IMPLEMENTATION OF HIGH SPEED SERIAL I/O USING AURORA PROTOCOL IN XILINX FPGA

A Project Report

Submitted by

P. ANUSHA	**(2172007)**
Y. BHOOMIKA	**(2172018)**
P. DIVYA	**(2172029)**

in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Under the guidance of

Mrs. T. Nirmala, M. Tech, (Ph. D),

Assistant Professor

ELECTRONICS AND COMMUNICATION ENGINEERING
SCHOOL OF ENGINEERING AND TECHNOLOGY
SRI PADMAVATI MAHILA VISVAVIDYALAYAM
(WOMEN'S UNIVERSITY)
TIRUPATI – 517502 A.P INDIA

2021-2025

CERTIFICATE

This is to certify that **P.Anusha(2172007),Y.Bhoomika(2172018) & P.Divya(2172029) of SRI PADMAVATI MAHILA VISVAVIDYALAYAM** has undergone project training from **16/12/2024** to **04/04/2025** in the Defence Electronics Research Laboratory, Hyderabad-05. The project **"Implementation of High-Speed Serial I/O using Aurora Protocol in Xilinx FPGA"** is a record of the bonafide work undertaken by her towards partial fulfillment of the requirements for the award of the Degree of **Bachelor's Technology.** They have completed the assigned task satisfactorily.

(M. Krishna Prasad)	(K.Radha Krishna)	(JRC Sarma)
Sc 'E'	Sc 'G'	Sc 'G'
Guide	Wing Head	Wing Head HRD
		DLRL, Hyderabad

Department of Electronics and Communication Engineering

STUDENT DECLARATION

I hereby declare that the project entitled **"IMPLIMENTATION OF HIGH SPEED SERIAL I/O USING XILINX TOOLS IN FPGA"** submitted by us to the Department of Electronics and communication Engineering, School of Engineering and Technology, Sri Padmavati Mahila Visvavidyalayam, Tirupati in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Electronics and Communication Engineering** is a record of bonafide work carried out by under the supervision of **Mrs. T. Nirmala, MTech, (Ph.D), Assistant Professor.** We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.

Place: Tirupati
Date:

Signature of the Candidates

P.Anusha

Y.Bhoomika

P.Divya

Department of Electronics and Communication Engineering

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"IMPLEMENTATION OF HIGHSPEED SERIAL I/O USING XILINX TOOLS IN FPGA"** submitted by **P.Anusha(2172007), Y.Bhoomika(2172018) & P.Divya(2172029)** to School of Engineering and Technology, Sri Padmavati Mahila Visvavidyalayam, Tirupati in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology** in **Electronics and Communication Engineering** is a record of bonafide work carried out by her under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Guide **Head Of the Department**

Mrs.T.Nirmala, M.Tech, (Ph.D.)
Assistant Professor,
Department of ECE

Internal Examiner(s) **External Examiner(s)**

ACKNOWLEDGEMENT

The satisfaction and elation that accompany the successful completion of any task would be incomplete without the mention of the people who have made it a possibility. It is our privilege to express our gratitude and respect to all those who have guided and inspired us during the course of the project work.

We express profound sense of gratitude and sincere thanks to our beloved Director,**Prof. P. MALLIKARJUNA**, for motivating us and providing necessary infrastructure to complete the project work. We convey our thanks to the Co-ordinator **Mrs. B. MADHAVI, Assistant Professor** for her timely help and guidance whenever required.

We find immense pleasure in expressing profound gratitude and thanks to our Guide **Mrs.T.NIRMALA, MTech, (Ph.D), Assistant Professor,** Department of **ELECTRONICS AND COMMUNICATION ENGINEERING** for standing by our side all through the project.

We also thank all staff members of the Department of ECE for their support. We also place on record special thanks to our parents and friends who were with us all through the course of our project.

P.Anusha (2172007)

Y.Bhoomika (2172018)

P.Divya (2172029)

ABSTRACT

The I/O (Input Output) module works as a mediator between I/O devices and the processor. It conveys the information from I/O device to processor and vice versa. I/O devices are usually electronic devices where processor is an electronic device. I/O devices are majorly of two types: Parallel I/O and Serial I/O.

Parallel I/O is a subset of parallel computing that performs multiple I/O operations simultaneously, rather than process I/O requests serially, one at a time. This allows a system to achieve higher write speeds and maximizes bandwidth, but the usage of parallel I/O devices is decreasing as time progresses because it involves complex design due to the usage of multiple wires for the transmission hence only limited to usage in shorter distances. It also uses a greater number of pins compared to serial I/O for the same number of data bits which makes its usage problematic in higher level devices.

Serial I/O transmits individual data bits sequentially. It uses lesser number of lines for data transmission thereby reducing the design complexity. Since, the data transmission is sequential the signal delay increases.

Thus, this project aims to develop a protocol which achieves high speed serial I/O. High speed serial I/O helps to increase the data rate from Mbps to Gbps, decrease the design complexity, to design hardware using fewer number of pins on PCB and reduce signal delay to maximum extent possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

S.No	Abbreviations	Full Form
1.	SERDES	Serializer Deserializer
2.	I/O	Input/Output
3.	FPGA	Field Programmable Gate Array
4.	MGT	Multi Gigabit Transceiver
5.	CDR	Clock Data Recovery circuit
6.	PLL	Phased Lock Loop
7.	ATCA	Advanced Telecom Computing Architecture
8.	ASIC	Application Specific Integrated Circuits
9.	CLB	Configurable Logic Blocks
10.	OTP	One Time Programmable
11.	SATA	Serial Advanced Technology Attachment
12.	PCI	Peripheral Component Interconnect
13.	EMI	Electro Magnetic interference
14.	BRAM	Block Random Access Memory
15.	NGC	Native Gate Constraint
16.	UNISIM	Unified Simulation
17.	NFC	Native Flow Control
18.	FIFO	First In First Out
19.	BER	Bit-Error Ratio
20.	SDK	Software Development Kit
21.	EDK	Embedded Development Kit

22.	JTAG	Joint Test Action Group
23.	LUT	Look Up Tables
24.	SDR	Single Data Rate
25.	DDR	Double Data Rate
26.	UCF	User Constraint File

CHAPTER-1

INTRODUCTION

Introduction

1.1 About project in brief

Input/output (I/O) has always played a crucial role in computer and industrial applications. But as signal processing became more sophisticated, problems arose that prevented reliable I/O communication. In early parallel I/O buses, interface alignment problems prevented effective communication with outside devices. As higher speeds became prevalent in digital design, managing signal delays became problematic.

Thus, this project aims at the implementation of High-Speed Serial I/O which gives effective communication with minimum delay and reduces the design complexity.

1.2 Problem Definition

The I/O (Input Output) module works as a mediator between I/O devices and the processor. It conveys the information from I/O device to processor and vice versa. I/O devices are usually electronic devices where processor is an electronic device. I/O devices are majorly of two types: Parallel I/O and Serial I/O.

Parallel I/O devices were used initially, as the number of input data bits increased the design complexity also increased proportionally which in turn resulted in the increase of the cost of equipment.

Therefore, serial I/O devices became prominent, in the early 1980s when the IBM PC was first introduced IBM and other companies quickly made RS232 serial communications addon boards available to allow the connection of the PC to external devices. But the only disadvantage of the serial I/O was they were not capable of transmitting data quickly because of the sequential transmission rather than simultaneous.

Thus project aims to develop a protocol for the implantation of High Speed Serial I/O devices which has the data rate up to Gbps and which can achieve nano seconds operations.

1.3 Formulation of Objectives

This project divided into two phases and the objectives are formulated as stated below:

Phase I

- Identification of hardware resources inside FPGA
- Installation of Tool and Aurora 8B/10B core module Generation
- Simulation of protocol

Phase II

- Porting
- Testing on hardware

1.4 Existing System

Parallel I/O is a subset of parallel computing that performs multiple I/O operations simultaneously, rather than process I/O requests serially, one at a time. This allows a system to achieve higher write speeds and maximizes bandwidth.

Drawbacks of the Existing System:

- Design becomes complex for higher-level devices.
- Requires more number of pins on PCB.
- Can only be used for shorter distances.
- Skew

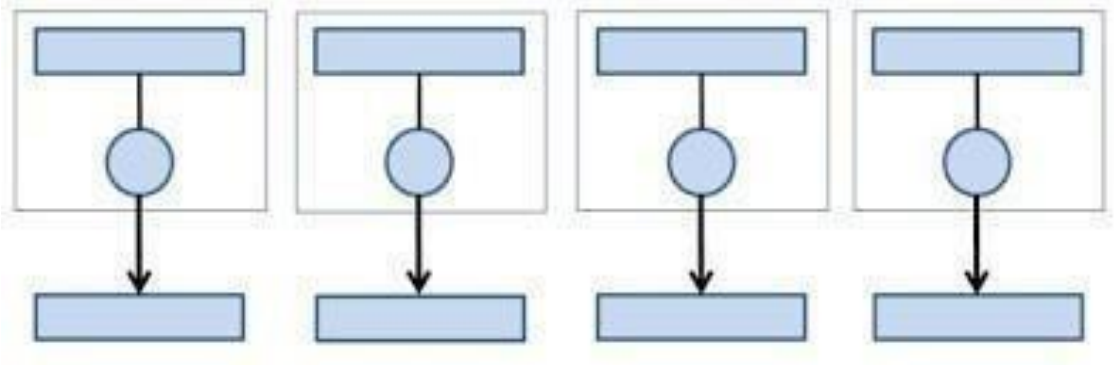

Fig 1.1: Parallel I/O

The data will be transmitted simultaneously in Parallel I/O as shown in Fig 1.1. It ensures faster data transmission.

1.5 Proposed System

This project aims to develop a protocol for **High Speed Serial I/O** which helps:

- To increase the data rate from Mbps to Gbps.
- To decrease the design complexity.
- To design hardware using fewer number of pins on PCB.
- Reduces signal delay to maximum extent possible.

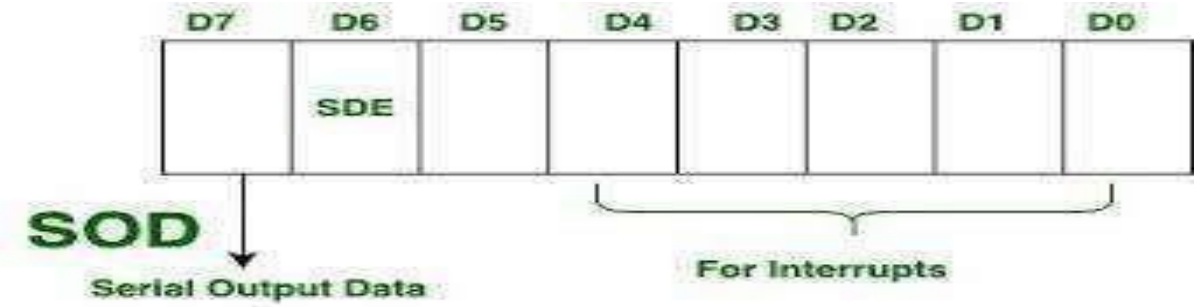

Fig 1.2: Serial I/O

The data will be transmitted sequentially in Serial I/O as shown in Fig 1.2. It ensures ordered data transmission.

CHAPTER -2

HARDWARE AND SOFTWARE REQUIRED

HARDWARE AND SOFTWARE REQUIRED

2.1 Hardware Required

2.11 Field Programmable Gate Arrays (FPGA)

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks. Although one-time programmable (OTP) FPGAs are available, the dominant types are SRAM based which can be reprogrammed as the design evolves. The Fig 2.1 shows a general FPGA board with Virtex-5 processor.

Fig 2.1: Virtex-5 family of FPGA

2.1.2. Virtex-5 FPGA Features

The features of Virtex-5 family of FPGA's are:

- **Input/Output Blocks (SelectIO)**

  IOBs are programmable and can be categorized as follows:

  • Programmable single-ended or differential (LVDS) operation

  • Input block with an optional single data rate (SDR) or double data rate (DDR) register

  • Output block with an optional SDR or DDR register

  • Bidirectional block

  • Per-bit deskew circuitry

  • Dedicated I/O and regional clocking resources

  • Built-in data serializer/deserializer

  The IOB registers are either edge-triggered D-type flip-flops or level-sensitive latches.

- **Configurable Logic Blocks (CLBs)**

  A Virtex-5 FPGA CLB resource is made up of two slices. Each slice is equivalent and contains:

• Four function generators
• Four storage elements
• Arithmetic logic gates
• Large multiplexers
• Fast carry look-ahead chain

The function generators are configurable as 6-input LUTs or dual-output 5-input LUTs. SLICEMs in some CLBs can be configured to operate as 32-bit shift registers (or 16-bit x 2 shift registers) or as 64-bit distributed RAM. In addition, the four storage elements can be configured as either edge-triggered D-type flip-flops or level sensitive latches. Each CLB has internal fast interconnect and connects to a switch matrix to access general routing resources. The Fig 2.2 shows an FPGA Virtex-5 plan.

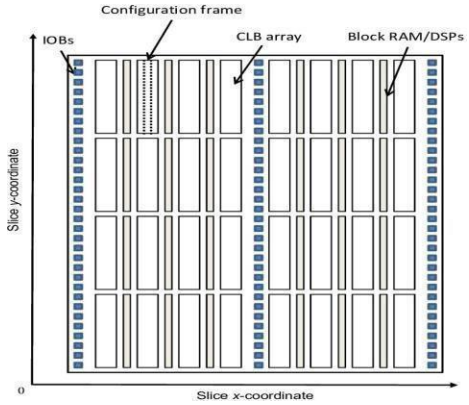

Fig 2.2: FPGA Virtex-5 Floor Plan

- **Block RAM**

The 36 bit true dual-port RAM block resources are programmable from 32 x 1 to 512 x 72, in various depth and width configurations. In addition, each 36-bit block can also be configured to operate as two, independent 18- bit dual-port RAM blocks. Each port is totally synchronous and independent, offering three “read-during-write” modes. Block RAM is cascadable to implement large embedded storage blocks. Additionally, back-end pipeline registers, clock control circuitry, built-in FIFO support, ECC, and byte write enable features are also provided as options

2.1.3 USB JTAG Programmer

JTAG allows device programmer hardware to transfer data into internal non-volatile device memory (e.g. CPLDs). Some device programmers serve a double purpose for programming as well as debugging the device. n the case of FPGAs, volatile memory devices can also be

programmed via the JTAG port, normally during development work. In addition, internal monitoring capabilities (temperature, voltage and current) may be accessible via the JTAG port.

JTAG programmers are also used to write software and data into flash memory. This is usually done using the same data bus access the CPU would use, and is sometimes handled by the CPU. In other case, the memory chips themselves have JTAG interfaces. Some modern debug architectures provide internal and external bus master access without needing to halt and take over a CPU. In the worst case, it is usually possible to drive external bus signals using the boundary scan facility. The Fig 2.3 shows USB JTAG Programmer.

Fig 2.3: USB JTAG Programmer

2.1.4 Optical Fiber Cable (OFC)

A fiber optic cable can contain a varying number of these glass fibers -- from a few up to a couple hundred. Another glass layer, called cladding, surrounds the glass fiber core. The buffer tube layer protects the cladding, and a jacket layer acts as the final protective layer for the individual strand. Fiber optic cables are commonly used because of their advantages over copper cables. Some of those benefits include higher bandwidth and transmit speeds. Fiber optics is used for long-distance and high-performance data networking. Fiber optic cables are now able to support up to 10 Gbps signals. Typically, as the bandwidth capacity of a fiber optic cable increases, the more expensive it becomes.

Fiber optic cables are used mainly for their advantages over copper cables. Advantages include the following:

- They support higher bandwidth capacities.

- Light can travel further without needing as much of a signal boost.
- They are less susceptible to interference, such as electromagnetic interference.
- They can be submerged in water.
- Fiber optic cables are stronger, thinner and lighter than copper wire cables.
- They do not need to be maintained or replaced as frequently.

2.2 Software Required

2.2.1 Xilinx ISE

Xilinx ISE (**I**ntegrated **S**ynthesis **E**nvironment) is a discontinued software tool from Xilinx for synthesis and analysis of HDL designs, which primarily targets development of embedded firmware for Xilinx FPGA and CPLD integrated circuit (IC) product families. It was succeeded by Xilinx Vivado. Use of the last released edition from October 2013 continues for in-system programming of legacy hardware designs containing older FPGAs and CPLDs otherwise orphaned by the replacement design tool, Vivado Design Suite. ISE enables the developer to synthesize their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Other components shipped with the Xilinx ISE include the Embedded Development Kit (EDK), a Software Development Kit (SDK) and Chip Scope Pro. The Xilinx ISE is primarily used for circuit synthesis and design, while ISIM or the Model-Sim logic simulator is used for system-level testing.

2.2.2 User Interface

The primary user interface of the ISE is the Project Navigator, which includes the design hierarchy (Sources), a source code editor (Workplace), an output console (Transcript), and a processes tree (Processes). The Design hierarchy consists of design files (modules), whose dependencies are interpreted by the ISE and displayed as a tree structure. For single- chip designs there may be one main module, with other modules included by the main module, similar to the `main()` subroutine in C++ programs. Design constraints are specified in modules, which include pin configuration and mapping. The Processes hierarchy describes the operations that the ISE will perform on the currently active module. The hierarchy includes compilation functions, their dependency functions, and other utilities. The window also denotes issues or errors that arise with each function. The Transcript window provides status

of currently running operations, and informs engineers on design issues. Such issues may be filtered to show Warnings, Errors, or both.

2.2.3 Aurora protocol (Xilinx)

Aurora is a relatively simple protocol that handles only link-layer and physical issues. It has been designed to allow other protocols such as TCP/IP or Ethernet to ride easily on top of it. It uses one or more high-speed serial lanes.

In addition to the physical interface definition, it defines a packet structure and a recommended procedure for embedding other protocol packets, data striping, and flow control. It defines an initialization procedure to validate links, and describes a procedure for not allowing links with excess errors to be used. It does not have any addressing scheme, so it does not support switching. It also does not define error detection and retry or correction within the data payloads. The protocol was developed by Xilinx and released for unrestricted public use.

2.2.4 ChipScope Pro

ChipScope Pro tool inserts logic analyzer, system analyzer, and virtual I/O low-profile software cores directly into the design, allowing us to view any internal signal or node, including embedded hard or soft processors. Signals are captured in the system at the speed of operation and brought out through the programming interface, freeing up pins for your design. Captured signals are then displayed and analyzed using the ChipScope Pro Analyzer tool.

The ChipScope Pro tool also interfaces with your Keysight Technologies bench test equipment through the ATC2 software core. This core synchronizes the ChipScope Pro tool to Keysight's FPGA Dynamic Probe add-on option. This unique partnership between AMD and Keysight gives you deeper trace memory, faster clock speeds, more trigger options, and system-level measurement capability all while using fewer pins on the FPGA device.

The ChipScope Pro Serial I/O Toolkit provides a fast, easy, and interactive setup and debug of serial I/O channels in high-speed FPGA designs. The ChipScope Pro Serial I/O Toolkit allows you to take bit-error ratio (BER) measurements on multiple channels and adjust high- speed serial transceiver parameters in real-time while your serial I/O channels interact with the rest of the system.

CHAPTER -3

IDENTIFICATION OF HARDWARE RESOURCES INSIDE FPGA

IDENTIFICATION OF HARDWARE RESOURCES INSIDE FPGA

Multi Gigabit Transceiver (MGT) is available on FPGA.

- As shown in Fig 3.1, MGT block has the differential pairing, synchronization, Clock Data Recovery circuit (CDR), Parallel to Serial, Serial to Parallel converters and also a SERDES block.

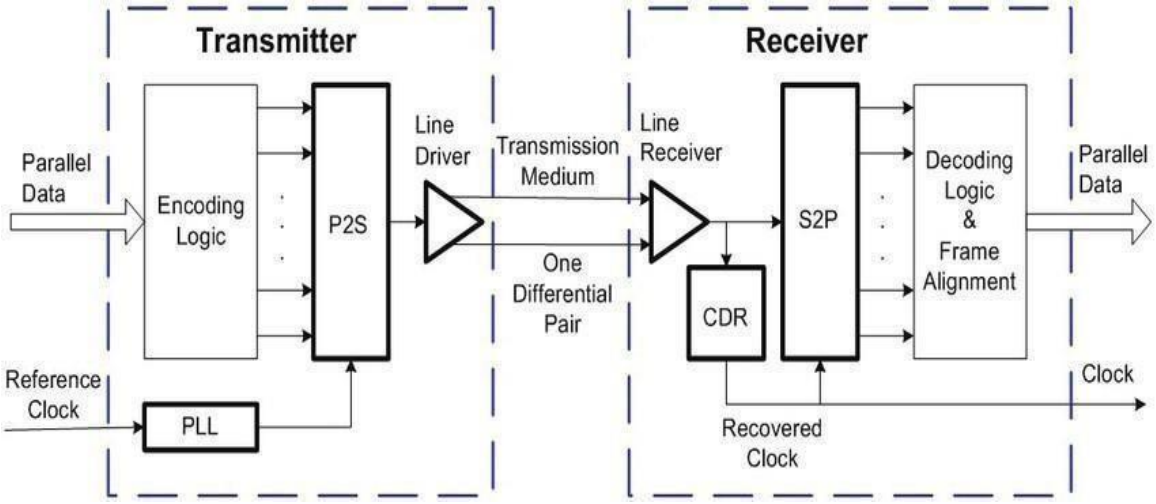

Fig 3.1: MGT Block

3.1 Differential Pair:

If the signal referenced as the positive node has a higher voltage than the one referenced negative, the signal is high, or one. If the negative referenced signal is more positive, the signal is low, or zero. The positive and negative pins are driven with exact complementary signals.

It is much less susceptible to noise. It helps to maintain a constant current flow into the driving IC.

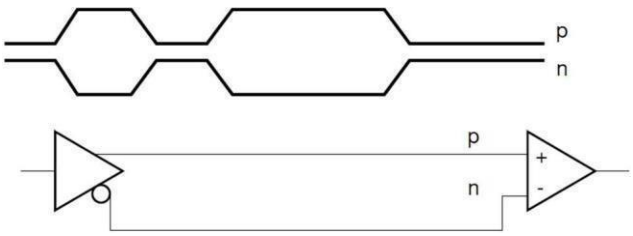

Fig 3.2: Signalling Methods

The signalling method used by differential pair is the usage of complementary signals p (positive) and n (negative) as shown in the Fig 3.2

3.2 Synchronization

There are three basic timing models used for communication between two ICs — system-synchronous, source-synchronous, and self-synchronous.

3.2.1 System-synchronous: Communication between two ICs where a common clock is applied to both ICs and is used for data transmission and reception, as shown in Fig 3.3. The timing model for the System-Synchronous is shown in Fig 3.4.

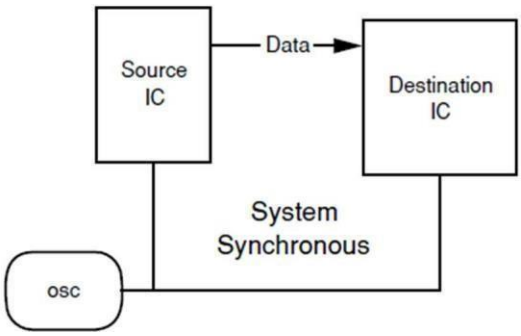

Fig 3.3: System Synchronous Diagram

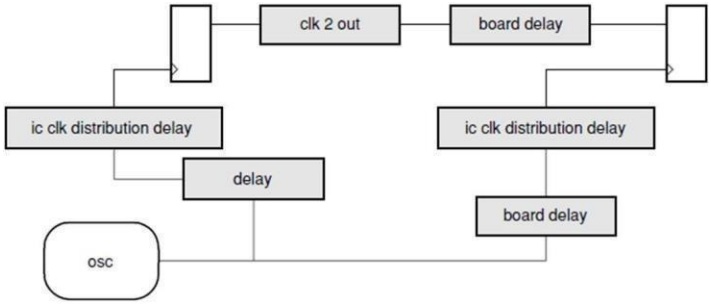

Fig 3.4: System Synchronous Timing Model

3.2.2 Source-synchronous: Communication between two ICs where the transmitting IC generates a clock that accompanies the data. The receiving IC uses this forwarded clock for data reception, as shown in Fig 3.5. The timing model for the Source-Synchronous is shown in Fig 3.6.

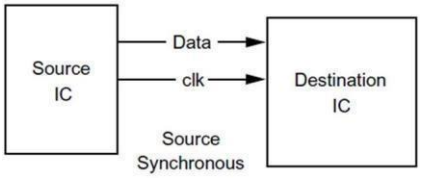

Fig 3.5: Source Synchronous Diagram

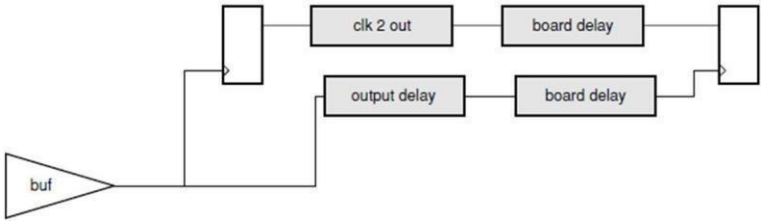

Fig 3.6: Source Synchronous Timing Model

3.2.3 Self-synchronous: Communication between two ICs where the transmitting IC generates a stream that contains both the data and the clock, as shown in Fig 3.7, the timing model of the Self-synchronous is shown in Fig 3.8.

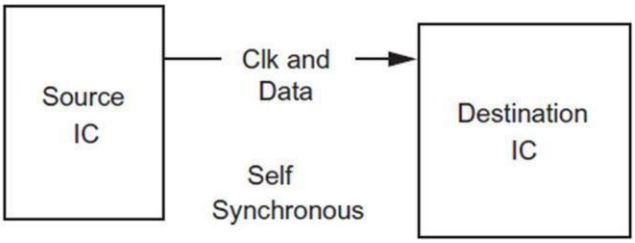

Fig 3.7: Self-synchronous Diagram

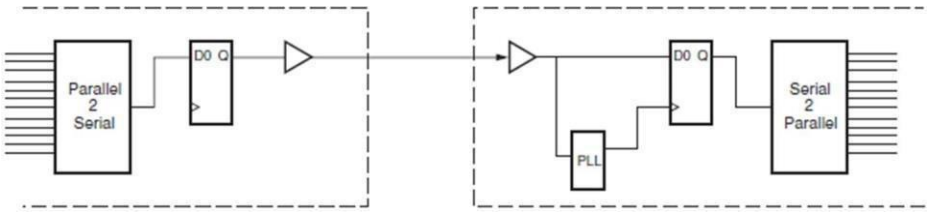

Fig 3.8: Self-synchronous Timing Model

The three main blocks of a self-synchronous interface are parallel-to-serial conversion, serial-to parallel conversion, and clock data recovery.

3.3 Parallel-to-Serial Conversion

There are two main methods of parallel-to-serial conversion—a loadable shift register and revolving selectors, as shown in Fig 3.9.

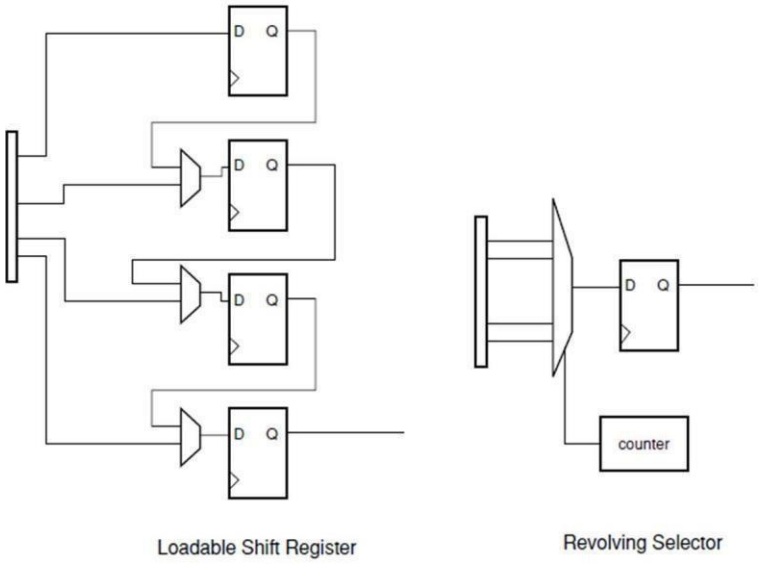

Fig 3.9: Parallel to Serial conversion processes

3.4 Serial-to-Parallel Conversion

Serial-to-Parallel conversion also contains a loadable shift register and revolving selectors, as shown in Fig 3.10.

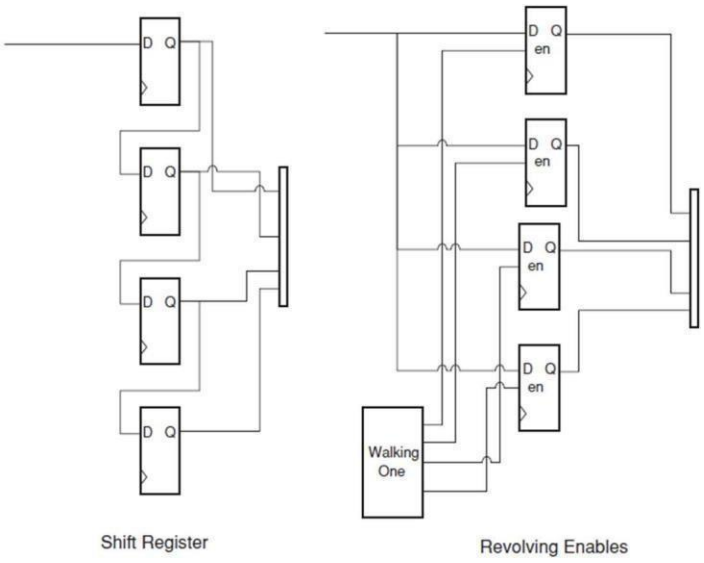

Fig 3.10: Serial to Parallel conversion processes

3.5 Clock Data Recovery Circuit (CDR):

The clock recovery process does not provide a common clock or send the clock with the data. Instead, a phased locked loop (PLL) is used to synthesizes a clock that matches the frequency of the clock that generates the incoming serial data stream. The clock/data recovery waveforms are shown in Fig 3.11.

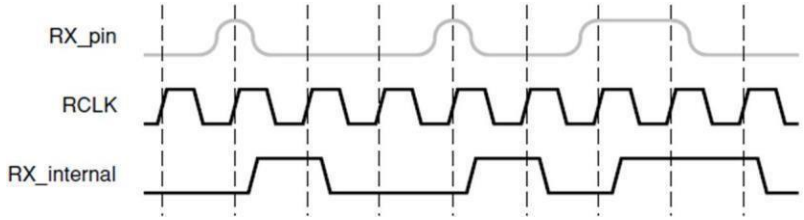

Fig 3.11: Clock/Data Recovery Waveforms

3.6 SERDES Block:

The SERDES Block diagram is shown in Fig 3.12, the various blocks in SERDES are explained as follows.

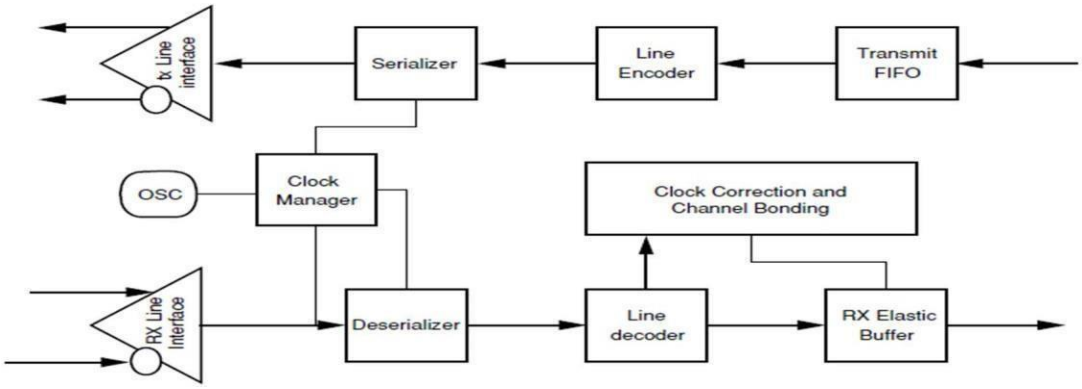

Fig 3.12: SERDES Block Diagram

- **Serializer**: Takes ***n*** bits of parallel data changing at rate ***y*** and transforms them into a serial stream at a rate of ***n*** times ***y***.
- **Deserializer**: Takes serial stream at a rate of ***n*** times ***y*** and changes it into parallel data of width ***n*** changing at rate ***y***.
- **Rx (Receive) Align**: Aligns the incoming data into the proper word boundaries. Several different mechanisms can be used from automatic detection and alignment of a special reserved bit sequence (often called a comma) to user-controlled bit slips.
- **Clock Manager**: Manages various clocking needs including clock multiplication, clock division, and clock recovery.

- **Transmit FIFO (First In First Out)**: Allows for storing of incoming data before transmission.
- **Receive FIFO**: Allows for storing of received data before removal; is essential in a system where clock correction is required.
- **Receive Line Interface**: Analog receive circuitry includes differential receiver and may include active or passive equalization.
- **Transmit Line Interface**: Analog transmission circuit often allows varying drive strengths. It may also allow for pre-emphasis of transitions.
- **Line Encoder**: Encodes the data into a more line-friendly format. This usually involves eliminating long sequences of non-changing bits. May also adjust data for an even balance of ones and zeros.
- **Line Decoder**: Decodes from line encoded data to plain data.
- **Clock Correction and Channel Bonding**: Allows for correction of the difference between the transmit clock and the receive clock. Also allows for skew correction between multiple channels.

3.7 Protocols

Serializer/Deserializers (SERDESs) by themselves are relatively flexible devices. To set them up, we must define an alignment sequence, a clock correction sequence, the line encoding method, and the physical connection, and data will flow between the two transceivers. What data is transmitted to where, what the data means, what is inserted in the data, what can be discarded, are defined by protocols.

There are some standard industry protocols, such as:

- **XAUI**: A 4-channel interface (2.5 Gb/s payload, 3.125 Gb/s wire speed) for 10-Gigabit Ethernet.
- **PCI Express**: Takes the old parallel PCI structure and updates it to a high-speed serial structure. Upper levels of the protocol remain compatible, providing an easy adaptation into legacy PCI systems.
- **Serial RapidIO**: Another serial version of an older parallel spec, RapidIO is quite flexible and sometimes used as a method of interfacing to multiple protocols such as PCI and Infiniband.

- **FiberChannel**: FiberChannel has always been a serial standard, but its speeds have increased over the years. As copper interconnects have advanced, it has also become available on copper as well as fiber optics.
- **Infiniband**: A box-to-box protocol run over either copper or fiber. Infiniband-style cables have become highly popular for multi-gigabit links of a few meters range. The specification allows for a variety of devices and complexity, and includes specifications for repeaters, and switches or hubs to expand the number of connected devices. Infiniband can also be used for complex system configurations. The InfiniBand switches and control consoles are interconnected as shown in Fig 3.13.

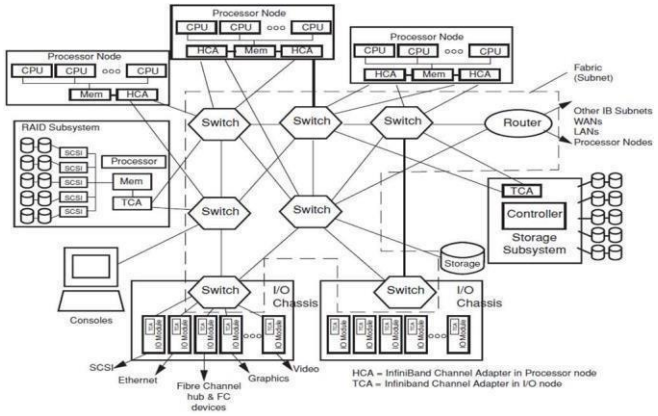

Fig 3.13: Infiniband Switches and Control Consoles

- **Advanced Switching**: A switched fabric protocol built on the same physical and data level protocol as PCI Express. An emerging standard set to be significant in the switched fabric area.
- **PICMG**: PICMG is a consortium of over 600 companies who collaboratively develop open specifications for high-performance standardized backplane architectures. Many of these standards use other industry standards such as PCI and Infiniband.
- **ATCA:** Also known as the Advanced Telecom Computing Architecture or AdvancedTCA, this PICMG standard is a specification for next generation telecommunication cabinets. Its aim is to ease multi-vendor inter operability while providing a very flexible, very scalable system. The standard has various implementations within a common theme. Included are architectures for star-based backplanes, redundant star-based backplanes, and fully connected mesh architectures.

3.8 AURORA Protocol

Aurora is a relatively simple protocol that handles only link-layer and physical issues. It has been designed to allow other protocols such as TCP/IP or Ethernet to ride easily on top

of it. It uses one or more high-speed serial lanes. The overview of the AURORA Channel is shown in Fig 3.14.

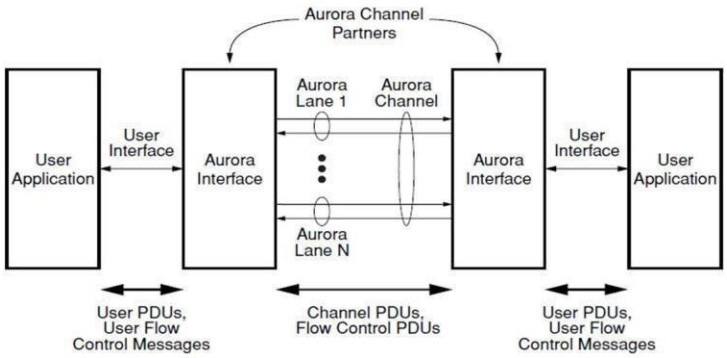

Fig 3.14: AURORA Channel Overview

In addition to the physical interface definition, it defines a packet structure and a recommended procedure for embedding other protocol packets, data striping, and flow control. It defines an initialization procedure to validate links, and describes a procedure for not allowing links with excess errors to be used. It does not have any addressing scheme, so it does not support switching. It also does not define error detection and retry or correction within the data payloads. The protocol was developed by Xilinx and released for unrestricted public use.

AURORA protocol loading other protocols into its data stream is shown in Fig 3.15.

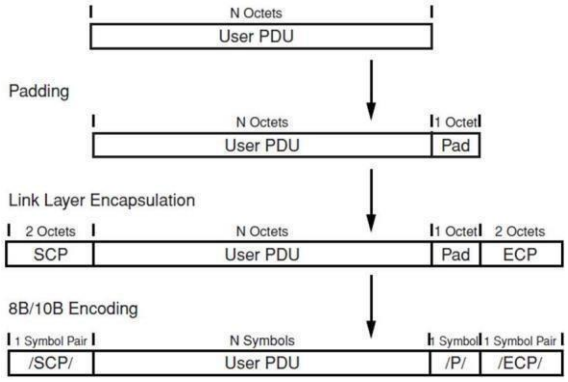

Fig 3.15: Aurora Loading Other Protocols into its Data Stream

3.8.1 AURORA 8B/10B Core Module

Aurora 8B/10B top level (block level) file instantiates Aurora 8B/10B lane module, TX and RX LocalLink modules, global logic module, and wrapper for GTP/GTX transceiver. This top level wrapper file is instantiated in the example design file together with clock, reset circuit and frame generator and checker modules. The top-level architecture of AURORA 8B/10B is shown in Fig 3.16.

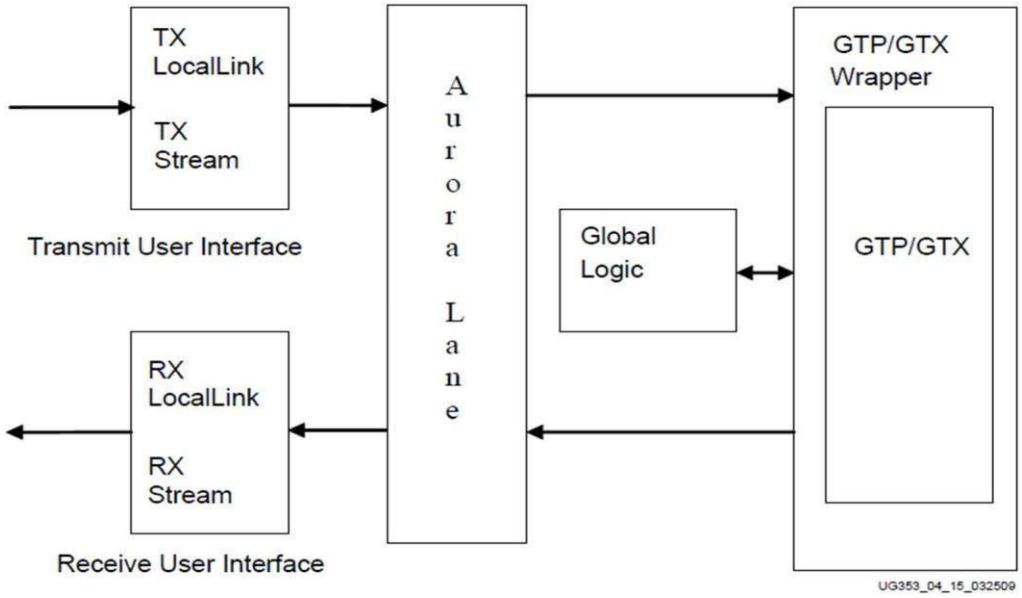

Fig 3.16: Top-Level Architecture of Aurora 8B/10B

3.8.2 AURORA 8B/10B Streaming Interface

Below Fig 3.17 shows the Aurora 8B/10B core configured with a streaming user interface.

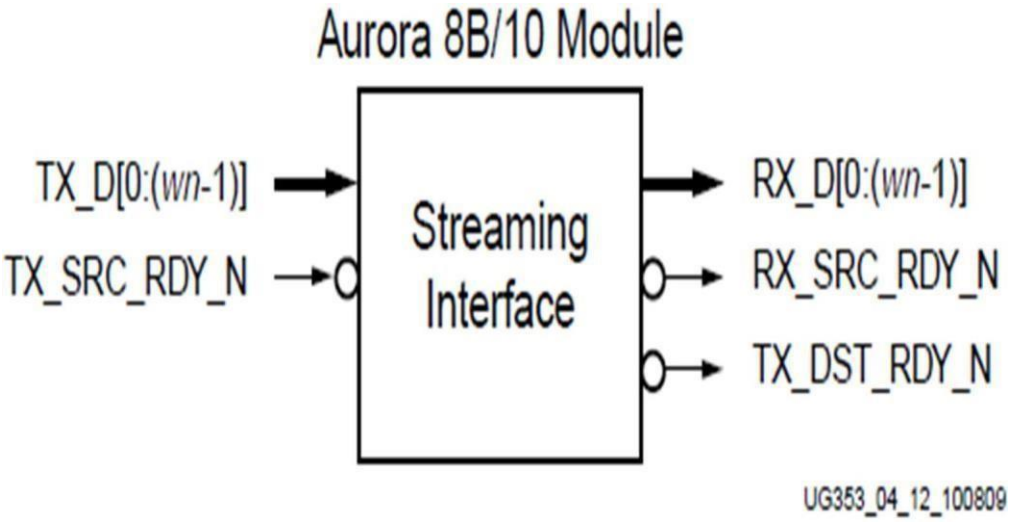

Fig 3.17: Streaming Interface of Aurora 8B/10B

Streaming TX Ports

The below Table 3.1 tells about the transmission signals their direction and description of "Streaming Interface of AURORA 8B/10B"

Table 3.1: Streaming User I/O Ports (TX)

Name	Direction	Description
TX_D[0:(*wn*-1)]	Input	Outgoing data (Ascending bit order).
TX_DST_RDY_N	Output	Asserted (Low) during clock edges when signals from the source will be accepted (if TX_SRC_RDY_N is also asserted). Deasserted (High) on clock edges when signals from the source will be ignored.
TX_SRC_RDY_N	Input	Asserted (Low) when LocalLink signals from the source are valid. Deasserted (High) when LocalLink control signals and/or data from the source should be ignored (active-Low).

Streaming RX Ports

The below Table 3.2 tells about the reception signals their direction and description of "Streaming Interface of AURORA 8B/10B"

Table 3.2: Streaming User I/O Ports (RX)

Name	Direction	Description
RX_D[0:(*wn*-1)]	Output	Incoming data from channel partner (Ascending bit order).
RX_SRC_RDY_N	Output	Asserted (Low) when data and control signals from an Aurora 8B/10B core are valid. Deasserted (High) when data and/or control signals from an Aurora 8B/10B core should be ignored (active-Low).

3.8.3 Transmitting and Receiving Data

The streaming interface allows the Aurora 8B/10B channel to be used as a pipe. Words written into the TX side of the channel are delivered, in order after some latency, to the RX

signal is asserted to send clock compensation sequences. Applications transmit data through the TX_D port, and use the TX_SRC_RDY_N port to indicate when the data is valid (asserted Low). The Aurora 8B/10B core will deassert TX_DST_RDY_N (High) when the channel is not ready to receive data. Otherwise, TX_DST_RDY_N will remain asserted.

When TX_SRC_RDY_N is deasserted, gaps are created between words. These gaps are preserved, except when clock compensation sequences are being transmitted. Clock compensation sequences are replicated or deleted by the GTP/GTX transceiver to make up for frequency differences between the two sides of the Aurora 8B/10B channel.

When data arrives at the RX side of the Aurora 8B/10B channel it is presented on the RX_D bus and RX_SRC_RDY is asserted. The data must be read immediately or it is lost. If this is unacceptable, a buffer must be connected to the RX interface to hold the data until it can be used. The timing diagram of the streaming data transfer is shown in Fig 3.18, and the timing diagram of the reception of data is shown in Fig 3.19.

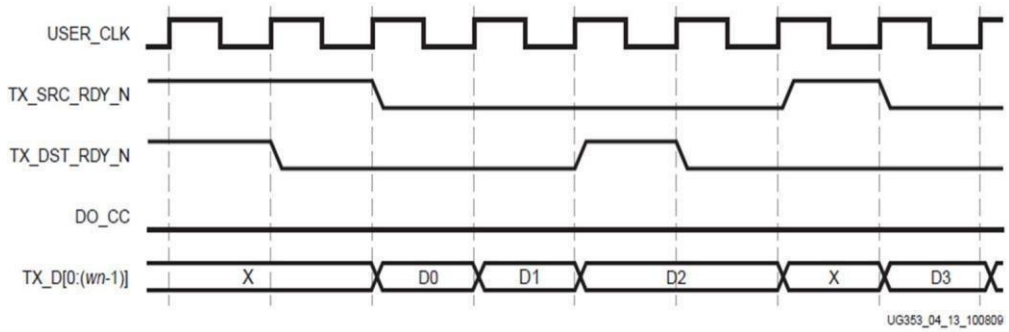

Fig 3.18: Typical Streaming Data Transfer

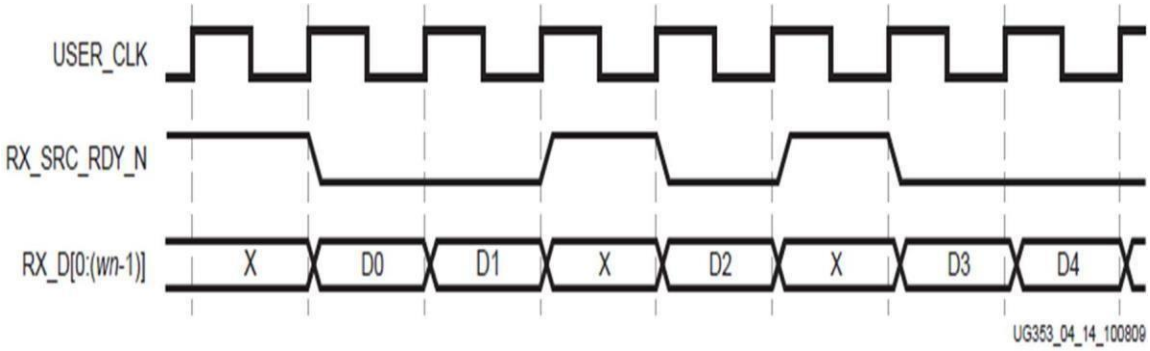

Fig 3.19: Typical Data Reception

CHAPTER -4

INSTALLATION AND TESTING OF AURORA 8B/10B TOOL

INSTALLATION AND TESTING OF AURORA 8B/10B TOOL

To generate an Aurora 8B/10B core with default values using the CORE Generator tool:

- Start the CORE Generator software from a required directory.
- Choose **File → New Project**.
- Type a project name.
- To set project options:
    - From the Part tab, select a silicon family, part, speed grade, and package that
    - supports the Aurora 8B/10B core, for example, Virtex-5 FPGAs.
- After creating the project, locate the Aurora 8B/10B core v5.3 in the taxonomy tree under: /Communication_&_Networking/Serial_Interfaces
- Double-click the core.
- In the Component Name field, enter a name for the core instance.
- Click **Generate**.
- Fig 4.1 shows the CORE Generator AURORA Customization Screen, where different parameters are defined.

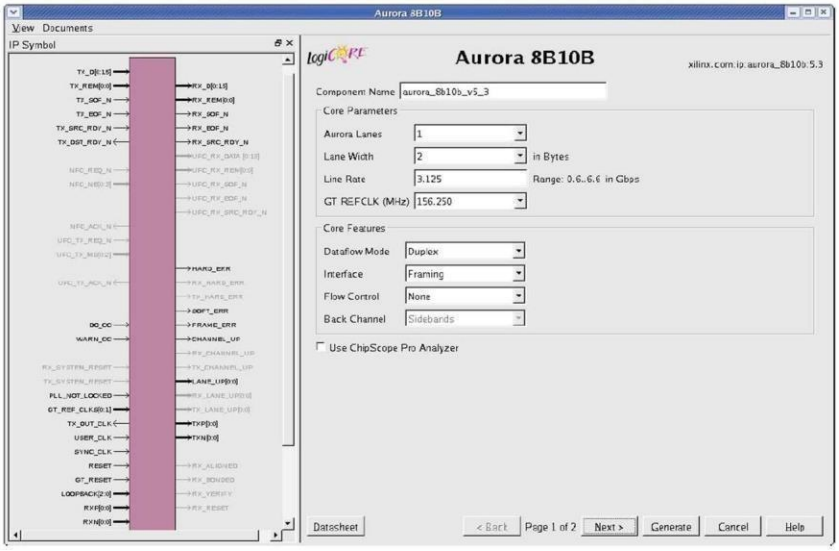

Fig 4.1: CORE Generator Aurora 8B/10B Customization Screen

- **Component Name**

Enter the top-level name for the core in this text box. Illegal names are highlighted in red until they are corrected. All files for the generated core are placed in a subdirectory using this name. The top-level module for the core also uses this name.

Default: aurora_8b10b_v5_3

- **Lane Assignment**

Each numbered column represents a GTP_DUAL/GTX_DUAL or two columns represents GTX Quad and each active box represents an available GTP/GTX transceiver,a s shown in Fig 4.2.

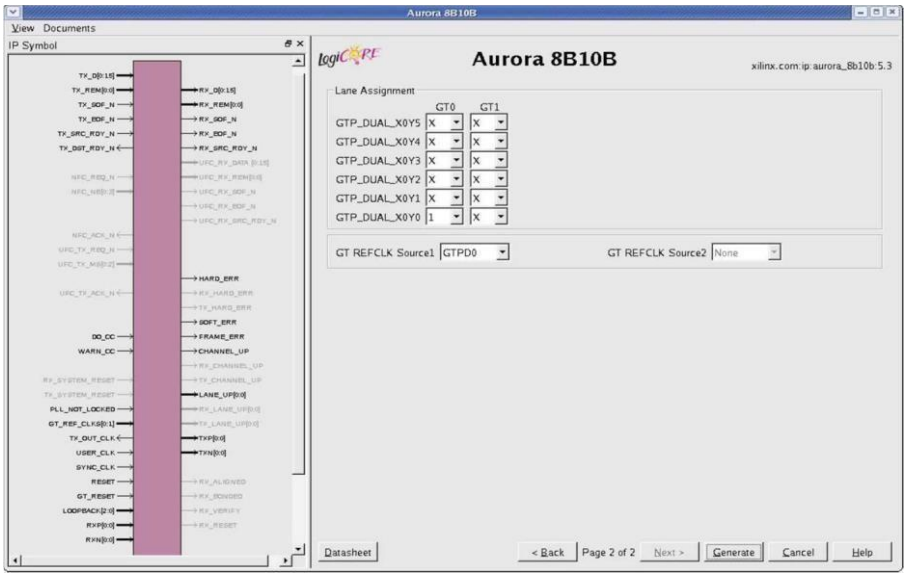

Fig 4.2: Lane Assignment specification

- **Aurora Lanes**

Select the number of lanes (GTP/GTX transceivers) to be used in the core. The valid range depends on the target device selected.

Default: 1

- **Lane Width**

Select the byte width of the GTP/GTX transceivers used in the core. This parameter defines the TXDATA/RXDATA width ofthe transceiver and the user interface data bus width. Valid values are 2 and 4.

Default: 2

- **Interface**

Select the type of data path interface used for the core. Select Framing to use a LocalLink interface that allows encapsulation of data frames of any length. Select Streaming to use a simple word-based interface with a data valid signal to stream data through the Aurora 8B/10B channel.

Default: Framing

- **Dataflow Mode**

Select the options for direction of the channel the Aurora 8B/10B core will support. Simplex Aurora 8B/10B cores have a single, unidirectional serial port that connects to a complementary simplex Aurora 8B/10B core. Available options are RX-only Simplex, TXonly Simplex, and Duplex.

Default: Duplex

- **Back Channel**

Select the options for Back Channel only for Simplex Aurora cores; Duplex Aurora cores does not require this option. The available options are:

• Sidebands

• Timer

Default: Sidebands

- **Flow Control**

Select the required option to add flow control to the core. User flow control (UFC) allows applications to send a brief, high-priority message through the Aurora 8B/10B channel. Native flow control (NFC) allows full duplex receivers to regulate the rate of the data send to them. Immediate mode allows idle codes to be inserted within data frames while completion mode only inserts idle codes between complete data frames.

Available options are listed below:

• None

• UFC

• Immediate Mode - NFC

• Completion Mode - NFC

• UFC + Immediate Mode - NFC

• UFC + Completion Mode - NFC

Default: None

- **Line Rate**

Enter a floating-point value in gigabits per second within the valid range. This determines the un-encoded bit rate at which data will be transferred over the serial link. The aggregate data rate of the core is (0.8 × line rate) × Aurora 8B/10B lanes.

Default: 3.125 Gbps

- **GT REFCLK (MHz)**

Select a reference clock frequency for the transceiver from the drop-down list. Reference clock frequencies are given in megahertz (MHz), and depend on the line rate selected. For best results, select the highest rate that can be practically applied to the reference clock input of the target device.

Default: 156.250 MHz

- **GT REFCLK Source1 and GT REFCLK Source2**

Select reference clock sources for the GTP/GTX_DUALs or GTX Quad from the dropdown list in this section.

• Default: GT REFCLK Source1 - GTXD0; GT REFCLK Source2 - None for Virtex-5 FPGA GTX transceivers

• Default: GT REFCLK Source1 - GTPD0; GT REFCLK Source2 - None for Virtex-5 FPGA GTP and Spartan-6 FPGA GTP transceivers

• Default: GT REFCLK Source1 - GTXQ0; GT REFCLK Source2 - None for Virtex-6 FPGA GTX transceivers

• GTXD0, GTPD0 and GTXQ0 will change based on the selected device and package.

- **Column Used**

Select the appropriate column of transceivers used from the drop down list as shown in Fig 4.3. The column used is enabled only for Virtex-5 TXT or Virtex-6 HXT devices and is disabled for all other devices.

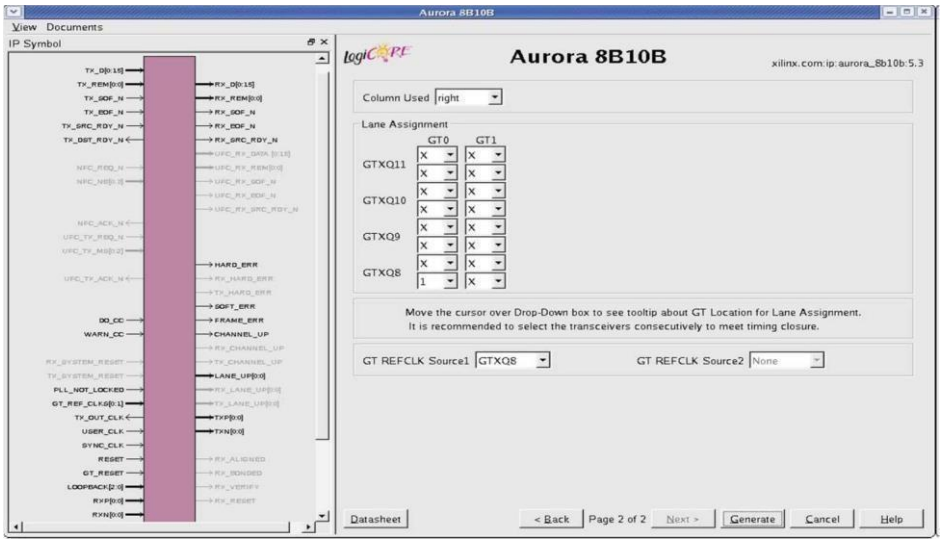

Fig 4.3: Column used specification

Default: left

- **Row Used**

Select the appropriate row of transceivers used from the drop down list, as shown in Fig 4.4. The row used is enabled only for Spartan-6 LXT devices and is disabled for all other devices.

Default: top

Fig 4.4 : Row used specification

- **Use ChipScope Pro Analyzer**

Select to add ChipScope Pro cores to the Aurora 8B/10B core. This option provides users a debugging interface that shows the core status signals in the ChipScope Pro analyzer tool.

Default: Unchecked

- **Generate**

Click Generate to generate the core. The modules for the Aurora 8B/10B core are written to the CORE Generator software project directory using the same name as the top level of the core.

The Aurora 8B/10B core provides a quick way to simulate and observe the behaviour of the core using the provided example design. Prior to simulating the core, the functional (gate level) simulation models must be generated and compilation of the source files is required.

Fig 4.5 and Fig 4.6 shows the simulation console windows.

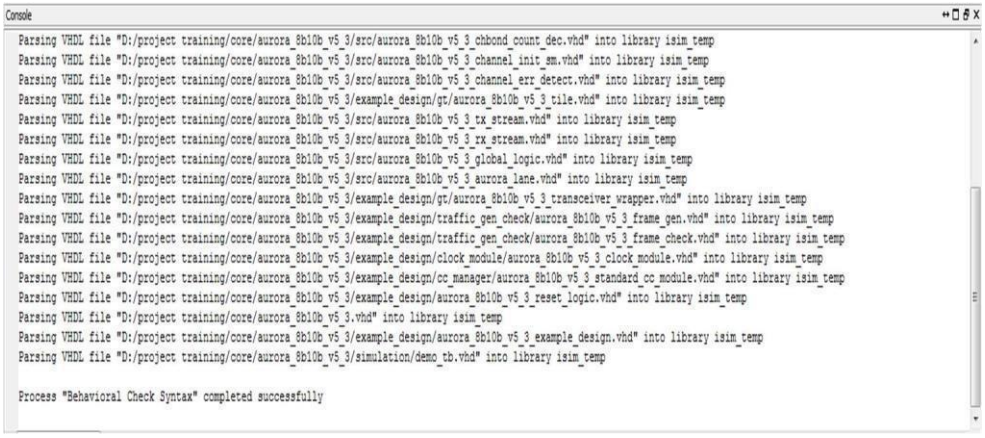

Fig 4.5: Behavioural Simulation Console

Fig 4.6: Simulation Console

4.1 Channel Initialization Problem

Channel_up is asserted when Aurora 8B/10B channel initialization is complete and channel is ready to send data. The Aurora 8B/10B core cannot receive data before channel_up. Lane_up is asserted for each lane upon successful lane initialization, with each bit representing one lane. The Aurora 8B/10B core can only receive data after all lane_up signals are high.

After simulation, we found that the lane_up signal was undefined as shown in Fig 4.7 and thus the channel_up signal was also found to be undefined. Since the channel_up is deasserted, it indicates that the channel is not ready to transmit/receive the data thus data transfer could not take place. As a part of debugging, we started to back-trace all the modules of code and found that lane_up_buffer was acting as an input to the lane_up signal. The lane_up signal was acting as an input to the channel_up due to channel_up signal was deasserted.

.

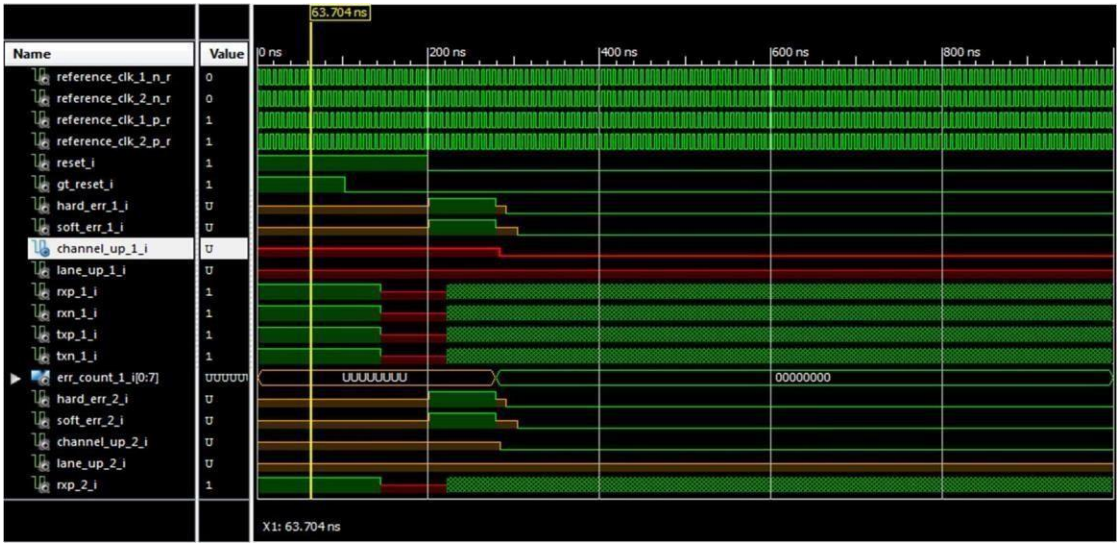

Fig 4.7: Channel Initialization Problem

4.2 D Flip-flop Problem

In the lane_up_buffer code module, there is a D flip-flop module, whose output is lane_up_buffer signal and its input is ready_r signal. Though the clock, reset and input signals were appropriate in the timing diagram, it is found the output signal was undefined, as shown in Fig 4.8 which is an error in the protocol.

We then included the library which has a special component called FDR which is defined in the D flip-flop module.

"UNISIM" is the library that is included in the source code. Inclusion of this library provided us the correct results eradicating the errors produced during the initial simulation.

Fig 4.8: D flip-flop problem

4.3 UNISIM Library

Xilinx users have a library UNISIM (Unified Simulation) which contains behavioral models for low level components. The UNISIM library is used in functional simulation and behavioral simulation when the RTL instantiates device primitives.

The VHDL UNISIM library is located at <Vivado_Install_Dir>/data/vhdl/src/unisims.
It is divided into the following files, which specify the primitives for the Xilinx device families:
The component declarations (unisim_VCOMP.vhdp)
Package files (unisim_VPKG.vhd)
To use these primitives, place the following two lines at the beginning of each file:
library UNISIM;
use UNISIM.Vcomponents.all;

As a part of the first phase of major project, Aurora 8B/10B module was generated and simulated. To observe the transmission of data, certain libraries were defined in the source code to fix the errors encountered.

- In the Fig 4.9, the result wave window is displayed where the data transmission is observed. This shows that the Aurora 8B/10B protocol was successfully simulated and the data transfer happened correctly.
- We observed that the same data which is being transmitted is being received.
- There was a delay of 40 clock cycles between the transmission and reception of data.

Fig 4.9: Data Transmission Wave Window

4.4 Conversion into loop-back mode (Single Tile Module):

- The block diagram of AURORA module is shown in Fig 4.10.

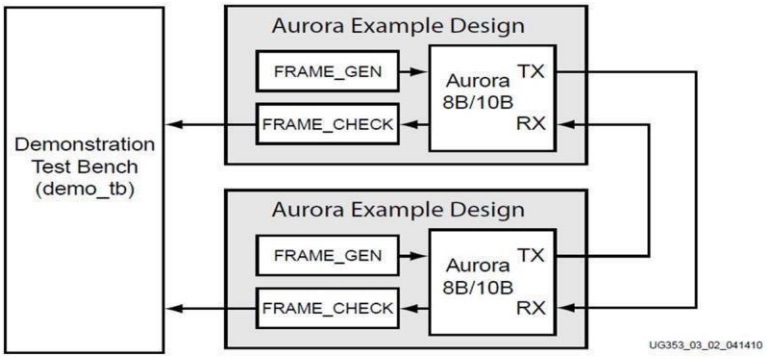

Fig 4.10: Block Diagram of the AURORA Module

- As seen in the block diagram there are two "Aurora Example Design" blocks, one duplicate module should be first deleted.
- Both the aurora example design blocks are same, thus one block can be deleted.
- This block can be deleted by commenting the code where the second block is instantiated, which is shown in Fig 4.11.

```
158
159  --Dut2
160  --
161  --    --Error Detection Interface
162  --    signal  hard_err_2_i        :  std_logic;
163  --    signal  soft_err_2_i        :  std_logic;
164  --
165  --    --Status
166  --    signal   channel_up_2_i       :  std_logic;
167  --    signal   lane_up_2_i          :  std_logic;
168  --
169  --
170  --    --GT Serial I/O
171  ----    signal   rxp_2_i              :  std_logic;
172  ----    signal   rxn_2_i              :  std_logic;
173  ----
174  ----    signal   txp_2_i              :  std_logic;
175  ----    signal   txn_2_i              :  std_logic;
176  --
177  --    -- Error signals from the Local Link packet checker
178  --    signal   err_count_2_i        :  std_logic_vector(0 to 7);
```

Fig 4.11: Commented code for the deletion of duplicate block.

- Tx and Rx pins of the first block are shorted, as shown in Fig 4.12.

```
   --_____________________GT Serial Connections_____________

--    rxn_1_i      <=    txn_2_i;
--    rxp_1_i      <=    txp_2_i;
--    rxn_2_i      <=    txn_1_i;
--    rxp_2_i      <=    txp_1_i;

   rxn_1_i      <=    txn_1_i;
   rxp_1_i      <=    txp_1_i;
```

Fig 4.12: Code for TX and RX pins shorted.

- The hierarchy of the modules after the deletion of the duplicate block is shown in Fig 4.13.

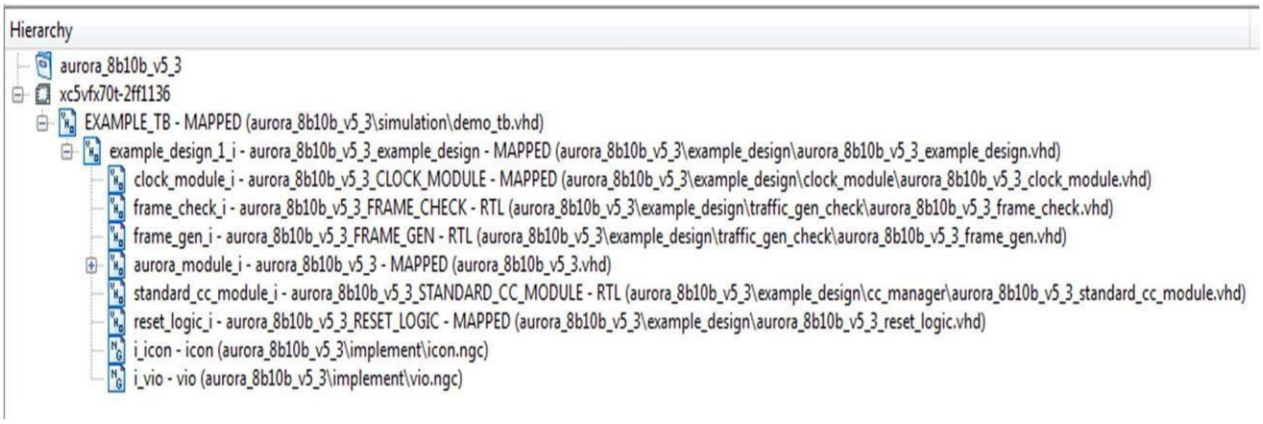

Fig 4.13: Hierarchy of the modules after removal of the duplicate block.

- The output should also be checked after deleting the duplicate block to check whether the functionality of the aurora module remains same or not, and it is observed that the functionality remained same as shown in Fig 4.14.

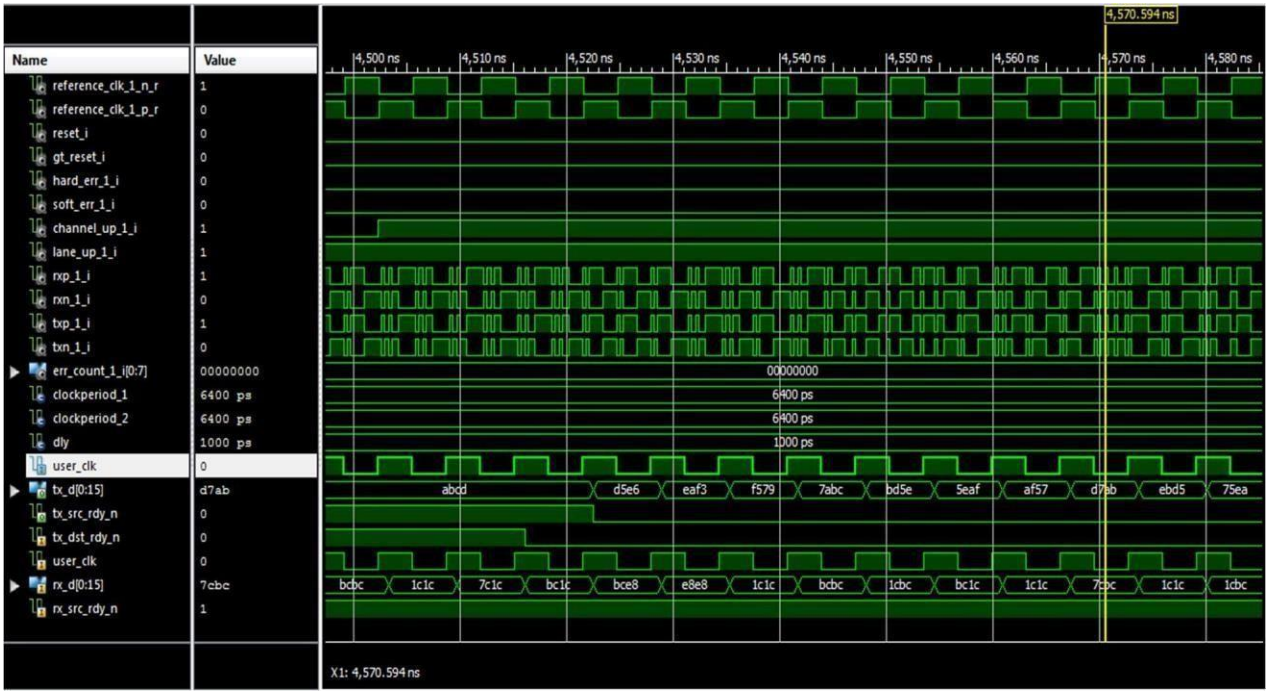

Fig 4.14: Functionality of code verified.

CHAPTER -5

IMPLEMENTATION, INSERTION OF GLUE LOGIC AND TESTING

IMPLEMENTATION, INSERTION OF GLUE LOGIC AND TESTING

5.1 Implementation of a glue logic

- As the second step, we wrote a glue logic of counter as in Fig 5.1.
- As a part of simulation, we first generated the test bench of counter code in ISE tool as in Fig 5.2.
- After performing behavioural check syntax, we had run the simulation and the results observed are in Fig 5.3, 5.4.
- The waveforms were correct and represented their respective correct actions.

```
1  module counter(clk,reset,q);
2  input clk,reset;
3  output reg [3:0]q;
4  always @(posedge clk)
5  begin
6  if(reset==1'b1)
7  q = 0; // state where u are pre defining
8  else
9  q = q+1; // checking the state of the reset and incrementing it
10 end
11 endmodule
```

Fig 5.1: UP-COUNTER Code in Verilog

```
module counter_tb;

   // Inputs
   reg clk;
   reg reset;

   // Outputs
   wire [3:0] q;

   // Instantiate the Unit Under Test (UUT)
   counter uut (
      .clk(clk),
      .reset(reset),
      .q(q)
   );

   initial begin
      clk = 0;
      reset = 0;
      end
   always
      #2 clk=~clk;
   initial begin
      reset = 1;
      #21 reset=0 ;
   end

endmodule
```

Fig 5.2: UP-COUNTER TestBench

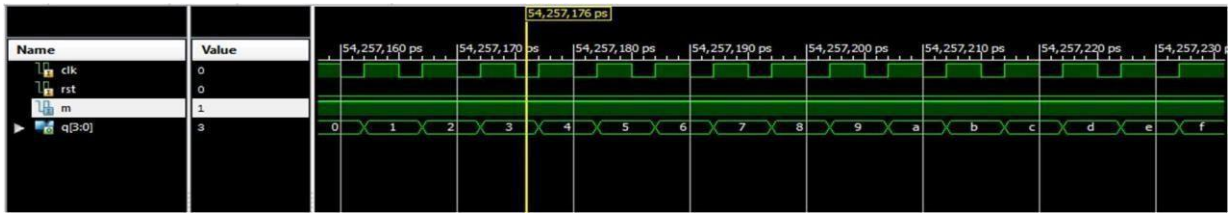

Fig 5.3: UP-COUNTER Output

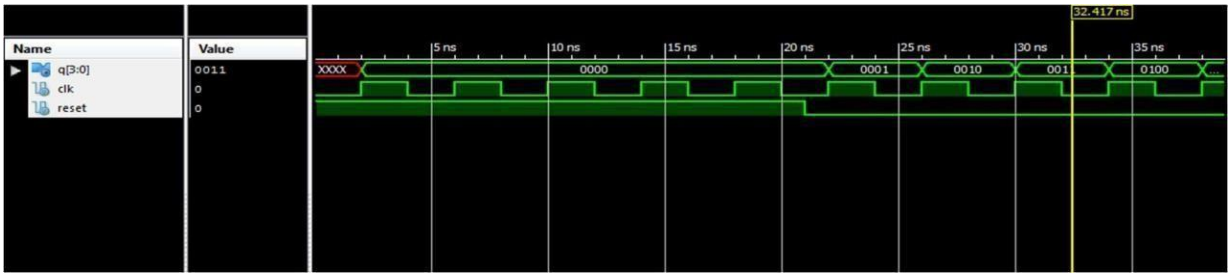

Fig 5.4: UP-COUNTER TestBench Output

5.2 Insertion of Glue Logic.

- The transmission logic that was already existing in the Aurora Core Module is replaced with the glue logic that was written and tested in the previous step.
- "Up-Counter" is the logic used for transmission of data instead of previous logic that transmitted random seed data. This is seen in Fig 5.5.
- After replacement of transmission logic, the module is simulated again to verify the results.
- With the up-counter logic also the data transmission took place with a delay of 40 clock cycles. The result is observed as shown in Fig 5.6.
- Hence, the data transmission and reception is verified to be correct.

```
--*********************************Main Body of Code**********************************

  --Generate RESET signal when Aurora channel is not ready
  reset_c <= RESET or not CHANNEL_UP;

    --_____________________________ Transmit Data _____________________________________
    --Transmit data when TX_DST_RDY_N is asserted.
    --Random data is generated using XNOR feedback LFSR
    --TX_SRC_RDY_N is asserted on every cycle with data
    process(USER_CLK)
    begin
        if(USER_CLK'event and USER_CLK='1') then
            if(reset_c = '1') then
--                data_lfsr_r      <=  X"ABCD" after DLY;  --random seed value
                  count            <= x"0000";
                TX_SRC_RDY_N     <=  '1' after DLY;

            elsif( (not TX_DST_RDY_N)='1' ) then
--                data_lfsr_r <= (not(data_lfsr_r(3) xor data_lfsr_r(12) xor data_lfsr_r(14) xor data_lfsr_r(15)) &
--                                data_lfsr_r(0 to 14)) after DLY;
                  count <= count +'1';
                TX_SRC_RDY_N     <=  '0' after DLY;
            end if;
        end if;
        o_counter <= count;
    end process;

    --Connect TX_D to the DATA LFSR register
    TX_D     <= o_counter; ---(data_lfsr_r);

end RTL;
```

Fig 5.5: Transmission logic replaced by the UP-COUNTER (Glue Logic)

Fig 5.6: Counter logic working correctly along with the data transmission with a clock delay of 40 clock cycles

5.3 Removal of frame_gen & frame_check blocks

- Customize frame_gen block such that there is no frame_check block in the block diagram of AURORA Module as shown in Fig 5.7, 5.8.

```
--    component aurora_8b10b_v5_3_FRAME_GEN
--    port
--    (
--        -- User Interface
--        TX_D             : out  std_logic_vector(0 to 15);
--        TX_SRC_RDY_N     : out  std_logic;
--        TX_DST_RDY_N     : in   std_logic;
--
--
--        -- System Interface
--        USER_CLK         : in  std_logic;
--        RESET            : in  std_logic;
--        CHANNEL_UP       : in  std_logic
--    );
--    end component;
--
--    component aurora_8b10b_v5_3_FRAME_CHECK
--    port
--    (
--        -- User Interface
--        RX_D             : in  std_logic_vector(0 to 15);
--        RX_SRC_RDY_N     : in  std_logic;
--
--
--        -- System Interface
--        USER_CLK         : in  std_logic;
--        RESET            : in  std_logic;
--        CHANNEL_UP       : in  std_logic;
--        ERR_COUNT        : out std_logic_vector(0 to 7)
--
--    );
--    end component;
```

Fig 5.7: FRAME_CHECK and FRAME_GEN code is entirely commented_1

```
--  --Connect a frame checker to the user interface
--    frame_check_i : aurora_8b10b_v5_3_FRAME_CHECK
--    port map
--    (
--        -- User Interface
--        RX_D             =>  rx_d_i,
--        RX_SRC_RDY_N     =>  rx_src_rdy_n_i,
--
--        -- System Interface
--        USER_CLK         =>  user_clk_i,
--        RESET            =>  reset_i,
--        CHANNEL_UP       =>  channel_up_i,
--        ERR_COUNT        =>  err_count_i
--
--    );
--
--
--    --Connect a frame generator to the user interface
--    frame_gen_i : aurora_8b10b_v5_3_FRAME_GEN
--    port map
--    (
--        -- User Interface
--        TX_D             =>  tx_d_i,
--            TX_SRC_RDY_N     =>  tx_src_rdy_n_i,
--        TX_DST_RDY_N     =>  tx_dst_rdy_n_i,
--
--
--        -- System Interface
--        USER_CLK         =>  user_clk_i,
--        RESET            =>  reset_i,
--        CHANNEL_UP       =>  channel_up_i
--    );
```

Fig 5.8: FRAME_CHECK and FRAME_GEN code is entirely commented_2

CHAPTER -6

PORTING CODE ONTO FPGA BOARD

PORTING CODE ONTO THE FPGA BOARD

For porting the code onto FPGA board, we have to follow the FPGA design flow which is a step by step implementation process as follows:

- **Simulation**

The code in Aurora Core module was simulated and the data transmission was observed as a result of simulation output. This is was done as a part of project phase-1.

- **Implementation**

In the ISE tool, to port code onto FPGA board we have to switch from simulation option to implementation option.

- **Synthesize**

As a part of synthesis, the initial .vhd file will be converted to .ngc (native gate constraint) file. The .ngc file code will be in the generic format. There are 3 processes in Synthesize.

View RTL Schematic:

The whole piece of code will be converted into a schematic consisting if LUT'S, Gates, MUX. The overview of RTL schematic design is shown in Fig 6.1.

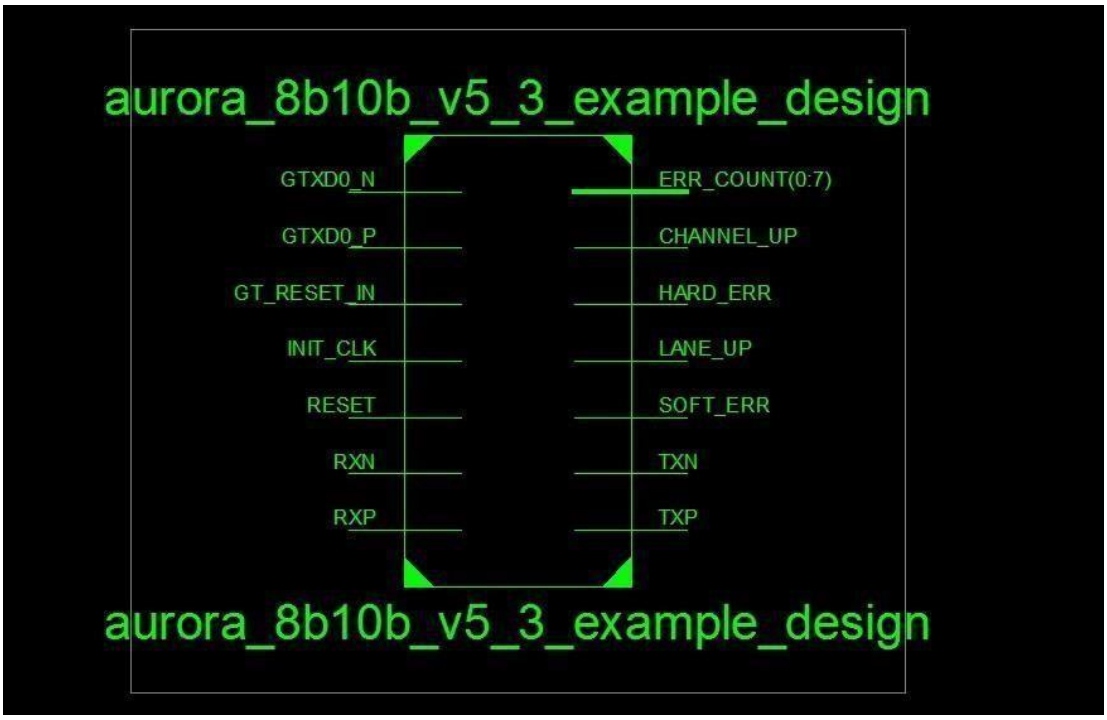

Fig 6.1: Overview of RTL schematic design

The Expanded view of RTL Schematics as a part of View RTL schematic function are shown in Fig 6.2, 6.3.

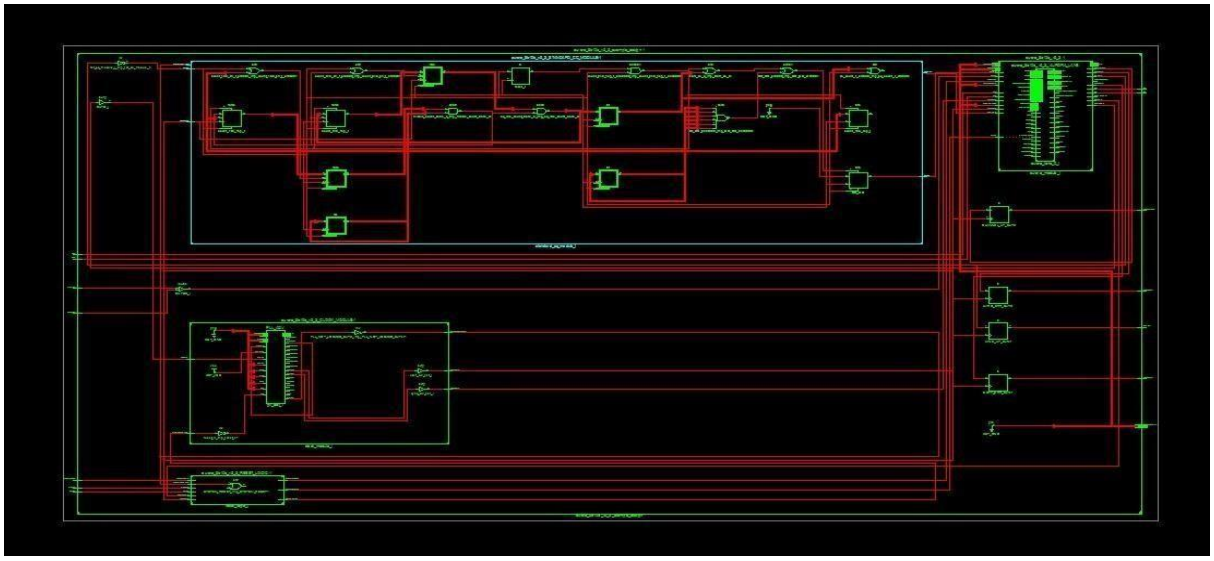

Fig 6.2: Expanded view of RTL Schematic_1

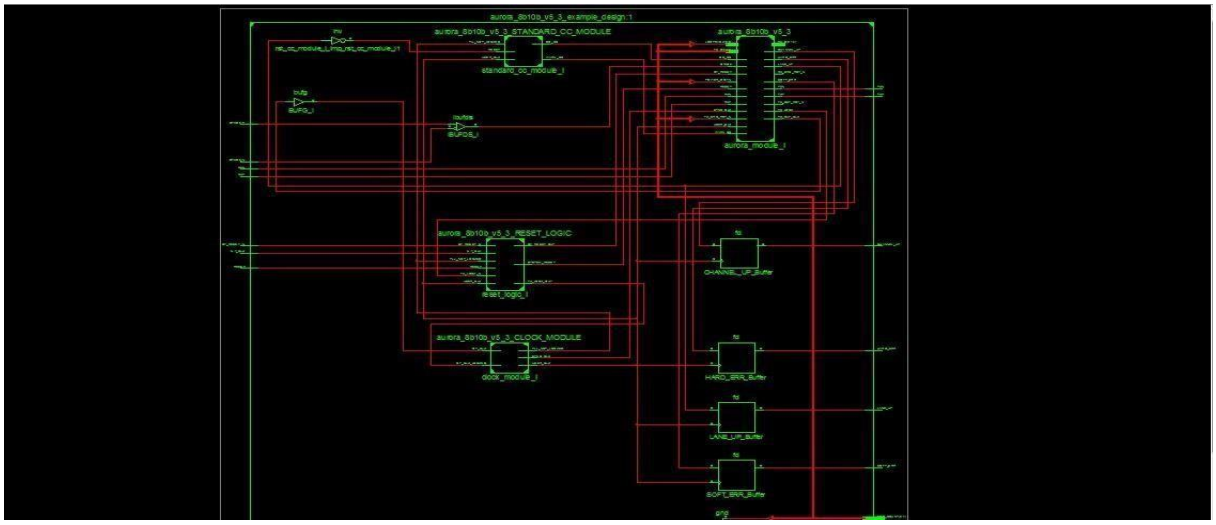

Fig 6.3 : Expanded view of RTL Schematic_2

View Technological Schematic

Check Syntax

This gives the count of errors, to proceed to the further steps. We can continue the process only if the number of errors are 0.

- **Implement Design**

The gate-level netlist will be taken as input. There are 3 processes in Implement Design.

- **Translate:**

The .ngc file is converted into .ngd file. In this step we need to chose either speed optimization or area optimization. Since our concern is speed, we chose speed optimization.

The user constraint file is also created here. The Summary reports of Translate function are shown in Fig 6.4, 6.5.

```
Release 14.7 ngdbuild P.20131013 (nt64)
Copyright (c) 1995-2013 Xilinx, Inc.  All rights reserved.

Command Line: C:\Xilinx\14.7\ISE_DS\ISE\bin\nt64\unwrapped\ngdbuild.exe
-intstyle ise -dd _ngo -sd aurora_8b10b_v5_3/implement -nt timestamp -uc
aurora_8b10b_v5_3/example_design/aurora_8b10b_v5_3_example_design.ucf -p
xc5vfx70t-ff1136-2 aurora_8b10b_v5_3_example_design.ngc
aurora_8b10b_v5_3_example_design.ngd

Reading NGO file "D:/project
training/aurora/aurora_8b10b_v5_3_example_design.ngc" ...
Gathering constraint information from source properties...
Done.

Annotating constraints to design from ucf file
"aurora_8b10b_v5_3/example_design/aurora_8b10b_v5_3_example_design.ucf" ...
Resolving constraint associations...
Checking Constraint Associations...
INFO:ConstraintSystem - The Period constraint <NET GTXD0_left_i PERIOD = 6.4 ns
   HIGH 50%;>
   [aurora_8b10b_v5_3/example_design/aurora_8b10b_v5_3_example_design.ucf(70)],
   is specified using the Net Period method which is not recommended. Please use
   the Timespec PERIOD method.

Done...

Checking expanded design ...

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------
```

Fig 6.4: Translate Summary Report_1

```
  No Partitions were found in this design.

-------------------------------

NGDBUILD Design Results Summary:
  Number of errors:     0
  Number of warnings:   0

Total memory usage is 173532 kilobytes

Writing NGD file "aurora_8b10b_v5_3_example_design.ngd" ...
Total REAL time to NGDBUILD completion:  2 sec
Total CPU time to NGDBUILD completion:   2 sec

Writing NGDBUILD log file "aurora_8b10b_v5_3_example_design.bld"...
```

Fig 6.5: Translate Summary Report_2

- **Map:**

The **.ngd** file is converted into **.pcf file(Physical Constraints File).** The .ngd file which was in generic format will now me mapped as per the physical design we chose. That is all the components will be mapped as per the Virtex-5 FPGA board. The Mapping Summary Reports are shown in Fig 6.6, 6.7.

```
Design Summary
--------------
Number of errors:      0
Number of warnings:    2
Slice Logic Utilization:
  Number of Slice Registers:                 246 out of  44,800    1%
    Number used as Flip Flops:               246
  Number of Slice LUTs:                      191 out of  44,800    1%
    Number used as logic:                    173 out of  44,800    1%
      Number using O6 output only:           152
      Number using O5 output only:            12
      Number using O5 and O6:                  9
    Number used as Memory:                    16 out of  13,120    1%
      Number used as Shift Register:          16
        Number using O6 output only:          16
    Number used as exclusive route-thru:       2
  Number of route-thrus:                      14
    Number using O6 output only:              14

Slice Logic Distribution:
  Number of occupied Slices:                 143 out of  11,200    1%
  Number of LUT Flip Flop pairs used:        292
    Number with an unused Flip Flop:          46 out of     292   15%
    Number with an unused LUT:               101 out of     292   34%
    Number of fully used LUT-FF pairs:       145 out of     292   49%
    Number of unique control sets:            26
    Number of slice register sites lost
      to control set restrictions:            38 out of  44,800    1%
```

Fig 6.6: Mapping summary report_1

```
IO Utilization:
  Number of bonded IOBs:                      15 out of     640    2%
    Number of LOCed IOBs:                     15 out of      15  100%
    Number of bonded IPADs:                    4
      Number of LOCed IPADs:                   2 out of       4   50%
    Number of bonded OPADs:                    2

Specific Feature Utilization:
  Number of BUFG/BUFGCTRLs:                    3 out of      32    9%
    Number used as BUFGs:                      3
  Number of BUFDSs:                            1 out of       8   12%
  Number of GTX_DUALs:                         1 out of       8   12%
    Number of LOCed GTX_DUALs:                 1 out of       1  100%
  Number of PLL_ADVs:                          1 out of       6   16%

Average Fanout of Non-Clock Nets:              2.78

Peak Memory Usage:  542 MB
Total REAL time to MAP completion:  12 secs
Total CPU time to MAP completion:   12 secs
```

Fig 6.7: Mapping summary report_2

- **Place and Route:**

The routing and placement of components basically defines the critical path. The parameters, "**Setup**" and "**Hold**" should be equal to zero to avoid setup and hold violations as shown in Fig 6.8, 6.9. If there are any such violations if might affect in deviation of output from that expected.

```
Device Utilization Summary:

   Number of BUFDSs                           1 out of 8      12%
   Number of BUFGs                            3 out of 32      9%
   Number of GTX_DUALs                        1 out of 8      12%
      Number of LOCed GTX_DUALs               1 out of 1     100%

   Number of External IOBs                   15 out of 640     2%
      Number of LOCed IOBs                   15 out of 15    100%

   Number of External IPADs                   4 out of 690     1%
      Number of LOCed IPADs                   2 out of 4      50%

   Number of External OPADs                   2 out of 32      6%
      Number of LOCed OPADs                   0 out of 2       0%

   Number of PLL_ADVs                         1 out of 6      16%
   Number of Slices                         143 out of 11200   1%
   Number of Slice Registers                246 out of 44800   1%
      Number used as Flip Flops             246
      Number used as Latches                  0
      Number used as LatchThrus               0

   Number of Slice LUTS                     191 out of 44800   1%
   Number of Slice LUT-Flip Flop pairs      292 out of 44800   1%


Overall effort level (-ol):   High
Router effort level (-rl):    High
```

Fig 6.8: Place and route summary report_1

```
Phase  1  : 1459 unrouted;      REAL time: 5 secs

Phase  2  : 1179 unrouted;      REAL time: 6 secs

Phase  3  : 218 unrouted;      REAL time: 6 secs

Phase  4  : 218 unrouted; (Setup:0, Hold:4, Component Switching Limit:0)     REAL time: 9 secs

Updating file: aurora_8b10b_v5_3_example_design.ncd with current fully routed design.

Phase  5  : 0 unrouted; (Setup:0, Hold:4, Component Switching Limit:0)     REAL time: 9 secs

Phase  6  : 0 unrouted; (Setup:0, Hold:4, Component Switching Limit:0)     REAL time: 9 secs

Phase  7  : 0 unrouted; (Setup:0, Hold:4, Component Switching Limit:0)     REAL time: 9 secs

Phase  8  : 0 unrouted; (Setup:0, Hold:4, Component Switching Limit:0)     REAL time: 9 secs

Phase  9  : 0 unrouted; (Setup:0, Hold:0, Component Switching Limit:0)     REAL time: 9 secs

Phase 10  : 0 unrouted; (Setup:0, Hold:0, Component Switching Limit:0)     REAL time: 9 secs
Total REAL time to Router completion: 9 secs
Total CPU time to Router completion: 9 secs
```

Fig 6.9: Place and route summary report_2

Generating Programming File:

A bit file will be generated as a final step of processing. The successful creation of programming file indicates and ensures that there are no software errors in the code and the hardware device is detected.

Boundary Scan:

To check whether the chip I/O pins are working properly or not. It will test the functionality of pins by giving some random test pattern. There are 5 important pins.

- **TDI:** Test Data Input
- **TMS:** Test Mode Select
- **TDO:** Test Data Output
- **TCK:** Test Clock
- **TRST:** Test Reset

Chip Scope:

We are trying to see whether our logic is functioning inside the chip or not through a tool called Chip Scope Pro.

- We also inserted **SFP_DISABLE <= '0'** line in the code for enabling hardware control.

6.1 Defining pins in the User Constraint File

- As a next step, we searched for all the Virtex_5 FPGA pin numbers in ML_50x schematics and got the pin numbers to insert into user constraint file.
- The variable, INIT_CLK is assigned AH7 pin on FPGA.
- GTXD0_P and GTXD0_N are assigned H4, H3 locations.
- Reset is assigned at location E9 and it is given as a PULL_UP
- GT_RESET_IN is assigned location AK7.
- SFP_DISABLE which is used for hardware control is assigned K24 location.
- CHANNEL_UP and LANE_UP are assigned AE24 and AD24 locations respectively and they are depicted as LED's.
- RXP, RXN, TXP, TXN are assigned G1, H1, F2, G2 locations respectively on FPGA board.

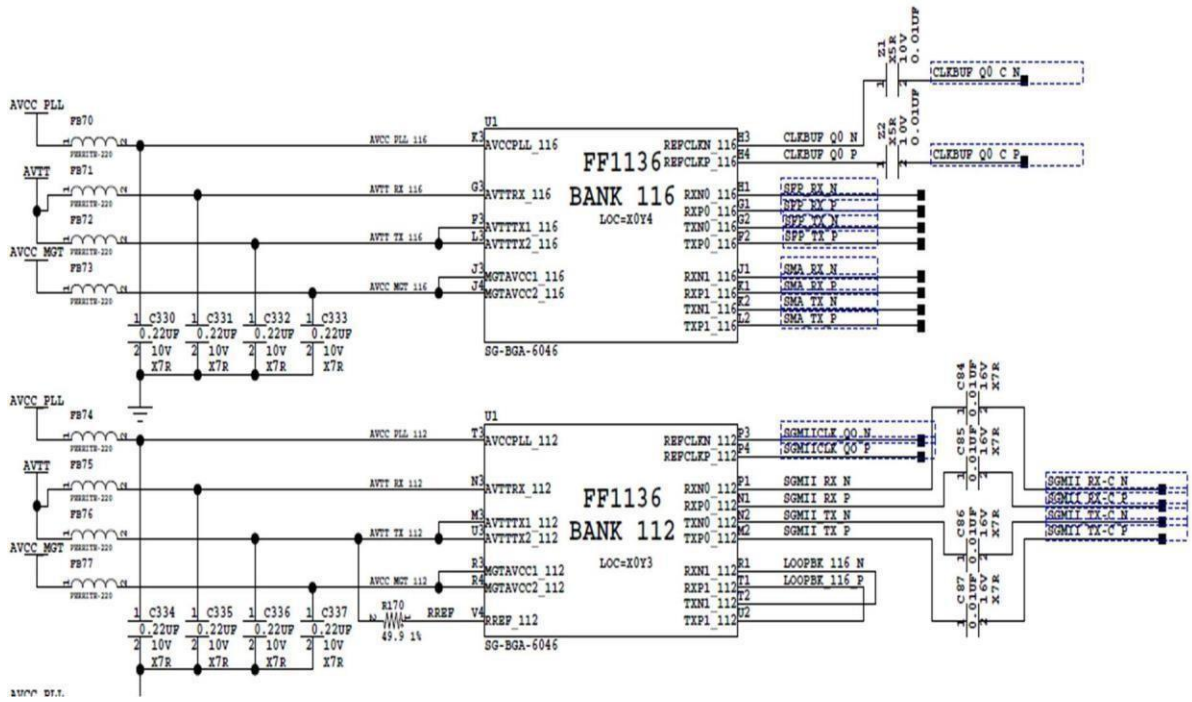

Fig 6.10: Example Schematic Diagram in the ML50x document where locations of RXP, RXN, TXP, TXN are shown

```
62 # User Clock Constraint: Value is selected based on the line rate (3.125 Gbps) and lane width (2-Byte)
63 NET "user_clk_i" TNM_NET = USER_CLK;
64 TIMESPEC TS_USER_CLK_I = PERIOD "USER_CLK" 6.4 ns HIGH 50%;
65 # Sync Clock Constraint: Value is selected based on the line rate (3.125 Gbps) and lane width (2-Byte)
66 NET "sync_clk_i" TNM_NET = SYNC_CLK;
67 TIMESPEC TS_SYNC_CLK_I = PERIOD "SYNC_CLK" 6.4 ns HIGH 50%;
68
69 # 156.25MHz GTX Reference clock constraint
70 NET GTXD0_left_i PERIOD = 6.4 ns HIGH 50%;
71
72 # 50MHz board Clock Constraint
73 NET "reset_logic_i/init_clk_i" TNM_NET = INIT_CLK;
74 TIMESPEC TS_INIT_CLK = PERIOD "INIT_CLK" 20 ns HIGH 50%;
```

Fig 6.11: User Constraint File_1

```
76  ###### No cross clock domain analysis. Domains are not related ##############
77  TIMESPEC "TS_TIG1" = FROM "INIT_CLK" TO "USER_CLK" TIG;
78  ################################ Init Clock Constraint #########################
79  NET INIT_CLK        LOC=AH17;   #50 Mhz Oscillator
80  ################## GTX CLOCK Locations of the pins on ML523 ###############################
81  # GTXD0 : LOC=AL7 --> J78 on Board; LOC=AM7 --> J87 on Board
82  # GTXD1 : LOC=AL5 --> J97 on Board; LOC=AL4 --> J88 on Board
83  # GTXD2 : LOC=AF4 --> J37 on Board; LOC=AF3 --> J28 on Board
84  # GTXD3 : LOC=Y4  --> J47 on Board; LOC=Y3  --> J38 on Board
85  # GTXD4 : LOC=P4  --> J57 on Board; LOC=P3  --> J48 on Board
86  # GTXD5 : LOC=H4  --> J13 on Board; LOC=H3  --> J15 on Board
87  # GTXD6 : LOC=E4  --> J77 on Board; LOC=D4  --> J68 on Board
88  # GTXD7 : LOC=D8  --> J67 on Board; LOC=C8  --> J58 on Board
89  NET GTXD0_P  LOC=H4;
90  NET GTXD0_N  LOC=H3;
91
92  ################################ Resets Buttons ##################################
93  NET RESET           LOC=E9;   #BUTTON
94  NET RESET           PULLUP;
95  NET GT_RESET_IN      LOC= AK7;   #BUTTON
96  NET SFP_DISABLE      LOC=K24;   #BUTTON
97  ################################ Errors Indicators ##############################
98  #NET HARD_ERR      LOC=AF24;   #LED
99  #NET SOFT_ERR      LOC=AG25;   #LED
100 #NET ERR_COUNT[0]    LOC=AA26;   #LED
101 #NET ERR_COUNT[1]    LOC=AA25;   #LED
102 #NET ERR_COUNT[2]    LOC=AK27;   #LED
103 #NET ERR_COUNT[3]    LOC=AK28;   #LED
104 #NET ERR_COUNT[4]    LOC=AJ26;   #LED
105 #NET ERR_COUNT[5]    LOC=AH27;   #LED
106 #NET ERR_COUNT[6]    LOC=AH25;   #LED
107 #NET ERR_COUNT[7]    LOC=AJ25;   #LED
108 ################################ Channel and Lane up Indicators #################
109 NET CHANNEL_UP      LOC=AE24;   #LED
110 NET LANE_UP         LOC=AD24;   #LED
```

Fig 6.12: User Constraint File_2

```
120
121
122 NET RXP  LOC=G1;
123 NET RXN  LOC=H1;
124 NET TXP  LOC=F2;
125 NET TXN  LOC=G2;
126
127 INST aurora_module_i/gtx_wrapper_i/GTX_TILE_INST/gtx_dual_i LOC=GTX_DUAL_X0Y4;
128
```

Fig 6.13: User Constraint File_3

The Fig 6.11, 6.12, 6.13 show the User Constraint File where the pin numbers are allocated to the specified pins as per Fig 6.10.

CHAPTER -7

TESTING HARDWARE(CHIPSCOPE PRO ENVIRONMENT)

TESTING HARDWARE (ChipSCOPE PRO ENVIRONMENT)

- To observe the date transmission clearly in form of data plots and list representations ChipScope Pro environment is used.
- The Fig 7.1, shows the initial ChipScope Pro environment used to verify the working of code and observe the results in various types of representations.

Fig 7.1 : ChipScope Pro Environment_1

- We then created “test” file to give all the required specifications, such as number of bits, number of channels, and the clock signal(user_clk)

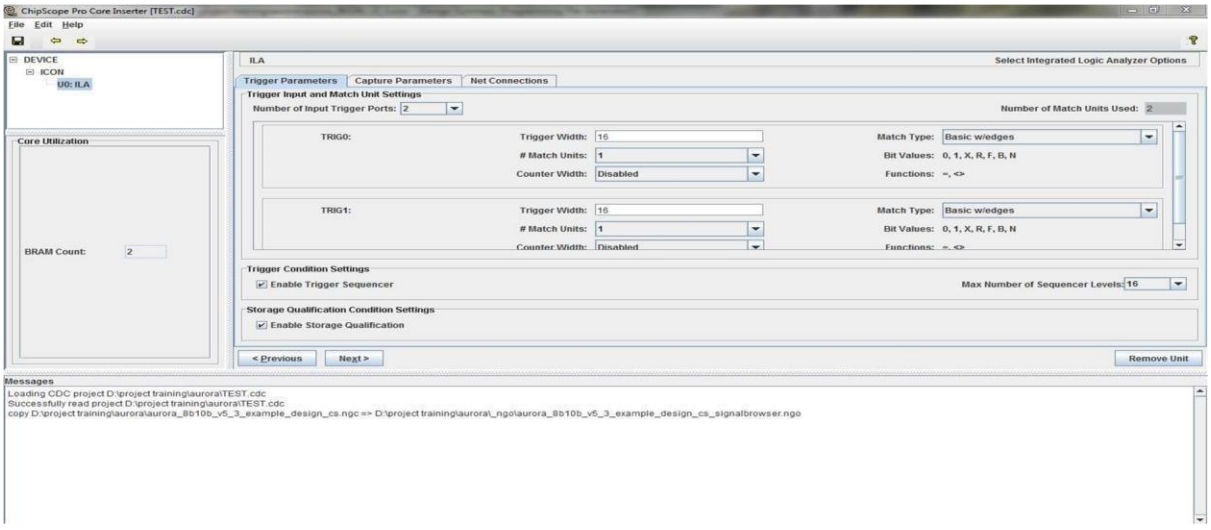

Fig 7.2: ChipScope Pro Environment_2

- As shown in Fig 7.2, we chose the number of trigger ports to be 2 in the trigger parameters section as we want input and output ports.
- For each port we chose trigger width to be 16 i.e the number of input and output data bits are 16 each.
- As shown in Fig 7.3, we chose the number of data samples to be 2048 in the capture parameters section.

- The number of data samples can be varied in the multiples of 2, it can be 1024, 4096, and so on.

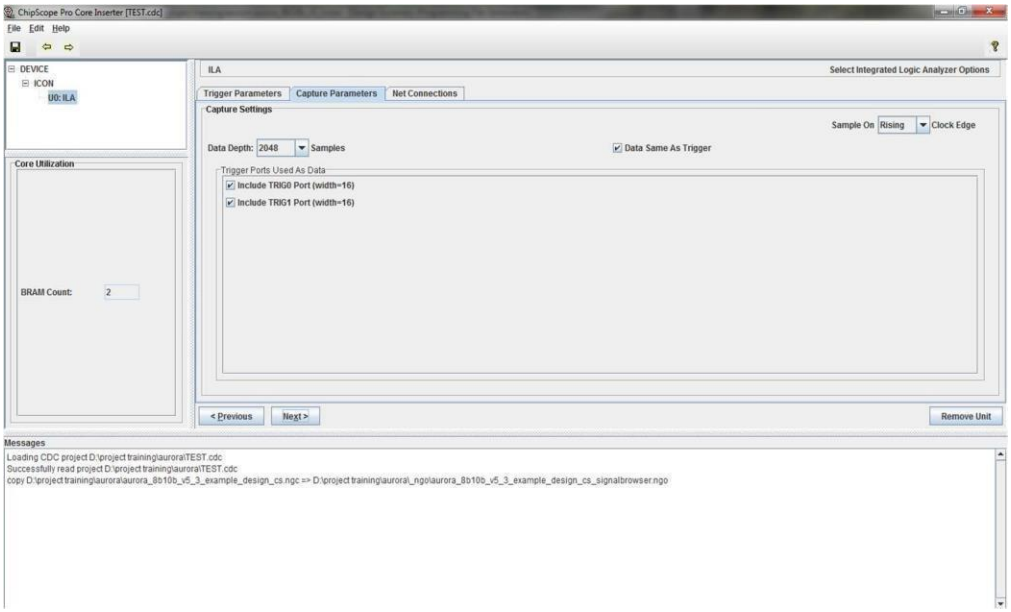

Fig 7.3: ChipScope Pro Environment_3

- As shown in Fig 7.4, in the net connections section we can see the creation of 16 ports i.e CH0 to CH15 in both input and output trigger ports namely TRIG0 and TRIG1
- As observed towards the left side, the BRAM count is set to 2 i.e number of block RAM's utilized in the FPGA board.

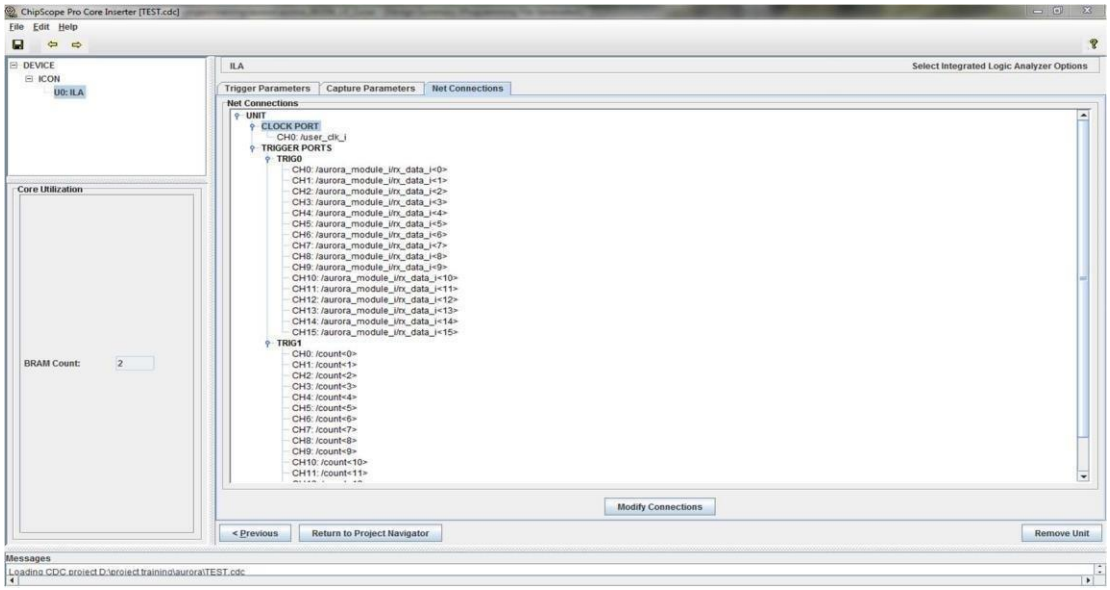

Fig 7.4: ChipScope Pro Environment_4

CHAPTER -8

RESULTS AND DISCUSSIONS

RESULTS AND DISCUSSIONS

After performing all the steps, the results are observed in the following manner.

- There are many in-built options to view the results, but we chose list representation and data plot representation to view the transmission and reception of data in the form of an up-counter data and also to analyze the number of delay clock cycles in streaming of serial data.
- We observed there were 37 clock cycles of delay when the code was ported onto the FPGA hardware and checked.
- After porting the code onto hardware the number of delay clock cycles reduced by 3.
- Hence, we verified and analyzed high-speed serial data communication through streaming interface.
- The Fig 8.1 and Fig 8.2 depict working model of project showing how the FPGA board is connected to PC using USB JTAG Programmer.

Fig 8.1: Working model of the Project_1

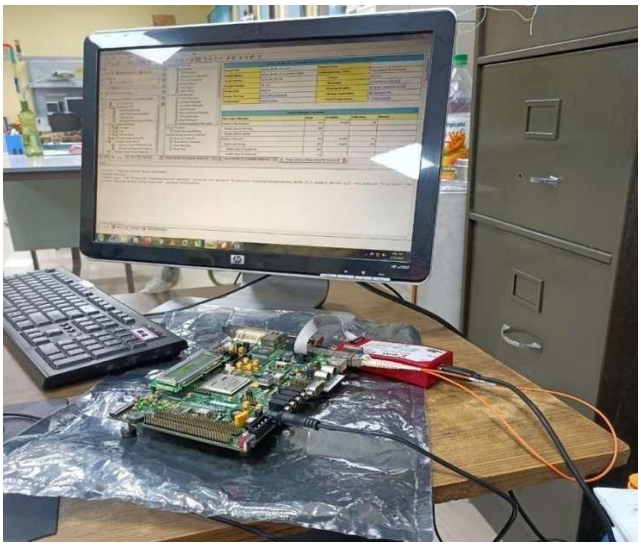

Fig: 8.2: Working model of the Project_2

- In the Fig 8.3, the "sample" column indicates the transmitted data in up-counter fashion.
- The second column named "count" represents the received data with 37 clock cycles of delay.
- The delay can be observed and analysed as : When the input data i.e sample is 0, the output data i.e clock is 37. Similarly, consider an instance, where the sample is 9, the count is 46. The delay is thus (46-9) 37 clock cycles.

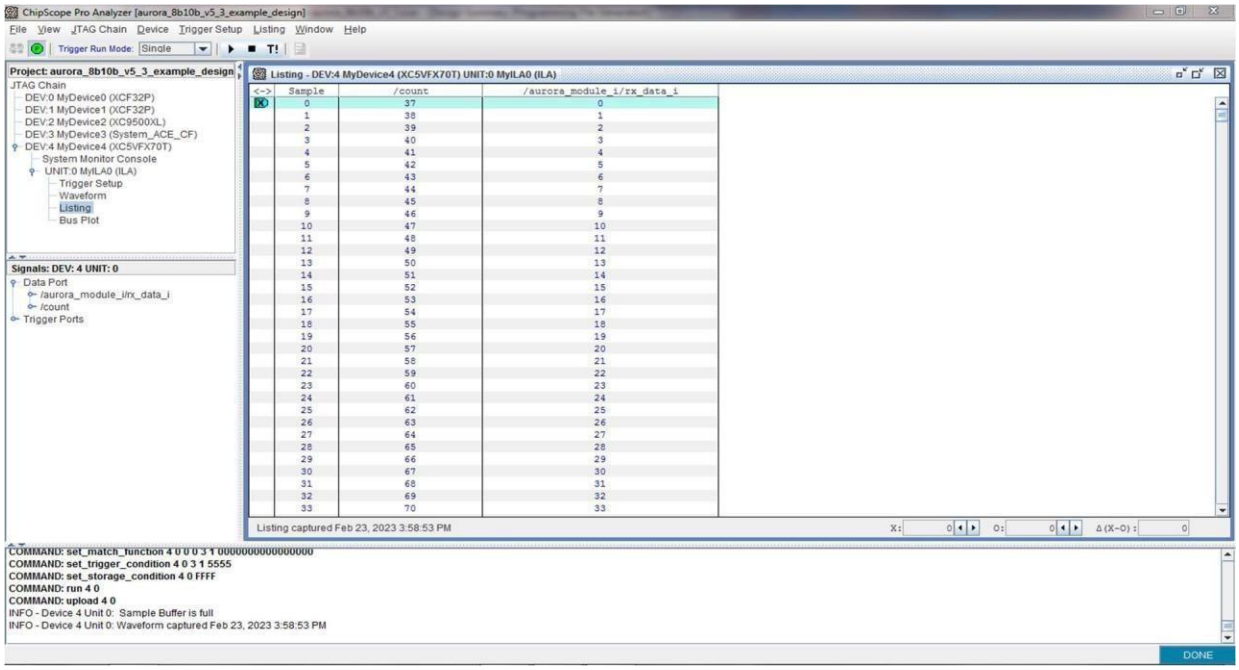

Fig 8.3: List representation of the result

- In the Fig 8.4, the X-axis indicates the transmitted data or the input data as per the written transmission logic.
- The Y-axis indicates the received data or the output data.
- As observed in the table to the left corner of the plot, when the X-axis data i.e input data is 0, the Y-axis data i.e output data is 37. Hence there are 37 clock cycles of delay between the transmitted and received data.

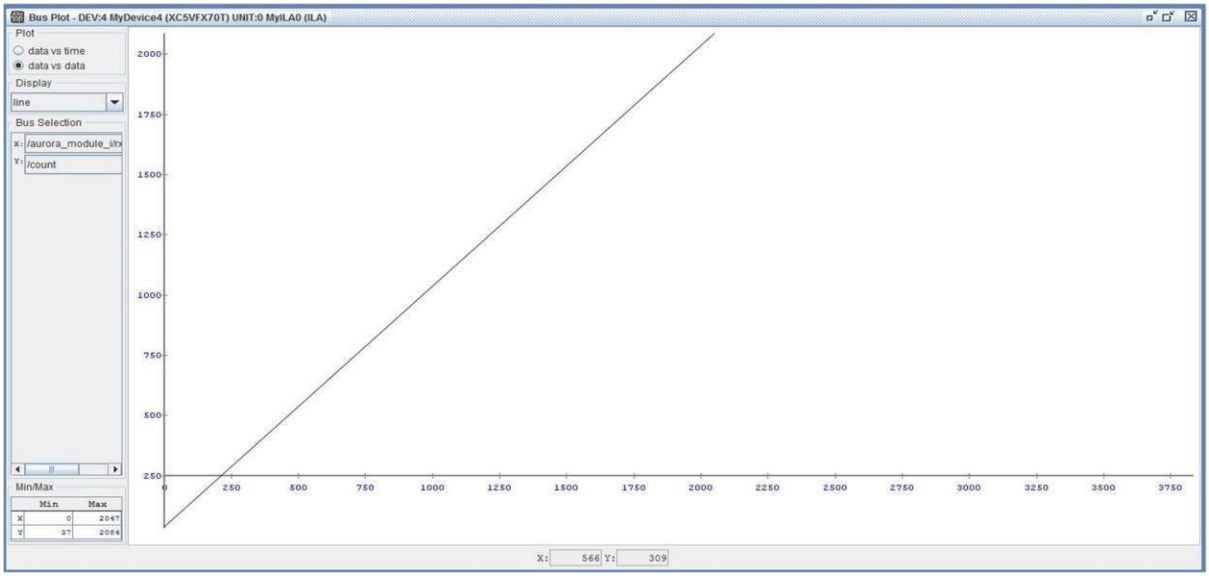

Fig 8.4: Data Plot representation of the result

CHAPTER -9

ADVANTAGES, DISADVANTAGES AND APPLICATIONS OF HIGH -SPEED SERIAL I/O

ADVANTAGES, DISADVANATGES AND APPLICATIONS OF HIGH-SPEED SERIAL I/O

9.1 Advantages

The advantages of High-Speed Serial I/O devices are listed below:

- **Maximum Data Flow**

With wire speeds from 1 to 12 Gb/s and payloads from 0.8 to 10Gb, that is a lot of data transfer.

- **Pin Count**

Pin count is the first problem encountered when trying to move a lot of data in and out of a chip or a board. The number of input and output pins is always limited. Although pin count tends to increase over time, it is never enough to keep up.

- **Simultaneous Switching Outputs**

A designer should consider SSO when using single-ended parallel buses. However, some of those outputs are going to toggle at the same time. When too many-switch simultaneously, ground bounce creates a lot of noise.

A designer could also employ differential signal processing on all I/O to get rid of the SSO problems, but that doubles the pin count. And if the data flow needs are more modest, the designer could use a parallel interface with a usable pin count.

- **Electro Magnetic Inference**

Experience has shown that as clocks get faster, emissions testing gets more difficult. Hence, gigabit design may seem nearly impossible. But a high-speed serial link will usually exhibit less radiated emissions than a large bus that moves at a slower rate. This is because functioning gigabit links require excellent signal integrity.

- **Cost**

Using MGTs will often result in lower overall system costs. With a smaller, cheaper package, the connectors can have fewer pins and the board design may be simpler as well. In the video mixer application, the parallel solution had nine more ICs (integrated circuits) than the serial solution. In this example, the cost of the serial solution is hundreds of dollars less than the parallel solution.

- **Predefined Protocols**

Another benefit of using MGTs is the availability of predefined protocols and interface standards. From Aurora to XAUI, designs already exist for many different needs.

9.2 Disadvantages

The major disadvantage of High-Speed Serial I/O Devices are the signal integrity issues.

- **Signal integrity issues**

It was observed that there was some failure rate with high-speed, multi-gigabit serial designs for a particular application. To improve the odds, we might need to perform analog simulations and use new, more complex bypassing schemes. In fact, we may even need to simulate and model the bypassing scheme.

We can also expect to pay more for impedance-controlled PC (printed circuit) boards, high-speed connectors, and cables. We will have to deal with complications and smaller time bases in digital simulations. And when taking advantage of a predefined protocol, we must plan time for integration and extra gates or Central Processing Unit (CPU) cycles for protocol overhead.

9.3 Applications

The applications of High-Speed Serial I/O devices are listed below:

- **Fiber-Channel (FC)**

Fiber Channel is a high-speed data transfer protocol that provides in-order, lossless delivery of raw block data. It is designed to connect general purpose computers, mainframes and supercomputers to storage devices. The technology primarily supports point-to-point (two devices directly connected to each other). Fig 9.1 shows a Fiber Channel used for connections.

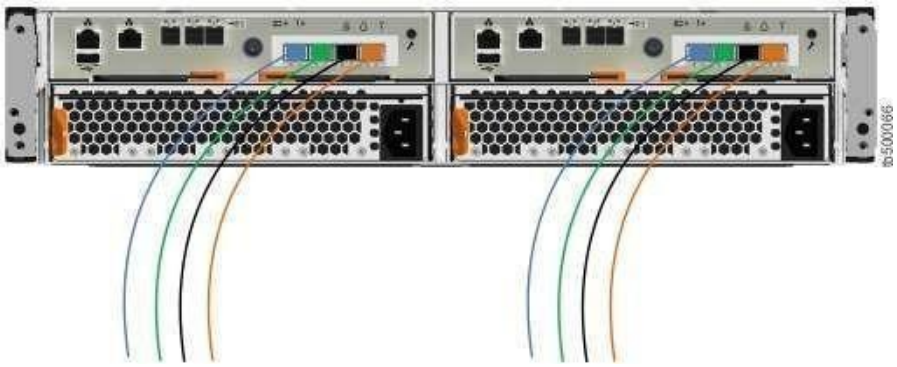

Fig 9.1: Fiber Channel

- **PCI Express**

Peripheral Component Interconnect Express (PCIe or PCI-E) is a serial expansion bus standard for connecting a computer to one or more peripheral devices. PCIe provides lower latency and higher data transfer rates than parallel busses such as PCI and PCI-X. Every device that's connected to a motherboard with a PCIe link has its own dedicated point-to-point connection.

The PCI Express with different number of slots used for connection between a computer and peripheral devices is shown in Fig 9.2

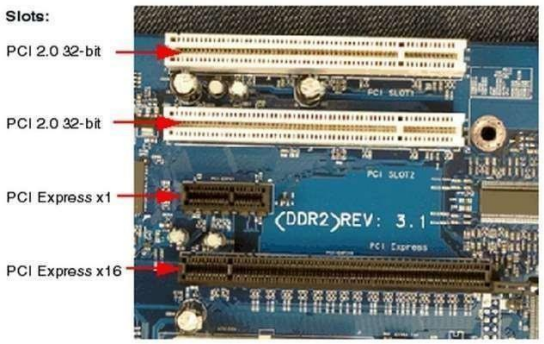

Fig 9.2: PCI Express

- **RapidIO Serial**

The RapidIO architecture is a high-performance packet-switched electrical connection technology. RapidIO supports messaging, read/write. Block Diagram of RapidIO Serial switch is as shown in Fig 9.3

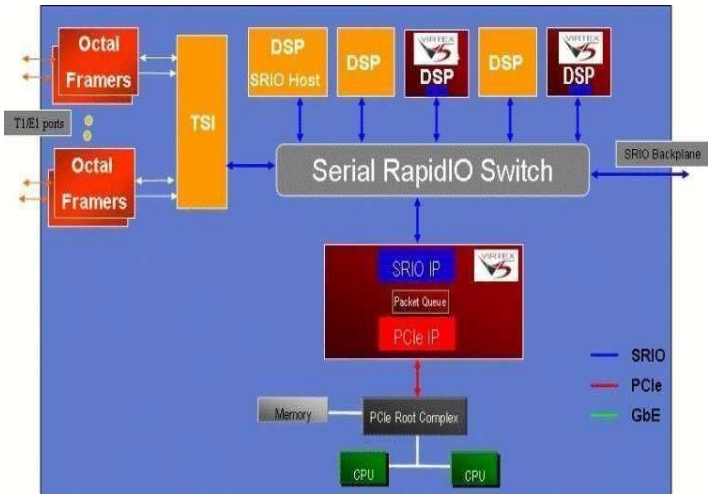

Fig 9.3: RapidIO Serial

- **Advanced Switching Interface**

Advanced Switching encapsulates data packets and attaches a header that routes it through the fabric regardless of the packet format. The header contains a PI (protocol interface) field that is used at the packet destination to determine the packet's format. Thus any protocol can be routed through an ASI network. An example flow diagram involving Advanced Switching Interface is shown in Fig 9.4.

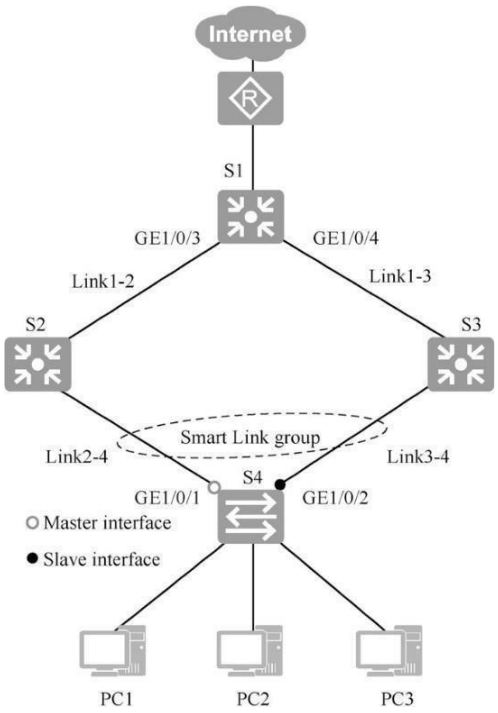

Fig 9.4: Advanced Switching Interface

- **Serial ATA**

SATA is a computer bus interface that connects host bus adapters to mass storage devices such as hard disk drives, optical drives, and solid-state drives. Serial ATA succeeded the earlier Parallel ATA standard to become the predominant interface for storage devices. A Serial ATA cable is as shown in Fig 9.5, which is easier for connection between hosts and storage drives.

Fig 9.5: Serial ATA

- **10-Gb Ethernet**

Ten gigabit Ethernet, also known as 10GbE, is a telecommunication technology that offers data speeds up to 10 billion bits per second, which is 10 times faster than traditional Gigabit Ethernet standard. With such speed, you can efficiently handle any network equipment.

Fig: 9.6: 10GB Ethernet

The 10GB ethernet cable is as shown in Fig 9.6 with its dimensions mentioned in mm (milli meters). High-Speed Serial communication can be achieved easily through these type of ethernet cables.

- **Chip-to-chip Links**

Replacing parallel connections between chips with high-speed serial connections can significantly reduce the number of traces and layers required on a PCB.

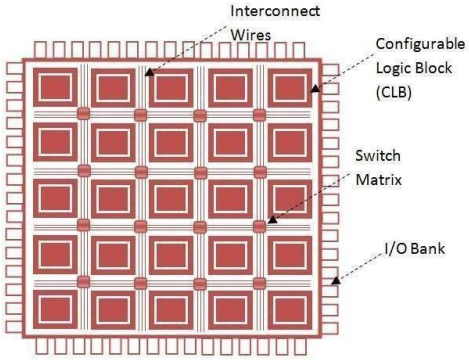

Fig 9.7: Chip to chip Links

The core provides the logic needed to use GTP/GTX transceivers, with minimal FPGA resource cost. Chip to Chip links between different blocks on a PCB is as shown in Fig 9.7.

- **Board to board and back plane Links**

The Aurora 8B/10B core uses standard 8B/10B encoding, making it compatible with many existing hardware standards for cables and backplanes. Aurora 8B/10B cores can be scaled, both in line rate and channel width, to allow inexpensive legacy hardware to be used in new, high performance systems.

Board to board and backplane links between different PCB's is shown in Fig 9.8.

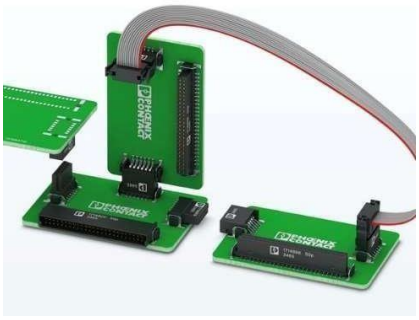

Fig 9.8: Board to board and backplane links

- **Simplex Connections (Unidirectional)**

In some applications there is no need for a high-speed back channel. The Aurora protocol provides several ways to perform unidirectional channel initialization, making it possible to use the GTP/GTX transceivers when a back channel is not available, and to reduce costs due to unused fullduplex resources. The Fig 9.9 shows a simplex connection between a TX (Transmitter) and RX (Receiver) of two devices.

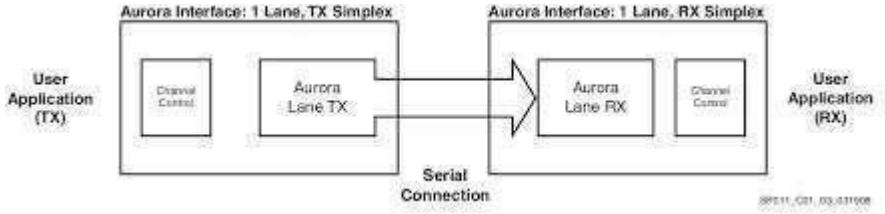

Fig: 9.9: Simplex Connections

- **ASIC Applications**

The Aurora protocol is not limited to FPGAs, and can be used to create scalable, high performance links between programmable logic and high-performance ASICs. The simplicity of the Aurora protocol leads to low resource costs in ASICs as well as in FPGAs, and design resources like the Aurora bus functional model (ABFM 8B/10B) with compliance testing make it easy to get an Aurora channel up and running. Various types of ASIC applications used in day to day life are shown in the Fig 9.10.

Fig 9.10: ASIC Applications

CHAPTER -10

CONCLUSION AND FUTURE SCOPE

CONCLUSION AND FUTURE SCOPE

10.1 Conclusion

In this project FPGA hardware resources are identified and are used appropriately to generate a 8B/10B Aurora core module in XILINX and then simulate the protocol using the software. After simulation is successful the core module is again tested using a glue logic which is then ported onto FPGA and tested.

Compared to Serial I/O, Parallel I/O provides faster communication but involves design issues and uses more number of pins on the PCB which also makes the devices costlier. Whereas, Serial I/O uses lesser number of pins but takes time to transmit data bits which increases the delay. Hence this project helps in generating a protocol which implements High-Speed Serial I/O which reduces the delay, reduces the design issues, is easier and convenient to use. It provides reliable and effective communication.

10.2 Future Scope:

Serial designers must contend with signal integrity, smaller time bases, and possibly the need for extra gates and additional CPU cycles. However, multi-gigabit advantages in box- to-box and chip-to-chip communication far outweigh the perceived shortcomings. For example, high speed, fewer pins, lower EMI, and lower cost make it the ideal choice in many communication designs. These advantages will ensure its continued use in communication applications far into the future and speed can be increased to higher level.

REFERENCES

[1] Abhijit Athavale, "High-Speed Serial I/O Made Simple", Priliminary Edition 1.0

[2] User Guide, "LogiCORETM IP Aurora 8B/10B", v5.3, ug_353,

[3] User Guide, "LogiCORETM IP Aurora 8B/10B", v5.3, ds_637,

[4] Xilinx, ML505/6/7 Block Diagram SCHEM

[5] https://docs.xilinx.com/r/1.5-English/pg314-versal-mrmac/Transceiver-SerDes- Interface, Date of Access: 29-12-2022

[6] https://en.wikipedia.org/wiki/Parallel_I/O, Date of Access: 10-11-22

[7] https://en.wikipedia.org/wiki/Virtex_(FPGA), Date of Access: 25-10-22

[8] https://www.geeksforgeeks.org/serial-i-o-lines-in-8085-microprocessor/, Date of Access: 18-11-22

[9] https://ieeexplore.ieee.org/document/1345340, Date of Access: 29-12-22

[10] https://www.sciencedirect.com/topics/engineering/clock-recovery-circuit, Date of Access: 20-11-22

[11] https://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html, Date of Access: 18-11-22

[12] https://www.xilinx.com/products/intellectual-property/aurora64b66b.html, Date of Access: 29-11-22