# D4.1 “Implementation of OPST Metro Ring-OBST Metro Mesh interconnection Node and Mesh bypass Metro Node”

| | | |
|---|---|---|
| Status and Version: | Final, 1.1 | |
| Date of issue: | 11.9.2011 | |
| Distribution: | Project Internal | |
| Author(s): | Name | Partner |
| Yixuan | Qin | UESSEX |
| Yan | Yan | UESSEX |
| Bijan | Rahimzadeh Rofoee | UESSEX |
| Georgios | Zervas | UESSEX |
| Dimitra | Simeonidou | UESSEX |

## Table of Contents

## 0 Executive Summary

This document reports on the implementation of MAINS TSON[1] metro node[2]. It is based on the design of TSON interconnection and bypass nodes (D2.3 "OBST Mesh Metro Network Control Plane and GMPLS Interworking") driven by the requirements set in D1.1 "Definition of network scenarios, drivers and requirements for metro-regional architectures combining optical transport and network resident IT resources".

In Section 1 the objectives and the scope of the document are stated. The reference documents and the relevant terminology used in the document are also listed.

In Section 2, the overall TSON metro node architecture with and without flexible frequency switching (Gridless) support in terms of node controller, data plane, and TSON work flow are given. The required hardware and interfaces between layer 1 and layer 2, layer 2 and upper layer controller are also described.

Section 3 is the major part of this document; it focuses on the Layer 2 design, implementation and measurement of TSON metro node. It presents the detailed hardware requirement and block design requirement. Following this, it describes implementation strategy. Then, the detailed node architecture and the implementation in terms of hardware components, IP components, and user logic components are reported. The last part of the section produces the simulation and implementation results.

Section 0 explains the Layer 1 implementation of the TSON metro node. It includes the optical data plane switch node configuration built with PLZT fast switches for time-slice switching and the corresponding optical components. Furthermore, Another flavour of Layer1 TSON optical node, supports flexible frequency switching by incorporating spectrum selective switch and optical backplane besides the TSON timeslice switching.

Section 5 introduces a novel software/hardware defined network (SHINE) solution for TSON metro node to enable flexibility and programmability. This section firstly defines the problem, which SHINE tries to solve. Then both high level design and detailed design are given. In the meantime, SHINE methodology and working flow are described. Finally the SHINE enabled TSON metro node architecture, implementation, and test results are given.

Finally, the document's conclusions are presented in Section 6.

[1] Since D2.3, OBST had been referred as TSON, and will be used throughout this deliverable.

[2] TSON metro node is the superset node, which covers the functions of both OPST-OBST interconnection node and OBST bypass node. Details can be found at Subsection 1.1.

# 1 Introduction

## 1.1 Purpose and Scope

This deliverable provides the OPST Metro Ring-TSON (OBST) Metro Mesh interconnection Node and TSON (OBST) Mesh bypass Metro Node final design and implementation details.

As a universal superset node implementation approach, the TSON metro node is deployed equally for both TSON interconnection node and TSON bypass node. It consists for two main functions; the L2 TSON functions (Ethernet to TSON frame conversion, aggregation, scheduling, etc.) needed for the interconnection node and the L1 TSON functions (e.g. transparent time-slice switching) required by both interconnection and bypass node. By using this approach, the following benefits can be achieved:

- More flexible experiment setup: For the given number of nodes, arbitrary combination of interconnection and bypass nodes can be selected for different experiment setup.
- Shorter hardware coding time: given the fact that there are certain number of common function, in addition that the function variance between interconnection and bypass nodes is limited in our case, a superset implementation approach is more efficient.
- Shorter hardware compile time: For one experiment, only compile once, and use the same generated bit file for all the nodes, and do not need to compile separately for interconnection and bypass nodes.
- Simpler design management: since only one implementation project is maintained, less management overhead can be achieved.

For the given reason, a superset TSON metro node, which performs the functionalities of both interconnection and bypass nodes is designed and implemented.

It covers three key areas relating to the delivery of TSON metro node that will fulfil the MAINS objectives, as follows

- Layer 1 implementation of the TSON metro node
- Layer 2 implementation of the TSON metro node
- Software/hardware defined network (SHINE) solution for TSON metro node to support intelligent adaptability

For this purpose, the TSON (OBST)-OPST testbed requirements defined in MAINS D1.1 and TSON (OBST) nodes detailed requirements and architectures defined in MAINS D2.3 are assumed as starting points.

Although this deliverable mainly focuses on Layer 2, Layer 1 implementation is also reported since it is an integral part of the TSON node. .

## 1.2 Reference Material

### 1.2.1 Reference Documents

[1] N. Amaya "Gridless Optical Networking Field Trial: Flexible Spectrum Switching, Defragmentation and Transport of 10G/40G/100G/555G over 620-km Field Fiber", Post Deadline ECOC 2011

[2] “PCI Express Myricom 10G NIC with two SFP+ Ports.” Storids 2010

[3] White Paper “HiTech Global HTG-V6HXT-100 User Manual” HiTech Global

[4] White Paper “Xilinx IP 10-Gigabit Ethernet MAC User Manual v10.2” Xilinx

http://www.xilinx.com/support/documentation/ip_documentation/ten_gig_eth_mac_ug148.pdf

[5] White Paper “Silicon Lab Si570 User Manual” Silicon Lab http://www.silabs.com/support%20documents/technicaldocs/si570.pdf

[6] White Paper “EMAC 100G Debug Application User guide” Hitek Systems

[7] White Paper “AEL 2006 Datasheet v2.3” Netlogic

[8] White Paper “Virtex-6 FPGA GTH Transceivers User Guide” Xilinx http://www.xilinx.com/support/documentation/user_guides/ug371.pdf

[9] White Paper “Any-Frequency Precision Sixx Clocks Family Reference Manual” Silicon Lab http://www.silabs.com/Support%20Documents/TechnicalDocs/Si53xxReferenceManual.pdf

[10] White Paper “LogiCore IP Virtex-6 FPGA GTH Transceiver Wizard v1.6” Xilinx http://www.xilinx.com/support/documentation/ip_documentation/v6_gthwizard_ds738.pdf

[11] White Paper “Virtex-6 FPGA GTX Transceivers User Guide” Xilinx http://www.xilinx.com/support/documentation/user_guides/ug366.pdf

[12] White Paper “Virtex-6 Libraries Guide for HDL Designs” Xilinx http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/virtex6_hdl.pdf

[13] White Paper “LogiCore IP Ten Gigabit Ethernet PCS/PMA v2.1” http://www.xilinx.com/support/documentation/ip_documentation/ten_gig_eth_pcs_pma_ug692.pdf

[14] White Paper “1 GB: x4, x8, x16 DDR3 SDRAM features” Micron http://download.micron.com/pdf/datasheets/dram/ddr/1GbDDRx4x8x16.pdf

[15] White Paper “UM10204 I2C-bus specification and user manual” NXP http://www.nxp.com/documents/er_manual/UM10204.pdf

[16] White Paper “FPGA Editor Guide” Xilinx

[17] White Paper “Vertex 6 Family Overview” Xilinx http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf

[18] Steve Kilts “Advanced FPGA Design” 2007

[19] “IEEE Communications” July 2010

[20] http://www.hitechglobal.com/

[21] http://www.digilentinc.com/

[22] http://www.xilinx.com/

[23] http://www.altera.com/

[24] http://www.fpga4fun.com/

[25] Transport of 10G/40G/100G/555G over 620-km Field Fiber", Post Deadline ECOC 2011

[26] "PLZT Optical Switch EPS0404 Series Nano-Second Speed 4x4 Optical Switch (Rack mounted)" User Manual, April 2011

[27] "DiamondWave FiberConnect Software Manual, Issue 2: DRAFT", 2007

[28] "WAVESHAPER 4000S, Finisar" 2010

[29] "Myricom Gen2 (5GT/s) PCI Express Myri 10G NIC with two SFP+ Ports"

### 1.2.2 Acronyms

| | |
|---|---|
| CEI | Common Electrical Interface |
| CAPEX | Capital Expenditure |
| CD | Chromatic Dispersion |
| DDR | Double Data Rate |
| EDC | Error Detection and Correction |
| FCI | Fast Causal Inference |
| FIFO | First In First Out |
| FMC | FPGA Mezzanine Card |
| FPGA | Field Programmable Gate Array |
| FPGA | Field Programmable Gate Array |
| Gbps | Giga bit per second |
| GE | Gigabit Ethernet |
| GMPLS | Generalized Multi-Protocol Label Switching |
| HDL | Hardware Description Language |
| I2C | Inter-Integrated Circuit |
| IP | Intellectual Property |
| ISP | Internet Service Provider |
| ITU-T | International Telecommunication Union-Telecommunication |
| LUT | Look Up Table |
| MAC | Media Access Control |
| MDIO | Management Data Input Output |
| MEMS | Micro Electro-Mechanical Systems |
| MMCM | Mixed-Mode Clock Manager |
| NE | Network Element |
| NIC | Network Interface Card |
| NPDR | Non-Predefined Definition Repository |

| | |
|---|---|
| OBST | Optical Burst Switching Technology |
| OEO | Optical-Electrical-Optical |
| OPEX | Operational Expenditure |
| OPST | Optical Packet Switching Transport |
| OPST | Optical Packet Switching Technology |
| PCI | Peripheral Component Interface |
| PCS | Physical Coding Sublayer |
| PDR | Predefined Definition Repository |
| PLZT | Polarized Lead Zirconium Titanate |
| PMA | Physical Medium Attachment Sublayer |
| PMD | Polarisation Mode Dispersion |
| QDR | Quad Data Rate |
| QoS | Qaulity of Service |
| SSS | Spectrum Selective Switch |

## 1.3 Document History

| Version | Date | Authors | Comment |
|---|---|---|---|
| 0.1 | 11/09/2011 | Y. Qin | First ToC draft |
| 0.2 | 15/09/2011 | Y. Qin | Final ToC version |
| 0.3 | 12/11/2011 | Y. Yan | Contribution on Section 3 |
| 0.4 | 13/11/2011 | Y. Qin | Contribution on Section 5 |
| 0.5 | 26/11/2011 | Y. Qin | Contributions on Section 1 |
| 0.6 | 28/11/2011 | B. Rahimzadeh | Contribution on Section 2 and 4 |
| 0.7 | 28/11/2011 | Y. Qin, Y. Yan | Contributions on Section 5, update section 4 |
| 0.8 | 28/11/2011 | Y.Qin | Review, document overview |
| 0.9 | 29/11/2011 | Y.Qin, Y. Yan, B. Rahimzadeh | Executive summary added, all sections updated |
| 1.0 | 30/11/2011 | Y.Qin | Final |
| 1.1 | 30/11/2011 | G. Zervas | Final Review and release |

## 2 Overview of TSON Metro Node Architecture

TSON node located in the TSON metro mesh cloud should interconnect with the OPST ring, access nodes, directly with end-points (clients, data centre servers), and other TSON nodes. This interconnection is controlled by GMPLS control plane, while each

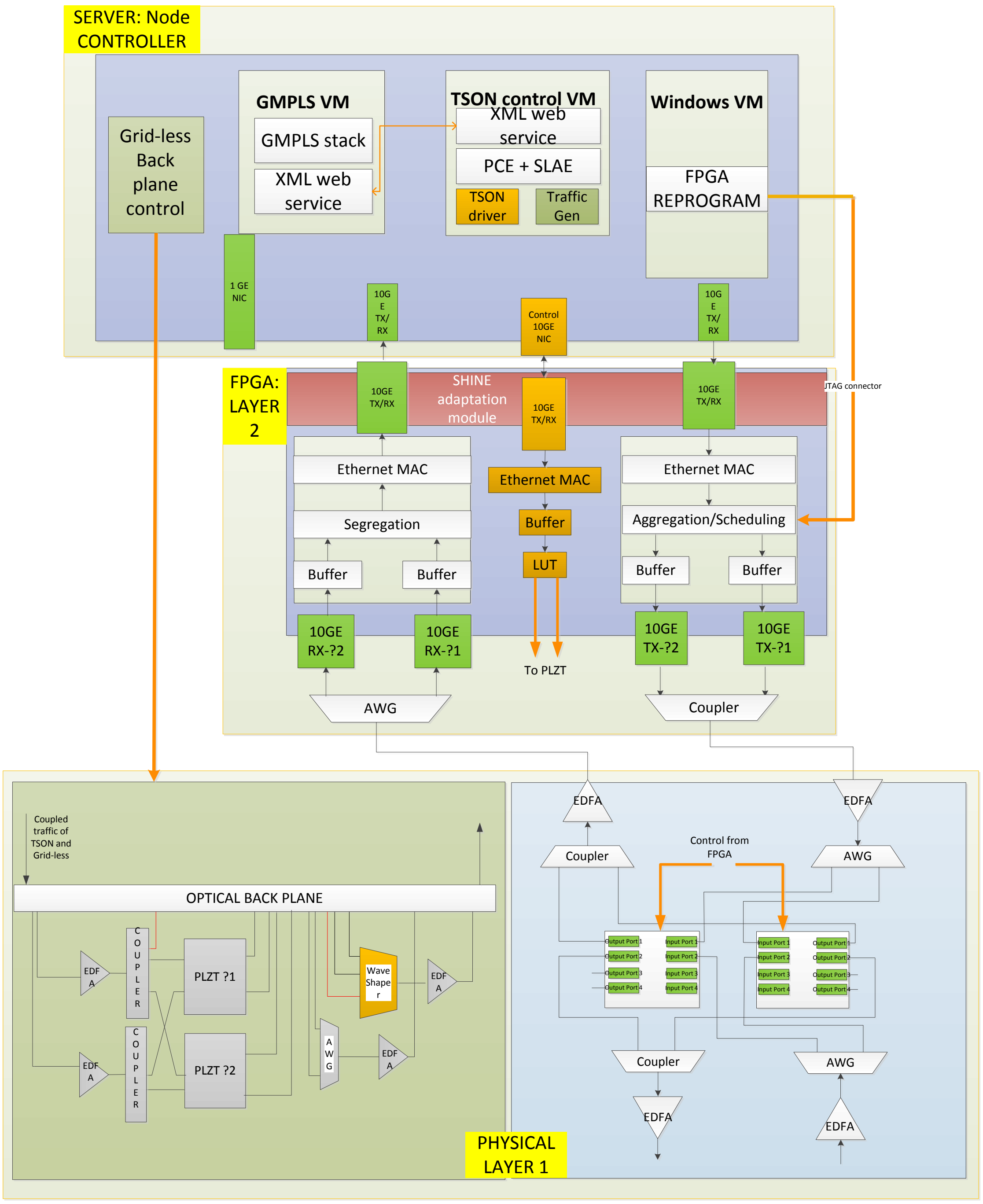


Figure 2-1: The TSON node complete structure consisting of controller on top, L2 and L1 elements.

TSON node uses the control information to manage the node`s operations vertical to the control.  and as such it needs to provide a vertical node management interfaceTSON nodes are built having Layer 1 and Layer 2 functionalities, to be capable of both operating as edge and bypass nodes. As shown in Figure **2-1**, the node design consists of three distinct elements. First, the node controller that operates in a server that supports the overall TSON control, the interface to high-speed Virtex6 FPGA and optical switches and can also host the GMPLS stack. The second element is the high-speed Virtex6 FPGA that has been programmed to support the L2 TSON functions. Finally, the Layer 1 element refers to two flavours of OXCs able to support either TSON over 50GHz multiplexed network or TSON over Gridless multiplexed network. In regards with Layer 1 elements, two node scenarios have been implemented; one capable of supporting TSON time-shared data transfers and switching over 50 GHz switched channels, and the other one supporting the time-shared data transfer and switching over Gridless frequency-shared channels that follows Gridless networking, besides the TSON networking.

The need for frequency flexible switching and transport and potential support of super high-speed communication channels (e.g. beyond 100Gbps) but also sub 50 GHz channels, is addressed by extending the TSON Layer 1 to support Gridless data switching. Such support can allow for aggregated metro traffic to be transported over ultra-wide channels to core network or carry 10Gbps channels over 25 GHz frequency-slices. This additional implementation is carried out having the emerging Gridless switching technologies such as spectrum selective switches in mind. The implemented L2 functions are reported on Section 3 and the different L1 OXC configurations on Section 0. The following sections reflect the workflow and interactions between node controller, L2 element and L1 element.

## 2.1 TSON Metro Node

The implemented TSON node is capable of performing edge and bypass functionalities. For this purpose, the node structure consists of a number of hardware and software modules to carry out the data and control functionalities.

In this regard, TSON node consists of a server as the management plane platform, needed both in the L2 and L1 TSON node configuration and control such as resource allocation; a FPGA, as the Layer 2 enabling Ethernet to TSON data conversion, aggregation, scheduling and transmission, carrying out OEO operation at the edge of the TSON mesh metro; and the physical layer (Layer 1) involving the OXC consisted of switching devices and other optical components.

### 2.1.1 Node Controller

The server on top running Ubuntu32 operating system is responsible for control and management tasks of the node. The server hosts virtual machines and other necessary software programs to support nodes communications whether as a TSON edge or a TSON bypass node, and also holds a resource allocation tool which assigns time slots over the wavelengths upon connection requests. This resource allocation tool will serve all the requests applying for the resources and connections on the TSON network. The description of the tool will be provided on Deliverable D4.5 “Implementation of sub-lambda assignment element”.

### 2.1.2 Data plane

The data plane of the TSON node consists of L2 and L1 enabling devices and components.

#### 2.1.2.1 Layer 2 with High-Speed FPGA

Regarding the L2 functionalities, a high performance FPGA is placed in the node. This FPGA exploiting a number of deployed function blocks takes care of the OEO operations, sending out the data over two transceivers with different wavelengths. Packet processing, traffic aggregation/segregation, time-slice scheduling and generation, and the fast switches control in the physical layer are the main task blocks carried out by the FPGA.

The OEO operations however, are specific to the TSON node operating as an edge node, since the edge nodes in the TSON mesh cloud are expected to handle the Ethernet traffic of the other network regions to being transferred over the TSON network. The TSON bypass nodes on the other hand, need to operate transparently to the TSON time-sliced traffic and to switch it considering the information received from the control plane.

#### 2.1.2.2 Layer 1 data transport

Data transport over the TSON Layer 1 is mainly conducted by exploiting two PLZT fast switches, each assigned for one wavelength following the wavelength-modular architecture. The generated data by the transceivers on the FPGA are directed using the switches and some supplementary optical components (couplers, MUX/DEMUX). The details for the implementations are discussed on Chapter 0.

### 2.1.3 TSON Work Flow

In order to orchestrate the operation of the TSON node components for the overall desired function of the node, the operation of some of the active components in the node architecture must be timely controlled. Considering Figure 2-2, In case of a TSON edge node, with a client requesting for end-to-end bandwidth service, the sub-wavelength path computation is performed by the resource allocation tool on the server. Before informing the client for data generation and transmission, the management plane on the server communicates with the FPGA through a 10GE link sending resource allocation and control commands such as sub-wavelength LUT information (bit-tables per wavelength) to the FPGA. Based on the LUT information, the FPGA aggregates and schedules the data transmission of the ingress Ethernet traffic from the server, and sends them out as timely organised traffic time-slice data sets.

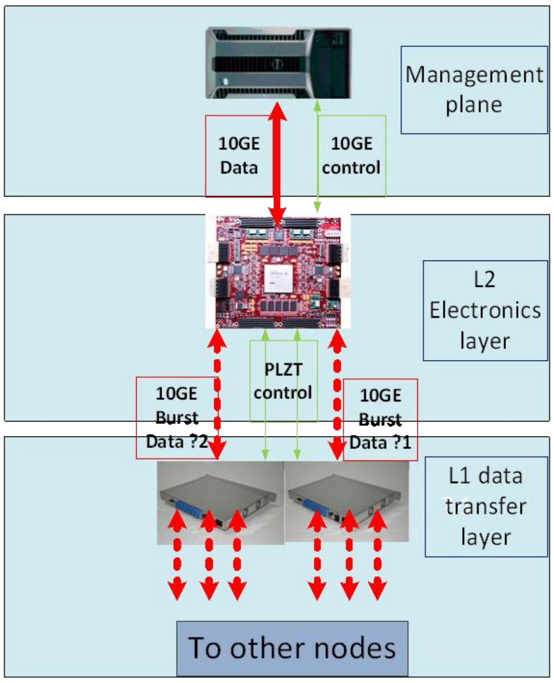


Figure 2-2: The TSON node work flow

Alongside of the timing information for the FPGAs operation, PLZT control commands are also sent to the FPGA, so the FPGA can configure the PLZT at the correct time instances. The information being used for controlling the PLZT is based on the time slice allocation information calculated in the server per input-output PLZT switch cross connection. The PLZT information will be passed to the module through a pair of custom made DB25 connectors explained in the interfaces section.

## 2.2 TSON Metro node with Gridless support

In the second node scenario, the TSON node architecture is extended to address the future need of supporting ultra high speed data channels that require broad spectral use and switching (e.g. 650 GHz) [1] between metro and core regions as well as frequency-based sub-wavelengths..

In this regard, since the channel spacing being used for the gird-less connection does not fit into the regular ITU-T 50GHz defined grid, extra components, mainly a waveshaper are used in the data plane to enable flexible spectrum selective switching.

In order to support TSON over Grid and Gridless networking as an add-on capability to the normal TSON operation, a dynamic data plane is built, taking advantage of an optical backplane as a repository for various components. So basically, the enabling devices for TSON and Gridless operation are selected from this backplane and included in the operation.

### 2.2.1 Node Controller

As explained for the TSON node, the server on top, running Ubuntu32 operating system is responsible for control and management tasks of the node. The server hosts virtual machines and other necessary software programs to support nodes communications whether as a TSON edge or a TSON bypass node, and also holds a resource allocation tool which assigns time slots over the wavelengths to connection requests. This resource allocation tool will be described in D4.5 “Implementation of sub-lambda assignment element “.

In addition to this, the resource allocator tool is extended to support Gridless networking. In this regard, the management plane communicates with the Gridless block and the dynamic backplane, for incorporating the Gridless switching into the node and building up the enabling data plane for that.

### 2.2.2 Data plane

In order to support both the TSON and Gridless operation, a dynamic data plane is built and the server in the node architecture controls the backplane.

In this regard, built TSON data plane and the Gridless data plane are explained.

#### 2.2.2.1 Layer 2 with FPGA (TSON mode)

The data plane of the TSON node consists of L2 and L1 enabling modules and components.

The L2 functionalities are only used for TSON metro node being used as an edge node. The reason is the bypass node whether TSON metro by pass or for the Gridless transmission do not carry out any data grooming and processing. The TSON L2 functionalities are as explained in Section 2.1.2.1.

#### 2.2.2.2 Layer 1 data transfer

The dynamic data plane, using a optical backplane together with optical switches and components will provide TSON and Girdless bypass switching. However, the backplane incorporates other modules for bypassing the Gridless traffic. This data plane is discussed in 4.1.

### 2.2.3 TSON plus Gridless Work flow

Having the node equipped with Gridless transmitter, the node should control the back plane in addition to the TSON control. Following this concept, the management plane is updated to support this extra functionality. This is while the TSON node work flow stays the same as it is discussed in 2.1.2. For Gridless functionality, the controller has been extended. in order to build up the desired data plane, OXC configuration using commands for cross-connecting the plug-in modules (e.g. PLZT, waveshaper) are sent to the back plane for incorporating the required components.

Therefore, upon receiving a request for Gridless transport, the optical backplane (Calient fiber switch) is informed about the required setup and the waveshaper gets included. After this, the Gridless transmitter starts to transmit.

The FPGA is not involved in the Gridless operations since there is no need for L2 functionalities, or time-slice generation and PLZT control.

## 2.3 Hardware Components for the node controller

Enabling hardware for implementing the higher layer node controller of the node are implemented using the following hardware.

### 2.3.1 Servers[3]

In order to deploy virtual machines and higher layer control, DellT701 servers have been purchased with the following characteristic:

Figure 2-3: Dell server

- Intel Xeon E5620 Processor (2.40GHz, 4C, 12M Cache, 5.86 GT/s QPI, 80W TDP,
- Turbo, HT), DDR3-1066MHz
- 12GB Memory for 1CPU (6x2GB Single Rank UDIMMs) 1333MHz
- 500GB SATA 7.2k 3.5" HD Hot Plug
- PERC H700 Integrated RAID Controller, 512MB Cache
- 16X DVD+/-RW Drive with SATA Cable
- Non-Redundant Power Supply (1 PSU) 1100W
- Rack Power Distribution Unit Power Cord
- Embedded Broadcom GbE LOM with TOE and iSCSI Offload HW Key, NOT
- compatible with H200/H700 on non-56xx bases
- iDRAC6 Express
- Sliding Ready Rack Rails with Cable Management Arm for Rack Configuration
- C2 - R1 for PERC H700, Exactly 2 Drives

[3] picture from http://www.glcomp.com/products/servers/dell-poweredge/tower-servers/poweredge-t710

These servers are equipped with 10GE PCI NIC Myricom cards. These interfaces will be used for communicating with the 10GE ports on the FPGA and also transmitting and receiving Ethernet traffic.

### 2.3.2 10GE Myricom Ethernet interfaces

These NICs are designed for high speed/ low cost high performance networking requirements. According to the data sheet, the NICs supporting 10GE interface are designed to meet High Performance computing (HPC), database, video, virtualization, and storage applications [2].

The properties of this module are presented in table 2-1 from[29].

Figure 2-4: Myricom NIC, figure from [2]

| | |
|---|---|
| **Data rate supported per port** | 10 Gb/s |
| **Bus type** | „Gen2“ PCI Express 2.0 (5GT/s or 2.5 GT/s) |
| **Bus width** | x8 lane PCI Express, operable in x8 or x16 slots |
| **Bus speed** | 4 GBytes/s (500 Mbytes/s per lane) each direction |
| **Compliance** | PCI Express Card Electromechanical 2.0 /PCI Express Base 2.0 |
| **Operating temperature** | 0°C to 55°C (32°F to 131°F) with 100 LFM min airflow |
| **Operating humidity** | 15% to 80% @ 50C, non-condensing |
| **Storage temperature** | -40C to 70C |

**Table 2-1: Myricom 10GE NIC propertis**

## 2.4 Interfaces

### 2.4.1 Interfacing servers with FPGA

The management plane of the TSON node needs control communication link with the other layers of the TSON, especially with the Layer 2 FPGA . This communication line will carry the resource allocation information as well as fast switch control for orchestration of the components forming the TSON node.

In this regard, server hosting the management plane, is connected to the FPGA as the Electronics layer, using Ethernet Sockets through a 10GE interface. For this purpose 10GE Myricom NIC with two LR SFP+ transceivers are installed on the server. The FPGA is interfaced with the server through this transceiver. To make this control communication happen, a java program has been developed and installed on the server, as an extension to the RWTA engine.. This java program utilizes a library called JPCAP to support communication through the Ethernet interface. Using this library, Raw Ethernet packets can be generated and customised with the desired data. The mentioned library requires a 32bit OS for operations.

### 2.4.2 Interfacing FPGA with PLZT switches

The fast switches, as the active components of the TSON Layer 1 data plane which needs to be configured for switching at correct times to direct the bursts of traffic, need to get the switching information in a timely and effective manner.

For this purpose, the FPGA as the layer which carries out the time-slice generation and transmission is made responsible for passing the switch control commands to the switch simultaneously with the data generation.

Each of the fast PLZT switches (two switches, one per wavelength) is configurable through 48 pins on two DB25 connectors. For this purpose, the expansion connectors on a daughter board card attached to the FPGA board are connected to the DB25 connectors on the PLZT switch using a custom made ribbon cables.

The desired switch configuration is then sent to the PLZT switch from the FPGA at the

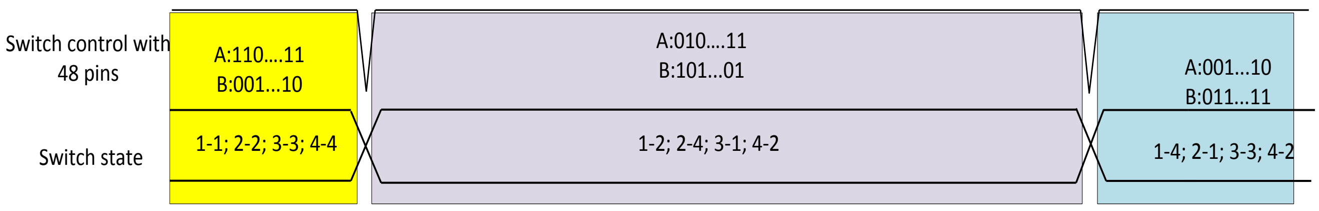


Figure 2-5: PLZT control bits and the correspondence switch state change

specific timings determined by the central controller.

Figure 2-5 illustrates the 48 control bits (split in two parts of A and B due to the PLZT functional structure), and the correspondence change in the state of the switch is displayed. So, as long as the control bits sent are constant, the switch remains at the same state. The control bits are raw zeros and ones (3.3 Vpeak-to-peak) directly applied to the switch driver.

# 3 Layer 2 of TSON Metro Node

In this section, the electronic design and implementation of L2 TSON Metro Node is presented. The work includes the design requirements, suitable hardware and implementation strategy, design implementation, experiments and experiment results analysis.

## 3.1 Design Requirement

The electronic design part requires implementing TSON Metro Node in a single FPGA, which can achieve the requirements in MAINS deliverable 1.1 Table 6-1. This Subsection analyses the node requirements, and gives a detailed FPGA overall and functional block design requirement.

### 3.1.1 Overall Requirement

GMPLS control plane provisions the resources of the whole TSON network and sends the time-slice allocation and PLZT switch commands to the node. After the sub-wavelength lightpath establishment, Ethernet frames arrive at the ingress TSON node. The TSON node then that parses and aggregates the frames based on the traffic destination and the allocated time-slice(s) and finally transmits them on the specific time slice. Here, one TSON time-slice is 10us, and one TSON frame is 1msAnd, one frame could contain 100 time-slices in theory. At the egress node in TSON network the optic time-slice data sets are segregated and send out in the form of Ethernet frames. 10Gbps transceivers are assigned for the Ethernet frame transmission, optical time-slice transmission and GMPLS control interface.

Furthermore, Reusability of the developed modules is considered to improve the value and flexibility of the design.

### 3.1.2 Component Requirement

#### 3.1.2.1 10Gbps Ethernet PCS/PMA

10Gbps Ethernet PCS/PMA is required to provide high-speed transmission as well as 64B/66B encode and decode, clock correction, TX and RX state machines and BER monitor. Meanwhile, it needs to be easily configured through management interface. Furthermore, a standard interface, i.e. XGMII interface, would be a benefit to connect it to the 10Gbps Ethernet MAC core.

#### 3.1.2.2 10Gbps Ethernet MAC

10Gbps Ethernet MAC is required to satisfy IEEE 802.3-2008 standard. It is also desired to be easily configured and have an XGMII interface.

#### 3.1.2.3 Ethernet Aggregation

Ethernet Aggregation module is the major component of the ingress function of the node. When the node receives the Ethernet packets, it is required to aggregate the Ethernet packets to a time-slice data set, check the Time-slice allocation Look-Up-Table (LUT), wait for an available Time-slice allocation, and then send out the burst in the specific time slice.

#### 3.1.2.4 Optical Burst Segregation

The requirement of optical burst segregation module is that, whenever the node receives an optical burst, it segregates the burst to Ethernet frames, and sends the Ethernet frames out immediately. The optical burst segregation module is required in the egress node.

#### 3.1.2.5 PLZT Switch Controller

PLZT switch controller is required to control the PLZT switch following the instruction of GMPLS. It is required to employ a LUT to store all the switching commands. Then based on the LUT, PLZT switch controller sends the control signals through parallel interface before sending out the burst.

#### 3.1.2.6 10Gbps Server/GMPLS to FPGA Interface

The TSON Metro Node functionally operates bases on the commands from GMPLS, which sends the instruction through a 10Gbps transmission link to update the LUTs of aggregation and PLZT switch controller in the FPGA. Therefore, this 10Gbps transmission link between GMPLS and FPGA is required to to process the Ethernet frames and update the LUT.

## 3.2 Implementation Strategy

Considering the design requirement in the Subsection 3.1, it is important to identify a suitable hardware platform, and then for, choose the proper design strategies to implement this specific design from electronic digital design point of view.

### 3.2.1 Hardware Selection

As the node is expected to support 10Gbps Ethernet and 10Gbps TSON interfaces; thus, to achieve this high-speed and low-latency performance, the hardware platform should be able to support 10Gbps interface and have considerable internal memory in the FPGA.

Comparing the FPGA technology and FPGA platforms' performance in the market, HiTech Global HTG-V6HXT-100G was chosen as the main platform, and HTG-SFP-PLUS was chosen as an extender card to support 12 SFP+ optical transceiver ports. HTG-V6HXT-100G is based on the latest Xilinx FPGA technology Virtex6 XC6VHX380T and features:

- 40nm ExpressFabric architecture,
- 600MHz clock management tiles,
- 600MHz block RAM,
- 600MHz DSP48E1 slices,
- 1.4Gbps SelectIO with ChipSync technology,
- PCI Express Endpoint/Root Port blocks,
- Ethernet Media Access Controller blocks,
- 27648KB Block RAM,
- 48 GTX transceivers up to 6.6Gbps and
- 24 GTH transceivers up to 11.18Gbps.

The detail description of the hardware platform will be described in Subsection 3.3.1.

### 3.2.2 FPGA Design Strategy

There are three primary physical characteristics of a digital design: *speed*, *area* and *power*. As described in the Subsection 3.1, the critical requirement of the node design is 10Gbps

***speed***. Therefore, this subsection describes the design strategy of architectural timing improvements; then, to make the components reusable, a universal interface of the components in the project is presented.

#### 3.2.2.1 Pipeline

The FPGA input and output port interface runs serially at 10Gbps, thanks to the parallel processing feature of FPGA. To improve the throughput and reduce the speed in the FPGA, the data bus width is set to 64 bits, and then the FPGA execution clock is 156.25MHz.

The idea with such kind of high-throughput design is the use of pipelining. A pipeline is a method to parse the task to various stages. When the data enters, it is passed through different stages and then exits. The benefit of pipeline is that new data can be processed before the prior data has finished.

For the ingress function of the node, to improve the system throughput, due to the storage function of different FIFOs in different stages, the tasks of ingress module can be pipelined as illustrated in Figure 3-1.

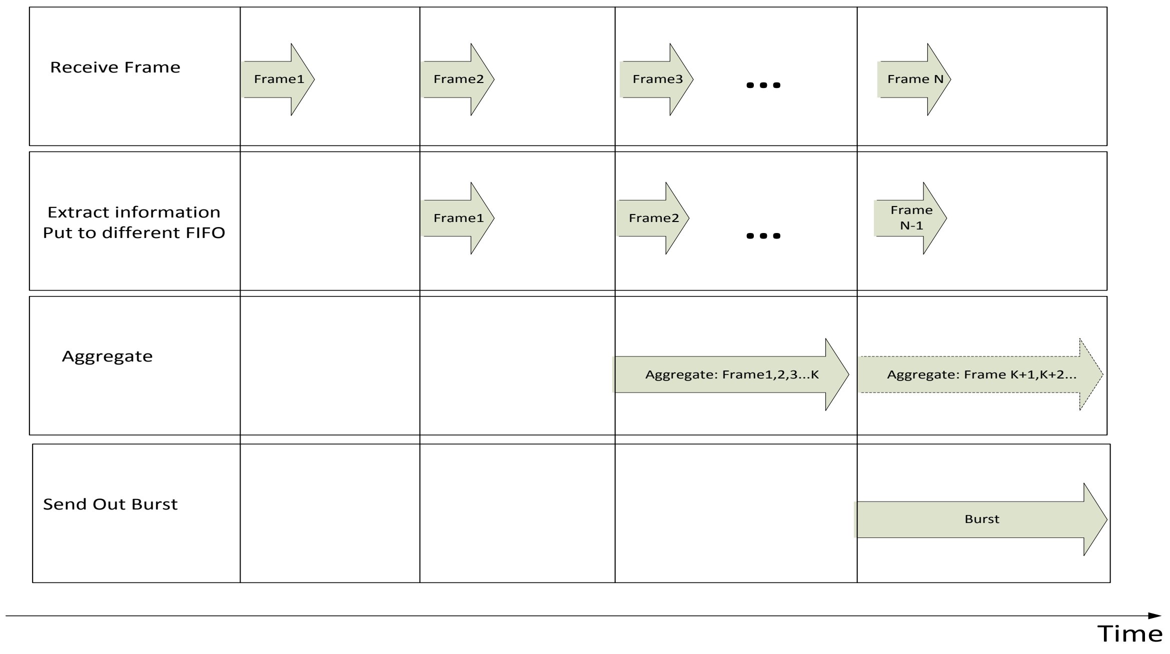


**Figure 3-1: Pipelined Processing Diagram**

The pipelined design includes four stages:

Firstly, when receiving Ethernet frames, store good frames and discard bad frames.

Secondly, extract Ethernet frame headers and send different kinds of frames to different buffers.

Thirdly, aggregate the Ethernet frames to a burst (time-slice)

Lastly, send out the burst according to the time-slice allocation.

#### 3.2.2.2 Reuse

Design reuse can improve the quality and value of the FPGA design, the benefits will become apparent on later work and future projects. Formal development policies can help to establish a design reuse methodology.

Some basic points need to be followed, such as, follow the HDL coding rule, use parameter to create maximum flexibility, follow the signal and parameter naming conventions, create testbench for the easy reuse, etc. Another important point is to adopt a common interface protocol on all components. Figure 3-2 shows a common interface adopted in the design. The interface includes clk, reset, input and output ports.

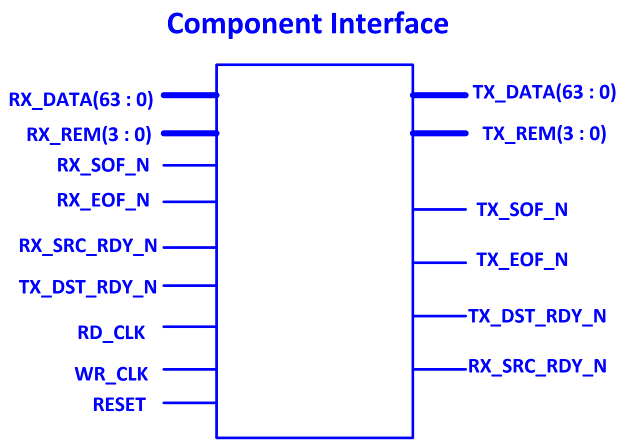


**Figure 3-2: Common components Interface**

The descriptions of the interface ports are listed in Table 3-1, all the flag signals are active low. The transmission timing diagram of some valid data is shown in Figure 3-3.

| Port Name | Dir | Description |
|---|---|---|
| WR_CLK | In | Receiver Port: Clock |
| RD_CLK | In | Transmitter Port: Clock |
| RESET | In | Reset, active high. |
| RX_DATA[63:0] | In | Receiver Port: input data |
| RX_REM[3:0] | In | Receiver Port: input data encrypted validation. |
| RX_SOF_N | In | Receiver Port: start of burst, active low |
| RX_EOF_N | In | Receiver Port: end of burst, active low |
| RX_SRC_RDY_N | In | Receiver Port: source burst ready to be read, active low |
| TX_DST_RDY_N | In | Transmitter Port: transmit side destination ready to receive the data, active low |
| TX_DATA[63:0] | Out | Transmitter Port: output data |
| TX_REM[3:0] | Out | Transmitter Port: output data encrypted validation. |
| TX_SOF_N | Out | Transmitter Port: start of burst, active low |

| | | |
|---|---|---|
| TX_EOF_N | Out | Transmitter Port: end of burst, active low |
| TX_SRC_RDY_N | Out | Transmitter Port: source burst ready to send out, active low |
| RX_DST_RDY_N | Out | Receiver Port: receiver side destination ready to send the data, active low |

**Table 3-1: Common Interface Port Description**

As illustrated in Figure 3-3, when the user wants to transmit or receive an Ethernet frame/burst, and the output is ready to receive (dst_rdy_n active), it starts to transmit/receive the data with asserting source ready (src_rdy_n), start of data (sof_n), and end of data (eof_n).

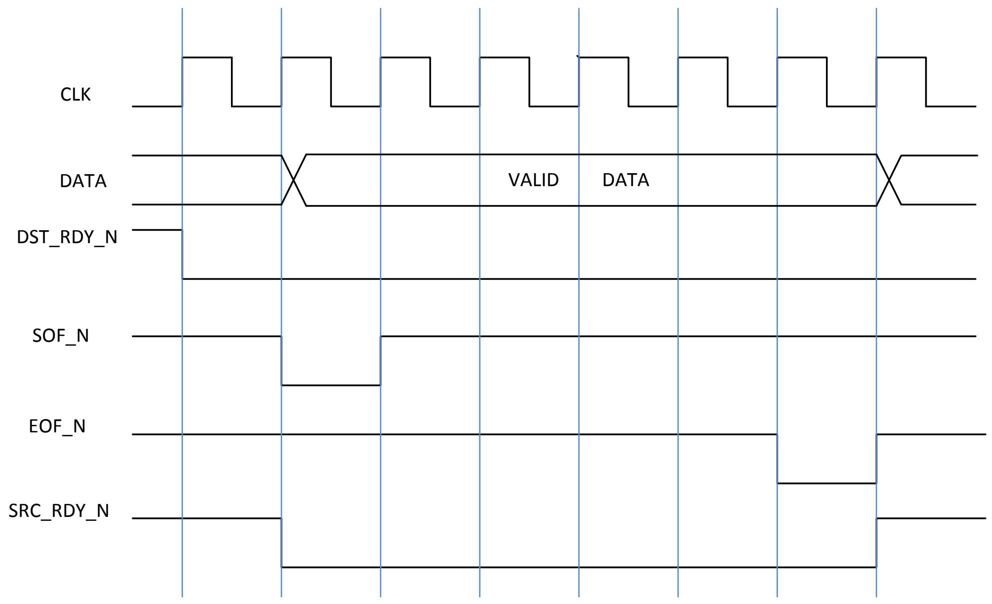


**Figure 3-3: Common Interface Timing Diagram**

## 3.3 Node Implementation

Based on the design requirement in Subsection 3.1 and design strategy in Subsection 3.2, considering the features and limitations of FPGA, this subsection describes the architecture for a single TSON Metro Node.

As described before, the architecture is designed according to the features and limitations of the hardware platform; therefore, the FPGA design is divided into several modules. The node block diagram is shown in Figure 3-4. The modules and module requirements are in Table 3-2.

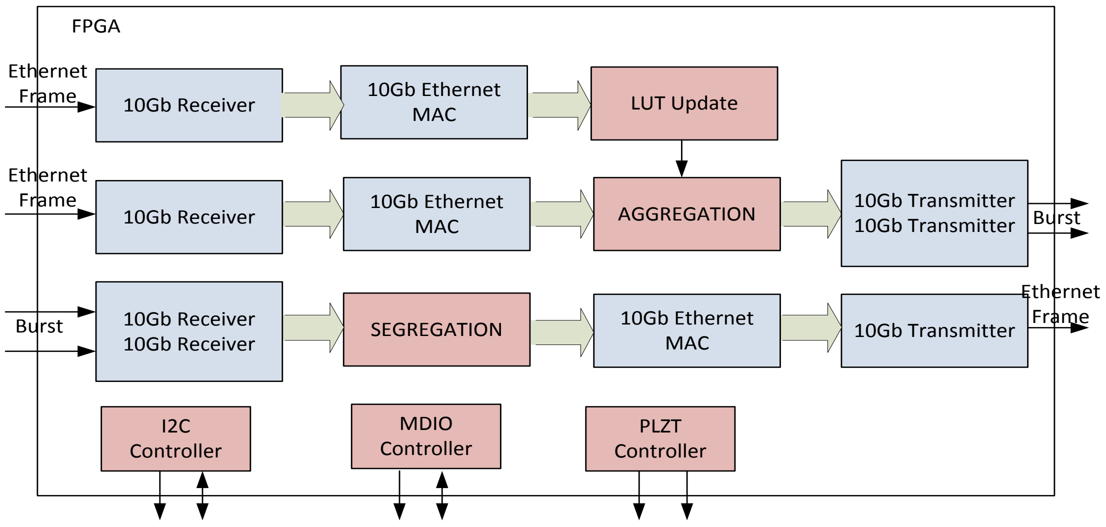


**Figure 3-4: TSON Metro Node Block Diagram**

In Figure 3-4, the blocks with blue colour are supplied by Xilinx as IP cores, which include 10Gb Transmitter/Receiver and 10Gb Ethernet MAC. The other modules were developed (coloured red) in VHDL. The design details will be described in the Subsection 3.3.3.

| Module Name | Functions |
|---|---|
| I2C controller | Set Si5368/Si570 on board to generate the clock. |
| MDIO controller | Initialize HTG-SFP+ extender card on-board-chip AEL2006. |
| PLZT controller | Control PLZT switches. |
| 10Gbps Transmitter/Receiver | Xilinx IP core GTH for 10Gbps Transmission. |
| 10Gb Ethernet MAC | Xilinx IP core 10Gb Ethernet MAC. |
| Aggregation | Aggregate Ethernet Packets to burst and send the burst out based on the Time-slice Allocation LUT. |
| Segregation | Segregate burst to Ethernet Packets and transmit out when received. |
| LUT Update | Receive and parse the information from PLZT, Update the LUT. |

**Table 3-2: FPGA design functional modules and requirements**

Furthermore, considering the clock domains, all 10Gbps receivers and transmitters work at 156.25MHz, but in different clock domains. The I2C controller and MDIO controller run in the clock domain of 50MHz. For the design of the TSON Metro Node, totally, there are 8 clock domains. Therefore, an important concern of the design is handling the cross-clock domain signals.

Following Figure 3-4 of the Node architecture, a diagram of detailed functional blocks of TSON Metro Node is shown in Figure 3-5.

Compared with Figure 3-4, Figure 3-5 gives more details of how the node is designed. The purple blocks construct the ingress function of the node, the orange blocks provide the egress function, and the green blocks complete a interface link from the server (connected to GMPLS) to the node. The data flow is following the arrow directions.

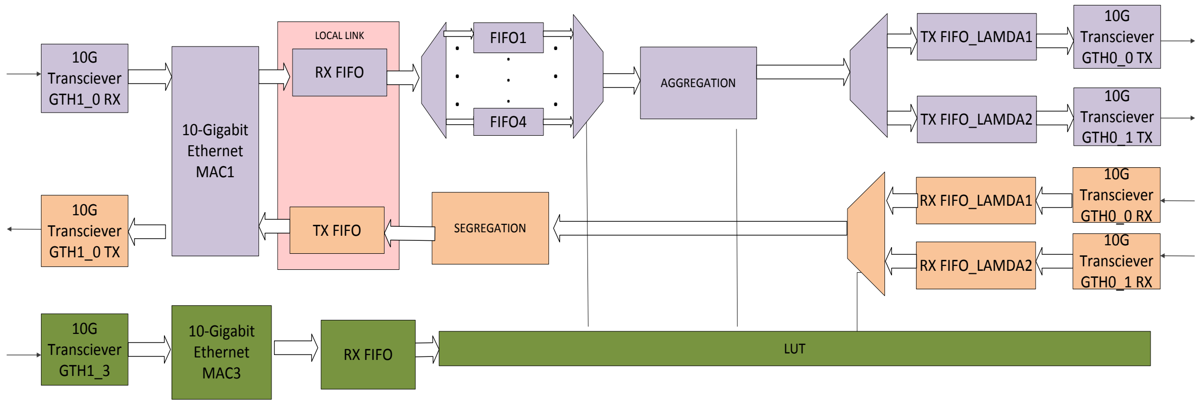


**Figure 3-5: TSON Metro Node Layer 2 FPGA Design Functional Blocks**

For the overall node controller to node interface link, time-slicethe server sends eight Ethernet frames through 10 Gbps link; Then the frames are passed to the 10GE MAC which drops the preambles and FCS, and tells the next block RX FIFO whether it is a good packet or not; RX FIFO updates the LUT with the packet information; After this, LUT updates the register file in aggregation block.

For the ingress part of the node, when the 10Gbps receiver receives the Ethernet frames, through XGMII interface, it passes them to the 10GE MAC; Then the MAC discards the preambles and FCS, transmits the data to the RX FIFO and indicates whether the packet is good or not; The RX FIFO receives the data, waits for the good/bad indication from MAC, sends it to the DEMUX block if they are valid data. The DEMUX analyses the Ethernet frame information (i.e. Destination MAC address, Source MAC address and so on) and puts them in different FIFO. After that, the FIFO doesn't send any data until the AGGREGATION gives a command; the register file of AGGREGATION, containing the Time-slice Allocation information, is updated by the LUT. AGGREGATION waits for the burst-length Ethernet frames ready in the FIFO and the time-slice allocation available, then transmits the bursts into the different wavelength TX FIFO. The TX FIFO adjusts the time with PLZT controller, and then sends the burst out.

For the egress part, when the 10Gbps receiver receives the burst (time-slice), it drops it in the RX FIFO Lamda1/Lamda2; after the burst is completely received, the SEGREGATION block segregates the burst to Ethernet frames and transmit them to a TX FIFO; every time TX FIFO receives a complete Ethernet frame, it sends it to the 10GE MAC; Finally, the MAC passes the data to the 10Gbps transmitter and transmit them out.

As described above, the cross-clock-domain signals should be taken care of. Several FIFOs were employed to overcome the cross-clock-domain problems. As shown in Figure 3-5, for the ingress part, the read out side of TX FIFO and 10Gbps GTH transmitters work in TX_CLK1_ingress/ TX_CLK2_ ingress domain; and the other modules work in RX_CLK_ ingress domain. For the egress part, the right side of RX FIFO and 10G GTH receivers work in RX_CLK1_egress/RX_CLK2_egress domain; and the other modules work in TX_CLK_egress domain. For the LUT update module, they work in the same clock domain CLK_LUT.

In the rest parts of this section, IP cores and the user logic blocks are described.

## 3.3.1 Hardware Components

The overall hardware platform is shown in Figure 3-6, there is a HTG-V6HXT-100G board used as the main board of the platform; on the left side, there is a HTG-SFP-PLUS-MDL extender card, optical SFP+ transceivers and fibre patchcords; on the right side, there is a HTG-AMAX-LOOPBACK card for 10Gbps transceivers debugging; the green boards are the Xilinx daughter cards to interface the HTG-V6HXT-100G board with PLZT switch.

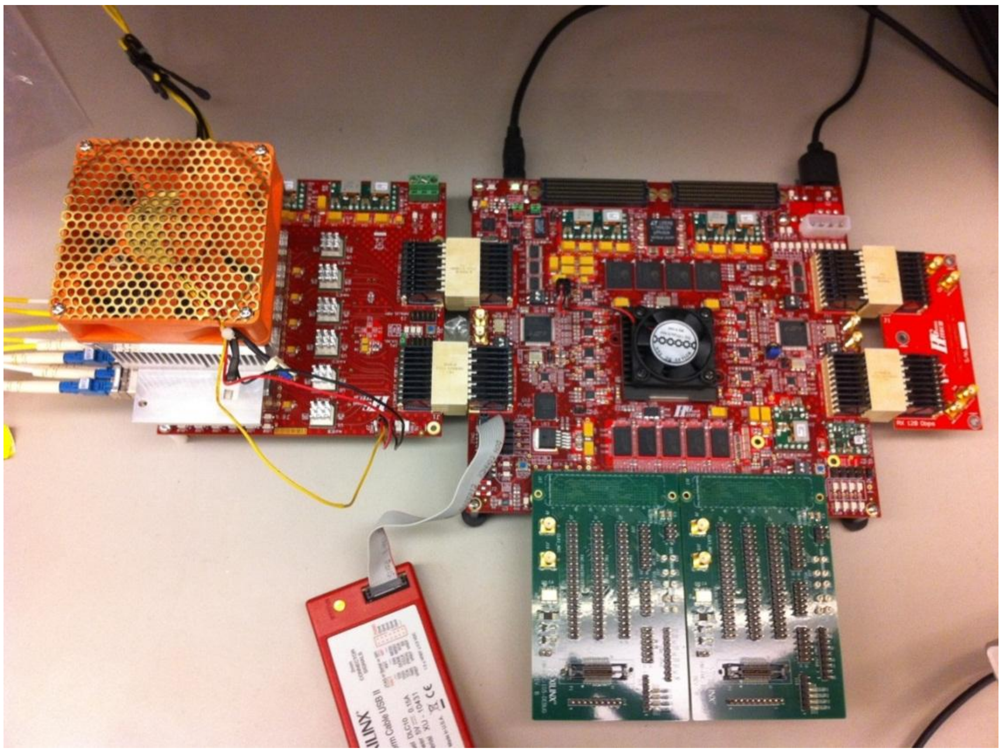


**Figure 3-6: Hardware Platform**

### 3.3.1.1 MAIN Board: HTG-V6HXT-100G

HTG-V6HXT-100G platform is based on the latest FPGA technology from Xilinx, it is featured with Virtex-6 HX380T device, 32 GTX serial transceivers (maximum speed 6.5Gbps), and 24 GTH lanes (maximum speed 11.3Gbps). It provides a simple user interface for connecting to higher layer data processing via interface as well as optical interfaces to connect to emerging 100G and 40G modules. The board block diagram is shown in Figure 3-7.

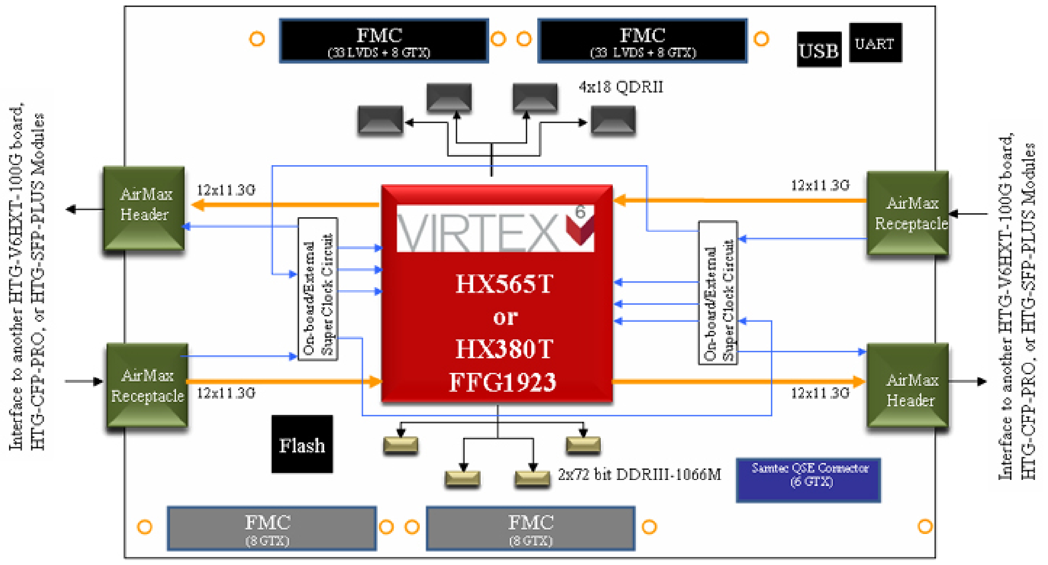

FMC (33 LVDS + 8 GTX)
FMC (33 LVDS + 8 GTX)
USB
UART
4x18 QDRII
AirMax Header
AirMax Receptacle
AirMax Receptacle
AirMax Header
12x11.3G
12x11.3G
12x11.3G
12x11.3G
On-board/External Super Clock Circuit
On-board/External Super Clock Circuit
VIRTEX 6
HX565T or HX380T FFG1923
Flash
2x72 bit DDRIII-1066M
Samtec QSE Connector (6 GTX)
FMC (8 GTX)
FMC (8 GTX)
Interface to another HTG-V6HXT-100G board, HTG-CFP-PRO, or HTG-SFP-PLUS Modules
Interface to another HTG-V6HXT-100G board, HTG-CFP-PRO, or HTG-SFP-PLUS Modules

**Figure 3-7: HTG-V6HXT-100G board block diagram** [3]

Except the Xilinx Virtex6 HX380T FPGA, the other hardware components on the HTG-V6HXT-100G platform include:

DDRIII SDRAM Memory: The HTG platform is populated with Ten Micron DDR-III components, each of which is 8Meg x 16 x 8 banks. The clock frequency ranges from 300MHz to 800MHz. It also supports self-refresh mode and automatic self-refresh.

QDRII SDRAM Memory: Quad Data Rate (QDR) SRAM is a type of static RAM that can transfer up to four words of data in each clock cycle. QDR SRAM transfers data on both rising and falling edges of the clock signal. The QDRII SRAMs are similar to QDR SRAMs in their operation but with some performance improvements. The HTG platform is populated with four Cypress 72-Mbit QDR-II+SRAM 4-Word Burst components. This QDR-II memory can work at up to 550MHz clock. There are two input clocks for precise DDR timing and echo clocks for simplifying data capture in high speed systems. It is also featured with separate independent read and write data ports.

FPGA Mezzanine Card (FMC) Connectors: Developed by a consortium of companies ranging from FPGA vendors to end users, the FPGA Mezzanine Card is an ANSI standard that provides a standard mezzanine card form factor, connectors, and modular interface to an FPGA located on a base board [3]. The HTG platform is populated with Four 400-pin Samtec connector for implementation of Vita 57 FMC Connectors, which provides access to LVDS IOS, Serial Multi-Gigabit IOs, JTAG signals, I2C signals, and multiple differential clocks. Each FMC connects 8 GTX serial transceivers.

FCI AirMax Connector: On side of the HTG-V6HXT-100G board, there are four High-Performance FCI AirMax connectors for interoperability with existing line cards, each AirMax connector supports 6 transmit and receive lanes(@11.3Gbps). All 12 GTH lanes are connected to these AirMax connectors.

Programmable Clock Generators & Synthesizers: On the HTG-V6HXT-100G board, there are two low jitter Silicon Labs crystal Si570 and two synthesizers Silicon Labs Si5368 which supply 6 pairs of clock for both sides of GTH. Therefore, each GTH Quad has a dedicated clock source. The Si570 and Si5368 are adjustable through I2C bus.

#### 3.3.1.2 HTG-SFP-PLUS-MDL Extender Card:

The HTG-SFP-PLUS-MDL extender card is designed for hosting 12 SFP+ optical transceiver ports. The card is designed matching with HTG-V6HXT-100G platform. It is supported by re-timers with Electrical Dispersion Compensation Puma AEL2006. Figure 3-8 shows the block diagram of one AEL2006 and 2 SFP+ on the HTG-SFP-PLUS-MDL. AEL2006 is a bidirectional dual-channel 10Gbps Ethernet transceiver with EDC; it contains integrated EDC circuits targeted for 10Gbps SFP+ applications. AEL2006 also contains a MDIO interface for device control and configuration.

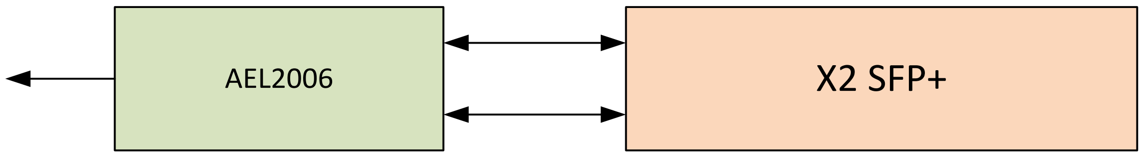


**Figure 3-8: HTG-SFP-PLUS-MDL partial block diagram**

Totally, there are six AEL2006 that support 12 SFP+ on the extender card.

#### 3.3.1.3 SFP+ Optical Transceiver

The small form-factor pluggable (SFP) is a compact, hot-pluggable transceiver used for both telecommunication and data communications applications. The SFP+ is an enhanced version of the SFP. It supports data rates up to 10Gbps.

Figure 3-9 shows the SFP+ modules which are used in the project. For the communication link between the central controller and the node, Avago’s 10Gbps 1310nm 10GBASE-LR SFP+ optical transceiver for SMF 10km links is used. The optical interface specifications are compliant with IEEE 802.3ae, and the Electrical interface specifications are compliant with SFF 8431 Specifications.

For the node transmission of bursts, two Gigalight’s 10Gbps 10GBASE-ER/EW SFP+ optical transceivers for SMF 80km reach are used. The transceivers are compliant with SFF-8413 and IEEE802.3ae.The wavelengths of the 10GBASE-ER SFP+ optical transceivers are: λ1=1544.53nm and λ2=1546.12nm.

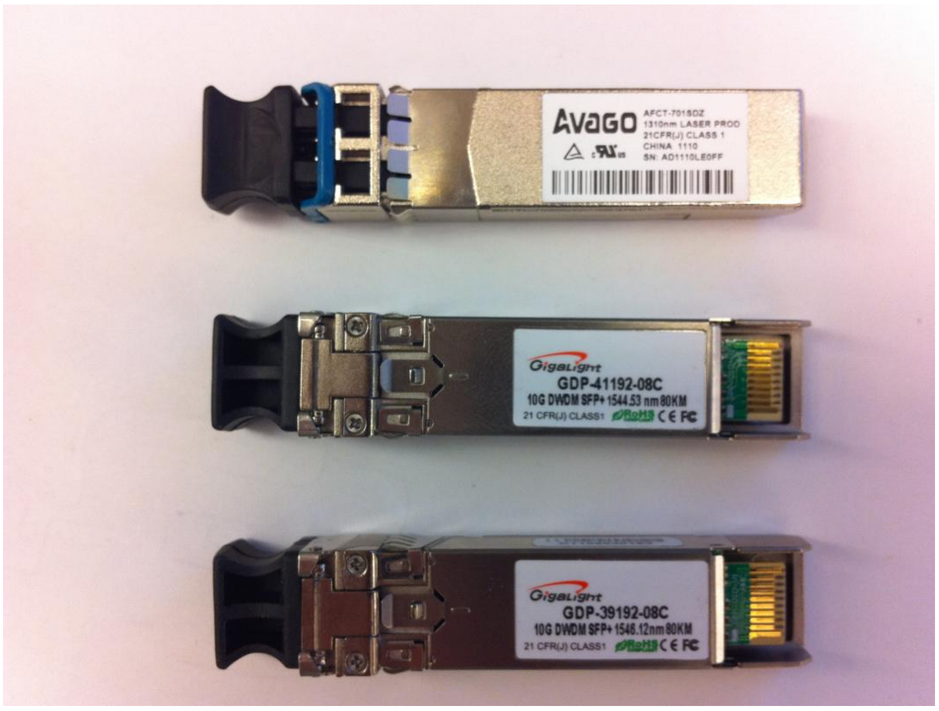


Figure 3-9 SFP+ Optical Transceivers

### 3.3.2 IP Components

As described in Table 3-2, Xilinx IP cores LogiCORE 10-Gigabit Ethernet MAC and Xilinx GTH Transceivers are instantiated in the projects. Other Xilinx IP cores, such as Xilinx Mixed-Mode Clock Manager (MMCM) and Xilinx Chipscope Pro are also used.

#### 3.3.2.1 Xilinx GTH Transceivers

XC6VHX380T-2FF1923 FPGA offers 24 GTH transceivers that run up to 11.4 Gbps. The total 24 GTH Lanes are split to 6 Quads, each quad shares one dedicated REFCLK, but each lane can be configured with different line rates that are integer multiples of each other. Xilinx Core Generator tool supplies a wizard to automatically configure GTH transceivers. Figure 3-10 shows a block diagram of one GTH Lane, which contains PMA, PCS and PCS to Fabric interface. A GTH quad contains four GTH lanes, a PLL, and resources for controller and initializing the Quad.

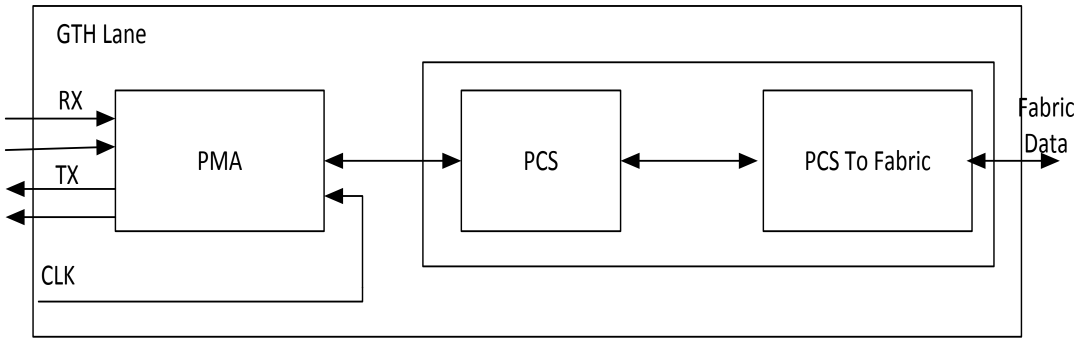


Figure 3-10: GTH Lane block diagram

In the project, 1 GTH Quads (4 GTH Lanes) are instantiated, 3 GTH lanes for the node's ingress/egress functions, and 1 GTH lane for updating LUT. Each lane works on its own clock domain. The RX clock data recovery circuit in each GTH transceiver extracts the recovered clock and data from an incoming data stream. To avoid losing any data, the design needs to include generating enough K-Characters before data streams. Figure 3-11 shows the test method of measuring the K Characters needs before the burst and between adjacent Ethernet frames. After the test on board, totally 136x8 Bytes K-Characters need to be put before a burst, something that limits number of time-slices to 91 Time-sliceinstead of 100 Time-slice within a TSON frame.

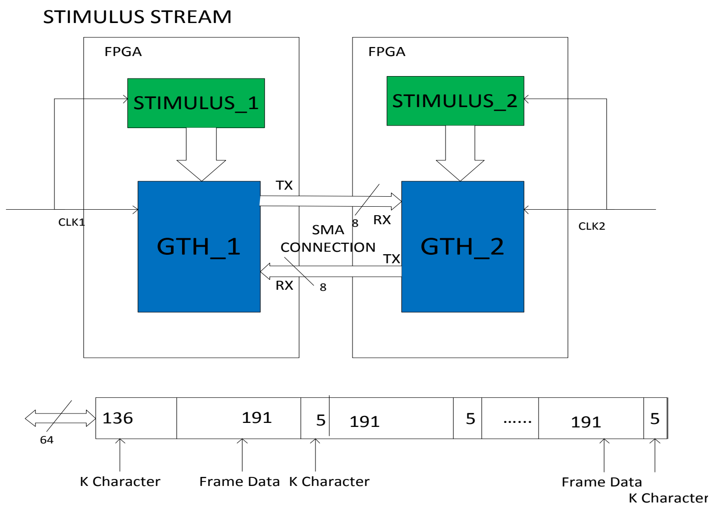


**Figure 3-11: GTH K-Characters Insertion Test and Result**

Furthermore, for the clock-recovery of GTH, when there is no burst/data to transfer, the node continuously sends K-Characters for transmission as shown in Figure 3-12. Proposing and implementing this transport scheme, there is no need for high-speed transient suppression EDFA and burst mode receivers (commercially available SFP+ transceivers are used), something that considerably minimizes implementation complexity and cost as well as allows for easier bit-rate scalability.

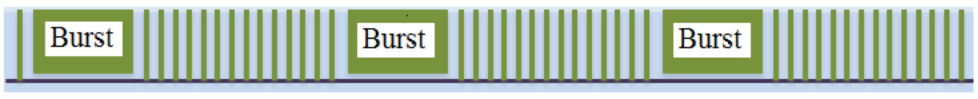


**Figure 3-12: ContinuousTime-slice Data Transmission**

#### 3.3.2.2 LogiCORE 10-Gigabit Ethernet MAC

The Xilinx IP 10-Gigabit Ethernet MAC core is a single-speed, full-duplex 10 Gbps Ethernet Media Access Controller. It is designed to meet 10-Gigabit Ethernet specification IEEE 802.3-2008 and supports Normal Frame Transmission, Transmission with In-Band FCS Pass, Back-to-Back Transfers, Transmission of Custom Preamble and VLAN Tagged Frames.

The implementation of the Core with User Logic on PHY interface is shown in Figure 3-13. The Transmit/Receive Engine interface is for transmitting/Receiving data; the Reconciliation Sublayer interface is for processing XGMII local fault and remote fault messages; the flow control interface is used to set the flow control frames from the core;

the management interface is for configuring the core and accessing to the statistics block; if the management interface is not chosen when initializing the core, the configuration and status vectors are exposed by the core; the MDIO interface is for controlling the outside devices.

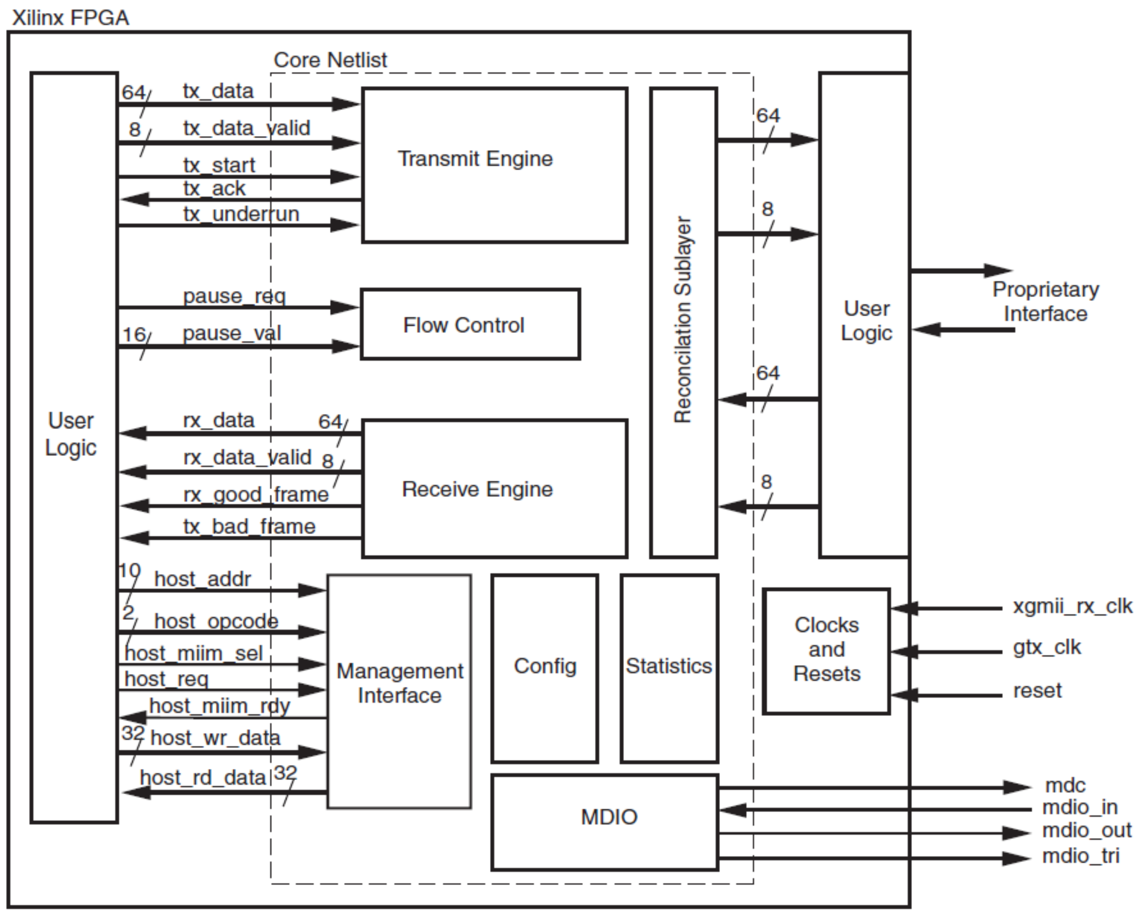


**Figure 3-13: Implementation of MAC core [4]**

In the project, the MAC core is generated with internal FPGA interface to PHY layer. In total, there are two MAC cores used. One is for ingress function processing input Ethernet packets and egress function reforming output Ethernet packets. And the other one is for GMPLS-to-FPGA interface link. The latter one is generated with MDIO interface which is used for controlling and configuring HTG-SFP-PLUS-MDL extender card.

#### 3.3.2.3 Xilinx Mixed-Mode Clock Manager (MMCM)

The MMCM primitive in Virtex-6 parts is used to generate multiple clocks with defined phase and frequency relationships to a given input clock. It is featured with configurable BUFG insertion and supporting MMCM_BASE and MMCM_ADV features.

The user defined MAC_CLK block, which supplies the transmit clock to the 10Gbps Ethernet MAC core is shown in Figure 3-14.

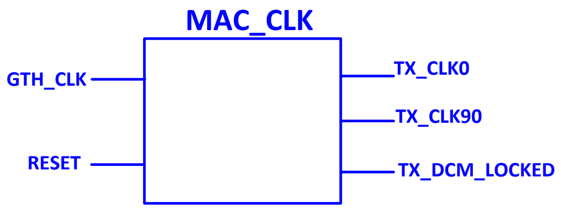


**Figure 3-14: MAC_CLK Top Level View**

The MAC_CLK block contains one MMCM and three BUFG XILINX primitives. The input clock is derived from GTH quad clock. TX_CLK0 and TX_CLK90 are output clocks. TX_DCM_LOCKED goes high when the output clocks are stable. Figure 3-14 shows the top level view of MAC_CLK block.

#### 3.3.2.4 Xilinx Chipscope Pro

Xilinx ChipScope Pro is a powerful on-chip debug tool. When starting the on board debugging stage, it enables inserting logic analyser and virtual I/O low-profile software cores directly into the design. Xilinx ChipScope Pro allows viewing any internal signals in the ChipScope Analyzer tool.

The Signals can be captured in the system at the speed desired. The virtual I/O canvirtualize the input/output through ChipScope Analyser tool. Therefore, it frees up pins for the hardware pin usage. All the captured signals can be easily analysed.

### 3.3.3 User Logic Components

#### 3.3.3.1 I2C Controller

I2C is a multi-master serial single-ended computer bus; it is used to attach low-speed peripherals to an electronic device. In the project, I2C controller is needed to program I2C programmable XO/VCXO SI570 and synthesizer SI5368. The I2C bus consists of a

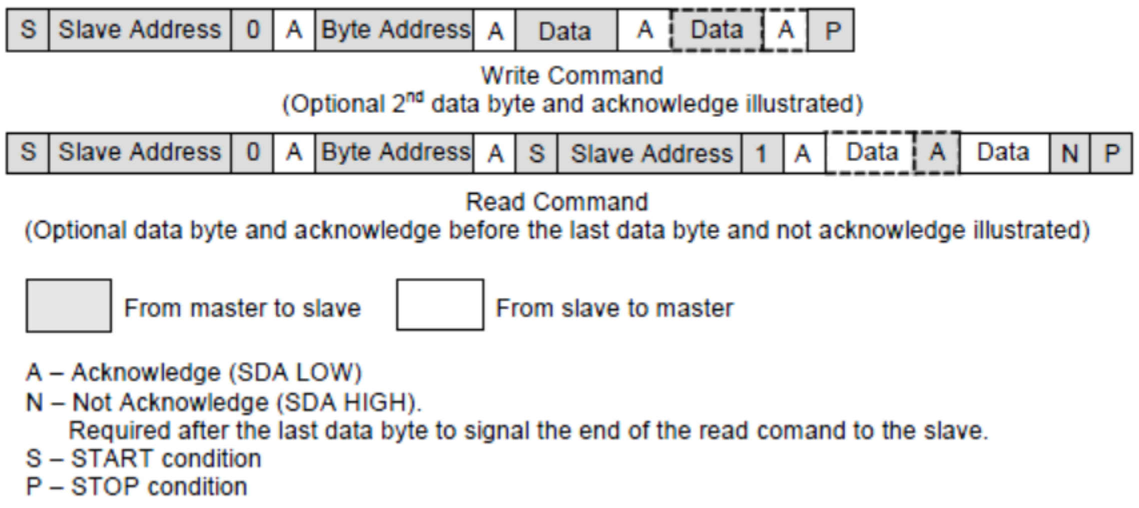

S
Slave Address
0
A
Byte Address
A
Data
A
Data
A
P
Write Command
(Optional 2nd data byte and acknowledge illustrated)
S
Slave Address
0
A
Byte Address
A
S
Slave Address
1
A
Data
A
Data
N
P
Read Command
(Optional data byte and acknowledge before the last data byte and not acknowledge illustrated)
From master to slave
From slave to master
A – Acknowledge (SDA LOW)
N – Not Acknowledge (SDA HIGH).
Required after the last data byte to signal the end of the read comand to the slave.
S – START condition
P – STOP condition

**Figure 3-15: I2C command format for both read and write access [5]**

bidirectional serial data line (SDA) and a serial clock input (SCL).

Figure 3-15 shows I2C command format for both read and write access. To configure SI570 and SI5368, Silicon Lab supplies the tool called Clock Builder to generate the register configuration tables.

I2C controller can also be used for UART control and communication with the computer.

#### 3.3.3.2 MDIO controller

Management Data Input/Output (MDIO) is a serial bus defined for the Ethernet family of IEEE 802.3 standards for Media Independent Interface. The MDIO bus provides access to the configuration and status registers of PHY.

In the project, MDIO controller is used to initialize AEL2006 on SFP+ board. Figure 3-16 shows an MDIO transaction. Each transaction is 64 MDC clock cycles long and contains 32 cycles preamble, 2 cycles start of frame, 2 cycles op code, 5 cycles port address, 5 cycles device address, 2 cycles turnaround and 16 cycles of address/data.

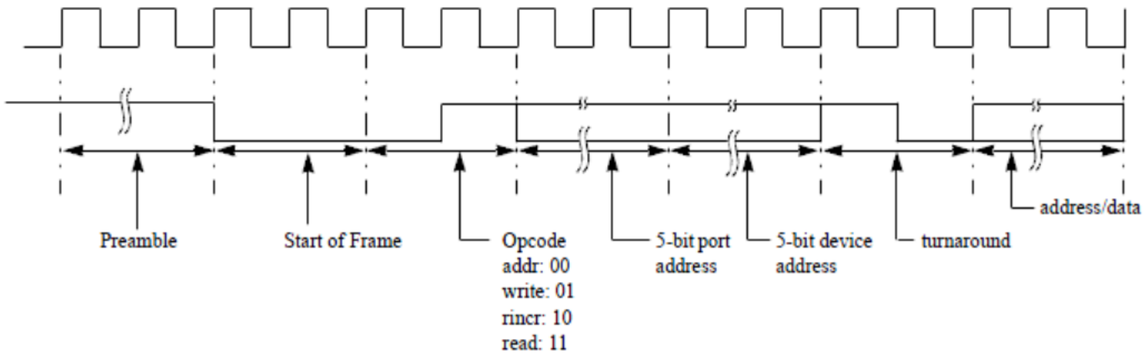


**Figure 3-16: MDIO transaction format**

#### 3.3.3.3 Look-Up-Table (LUT) Update Block:

The LUT UPDATE block is employed in GMPLS-to-node communication module shown in Figure 3-17. The LUT Update block contains a LUT block for Time-slice Allocation and PLZT switching, and an Update block for updating the register files in the AGGREGATION block.

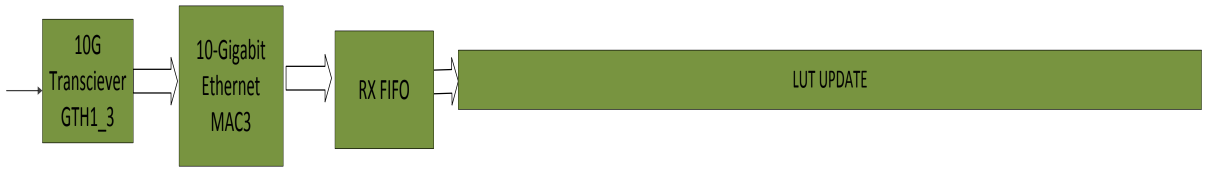


**Figure 3-17: GMPLS-to-node Link Layer 2 Function Block Diagram**

As shown in Figure 3-17, the GMPLS and FPGA communicate through 10Gbps link. |Central controller calculates the time-slice allocation and PLZT switch information, sends eight Ethernet packets through SFP+, and then updates the LUT in the FPGA design. Other components, such as PLZT controller and aggregation modules, work under the instruction of the LUT Update block. Table 3-3 shows the LUT address map.

| LUT Address Map (Bytes) | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Address | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | Wavelength 1 Time Slice Allocation(91 bits) | | | | | | | | | | | |
| 1 | Wavelength2 Time Slice Allocation (91 bits) | | | | | | | | | | | |

| | |
|---|---|
| 2 | Wavelength 1 PLZT Switch Information, based on Time-slice Allocation 1-12 |
| 3 | Wavelength 1 PLZT Switch Information, based on Time-slice Allocation 13-24 |
| 4 | Wavelength 1 PLZT Switch Information, based on Time-slice Allocation 25-36 |
| 5 | Wavelength 1 PLZT Switch Information, based on Time-slice Allocation 37-48 |
| 6 | Wavelength 1 PLZT Switch Information, based on Time-slice Allocation 49-60 |
| 7 | Wavelength 1 PLZT Switch Information, based on Time-slice Allocation 61-72 |
| 8 | Wavelength 1 PLZT Switch Information, based on Time-slice Allocation 73-85 |
| 9 | Wavelength 1 PLZT Switch Information, based on Time-slice Allocation 86-91 |
| 10 | Wavelength 2 PLZT Switch Information, based on Time-slice Allocation 1-12 |
| 11 | Wavelength 2 PLZT Switch Information, based on Time-slice Allocation 13-24 |
| 12 | Wavelength 2 PLZT Switch Information, based on Time-slice Allocation 25-36 |
| 13 | Wavelength 2 PLZT Switch Information, based on Time-slice Allocation 37-48 |
| 14 | Wavelength 2 PLZT Switch Information, based on Time-slice Allocation 49-60 |
| 15 | Wavelength 2 PLZT Switch Information, based on Time-slice Allocation 61-72 |
| 16 | Wavelength 2 PLZT Switch Information, based on Time-slice Allocation 73-85 |
| 17 | Wavelength 2 PLZT Switch Information, based on Time-slice Allocation 86-91 |

**Table 3-3: Design Look-Up-Table Address Map**

#### 3.3.3.4 PLZT Switch Control Block

PLZT Switch is described in the previous section. The PLZT switch control block is designed to control the PLZT switches. Currently, the design supports the control of two 4x4 PLZT switches. As shown in Figure 3-18, the traffic is continuous with K-Characters between burst. The timing diagram is shown in Figure 3-18. It is important to switch the PLZT on-time to avoid chopping any data or K-Characters.

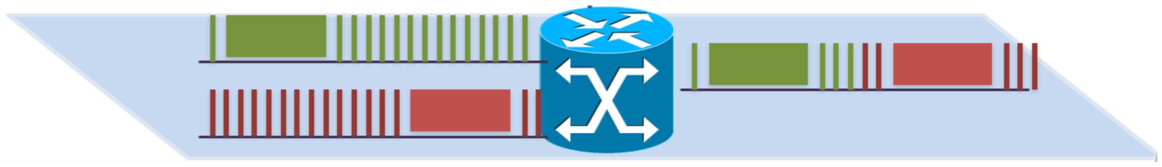


**Figure 3-18: Burst Transport through PLZT Switch**

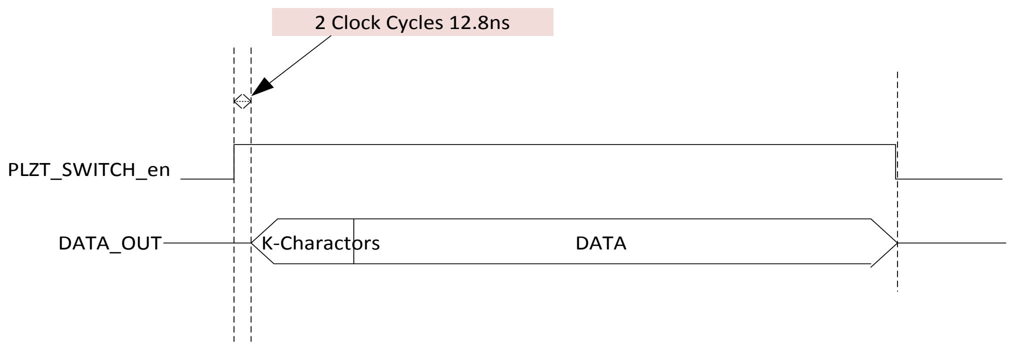


**Figure 3-19: PLZT Switch Control Block Timing Diagram**

As illustrated in Figure 3-19, to allow PLZT taking time to switch and to avoid losing K-Characters, the design leaves 2 clock cycles' gap between PLZT switch enable and send out the data. Also because PLZT switch takes time to switch, PLZT_SWITCH_en goes down immediately after finishing sending the data in case of PLZT configuration change.

#### 3.3.3.5 Aggregation block

Aggregation block is a functional block in the node for the ingress function, as shown in Figure 3-20. Apart from Transceivers and Ethernet MAC, the flow when TSON Metro Node receives the Ethernet packets in the RX FIFO is as below:

When RX FIFO receives good Ethernet frames, it sends them to the DEMUX.

The DEMUX parses the information (i.e. destination MAC address, source MAC address and etc.) from the packets and put them into different FIFOs. Currently, the design implementation is based on different destination MAC address.

According to the Register File (LUT information) of the Time-slice Allocation, the aggregation block calculates the frames needed to construct the burst, aggregates the frames, waits for the valid Time-slice, adds necessary K-Characters, and then reads the Ethernet frames from the FIFOs and finally sends the burst into different wavelength FIFO.

TX_FIFO_LAMDA1 and 2 adjust the clock with PLZT switch controller and send out the burst.

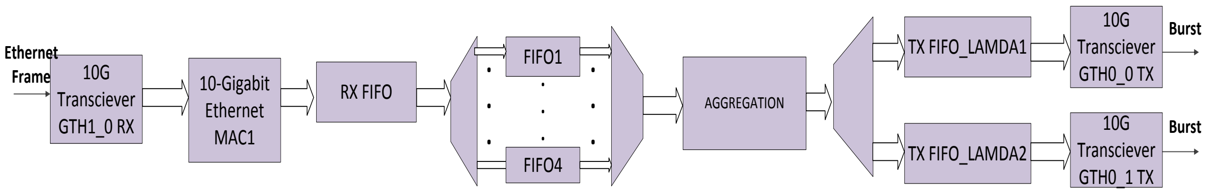


**Figure 3-20: Layer 2 Ingress Function block diagram**

One purpose of TX FIFO LAMDA1&2 is for cross-clock-domain data.

#### 3.3.3.6 Segregation Block

Segregation block is a functional block of TSON Metro Node as an Egress function. As shown in Figure 3-21, the flow from RX FIFO Lamda 1 and Lambda 2, receiving burst until TX FIFO transmits out the Ethernet frames is as follows:

When receiving the burst in RX FIFO Lamda 1 or Lamda 2, the burst order is recorded. Whenever receive 2 bursts from different wavelength at the same time, the one from lamda1 is set as the first one.

The SEGREGATION block reads the burst from RX FIFO according to the recorded burst order, segregate the burst to Ethernet frames and put them to the TX FIFO.

TX FIFO notifies 10 Gbps Ethernet MAC that the Ethernet packets are ready; the Ethernet packets will be sent out through 10 GE MAC and GTH Lane.

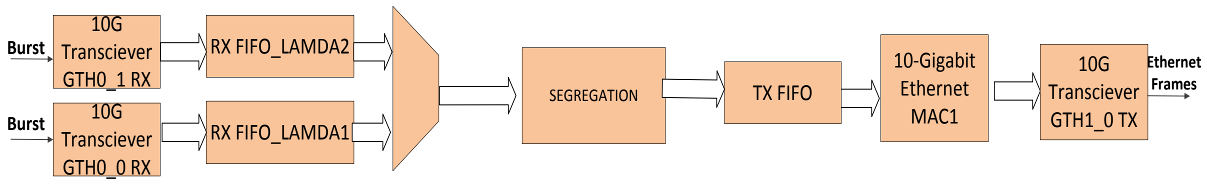


**Figure 3-21: Layer 2 Egress Function block diagram**

In the design, TX FIFO is mainly used for cross-clock-domain signals.

## 3.4 Incremental Test Results (both simulation and experiment)

### 3.4.1 Simulation Result about LUT and Register File Update

The simulation result is collected using Mentor Graphics Modelsim. Modelsim is a unified debug environment for Verilog, VHDL and SystemC. It is a very popular hardware simulation and debug tool. Apart from the node design described in Subsection 3.3.3, a testbench, which includes a stimulus block to generate Ethernet frames and a check block to check the output data, was written in VHDL. All the signals can also be checked through Modelsim waveform windows. The testbench clock frequency is 156.25MHz, the clock cycle is 6.4ns.

When central controller updates the Switch and Time-slice Allocation table in the FPGA, as described in Section 3.3.3.3, the LUT Update block updates the Register File in the Aggregation block, and then with this information, the Aggregation block would be able to transmit the burst in the allocated Time-slice.

The simulation result in Figure 3-22 illustrates the clock cycles it takes for updating the LUT and Register File in an idle occasion when the burst has been received and time-slice is allocated. The simulation results which start from central controller finishes updating LUT, includes Aggregation Module starts to update the Register File, Register File updates finished, until the aggregation module begins to send out the burst. The specifications of this timing in different periods, (from LUT receiving, until the update of the register file, and from register file until the transmission) are cited in the following:

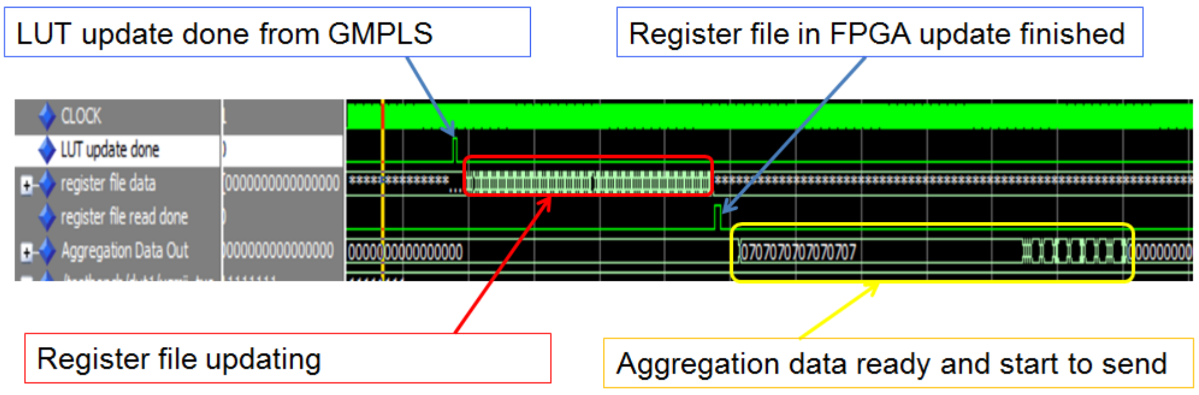


**Figure 3-22: Simulation Result 1: LUT Update**

From “GMPLS update finished” to “Register file updated”: **6 clock cycle delay**+ LUT length clock cycles.

From “Register file update finished” to “Aggregation start to send burst”: **12 clock cycles delay**

Total delay: (**18** + LUT LENGTH)***6.4ns**.

### 3.4.2 Implementation Results

All the modules in the project were developed in VHDL which were synthesized, implemented by Xilinx ISE software. Xilinx ISE generated the bit files, and the bit files were downloaded to the FPGA on board through JTAG. The Xilinx XC6VHX380T-2FF1923 Device utilization information for one TSON Metro Node is summarised in Table 3-4.

| Device Utilization Summary | | | |
|---|---|---|---|
| **Slice Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slice Registers | 27632 | 478080 | 5% |
| Number of Slice LUTs | 17,365 | 239040 | 7% |
| Number of fully used LUT-FF pairs | 11,369 | 48764 | 23% |
| Number of bonded IOBs | 89 | 720 | 12% |
| Number of Block RAM/FIFO | 258 | 768 | 33% |
| Number of BUFG/BUFGCTRLs | 28 | 32 | 87% |

**Table 3-4: Device Utilization Summary**

Figure 3-23 shows the hardware connection for measuring the implementation results. Anritsu MD1230B is an Ethernet/IP network data analyser. It is used as the Ethernet frames traffic generator and it is also used to analyse the latency and jitter of the received Ethernet packets. When the FPGA node receives the Ethernet packets, it aggregates them and sends them out as optical time-sliced data sets (bursts). The bursts travel back to the FPGA through a fibre patchcord. When egress part receives the optical traffic bursts, it segregates the burst, extracts the Ethernet frames, and sends out in 10GE links , which can then be analysed by MD1230B. In the following, several performed tests for measuring and analysing the implementation results are discussed.

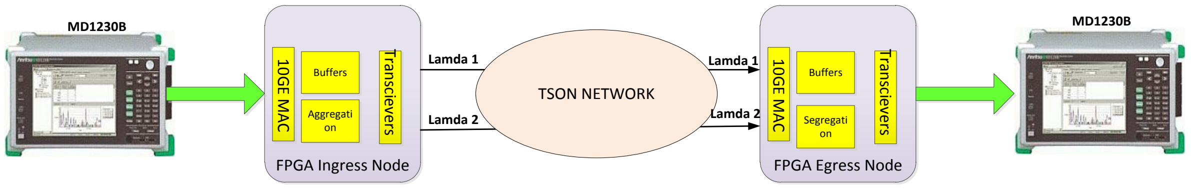


**Figure 3-23: Hardware Measurement Connection Diagram**

For all the experiment, the test results were taken based on min and max Ethernet packets lengths: 64 bytes and 1500 bytes.

### 3.4.2.1 TSON Metro Node Experiment and Measurement: Bit Rate

The first experiment is to test the maximum Ethernet frame bit rate TSON Metro node is capable to handle without losing any frames. It is supposed and set that all the time-slice allocation is available.

For Ethernet stream with packet length 64 bytes:

- The maximum Ethernet speed is **7.619Gbps**;
- The maximum TSON speed is **7.191Gbps**.
- The utilization is **94.38%.**

For Ethernet stream with packet length 1500 bytes:

- The maximum Ethernet speed is **9.868Gbps**;
- The TSON node maximum theoretical speed/throughput is **9.1 Gbps** (91 slots/frame due to overheads)
- The maximum experimentally measured speed for TSON is **8.68Gbps**.
- The utilization is **95.38%.**

#### 3.4.2.2 TSON Metro Node Experiment and Measurement: Time-slice Overhead

This experiment is to measure time-slice overhead, because some pre-amble K-Characters are used for clock recovery purpose before sending out each burst and between every adjacent Ethernet packets. time-slicetime-sliceThe number of the needed time-slices mainly depends on the Ethernet Frame Length. For example, if 16 K-Characters exist between every two Ethernet Packets, then for Ethernet frame with 64 bytes, the overhead is 16/64= 25%, but for Ethernet frame with 1500 bytes, the overhead is 16/1500=1%. The implementation results are show in Figure 3-23 and Figure 3-24.

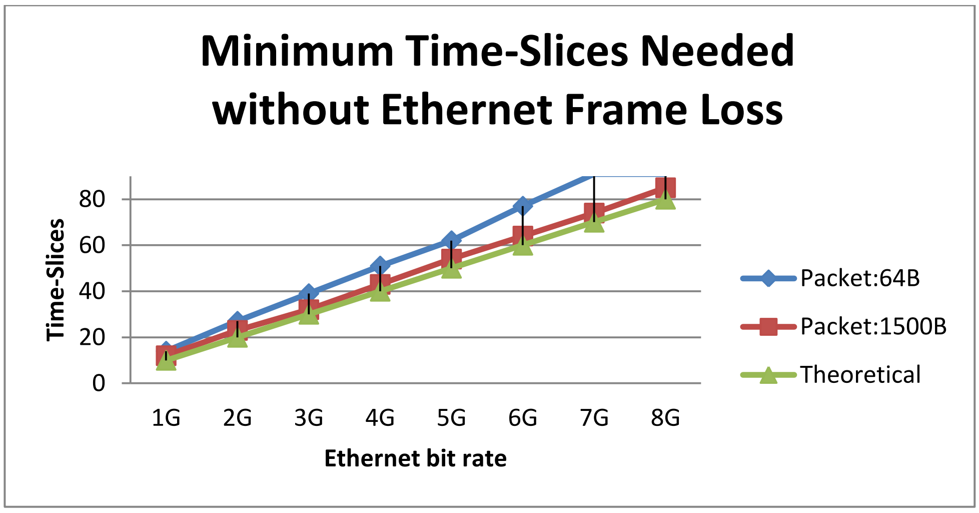


**Figure 3-24: FPGA Measured Results: Minimum Time-slices Needed without Ethernet Frame Loss**

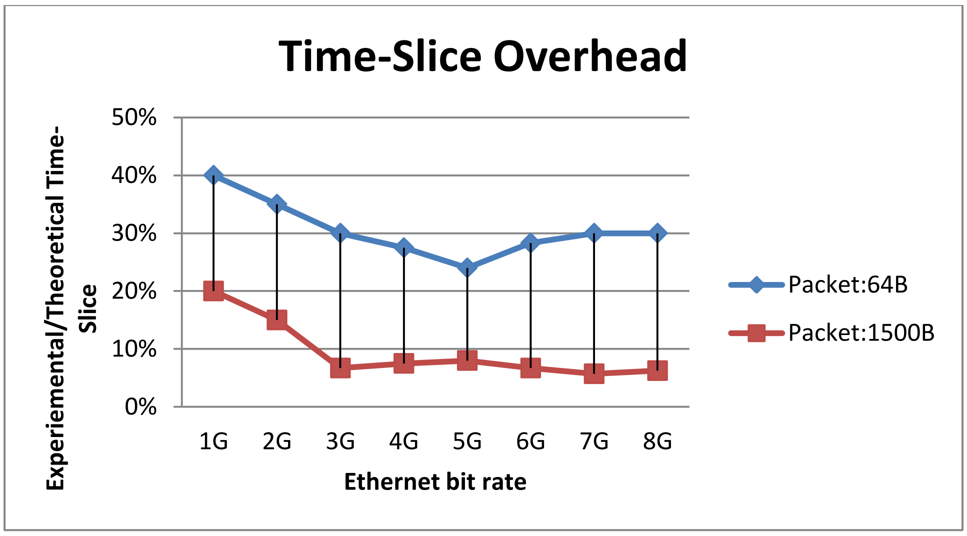


**Figure 3-25: FPGA Measured Results: Time-slice Overhead**

The Figure 3-24 shows the minimum number of Time-slice needed without any Ethernet Packet lost. The Figure 3-25 shows the same result as Figure 3-24 but calculated as Time-slice overhead. From both of the figures, another factor, which affects the Time-slice overhead, is the input Ethernet frame bit rate. When the bit rate is lower than 3Gbps, the Time-slice overhead is high, but when the bit rate is above 3G, the Time-slice stays comparably stable.

### 3.4.2.3 Experiment and Measurement: Maximum Contiguous unallocated Time-slice

This aim of this experiment is to measure the maximum contiguous unallocated Time-slice the TSON node can handle without any Ethernet Packets lost. Because of the limited capacity of FPGA on-chip RAM, current TSON FPGA design uses 131K RAM as rx_fifo buffer and 524K RAM as aggregation buffer. It is able to hold maximum 6.5 Time-slice data. When Ethernet traffic comes in continuously, if the Time-slice is contiguous unallocated, the buffer in FPGA will overflow and it will cause the Ethernet frame loss. Therefore, to prevent the Ethernet frame loss, the maximum number of contiguous unallocated time-slices is measured.

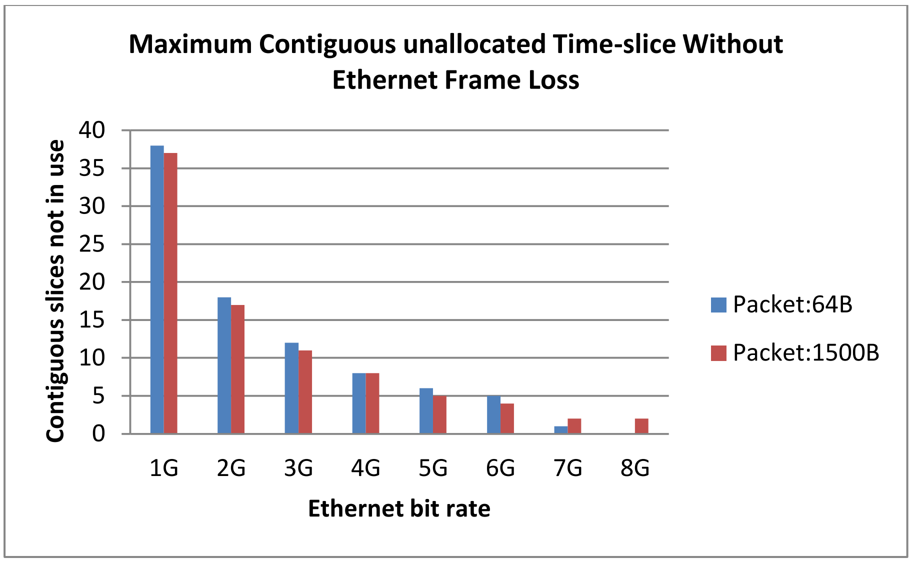


**Figure 3-26: FPGA Measured Results: Contiguous '0'**

Figure 3-26 shows the maximum contiguous unallocated Time-slices without any Ethernet frame losses. When the bit rate is 1Gbps, the maximum number of contiguous unallocated Time-slices is 38, and then decreases with bit-rate increase. When the bit rate goes up to the maximum bit rate measured in Section 3.4.2.3, all the Time-slices need to be allocated.

### 3.4.2.4 Experiment and Measurement: Latency

The purpose of this experiment is to measure the Latency of the node and to analyse the factors that affect the latency. The latency is measured by MD1230B. The path is from MD1230B generates the Ethernet packets, the TSON node aggregates the packets and sends out the burst, the burst loops back to the TSON node for the segregation, to the TSON node send the Ethernet packets to MD1230B. To analyse the factors that affect the latency, the latency was measured in different cases with different bit rates, Time-slice allocation and Ethernet Packet Size.

Figure 3-27 and Figure 3-28 show the measured latency results of Ethernet packet size 64B and 1500B when using one wavelength. In both 64B and 1500B Latency results, Time-slice Allocation1 makes all the time-slices available; Time-slice Allocation2 spreads the time-slices equally into the frame; Time-slice Allocation3 gathers the most possible numbers of time-slices together. However, as described in the previous experiments, the minimum Time-slice allocation and the maximum contiguous unallocated Time-slice allocation are different based on different Ethernet packet size and bit rates. So the Time-slice Allocation2 and Time-slice Allocation3 are set at different patterns.

Here, it presents an example of three Time-slice allocations that are used for the measurement at 1Gbps for both 64B and 15000B. The allocation is based on the measurement results 2 and 3.

In case for Packet Size 64B, Ethernet Bit rate 1Gbps, Time-slice Allocations are like below (91 Time slices, '1' indicates valid Time-slice, '0' indicates invalid Time Slice):

Time-slice Allocation 1:
111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111

Time-slice Allocation 2:
000001000001000001000001000001000001000001000001000001000001000000010000
0010000001000001000

Time-slice Allocation 3:
000000000000000000000000000000000001111111000000000000000000000000000000
0000000001111111000

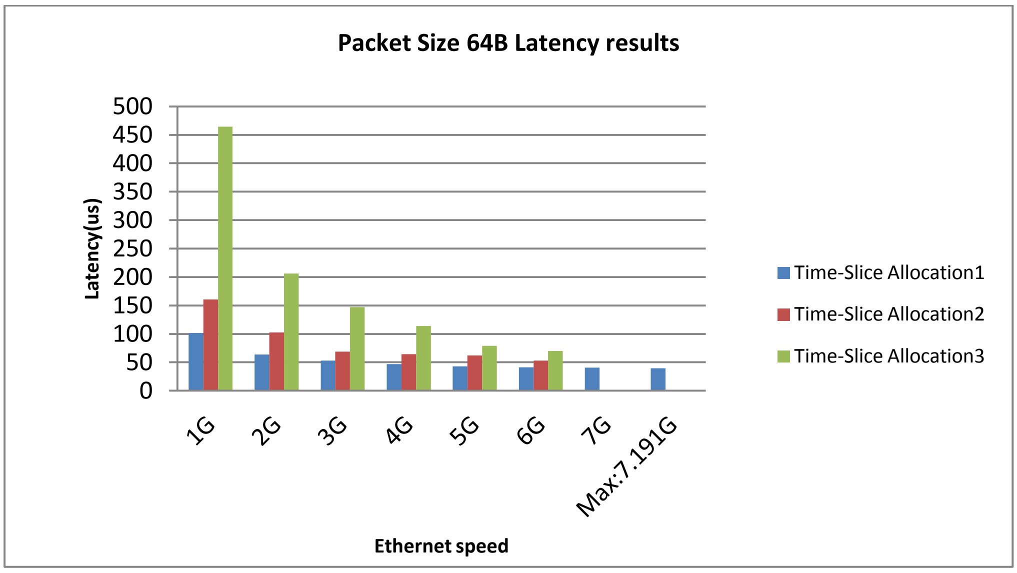


**Figure 3-27: FPGA Measured Results: Latency (Packet Size 64B)**

In Figure 3-27, it shows that, in any case, Time-slice Allocation 1 and 2 have low latency, when the bit rate is low, allocate as many Time-slice as possible together, the latency of Time-slice Allocation 3 is more than double of Time-slice Allocation 1/2. However, for the high bit rate like 8Gbps, the latency of Time-slice Allocation 2 and 3 are similar.

In case for Packet Size 1500B, Ethernet Bit Rate1Gbps, Time-slice Allocations are like below (91 Time slices, '1' indicates valid Time-slice, '0' indicates invalid Time Slice):

Time-slice Allocation 1:

111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111

Time-slice Allocation 2:

000000100000010000001000000100000010000001000000100000010000000010000000
1000000100000001000

Time-slice Allocation 3:

0000000000000000000000000111100000000000000000000000000111100000000000000
0000000000011110000

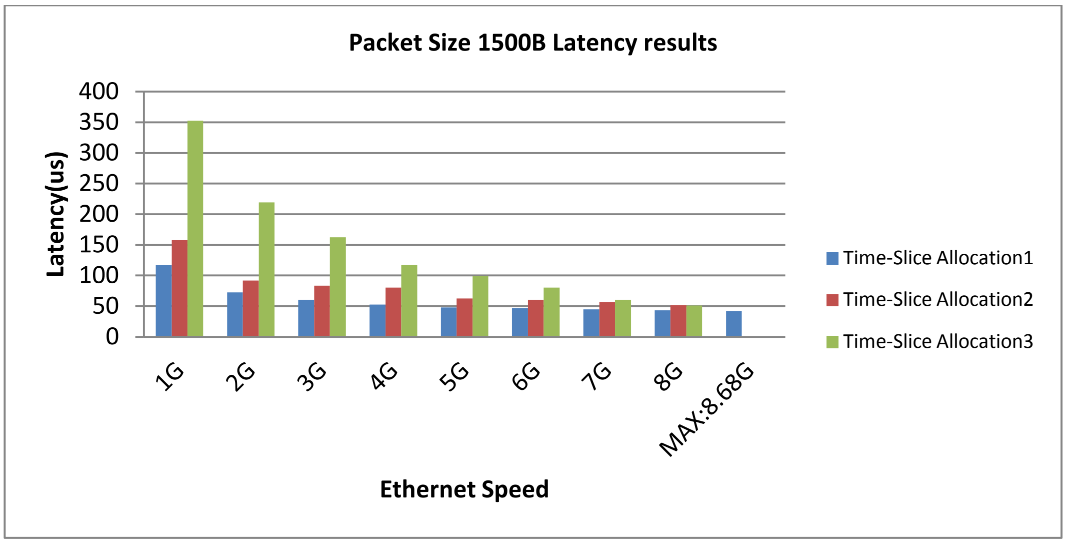


**Figure 3-28: FPGA Measured Results: Latency (Packet Size 1500B)**

The latency shown in Figure 3-28 is similar to the Figure 3-27. For the low bit rate, Time-slice Allocation 3 is much higher than Time-slice Allocation 1 and 2, but for high bit rate, the difference is little.

It is also interesting to measure the latency when allocate the Time-slice in different wavelength. The Figure 3-29 shows the latency results when allocate Time-slice on different wavelength. For example, for Ethernet packet size of 1500B, at bit rate 1Gbps, the utilization of Wavelength1 is 25% and wavelength2 is 75%. The detailed Time-slice allocation is as below, (91 Time-slice, '1' indicates valid Time-slice, '0' indicates invalid Time Slice):

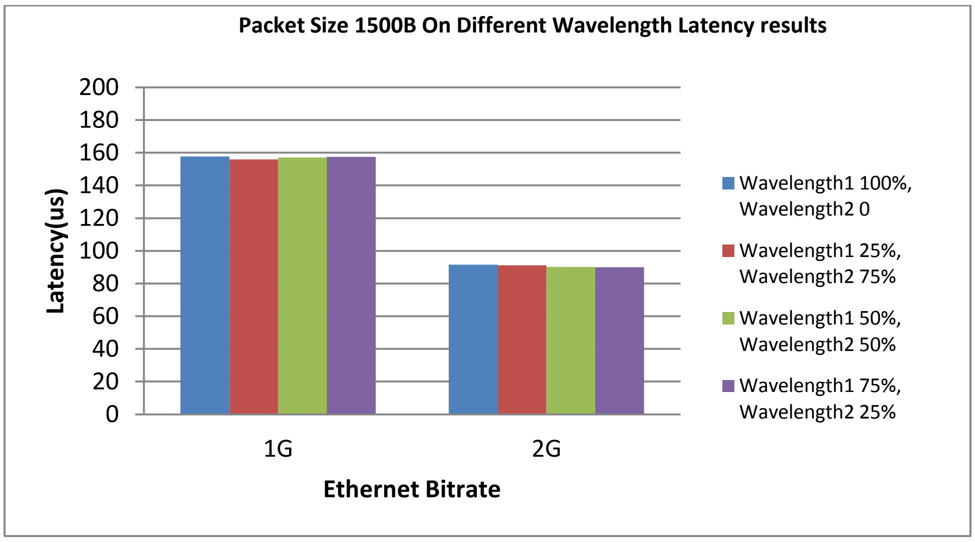


**Figure 3-29: FPGA Measured Results: Latency when use Different Wavelengths**

Wavelength 1: 0000001000000000000001000000100000010000000000000100000010000000010000000 100000000000000100

Wavelength 2: 0000000000000100000000000000000000000000010000000000000000000000000000000 000000010000000000

The measured latency result in Figure 3-29 shows the latency results are similar when allocate time-slice in different wavelengths. It indicates that there is little impact on switching between different wavelengths.

#### 3.4.2.5 Experiment and Measurement: Jitter

The purpose of this experiment is to measure the Jitter of the node and to analyse the factors that affect the Jitter. The Jitter is measured by MD1230B. The path is the same as last experiment when measuring latency.

The Figure 3-30, Figure 3-31, Figure 3-32 show, with the input Ethernet Packet Size 64, for different Time-slice Allocation, the percentage of packets that are not received in 1us. The time-slice Allocation1, 2, 3 are the same as last experiment mentioned in Section 3.4.2.4.

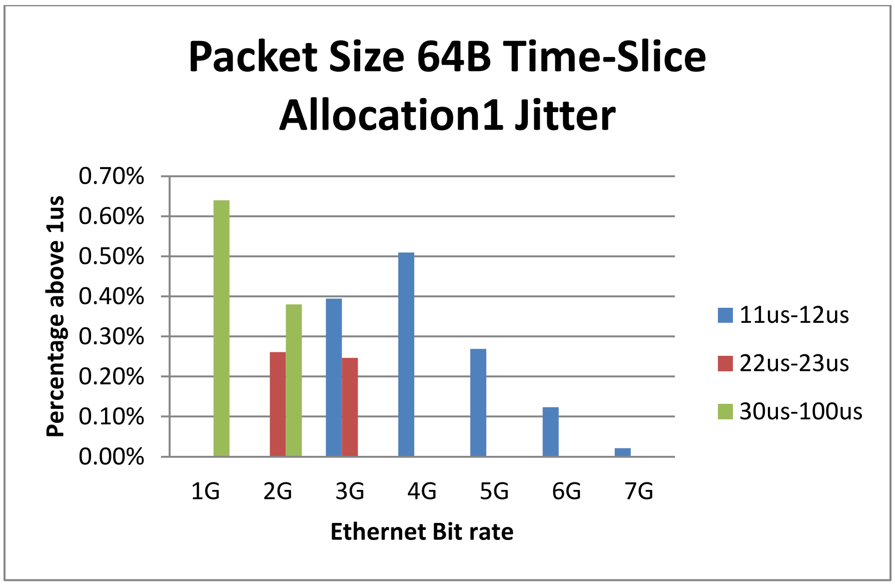


**Figure 3-30: FPGA Measured Results: Jitter for Frame Size 64B, Time-slice Allocation1**

In Figure 3-30, for Time-slice Allocation1, when the bit rate is low, the packets that not received in 1us have high latency. As the figure shows, with the increase ni the bit rates, the jitter of arrived packets reduces. It should be noted that this chart does not show jitters below 1us.

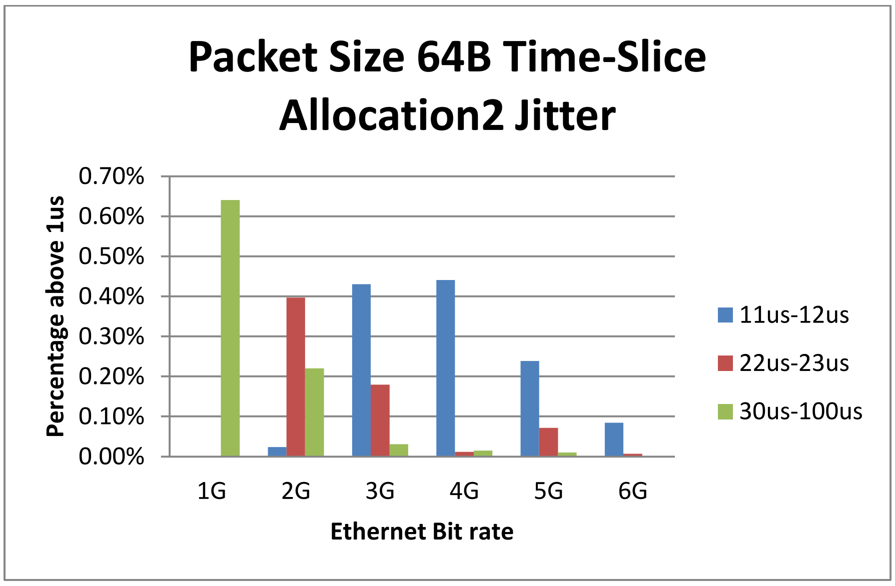


**Figure 3-31: FPGA Measured Results: Jitter for Frame Size 64B, Time-slice Allocation2**

As shown in Figure 3-31, similar to Figure 3-30, the jitter of Time-slice Allocation2 is also impacted by the bit rate.

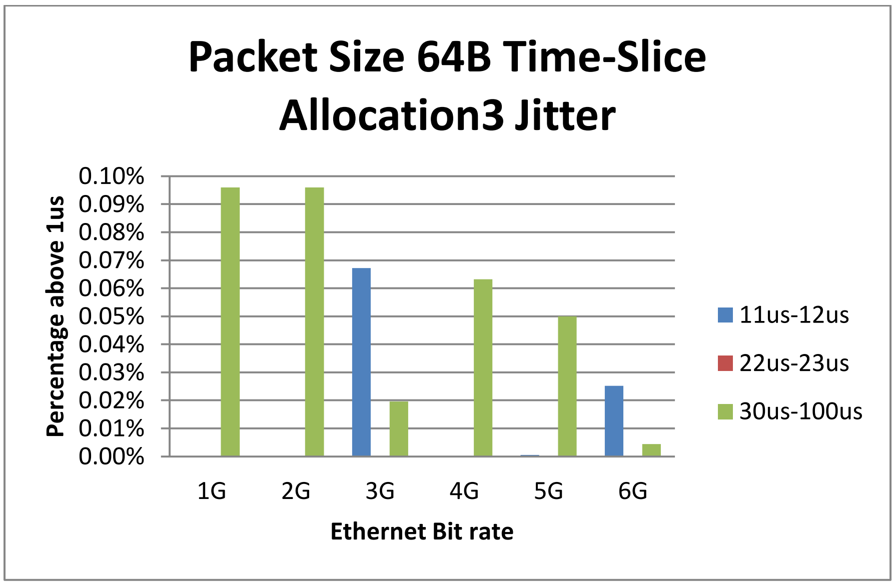


**Figure 3-32: FPGA Measured Results: Jitter for Frame Size 64B, Time-slice Allocation3**

Figure 3-32 shows the Jitter result of Time-slice Allocation3. Compared with Figure 3-30 and Figure 3-31, the frames t with less than 1 μs jitter.

Calculating the jitter results of Figure 3-30, Figure 3-31, Figure 3-32, 99.35% of packets are received within 1μs jitter.

The experiment results of Figure 3-33, Figure 3-34, Figure 3-35, are measured with the input Ethernet Frame Size 1500B, for different Time-slice Allocation. The results are based on the percentage of frames that are not received within 2μs jitter. The time-slice Allocation1, 2, 3 are the same as last experiment of 1500B mentioned in Section 3.4.2.4.

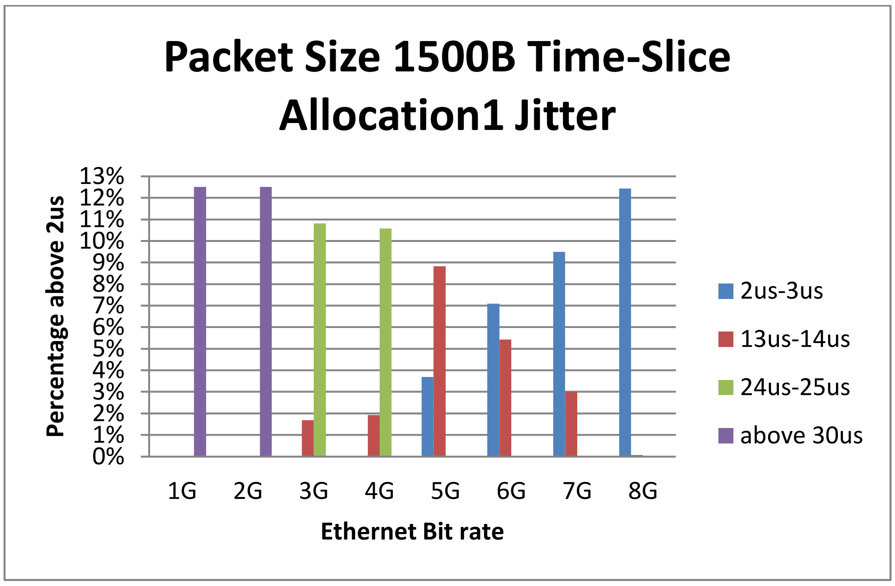


**Figure 3-33: FPGA Measured Results: Jitter for Frame Size 1500B, Time-slice Allocation1**

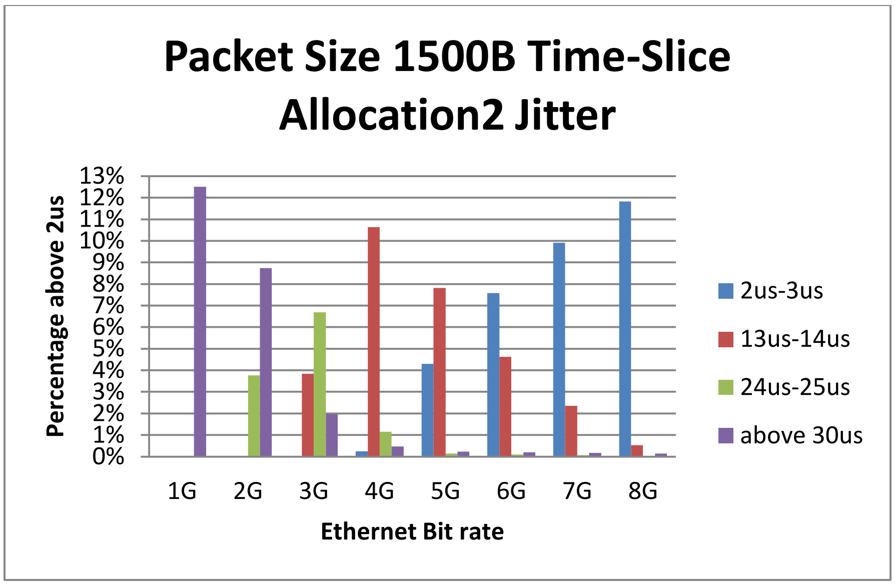


**Figure 3-34: FPGA Measured Results: Jitter for Ethernet Frame Size 1500B, Time-slice Allocation2**

As shown in Figure 3-33 and Figure 3-34, the jitter results are similar for Time-slice Allocation 1 and 2. They also have similar jitter results of the frames not received within 2 μs,

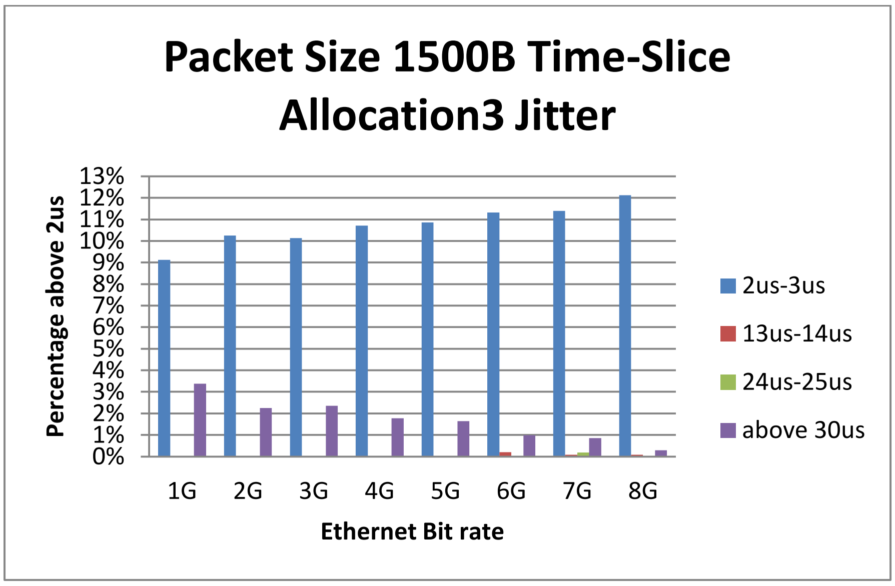


**Figure 3-35: FPGA Measured Results: Jitter for Packet Size 1500B, Time-slice Allocation3**

Figure 3-35 shows the Jitter result of Time-slice Allocation3. Compared with Figure 3-33 and Figure 3-34, the packets that not received in 1 μs have the highest jitter.

For all the cases measured in Figure 3-33, Figure 3-34 and Figure 3-35, 87.5% of the packets are received with less than 2 μs jitter.

Layer 1 of TSON Metro Node with time and frequency switching

## 4 TSON Layer 1 data transfer

The ingress traffic to a TSON edge node is switched through intermediate TSON bypass nodes and transported to egress TSON edge node of the TSON metro mesh cloud.

As for this scenario, the traffic at the ingress edge node, transported on two wavelengths, is first coupled together and then sent over the fibre link. The traffic is then switched

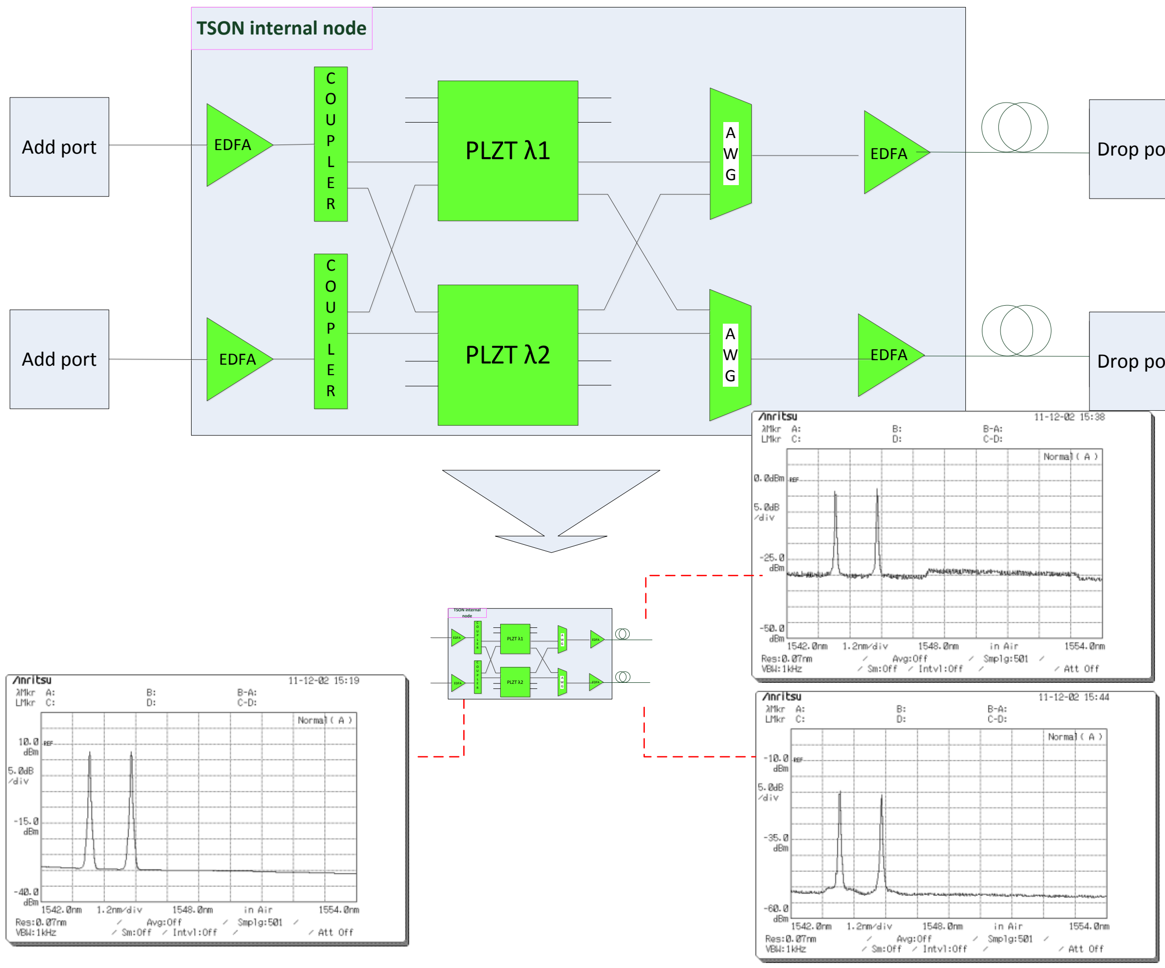


**Figure 4-1: The TSON metro node data plane with the corresponding snap shots on the ingress and egress ports**

using PLZT fast switches, as the main enabling elements of time-slice switching in the transparent TSON data plane. The coupled signals need to be amplified to compensate for the insertion loss introduced by the switches. After that, the wavelengths on the signal are decoupled, and then passed to the dedicated fast switch (wavelength modular architecture). No matter being a TSON edge or a TSON bypass node, the traffic will be switched through PLZT fast switches to be directed on the requested fibre link. The PLZT fast switches can change state in 10 ns, which is a speed/overhead requirement for the high speed time-slice switching. They need to be controlled in highly precise manner for the high speed operation of the network. The control of the switches takes place with the functionality deployed inside the FPGA, through a parallel interface.

The switched traffic, on both wavelengths, will be multiplexed, amplified, and sent on the fibre links to be transported to the next node.

The lower part of Figure 4-1 illustrates three snapshots taken from the input and output ports of the TSON L1 OXC. The TSON traffic on the input is being directed on two different paths as the optical spectrum analyzer displays.

## 4.1 TSON with Gridless Layer 1 data switching

The physical layer structure of the TSON metro node is changed to be dynamic and responsive to requests both for TSON mode, and Gridless mode, by using a 3D MEMS switch as an optical backplane for optical components. So, based on the requests, the desired OXC whether for TSON or Gridless is built using the plug-in components on the backplane switch with the respective commands sent from the controller to the backplane manager. This dynamic data plane OXC configuration supports time-sliced data transfer alongside of the Gridless data transport.

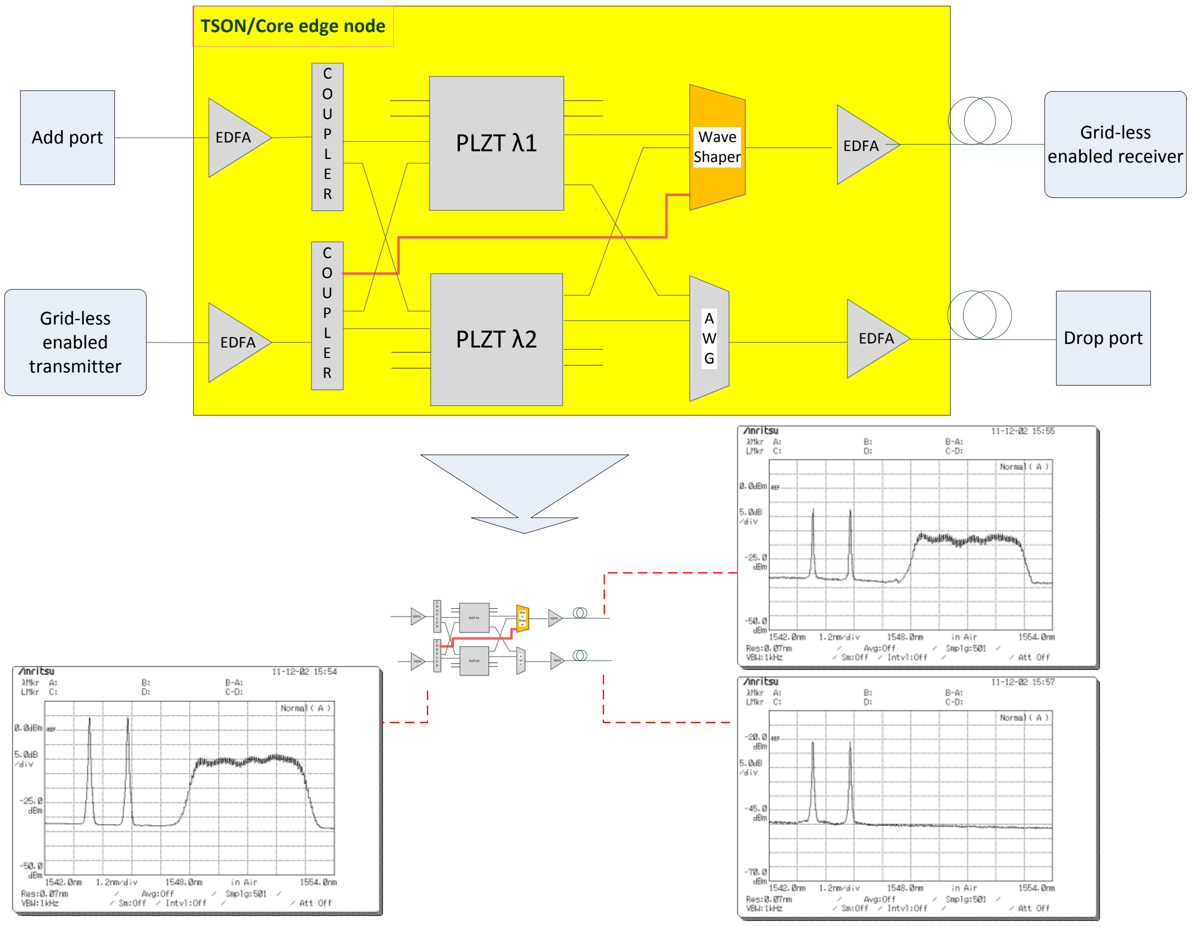


**Figure 4-2: Data plane for the metro-core, TSON extended with Gridless node, with the corresponding snap shots on the ingress and egress ports.**

This dynamic data plane is made using a 3D MEMS switch, hosting the required optical components. All the components are connected to the backplane ports, and they can be put in to circuit by configuring the 3D MEMS. By this approach data plane can have those components involved as needed.

Using this backplane, for TSON operation, the same data plane nodes as shown in Figure 4-1 and described earlier will be built up.

However, in order to support Gridless switching besides the TSON, a spectrum selective switch (SSS) is incorporated in the node. The generated Gridless traffic, won't pass through TSON data plane, but instead, it will directly be sent to the SSS (waveshaper), and then directed on to the next hop on the core network, as shown in Figure 4-2.

The traffic might still need amplification to compensate for the losses of the wave shaper or the transmission links, so amplifiers can be added to the Gridless data plane.

The lower part of Figure 4-2 illustrates the spectrum of the input and output traffic. The results show, in the input, three signals (two 10 Gbps TSON and one Gridless), where in the outputs, TSON signals can be seen on both ends since the signals where time-shared to both destinations, while the super channel at 555 Gbps with spectrum requirement of 650 GHz can be observed on upper path only.

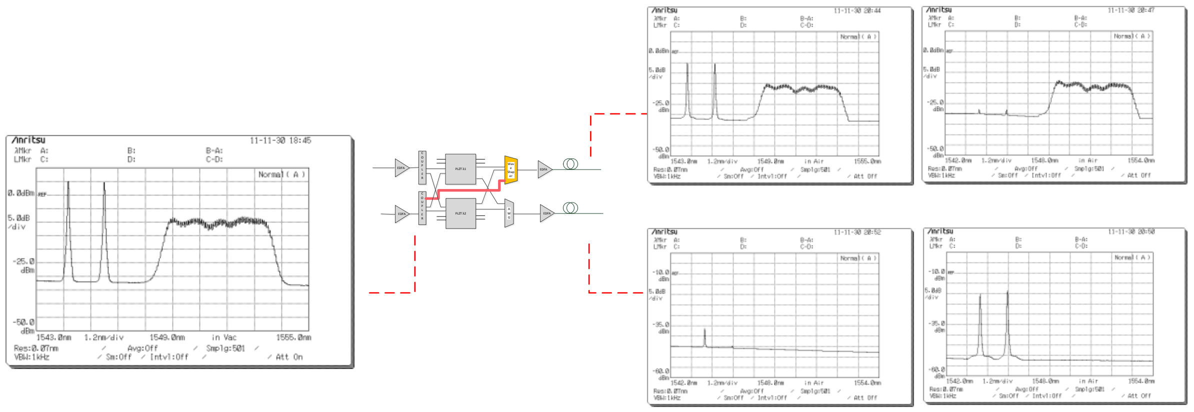


**Figure 4-3: Extended results from the TSON and Gridless scenario, in which, the PLZT switches direct the traffic from the source node to either of the receiving ends.**

Besides, more results demonstrate the PLZTs that switch the TSON traffic only to one of the destination (wavelength switching) output ports as per Figure 4-3. In the Figure 4-4 it can be seen how the elements are connected to the back plane. Shown in the figure, PLZT switches with some accompanying optical components are assumed to be the elements for building up TSON data plane, and the waveshaper for supporting Gridless data switching. The Gridless data (indicated by red line in the figure), just passes through the ports of waveshaper unlike the TSON traffic which is being MUXed and DeMUXed.

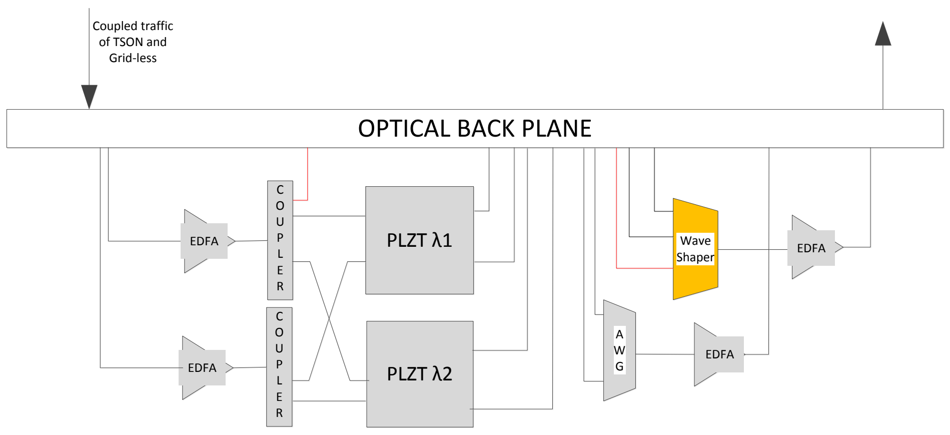


**Figure 4-4: The optical back plane used for supporting intra-metro and metro-core data plane**

Figure 4-4 clearly shows the PLZT switches as the TSON enabling devices and the waveshaper for Gridless operation are selected from the back plane.

## 4.2 Hardware Components

These hardware components are used in the Layer 1 node implementation of the TSON node.

### 4.2.1 PLZT fast switches[4]

The PLZT 4x4 switches are nano-second speed optical switches designed for advanced optical communications and interconnections systems. The series feature switching time below 10 nanoseconds. [26]

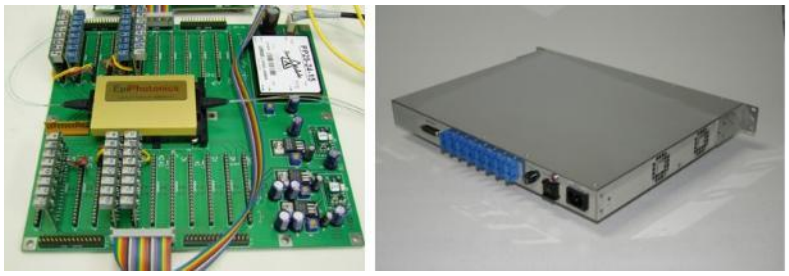


**Figure 4-5: PLZT switches**

The switch has been tested for a sample combination of 1-1, 2-2, 3-3, and 4-4, and the results are displayed in the table below:

| Input - output ports | **1 - 1** | **2 - 2** | **3 - 3** | **4 - 4** |
|---|---|---|---|---|
| Insertion Loss | 10.6 | 36.3 | 48.7 | 65.5 |
| | 28.4 | 11.7 | 52.4 | 60.6 |
| | 59.6 | 46.9 | 11.3 | 40.1 |
| | 57.8 | 57.5 | 34.3 | 11.2 |
| Cross Talk | 0 | 24.6 | 37.4 | 54.3 |
| | 17.8 | 0 | 41.1 | 49.4 |
| | 49 | 35.2 | 0 | 28.9 |
| | 47.2 | 45.8 | 23 | 0 |

**Table 4-1: PLZT switch characteristics**

[4] figure from http://epiphotonics.com/products.html

### 4.2.2 Calient's DiamondWave FiberConnect backplane

The Calient 3D MEMS connect single-mode optical fibres of any bandwidth, protocol or wavelength. It connects any input fibre to any available output fibre all optically through a web browser based GUI interface. port changes commands will drive a driver circuitry to establish and change connections [27].
The input fibres are connected to the 3D-MEMS optical switch, which serves as the traditional patch cord connecting to the output fibres.
The Calient switch features 96x96 patch cord on a 320x320 port chassis with extendible modularity.

Figure 4-6: Calient Switch [27]

### 4.2.3 Waveshaper[5]

A Finisar waveshaper 4000s is being used in the back plane to allow spectrum switching enabling transport of Gridless traffic. According to user manuals[5], the Wave Shaper is a fully programmable, flat-top optical filter or DWDM channel selector. The filter or channel bandwidth is programmable in 1 GHz increments from 10 GHz up to the whole C-band, with the centre frequency programmable in 1 GHz increments over the whole band. Band-stop and optical comb filters are also supported, as is attenuation control on a per-channel basis up to 35dB.

This waveshaper supports arbitrary user-generated channel and filter shapes In addition to the classical 'flat-top' channel shape. The required filter profile (both amplitude and phase) can be generated by the user and then loaded into the WaveShaper software which translates the user specification into the required filter shape. Any arbitrary filter can then be used as the switching transfer function for the Wavelength Switching capability of the 4000S [28].

**Figure 4-7: Wave shaper as spectrum selective switch**

[5] Figure from http://www.finisarcables.com/optical_instrumentation

# 5 SHINE Enabled TSON Metro Node

Following D2.3, a flexible and programmable TSON metro node is highly desired. Given the implemented TSON metro node presented in previous sections 3 and 0, this section will report the SHINE solution which fulfils the desired features as an additional module to the TSON metro node. This solution includes hardware modules and software tool.

## 5.1 SHINE Solution

SHINE is an intelligent solution that enables on-demand re-programmability of components (e.g. FIFOs), sub-systems (e.g. aggregation, MAC, PHY), interfaces (e.g. 10G Ethernet, 10G TSON), nodes (e.g. Ethernet switch, TSON switch) and complete network to deliver diverse set of functions over the same hardware infrastructure. As such it enables adaptation, construction of network capability, functionality and services on demand. It allows dynamic manipulation and modification of the network elements and their functionalities in both software and hardware dimensions, also fast provisioning of new technologies and new services in the process of approaching future network. It allows for innovative optical layer electronic and optical components dynamic composition either based on passively upper layers service requests or actively self-monitoring and self-optimisation according to network status.

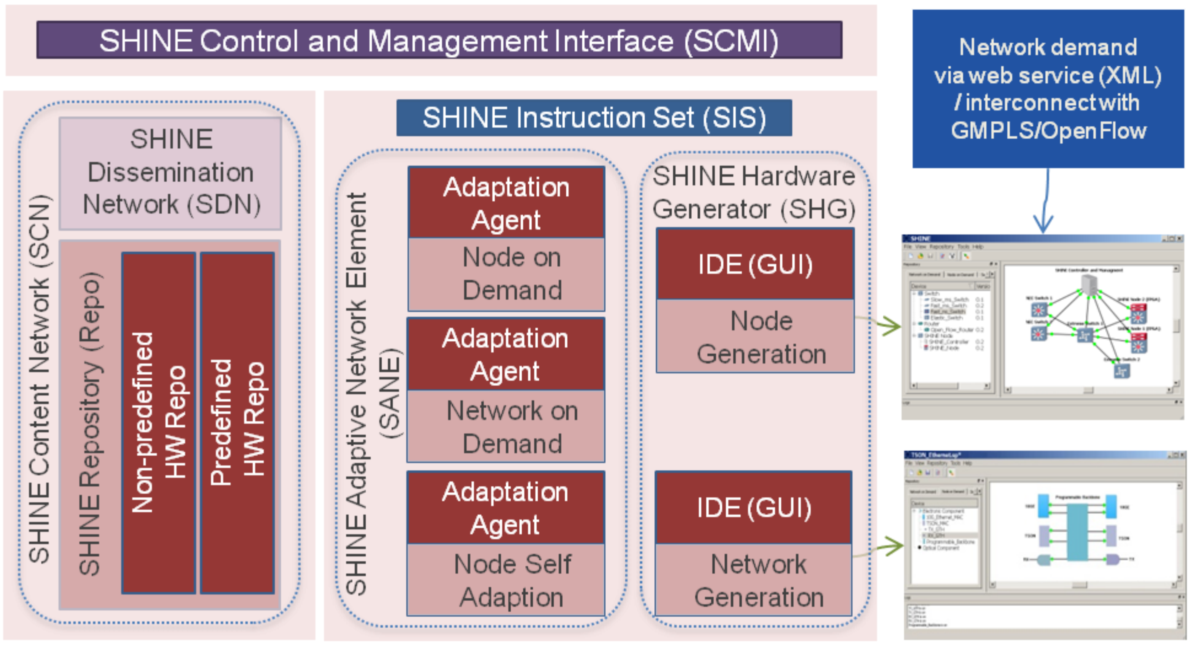

SHINE Control and Management Interface (SCMI)
Network demand via web service (XML) / interconnect with GMPLS/OpenFlow
SHINE Content Network (SCN)
SHINE Dissemination Network (SDN)
SHINE Repository (Repo)
Non-predefined HW Repo
Predefined HW Repo
SHINE Instruction Set (SIS)
SHINE Adaptive Network Element (SANE)
Adaptation Agent
Node on Demand
Adaptation Agent
Network on Demand
Adaptation Agent
Node Self Adaption
SHINE Hardware Generator (SHG)
IDE (GUI)
Node Generation
IDE (GUI)
Network Generation

**Figure 5-1: SHINE Solution**

According to the SHINE solution definition, it consists of the following five major elements as shown in Figure 5-1:

- SHINE Adaptive Netwok Element (SANE)
- SHINE Hardware Generator (SHG)
- SHINE Content Network (SCN)
- SHINE Instruction Set (SIS)
- SHINE Control and Management Interface (SCMI)

## 5.2 SHINE Enabled TSON Metro Node Implementation

### 5.2.1 SANE

SANE plays a significant role in the SHINE solution. It consists of the SHINE adaptation agent (SAA), SHINE adaptive hardware (SAH) and the SIS. This combination provides a flexible portal to the underneath adaptive and programmable hardware for upper layer services and users. For the sake of full intelligence, SAA is 2D control logic. To elaborate, the SAA can not only support *passive* control over the SAH based on the demands from upper layer services and users to support node on demand and network on demand respectively, but also support *active* self-adaptation of the node itself based on the changing network environment (e.g. traffic profile, load, network failure, and many others).

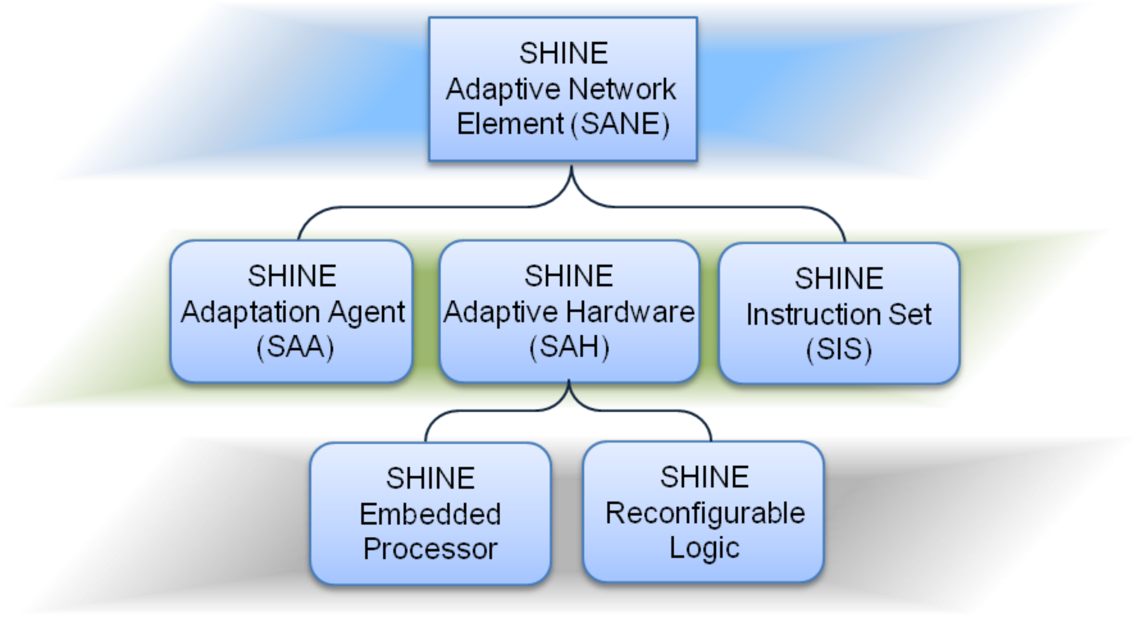

SHINE Adaptive Network Element (SANE)
SHINE Adaptation Agent (SAA)
SHINE Adaptive Hardware (SAH)
SHINE Instruction Set (SIS)
SHINE Embedded Processor
SHINE Reconfigurable Logic

**Figure 5-2: Anatomy of SANE**

Apart from presenting SANE from the logical point of view in Figure 5-1, the anatomy of SANE is depicted in Figure 5-2. The SAH can contain two types of elements, i.e. embedded processor and/or reconfigurable logic (FPGA), to deliver the comprehensive control and user logic.

#### 5.2.1.1 SHINE Enabled TSON Metro Node Function Blocks

Figure 5-3 shows the SHINE enabled TSON metro node architecture. The node is fully adaptable by deploying the intelligent SAA for each of the user function blocks, which need to deploy service/application requirements.

Based on the design requirement in Subsection 3.1 and design strategy in Subsection 3.2, considering the features and limitations of FPGA, this subsection describes the architecture for a single TSON Metro Node.

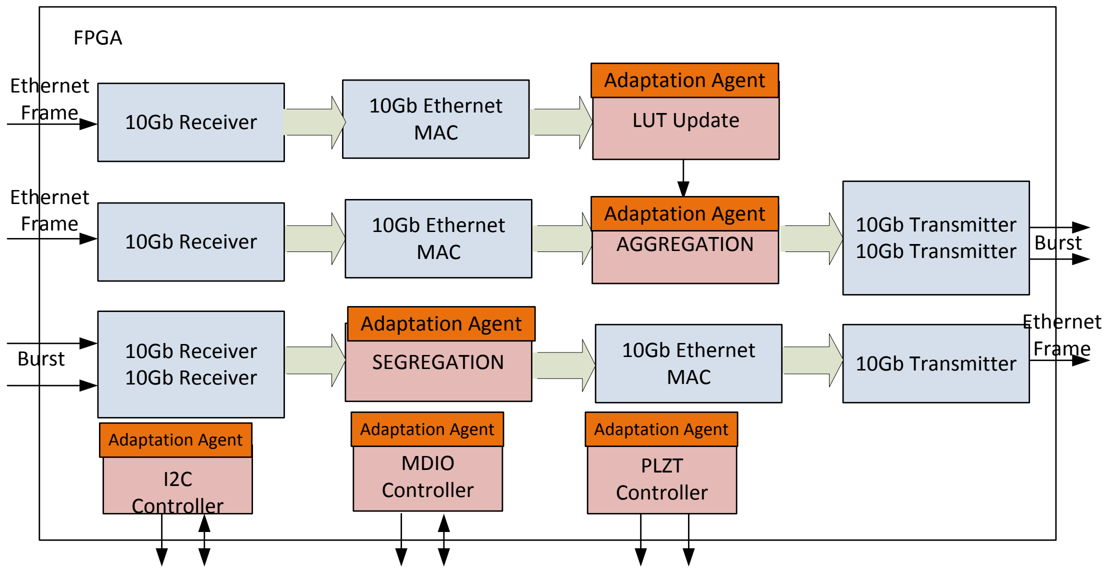


**Figure 5-3: TSON Metro Node with SHINE Support Function Blocks**

As shown in Figure 5-3, blocks with blue colour are supplied by Xilinx as IP cores, which include 10Gbps Transmitter/Receiver and 10Gbps Ethernet MAC. The other modules were implemented (coloured red) in VHDL. Compare to Figure 3-4, the major difference is that on each user function modules (coloured red), an adaptation agent (SAA) module is deployed to control, configure, program, and monitor the module underneath.

So that the modules and module requirements shown in Table 3-2 is updated to new Table 5-1. The design details will be described in the Subsection 5.2.1.2.

| Module Name | Functions |
|---|---|
| I2C controller | Set Si5368/Si570 on board to generate the clock. |
| MDIO controller | Initialize HTG-SFP+ extender card on-board-chip AEL2006. |
| PLZT controller | Control PLZT switches. |
| 10Gbps Transmitter/Receiver | Xilinx IP core GTH for 10Gbps Transmission. |
| 10Gb Ethernet MAC | Xilinx IP core 10Gb Ethernet MAC. |
| Aggregation | Aggregate Ethernet Packets to burst and send the burst out based on the Time-slice Allocation LUT. |
| Segregation | Segregate burst to Ethernet Packets and transmit out when received. |
| LUT Update | Receive and parse the information from PLZT, Update the LUT. |
| SHINE Adaptation Agent (SAA) | Configure, program, and monitor the module underneath, i.e. the modules listed above |

**Table 5-1: FPGA design functional modules and requirements with SAA**

Furthermore, considering the work clock domains, all 10Gbps receivers and transmitters work at 156.25MHz, but in different clock domains. The I2C controller and MDIO controller run in the clock domain of 50MHz. For the design of the TSON metro node, totally, there are 8 clock domains. Therefore, an important concern of the design is handling the cross-clock domain signals.

Following Figure 5-3 of the Node architecture, a diagram of detailed functional blocks of TSON Metro Node is shown in Figure 5-4.

Compared with Figure 5-3, Figure 5-4 gives more details of how the node is designed. The purple blocks construct the ingress function of the node, the pink blocks can achieve the egress function, and the green blocks complete a link from GMPLS to the node. The data flow is following the arrow directions.

#### 5.2.1.2 Node Implementation Details

SHINE is a flexible method of changing hardware in real-time with the aid of software. In the SHINE software, the FPGA sub-modules, such as aggregation, segregation, switch and etc., can be dragged and connected by the users to generate the whole system. A demonstration of SHINE theory is implemented by controlling the FPGA to switch between TSON metro node and Ethernet node that forwards Ethernet frames to the appropriate output port. Figure 5-4 shows the SHINE block diagram.

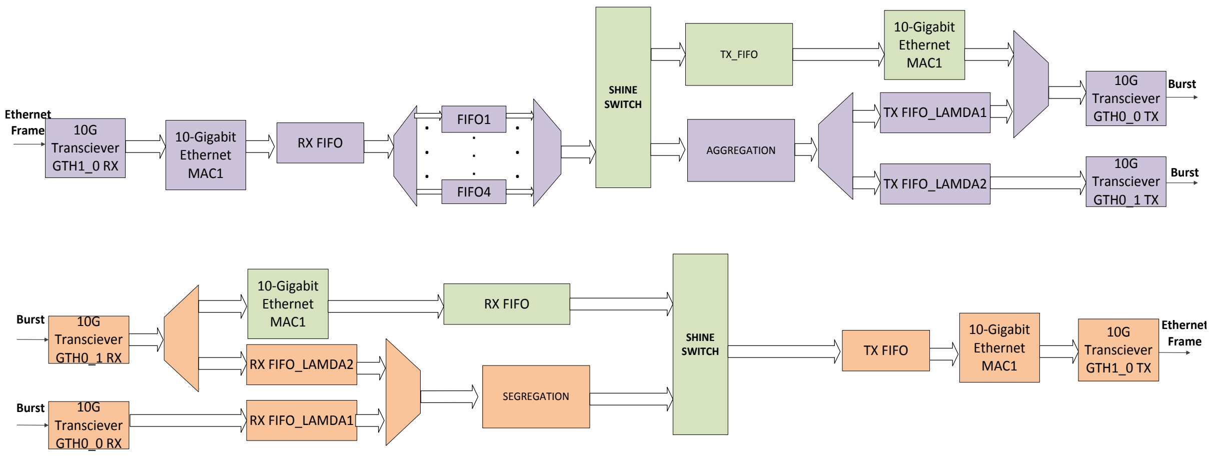
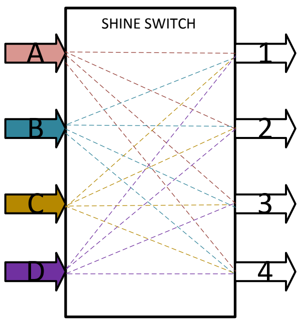


**Figure 5-4: Layer 2 SHINE Block Diagram**

In the Figure 5-4, the purple blocks are the original ingress function blocks, and the pink ones are the original egress function blocks. The light green blocks are added to implement a bypass function to transmit all the Ethernet packets directly. One more 10Gbps Ethernet MAC is employed for receiving and transmitting the Ethernet packets.

The idea is to allow a *hitless switch over from TSON mode to Ethernet mode and vice versa*. To implement this, A SHINE SWITCH block was designed to control the data and signal transferring. When it receives a switch command, it should be able to tell all input ports to temporarily pause after sending current packet/burst; when the entire input ports pause, based on the command, the SHINE SWITCH block reconnects the inputs to the output, then notify all the input ports to retransmit data.

For the SHINE Switch Block used in Figure 5-4, the number of the ports can be configured by the user. An example of 4x4 top level view of SHINE Switch Block is shown Figure 5-5. When the users want to change the FPGA operational mode, they can easily change the switch LUT in the FPGA by UART/Serial Port. The Switch LUT illustrated in Figure 5-5 is an example of input port A, B, C, and D connecting directly to output port 1,2,3,4.



| SWITCH LUT | | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| A | X | | | |
| B | | X | | |
| C | | | X | |
| D | | | | X |

**Figure 5-5: Switch Block and Switch LUT**

### 5.2.2 SIS

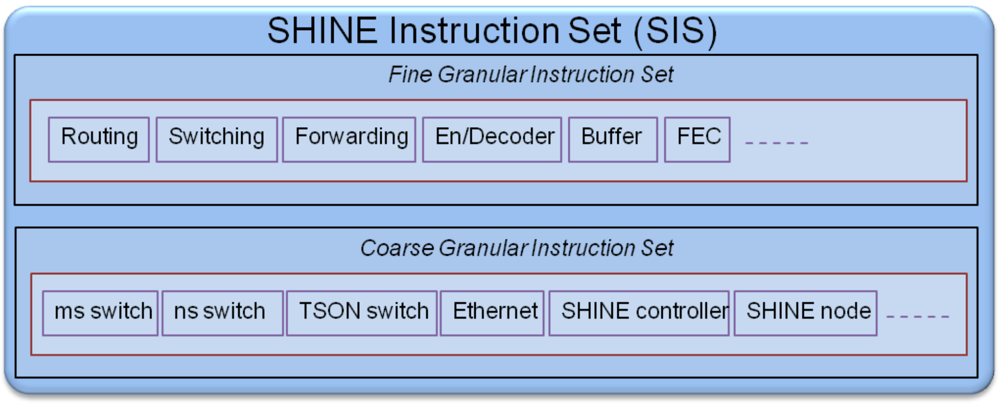


**Figure 5-6: Anatomy of SIS**

SIS is used for users (e.g. network operators) to define and control their own SANE to accommodate specific service requirement and traffic demands for networking. The anatomy of SIS is depicted in Figure 5-6. It shows that SIS has two granularities of *library*, the fine granular instruction set and coarse one. The former includes the following modules to routing, switching, forwarding, buffer, FEC, and is extendable to others. The latter consists of the following sub-systems/systems such as fast switch, slow switch, Ethernet, SHINE controller, SHINE node, etc. The fine granular instruction set is mainly used for node level, but the coarse one is used for network level. Both *libraries* can be local or remotely stored in SCN repositories. The screenshot of SHINE IDE instruction set is shown in Figure 5-7.

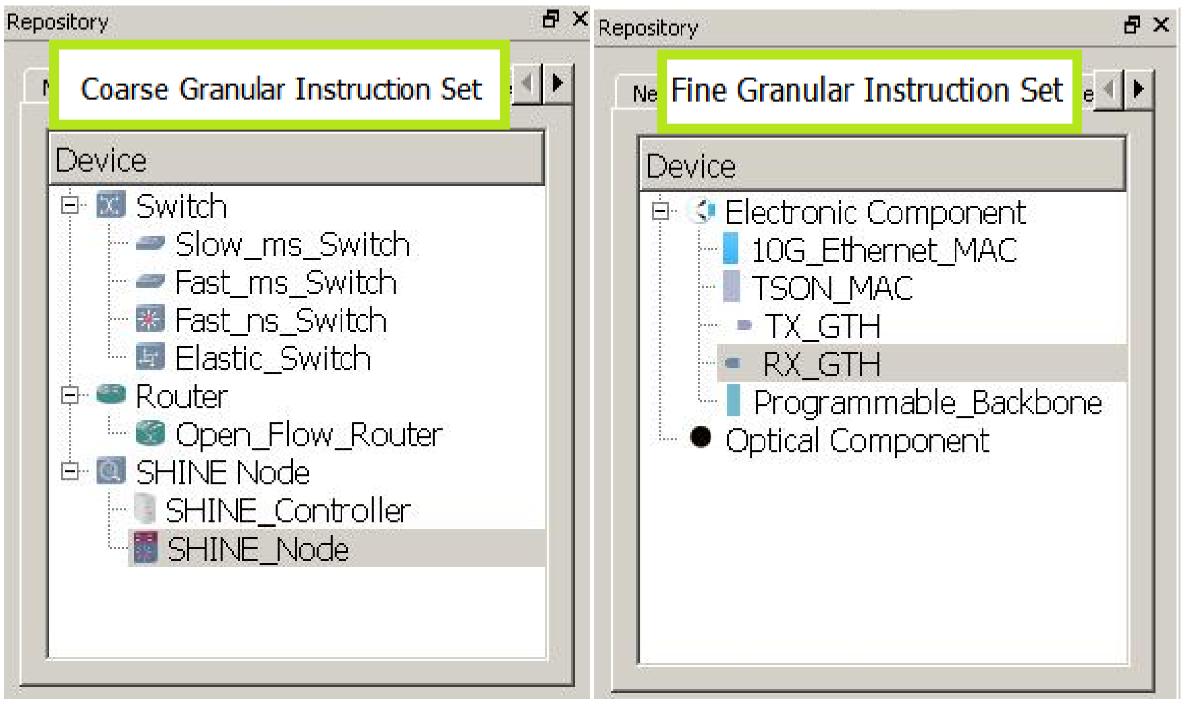


**Figure 5-7: SHINE IDE Instruction Set Screenshot (Coarse and Fine Granular)**

### 5.2.3 SHG

The SHG is another very important element of SHINE solution. It is a set of software tools developed to either automatically or manually generate SANE. Consequently it can be invoked by external control plane or utilised by network architect or NE designer.

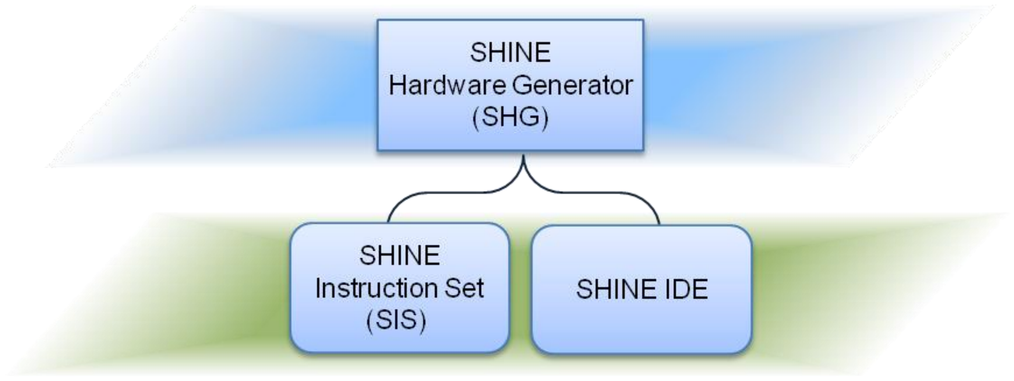
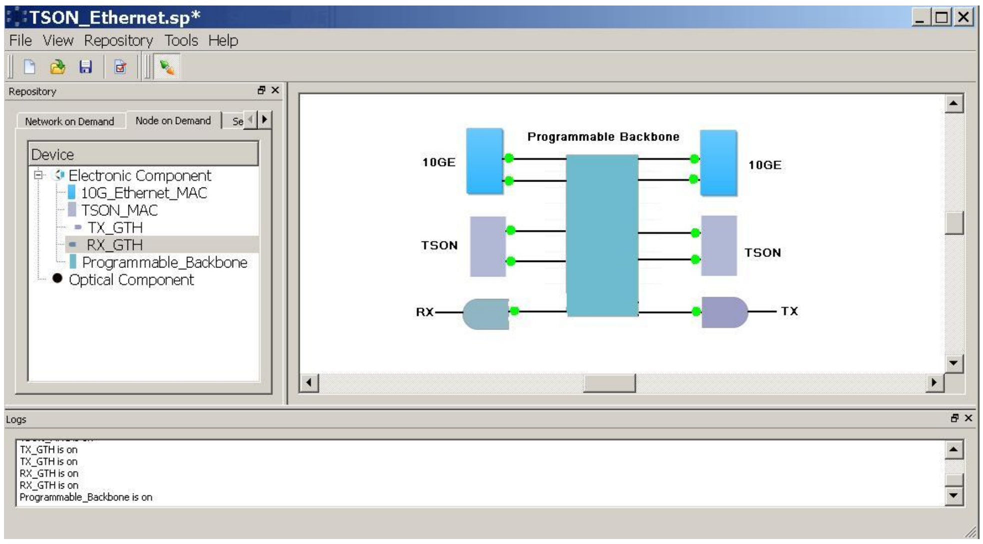


**Figure 5-8: Anatomy of SHG**

Apart from presenting SHG from the logical point of view in Figure 5-1, anatomy of SHG is depicted in Figure 5-8. The SHG contains SIS and SHINE IDE to design or generate node/network.

### 5.2.3.1 Node generation



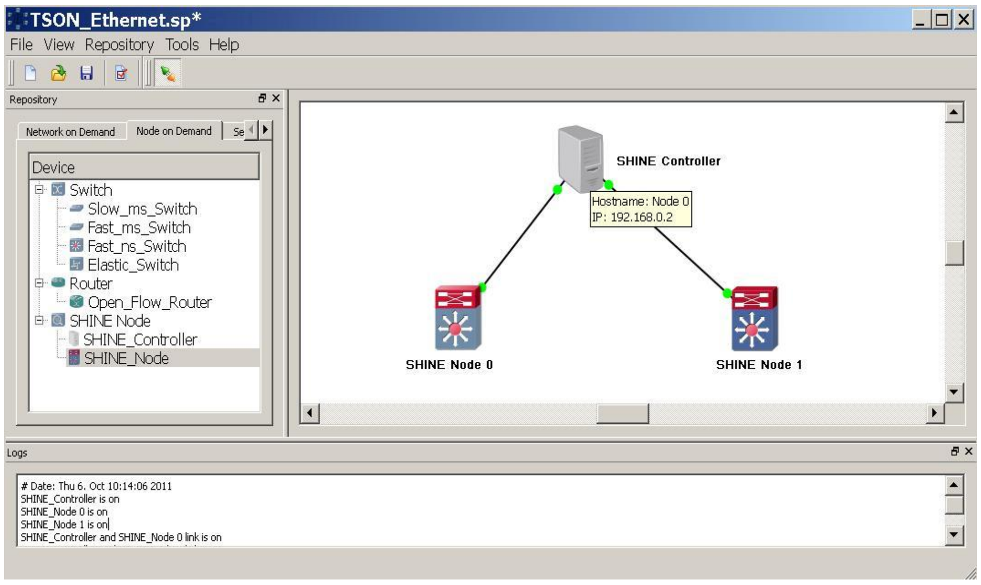
```
TX_GTH is on
TX_GTH is on
RX_GTH is on
RX_GTH is on
Programmable_Backbone is on
```

**Figure 5-9: SHINE IDE Node Generation Mode**

The fourth type (node generation) is different from the above three categories of which the SHINE function happens inside the SANE, instead, the network generation actually is realised in software, in this case the SHINE IDE as show in Figure 5-9.

It supports manually reconstruction of SANE using the unified fine granular SIS APIs if the requested service is currently not supported yet. For example, a network architect or network element designer can use the APIs to build SHINE enabled TSON from scratch without detailed hardware knowledge. As more software/hardware APIs pour into the instruction set, more area and applications can be supported without redesign the wheels which is happening in the industry. This is the benefit of unified SIS.

### 5.2.3.2 Network generation



```
# Date: Thu 6. Oct 10:14:06 2011
SHINE_Controller is on
SHINE_Node 0 is on
SHINE_Node 1 is on
SHINE_Controller and SHINE_Node 0 link is on
```

**Figure 5-10: SHINE IDE Network Generation Mode**

The fifth type (network generation) is also different from the first three categories of which the SHINE function happens inside the SANE, instead, the network generation actually is realised in software, in this case the SHINE IDE as show in Figure 5-10.

It supports manually reconstruction of network using the unified coarse granular SIS APIs if the requested service is currently not supported yet. The coarse granular SIS APIs can be generated in node generation mode, and registered in the SHINE solution. This leverages the possibility of completely re-purpose the existing SANE.

#### 5.2.3.3 SIS Registration

In the SHINE IDE, the SIS is registered in XML format, an example of the fine and coarse granular instruction sets are shown in Figure 5-11 and Figure 5-12 respectively.

```
<?xml version="1.0" encoding="UTF-8"?>
<DeviceRepository version="1.0">
      <category term="Electronic Component">
            <icon_normal>:/icons/Electronic.normal.svg</icon_normal>
            <devicetype>-1</devicetype>
            <device>
                  <devicetype>0</devicetype>
                  <devicename>10G_Ethernet_MAC</devicename>
                  <icon_normal>:/icons/10G_Ethernet_MAC.normal.svg</icon_normal
            >
                  <icon_selected>:/icons/10G_Ethernet_MAC.selected.svg</icon_se
            lected>
                  <version>0.1</version>
                  <license>free</license>
            </device>
            <device>
                  <devicetype>1</devicetype>
                  <devicename>TSON_MAC</devicename>
                  <icon_normal>:/icons/TSON_MAC.normal.svg</icon_normal>
                  <icon_selected>:/icons/TSON_MAC.selected.svg</icon_selected>
                  <version>0.2</version>
                  <license>commercial</license>
            </device>
            <device>
                  <devicetype>2</devicetype>
                  <devicename>TX_GTH</devicename>
                  <icon_normal>:/icons/TX_GTH.normal.svg</icon_normal>
                  <icon_selected>:/icons/TX_GTH.selected.svg</icon_selected>
                  <version>0.1</version>
                  <license>commercial</license>
            </device>
            <device>
                  <devicetype>3</devicetype>
                  <devicename>RX_GTH</devicename>
                  <icon_normal>:/icons/RX_GTH.normal.svg</icon_normal>
                  <icon_selected>:/icons/RX_GTH.selected.svg</icon_selected>
                  <version>0.1</version>
                  <license>free</license>
            </device>
            <device>
                  <devicetype>4</devicetype>
                  <devicename>Programmable_Backbone</devicename>
                  <icon_normal>:/icons/Programmable_Backbone.normal.svg</icon_n
            ormal>
                  <icon_selected>:/icons/Programmable_Backbone.selected.svg</ic
            on_selected>
                  <version>0.1</version>
                  <license>free</license>
            </device>
      </category>
      <category term="Optical Component">
            <icon_normal>:/icons/Optical.normal.svg</icon_normal>
            <devicetype>-2</devicetype>
      </category>
</DeviceRepository>
```

**Figure 5-11: Example of Fine Granular Instruction Set XML**

```
<?xml version="1.0" encoding="UTF-8"?>
<DeviceRepository version="1.0">
      <category term="Switch">
            <icon_normal>:/icons/Switch.normal.svg</icon_normal>
            <devicetype>-1</devicetype>
            <device>
                  <devicetype>3</devicetype>
                  <devicename>Elastic_Switch</devicename>
                  <icon_normal>:/icons/Elastic_Switch.normal.svg</icon_normal>
                  <icon_selected>:/icons/Elastic_Switch.selected.svg</icon_selected>
                  <version>0.1</version>
                  <license>free</license>
            </device>
      </category>
      <category term="Router">
            <icon_normal>:/icons/Router.normal.svg</icon_normal>
            <devicetype>-2</devicetype>
            <device>
                  <devicetype>4</devicetype>
                  <devicename>Open_Flow_Router</devicename>
                  <icon_normal>:/icons/Open_Flow_Router.normal.svg</icon_normal>
                  <icon_selected>:/icons/Open_Flow_Router.selected.svg</icon_selected>
                  <version>0.2</version>
                  <license>free</license>
            </device>
      </category>
      <category term="SHINE Node">
            <icon_normal>:/icons/SHINE.normal.svg</icon_normal>
            <devicetype>-3</devicetype>
            <device>
                  <devicetype>5</devicetype>
                  <devicename>SHINE_Controller</devicename>
                  <icon_normal>:/icons/SHINE_Controller.normal.svg</icon_normal>
                  <icon_selected>:/icons/SHINE_Controller.selected.svg</icon_selected>
                  <version>0.2</version>
                  <license>free</license>
            </device>
            <device>
                  <devicetype>6</devicetype>
                  <devicename>SHINE_Node</devicename>
            <icon_normal>:/icons/SHINE_Node.normal.svg</icon_normal>
                  <icon_selected>:/icons/SHINE_Node.selected.svg</icon_selected>
                  <version>0.2</version>
                  <license>free</license>
            </device>
      </category>
</DeviceRepository>
```

**Figure 5-12: Example of Coarse Granular Instruction Set XML**

### 5.2.4 SCN

The SCN is a network responsible for storage and dissemination of SHINE software and hardware definition images for electronic and optical parts. The definition can be as small as a module, or as big as a subsystem/system. These definitions are stored in either networked repositories, i.e. databases, or even in the SANE locally. The dissemination of the definitions is provided by the SDN, which can be either in-band network or out-of-band network, say, traditional IP network. The SDN is not only responsible for repositories synchronisation, but also for the definition delivery from repositories to local SANE.

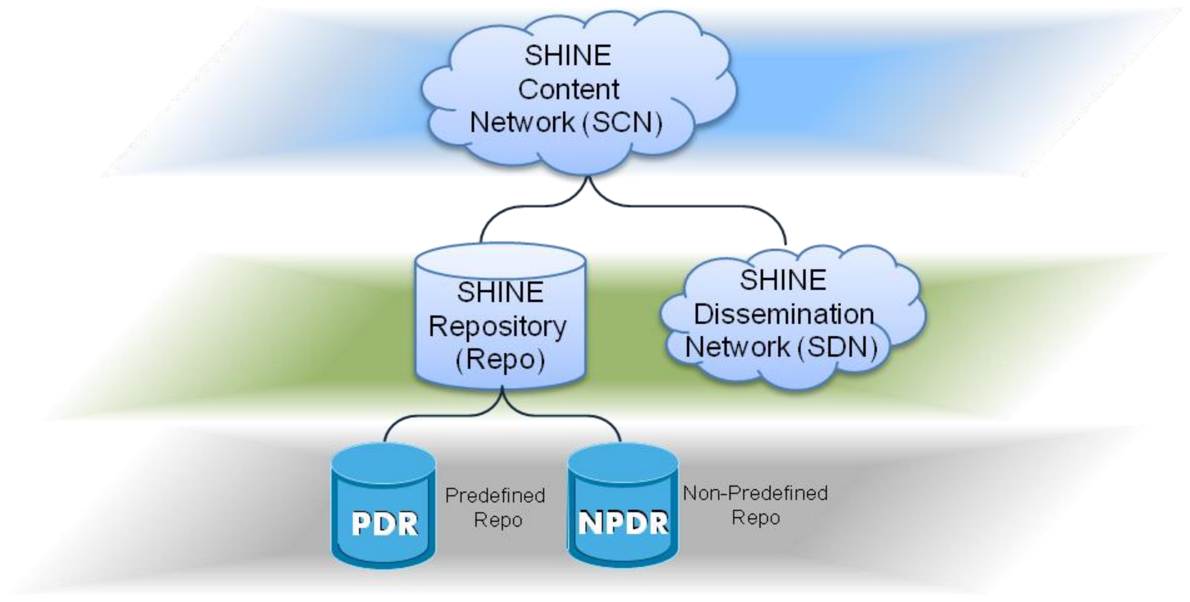


**Figure 5-13: Anatomy of SCN**

Apart from presenting SCN from the logical point of view in Figure 5-1, the anatomy of SANE is depicted in Figure 5-13. The SHINE repository is categorised into the following two types:

- predefined repository (PDR)
- non-predefined repository (NPDR)

### 5.2.5 SCMI

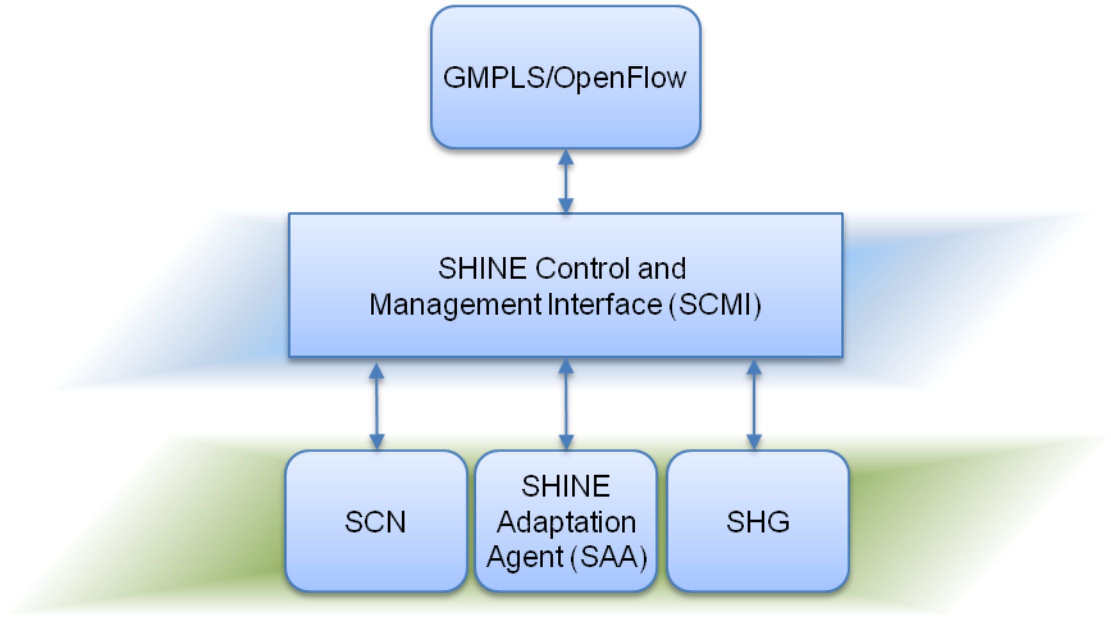

GMPLS/OpenFlow
SHINE Control and Management Interface (SCMI)
SCN
SHINE Adaptation Agent (SAA)
SHG

**Figure 5-14: Anatomy of SCMI**

The SCMI is an interface between the SHINE solution and the external control plane, such as GMPLS protocols. Through this interface, application/services demands can be conveyed to SHINE solution, particularly, the SCN, SAA and SHG. On the other direction, the current network status and the acknowledgement of the previous demands can be fed back to the external control plane.

When the SAA receives the demand, it will coordinate with the SANE to realise the system adaptation, e.g. reconfiguration, reprogram, switchover, and many others. This process is automatic and passive for SANE.

When the SHG receives the demand, it can analyse then automatically compose and synthesise SANE. The generated SANE will then be applied to the network. If the requested software/hardware definition exists, then it will be directly applied to the network without regeneration. Apart from using the SHG as an automatic operation tool, it can also be used as a design tool for the users to build their own SANE, and applied the network without the interaction with external control plane.

Via the SCMI, new hardware definition images can be pushed into SCN by external control plane, and get disseminated across SCN.

## 5.3 Incremental Test Results (both simulation and experiment)

### 5.3.1 Simulation Result

The simulation result is collected in Mentor Graphics Modelsim. Modelsim is a unified debug environment for Verilog, VHDL and SystemC. It is a very popular hardware simulation and debug tool. Apart from the node design described in Subsection 3.3.3, a testbench, which includes a stimulus block to generate Ethernet frames and a check block to check the output data, was written in VHDL. All the signals can also be checked through Modelsim waveform windows. The testbench clock frequency is 156.25MHz, the clock cycle is 6.4ns.

### 5.3.2 Simulation Results about SHINE Switch

For SHINE implementation described in Section 5.2.1.2, when SHINE switch command is set, the SHINE Switch notifies the input data to stop after sending out current packet/burst, and then the SHINE Switch will switch the input to a new output.

Figure 5-15 shows the SHINE simulation result of switching between TSON Metro Node and Ethernet Node.

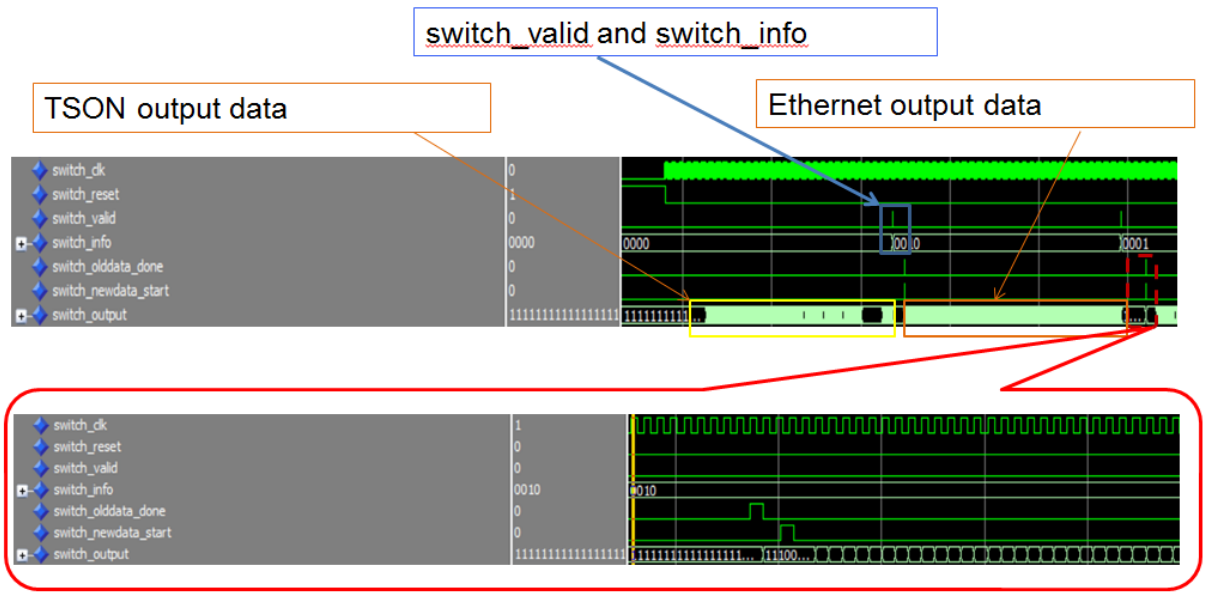


**Figure 5-15: Simulation Result 2: SHINE Switch**

When switching command is set to switch from TSON output to Ethernet packets output, the aggregation block sends out the burst already in the FIFO and notifies the switch it has finished, then the switch change the mode to Ethernet frames output.

When switching command is set to switch from Ethernet frames output to TSON output, the Ethernet module finishes sending current Ethernet frames, sends the flag to notify the SHINE switch to switch the output as TSON.

The clock cycles it takes for both situations depend on the data of Burst/Packet already in the FIFO considering that the goal is to have hitless switch over.

### 5.3.3 Implementation Results

All the modules in the project were written in VHDL which were synthesized, implemented by Xilinx ISE software. Xilinx ISE generated the bit files, and the bit files were downloaded to the FPGA on board through JTAG. The Xilinx XC6VHX380T-2FF1923 Device utilization summaries of one TSON Metro Node without and with SHINE are listed in Table 5-2.

| Device Utilization Summary | | | | | |
|---|---|---|---|---|---|
| **Slice Logic Utilization** | **Available** | **Used** | | **Utilization** | |
| | | **with SHINE** | **w/o SHINE** | **with SHINE** | **w/o SHINE** |
| Number of Slice Registers | 478080 | 37105 | 27632 | 7% | 5% |
| Number of Slice LUTs | 239040 | 24392 | 17,365 | 10% | 7% |
| Number of fully used LUT-FF pairs | 48764 | 12733 | 11,369 | 26% | 23% |
| Number of bonded IOBs | 720 | 89 | 89 | 12% | 12% |
| Number of Block RAM/FIFO | 768 | 318 | 258 | 41% | 33% |
| Number of BUFG/BUFGCTRLs | 32 | 31 | 28 | 96% | 87% |

**Table 5-2: Device Utilization Summary**

Figure 5-16 shows the hardware connection for measuring the implementation results. Anritsu MD1230B is an Ethernet/IP network data analyser. It is used as the Ethernet frames traffic generator and it is also used to analyse the latency and jitter of the received Ethernet packets. When the FPGA node receives the Ethernet packets, it aggregates them and sends them out as optical bursts. The bursts travel back to FPGA through a loop back fibre. When egress part receives the optical traffic bursts, it segregates the burst and sends out the Ethernet packets, which can be analysed by MD1230B. This section set several tests measuring and analysing the implementation results.

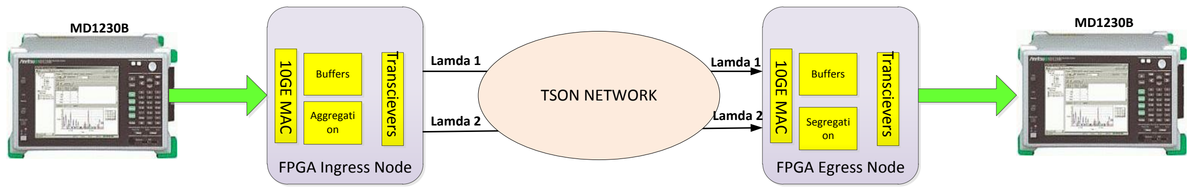


**Figure 5-16: Hardware Measurement Connection Diagram**

For all the experiment finished, the test results were taken based on 2 Ethernet packets length: 64 bytes and 1500 bytes.

#### 5.3.3.1 Experiment and Measurement: Bit Rate

As described in Sub subsection 5.2.1.2, when switch from TSON Node to Ethernet Node, the output transmits Ethernet packets directly without aggregation and segregation. This experiment is to test the maximum Ethernet Packet bit rate the Ethernet Node is capable to handle without losing any packets.

For Ethernet stream with packet length 64 bytes:

- The maximum Ethernet speed is 7.619Gbps;
- The maximum experimentally measured speed for Ethernet Node is 7.5294Gbps.
- The utilization is 98.82%.

For Ethernet stream with packet length 1500 bytes:

- The maximum Ethernet speed is 9.868Gb;
- The maximum experimentally measured speed for Ethernet Node is 9.772Gbps.
- The utilization is 99.27%.

#### 5.3.3.2 Experiment and Measurement: Latency

The aim of this experiment is to measure the latency results of Ethernet Node with input stream of Ethernet Packet size 64B and 1500B, and then compared them with TSON Node. Figure 5-17 shows the latency results of Ethernet Node, and TSON Node Time-slice Allocation 2 latency results which were described in subsection 3.4.2.4. As shown in Figure 5-17, for all the Ethernet speed, the latency results for one packet size stay stable. The latency of Packet Size 1500B is much higher than Packet Size of 64B, which is because the FIFO that was implemented in the project waits for one complete packet size, makes sure the Ethernet packet is good, and then sends it out.

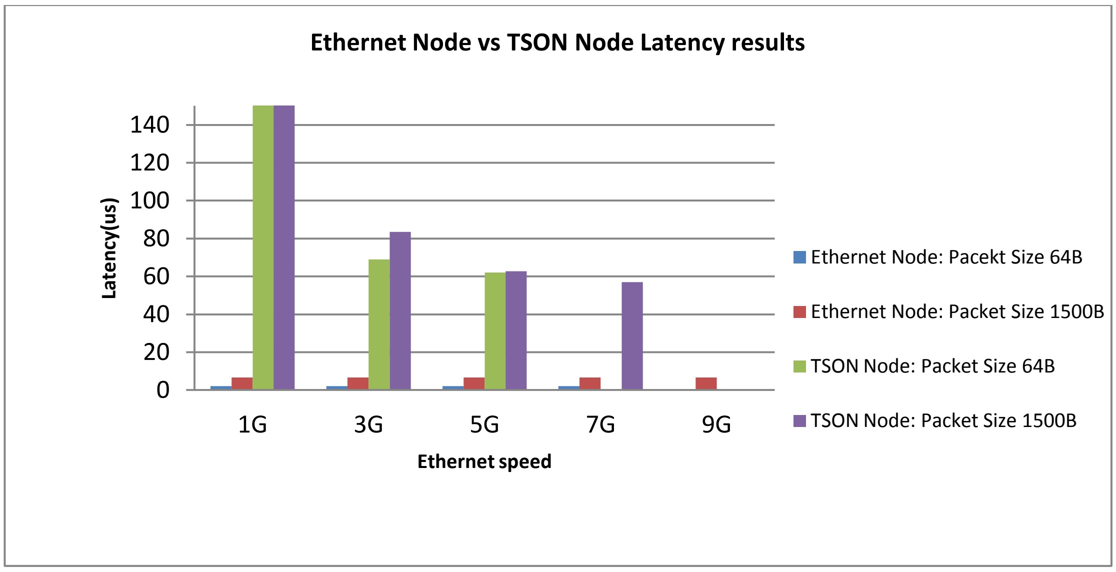


**Figure 5-17: FPGA Measured Results: Ethernet Node Latency Results**

Compare Ethernet Node latency and TSON Node latency, TSON has much higher latency due to aggregation and buffering for the time-slice allocation. Something required to deliver high network utilization. However, certain users/applications require ultra-low latency, and as such Ethernet Node is a better choice. But TSON has efficient bandwidth and spectrum management and utilization as shown in Sub section 3.4. Consequently the capability to dynamically offer switchover between Ethernet and TSON services gives broader options.

#### 5.3.3.3 Experiment and Measurement: Jitter

This experiment measure Jitter of Ethernet Node for Ethernet Frame 64B and Ethernet Frame 1500B in the capable speed.

For the input stream of Ethernet Frame Size 64B, 100% of the Ethernet packets are received within 1 μs.

For the input stream of Ethernet Packet Size 1500B, 100% of the Ethernet packets are received in 2 μs.

# 6 Conclusions

In this deliverable, the completed low-level design and implementation of superset TSON metro node for mesh topologies, which supports both OPST Ring-Mesh interconnection node and TSON (OBST) Mesh bypass node is reported. Particularly, as a layered design approach, the Layer 2 and Layer 1 functions are implemented, evaluated, analysed, and reported respectively. In addition, a novel SHINE solution to bring extreme flexibility and programmability towards the nodes is proposed, demonstrated, and evaluated.

To conclude, the following expectation is met:

- Deliver time-slice aggregation and scheduling strategies for synchronous burst trains and/or variable-size burst transmission.
- 
    - L2 functions of TSON node are implemented and able to deliver a maximum throughput of 8.68 Gbps per 10Gbps port.
    - A 95.38% utilization considering the 9.1 Gbps (91 time-slices) TSON theoretical maximum throughput.
    - Latency and jitter results have been experimentally measured for transport of min/max Ethernet frames.
    - Both Layer 2 and Layer 1 of TSON node support 2 wavelengths.

- Hardware-accelerated control and configuration of optical switches to support TSON traffic on a time-slice transport format.
- Implement northbound interface to TSON controller that can interconnect with XML interface.
- Implement gateway to interact with control plane and provide application-awareness to transport devices.
- Deliver the design of flexible and dynamic software-controlled electronic elements to adjust node parameters and deliver hitless switch over from TSON to Ethernet.