UNIVERSITY OF TRENTO - Italy

POLITECNICO DI TORINO

Multi-terabit per Second Scalable Switch Prototype

Technical Report

*University of Trento Via Sommarive, 14; I-38050 POVO, Trento – ITALY

**Politecnico di Torino, Corso Duca degli Abruzzi, 24, 10129 Torino – ITALY

January 2006

This work was supported in part by funds from the European Commission (contract N° MC-EXC 002807).

Table of Contents

Table of Figures

Abstract

As data traffic on the Internet continues to grow exponentially, there is a real need to solve the switch bottleneck by developing ultra-scalable switching fabrics. Although in recent years there have been a lot of efforts to solve the switching fabric scalability problem, in the optical domain, the proposed solutions have been (very) expensive, (very) complex and (much) larger than electronic switching fabric alternatives. Unfortunately, electrical interconnection of existing (off-the-shelf) electronic switching devices is very difficult due to wave reflections on transmission lines, impedance mismatching, crosstalk and noise.

Consequently, electronic switching with electrical interconnections has major scalability limitations. Thus, the question is whether and how optical interconnects can be used to link off-the-shelf electronic switching devices in order to develop optoelectronic switching fabrics that can scale up to 10-100 terabit per second (Tb/s) capacity in a single chassis. In order to further increase the scalability of the proposed novel optoelectronic switching fabric a UTC-based (coordinated universal time) time driven switching (TDS) or fractional lambda switching (FλS) architecture is proposed. This novel low-complexity switching architecture capitalizes on the ubiquitous of UTC from GPS and Galileo. TDS architecture is especially suitable to support high capacity streaming media applications over the Internet.

Keywords: *optical networks; optical switching, sub-lambda switching; fractional lambda switching, time-driven switching, optical interconnect, scheduling, streaming media, switch prototype (key words)*

I. Introduction and Basic Principles

Despite recent slow-down in the telecom industry, the Internet traffic is still doubling roughly 12-18 months. Driven by the proliferation of services such as online interactive entertainment, voice/video/TV over IP and various other real-time streaming media applications, the high-speed Internet access to homes and businesses is becoming a reality. These new applications are projected to be a major source of revenue growth in the networking businesses. Thus, streaming media applications are the main driving force determining the technologies that should be adopted to ensure the continuous profitability of the telecom business.

This paper proposes a low cost optoelectronic switch design. The switch architecture guarantees deterministic QoS for streaming media over the Internet and is scalable to 10-100 Terabit per second. In order to achieve such performance pipeline forwarding of IP packets is used. Pipeline forwarding is a method known to provide optimal performance independent of specific implementation. Invented by Henry Ford, pipeline forwarding is still the most efficient manufacturing process today. All computers today operate using pipelines, a simple extension of Ford's assembly line.

Optical transmissions in the form of wavelength division multiplexing (WDM) have allowed a huge capacity increase. While WDM solves the link bottleneck, the continuous exponential traffic growth, which has to be routed through the Internet backbone, constitutes a major switching bottleneck. The only possible exception is whole wavelength (or lambda) switching, which allows the provisioning of a whole wavelength (WDM optical channel) between source and destination. The whole lambda switching approach suffers from poor scalability, since it requires $O(N^2)$ lambdas, where N is the number of access nodes. Consequently, the whole wavelength switching approach is inefficient (i.e., wasteful) and expensive in the manner in which WDM optical channels are used.

I.A Related Works

Optical burst switching (OBS) [7], was proposed as a middle stage towards the realization of optical packet switching (OPS). A burst accommodates a possibly large number of packets. In some OBS designs, control packets are forwarded in a control channel to configure switching nodes before the arrival of corresponding bursts, hence reducing the requirement of optical buffers. Though OBS is interesting and some protocols were defined for it [8][9], the behavior of burst switching as an asynchronous switching system makes it hard to implement and control the optical switching fabric even when the traffic load is moderate or even low. Besides, grooming traffic into bursts at ingress nodes of OBS networks is another

difficult issue. In general, an asynchronous optical packet switching network may be the ultimate goal for all-optical networking. However, two key technologies have still long way for maturity: realizing asynchronous optical random access memory and asynchronous optical packet header processing.

Time-driven switching (TDS) or fractional lambda switching (FλS) utilizes common time reference (CTR), which can be realized with UTC (coordinated universal time). UTC provides phase synchronization or time of day with identical frequencies everywhere. In contrast, traditional TDM (time division multiplexing) systems, such as SONET/SDH, have neither phase synchronization nor identical frequencies. Thus, unlike systems with UTC, TDM systems are using only frequency (or clock) synchronization with known bounds on frequency drifts. Early results on how UTC is used in packet switching were published in [10][12]. In addition, there are major challenges for implementing SONET/SDH TDM in the optical domain. Nevertheless in the past ten years there were number of works on combining WDM with TDM [13]-[15]. None of these works was using UTC with pipeline forwarding, as discussed in Section I.B., and lack of the detailed treatment of critical timing issues. Specifically, regarding the accumulation of delay uncertainties or jitter and clock drifts, which is solved by using UTC with pipeline forwarding.

In [13], an optical time slot interchange (TSI) utilizing sophisticated optical delay lines is described with no detailed timing analysis. In [14] and [15] two experimental optical systems with in-band master clock distribution and optical delay lines are described, with only limited discussion about timing issues. In [15] a system with constant delays and clocks is described, which can be viewed as a close model to immediate forwarding (see Section I.C), however, no timing analysis and no consideration of non-immediate forwarding (see Section I.C).

I.B TDS Timing Principle

Sub-lambda or fractional lambda switching (TDS) was proposed as an effort to realize highly scalable dynamic optical networking [3]-[6], which requires minimum optical buffers. TDS has the same general objectives as in OBS and OPS: gaining higher wavelength utilization, and realizing all-optical networks. In TDS, a concept of *common time reference* using UTC (coordinated universal time) is introduced. A UTC second is partitioned into a predefined number of time-frames (TFs). TFs can be viewed as virtual containers for multiple IP packets that are switched at every TDS switch, based on and coordinated by the UTC signal. As shown in Figure 1, a group of K TFs forms a time-cycle (TC); L contiguous time-cycles are grouped into a super cycle (SC), for example, in Figure 1, K=1000 and L=80. To enable TDS, TFs are aligned at the inputs of every TDS switch before being switched. After alignment, the delay between any pair of adjacent switches is an integer number of TFs. The central element of TDS is the method of pipeline

forwarding with the necessary requirement for common time reference – e.g., UTC. In TDS, a fractional lambda pipe (FλP) p is a predefined schedule for switching and forwarding TFs along a path of subsequent TDS switches. The FλP capacity is determined by the number of TFs allocated in every time cycle (or super cycle) for the FλP p. For example, for a 10 Gb/s optical channel and K=1000, L=80 if one TF is allocated in every time cycle or super cycle the FλP capacity is 10 Mb/s or 125 Kb/s, respectively.

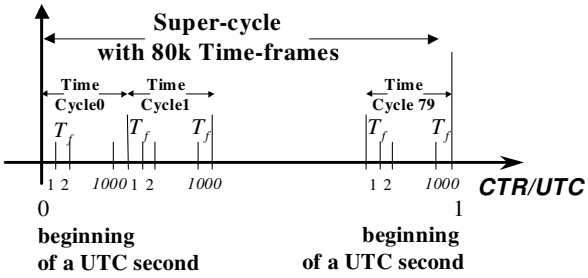


Figure 1. Division of an UTC second in TDS

I.C TDS Forwarding Principle

TDS defines two possible types of forwarding, as shown in Figure 1. The first one is immediate forwarding (IF): upon the arrival of each TF to an TDS switch, the content of the TF (e.g., IP packets) is scheduled to be "immediately" switched and forwarded to the next switch during the next TF. Hence, the buffer that is required is bounded to one TF and the end-to-end transmission delay is minimized.

The other type of packet forwarding is called non-immediate forwarding (NIF). NIF requires buffers at TDS switches. Let us assume that, at each switch, there is a buffer of B TFs at each input channel. The content of each TF arriving to the TDS switch can be buffered for an arbitrary number k_b of TFs ($1 \le k_b \le B$) before being forwarded to the next switch. NIF offers greater scheduling feasibility than IF.

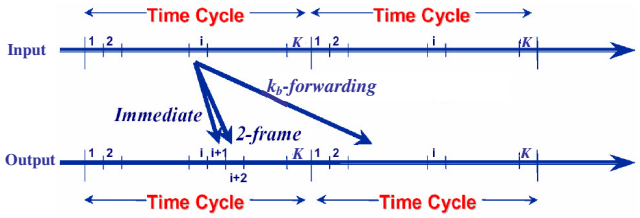


Figure 2. Illustration of IF and NIF in time domain

I.D Time-driven priority

Previous sections presented TDS principles of work, focusing on its features and strengths. These sections should point out as TDS is more suitable in very high speed optical backbones, where traffic could

be divided in FλPs outperforming a layer-2 switching based only on the time. More flexibility could however be required at the edge of the network – for example, the conventional IP destination-address-based routing. Time-driven priority (TDP) [18] is a synchronous packet scheduling technique that can ensure this level of flexibility while maintaining the same principles of work and the same strengths that characterize the TDS technology. The difference is that packets belonging to the same TF that enter the switch from the same input port could be sent out from different output ports, according to the rules that drive an internal forwarding module. So, routing and forwarding may be indifferently based on the already mentioned conventional IP destination-address-based routing, or multi protocol label switching (MPLS), or any other technology of choice.

II. Switch Prototype General Description

II.A Design and implementation considerations

The innovation introduced and presented in the first section of this article makes experimental demonstration mandatory for the verification of concept. Simplicity, user friendliness and low cost, using off-the-shelf components are the main features of proposed high speed switch architecture. Keeping in mind these features/objectives the biggest challenge is to select suitable components and device for constructing the switch fabric on a single chassis.

After surveying the market and performing in-depth analysis of available components e.g. switching devices and optical links, we short listed, High-Performance Evaluation Board manufactured by Mind Speed. Mind Speed cross point switch is a low-power CMOS, high-speed 144 x 144 crosspoint switch with integrated CDRs, input equalization, and built-in system test features. Each switch maximum capacity is 3.2Gbps. The Mind Speed switch board supports SONET, GbE, 10GbE, Fibre Channel (1x, 2x, 10x) and Infiniband applications, it provides flexibility in selecting suitable optical gigabit Ethernet transceivers and related fiber links for physical connection for data transport. The board features also include programmable input equalization with Clock and Data Recovery (CDR) for random and deterministic jitter reduction. To achieve the best possible signal, each CDR is preceded by a programmable input equalizer (IE), and each output includes Output Pre-emphasis (PE). The IE removes ISI jitter usually caused by PCB skin effect losses. The IE circuit opens the input data eye in applications where long PCB traces and cables are used. As a result, the device exceeds the SONET requirements with jitter tolerance of 0.65 UI, and jitter generation of less than 1.5mUI (rms) or 8mUI (peak-to-peak). The PE provides a boost of the high

frequency content of the output signal, such that the data eye remains open after passing through a long interconnect of PCB traces and cables.

The Mindspeed switching matrix chip is installed on a board having 8 input and 8 output ports high speed data links with SMA connectors for balanced electrical connections. It is complete with parallel and high speed serial control busses and an interchangeable computer interface (controller).

A FPGA controller for dynamic configuration of the switch has been designed and fabricated. The FPGA controller allows the dynamic configuration of the switch and implements the UTC controlled routing at the switch level. The GUI for the UTC controlled routing is implemented on a personal computer, which is connected to the FPGA controller by means of a USB link. The GUI facilitates the connection of the various input, output ports at different time frames.

UTC time and frequency reference is obtained from a high performance GPS signal receiver connected to GPS antennas.

Ten megahertz and one pulse per second signals are generated by a Tekelec low cost GPS frequency generator, operating with accuracy comparable to rubidium oscillators. The most important feature that GPS time is distributed all over the world in a consistent way. With the help of this technology all switches have the same time signals with discrepancies of the order of one microsecond.

The combination of the switching board, GPS receiver and FPGA controller plus a windows PC dedicated to controller initialization, implements a switching layer. A dual layer Banyan network switch has been implemented by connecting two switching boards to a single controller.

A complete network has been also implemented by using the dual layer switch and a single layer switch interconnected by a 25 km gigabit fiber data link.

The Mindspeed experimental board can be programmed to correctly interface to standard GBIC transceivers. Standard multimode transceivers have been used for optical interconnections between the switch and data sources and destinations. Bidirectional 1300-1550 nm single mode transceivers have been used for interconnecting the two switches to create the demonstration network.

II.B End-to-end prototype setup

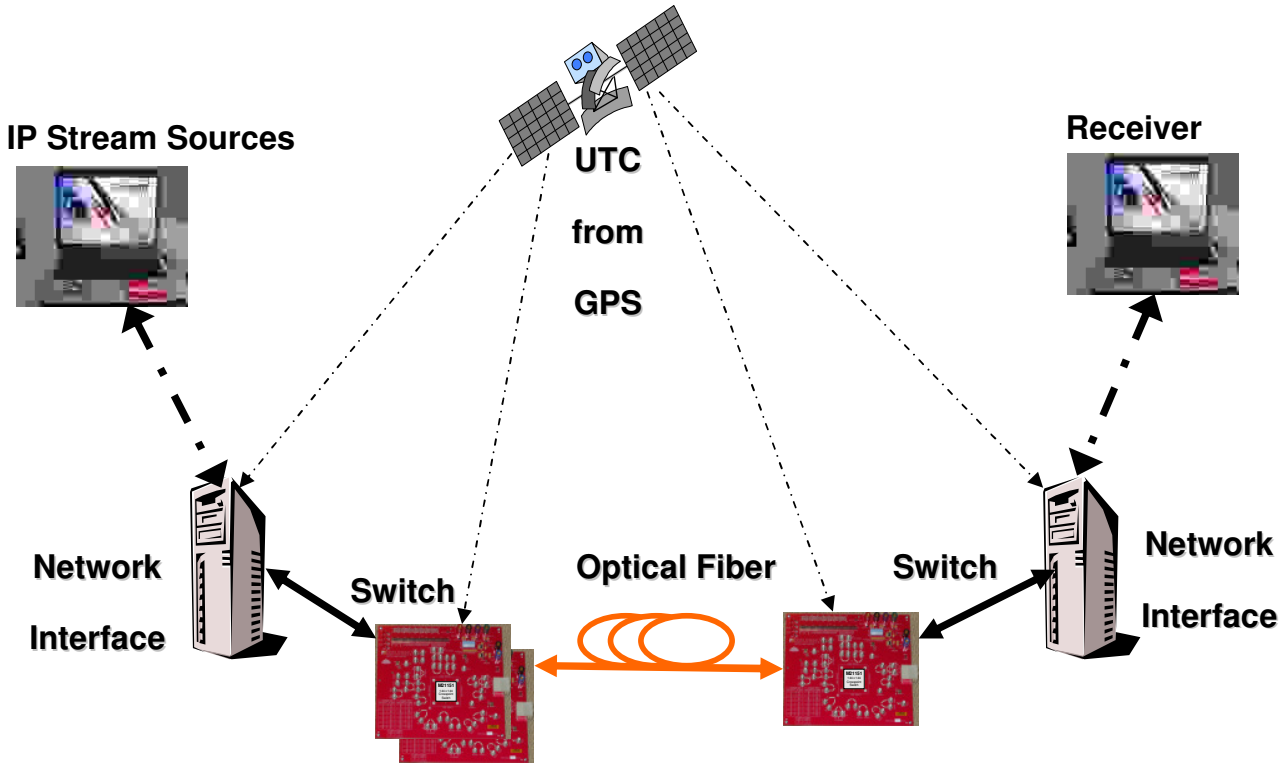


Figure 3. End-to-end Prototype setup

The diagram shown above is an overview of the prototype. The setup major components are streaming sources for streaming audio, video and or text, a network interface which schedule the packets forwarding, the Switch Fabric consists of two Mindspeed switch boards and FPGA based switch controller for dynamic configuration of switch, 25km optical fiber. On the other side of optical fiber there is another switch fabric consists of one Mindspeed Switch Board, network interface and two receivers for receiving and playing the two movie streams with sound transmitted from the source.

The detail setup diagram (one side of optical fiber) with all components and devices used is shown in Figure 4.

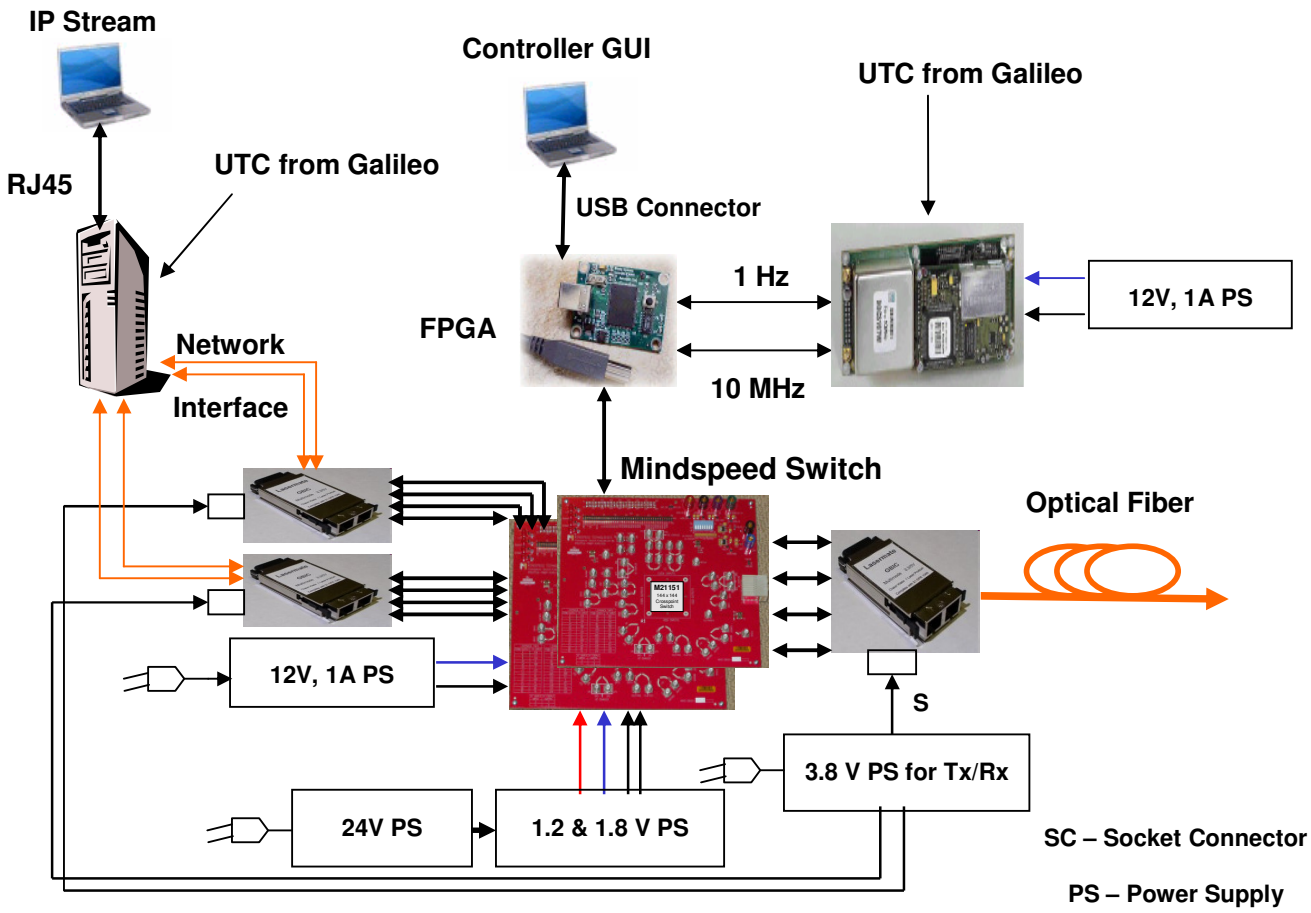


Figure 4. Detailed setup diagram

Two streaming films destined for two different receivers, one DVD film with soundtrack and subtitle while other animation film with soundtrack only, are transmitted from one PC (shown IP Stream) using VLC media player. VLC (initially VideoLAN Client) is a highly portable multimedia player for various audio and video formats (MPEG-1, MPEG-2, MPEG-4, DivX, mp3,...) as well as DVDs, VCDs, and various streaming protocols. It can also be used as a server to stream in unicast in IPv4 or IPv6 on a high-bandwidth network. The asynchronous packets are sent to the router which schedules the packets forwarding in synchronization with 1pps signal from GPS through GPS antennas and subsequently through TekLec card. The TekLec card is powered by 12V, 1A power supply. The switch controller has been implemented on Opal Kelly FPGA module. Each FPGA module can support two Mindspeed switch boards. Each Switch board is powered by two power sources 12v, 1A and 1.2V, 1.8V. A dual layer Banyan network switch has been implemented by connecting two switching boards to a single controller. Standard multimode transceivers have been used for optical interconnections between various Electrical to Optical

and vice versa conversion points. Bidirectional single mode transceivers have been used for interconnecting the two switch boards. A 25km optical fiber is used to connect the two switch fabric of the network. Switch fabric on the other side (or receiver side) of 25km fiber consists of one Mindspeed switchboard unlike the two boards used on the source side of fiber. Two PCs having VLC multimedia player are connected to switch fabric for receiving and playing the two video streams.

III. Time-driven Switch Interface

The Internet is mostly based on asynchronous packet switches. Thus, especially in an initial deployment phase, TDP/TDS switches would have to coexist and interoperate with current asynchronous packet switches as depicted in Figure 5. Synchronous *edge routers* would be required in order to control the access to the synchronous backbone by policing and shaping the incoming traffic flows. This section reports on the implementation of a PC-based TDP router developed over a FreeBSD platform, which could be used as edge router for interfacing the time-driven switch prototype. In particular, this section focuses on the router architecture and on its programming procedure.

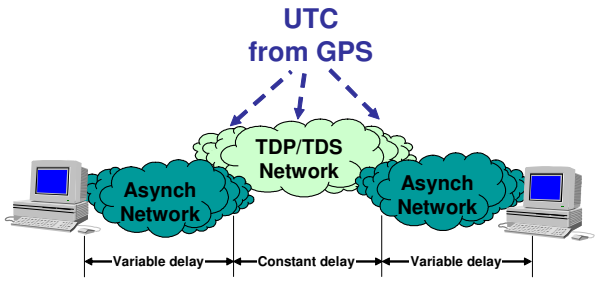


Figure 5. Interoperation between TDP/TDS networks and asynchronous networks.

III.A Software architecture

In any generic router data plane, packets are moved from input ports to output ports going through three modules that perform *input processing*, *forwarding*, and *output processing*. The operations to be performed by a TDP router in each module are discussed here.

Edge routers need to shape asynchronous traffic entering the TDP network. Consequently, their input module comprises mechanisms to classify incoming packets, identify the data flow they belong to, and select a TF in which they will be forwarded according to the current resource reservation setup. The evaluated *forwarding TF* determines the output buffer where packets will be stored by the output module.

Dummynet [19] is an *ipfw* (a FreeBSD firewall) extension for selecting packets using programmable rules and pass them through objects called pipes, which are used to emulate bandwidth and resource limitations, propagation delays and packet losses. The input module uses Dummynet to classify incoming asynchronous data flows and store packets up to one TF before they are fetched by the output module for transmission.

The forwarding module processes packets according to the technology on which TDP is deployed, which does not require any modification for supporting TDP. This stems from the fact that TDP is just a packet-scheduling technique that can be deployed in the context of any packet-switching technology, without any requirements or impact on the forwarding module operation.

The output module deploys a per-TF, per-output queuing system, where packets to be forwarded during the same TF through the same interface are buffered in the same queue. The queue in which each packet is stored is determined by both the input module, which decides the forwarding TF, and the forwarding module, which selects the output interface. The output module is also responsible for the timely transmission of all the packets stored in the queues to be flushed in the current TF.

We decided to use the FreeBSD open-source operating system because it is a reference for networking-related projects for historical reasons (the TCP/IP network stack was first developed on the BSD platform), it is well documented in [20], and comes with high-quality traffic-management tools, such as the alternate queuing (*ALTQ*) framework [21], [22]. Because ALTQ includes many packet-scheduling policies such as First-In First-Out (FIFO), Weighted Fair Queuing (WFQ), Class Based Queuing (CBQ), hierarchical fair service curve (HFSC), TDP output module can be simply implemented as just another scheduling discipline.

The CTR is provided to the router using a Symmetricom [23] GPS receiver PCI card that can generate interrupts at a programmable rate ranging between less than 1 Hz and 250 kHz. Two global variables are added to the FreeBSD kernel to hold the current TF and TC number; they are updated whenever these interrupts occur.

The current development version supports only immediate forwarding and does not include signaling functions, so that TFs and TCs are allocated statically by manual configuration. Nevertheless, the prototype does not loose the generality and we paid the utmost attention to accurately avoid any design choice that could prevent us from adding signaling functions and implementing non-immediate forwarding in the future.

III.B Detailed programming procedure

This user guide addresses to how to configure the TDP router's parameters and networking in the scenario of TDP system.

1. Configuration

The test-bed is combined of the following main components:

- TDP router
- FλS switches
- Some sources
- Some destinations

One of the possible configurations can be as follow:

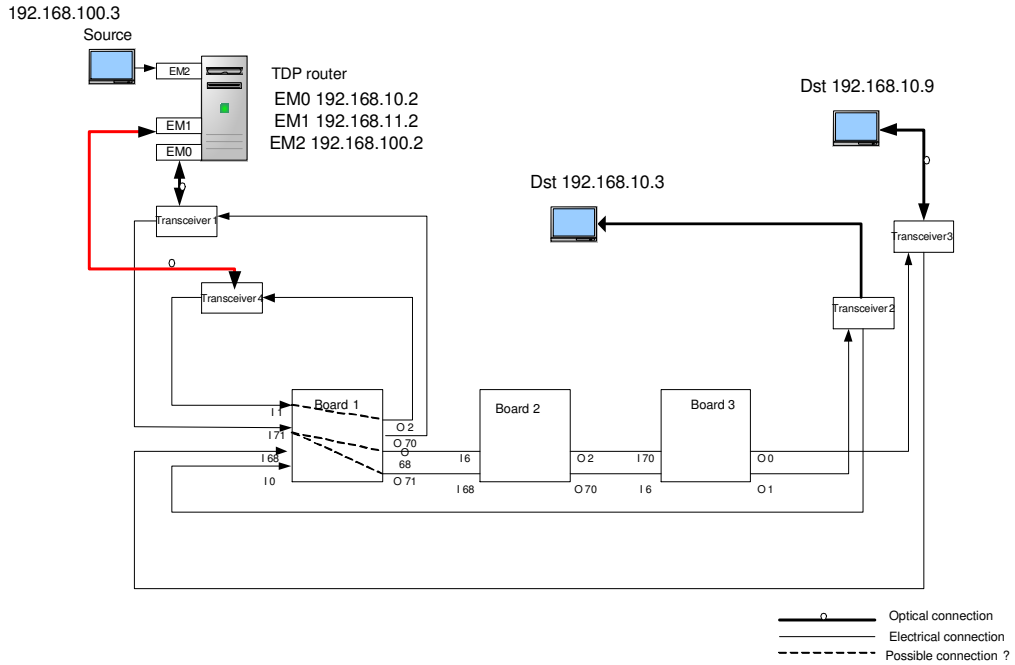


Figure 6. Simple configuration

- From the same source 192.168.100.3, two traffic flows are transmitted to different destinations. They are all forwarded from the source to the interface em2 of the TDP router (192.168.100.2), to the em0 (192.168.10.2), and going out to the switches.
- After the third switch, the 2 flows are split into 2 PHY links going to the 2 destinations (192.168.10.3 and 192.168.10.9)

In fact, this script only guides you to set up the TDP program that relates to the issue of routing table. Every step of setting the routing table and enabling the IPFW is given in the script file of the TDP router, which can be found in the directory /usr/TDPprogram/scripts. To enable the ALTQ, run the file in /usr/TDPprogram/altq-configuration-files/

2. Script files

The first step is understanding and learning to develop a simple script file used for configuring some networking aspects in the TDP router.In particular, this script file has to set up the ARP table, the routing table, some firewall rules.

ARP table. TDP interfaces cannot currently handle the ARP protocol, so manually configuring ARP entries for these ones is required. The *arp* command could be used.For example, in the scenario presented in Figure x, there is only one TDP interface (em0) that has to reach two different destinations. Commands should be in this case:

arp –s 192.168.10.3 [MAC address destination 1]

arp –s 192.168.10.9 [MAC address destination 2]

Routing table. Some routing tables could be required: it depends on the particular scenario. In the example one in Figure 1, destinations belong to the same IP subnet of the TDP router (the subnet 192.168.10.0/24), so no routing is required. Note that the FλS switch is a Layer 2 one, so routing is not required in any scenario similar to the one in Figure 1, where destinations are directly connected to the switch. Routing could however be required if other routers are put after the FλS switch. A possible scenario could be:

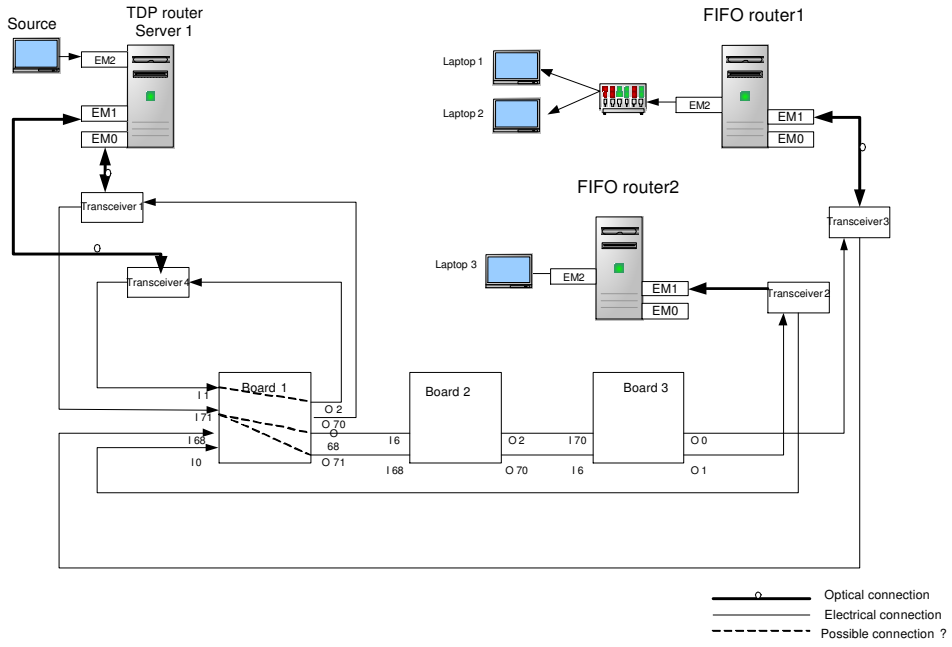


Figure 7**.** Another configuration

Here routing tables must be set in the TDP router using *route* command:

route add [dest1] [IP address of next-hop] [netmask]

route add [dest2] [IP address of next-hop] [netmask]

etc…

Firewall rules. Packets that arrive in a TDP router from an asynchronous network could become TDP ones (if a schedule is defined for them) or remain Best-Effort ones (if no schedules are defined for them). The procedure for defining a schedule will be presented in the following session, but here we want to point out how packets have to become TDP ones are intercepted by the system. The FreeBSD firewall *ipfw* is used for this purpose: a particular rule is required for each asynchronous flow that has to become synchronous. In the example, we have two video streaming asynchronous flows: the flow 1 (192.168.100.3 → 192.168.10.3) and the flow 2 (192.168.100.3 → 192.168.10.9). Both have to enter the Time-Driven network becoming synchronous, so two rules for intercepting them are required in the TDP router. They are:

ipfw add pipe 1 ip from 192.168.100.3 to 192.168.10.3 in recv em2

ipfw pipe 1 config queue 100

for the flow 1 and

ipfw add pipe 2 from 192.168.100.3 to 192.168.10.9 in recv em2

ipfw pipe 2 config queue 100

for the flow 2.

Note that the TDP router receives both flows through the em2 interface, so the last parameter of the just described rules is “in recv em2”.

Note also that the ipfw default rule is “ipfw deny any from any”, so an “allow everything” rule is required, too. So type:

ipfw add allow ip from any to any

In order to allow all Best-Effort packets (pings, no-TDP flows, etc.) go through the network.

PAY ATTENTION:

1. The order of the rules is very important! Follow the previous presented one (pipe1 rule, pipe 2 rule, pipe xxx rule,…,"allow everything" rule).

2. Before changing a pipe or adding a new one, please type "ipfw flush" in order to prevent the just mentioned out-of-order mistakes…And then remember to add the "allow everything" rule again! So, if you want to change the pipe 2 of the previous example for intercepting packets that go to 192.168.10.7 instead of the ones that go to 192.168.10.9, you will write:

    ipfw flush

    (type 'y' at the confirmation request)

ipfw add pipe 1 ip from 192.168.100.3 to 192.168.10.3 in recv em2

ipfw pipe 1 config queue 100

ipfw add pipe 2 from 192.168.100.3 to 192.168.10.7 in recv em2

ipfw pipe 2 config queue 100

ipfw add allow ip from any to any

Script file. All these commands could be inserted in a "script file" that you can run at system startup. For some example, see /usr/TDPprogram/scripts.

3. TDP router's parameters

The file containing the TDP parameters is in the path /usr/TDPprogram/altqd-configuration

In order to enable the altqd, type the command:

altqd –f [configuration file]

An example of the ALTQ configuration files for configuring and enabling TDP is as follow:

tdp 3 1000 3 10 100 500 2

interface em0 bandwidth 1000M tdp

pipe 1 2 10

pipe 1 12 10

pipe 2 5 6

pipe 2 7 6

The first row consists of:

- Maximum TDP buffer length (in number of packets)
- Number of TF/TC
- Maximum BE buffer length (in number of packets)
- TDP packet length (in bytes)
- TF duration (in μs)
- Number of dummy net pipes

The second row specifies the interface for TDP required by APTQ

interface [name of the interface] bandwidth [rate of the interface] tdp

The last rows contain Dummy net rules relating to input processing

Pipe [flow] [TF to transmit packets] [number of packets transmitted in the defined TF]

To change into the Debug mode, replace the last parameter of the first row into Zero

tdp 3 1000 3 10 100 500 0

The debug mode provides information about synchronization in the first console, and the index of TF in that packets are sent. To see the information of transmitting TF, use the command:

tcpdump –i [NIC interface] –vv

for example:

tcpdump –i em0 -vv

III. C Symmetricom time card

Symmetricom bc637PCI is a receiver module which can work acquiring time from either GPS (*GPS mode*) or time code signals (*Time code mode*). If used in GPS mode, it provides a time reference aligned with UTC with the accuracy of better than 1μs to the host computer. It can transmit the information about the current time over the PCI bus whenever the PC makes a time request. This card can trigger a programmable periodic interrupt over the PCI bus used to perform TF update as explained in Section III.A.

Since transmissions are driven by the CTR, packets should be sent out as soon as the TF begins. Driving these transmissions using the above described interrupt intrinsically introduces a certain level of imprecision and unpredictability due to the non-zero variable latencies with which a PC handles incoming interrupts. Three fundamental steps have to elapse before sending procedure can start: *(i)* the card has to acquire the PCI bus, *(ii)* the system has to react to the incoming interrupt calling the respective interrupt

handler, *(iii)* the interrupt handler has to acknowledge the interrupt to the card (i.e., it has to acquire the bus again for setting some card registers). These steps take an unpredictable amount of time because of the non-real-time design of the PC architecture. This is an open issue we are investigating in order to find out a solution for guaranteeing very precise packet transmissions.

Another open problem is related to the time that is necessary to synchronize the card with the GPS. First of all, the receiver has to acquire one or more satellites, and then its internal oscillator has to be synchronized in phase and in frequency with the UTC time provided by the GPS system. This procedure takes several minutes, resulting in absolutely unacceptable delay in a networking environment.

The just discussed issue is certainly related to the internal design of the Symmetricom card, but it could also be due to the antenna type and positioning; there are in fact two antenna types that could be used with GPS timing receivers: roof antennas and window antennas. The first one provides a more accurate time reference but it requires at least three satellites always in view to maintain timing accuracy. The second one could be positioned near the window viewing only a portion of the sky; in fact it can work with an intermittent view of only one GSP satellite providing a less accurate time reference. Since we are using a roof antenna, its stringent requirement related to the view of the satellites and the degree of accuracy to be provided could be the cause of the raised amount of time required for obtaining the system synchronization. However, since the accuracy provided by window antennas is typically microseconds to UTC (the accuracy provided by roof antennas is nanoseconds to UTC), this type of antenna should be enough for our system requirement, and probably further reduces the above discussed synchronization time.

III. D Gigabit Ethernet controller

The Intel 1000 MF Server Adapter is a high performance network card which can support many of the IEEE standards related to local networks. We are however more interested in some of its technical features: a *busmaster DMA* PCI card rigged with a fairly good size transmission buffer (64 KB).

Regarding the DMA transfers, the on-board DMA controller can use a certain number of *DMA descriptors* which can be programmed by users between 80 and 4096. Each packet must be associated with a descriptor in order to allow its transfer over the PCI bus. When a DMA transfer starts, all packets already associated with a descriptor are transferred in burst over the bus, forming a *DMA chain*. If TDP packets that must be sent out in a certain TF are transferred over the bus by a DMA chain and the amount of bytes related to them is less than 64 KB (the transmission buffer size), our TDP implementation performances increase, because links and PCI bus bandwidth would be better utilized. In fact, once packets are into the

transmission buffer, they can be sent out in burst mode without interruptions which would cause link bandwidth wastes; furthermore, burst transfers over the bus reduce the PCI resource acquisition overhead.

Unfortunately there are some issues that clash with TDP requirements. We have already discussed about the variable delay which characterizes the TDP transmission procedure due to the unpredictable latencies with which the GPS receiver interrupt is handled by the system. But a further unpredictable delay is added by the above described transmission procedure. In fact, once the TDP scheduler driven by the GPS card interrupt begins packet transmissions, the Intel NIC has to start a DMA transfer, which takes an unpredictable amount of time for being performed. Furthermore, we don't exactly know how the Intel NIC handles outgoing packets. We don't exactly know how the internal buffer affects the time at which packets are sent out.

The development of a high performance NIC where packets could be stored during the TF before the one in which they have to be sent would be the best solution that probably resolves these variable latency issues; this card should then be directly driven by the GPS timing signal in order to allow packets to go out as soon as possible.

I. IV. Switch Controller

The switch controller architecture is shown in Figure 8.

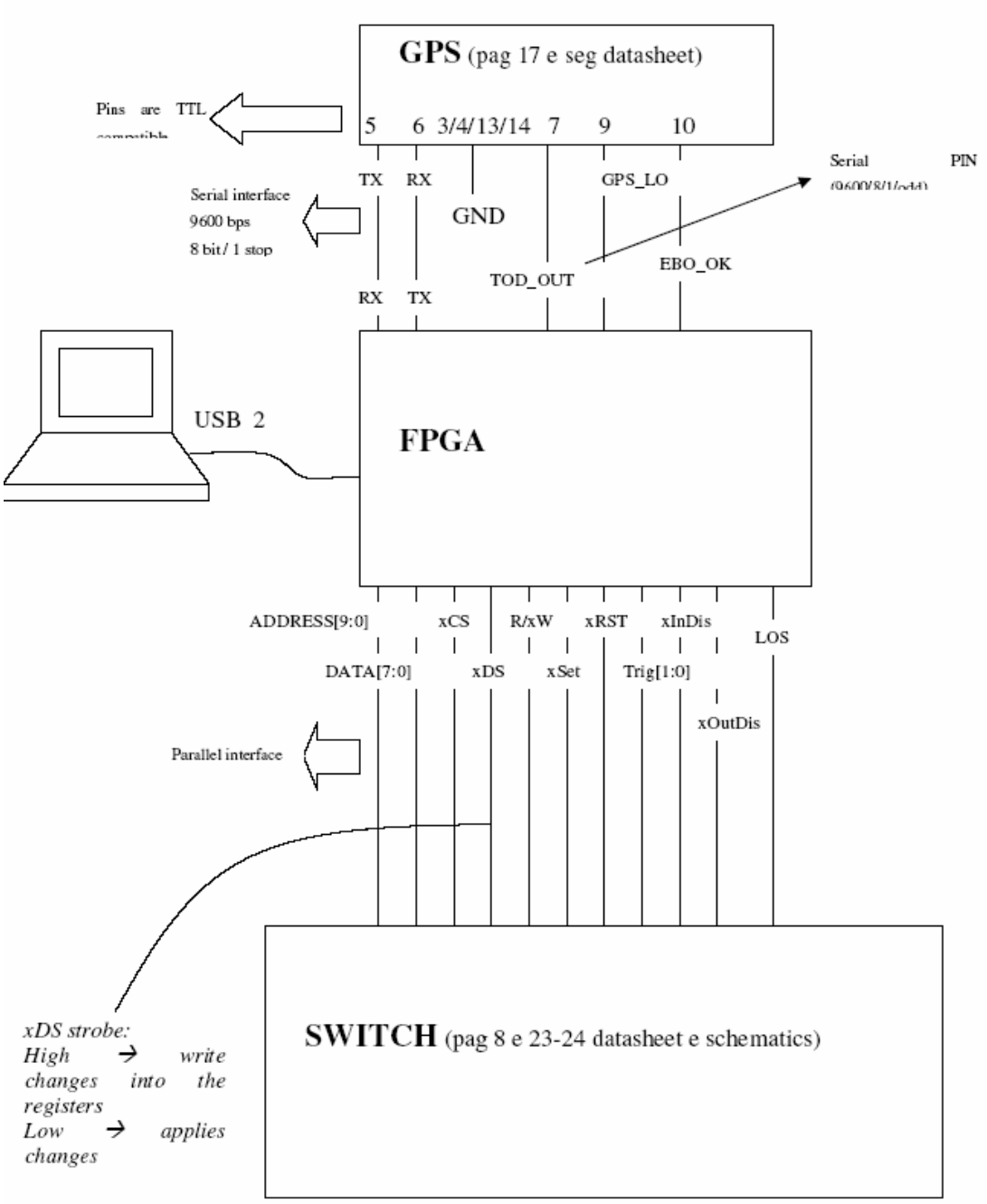


Figure 8. Switch Controller Block Diagram

Following is a short description of the hardware interconnections required in order to connect the three boards.

Epsilon GPS Board

- Tx/Rx : serial connection 9600 baud, 8 bit + 1 stop, odd parity

- GPS_lock : status of the link to GPS system: stable, unstable or unexistent
- EBO_OK : status of correct working of the board
- TOD : “time of day”, signal containing information about timestamp
- Reset : reset signal
- J1 : sine-wave signal @10 Mhz (5 db load 50 Ohm)
- J2 : 1PPS, one Pulse Per Second (H > 2.4 V @ 50 Ohm, L < 0.8 V @ 50 Ohm)

Switch Mindspeed M21151

- Address[9:0] : Address Bus
- Data [7:0] : Data Bus
- xCS : Serial/parallel: active low I/O enable
- xDS : Parallel I/0: data latch, serial I/0: serial clock (hysteresis)
- R/xW : Parallel I/0: H = read, L = write
- xSet : Hardware xSet terminal enables switching multiple channel configurations simultaneously (active low with internal pull up)
- xRST : Hardware reset (active low with internal pull-up)
- xTEST : Mindspeed test terminal (active low with internal pull-up)
- Trig [1:0] : CLKTX/16 for use as trigger
- xInDis : Hardware disable of all inputs (active low with internal pull-down)
- xOutDis : Hardware disable of all outputs (active low with internal pull-down)
- LOS : Global loss of signal

Following are the major modules used for implementation of the controller

- Opal Kelly XEM3001
- Mindspeed M21151

Next subsection describes the first module and second one will be described in subsequent one.

IV.A FPGA Module

Field Programmable Gate Arrays (FPGAs) have gone main stream. No longer are applications limited to performance and cost insensitive designs. The newest generation of FPGAs have hit performance and cost goals which allow a much wider spectrum of applications support.

Now a day, more functions are needed in each protocol layer to provide flexible services in telecommunication networks. However, current telecommunication systems are often constructed by dedicated hardware. This is because data transmission requires high throughput and various bit-level manipulations must be performed, which CPUs or DSPs cannot handle well. Hence, current telecommunication circuits are far from rich in terms of flexibility. To help remedy this situation, we have

developed a FPGA based controller for ultra scalable switch for dynamic configuration of switch. Opal Kelly XEM3001 module has been used for switch controller.

Opal Kelly XEM3001

The XEM3001 is an experimentation module based on a 400,000-gate Xilinx Spartan-3 FPGA (XC3S400-4PQ208C). In addition to a high gate-count device, the XEM3001 utilities the high transfer rate of USB 2.0 for configuration downloads enabling an almost instant reprogramming of the FPGA. For flexible clocking, a multi-output clock generator can generate clock frequencies from 1 MHz to 150 MHz. If higher frequencies are needed, the clock multipliers in the FPGA can be used. The XEM3001 is ideally suited to prototype systems and integration into OEM devices where a USB interface, flexible hardware solution, or PC software interface would be useful. The XEM can easily be added to a new board design to provide turnkey USB integration with the convenience of our software programmer's interface and existing USB drivers.

USB Interface: The XEM3001 uses a Cypress CY68013 FX2 USB microcontroller to make the XEM a USB 2.0 peripheral. USB interface makes FPGA downloads quick and data transfer much faster than the parallel port interfaces common on many FPGA experimentation boards. The USB interface also allows the XEM to be bus-powered which means it is ultra-portable requiring just a USB cable and the proper drivers to connect to any supporting PC, including laptops.

EEPROM: A small serial EEPROM is attached to the USB microcontroller on the XEM3001, but not directly available to the FPGA. The EEPROM is used to store boot code for the microcontroller as well as PLL configuration data and a device identifier string.

Multi Output Clock Generator (PLL): A multi-output, single-VCO PLL can provide up to five clocks, three to the FPGA and another two to the expansion connectors JP2 and JP3. The PLL is driven by a 48-MHz signal output from the USB microcontroller. The PLL can generate frequency from 1-150MHz and is configured through the FrontPanel software interface.

EXPANSION CONNECTORS: Three 0.1”-spaced expansion connectors (JP1, JP2, JP3) are available to connect the XEM to external devices. These connectors provide 3.3v power, ground, PLL outputs, and 88 FPGA pins for general I/O. All expansion connectors are on a 100-mil grid so that the entire XEM can piggyback onto a standard 100-mil PCB board.

LEDS and PUSHBUTTONS: Eight LEDs and four pushbuttons are available for general use e.g. for debugging inputs and outputs.

PC INTERFACE: The XEM3001 is fully supported by Opal Kelly's FrontPanel software. FrontPanel augments the limited peripheral support with a host of PC-based virtual instruments such as LEDs, hex displays, pushbuttons, toggle buttons, and so on. Essentially, this makes PC a reconfigurable I/O board and adds enormous value to the XEM3001 as an experimentation or prototyping system.

In addition to complete support within FrontPanel, the XEM3001 is also fully supported by the FrontPanel programmer's interface (API), a powerful C++ class library (and Python wrapper) allowing you to easily interface your own software to the XEM.

The controlling software on PC is implemented in LABVIEW programming environment, interactions between PC and FPGA board will be based on a *.dll dynamic library compiled starting from a C++ class library provided with the board. This type of interface uses more than one 8-bit pipes in order to transfer data. This type of internal data division implies an additional overhead which decreases the transfer rate: instead of 40 MB/s (theoretically guaranteed by USB chip) we can reach 7 MB/s from FPGA and 12.5 MB/s from PC.

One limiting factor is that the XEM is designed to abstract the USB interface from the user. In the interest of providing several available channels (called pipes) between the PC and the FPGA chip, some overhead is incorporated. USB itself is a shared bus, therefore the speed depends on what bandwidth other devices are consuming.

FPGA Implementation is done using Very High Speed Integrated Circuit Hardware Description Language (VHDL). The complete implementation can be divided into following blocks.

SERIAL COMMUNICATION

In order to communicate with the GPS board via a serial interface (not RS232). This block implements the serial standard defined on the Epsilon board manual. The remote control interface allows remote configuration and remote status reporting of the board. The TTL interface operates at 9600 bauds and is set to 8 bits, 1 stop bit and odd parity. The protocol used is Master (Host) / Slave (EPSILON BOARD II) with a systematic reply to all messages with the following exceptions for which no reply is expected:

- The time code message sent periodically.
- The reset board command.

Another serial block is implemented in order to receive the actual Time of the Day (TOD), which is transferred through serial link on another wire. This block is simple because it can only receive data.

GPS STATUS CONTROL

This block controls the status of the board analyzing two wires (GPS_LOCK and EBO_OK) and the messages received by the serial interface. It enables the clock counter and the data transfer between GPS board and PC via USB.

PARALLEL/SPI BLOCK

This block is used to communicate with the Mindspeed Switch Board via serial or parallel interface (the board supports both interfaces). It can transfer data (state register which control the switch settings) between USB interface (PC) and Mindspeed board in both the directions (state register can be red or written). Data transferred by PC are not processed in this block: the bit-configuration of the registers is set by the Labview application.

SWITCH STATUS CONTROL

This block controls the status of the board checking and analyzing the LOS signal.

USB BLOCK

This block receives all messages sent by PC via USB and delivers the commands to one of the two boards (via parallel or serial interface) or to the status control blocks.

COUNTER

The counter is a 64-bit register (it could be only 55 bit = log (150 years expressed in 100ns least count). On the PC a 64-bit register is used, which can be initialized with the UTC reference time and is incremented every 1/(10 million) seconds. This is a clock with the precision of 100 ns. The 10MHz clock is derived by the J1 connection of the Epsilon board. It's possible to use this clock because the FPGA can reach higher clocks (since 150 MHz). The exact value depends on the internal implemented connections and devices.

STORAGE REGISTER

Time value (expressed in 100ns) is stored here i.e. when to apply changes settled into latch registers of the Mindspeed board. In order to change the switch configuration new setting values must be downloaded onto the Mindspeed board and a valid value must be pre-stored into this register. When the counter reaches this value the CONTROL LOGIC block enables the changes on the switch. In this way the change time can be with the precision of 100ns is used.

CONTROL LOGIC

This block checks if the COUNTER and the STORAGE REGISTER contain the same value. If the values are same in that case the signal xDs is switched low in order to load the new values into the M21151 board registers.

HARDWARE VHDL IMPLEMENTATION

The whole control machine is composed by following blocks

PC_GPS_MINDSPEED
GPS_BOARD STATUS CONTROL

MINDPSEED CONTROL

CONFIGURATION MEMORY

SPI INTERFACE

SWITCH CONTROL

USB INTERFACE

Follows a description of the implemented blocks:

CONFIGURATION MEMORY

This entity allocates the RAM space to store the data to be transferred to the MINDSPEED switch registers. Memory is a bi-dimensional matrix with Length = 1024 cells and Width = 18 bit _ 10 register address bit + 8 data bit. The memory written operation is performed when the board is in the update status (status register at bits 5 and 6) and is read operation when the board is in the active status

SPI INTERFACE

This entity implements a FSM that follows the timing rules explained in Figure 9 and 10. The maximum input clock is 50 MHz, the SPI works at 25 MHz.

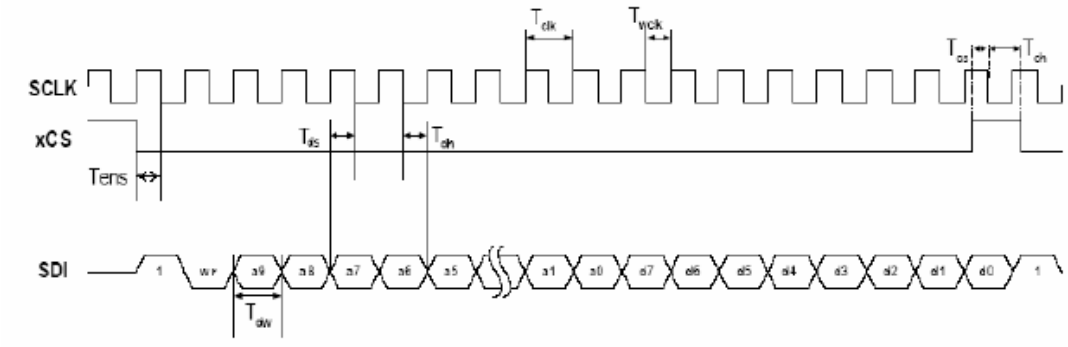


Figure 9. Timing Diagram for Writing

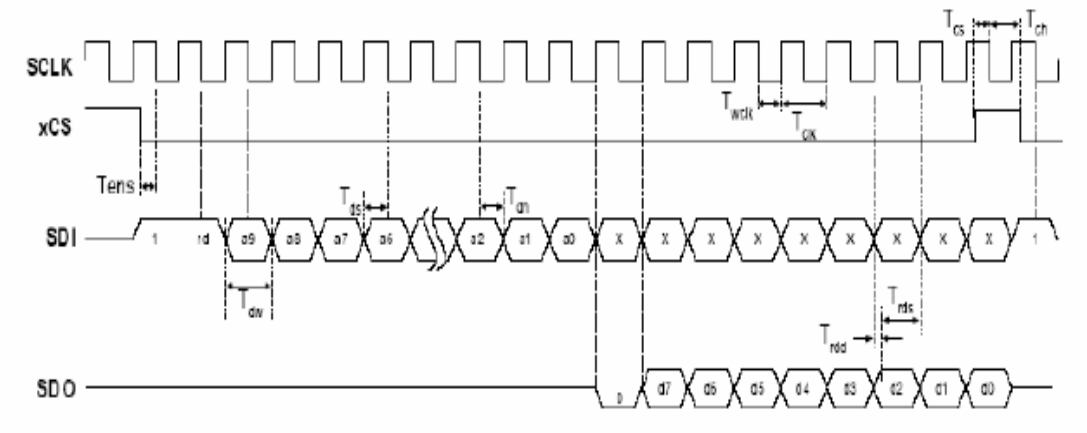


Figure 10. Timing Diagram for Reading

SDI and SDO are in reference to FPGA, inverted with respect to the Mindspeed board. As from the diagram we can see when a read/write operation is terminated, signal (r_term/w_term respectively) goes high until the control signal (read/write) goes down. In this way one can know when an operation is terminated.

Master clock is the input clock (in this case 50 MHz), and SCLK is the output clock (in this case 25 MHz). The output clock is halved of the input clock.

An example of reading and writing operation is shown in figure 11.

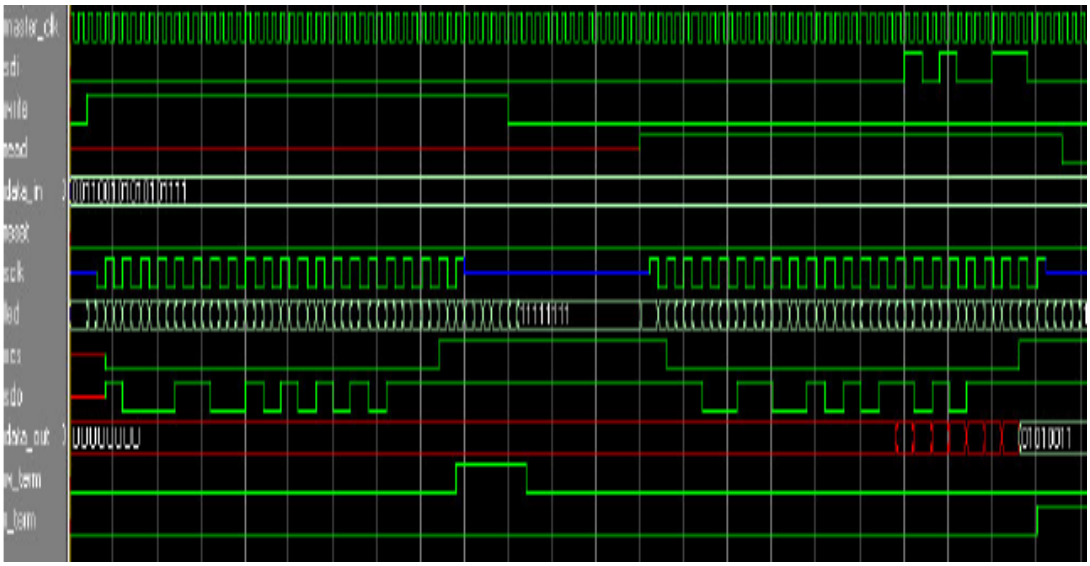


Figure 11. Example of reading and writing operations.

SWITCH CONTROL

Switch Control is time driven and it controls the Minspeed status in the correct time slice. It is composed by following counters: TF_counter, TC_counter.

Every second is divided into Time Cycles (TC) and every time cycle is divided into Time Frame (TF). The base unit is given by 1/(10 MHz) = 100 ns. The TC and TF duration can be modified by changing the inputs (TF_max is maximum number TF in one TC, TC_max is the maximum number TC in one second.

Every TF a new configuration is cyclical downloaded to the Mindspeed board (by putting high the xCS signal). Data are stored into a memory that is read periodically. When whole memory is read and the bottom is reached then it operation restarts from the top (every TC).

There are 2 input clocks: a clock at 10 MHz, which gives the download timing, and a clock at 1 Hz. When this second clock goes high the whole mechanism restarts even if it causes an interruption in the sequential memory download.

If the download terminates before the 1 Hz clock goes high the FSM stops in a waiting status until the clock causes a restart.

SOFTWARE IMPLEMENTATION

Using the functions library provided by OpalKelly to communicate with the XEM3001 board, a C++ interface has been implemented in order to simplify all the interactions between the software running on PC and the circuit implemented into the FPGA. There is public function for every function of the board. Following is a simple description:

class:

switch_control

methods:

reset_device(); // resets the device cleaning everityng on the FPGA

set_PLL(); // sets the PLL values (these values are fixed and not changeable)

configure_FPGA(); // downloads the given bitstream onto the FPGA device

set_clk_source(); // sets the clock source GPS_CLOCK / INTERNAL_CLOCK

set_time_delay(); // sets the timeframe delay (anticipate)

init_switch(); // downloads the control values for the FSM (TF, TC, SEC length)

reset_switch(); // sends a reset pulse to Mindspeed chip

start_switch(); // starts the FSM loaded into the FPGA

stop_switch(); // stops the FSM loaded into the FPGA

memory_write(); // writes data into the FPGA memory

memory_read(); // readds data from the FPGA memory

direct_write(); // writes data directly onto the SPI bus

direct_read(); // reads data directly from the SPI bus

check_status(); // returns the value of the status register

Hers is a brief explanation of methods:

method	reset_device(void)
arguments	*none*
return type	bool
description	calls a function present into the original OpalKelly .dll; resets everything present into the FPGA. It should be used every time before a new bitstream configuration download

method	set_PLL(void)
arguments	*None*
return type	Bool
description	sets the desired CLOCK configuration into the PLL

method	configure_FPGA (char star* FileName)
arguments	char star* FileName // gives the full path of the bitstream
return type	Bool

description	downloads the *bitstream* given by FileName (full path) into the FPGA

method	set_clk_source (bool clk_source)
arguments	bool clk_source can be GPS_CLOCK or INTERNAL_CLOCK
return type	Bool
description	sets the clock source GPS_CLOCK / INTERNAL_CLOCK

method	set_time_delay (unsigned char board_n, unsigned short time_delay)
arguments	unsigned char board_n, // selects which board update unsigned short time_delay // sets the # of delay steps (100ns per step)
return type	Bool
description	sets the timeframe delay (anticipate) Limitation exist to time_delay: **MUST** be less than (TF_l – Time necessary to download a configuration)

method	init_switch (unsigned char board_n, …
arguments	unsigned char board_n, // selects which board update unsigned char channel_n, // sets the number of switching channels unsigned short TF_l, // sets the # of TF steps (100ns per step) unsigned char TC_l, // sets the TF per TC counter unsigned char SEC_l // sets the TC per SEC counter
return type	Bool
description	sets the values to the FSM implemented into the FPGA. The master clock is 10 MHz, so the time unit is equal to 100ns. One second is divided into *Time Cycles* and each *Time Cycle* into *Time Frames*. *Time Frame Length* is a 16-bit counter (TF_l is an unsigned short). *Time Frames / Time Cycles* is a 8-bit counter (TC_l is an unsigned char). *Time Cycles / Second* is a 8-bit counter (SEC_l is an unsigned char). (TF_l / 10) * FC_l * SEC_l product **MUST** be equal to 1.000.000, i.e. 1 second!

method	reset_switch (unsigned char board_n)
arguments	unsigned char board_n // selects which board update

return type	Bool
description	sends an hardware reset pulse to Mindspeed chip

method	start_switch (unsigned char board_n)
arguments	unsigned char board_n // selects which board update
return type	Bool
description	starts the FSM implemented into the FPGA relative to the selected board

method	stop_switch ((unsigned char board_n)
arguments	unsigned char board_n // selects which board update
return type	Bool
description	stops the FSM implemented into the FPGA relative to the selected board

method	memory_write
arguments	unsigned char board_n, // selects which board update unsigned short mem_add, // sets the FPGA memory address to write to unsigned short reg_add, // value to write into memory (address) unsigned char reg_data // value to write into memory (data)
	Bool
description	writes the data into FPGA table memory mem_add **MUST** be less than 512

method	memory_read
arguments	unsigned char board_n, // selects which board update unsigned short mem_add, // sets the FPGA memory address to read from unsigned short *reg_add, // data read from memory (address) unsigned char *reg_data // data read from memory (data)
return type	Bool
description	reads the data from FPGA table memory mem_add **MUST** be less than 512

method	direct_write

arguments	unsigned char board_n, // selects which board update unsigned short address, // Mindspeed chip registers address to write to unsigned char data // Mindspeed chip register value to write
return type	Bool
description	writes data directly onto SPI bus (to Mindspeed chip registers)

method	direct_read
arguments	unsigned char board_n, // selects which board update unsigned short address, // Mindspeed chip registers address to read from unsigned char* data_out // Mindspeed chip register value read
return type	Bool
description	reads data directly from SPI bus (from Mindspeed chip registers)

method	check_status
arguments	unsigned char board_n, // selects which board update unsigned char* buffer // returns the status register
return type	Bool
description	returns the status register values Bit 0 ➔ PERROR 0 Bit 1 ➔ PERROR 1 Bit 2 ➔ LOS Bit 3 ➔ LOL Bit 4-7 ➔ 0

Here follows a simple example:

```
#include "geth_switch_class.h"
:
:
void main (void)
{
switch_control *test;
test = new switch_control;
test->reset_device();
test->set_PLL();
test->configure_FPGA();
:
```

```
:
test->set_clk_source(GPS_CLOCK);
test->set_time_delay(BOARD, 00 * 10);
test->reset_switch();
test->init_switch();
:
:
:
test->start_switch();
:
:
test->stop_switch();
delete (test);
}
```

IV. B. Mindspeed Switch Board

The M21156, designed for today's demanding telecom and datacom applications, is a low-power CMOS, high-speed 144 x 144 crosspoint switch with integrated CDRs, input equalization, and built-in system test features.

To simplify board design and increase system reliability, each input has a high-jitter tolerant, low-jitter generation CDR with internal-loop filter. All CDRs use a common, single-frequency, external reference clock (19.44 MHz) for internal calibration and acquisition. Each CDR operates independently at any data rate from 1.0 to 1.6 Gbps, or 2.0 to 3.2 Gbps. As a result, the device supports any combination of OC-48, OC-48 FEC, Fibre Channel (1x, 2x, 10x), InfiniBand, Gigabit Ethernet and 10 Gbps Ethernet, or other types of traffic.

To achieve the best possible signal, each CDR is preceded by a programmable input equalizer (IE), and each output includes Output Pre-emphasis (PE). The IE removes ISI jitter usually caused by PCB skin effect losses. The IE circuit opens the input data eye in applications where long PCB traces and cables are used.

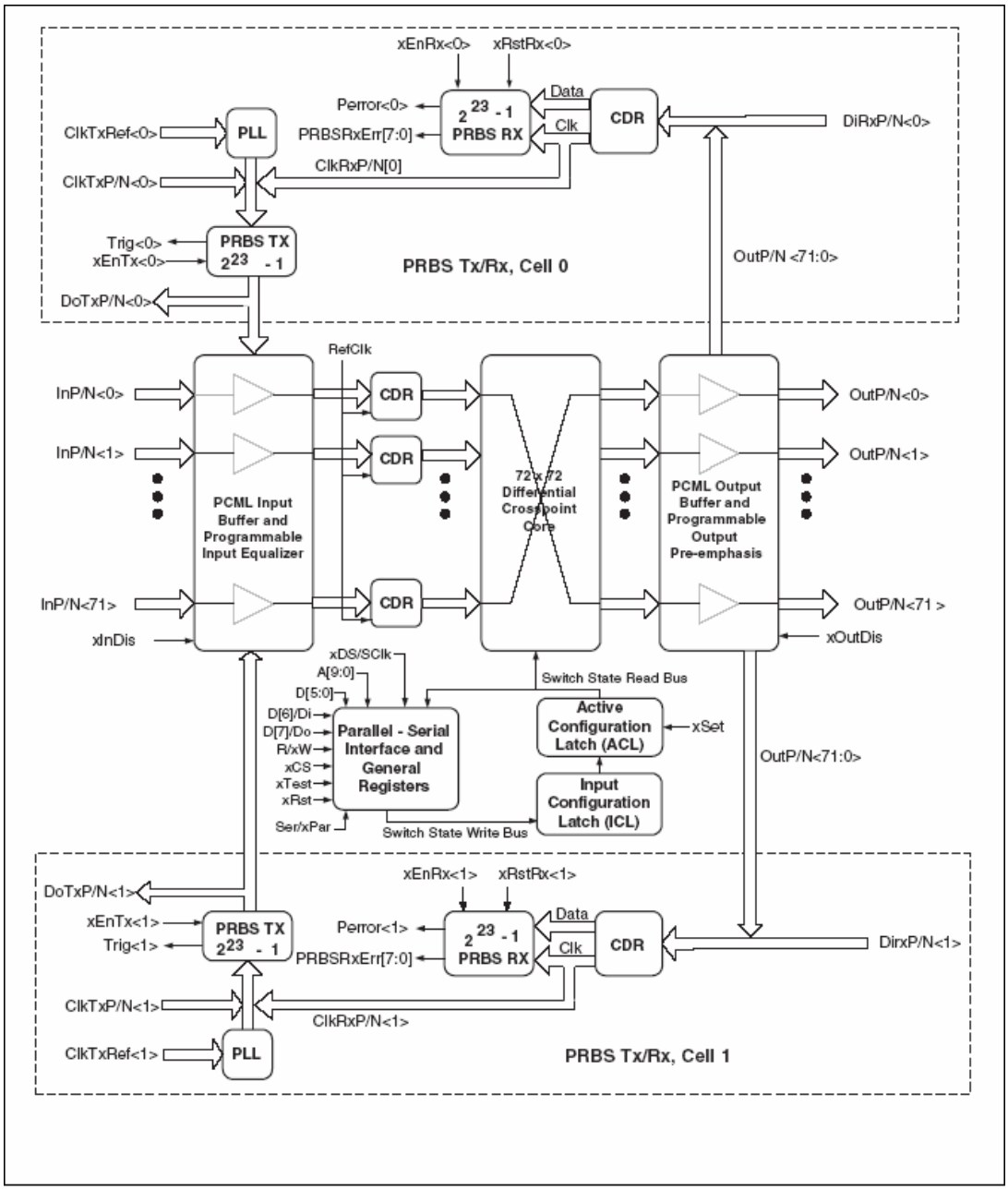
As a result, the device exceeds the SONET requirements with jitter tolerance of 0.65 UI, and jitter generation of less than 1.5 mUI (rms) or 8 mUI (peak-to-peak). The PE provides a boost of the high frequency content of the output signal, such that the data eye remains open after passing through a long interconnect of PCB traces and cables. There are two amplitude settings and two duration settings that can be selected on a global basis. Pre-emphasis can be enabled on a per-channel basis.

The device consumes as low as 18 W of power, typical, with all channels and CDRs operational. The PowerScaler^{TM} features offer dynamically scalable switch settings to further reduce power consumption. Each CDR can be independently bypassed and turned off if not in use. In addition, unused portions of the core can be automatically (SmartPower^{TM}) or manually turned off, without affecting the operation of the remaining channels.

Built-in system test features simplify design, verification, and production testing of the system. These features include a third generation PRBS transmitter/receiver, and the JitterMeter^{TM}. The switch includes a pair of on-board 223-1 pseudo-random bit sequence transmitters (PRBS TX) and receivers (PRBS RX). In addition, the JitterMeter feature allows the host controller to measure jitter of an incoming signal.

Three-stage switch fabrics with up to 10,368 x10,368 ports, carrying up to 33 Terabits per second of traffic, can be designed using this non-blocking switch, with multi-cast and broadcast abilities.

All inputs and outputs are differential PCML (positive current mode logic) with supply voltages ranging from 1.2 V to 2.5 V. The output levels are programmable at 500 mV, 900 mV, and 1200 mV. so, for applications that do not require an integrated CDR, the M21151 144 x 144 Crosspoint Switch with Input Equalization offers a very competitive solution.



This Mindspeed Switch Borad description document uses the following text styling conventions:

CDR signal names and terminal numbers are listed with initial caps denoting each functional name part, with its related signal polarity indicated by a upper case 'N' or 'P'. Thus, data input signal names are indicated, for example as: 'DinP' and 'DinN'.

A signal name and an associated CDR channel number (0 through 143) are indicated as DinP[n] and DinN[n] where 'n' is a channel number.

A register name is typed in all upper case preceded by the word register with each functional name part separated by an underscore, additionally, brackets group register bit numbers, and sub-function names are in initial caps such as for example only: 'register CONTROL_FUNCTION[5:4] Los_hyst'.

Note: There is no space between the name and the first bracket, and a space after the last bracket. The underscore is used to break up parts of a register name, as required.

To distinguish terminal names from internally generated signals the word ‘terminal’ is included in a reference to an input, output, or control, such as for example only: ‘the signal on terminal ABC controls function x', or ‘when the signal on terminal ABC = H function x is enabled'.

Switch State Register Concept

The M21131 switch-state controller uses a double-buffered register. The active configuration latch (ACL) holds the actual switch setting while the input configuration latch (ICL) holds either the actual switch setting or the next switch setting, depending on the mode of operation. The xSETmode register selects one of three modes of operation:

Defalut Mode—core configuration updated after every register write.

With xSETmode = 00h, the first mode is enabled and is the default mode after a reset. Consequently, the state of the switch changes with each write to a register determining the switch state. In the write mode, as soon as the signal on terminal xDS makes a low-to-high transition, the input channel specified by data for the output selected by the 10-bit address bus passes directly through the double buffer memory (ICL/ACL). As soon as the desired data pass through the ACL, the crosspoint core routes the selected input to the desired output to physically change the switch state. On the rising edge of xDS, this channel is stored (latched) into both the ICL and ACL.

XSET Mode—core configuration updated after hardware xSET command.

When register xSETmode = 10b the hardware xSET mode is enabled. In this mode, the desired switch state (which may contain one or more routing changes) is written first to the ICL, but the switch state does not change since the data is blocked from the ACL. With either the hardware or software xSET command, the contents of the ICL are transferred to the ACL, which physically changes the switch state in the switching core. This mode allows 1 to 72 channels to change at the same time. On the falling edge of the xSet signal, the ICL contents are passed to the ACL and the switch state changes. On the rising edge of the xSet signal, the switch state is latched.

XSET Mode—core configuration updated after software xSET command.

When register xSETmode = 01b the software xSet mode is selected, and the desired switch routing is written into the appropriate registers to update the ICL without affecting the ACL. Then, a write of any value to the xSETcmd register will update the ACL with the current contents of the ICL, and the switch state changes. The interface is configured into the parallel mode by forcing terminal Ser/xPar low.

IV. C. GUI Software architecture

There are two software layers implemented. The first layer is C++ classes and public functions developed from OpalKelly library. The second layer is graphical user interface (GUI), developed using VC6++, which takes all classes and functions from the first layer as libraries. The GUI basically is a set of dialog boxes which react to events (e.g. button press, check box…) and translate inputs typed by a user into parameters, variables and send them to corresponding functions layer-1.

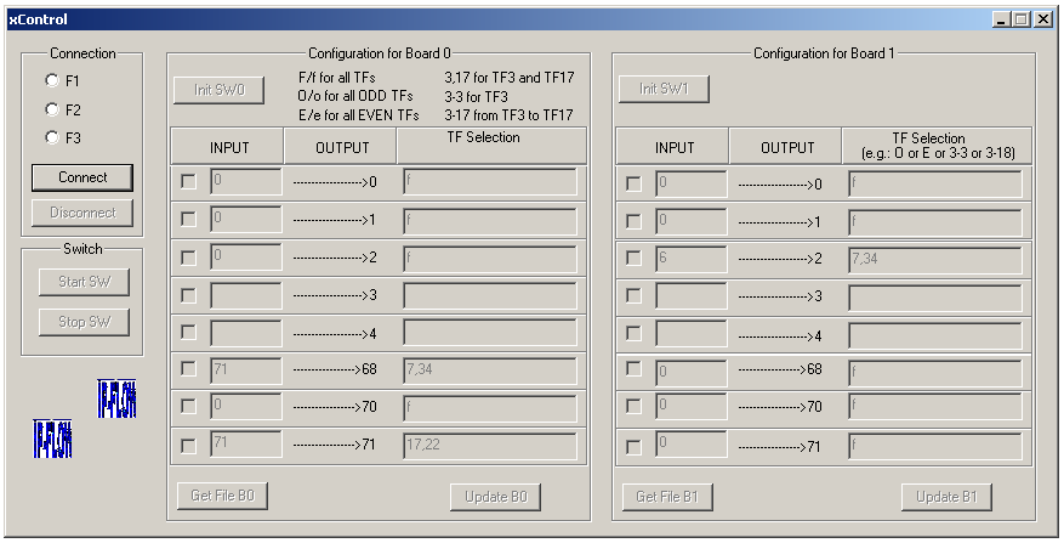


Figure 12. Graphical user interface.

Figure 12 shows the main dialog box. Hereafter we describe detail functionalities of each control group.

Connection group let a user initiate a connection to a selected FPGA. Since one single PC has more than one USB ports and each can be connected to a separate controller, a check box F1/F2/F3 identifies which controller a user wishes to communicate with. A controller is identified through its unique ID string which is pre-written in EEPROM on the XEM3001.

Once a controller is selected, a "Connect" button should be pushed. This event initiates a switch-controller instant and triggers functions for following jobs:

- Call reset_device().
- Call set_PLL().
- Download the configuration file (bit file) to configure the FPGA configure_FPGA(filename).
- Call set_time_delay(…) if required.[1]
- Call set_clock_source(…), can be GPS clock or internal clock.
- Enable other functional buttons.

"Disconnect" button is used to destroy the switch-controller instant, de-allocate any used memory and so on. It also triggers some function calls to set the Mindspeed chip back to default configuration (e.g. input/output voltage levels…).

Configuration for Board X group is used to initiate the default switching table, update switching table and get the current switching table written into a file. When a button "Init SWX" is pushed, a series of functions are called as following:

- Call to init_switch(…) to initiate the switching board X (i.e. X is either 0 or 1). Parameters transferred to this function are pre-defined in a file "parameters.h". this function is very important since it define the range of memory used in FPGA to store the switching table, which late is read by a Mindspeed switch. More detail can be found in FPGA implementation section.
- Write default switching table into FPGA memory. The default switching should be read from an existing file.
- Call direct_write(…) to set xSet to hardware mode (see details for this switching in Mindspeed description section).
- Enable other buttons for updating and getting configuration file.

---

[1] When f(…) stands for a call to a function which has parameter variables, detail of the function can be found in section… (describe layer 1 software).

The GUI is designed so that each control dialog box can work with two Mindspeed switching boards (accordingly two separate memories are initiated in the FPGA). Once the dialog box works with two switching boards, both of "Init Board0" and "Init Board1" should be activated before further steps. When a switch instant has been initiated and the default switching table has been written into FPGA memory, the FSM can start through pushing "Start SW". We will be back to the start and stop of FSM later.

Now we describe how a user can update the switching table. This action can be done through manipulations of checkboxes and edit boxes of the sub-group "INPUT" and those of sub-group "TF selection". To avoid switching conflict at any output, the output value is fixed so that at any time, only a single input can be connected to an output. To update a switching path, a user must define which input should be connected to which output in which time-frame. A user check the corresponding checkbox, edit the input value and the time-frame values. Error messages will appear if a user put a wrong value (e.g. out of range if input value is minus or larger than 143…). After editing all necessary values, a user pushes "Update BX", which does:

- Call stop_switch(…) to stop the FSM if it is running in order to avoid conflicts (write from PC to FPGA memory and read to Mindspeed memory at the same time, same memory location).
- Call memory_write(…) to update desired memory locations following updated values from edit boxes.
- Call start_switch(…) to restart the FSM when updating finishes.

Many formats of updating can be followed:

- If input value of an "TF selection" edit box is F, the switching path to the corresponding output will be static (i.e. always connect the selected input to that output, no change according to time-frame).
- If input value of an "TF selection" edit box is O, the switching path will be setup for every ODD time-frames (i.e. time-frame 1, 3, 5, 7, …).
- If input value of an "TF selection" edit box is E, the switching path will be setup for every EVEN time-frames (i.e. time-frame 2, 4, 6, 8, …).
- If a user wishes to setup the switching path for a single time-frame, say time-frame 7, the input value must be 7-7.

- If a user wishes to setup the switching path for a range of continuous time-frames, say from time-frame 7 to time-frame 11, the input value must be 7-11.
- If a user wishes to setup the switching path for two discrete time-frames, say time frame 7 and time-frame 11, the input value must be 7,11.

The button "Get File BX" is pushed, it does:

- Call stop_switch(…) to stop the FSM if it is running.
- Call memory_read(…) to read values from FPGA memory, output them into a structrured text file named "SWconfigBXFY.txt". (X is either 0 or 1, Y is either 1 or 2 or 3).
- Call start_switch(…) to start the FSM again.

Switch group includes two buttons "Start SW" and "Stop SW". When a user pushes "Start SW", the following calls are triggered:

- Call direct_write(…) and write 0x03 into 0x00BA of Mindspeed chip to enable I/O (500mV).
- Call start_switch(…) to start the FSM.
- Enable some control items (Get File BX, Update…).

Once a "Stop SW" button is pushed, it simply stop the FSM.

IV.D Detailed programming procedures

Detail codes are more than 2000 lines of VC6++. See attached zip file "X-controller.zip" for details.

V. Switching Fabric

V.A. Topological design

The switching fabric consists of 3 MindSpeed single switching matrix boards (board 0, board 1, board 2) connected together. The block diagram is showed in Figure 12. Switch#1 consists of the combination of two of them are interconnected by means of electrical coaxial cables. The third board is used for the second switch (switch #2). The link between swtch#1 and switch#2 is optical fiber. The packets assigned by TDP router to the corresponding time frame (which is a parameter to be set up in advance) are to be sent trough the optical fiber to transceiver 1 which is connected to switch#1. On the figure below you can see that data

coming from transceiver 1 go to input 71. Input 0 receives the idle pattern. Issues related to the idle pattern are covered in the next chapter. The data after entering switch#1, are split into two streams by board 0, one stream for each destination: one stream, named pipe 1 (assigned to 7, 14 TFs), goes to Output 68, the second stream, pipe 2 (12, 17 TFs), goes to Output 71. Outputs 68 and 71 of board 0 are connected electrically to Inputs 6 and 70 of board 1 respectively. On board 1 both streams join together and go to Output 2 of board 1. Then the data are transmitted to the transceiver and then further via optical link, that can be either single mode or multimode. After passing through the optical link the transmitted data go through transceiver 6 to Input 6 of board 2 and then split again into two streams which are switched to Output 2, pipe 2 (12, 17 TFs) and to Output 70, pipe 1 (7,14 TFs). Then both streams are transmitted by transceiver 2 and 3 and two separated optical links to two destinations: destination1 and destination 2.

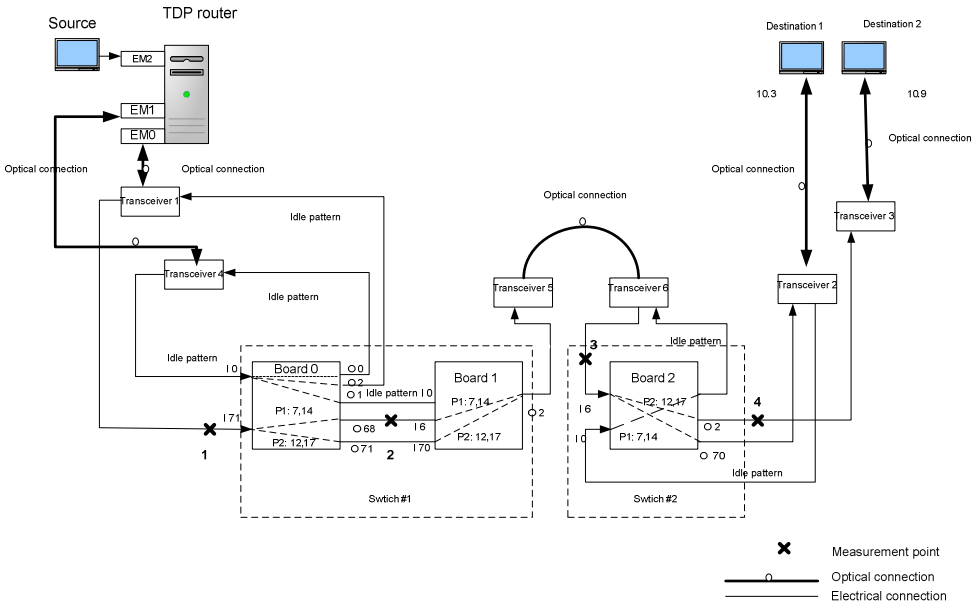


Figure 13. Block diagram

V. B Idle pattern and bit synchronization

There are five level of synchronization in our data transport and switching architecture. The first level is bit synchronization, then we have the 8/10 modulation and byte synchronization, then Ethernet framing, TCP framing and GPS/UTC time framing. All levels depend on the previous one for successfully lock onto the data stream.

When switching from one stream to the other all levels must be successfully locked by the client to allow the correct reception of the data stream.

Bit synchronization is the first level and it is necessary to reach locking in few microseconds after switching. After locking, data are received correctly at the bit level.

Gigabit Ethernet resorts to 8/10 modulation for clock and data recovery (CDR). With the modulation the spectrum of the signal is compatible with optical transmission and data regeneration with an acceptable increase in bandwidth , in fact the effective data rata of Gigabit Ethernet after modulation is 1.25 Gb/s. The 8/10 modulation make the use of PLL mandatory for CDR.

The typical locking time of CDR's PLL is of the order of milliseconds, if we refer to what chip manufactures specify in the manuals of Gb Ethernet controllers.

We may indeed observe that our switch do indeed switches among different streams which are almost running at the same clock frequency, we may therefore expect that re-locking time should be reduced; how much is the question.

Our measurements show that re-locking time after switching is of the order of few microseconds with the Intel Pro Gb Ethernet controller. Our findings are consistent with the literature, for instance the Lattice Semiconductor ORT82G5 3 Gb/s SERDES and our system exhibit comparable re-locking time. The Lattice Semiconductor ORT82G5 3 Gb/s SERDES is characterized by sub-microsecond locking time after signal detection (see: TN1025 by Lattice Semiconductor, April 2003). In fact the ORT82G5 switches its input to a local signal with an almost "on frequency" data rate when no useful signal is applied. The ORT82G switches periodically to its physical input to look for useful signal, being the PLL almost at the expected locking frequency of the input data flow. If no acceptable signal is present, the receiver input idles on the local signal. Similarly, but none identically, with our switching strategy, Gb Ethernet SERDES switches deterministically to the required input when data are expected to come. The effect is the same: locking time is drastically reduced.

Idle pattern is transmitted through the whole system in the following way:

The Ethernet card of TDP router is connected to transceiver 4 in order to deliver idle pattern to switch#1. Idle pattern is transmitted via optical link to Input0 of board 0 and then switched to Outputs 0, 1 and 2. Output 0 is connected back to transceiver 4 and Output 2 is connected back to transceiver 1. Output 1 is connected to board 1 of switch#1 with coaxial cable. Switch #2 receives the idle pattern from the receiving side: transceiver 2 is connected to destination 1/pc and the idle pattern is transmitted to Input 0 and then switched to Output 0 and further goes to transceiver 6.

V. C. Measurements

The quality of digital signals transferred over analog media can be evaluated by standard procedures. Specifically, international standards must be applied to any specific kind of data transmission. Our laboratory oscilloscope is complete with Gigabit Ethernet 1000 base SX-LX mask test automatic procedure. Data are digitized in real time at 20 Gigasample/s or up to 1000 Gs/s equivalent repetitive sampling. A "golden PLL", as required by the standard, is then employed for mask testing. Our system, composed by four electro-optical conversions and three switching layers, is compliant to Gb Ethernet international standards regarding mask testing. Mask tests are presented, reporting the quality of the signal along the signal path.

Summing up everything mentioned above the eye diagrams were obtained at different measurement points according to Ethernet 1000-baseSX/LX standards. The measurements were taken at the 4 points which are marked on Figure 13. Multi mode fibre was used for the optical link. Eye patterns are showed in figures 14-19. At-points 3 and 4 measurements were repeated using a roll of 25 km single mode fibre.

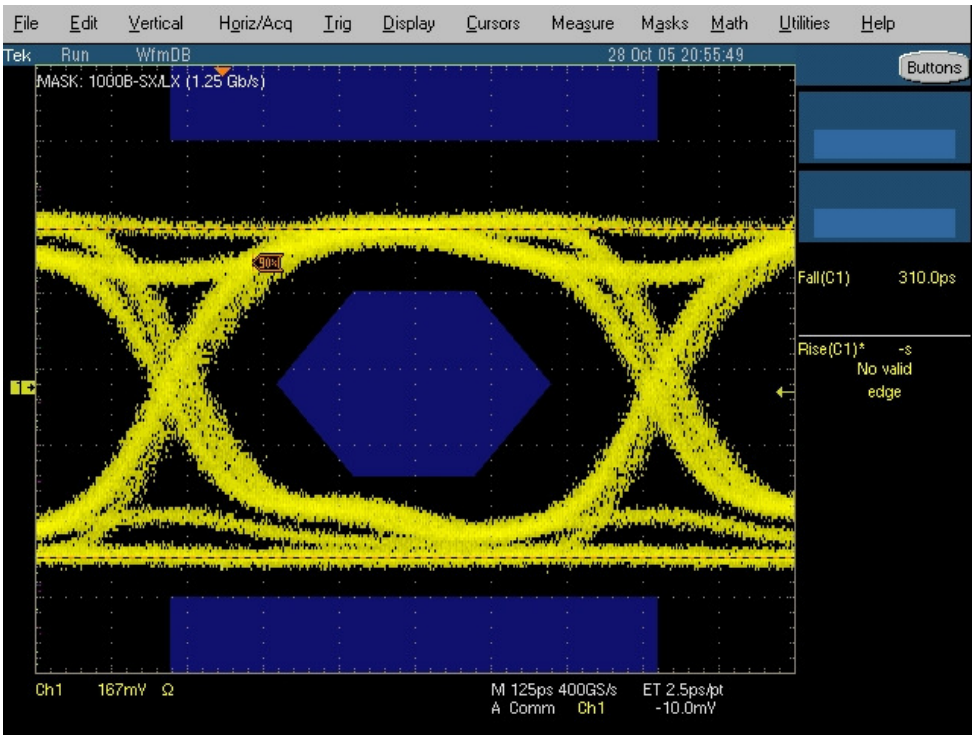


Figure 14. Eye Pattern with Mask: 1000B – SX/LX (1.25Gb/s) at location 1

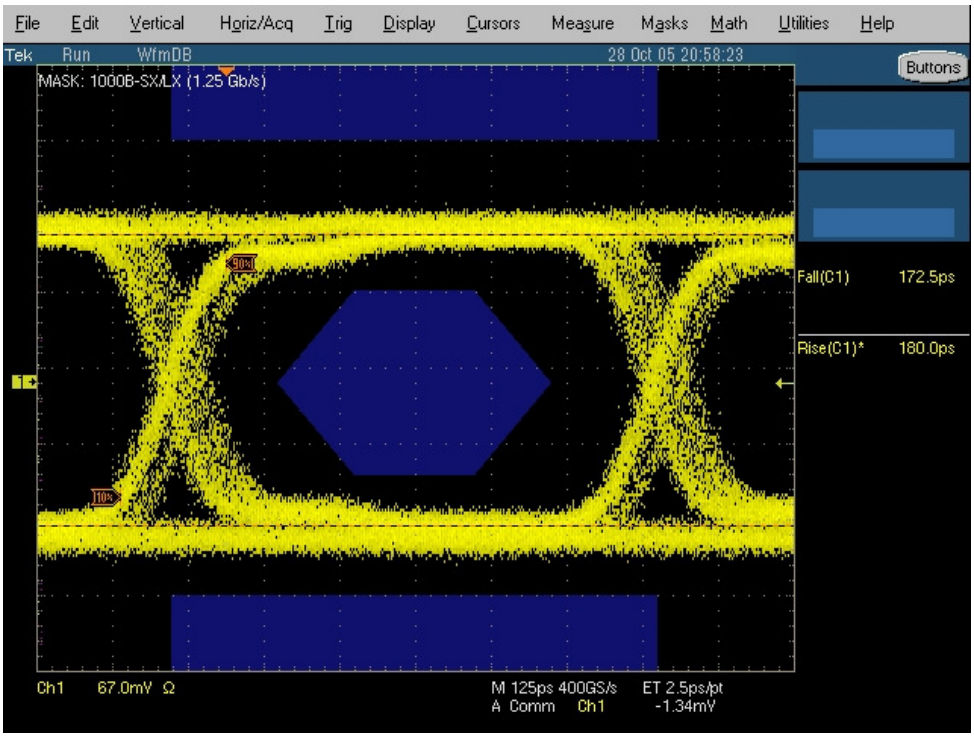


Figure 15. Eye Pattern with Mask: 1000B – SX/LX (1.25Gb/s) at location 2

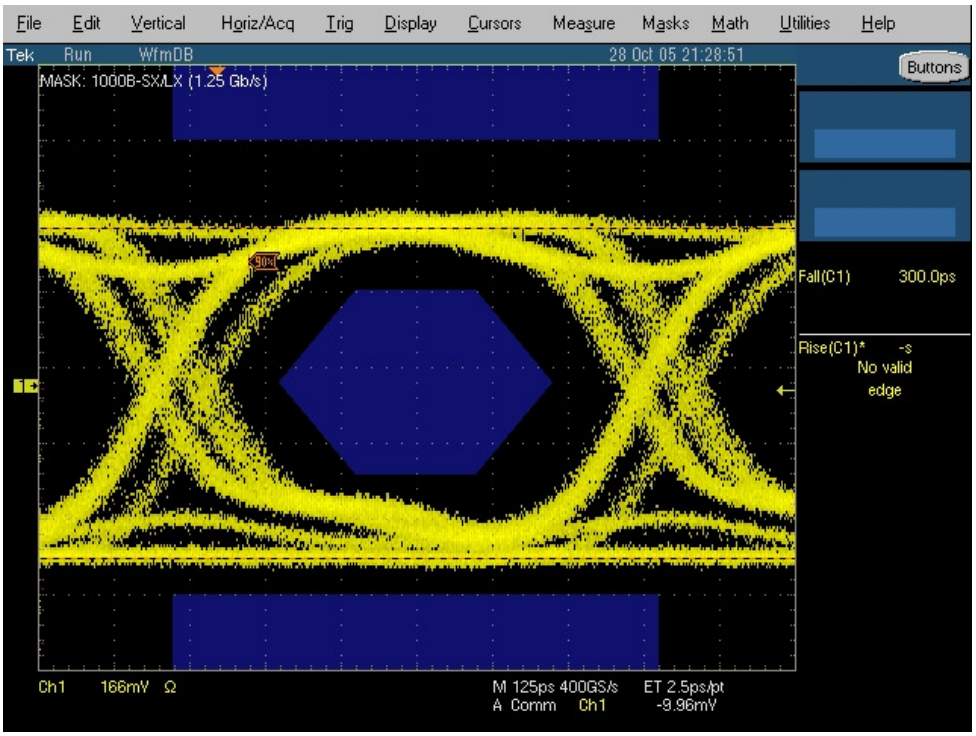


Figure 16. Eye Pattern with Mask: 1000B – SX/LX (1.25Gb/s) at location 3

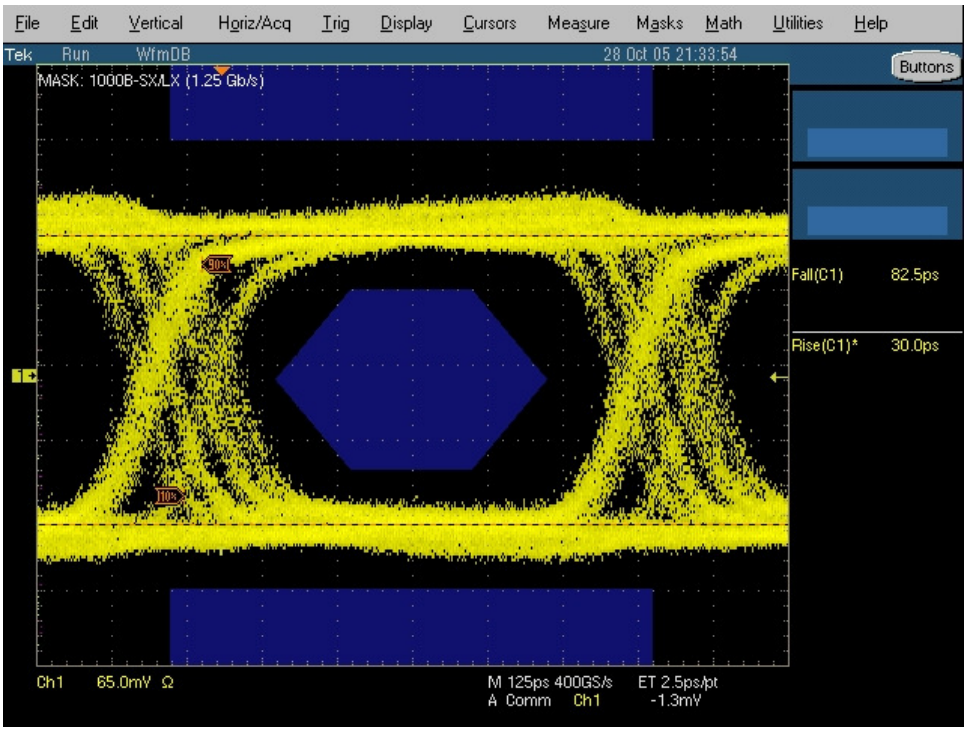


Figure 17. Eye Pattern with Mask: 1000B – SX/LX (1.25Gb/s) at location 4

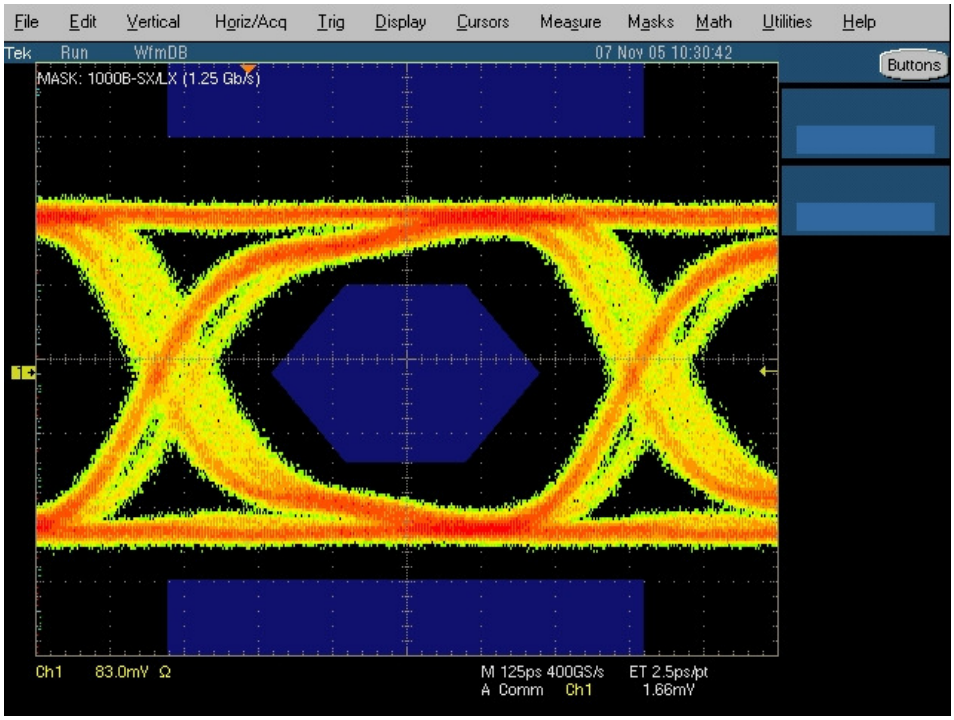


Figure 18. Eye Pattern with Mask: 1000B – SX/LX (1.25Gb/s) at location 3 with single mode Fibre

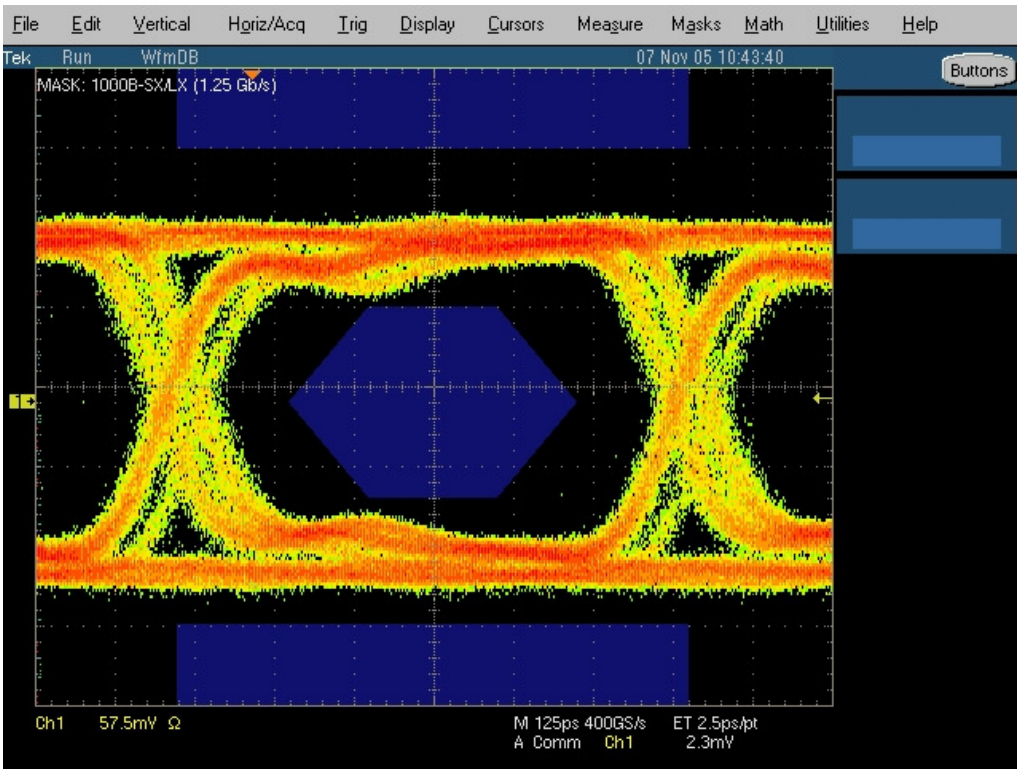


Figure 19. Eye Pattern with Mask: 1000B – SX/LX (1.25Gb/s) at location 4

VI. Streaming Media Demonstration

The performance of a network can be evaluated quantitatively and qualitatively. The network performance evaluation tools, that are able to send and receive packets, are used to evaluate distinctive parameters of networks i.e. mean throughput, peak throughput, latency, RTT … These tools produce measurements quantitative parameters of the network, but to evaluate qualitative performance of the same network, one can use a video streaming, audio and or text flows.

The video streaming is a technology that allows the diffusion of audio and video service via Internet without downloading video or audio file, but one can see or hear them in real time.

VI.A. Media streaming demonstration setup

In a generic configuration to show the performance of the network a set up as shown in Figure X can be used. The test bed can stream more than one source at a time and, obviously, more than one streaming receivers. The demonstration configuration is made up of two streaming source on one pc and two streaming receivers on two different PCs.

Some streaming sources are able to send streams to specific client identified by an IP address by using both HTTP and UDP protocol. The client in order to see video-streaming sent by a server, must know the IP port that must be listened if the sending is in multicast format or the IP port and address if the sending is in unicast format. “Multicast” [23] is one way to use the IP protocol: one computer sends traffic to a special IP address (a pool is defined in the standard) and the clients can choose to register the source IP address and receive the traffic. The advantage is that in this way the source sends only one flow, but the disadvantage is that the network can be flooded by unused packets. Multicast is not always supported by tool and by switch or router: in that case unicast communication is used.

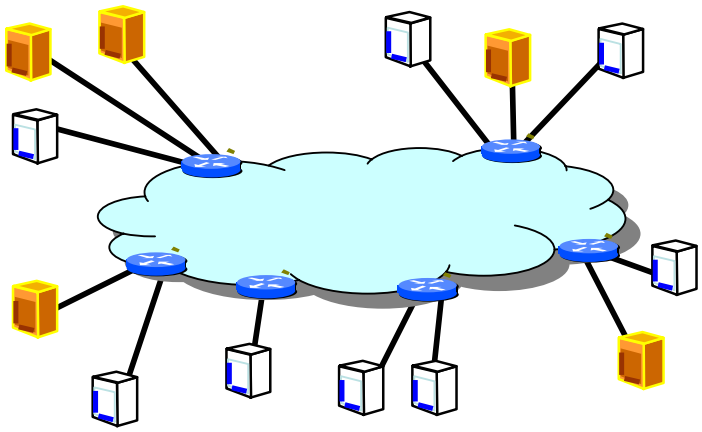


Figure 20. Structure of a generic client-server network for streaming.

In this demonstration the specific configuration is shown in Figure 20 and can be described as:

- Server 1 (S1) has two streaming sources (V1 and V2). There are two instances of the same tool.
- Client 1 (C1) receives the video streaming V1.
- Client 2 (C2) receives the video streaming V2.

The server can define the protocol to use, the destination IP address and port of the client. The clients can also define the protocol to use and the IP port that receives the streaming flows. In this case the communications are in unicast format.

In this test-bed the streaming server and the video client are the same program: VLC Media Player [24]. The same tool can be used as a local media player and implements also some features to sending streaming to a remote user.

VI. B. Detailed media-player programming

VLC (VideoLAN Client) Media Player is a portable multimedia player that can support various audio and video formats: MPEG-1, MPEG-2, MPEG-4, DVD and VCDs formats. Besides it can support different streaming protocol. The same player can be used also as server to stream audio or video and it can support unicast or multicast protocol for both IPv4 and IPv6 network. Our test is based on DVD streaming video because in this way the quality of the video is high and the network can be tested with a high load.

VLC Media Player has features that depend on the Operating System. For Windows platform, it can support several input format (UDP, HTTP, DivX, DVD, audio CD, webcam …) with different formats (MPEG-1, -2, -3, -4, Ogg, WAV, RAW …) and codec for audio and video. The output format for streaming video can be UDP, HTTP or MMSH and the audio and video are in MPEG format. Further information about input and output format and codec can be found in the VLC web site [24]. The streaming server can send one source to different clients using different protocol and the same server PC can start different instances of VLC Media Player to send more than one sources to clients with the same protocol.

VI. C. Streaming over wireless

The previous set up shown before is using wired connections, but a similar set up can be used for wireless client, as showed in the Figure 21.

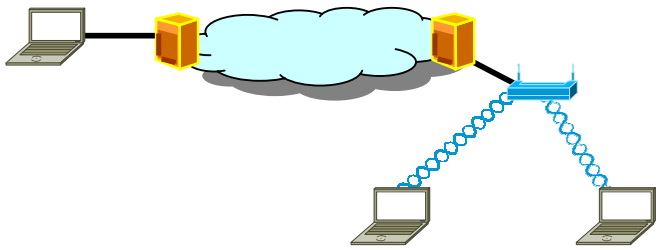


Figure 21. Set up for Wireless streaming transmission

In this configuration the border router (Network Interface) is directly connected to an AP which sends the packet to the wireless network. The mobile users must connect to the AP using the broadcasted SSID. For easiness (but without loss of generality) the clients have a static IP addresses and there are no MAC allowed list to restrict the accesses.

VII. Why TDS is Ultra Scalable

To cope with the present Internet growth rate, an IP routing capacity of multi Tb/s is required. Existing architectures can handle a maximum load of about only one Tb/s (in a single chassis), however much higher switching capacities are required. To get beyond the limitations of conventional electrical switch technologies, alternative architectures based on optical interconnects are proposed (see, for example, [1][2]). Optical interconnects allow, at least in principle, any desired interconnection topology without introducing crosstalk, thus minimizing noise sources. On the other hand, electrical switching with electrical interconnects cannot support the huge information flow from optical networks, while suffering from increased wire resistance, residual wire capacitance and inter-wire crosstalk as the length and/or the density of the electrical interconnections increases.

VII. A. Switching Fabric Design

In order to maximize the switching fabric scalability it is necessary to minimize the switching fabric complexity. The lowest complexity fabric are multistage Banyan interconnection networks, with switching complexity of O(a**N***lg*$_a$*N*), where *N* is the number of input/output and *a* is the size each switching block.

Switching elements	Multistage $a*N*\lg_a N$	Crossbar N^2	
For N=256, a=4	4K	64K	(factor of 16)
For N=1024, a=4	20K	1,000K	(factor of 50)

Figure Figure 22, is a switching complexity comparison between a multistage Banyan and a crossbar.

Switching elements	Multistage $a*N*\lg_a N$	Crossbar N^2	
For N=256, a=4	4K	64K	(factor of 16)
For N=1024, a=4	20K	1,000K	(factor of 50)

Figure 22. Switching fabric complexity

The main disadvantage of Banyans is *space blocking*, which means that a connection between an available input and an available output may not be possible because there is no available pass (route) through the switch interconnection network.

One of the interesting properties TDS is that it significantly reduces the blocking phenomenon of Banyan based switching fabrics. Intuitively, In order to connect an available input to an available output there is another degree of freedom in the time domain. Namely, it is possible to select one of the *K* time frames (TFs), which mean that there are *K* possible choices. Figure shows the blocking of a Banyan as a

function of the number time frames, *K*, per time cycles: *K* = 1, 4, 16, 32, 64 and 1000. Clearly, as the number of time frames per time cycle increases the blocking probability decreases (see a more detailed performance study in [17]).

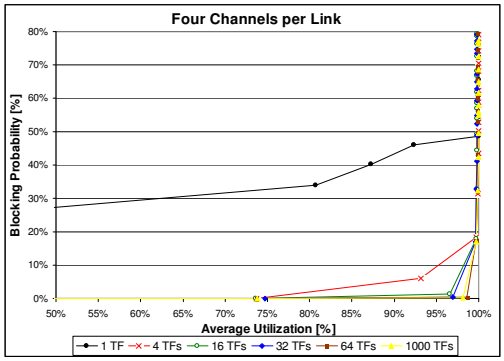


Figure 23. Blocking probability

TDS enables the construction of Banyan based fabrics, which are the most scalable switch design (in the next subsection it will be shown that TDS also minimizes buffer requirements).

Figure 24 and 25 are two design examples of Banyan based fabrics with switching capacities of 10Tb/s and 40Tb/s, respectively. The two designs are based on commercially available electrical switching blocks from Mindspeed Inc. that are interconnected either electronically or optically.

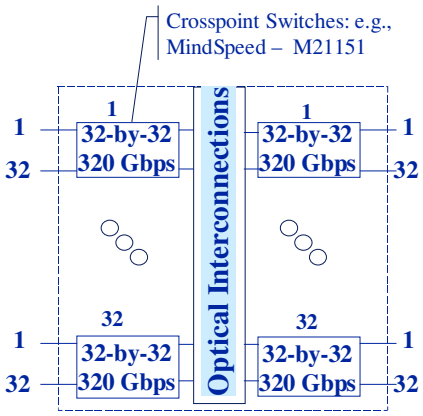


Figure 24. 10Tb/s switching fabric

In comparison, optical switches with below 1μs switching have lower capacity, larger physical size and are much more expansive. The main challenge in constructing the large switching fabrics shown in figures 24 and 25 is interconnection. Electronic interconnection will not scale given their inherent problems. Consequently, the most promising approach is a hybrid opto-electronic switching fabric. As discussed in the next section, electronic may be also the most suitable approach for buffering.

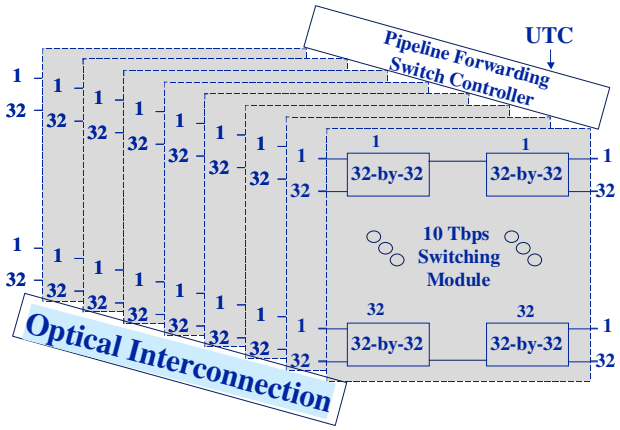


Figure 25. 40Tb/s switching fabric

VII.B Ultra Scalable Buffer Requirements

TDS enables the switching of IP packets in time frames (TFs) without decoding the headers of each packet. Namely, TDS eliminates the need for header processing. TDS is based on the setting of a switching schedule of IP packets in TFs along a predefined route in the network.

Clearly, if the IP packets in TFs arrive to the switch at the exact time no buffers are required. However, since the delay between TDS switches may not be an integer number of TFs it is necessary to align the incoming TFs on the optical links with the UTC TFs. Alignment consists in aligning the beginning and end of each time frame on each optical channel with the beginning and end of the UTC time frames. Realizing the alignment operation requires some buffers.

Such buffer can be realized in the optical domain using programmable optical delay line, as shown, for example, in Figure 26. Skipping the detailed analysis of such programmable delay line, it is based on a non-trivial dynamic optical switch, which may introduce significant amount of power loss. In addition, such programmable optical delay lines are expensive.

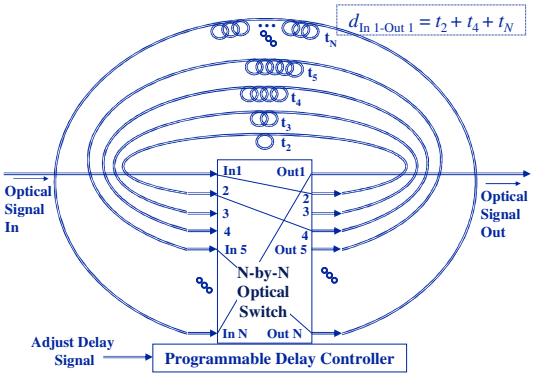


Figure 26. Optical alignment

In comparison, electrical alignment, as shown in Figure 27, is simple with no power loss and low cost. Obviously, the electrical alignment matches the use of electronic switching. The electrical alignment is based on buffers implemented with electronic memory. An electrical alignment consists of a number of queues (usually 3) in which arriving IP packets are stored and IP packets to be switched are retrieved.

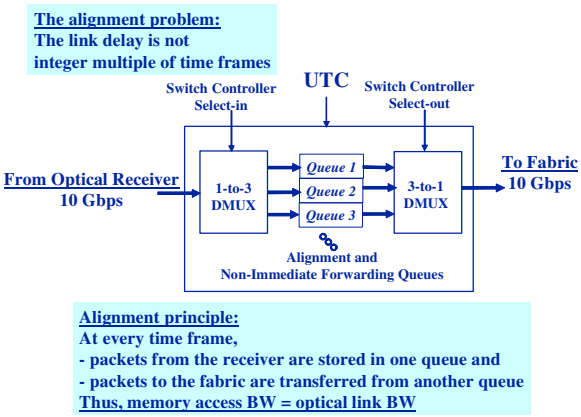


Figure 27. Architecture of electrical alignment

Note that the switch controller, which controls the electrical switching fabric, is also responsible for changing the configuration of the electrical alignment responsive to the current value of UTC. The alignment buffering requirements of the electronic alignment, shown in, are three queues each contains the bits transmitted during one time frame. For example, for 10Gb/s optical channel and 8μs TF is only 3*80,000/8=30KB. Note the structure of electrical alignment buffer, shown in Figure 27, can be easily extended to support non-immediate forwarding (as discussed in Section I.C).

In essence, the switch controller, the switching fabric and the electronic alignment buffers are all what is needed for implementing ultra scalable TDS switch.

VIII. Discussion

As the Internet exponential traffic growth continuous, in the foreseeable future, it will be due more and more to (all IP) streaming media applications. Streaming media applications will “fill-up” the optical “pipes.” One of the main remaining challenges is how to efficiently switch this huge amount of IP traffic. UTC-based TDS provides the necessary switching solution, which has following desired requirements for streaming applications:

- Switch and forward IP packets as a whole (i.e., IP packets are remained intact at all times).
- Provisioning granularity from 1Mb/s to full optical channel capacity.

- Minimum delay with constant jitter and no packet loss.
- Multicast and broadcast of IP packets with any allocated capacity with the above mentioned properties.

Furthermore, UTC-based TDS facilitates the most scalable electronic design with optimal switch complexity (O[a**N***lg*$_a$*N*]), minimum electronic buffers (1-3 time frames) and optimal switching speedup of one. However, in order to actually construct ultra scalable switches it is necessary to overcome the switching modules interconnection problem.

Since optical transmission has superior interconnection properties over electric wires, it has been proposed to use optical interconnects. However, such an approach is novel and requires further research and development. The optical interconnection objective of high-capacity electronic switching modules is:

> "*maximizing the density of optical interconnections, while minimizing power dissipation for a given distance bound among electronic switching modules (each consists of one or more electronic switching devices).*"

A UTC-based TDS switch prototype, shown in **Error! Reference source not found.**, using Mindspeed cross-point devices (M21151) is being implemented at the University of Trento. Each of the M21151 cross-point is a single electronic chip with 144 inputs and outputs, each input/output has capacity of 3.2Gb/s or total raw capacity of 460Gb/s. The cost of such cross-point chip is around 500 USD or 1 dollar per (raw) Gb/s! Moreover, the switching time for changing the input/output permutation is less than 10ns. The switch prototype is using UTC-base time-driven priority software developed at the Politecnico di Torino.

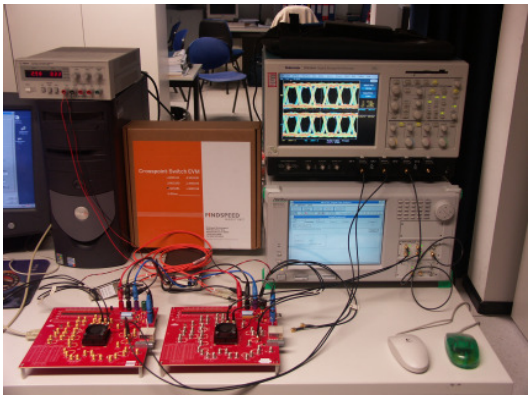


Figure 28. Early TDS prototype

Although UTC-based TDS is optimally designed to support streaming media applications it is straight forward how to combine its operation with "best effort" (non-scheduled) IP/MPLS, as shown in Figure 29.

This is done by adding an IP packet filter before the alignment sub-system and then diverting non-scheduled IP packets to IP/MPLS switching module.

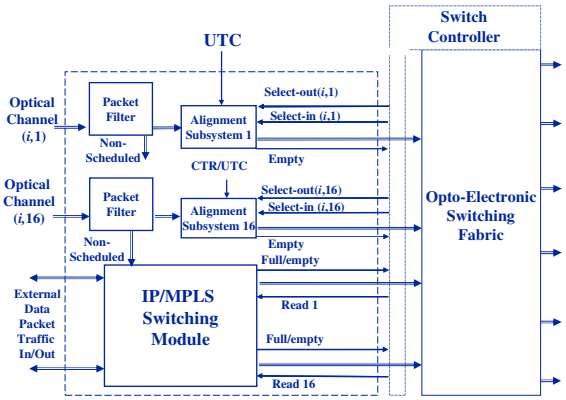


Figure 29. Hybrid IP/MPLS and TDS system

References

[1] W. Gibbs, "Computing at the Speed of Light" Scientific American, Nov. 2004, pp. 80-87.

[2] "Silicon Photonics," Edited by L. Pavesi and D. J. Lockwood. Springer, 2004.

[3] M. Baldi, Yoram Ofek, "Fractional Lambda Switching," Proc. of ICC 2002, New York, vol.5, pp 2692 – 2696.

[4] A. Pattavina, M. Bonomi, Y. Ofek, "Performance evaluation of time driven switching for flexible bandwidth provisioning in WDM networks," Proc. of Globecom 2004, Dallas, Texas, vol. 3, pp 1930-1935.

[5] Mario Baldi and Yoram Ofek, "Fractional Lambda Switching - Principles of Operation and Performance Issues", *SIMULATION: Transactions of The Society for Modeling and Simulation International*, Vol. 80, No. 10, Oct. 2004, pp. 527-544.

[6] Donato Grieco, Achille Pattavina and Yoram Ofek, "Fractional Lambda Switching for Flexible Bandwidth Provisioning in WDM Networks: Principles and Performance", *Photonic Network Communications*, Issue: Volume 9, Number 3, Date: May 2005, Pages: 281 – 296.

[7] C.Qiao and M.Yoo, "Optical burst switching (OBS) a new paradigm for an optical internet," *Journal of High Speed Networks*, vol. 8, no. 1, pp 69–84, Jan 1999.

[8] Y. Xiong, M. Vandenhoute, and H. C. Cankaya, "Control architecture in optical burst switched WDM networks," *IEEE Journal on Selected Areas of Communication*, vol. 18, no. 10, October 2000, pp 1838–1851.

[9] K. Dolzer, C. Gauger, J.Spaeth, and S. Bodamer, "Evaluation of reservation mechanisms for optical burst switching," *International Journal of Electronics and Communications*, vol. 55, no. 1, 2001, pp 18–26.

[10] M. Baldi and Y. Ofek, "End-to-end delay of videoconferencing over packet switched networks," IEEE/ACM Transactions on Networking, Vol. 8, No. 4, Aug. 2000, pp. 479-492.

[11] C-S. Li, Y. Ofek, A. Segall and K. Sohraby, "Pseudo-isochronous cell forwarding," Computer Networks and ISDN Systems, 30:2359-2372, 1998.

[12] M. Baldi, Y. Ofek and B. Yener, "Adaptive group multicast with time-driven priority," IEEE/ACM Transactions on Networking, Vol. 8, No.1, Feb. 2000, pp. 31-43.

[13] D. K. Hunter and D. G. Smith, "New architectures for optical TDM switching," IEEE/OSA Journal of Lightwave Technology, vol. 11, no. 3, pp. 495-511, Mar. 1993.

[14] I. P. Kaminow et al., "A wideband all-optical WDM network," IEEE Journal on Selected Areas in Communications, vol. 14, no. 5, pp. 780-799, June 1996.

[15] P. Gambini et al., "Transparent optical packet switching: network architecture and demonstrators in the KEOPS project," IEEE Journal on Selected Areas in Communications, vol. 16, no. 7, pp. 1245-1257, Sept. 1998.

[16] Nen-Fu Huang, Guan-Hsiung Liaw, and Chuan-Pwu Wang, "A novel all-optical transport network with time-shared wavelength channels," IEEE Journal on Selected Areas in Communications, vol. 18, no. 10, pp. 1863-1875, Oct. 2000.

[17] M. Baldi and Y. Ofek, "Fractional Lambda Switching - Principles of Operation and Performance Issues," *SIMULATION: Transactions of The Society for Modeling and Simulation International*, Vol. 80, No. 10, Oct. 2004, pp. 527-544.

[18] C.-S. Li, Y. Ofek, and M. Yung, "Time-driven priority flow control for real-time heterogeneous internetworking," in Proc. IEEE (INFOCOM' 96), vol. 1, Mar. 24–28, 1996, pp. 189–197.

[19] “Dummynet.” [Online]. Available: http://info.iet.unipi.it/_luigi/ip dummynet.

[20] G. R. Wright and W. R. Stevens, TCP/IP Illustrated, Volume 2: The Implementation. Addison-Wesley, 1995.

[21] “ALTQ.” [Online]. Available: http://www.csl.sony.co.jp/_kjc/software.html.

[22] K. Cho, “A framework for alternate queuing: Towards traffic management by PC-UNIX based routers,” in Proc. USENIX (Annual Technical Conference’98), New Orleans, LA, June 1998.

[23] RFC1112, Host Extension for IP Multicast, August 1989

[24] VLC Media Player, http://videolan.org/vlc