Indian Journal of Science and Technology, *Vol 11(23), DOI: 10.17485/ijst/2018/v11i23/125648, June 2018*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Development and Implementation of Parallel to Serial Data Transmitter using Aurora Protocol for High Speed Serial Data Transmission on Virtex-7 FPGA

C. Mani Pradhitha* and S. Kolangiammal

SRM University, Kattankulathur, Chennai - 603203, Tamil Nadu, India; cpraditha@gmail.com, kolangiammal.s@ktr.srmuniv.ac.in

Abstract

The objective of this paper is Development and Implementation of parallel to serial data converter using Aurora protocol for high speed serial data transmission at the rate of 3.125Gbps by using architectural features of Virtex-7 FPGA. It involves the study and configuring the Xilinx Core Generator Tool to achieve the required high speed serial data transmission by using of Aurora 8b/10b Protocol & Multi-Gigabit Transceivers present in Virtex-7 FPGA. Firstly, a 192-bit parallel data is generated using simulators, which is implemented using VHDL language. The 192-bit data is sent to Asynchronous First-In First-Out (AFIFO) as input and produces an output of 32-bit parallel data. This data is sent to the aurora module in parallel form as successive frames (i.e. 6 frames, each frame consists of 4 bytes). Finally, the 192-bit parallel data is transmitted to the receiver module serially over fiber optic cable at the rate of 3.125Gbps using architectural features of virtex7 FPGA. Finally, the data is transmitted on dual independent aurora channels and the entire logic will be tested for its complete functionality in standalone mode by porting on to the Virtex-7 FPGA based custom Hardware.

Keywords: Aurora Protocol, Independent Aurora Channels, Serial Data Transmission, Virtex-7 FPGA, AFIFO, GTX TILE, MGT's

1. Introduction

Communication plays one of the most significant role in day to day life. There should be a transmitter and receiver in order to communicate. There are two ways of transmission i.e., serial communication and parallel communication. In earlier days in order to achieve the high speed we used parallel transmission. In parallel transmission, Binary data consisting of 1s and 0s are arranged into groups of n bits each. By grouping, we can send n bits data at a time instead of one. We use n wires to send n bits once at a time. That way each bit will have its own wire, and all n bits of one group can be transmitted with each clock pulse from one device to another. For n = 9. Typically, the nine wires are bundled in a cable with a connector at each end. The main advantage of parallel transmission is its speed. Parallel transmission increases the transfer speed by a factor of n over serial transmission[6]. A significant disadvantage of parallel transmission is its cost. Parallel transmission requires n communication lines (wires in the example) to transmit the data stream. Because this is very expensive, parallel transmission is usually suitable to short distances.

In serial transmission one bit follows another, so only one communication channel is required rather than n channels to transmit data between two communicating devices. The main advantage of serial over parallel transmission is that with only one communication channel, serial transmission reduces the cost of transmission roughly by a factor of n. Since communication within devices is parallel, parallel to serial conversion devices are required at the interface between the sender and the line (parallel-to-serial) and between the line and the receiver (serial-to-parallel). Serial transmission occurs in any one of two ways: asynchronous or synchronous.

**Author for correspondence*

Multi-gigabit transceivers are transceivers which can be used for data transmission which are present in the FPGA as hard IP and they can run over longer distances. They can be used with the help of the soft IP and their main function is to convert the parallel data into serial data and this action can be performed by configuring the MGT's using aurora core. These MGT's are present in the Virtex-7 FPGA as hard IPs and we have to interface to our application by configuring hard IPs using soft IPs such as aurora core to achieve the high speed serial data transmission. The MGT's present in the FPGA are configured as either transmitter or receiver to perform the data transmission.

In Virtex-7 FPGAs the Rocket IO GTX transceivers are a power-efficient transceiver. The GTX transceiver is highly configurable and it is integrated very tightly with the programmable logic resources of the FPGA. GTX transceivers are placed as Quad transceiver GTX_QUAD tiles in Virtex-7 FXT devices. This configuration allows four transceivers to share a single PLL with the TX and RX functions of both, size reduction and power consumption. Multi Gigabit Transceiver (MGT) is a Serialiser /Deserialiser (SerDes) it can be operated at serial bit rates above 1 Gigabit/second.

2. AFIFO

2.1 Introduction

A synchronous FIFO is a FIFO where the same clock is used for both writing and reading[6]. An asynchronous FIFO uses different clocks for writing and reading. AFIFO is used for clock transition and non-asymmetric ratio of the data in and out. AFIFO is generated using BRAM and customized according to our application.

FIFOs are used commonly in electronic circuits for flow control and buffering which is from hardware to software. In hardware form a FIFO primarily consists of a set of read and writes pointers, storage and control logic. Storage may be SRAM, flip-flops, latches or any other forms of storage. For FIFOs of significant size, a dual-port SRAM is usually used where one port is used for writing and the other is used for reading[1]. FIFOs are often used to safely pass data from one clock domain to another asynchronous clock domain. There are multi-asynchronous clock design techniques to pass FIFO data.

2.2 Asynchronous FIFO

An asynchronous FIFO is a FIFO design where data values are written to a FIFO buffer from one clock domain and the data values are read from the same FIFO buffer from another clock domain, where the two clock domains are asynchronous to each other. Asynchronous FIFOs are used to safely pass data from one clock domain to another clock domain.

2.3 FIFO Usage and Control

- Write Operation:
  When write enable is asserted and the FIFO is not full, data is added to the FIFO from the input bus (DIN) and write acknowledge (WR_ACK) is asserted. It fills with data, if the FIFO is continuously written to without being read. If FIFO is not full then Write operations are successful. If FIFO is full and a write is initiated, the request is ignored, then the overflow flag is asserted and there will be no change in the state of the FIFO.
- Read Operation:
  When read enable is asserted and the FIFO is not empty, data is read from the FIFO on the output bus (DOUT), and the valid flag (VALID) is asserted. The FIFO empties If the FIFO is continuously read without being written. When the FIFO is not empty, then the read operations are successful. The read operation is ignored when the FIFO is empty and a read is requested, the underflow flag is asserted and there will be no change in the state of the FIFO.

2.4 Implementation of AFIFO with FWFT

The 512X36 Bram is used as the output of FIFO should be given as 32-bit word to AURORA protocol[1]. The input to the FIFO is 192-bit PD word data so FIFO will write these 192-bits and it is given as six 32 bit words output to AURORA protocol. FIFO consists of 2 clocks read clock (156.25 MHz) and write clock (50 MHz). The read clock is always faster than write clock[6]. If FIFO is not Full and write enable is high, then write flag is generated similarly if FIFO is not Empty and read enable is set to one then read flag is generated. Here writes flag is given with 1-bit latency and read flag is given with 2-bit latency so read

flag is generated after only write flag generates. If read flag and read clock is enabled, then address pointer goes on increasing similarly for write flag also. The binary address is converted to grey code address to clear data in read and write flags and never in unknown state due to asynchronous relationship of the read and write clocks. If read flag is enabled, then pointer moves on increasing until it reads last grey code similarly if write flag is enabled then pointer moves on increasing until it writes last grey code.

From Figure 1, read enable is checked and if FIFO is not empty for every transfer of read date then FIFO data will be read into user data automatically without any external operation being executed. By using this algorithm, the AFIFO is implemented.

3. Aurora Protocol

3.1 Introduction

The Aurora 8B/10B protocol is implemented by the Logic CORE™ IP Aurora 8B/10B core using the high-speed serial Transceivers on the Virtex-7 LXT, SXT, FXT, and TXT Family[5]. The Aurora 8B/10B core is a lightweight, scalable, link layer protocol for high-speed serial communication. The protocol is open source and can be implemented using Xilinx® FPGA technology. The protocol is typically used in applications requiring, low-cost, simple, high rate, data channels. In our application virtex7 FXT is used because it supports High-performance embedded systems with advanced serial connectivity.

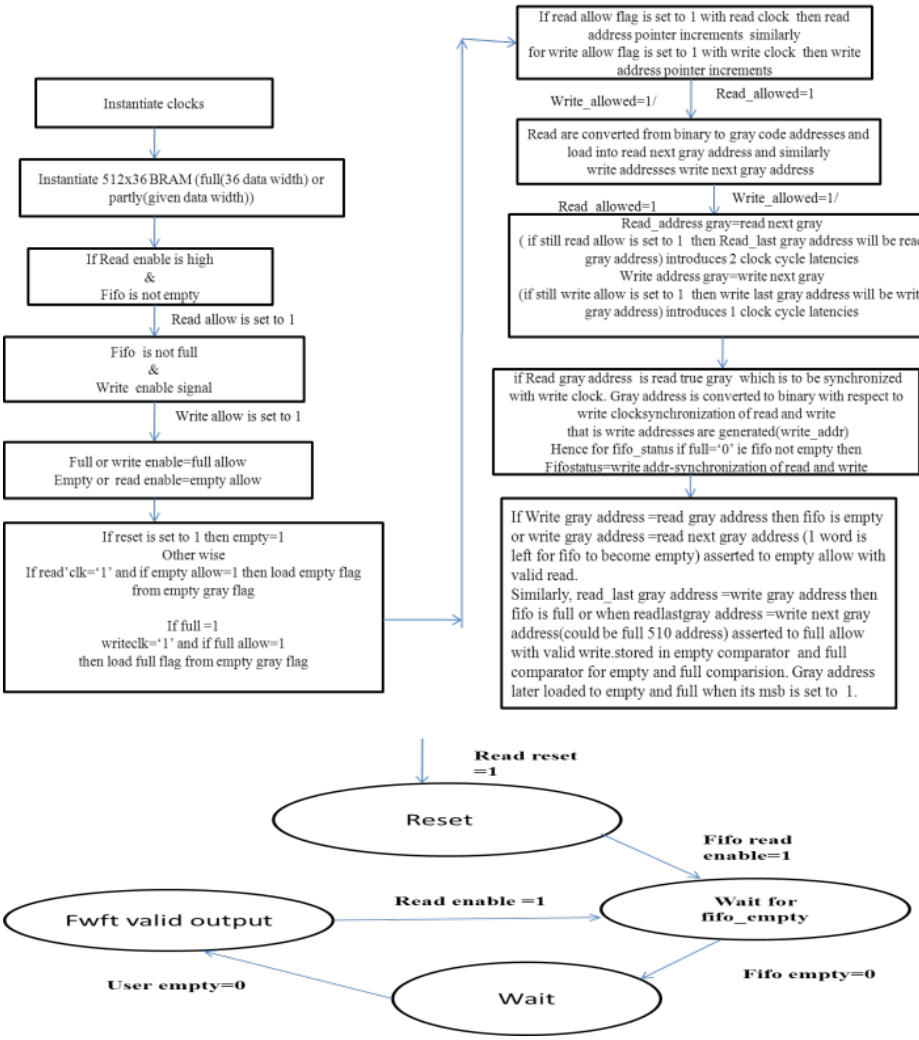

Figure 1. FWFT algorithm.

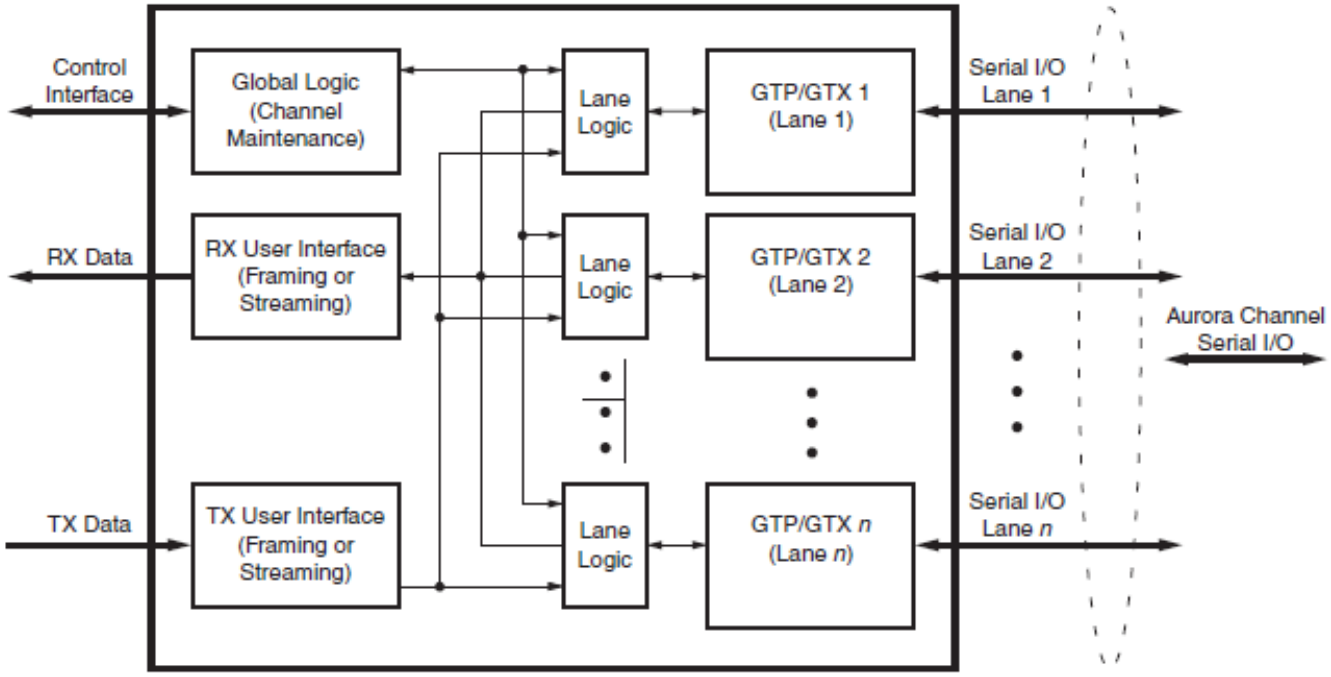

Figure 2. Aurora 8B/10B core block diagram.

The source code is produced by the CORE Generator software for Aurora 8B/10B cores with variable data path width. The cores can be simplex or full-duplex.

3.2 Functional Blocks

Figure 2 shows a block diagram of the Aurora 8B/10B core. The major functional modules of the Aurora 8B/10B core are:

- **Lane logic**: Each GTP/GTX transceiver is driven by an instance of the lane logic module, which initializes each individual GTP/GTX transceiver and handles the encoding and decoding of control characters and error detection.
- **Global logic**: The global logic module in performs the bonding and verification phases of channel initialization.
- **RX user interface**: This interface moves data from the channel to the application. Frames are presented using a standard Local Link interface.
- **TX user interface:** This interface moves data from the application to the channel. A standard Local Link interface is used for the data frames. This module has an interface for controlling clock Compensation

The Aurora 8B/10B protocol uses a symbol-based method. Two symbols of information is transferred across an Aurora 8B/10B channel therefore it is called a symbol-pair. The information on an Aurora 8B/10B channel (or lane) always consists of multiple symbol-pairs. Implementations of the Aurora 8B/10B protocol takes a stream of octets from user applications and transfer them across the Aurora 8B/10B channel as one or more streams of symbol-pairs. Transmission of user PDUs requires the following procedures:

- Padding
- Encapsulation with channel PDU delimiters
- 8B/10B encoding of channel PDU payload
- Serialization and clock encoding.

Reception of user PDUs involves the following procedures:

- • Deserialization
- • 8B/10B decoding of channel PDU payload
- • Link layer stripping
- • Pad stripping.

The Aurora 8B/10B core is a lightweight, serial communications protocol for multi-gigabit links. It is used to

transfer data between devices using one or many GTP/GTX transceivers. Connections can be full-duplex (data in both directions) or simplex. Aurora 8B/10B cores automatically initialize a channel when they are connected to an Aurora channel partner. After initialization, applications can pass data freely across the channel as frames or streams of data. Whenever data is not being transmitted, idles are transmitted to keep the link alive.

Aurora frames can be of any size, and they can be interrupted at any time. Gaps between valid data bytes are automatically filled with idle to maintain lock and to prevent excessive electromagnetic interference. The Aurora 8B/10B core detects both single-bit and multi bit errors using 8B/10B coding rules. Excessive disconnections, bit errors or equipment failures causes the core to reset and it attempts to re-initialize a new channel.

The 7 series FPGA from Xilinx introduces high-speed serial communication and some techniques are used to increase performance on transmission channels.

A multi gigabit transceiver, or MGT for short, is the heart of a high-speed serial I/O interface. Simplified, its work is to take a word in parallel on each clock cycle at some frequency A. Then serialize the word at frequency B= length(word)_A and transmit this serial stream data over a channel. On the receiver end, the serial stream data is de-serialized at frequency B. It is then sent to the receiving application with a word in parallel, at frequency A. The circuitry that does the serializing and de-serializing is commonly called a SerDes.

The 7-series FPGA from Xilinx contains transceivers which can cope with speeds up to 28 Gbps using a GTZ transceiver. The 7-series could also be equipped with a GTP or a GTH transceiver, with 6.6 and 13.1 Gbps respectively. A transceiver in the 7-series is located in a GTX Quad. Collection of four GTX transceivers is called a quad and it is placed near each other on the silica, sharing resources. Each quad contains four CPLL and one QPLL. These are Xilinx names for the PLLs that synthe-

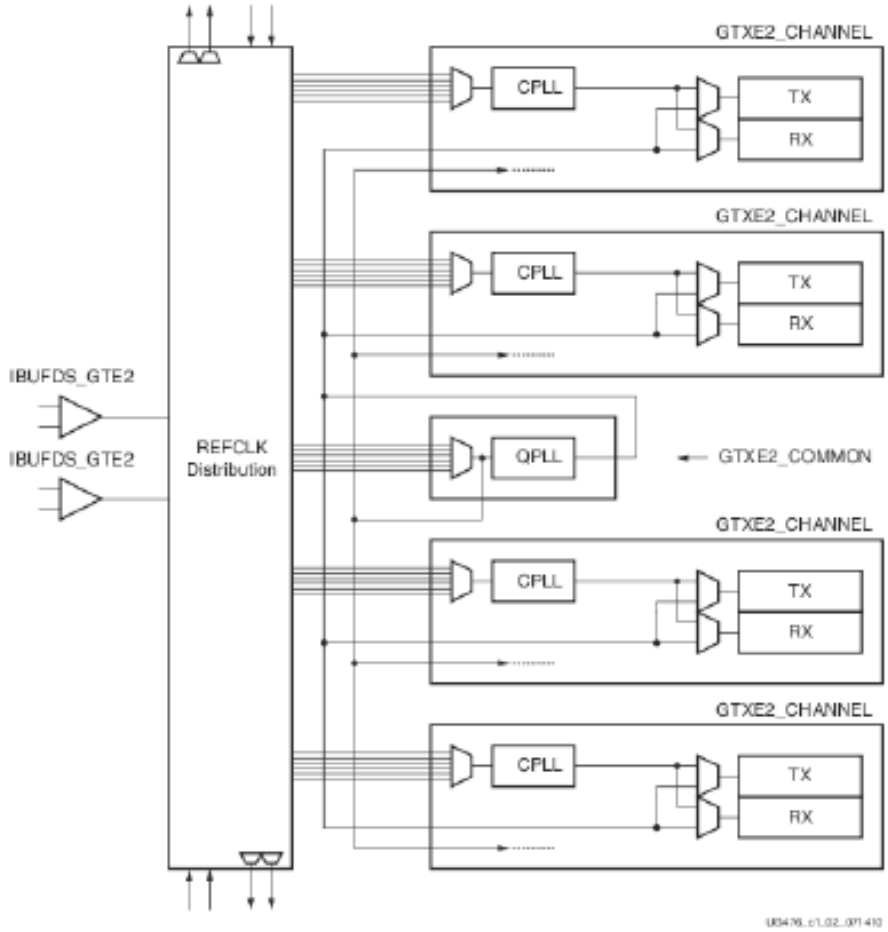

Figure 3. Xilinx 7-series GTX transceiver quad.

size the reference clocks for the transceivers. Each CPLL can synthesize different clock frequencies and it allows the four transceivers to run at different speeds which are independent to each other. The QPLL can operate between 5.93 and 12.5 GHz whereas the CPLL can operate at frequencies between 1.6 and 3.3 GHz. Each quad has two reference clock inputs.

3.3 Applications

Aurora 8B/10B cores can be used in a wide variety of applications because of their low resource cost, scalable throughput, and flexible data interface. Examples of Aurora 8B/10B core applications include:

- **Chip-to-chip links**
- **Board-to-board and backplane links**:
- **Simplex connections (unidirectional)**:
- **ASIC applications**

4. Implementation of the Proposed work

The implementation of above application involves the following stages. They are,

- Implementation of Simulator with AFIFO.
- Implementation of Quad Independent Aurora Links using Multi-Gigabit transceivers.

The resources used for design and implementation of the high speed data transmission are as follows:

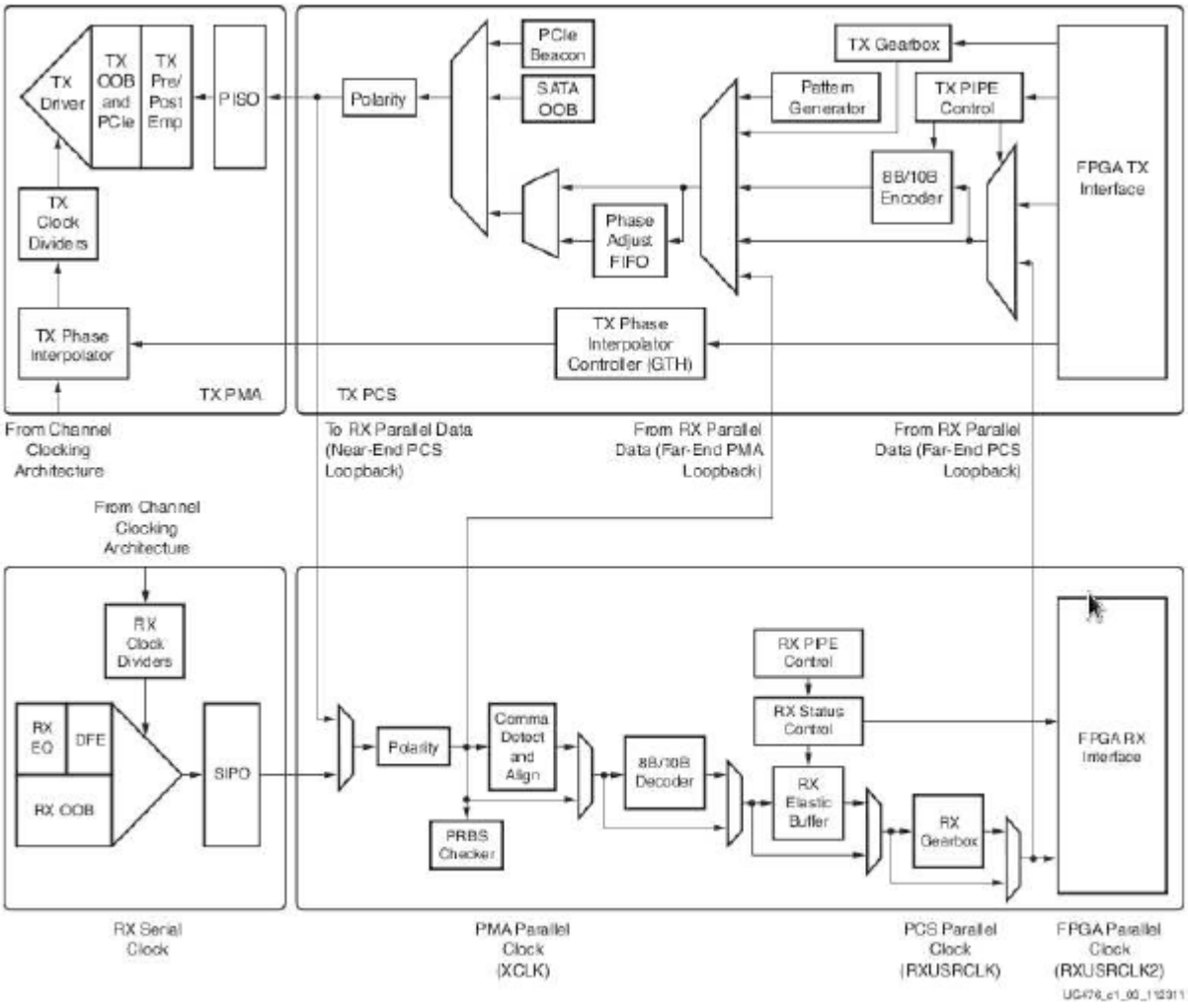

Figure 4. Xilinx 7-series GTX transceiver.

A. Software Resources:
   1. Xilinx Vivado
   2. Xilinx core generator tool

B. Hardware Resources:
   1. Virtex-7 Rocket IO development board.
   2. Fibre optic cable
   3. Switch Mode Power Supply(SMPS)
   4. JTAG cable

The Xilinx FPGA device (XCV7FX585T-1FFG1761) used is XILINX CMOS VIRTEX-7 FX585T series with package ffG1761 and speed grade -1. Here Virtex-7 FXT is a High-performance embedded system with advanced serial connectivity. It contains GTX transceivers capable of running up to 6.5 GB/s. Each GTX transceiver supports full duplex, clock and data recovery. And also contains Embedded IBM Power Pc 440 RISC CPUs.

5. Results

The following Figures 4–6 are the results obtained during the transmission of 192-bit parallel data and reception.

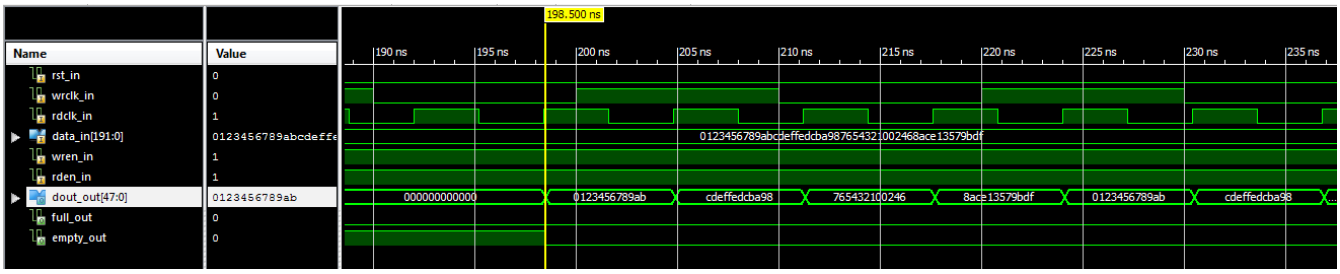

Figure 5. AFIFO.

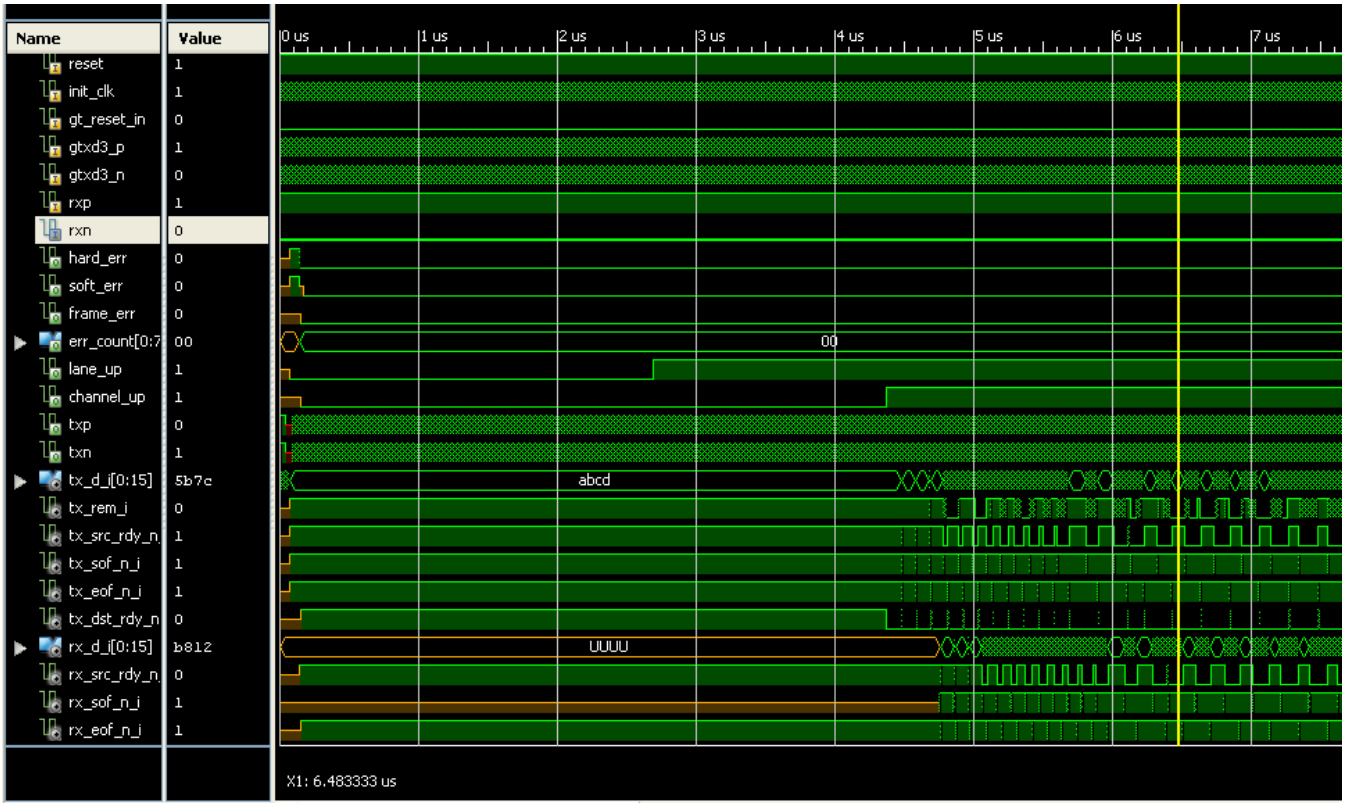

Figure 6. Aurora core.

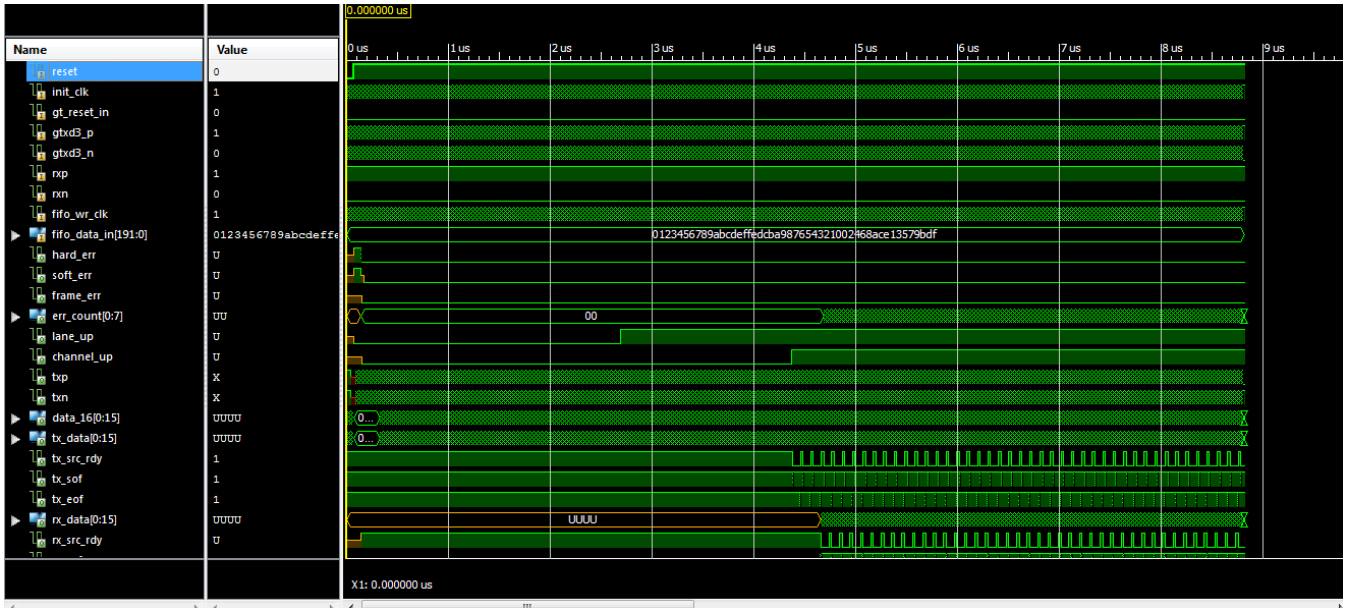

Figure 6. FIFO along with Aurora protocol.

6. Conclusion

The 192-bit parallel data is transmitted serially at the rate of 3.125Gbps over dual independent aurora links present on one GTX DUAL TILE through fiber optic cable using multi gigabit transceivers and they are received by the multi-gigabit transceivers of the same TILE using optical fiber cable. Finally, both the transmitted data and received data are verified by using logic analyzer software.

7. Future Scope

The present paper was implemented over dual independent aurora links on one GTX DUAL TILE for serial data transmission at the rate of 3.125Gbps using aurora protocol. We can also achieve 6.25Gbps speed with some other vertex series boards which supports 6.25Gbps line rate using aurora protocol. The other protocol is Serial Rapid IO (SRIO) which works on the principle of data packets switching which is more efficient for error free transmission and speed can be increased to higher level.

8. Acknowledgement

We wish to acknowledge with thanks, the support given by SRM University, in carrying out this work.

9. References

1. Reddy TVB. A real time implementation of high speed data transmission using aurora protocol on multi-gigabit transceivers in Virtex-5 FPGA. International Journal of Research in Computer and Communication Technology. 2012 Aug; 1(3):106–11.
2. Clive MAX Maxfield. The design warriors' guide to FPGAs; 2004.
3. Athavale A, Christensen C. High speed serial I/O made simple- A designers' guide with FPGA applications; 2005.
4. Xilinx, Aurora 8b/10b protocol specification, available at "http://www.xilinx.com/support/documentation/ip_documentation/aurora_8b10b_protocol_spec_sp002.pdf", SP002 (v2.2) April 19, 2010.
5. Xilinx, LogiCORE ip aurora 8b/10 v6.2 user guide [Internet]. 2010 Jul 23. Available from: available at http://www.xilinx.com/support/documentation/ip_documentation/aurora_8b10b_ug353.pdf.
6. Kishore V. Design and implementation of high speed data transmission over dual independent aurora channels on one GTX dual tile usingVirtex-5. 2013 International Journal of Scientific & Engineering Research; 2013.