable for hardware testing.

We will now describe two other generators developed, and show the advantages and disadvantages with respect to the one we developed. In [Tagami 2002], an OC- 48c traffic tester is presented that is capable of generating and analyzing long-range dependent traffic. Here we are interested only in the generator part. The development platform is a custom PCB equipped with an Altera FLEX10K250E-1, SONET framer

within an optical transceiver, and a 64 M-byte off-chip memory. The time series, which determines whether packets are generated or not in the successive time slots, are computer-generated and pre-loaded on the memory via a PCI interface to the main motherboard. Their packets are of IP format only. Our architecture has the advantage of using a soft-processor to produce the time series, as well as having the capability to interrupt the current transmission and perform a different one with new parameters. The custom user interface is another advantage. A similar module available in the market today is SmartBits from Spirent Communications [Spirent 2005]. It can be used to analyze networks at various scales, from a single module to a LAN. Being a commercial product, we could not investigate the internal architecture. However, from the datasheets available on the Spirent communications website, we could understand the capability of this product. First, we would like to note that it is widely used in telecom companies. It provides a powerful interface to choose the parameters of the generated data, and the type of test data of interest. There is no indication of self-similar traffic generation; however, it has tremendous features capable of testing applications for xDSL, cable modem, IPQoS, VoIP, MPLS, IP Multicast, TCP/IP, IPv6, MPLS, routing, SANs, and VPNs.

5.2 Traffic Models Implemented

It is well known from high time-resolution measurement studies in the 1990s that self-similarity is present in local area network (LAN, e.g. Ethernet networks) and wide area network traffic (WAN, e.g. www) [Stallings 2002] [Beran 1992]. Self similarity means that the process distribution has fractal-like behavior, so it looks roughly the same on any time scale. Many statistical models (e.g. Brownian Motion Process, Fractional

Brownian Motion Process and Fractional Gaussian noise) were proposed to generate an artificial traffic trace that poses self-similarity. Another method to obtain self similar traffic consists of aggregating many On/Off traffic sources with heavy-tailed distributed (e.g. Pareto distributed) On and Off times.

The probability density function of the Pareto distribution:

$$f(x) = ab^a / x^{a+1}$$

And the cumulative distribution function:

$$F(x) = 1 - (b / x)^a$$

Where "b" is the minimum value of x, and "a" is the shape parameter.

The traffic below, Figure 56, is an aggregation of six Pareto On/Off sources implemented following the same model as in [Paredes 2004] and [Taebi 2004]. The subroutine was coded in C, downloaded to the Nios processor, and the generated bit stream, which indicates whether a packet should be generated, was copied to a local computer via a serial port connection. Only a window of results is shown for visualization purposes.

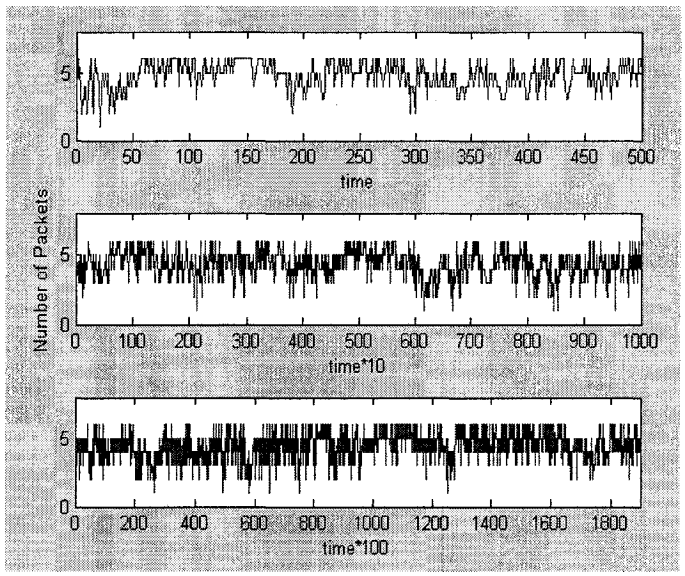


Figure 56: Generated self similar traffic

Another mathematical model widely used to emulate network traffic is Markov modulated process; this is a doubly stochastic process, it includes a "top level" process and a "bottom level" process [Chu 1995]. As illustrated in Figure 57, the top level could include many states; our design implements only 2 phases. The duration of each phase is controlled by a random variable with exponential distribution.

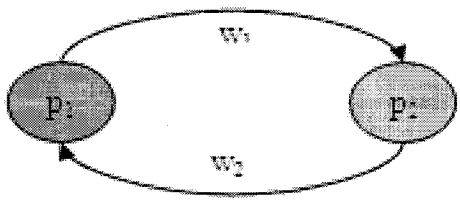


Figure 57: Two states MMBP

The occurrence of a packet in the bottom level is a Bernoulli process. In Figure 57, we show the state diagram of Markov Modulated Bernoulli Process, for each phase the probability of packet occurrence is different, and it depend on the value of chosen success probability p.

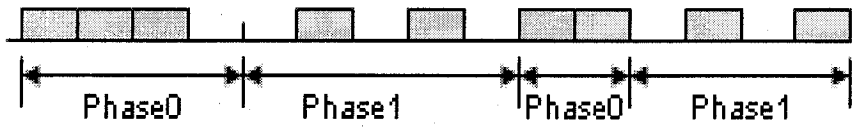


Figure 58: Two state modulated cell arrival [Chu 1995]

The occurrence of a packet is as follow:

$$f_{bernoulli} = \begin{cases} p \\ 1-p \end{cases}$$ At a given time slot, a packet is generated with probability p.

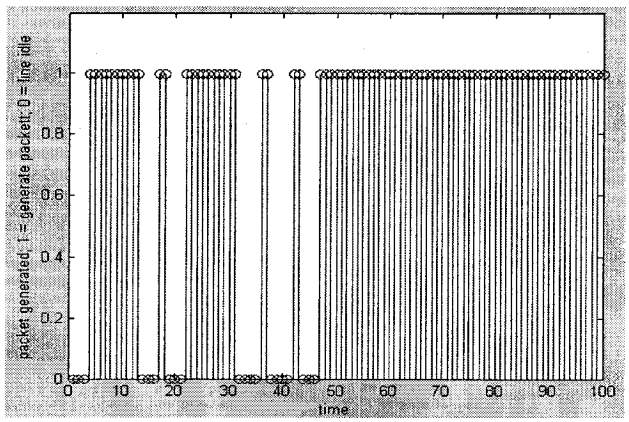


Figure 59: Generated two state MMBP

For the MMBP case, at a given time slot, there is a probability 'p' that a packet will be generated and hence, a probability 1-p that a packet will not occur. The parameter p is, therefore, the load offered by the source. Figure 59 shows a Markov Modulated Bernoulli Process of two states with 0.6 probability of generating a packet in the first

state, 0.4 in the second one, six as the mean of the first state top level process and four of the second state top level process.

5.3 Hardware/Software Implementation

When we decided to implement the traffic generator with different distributions, the intention was to develop everything in Hardware. The first task which we needed was the implementation of an integer random number generator. Three solutions were attempted in order to complete this task. First solution we tried was to use $random function from Verilog 2001. It worked in simulation but was not synthesizable on the Altera FPGA's. The second method was to use the integer division in VHDL, which worked in simulation and it did synthesize, however it occupied 5196 LEs of our Stratix chip (half of our chip). Finally, we tested the VHDL math package. However, when we included it with Quartus II, it did not synthesis. So, the decision was made to use a co-design and co-implementation of hardware and software by integrating a soft-processor with our custom packet generator HDL modules. Altera' development kit provided us with all the needed tools.

The tasks, which perform the calculations as well the communications with the computer and the hardware blocks, are programmed in C and compiled/debugged using the Nios II IDE. They run on top of the MicroC/OS-II real-time operating system (RTOS) [Labrosse 2001], which provides a powerful multi-threaded environment, as well as efficient inter-task communication through message passing. The TCP/IP communication is done using the popular small foot-print Lightweight IP TCP/IP stack [Dunkels 1998], which is suited for embedded systems. The binary codes generated from the custom

software codes are interpreted using the soft-core 32-bit reduced instruction set computer (RISC) Nios II processor. All hardware blocks (custom transmitter, custom receiver, time-stamping block, random packet generator, statistics counter, etc.) were modeled using VHDL or Verilog hardware description language at the Register Transfer Logic (RTL) level of abstraction. The compilation and the simulation were done using Quartus II v4.1, as well as the optimization in order to best fit the design in the Stratix chip. The on-chip debugging of the hardware blocks was performed using SignalTap II Logic Analyzer from Altera.

As part of the software modeling, we developed the following modules: LFSR, parallel to serial converter and packet generator modules in C. The idea was to verify the functionality and the partitioning of our design before tunnel in the development of device drivers and hardware modules.

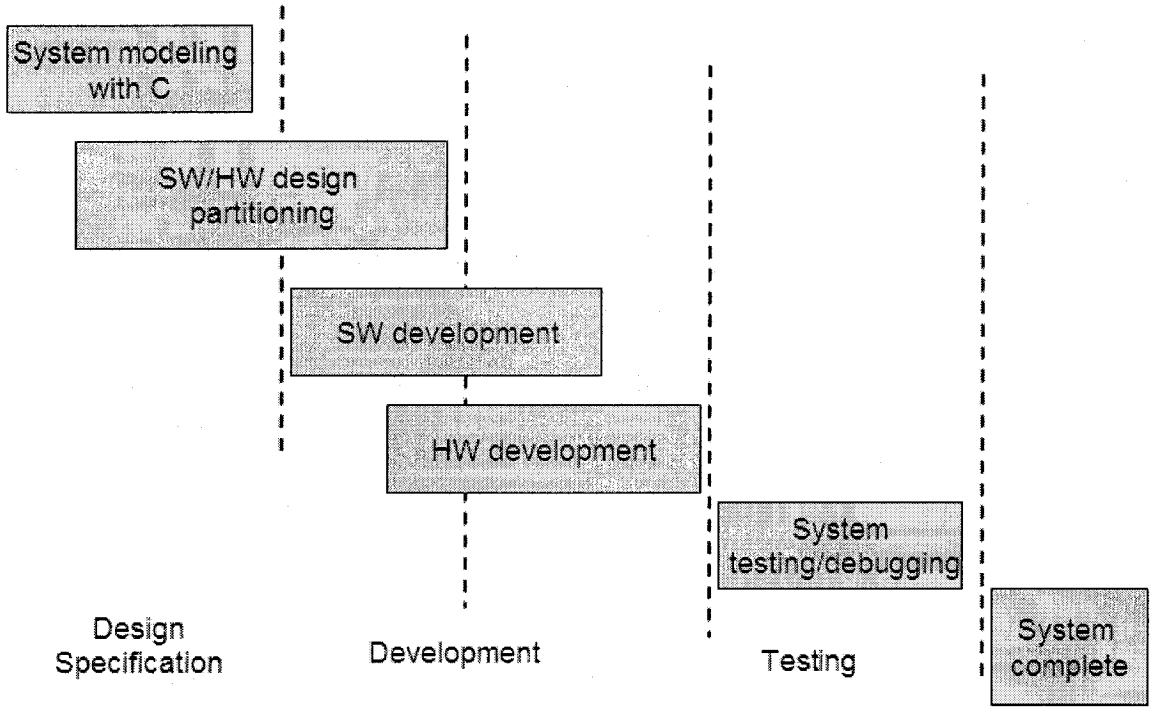


Figure 60: Hardware/Software design flow [Pellerin 2005]

The architecture, shown in Figure 61, consists of an embedded Nios II RISC processor and a programmable hardware module implemented on a Stratix FPGA chip. The communication between on-chip modules is realized by means of a shared data bus (Avalon Bus) [Avalon 2004] embedded in the FPGA. The integration of a processor within the FPGA device allows high-performance since all the processing and data-sharing are done on chip, which will minimize inter-chip communication delay. The Avalon bus is a simple bus architecture designed by Altera to connect on-chip processor and peripherals (UART, RAM, etc) together to form a SOPC [SOPC 2004].

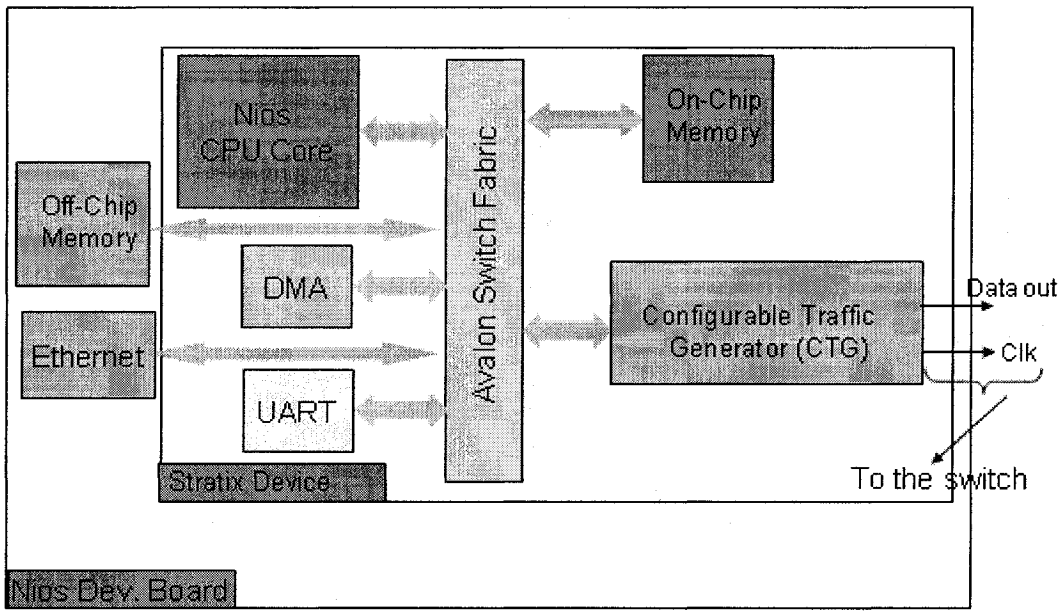


Figure 61: Traffic Generator high level architecture

The 32-bit RISC processor is used to perform procedural calculations (e.g. calculation of Pareto and Exponential random variables); provide communication with the GUI and send control information to the Configurable Traffic Generator (CTG), in order to determine which data distribution should be used and whether a packet should be generated or not at a certain time slot. Our design takes advantage of the SOPC Builder, by composing bus-based systems out of available "library components" such as CPUs, memory interfaces and peripherals. In order to integrate our custom packet generator within a Nios II-based system-on-chip, we had to define the register files and software driver's functions [QuatrusII 2005]. Table 22 and Table 23 present these files with a description of their purposes.

File	Description
tf_gen_avalon_interface.v	Instantiates task logic and register file, and provides an Avalon slave interface. This file contains the top-level module.
tf_gen_register_file.v	Contains logic for writing packet generator registers.
tf_gen_task_logic.v	Contains the core of the packet generator functionality.

Table 22: Traffic generator hardware files

File	Description
Altera_avalon_tf_gen_routines.h	Declares function prototypes for accessing the packet generator.
Altera_avalon_tf_gen_routines.c	Defines functions for accessing the packet generator.
Altera_avalon_tf_gen_regs.h	Defines macros to access registers in the packet generator component.

Table 23: Traffic generator software files

In order to generate random number on the Nios II processor, we coded the following the subroutine which implement the multiplicative congruential method.

```
      SUBROUTINE PRAND (SEED)
C Program to generate pseudo-random
numbers
      INTEGER SEED
      SEED = SEED * 65539
C Reduce results modulo 2^31
      IF (SEED) 5, 6, 6
        SEED = SEED + 2147483647 + 1
        RETURN
      END
```

In this program, the modulo chosen is 2^{31} and a = 65539. The value of x_0 is not given, but it could carry any value that respects the following conditions:

1. x_O and M have to be relatively prime (i.e., x_O must be odd, since all of M's factors here are 2)
2. a/3 or 5 modulo 8

The value we chose for x_O is three which meets the above requirements.

Depending on the 2 bits representing the "traffic_type" value sent at the beginning of each stream vector, the state machine will either enter the s_ssx part or the s_m part. For the first one, at each of the state showed in Figure 62, we compare the decision bit to the previous one from the same source if:

- We had 0 and now 1, generate a new header.
- We had 1 and now 1, keep the same header
- We had 1 and now 0, no packet will be generated.
- We had 0 and now 0, no packet will be generated

The s_m state will only read the bit, if it is 0 nothing will be generated, if it is one it will generate a packet. The destination header is initialized to "000" and is increased by 1 at each time slot. If rst is asserted, we return to the initial state. Otherwise, we jump to the next source-state.

As mentioned above, the 2 least significant bits from the streaming vectors sent through the Avalon bus determine the traffic to be generated. In our case, we only made use of one bit. However, the second bit was kept in order to allow for scalability by permitting 4 different combinations. As well, because we are generating self similar

pattern using 6 sources, we maintained the number of bits as a multiple of five in order to minimize the complexity of the "Traffic and Destination Controller"

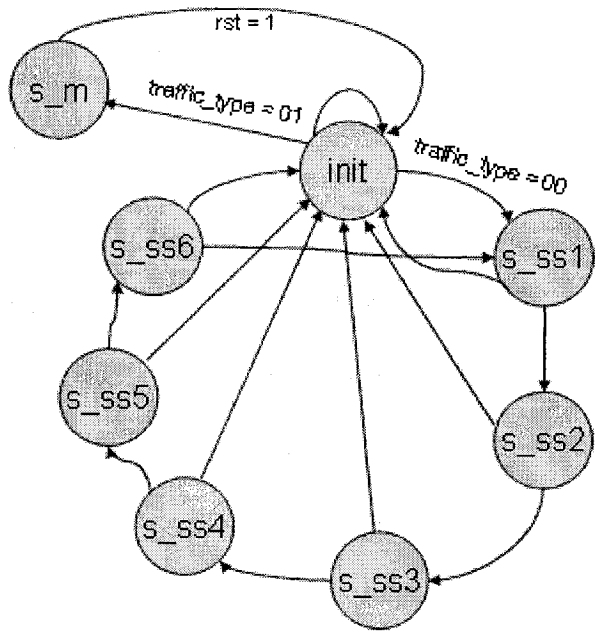


Figure 62: State machine of the "Traffic and Destination Controller"

Each vector will contain the decision for 5 time slots from each source. The "Interface Controller" will generate an interrupt if the length of the FIFO is less than 3, so, even if the traffic generator was communicating with the GUI, it will stop sending/receiving data and perform calculation to fill the FIFO with appropriate data. A more detailed look to the interface signals between the CTG and the Avalon bus are presented in Figure 64. The hardware part is illustrated in Figure 63; it operates around the "Traffic and Destination Controller", which state machine was shown in Figure 62. The LFSR is used to generate a random payload, where the data are loaded into a 16 bits register which will send the data to the parallel to serial converter when a packet need to

be generated. The traffic type is determined by the "bus" of 2 bits set by the "Interface Controller and Traffic Choice" module.

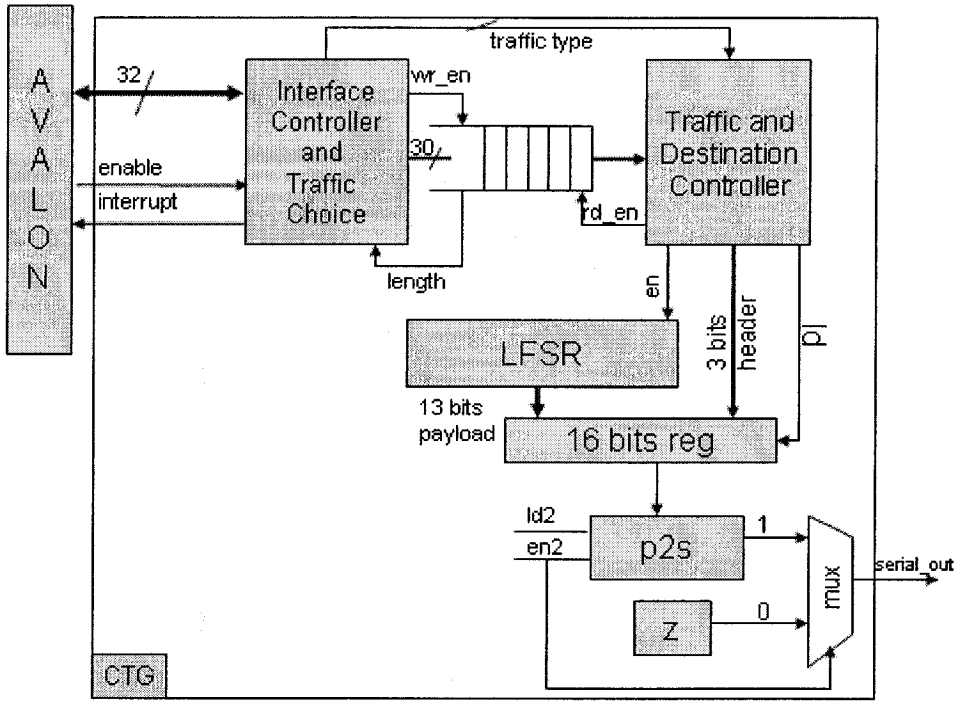


Figure 63: CTG hardware blocks

```
module tf_gen_avalon_interface
(       clk,
        resetn,
        avalon_chip_select,
        address,
        write,
        write_data,
        read,
        read_data,
        serial_out
);
```

```
module tf_gen_register_file
(       //Avalon Signals
        clk,
        resetn,
        chip_select,
        address,
        write,
        write_data,
        read,
        read_data,

        //Traffic Generator Output Signals
        tf_gen_vector,
        tf_gen_enable
);
```

Figure 64: Interface signals of the custom developed hardware

The simulation of the CTG using Quartus II was done with a 50 MHz clock (50 Mbps data rate). Figure 65 presents a window of the simulation.

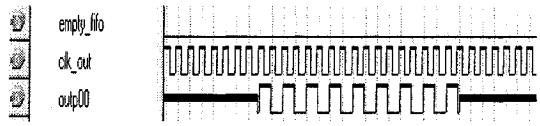


Figure 65: Idle (z) + packet (1010101010101010) + Idle (z)

5.4 User Interface

In order to provide a friendly means to manage the traffic generator and to capture results from the traffic analyzer (packet delay in this chapter), a custom graphical user interface (GUI) was designed and developed using Java. We used the open source "BlueJ" Integrated Java Environment [BlueJ] to edit, compile and debug our codes. We chose Java because of the following features: portability, reliability and security. Our multi-threaded GUI communicates with the Nios processor via RS-232. The connection is made over Universal Asynchronous Receiver Transmitter (UART), a single thread, which is always monitoring the serial port, takes care of updating the statistics record. We developed 2 versions of GUI, a basic one showed in Figure 66 and more advance one with many new features in Figure 67 and Figure 68.

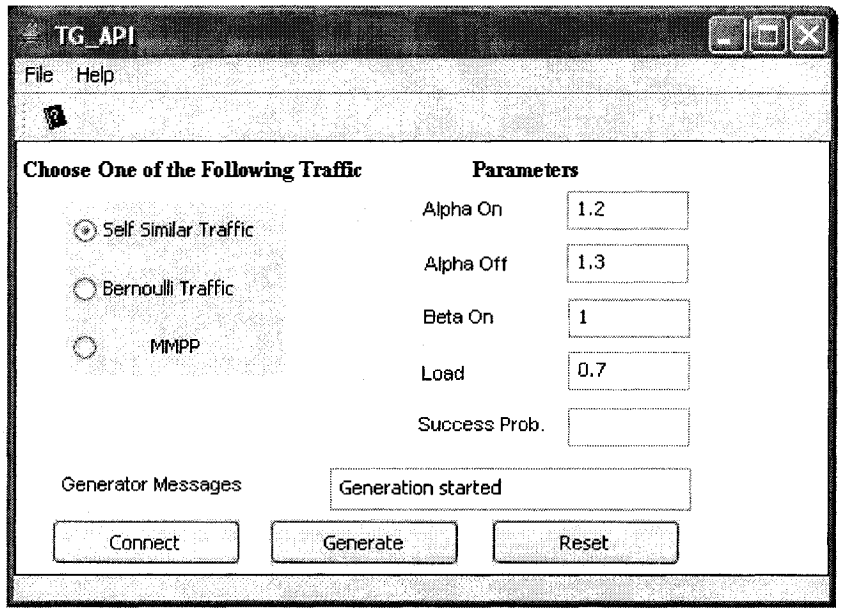


Figure 66: Snapshot of Version 1 - Traffic Generator GUI

The first version was based on the Abstract Window Toolkit and it was developed using JBuilder from Borland. However due to the tremendous capabilities of swing over AWT in term of visualization and features, we developed a second version, the traffic generator/analyzer GUI is more than 1000 lines of Java codes. It boasts many features such as the possible to go from the traffic generator frame, presented in Figure 67, to the traffic analyzer frame, shown in Figure 68, by clicking on the "switch" button; the ability to choose serial port (COM1 or COM2) from a drop down menu, the user guide is accessed from the "Help" menu, among other features.

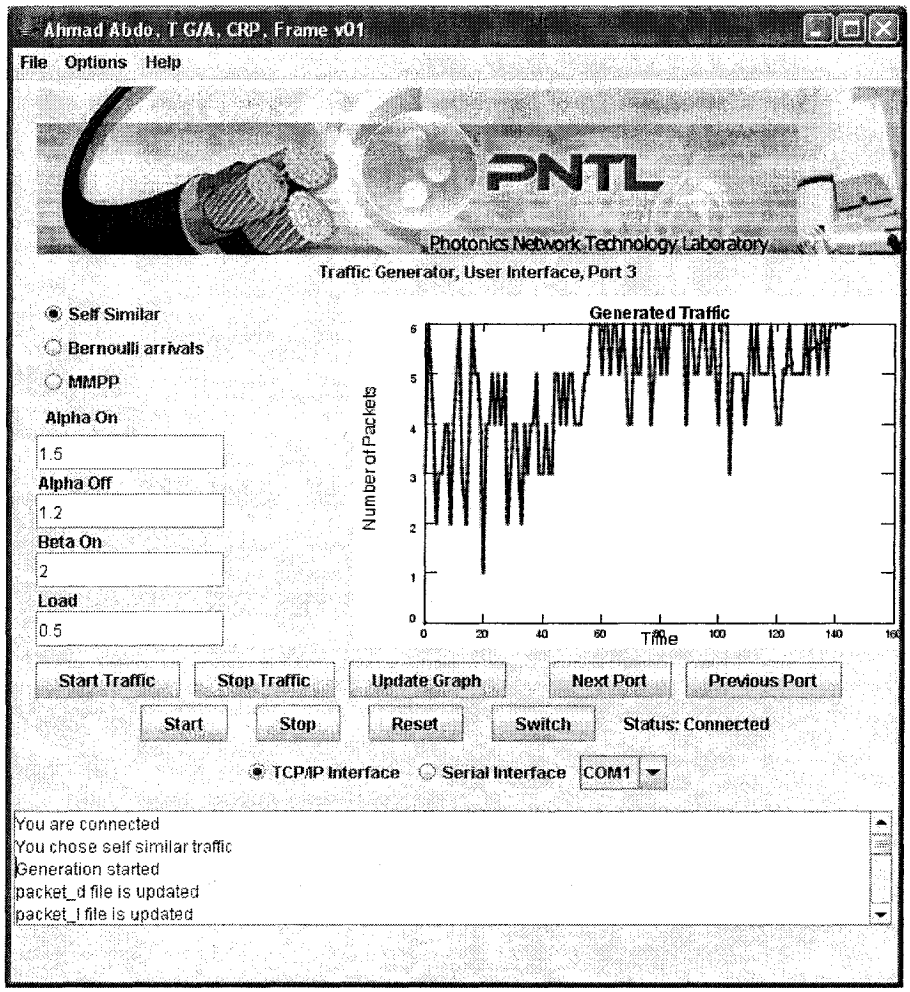


Figure 67: Snapshot of Version 2 - Traffic Generator GUI

The 2-D graphics were initiated as JPLot2D objects; they were imported from the Chapman package [Chapman 2002]. The open, close and read/write from the serial port required including the javax.comm package. All the buttons, drop list, radio buttons, and text fields are part of the swing library.

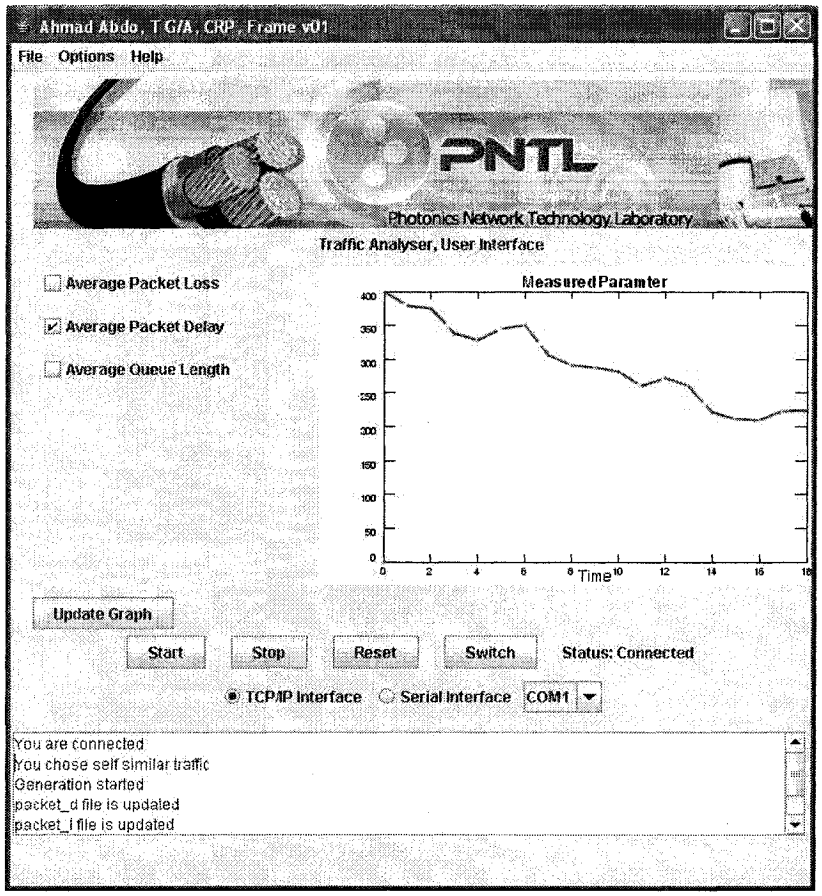


Figure 68: Snapshot of Version 1- Traffic Analyzer GUI

Panelv01 is the main panel, it is declared as a public class and as an extension of JPanel. In the traffic analyzer panel, we initiated 3 arrays for each performance parameter of interest (packet loss, packet delay and queue length). The file_pa_l, tg_traffic, file_qu_l and file_pa_d are four threads, they are responsible to listen to the specific TCP ports and copy any data in the appropriate file system. That information gathered by each thread is used to update the graphs when it is requested by the user. The board emulator is a simple TCP server; it was developed for testing purposes.

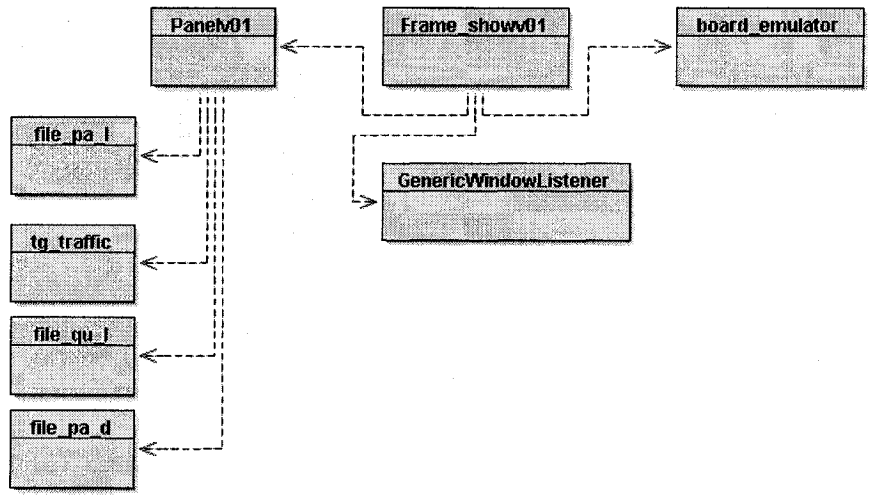


Figure 69: Classes relation traffic generator/analyzer GUI

5.5 Testing the Switch

In order to validate all the developed modules in hardware, we performed a basic functional and timing testing on the FPGA for everything presented in this thesis. For example, for the Output Sector, we sent one packet to different inputs of the Output Sector and make sure we are receiving it at the corresponding output. We used the built UART interface, presented in chapter 3, to send the received information to the PC. The development of the traffic analyzer integrated with the traffic generator on the same chip is underway.

The primary test parameters of interest are:

- Delay versus time
- Delay versus traffic load
- Queue length versus time
- Queue length versus load
- Delay versus inter-configuration period

To implement the traffic analyzer, we investigated the use of ML310 [Xilinx 2005] development board from Xilinx because it is featured with an on-chip PowerPC hard-processor working at 400 MHz and has plenty of room on the Virtex-II Pro chip in order to implement custom hardware logic, as well as porting a QNX Neutrino v6.3 [QNX 2005] to it. However, due to the time and the complexity of porting a real time operating system to a new development board, we decided to use MicroC OS II and Nios II. It is worth noting that numerous advantages would make the Virtex-QNX option more promising:

- PowerPC is much faster than Nios II.
- The availability of 256 MB DDR DIMM off-chip memories on the ML310 board.
- QNX enables the user to provide fairness to the various tasks running by time-slicing the processor, while MicroC requires assigning a specified priority to each of the running tasks.

In this thesis, we implemented the time stamping module presented in Figure 70. The receiver presented in chapter 2 will convert the serial packet into parallel one.

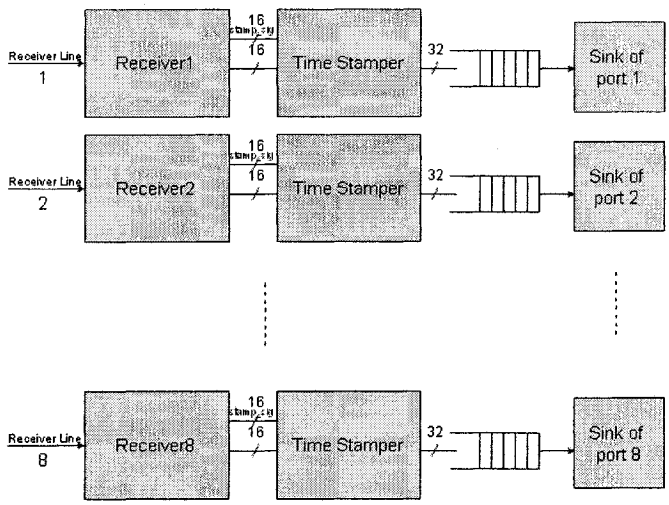


Figure 70: Architecture of the analyzer time-stamping module

The "Time Stamper" module will add a "16 bits" label which determines the current time slot. The new load will enter the FIFO, the presence of the FIFO is not required in our case but we kept it in case the sink clock is different that the "Time Stamper" one. At the sink, a differentiator will extract the variation of time between the received time and the generation time. Figure 71 illustrates a part of the time-stamping process. A 16-bit packet comes in serial format at the input ("in_ser") with a start bit of '1'; the receiver changes its format to parallel and then stamps it with the current time slot (in this case, it is 6).

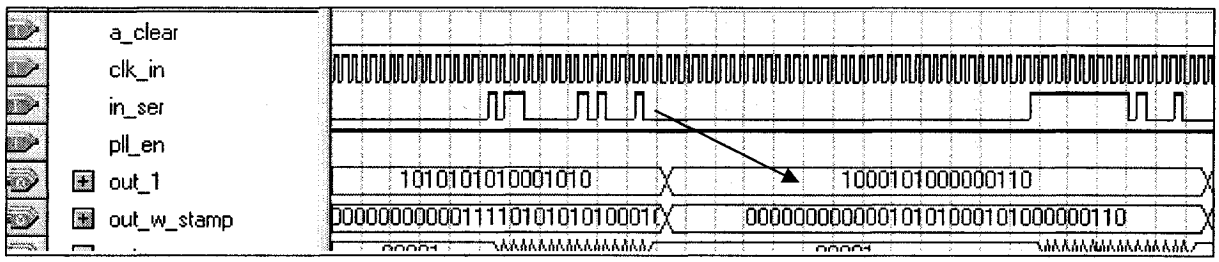


Figure 71: Simulation of the time-stamping module

5.6 Future Packet Format

The extended packet format consists of 512 bits divided as follows: 17 for destination, three unassigned bits that are reserved for special use such as providing quality of service and class of service (which will permit eight classes of services as in Multi-Protocol Label Switching networks) and finally the 492 bits payload, which is large enough to accommodate different upper layer traffic types (e.g. TCP).

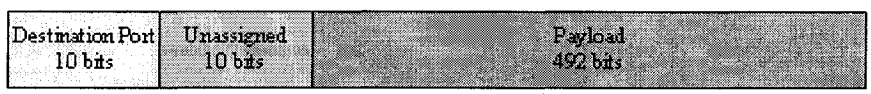


Figure 72: Extended packet format

6 Conclusions and Outlook

In this chapter will be formed by two sections: conclusions and future work. First section will highlight the challenges we faced and the advantages of our various design and implementations. In the second section, we will identify several directions for future work. We would like to note that at the time of the submission, the integration of the switch was not complete. We performed Quartus II compilation of the various modules all together in one schematic and it was successful, however, software testing of the functionality and the timing requires a Modelsim test-bed. And the hardware testing need further work such as completing the traffic analyzer.

6.1 Conclusions

This thesis includes contributions in many areas: optical and electronics (mainly FPGA). As well, various hardware (VHDL and Verilog) and software (Java, C and Matlab scripts) programming languages were used. The conclusions advantages of our packet switch are:

From this thesis perspective:

- The implementation of the O-Sector, and mainly the complexity of the memory manager, showed the importance of performing the switching in two stages: one at the I-Sector per O-Sector and the second at the O-Sector per output port.

- As for the custom traffic generator, in which all the inter-module communication is done on-chip which increases system performance compared to inter-chip communication [Stiliadis 1996]. Being an FPGA-based solution, this innovative

generator has a relative low cost with respect to other commercial ones. Scalability is guaranteed by the ability to include more soft-core processors to the design in order to help with hardware blocks monitoring and communication with the GUI.

General perspective:

- This Clos-like switch architecture allows for a partition of labor and capitalizes on the strengths of the two technologies:
  - electronics for switching, buffering and packet processing
  - photonic core for transport

- Reconfiguration does not need to be on a per-timeslot basis (as in other switching architectures and scheduling methods)

6.2 Future Work

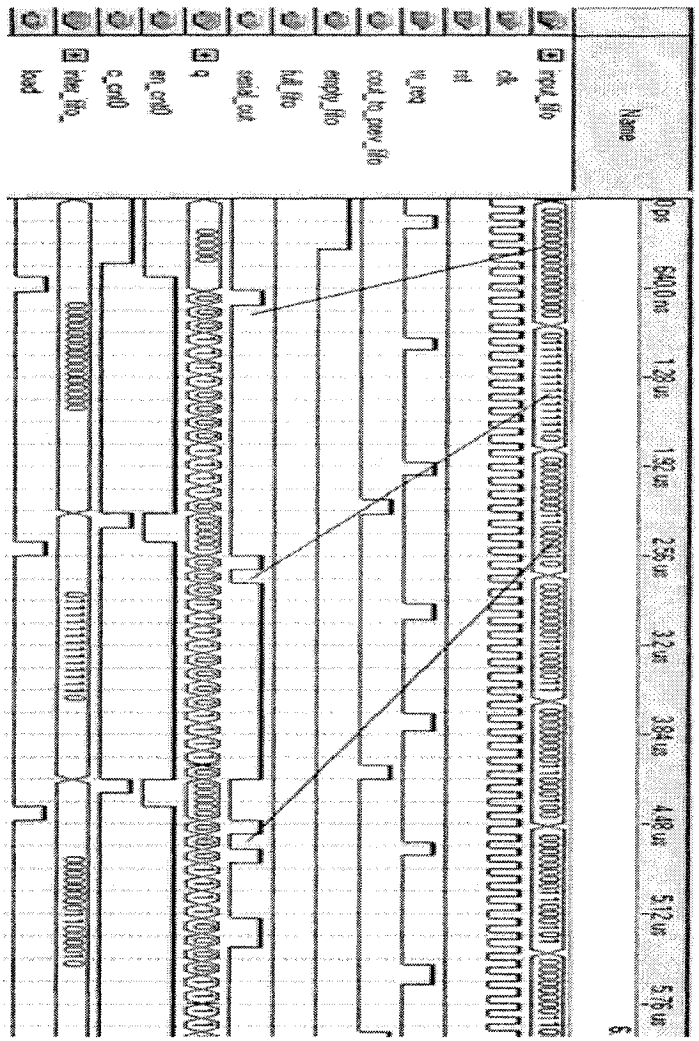
The modules developed and presented in this thesis are only a portion of the hardware demonstrator. Some of them are generic and scalable (e.g. the custom transmitter and receiver) so they can be used in any FPGA work that requires communication between various entities. The implementation of phase v2 of the demonstrator has already started. A complete characterization of the SFP is underway in collaboration with Hazem Awad at BTI Photonics. Moreover, the SFP Hardware driver for the Stratix GX has been designed and is partially implemented.

We are currently working on the integration of the functionality of our traffic generator/analyzer, whose architecture is based on the traffic generator already implemented. We added a traffic analyzer part including the time stamping module, which is used to measure the delay of packets when traversing the switch. Packet comparator is another module that is required as well in order to complete the switch characterization.

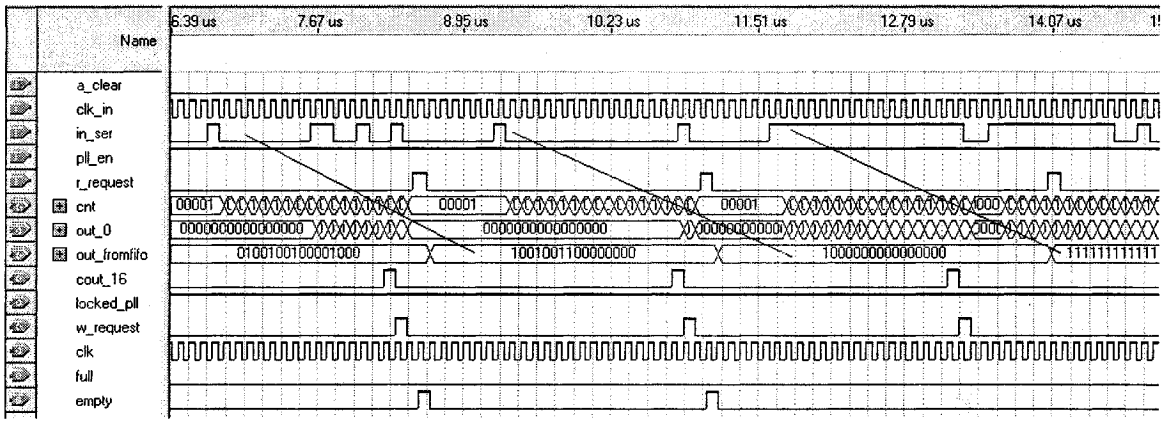
Appendix A: Modules Schematics and Simulations

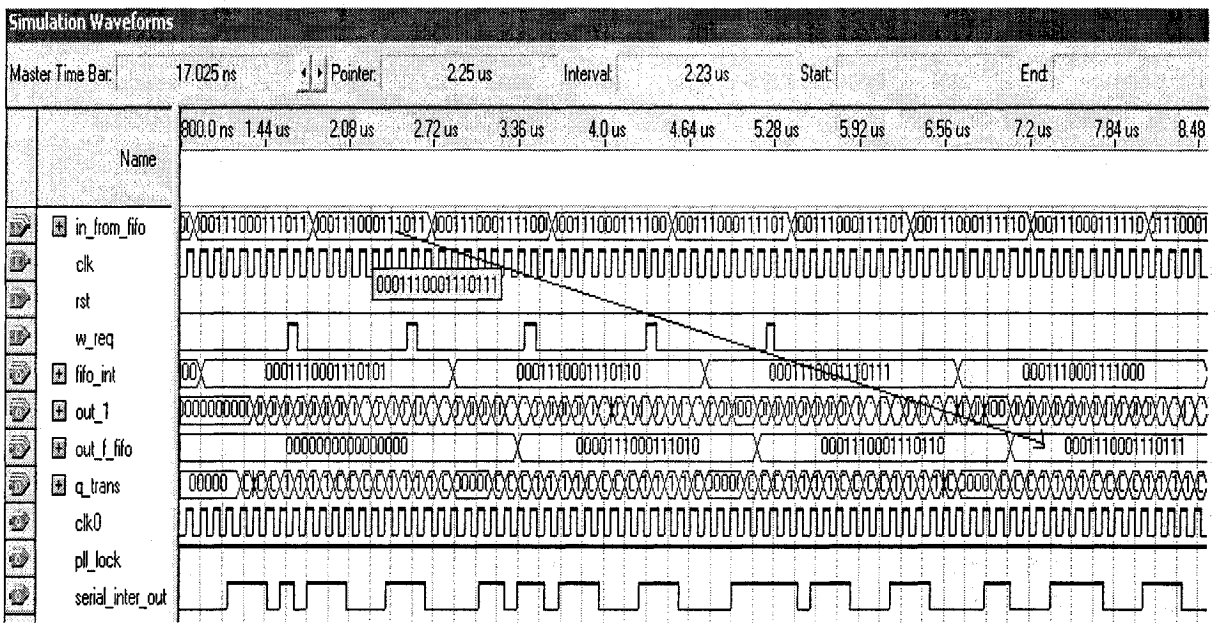
In this appendix, we will present various simulation results of the implemented modules, as well some schematics.

1 - Simulation of the 10 Mbps transmitter: the picture below illustrates the functionality of the transmitter when a packet is stored in the FIFO, pre-transmit FIFO queue in this case, then loaded to the parallel to serial converter and where shifting is enabled until the whole packet is out.

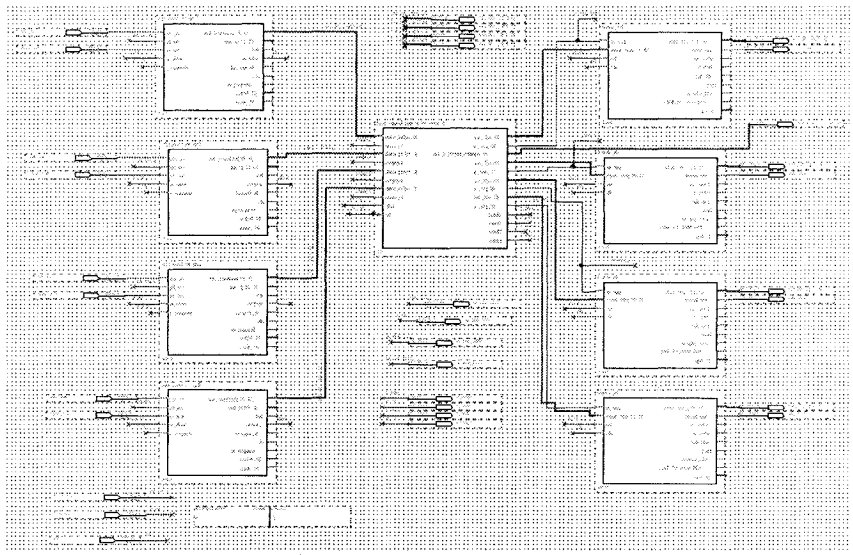


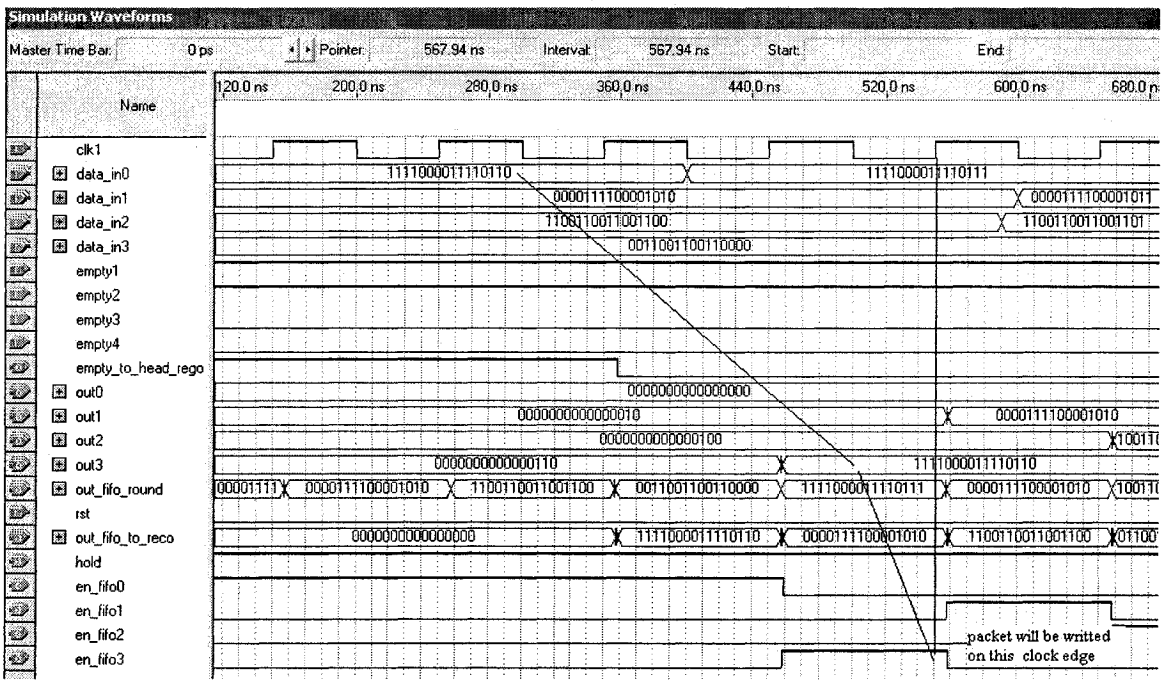
2- Simulation of the 10 Mbps receiver: the following timing diagram illustrates the functionality of the receiver. A packet, at in_ser, is coming at 10 Mbps line rate with a start bit of 1, it is sensed by the receiver controller, which will enable the shift-register. Once the packet is completely in the shift-register, the controller will enable the write signal of pre-store FIFO queue and the packet will be stored. The second picture shows the simulation of the custom transmitter-receiver, this simulation is to visually prove the functionality of the transmitter and the receiver.



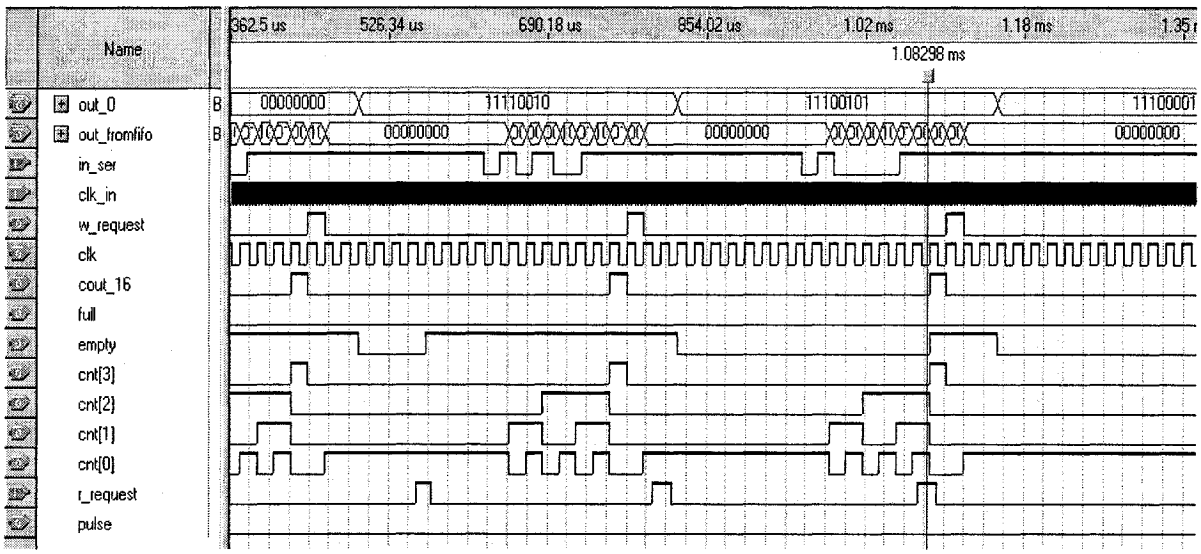


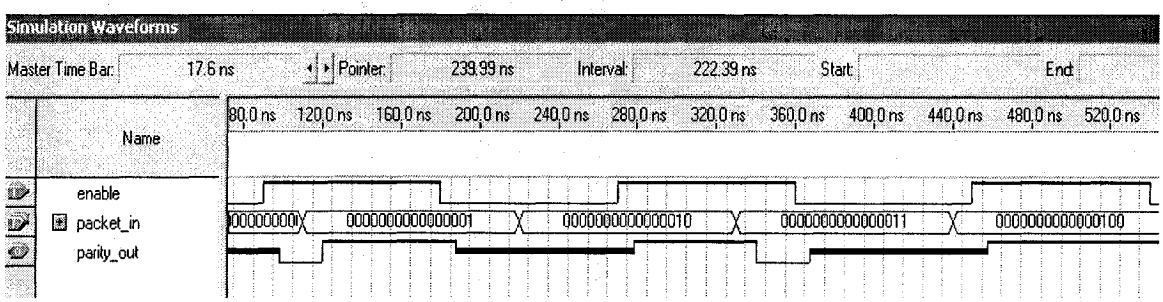
3 - Schematic of the 4x4 O-Sector and simulation of header recognition: the picture below presents the schematic of the output sector. It consists of 4 receivers, 4 transmitters and the core element. The latter includes the round robin/header recognition state machine, which take care of serving the receivers FIFOs and writing the packet in the assigned pre-store FIFO queue depending on the destination port. The second picture shows the packet coming at input port 0 is being written to the FIFO assigned to output port 3, because it has a header of "11".





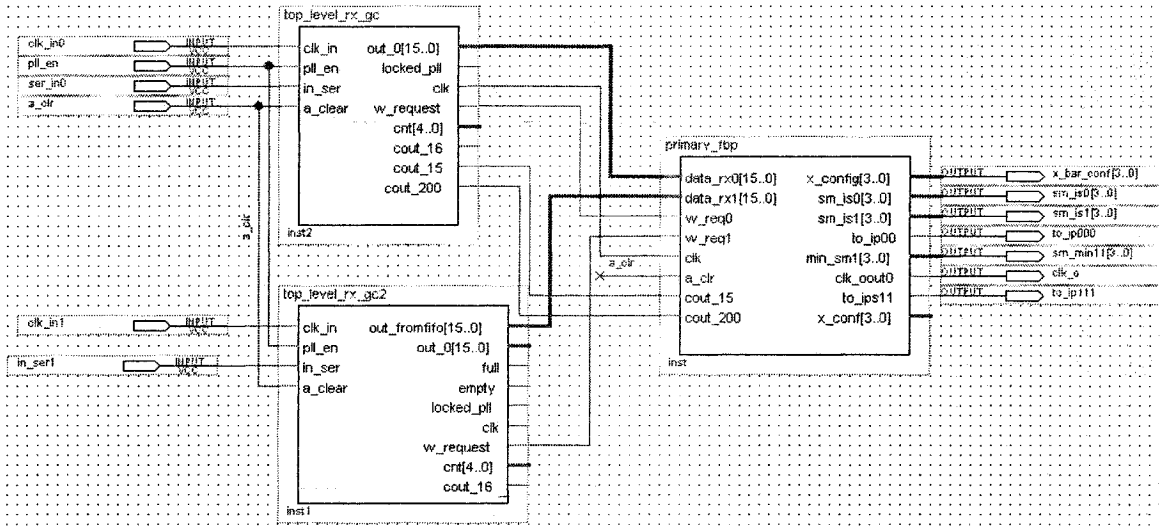
4 - Simulation of UART receiver: This simulation illustrates the functionality of the UART receiver, where 8 bits packets coming serially are sensed by the controller when the voltage line drops from 3.3 volts (after the voltage regulator from 12 to 3.3 available on the development board) to 0 volt. The second picture presents the simulation of parity checking/generating module; this module is common in UART implementations used to provide a basic error checking on the received packets.

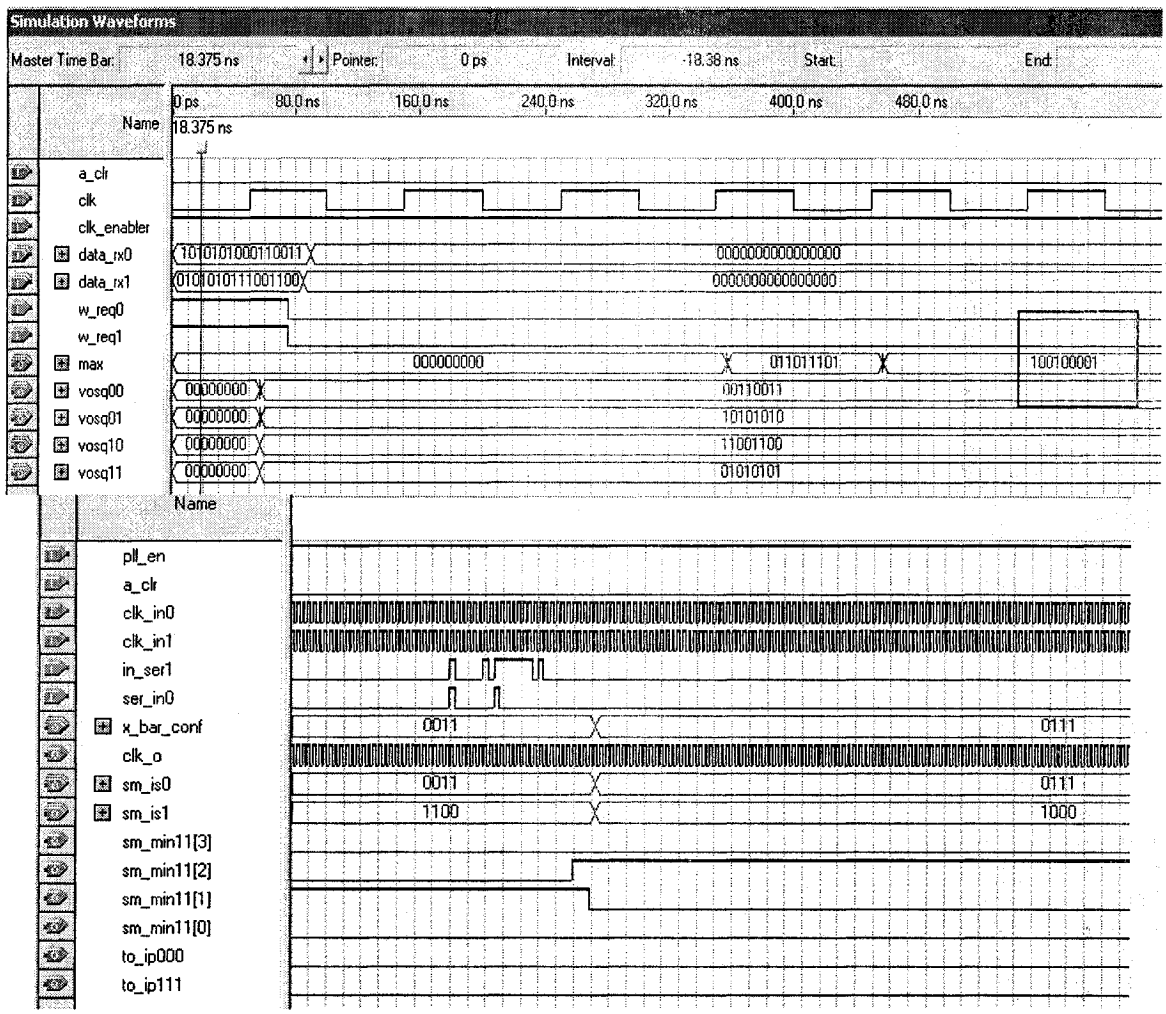




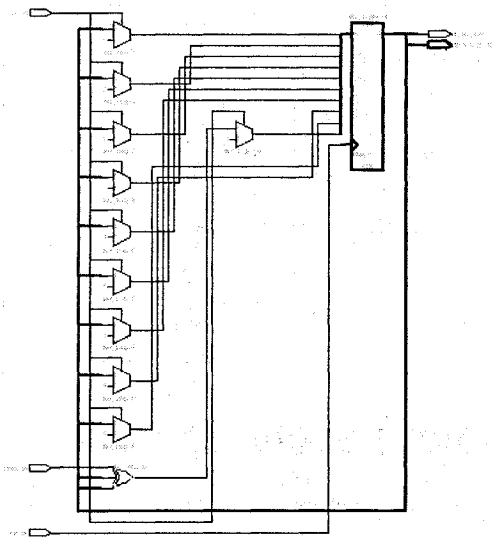
5 – FBP schematic and simulations: The first picture below presents the top level schematic of the FBP hardware implementation. It shows two receives, one per I-Sector, used to get the traffic information and the "primary_fbp" module. The latter contains the blocks presented in chapter 3. We can see at the output the following signals:

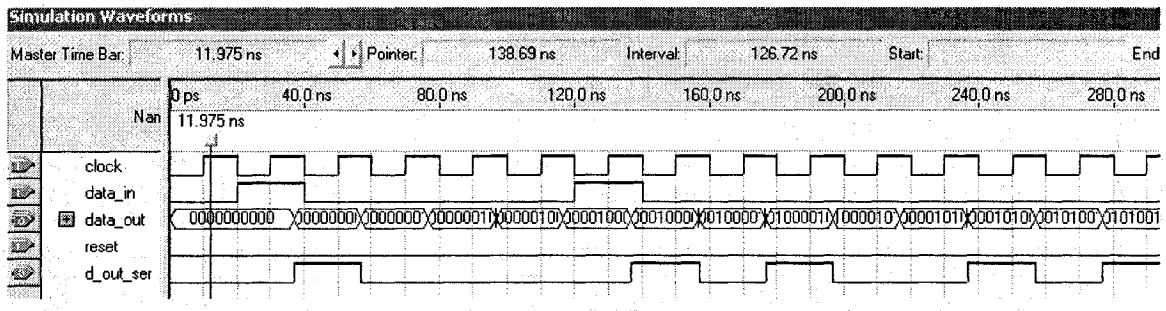
x_bar_conf[3..0] used as inputs to the crossbars control pins. The two pictures that follow show the simulation of stage1 and the generation of the x_bar_conf respectively.



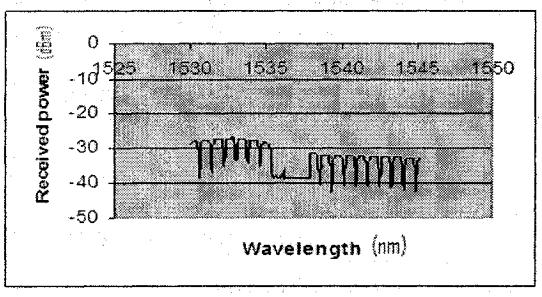


6 - LFSR schematic and simulation: The LFSR is a commonly used method of producing pseudo-random sequences. The picture below presents the RTL implementation, where we used the one-to-many configuration. Finally, the last picture illustrates a window the of LFSR simulation.



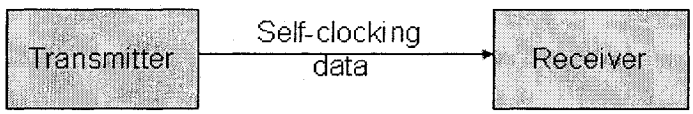


7 – Power of different wavelengths through the WSS: The figure below shows the measured power of 16 different wavelengths at the output of the WSS 5400.

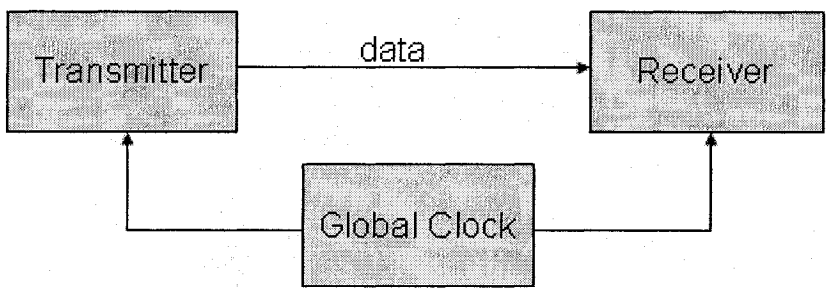


Appendix B: Various Synchronization Methods in Serial Data Transfer

1- Modulating the data in such a way that the clock signal is part of the channel code. Manchester and Differential Manchester codes are a common implementation of this kind of modulation. It has a lot advantages such as the elimination of the DC component and error detection capability; however, the design of the transmitter and receiver is relatively complex.

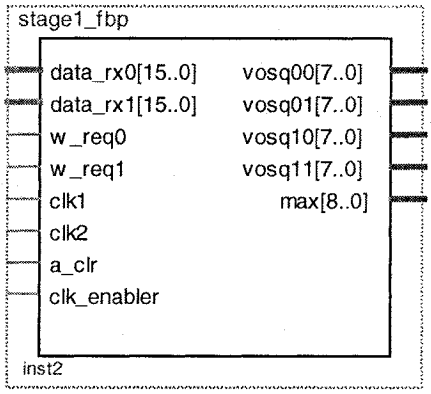


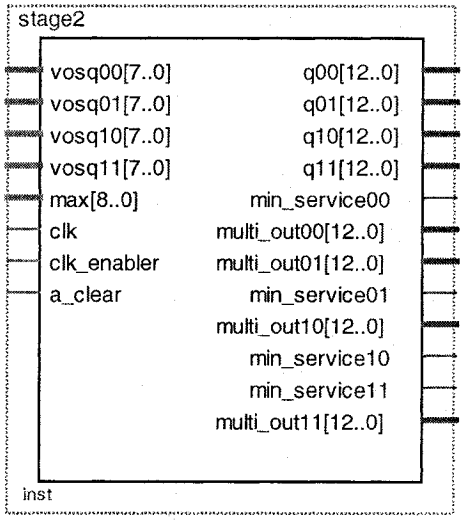
2- Using the global controller clock as a reference for the other local clock running on the input and output sectors is another synchronization possibility that is best suited for optical communication since it would not be cost-effective to send the clock in parallel on a different fibre to the receiver.

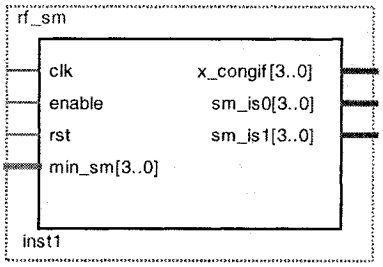


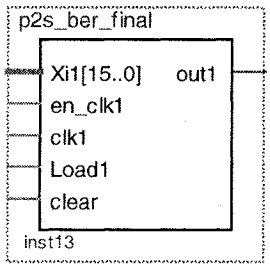
Appendix C: Schematic of Various Modules

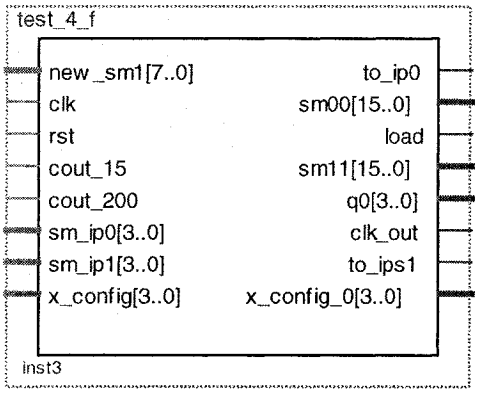
In this appendix illustrates the schematics of the blocks presented in chapter 3 for clarity purposes.

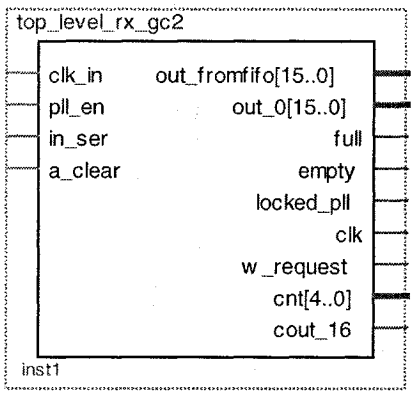







p2s_ber_final
Xi1[15..0]
en_clk1
clk1
Load1
clear
out1
inst13


test_4_f
new_sm1[7..0]
clk
rst
cout_15
cout_200
sm_ip0[3..0]
sm_ip1[3..0]
x_config[3..0]
to_ip0
sm00[15..0]
load
sm11[15..0]
q0[3..0]
clk_out
to_ips1
x_config_0[3..0]
inst3


top_level_rx_gc2
clk_in
pll_en
in_ser
a_clear
out_fromfifo[15..0]
out_0[15..0]
full
empty
locked_pll
clk
w_request
cnt[4..0]
cout_16
inst1

Appendix D: FBP Algorithm

As mentioned in chapter three, this algorithm was introduced by Sofia Paredes and it was presented in details in her PhD thesis titled: "Flexible Bandwidth Provision and Scheduling in a Packet Switch with an Optical Core" [Paredes 2005].

Main FBP Algorithm:

1. Obtain an *inter-sector traffic matrix* $\Lambda = \left[\lambda_{ij}\right]$ that represents the traffic demand per interconfiguration period.
2. Calculate the *inter-sector service rate matrix*, $S = \left[s_{ij}\right]$, $\sum_i s_{ij} = m$, $\sum_j s_{ij} = m$; from the inter-sector traffic matrix: $S = f(\Lambda)$.
3. Construct a bipartite graph G with bi-adjacency matrix $A(G) = S$. The vertices a_i correspond to the input sectors and the vertices b_j correspond to the output sectors. G is bipartite with bipartition (A, B), $|A| = |B| = l$ and $\Delta(G) = \delta(a_i) = \delta(b_j) = m$.
4. Create a set P, $P = \varnothing$, that will contain m permutation matrices sized $l \times l$.
5. Decompose G.
6. P now contains m permutation matrices such that $S = p_0 + p_1 + ... + p_{m-1}$. Use these permutations to configure the m switches in the central stage and the read operations in the input sectors.

The pseudo code of the *Rate Filling* algorithm is as follows:

1. Make $S = S_{\min}$
2. If $\sum_i s_{ij} = m$ and $\sum_j s_{ij} = m$, exit.

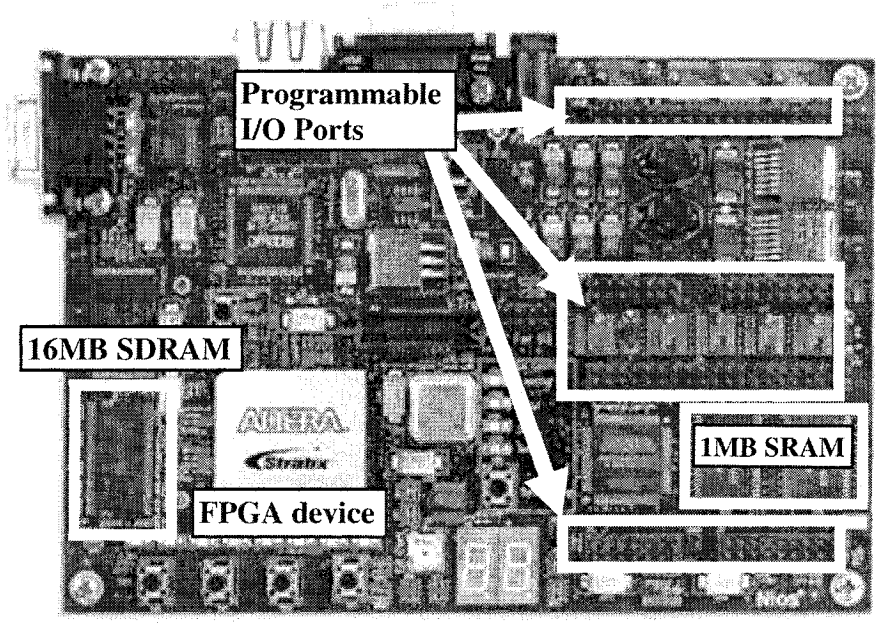
3. Calculate the excess matrix $X = \varpi S - \Lambda_e$
4. Create a list Ω with the elements of X in ascending order
5. Make ω the first element in Ω
6. Make i and j equal to the original indices of the element ω in X
7. If $\sum_i s_{ij} < m$ and $\sum_j s_{ij} < m$, make $s_{ij} \leftarrow s_{ij} + 1$
8. If ω is the last element in Ω, go to step 2
9. Make ω the next element in the ordered list Ω, go to step 6

Appendix E: Introduction to the Hardware Used

Many reasons were behind the choice of the Nios development board, Stratix Edition [Stratix 2005]; mainly:

- Its capability to integrate programmable logic and processors onto a single device, which allows the partitioning of the design into software and hardware.

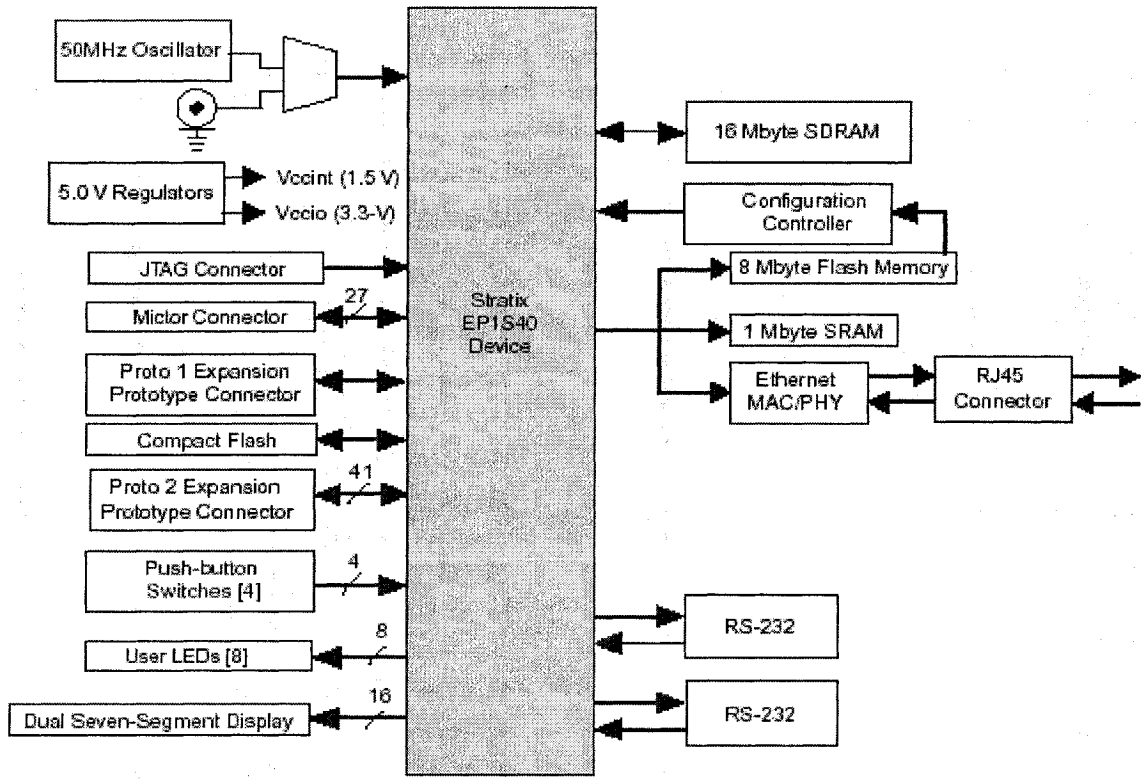
- The availability of various interfaces (2 serial ports, one RJ-45 and many programmable I/O).



Development Platform ALTERA® NIOS™ [Stratix 2005]

The development board is built around an FPGA chip, Stratix ©, which has on-chip memory of 3 Mbits worth of RAM and 48 blocks of DSP. The off-chip modules are connected through transmission lines to the FPGA with certain interface pins, which can be initialized when assigning the pins of the design, before running the place-and-route algorithm. The blocks of interest for our development in this thesis, are the following **Error! Reference source not found.**:

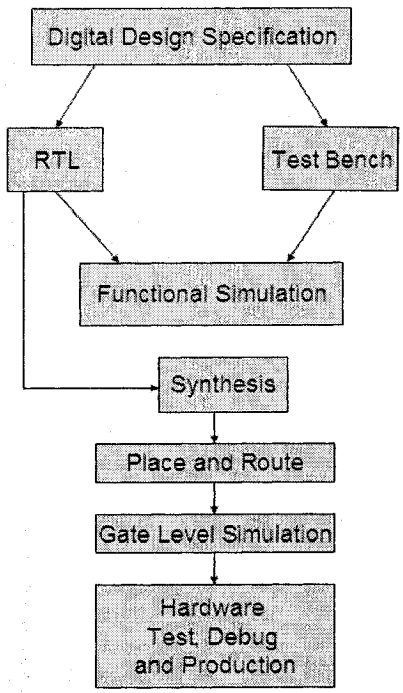
- **RS-232**: is equipped with MAX chip in order to perform the level shifting from 5 V to 12 V, we employ it to establish communication between the PC and the chip.
- **16 Mbytes SDRAM**: is needed to store packets before transmission
- **50 MHz oscillator**: to feed the digital blocks with a reference clock.
- **JTAG connector**: used to program the FPGA.
- **User Leds**: used to perform basic testing; such as showing the number of received packets.



Nios II dev. Kit, Stratix Professional Edition Diagram [Stratix 2005]

. We performed the design and implementation of hardware modules by following the standard digital design flow diagram shown in (**Error! Reference source not found.**)

[Mentor 2005]. For editing, Functional Simulation and Place and Route, we used Quartus II. For Test Bench development and Gate Level Simulation, Modelsim was used.



Standard digital design flow diagram used to develop the modules [Mentor 2005]

Bibliography

[Abdo 2005] A. Abdo, T. J. Hall, "Programmable Traffic Generator with Configurable Stochastic Distributions", Canadian Conference on Electrical and Computer Engineering (CCECE 2005), May 1-4, 2005 Saskatoon, Canada.

[Abdo1 2005] A. Abdo, V. Bishtein, P. G. Dicorato, S. A. Paredes, and T. J. Hall, "Packet Switch Using Reconfigurable Optical Central Crossbars", Conference on Lasers and Electro-optics Europe CLEO/Europe 2005, Munich, 12-17 June 2005.

[Abdo2 2005] Ahmad Abdo, V. Bistein, S. A. Clark, P. Dicorato, D. T. Lu, S. A. Paredes, S. Taebi, Trevor J. Hall, "Adaptive Packet Switch with an Optical Core", Photonics North 2004, Ottawa, Canada.

[Agelis 2005] S. Agelis and M. Jonsson, "Optoelectronic Router with a Reconfigurable Shuffle Network Based on Micro-optoelectromechanical Systems", January 2005, Vol. 4, No. 1, Journal of Optical Networking.

[Anderson 1993] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High Speed Switch Scheduling for Local Area Networks", ACM Transaction Computer System, Pages: 319–352, Nov. 1993.

[Anderson 1998] N. McKeown and T. E. Anderson, "A Quantitative Comparison of Scheduling Algorithms for Input-queued Switches", Computer Networks and ISDN Systems, vol. 30, no. 24, Pages: 2309-2326, Dec. 1998.

[Avalon 2004] "Avalon Interface Specification", Manual, Altera Corp., 2004.

[Bae 1991] Bae, J.J.; Suda, T., "Survey of Traffic Control Schemes and Protocols in ATM Networks", Proceedings of the IEEE, 79, 2, Pages: 170 – 189, 1991.

[Beran 1992] J. Beran, "Statistical Methods for Data with Long-Range Dependence", Statistical Science 7, No. 4, 1992.

[Berger 2001] C. Berger, M. Kossel, C. Menolfi, "High-density Optical Interconnects within Large-Scale Systems", IBM Research, Zurich Research Laboratory, Switzerland.

[Beshai 2000] Maged Beshai, Richard Vickers, "PetaWeb Architecture", Networks 2000, Towards Natural Networks: 9th International Telecommunication Network Planning Symposium, Toronto 2000.

[Bishtein 2004] V. Bishtein, T. Hall, "Input Sector Design and Implementation", project report, University of Ottawa, 2004.

[Chapman 2002] Stephen J. Chapman, "Java for Engineers and Scientists", Prentice Hall, USA, 2000.

[Chu 1995] P.P. Chu, "ATM Burst Traffic Generator", VLSI, 1995. Proceedings, Fifth Great Lakes Symposium on, 16-18 Pages: 262 – 265, 1995.

[Chi 1991] H. S. Chi and Y. Tamir, "Decomposed Arbiters for Large Crossbars with Multi-queue Input Buffers", Proceedings of International Conference on Computer Design, Cambridge, Massachusetts, Pages: 233-238, October 1991.

[Crovella 1997] M. E. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes", IEEE/ACM Transactions on Networking, Pages: 835--846, December 1997

[Dunkels 1998] A. Dunkels, "lwIP - a lightweight TCP/IP stack", URL: http://www.sics.se/~adam/lwip/

[Hluchyj 1988] M. Hluchyj, M. Karol, "Queuing in High-Performance Packet Switching", IEEE Journal on Selected Areas in Communications, Vol. 6, No. 9, December 1988.

[Huang 1993] K. Hwang, "Advanced Computer Architecture Parallelism, Scalability, Programmability", McGraw-Hill, 1993.

[Gospodinova 2004] E. Gospodinova, "Analysis of High Performance Packet Switch with Input Smoothing", International Conference on Computer Systems and Technologies - CompSysTech'2004.

[Karol 1992] M. Karol, K. Eng, and H. Obara, "Improving the Performance of Input Queued ATM Packet Switches", INFOCOM'92, Pages: 110–115.

[Katevenis 2005] Manolis Katevenis, "Packet Switch Architecture", University of Crete, Greece, Spring 2005. URL: http://archvlsi.ics.forth.gr/~kateveni/

[Labrosse 2001] J. Labrosse, "MicroC/OS-II, the Real-Time Kernel", Book and Disk 2nd Edition, 2001.

[Lekkas 2003] Panos C. Lekkas, "Network Processors", McGraw-Hill Professional, 2003.

[Leland 1994] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)", IEEE/ACM Transactions on Networking, Vol. 2, No. 1, Pages: 1-15, February 1994.

[Paredes 2005] S.A. Paredes, "Flexible Bandwidth Provision and Scheduling in a Packet Switch with an Optical Core," Ph.D. thesis, University of London, U.K., 2005.

[Paredes1 2005] S. A. Paredes and T. J. Hall, "Flexible Bandwidth Provision and Scheduling in a Packet Switch with an Optical Core". OSA Journal of Optical Networking, Vol. 4, No. 5, Pages: 260-270, May 2005.

[Paredes2 2005] S. A. Paredes, T. J. Hall, "Flexible Bandwidth Provision in a Sectored Packet Switch with an Optical Core", The Tenth IEEE Symposium on Computers and Communications (ISCC 2005), La Manga del Mar Menor, Cartagena, Spain, June 27-30, 2005.

[Pmc-sierra 1999] "A New Architecture for Switch and Router Design", TT1 Chipset, white paper PMC-Sierra, 1999.

[Pellerin 2005] D. Pellerin and S. Thibault, "Practical FPGA Programming in C", Prentice Hall, 2005.

[QNX 2005] QNX Systems Software, URL: www.qnx.com

[QuatrusII 2005] "Quartus II Development Software Handbook", Altera Corp., 2005.

[Metconnex 2005] "Metconnex WSS 5400 Evaluation Manual", Metconnex Inc., 2005.

[Mckeown 1996] N. McKeown, V. Anamtharam, and J. Warland, "Achieving 100% Throughput in an Input-queued Switch", Proceedings INFOCOM'96, San Francisco, Pages: 296-302, March 1996.

[Mckeown 1999] N. McKeown, "The iSLIP Scheduling Algorithm for Input-queued Switches", IEEE Transactions on Networking, vol. 7, no. 2, Pages: 188-201, April 1999.

[McKeown 2005] N. McKeown, B. Prabhakar, "Packet Switch Architectures", Stanford, 2005.

[Mekkittikul 1998] A. Mekkittikul and N. McKeown, "A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-queued Switches", Proc. IEEE INFOCOM 1998, vol. 2, San Francisco, Pages: 792-799, Apr. 1998.

[Mekkittikul 1996] A. Mekkittikul and N. McKeown, "A Starvation-free Algorithm for Achieving 100% Throughput in an Input-queued Switch," Proc. ICCCN'96, Washington D.C., Pages: 226-231, October 1996.

[Mentor 2005] "Modelsim User Guide", Mentor Graphics, 2005.

[Rumsey 2003] Francis Rumsey, John Watkinson, "Digital Interface Handbook", 2003, Focal Press.

[SearchNet 2006] http://searchnetworking.techtarget.com

[Stallings 2000] William Stallings, "Data & Computer Communications", 6th edition. Prentice Hall, USA, 2000.

[Stallings 2002] William Stallings, "High-speed Networks and Internets: Performance and Quality of Service". Upper Saddle River, N.J., Prentice Hall, 2002

[Stratix 2005] "Nios Development Board Reference Manual, Stratix Professional Edition", Altera Corp., 2005.

[Stiliadis 1996] D. Stiliadis, "FAST: A Simulation Testbed for ATM Networks", Compcon '96. 'Technologies for the Information Superhighway', Digest of Papers 25-28 Pages: 32 – 37, Feb. 1996.

[Taebi 2004] S. Taebi, S. Paredes, T.J. Hall, "Performance of a Sectored Packet Switch under Self-Similar Traffic", IEEE CCECE 2004, Niagara Falls, May 2004, pp. 747-750.

[Tagami 2002] Tagami, A., Hasegawa, T., Nakao, K, "OC-48c Traffic Tester for Generating and Analyzing Long-range Dependence Traffic", Computers and Communications, 2002, Seventh International Symposium Pages: 975 – 982, 1-4 July 2002.

[Turner 1998] J. Turner and N. Yamanaka, "Architectural Choices in Large Scale ATM Switches," IEICE Transactions in Communications, vol. E81-B, no. 2, Pages:.120-137, February 1998.

[Xilinx 2005] ML 310 Documentation: http://www.xilinx.com/products/boards/ml310/current/index.html

[Yeh 1987] Y. Yeh, M.G. Hluchyj, and A. S. Acampora "The Knockout Switch: A Simple, Modular Architecure for High-performance Packet Switching," 90 IEEE Journal on Selected Areas in Communications, vol. SAC-5, no. 8, Oct. 1987.