International Conference on Emerging Trends in Engineering, Science and Management
Sphoorthy Engineering College, Hyderabad, India (ESM-17)
17th and 18th March 2017 , www.conferenceworld.in ISBN: 978-93-86171-32-0

DESIGN AND IMPLEMENTATION OF AFIFO USING BRAM AND HIGH SPEED DATA TRANSMISSION USING AURORA ON VIRTEX-7 FPGA

Juveria Fatima[1], K.Anjani Sraddha[2]

[1]*Digital systems (M.E), ECE Department, Muffakham JAH College of Engineering Hyderabad,Telangana,, (India)*

[2]*VLSI (M.Tech), ECE Department, Gitam University, Vizag, Andhra Pradesh,, (India)*

ABSTRACT

This paper proposes a design and implementation of AFIFO using BRAM and high speed data transmission over Quad independent aurora channels on One GTX(Gigabit Transceivers)TILE by configuring multi-gigabit transceivers(MGT's), which are present in the virtex-7FPGA using aurora protocol. (Here GT means Gigabit transceivers (GT) and X indicates that these transceivers belong to Virtex-7 FXT platform.FXT platform supports High-performance embedded systems with advanced serial connectivity). Firstly, a 192 bit parallel data is to be generated using simulators, which are implemented using VHDL language. The asynchronous first-in first-out (AFIFO) takes this 192 bit data as input and produces an output of 32 bit parallel data. This data goes to the aurora module in parallel form as successive frames (i.e. 6frames, 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. To achieve high speed, Multi-Gigabit Transceivers (MGT)are used. In virtex-7 FPGA, these Multi-Gigabit Transceivers are available as hard IP switch operates at the clock rate of 156.25 MHz (MGT clock).For configuring these MGT's, Aurora protocol is used, which converts the parallel data into serial data and vice versa. Further testing is carried out

Keywords: Aurora protocol, Virtex-7 FPGA, AFIFO, BRAM, Serial Data Transmission.

I. INTRODUCTION

The transfer of data over a point-to-point or point-to-multi point communication channel. Examples of such channels are copper wires, optical fibers, wireless communication channels, storage media and computer buses. Earlier days to achieve the high speed we were using parallel/serial transmission. In parallel transmission, Binary data consisting of 1s and 0s which are transmitted in framing/streaming. By grouping, we can send data *n* bits at a time instead of one. We use *n* wires to send *n* bits at one time. That way each bit has its own wire, and all *n* bits of one group can be transmitted with each clock pulse from one device to another. For*m* = 8. Typically, the n number of bits are bundled in a frame with a connector at each end that may be 2:4:8:16:32. The advantage of parallel transmission is speed. All else being equal, parallel transmission can increase the

transfer speed by a factor of *n* over serial transmission. Insignificant disadvantage of parallel transmission is cost.

In serial transmission one bit follows another, so we need only one communication channel rather than *n* to transmit data between two communicating devices. The advantage of serial over parallel transmission is that with only one communication channel, serial transmission reduces the cost of transmission over parallel by roughly a factor of *n* Since communication within devices is parallel, 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 one of two ways; asynchronous or synchronous.

MGTs are used increasingly for data communications because they can run over longer distances. The main function of these MGT's are to convert the parallel data into serial data and vice versa and this action is performed by configuring the MGT's using aurora core. These MGT's hard IPs core and we have to interface to our application by configuring hard IPs using soft IPs such as aurora core in order to achieve the high speed serial data transmission. The MGT's are configured as either transmitter or receiver to perform the data transmission.

The Rocket IO GTX transceiver is a power-efficient transceiver for Virtex-7 FPGAs. The GTX transceiver is highly configurable and tightly integrated with the programmable logic resources of the FPGA. GTX transceivers are placed as transceiver GTX tiles in Virtex-7 FXT devices. This configuration allows one or more transceivers to share a single PLL with the TX and RX functions of both, reducing size and power consumption. Multi Gigabit Transceiver(MGT) is a Serialiser/Deserialiser (SerDes) capable of operating at serial bit rates above 1 Gigabit/second.

II. AFIFO USING BRAM

A: INTRODUCTION:

FIFO, or First In, First Out, is a method that relates to the organization and manipulation of data according to time and prioritization. In essence, the queue processing technique is done as per a first-come, first-served behavior. The algorithm of the operating system scheduling gives every process CPU time according to the order it comes. Each item is stored in a queue data structure. The first data which is added to the queue will be the first data to be removed. Processing continues to proceed sequentially in this same order. FIFO is used for synchronization purposes in computer and CPU hardware. FIFO is generally implemented as a circular queue, and thus has a read pointer and a write pointer. A synchronous FIFO uses the same clock for reading and writing. An asynchronous FIFO, however, uses separate clocks for reading and writing. AFIFO is generated using BRAM and customized according to the application.

B. Asynchronous FIFO

An asynchronous FIFO refers to 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.

In order to understand FIFO design, one needs to understand how the FIFO pointers work. The write pointer always points to the next word to be written; therefore, on reset, both pointers are set to zero, which also

happens to be the next FIFO word location to be written. On a FIFO-write operation, the memory location that is pointed to by the write pointer is written, and then the write pointer is incremented to point to the next location to be written.

C.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. If the FIFO is continuously written to without being read, it fills with data. Write operations are only successful when the FIFO is not full. When the FIFO is full and a write is initiated, the request is ignored, the overflow flag is asserted and there is 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. If the FIFO is continuously read without being written, the FIFO empties. Read operations are successful when the FIFO is not empty. When the FIFO is empty and a read is requested, the read operation is ignored, the underflow flag is asserted and there is no change in the state of the FIFO (under flowing the FIFO is non-destructive).

- **Modes of Read Operation:**

The FIFO Generator supports two modes of read options, standard read operation and first-word fall-through (FWFT) read operation. The standard read operation provides the user data on the cycle after it was requested. The FWFT read operation provides the user data on the same cycle in which it is requested.

- **First-Word Fall-Through FIFO Read Operation:**

The first-word fall-through (FWFT) feature provides the ability to look-ahead to the next word available from the FIFO without issuing a read operation. When data is available in the FIFO, the first word falls through the FIFO and appears automatically on the output bus (DOUT). Once the first word appears on DOUT, EMPTY is de-asserted indicating one or more readable words in the FIFO, and VALID is asserted, indicating a valid word is presentation DOUT.

Note: For every write operation, an equal number of read operations is required to empty the FIFO – this is true for both the first-word-fall-through and standard FIFO.

- **Non-symmetric Aspect Ratios:**

This feature is supported for FIFOs configured with independent clocks implemented with block RAM. Non-symmetric aspect ratios allow the input and output depths of the FIFO to be different. The following write-to-read aspect ratios are supported: 1:8, 1:4, 1:2, 1:1, 2:1,4:1, 8:1.

For non-symmetric aspect ratios, the full and empty flags are active only when one complete word can be written or read. The FIFO does not allow partial words to be-accessed. For example, assuming a full FIFO, if the write width is n bits and read width is highly low, the user would have to complete four valid read operations before full de-asserts and a write operation is accepted. Write data count shows the number of FIFO words according to the write port ratio, and read data count shows the number of FIFO words according to the read port ratio. Table identifies support for non-symmetric aspect ratios.

ISBN: 978-93-86171-32-0

Impl Implementation of AFIFO with FWFT:

The 512X36 Bram is used since the output of FIFO should be given as 32 bit word to AURORA protocol. The input to the FIFO is 192 bit PD word data so FIFO has to write these 192 bits and should be 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 should be always faster than write clock. 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 write 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. So read last grey code address is write next grey code address. Then FIFO status will be subtraction of grey code addresses of read and write. Reconvert binary address from grey code address for synchronizing read address pointer with write address pointer with one delay.

FIFO Implementation		**Non Symmetric Aspect Ratios Support**
Independent Clocks	Block RAM	Yes
	Distributed RAM	No
	Built-In	No
Common Clocks	Block RAM	No
	Distributed RAM	No
	Shift Register	No
	Built-In	No

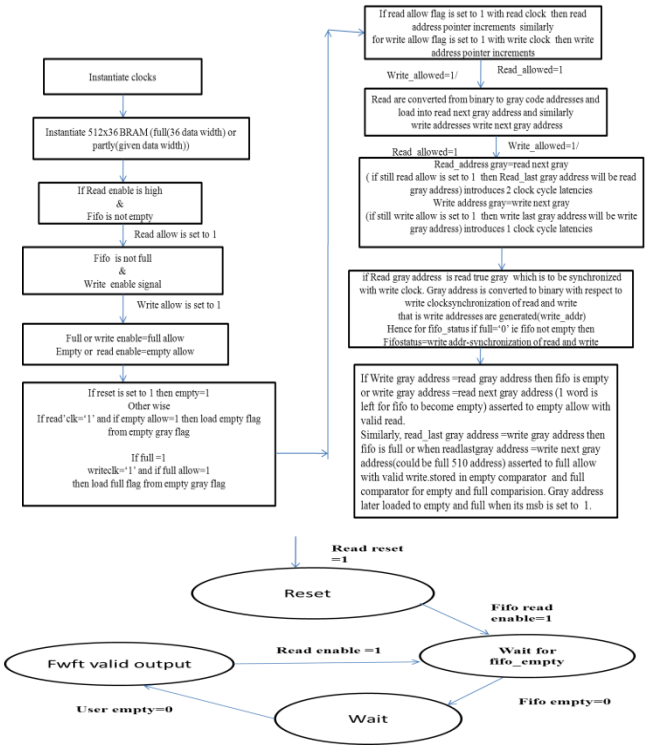


Figure: FWFT algorithm.

From the figure 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.

IMPLEMENTATION

This design is implemented in two steps

Initially the FIFO along with the glue logic is implemented so that the input of the FIFO is 192 bit and the output of the FIFO 32 bit which will be given as the input to the aurora.

The second step again involves two steps i.e., customizing the aurora core and then combining the aurora core with FIFO and glue logic.

III. AURORA PROTOCOL

A: Introduction:

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

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

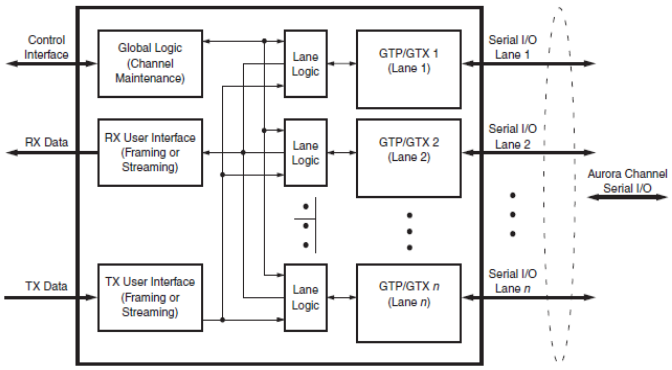


Fig.2: Aurora 8B/10B Core Block Diagram

B: Functional Blocks:

Fig 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 each Aurora 8B/10B core performs the bonding and verification phases of channel initialization. While the channel is operating, the module generates the random idle characters required by the Aurora protocol and monitors all the lane logic modules for errors.

• **RX user interface**: The RX user interface moves data from the channel to the application. Frames are presented using a standard Local Link interface.

• **TX user interface:** The TX user interface moves data from the application to the channel. A standard Local Link interface is used for data frames. The module has an interface for controlling clock Compensation (the periodic transmission of special characters to prevent errors due to small clock frequency Differences between connected Aurora 8B/10B cores).

The Aurora 8B/10B protocol uses a symbol-based method. The minimum unit of information that is transferred across an Aurora 8B/10B channel is two symbols, called a symbol-pair. The information on an Aurora 8B/10B channel (or lane)always comprises multiple symbol-pairs. Implementations of the Aurora 8B/10B protocol accept 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:

• De-serialization

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

The 7 series FPGA from Xilinx introduces high-speed serial communication and some techniques used to increase performance on non-ideal transmission channels. A multi gigabit transceiver, or MGT for short, is the heart of a high-speed serial I/O interface. Simplified, its mission 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 over a channel. On the receiver end, the serial stream is de-serialized at frequency B. It is then presented 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.

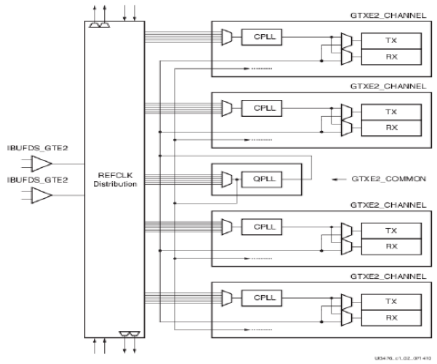


Figure :Xilinx 7-series GTX transceiver .

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 GTH or a GTP transceiver, with 13.1 and 6.6 Gbps respectively. A transceiver in the 7-series is located in a so-called GTX Quad. A quad is a collection of four GTX transceivers placed near each other on the silica, sharing resources. Each quad contains four so called CPLL and one QPLL. These are Xilinx names for the PLLs that synthesize the reference clocks for the transceivers. Each CPLL can synthesize different clock frequencies and this allows the four transceivers to run at different speeds independent of each other. The CPLL can operate at frequencies between 1.6 and 3.3 GHz, whereas the QPLL can operate between 5.93 and 12.5 GHz. Each quad has two reference clock inputs.

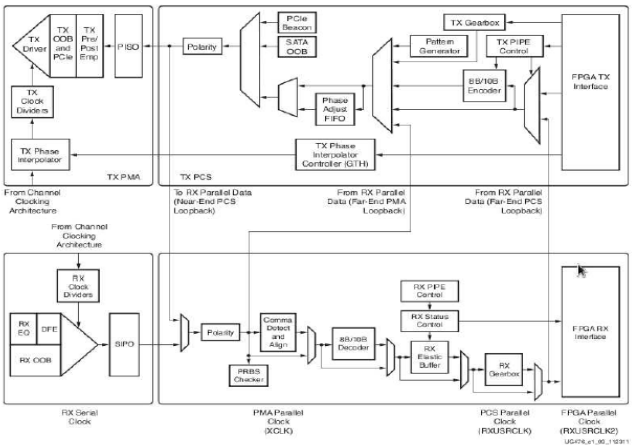


Figure : Xilinx 7-series GTX transceiver.

C: 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**: Replacing parallel connections between chips with high-speed serial connections can significantly reduce the number of traces and layers required on a PCB. The core provides the logic needed to use GTP/GTX transceivers, with minimal FPGA resource cost.

• **Board-to-board and backplane links**: The Aurora 8B/10B core uses standard 8B/10B encoding, making itcompatible 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.

• **Simplex connections (unidirectional)**: 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 full duplex resources.

• **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.

IV. DESIGN OF APPLICATION

The following figure explains the design and its procedure to implement the high speed data transmission serially over dual independent aurora channels

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

A. Software resources:

1. Xilinx ISE design suite 14.7

B. Hardware Resources:

ISBN: 978-93-86171-32-0

1. Virtex-7 Rocket IO development board.
2. SFP transceivers
3. Fibre optic cable
4. Switch Mode Power Supply(SMPS)
5. 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 440RISC CPUs.

V. IMPLEMENTATION OF THE PROJECT

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

1) Implementation of Simulator with AFIFO.
2) Implementation of Independent Aurora Links using Multi-Gigabit transceivers.

1) Implementation of Simulator with AFIFO :

Simulator logic is designed in such a way that it produces an output of 192 bit parallel data and developed using VHDL language. We have different types of FIFO's are available, among all the FIFO's block RAM FIFO is used in our application because it is having all the features such as FWFT and non-asymmetric ratio which are required in our project. These FIFO's are generated using Xilinx core generator tool. While generating the FIFO, user should select the write width as 192 bit, read width as 32 bit, native and the write depth is4096.for writing and reading of data into FIFO and from FIFO, we use independent clocks.

Designing a FIFO memory subsystem seems simple and straight forward because it consists of a Random Access Memory (RAM) with two independently clocked ports. One port is used for writing. The other port is used for reading. In addition, the FIFO has two independent address counters to steer write and read data. Independently clocked means that there are no restrictions on the relation ship between read and write clock frequency and phase. Independently clocked is also referred to as multi-rate or asynchronous FIFO operation.

To meet the challenge of fast, asynchronous FIFO design, the Virtex-7 family includes a dedicated, hard-coded FIFO controller inside each block RAM. The designer supplies 4-bit, 9-bit, 18-bit, or 36-bit parallel input data, a continuously running write clock, a write clock enable signal, a continuously running read clock, and a read clock enable signal.

The FIFO Empty signal goes high as a result of the read clock when the last data entry is read out of the FIFO. The designer must disable reads until the FIFO Empty signal goes Low again. The rising and the falling edges of the FIFO Empty signal are synchronous with the read clock, providing a totally synchronous interface. If the read clock enable signal is accidentally kept active after the FIFO is empty, a read error flag is asserted, but the FIFO contents and addressing are not affected. FIFO Almost Empty and FIFO Almost Full are programmable status signals. These two signals can be used as a warning to slow the read or the write operation, or they can be

used to indicate the data level in the FIFO. Non-asymmetric ratio of inputs and outputs is required in our application, the input to the FIFO is 192 bit parallel data and output is 32 bit parallel data.

2) Implementation of dual independent aurora links using multi-gigabit transceivers:

Multi-gigabit transceivers are present in FPGA and we use two different MGT's which are belongs to one GTX dual tile. These MGT's are configured using aurora protocol for developing the application of dual independent aurora channels for high speed serial data transmission at the rate of 3.125 GB/s.

A. Customizing Aurora core:

In 7 series (XCV7FX585T-1FFG1761) we have mainly two different types of aurora versions are available, they are aurora 8B10B 5.2 and aurora8B10B 6.1.among these two types, select appropriate version depending upon application being used in your project. In order to use AXI interface is required for implementing the project I was selected aurora 8B10B 6.1.

To generate any aurora core, it involves mainly two steps. In first step user should select some core parameters for ex; number of aurora lanes, lane width, line rate and GT REFCLK etc. Here in my application, I was selected the number of lanes as ONE, lane width as 4. In each transceiver the user can select from 1 to 16 lanes. Lane width indicates that how many number of bytes that are taken as input to the aurora module. Available options are either TWO bytes or FOUR bytes. If we were selected the lane width as TWO then USER_CLK must equal to the REF_CLK in our design or if it is FOUR the USER_CLK becomes half of the REF_CLK. Since, four bytes are selected so that the input to the aurora module has 32bits.

Here, I was selected the line rate as 3.125Gbps means that in each lane the data was transmitted at the rate of 3.125 Giga bits per second. Using 7 series boards we can achieve high speeds up to 3.125Gbps only. After selecting the core parameters, user select core features namely dataflow mode, interface and flow control. I was selecting DUPLEX, FRAMING and NONE respectively. I was selected in such a way that each has a full duplex communication and data has transmitted in each lane in the form of frames. Here in this we have two types of interfaces namely FRAMING and STREAMING. In framing there must be include or assign a SOF and EOF signals to each frame, whenever the destination is ready then only data is transmitted through the channel.

7 series FPGA () supports a maximum of 16 GTX transceivers. These 36 transceivers are placed on FPGA in the name of GTX_QUAD_X0Y0 to GTX_QUAD_X0Y35 as two columns. Each GTX_QUAD_TILE consists of two transceivers so they have total of 36 GTX transceivers are present. In order to generate the QUAD independent aurora links in one GTX transceivers follow the few important steps.

First generate the aurora core from the core generator tool by selecting the appropriate transceiver in the GTX QUAD TILE. The selection of transceiver is based on the application being used in your project. I was successful with ISE114.7, IP core Aurora 8B10B v8.3 and followed user guides of aurora core. It's not too complicated, but it took a few steps including renaming/mapping/remapping signals.

a. In step1, Select the aurora lane=1, lane width=4 bytes, line rate 3.125Gbps, dataflow mode=duplex, interface=framing, flow control= none.
b. In step2, select any one of the Tile for ex, GTX_QUAD _X0Y33. Next click on generate, the aurora source code for X0Y33was generated.

Then do the following modifications in each module of aurora, these are required for generation of dual independent aurora channels.

- <Design name>_tile.vhd: no changes required.
- <Design name>_tranceiver_wrapper.vhd: I changed each and every port of this module. Simply duplicate each port by renaming signal name as _0 and _1.

  For example, LOOPBACK_IN becomes two ports: LOOPBACK_IN_0 and LOOPBACK_IN_1.

  Note: the only ports that are not duplicated in this way are: ENCHANSYNC_IN, CHBONDDONE_OUT, CHBONDDONE_OUT_unused, REFCLK, and GTPRESET_IN and its related signals.
- <Design name>.vhd: In this module, again all the ports need to be duplicated for the 0 and 1 transceiver.

  For example, DO_CC becomes DO_CC_0 and DO_CC_1 .the following ports do not need to be duplicated.

  GTPD2, RESET, GT_RESET.

  Similarly, you will need to instantiations of each of the following
- <design name>_AURORA_LANE_4BYTE
- <design name>_GLOBAL _LOGIC
- <design name>_AXI_TO_LL
- <design name>_TX_STREAM
- <design name>_LL_TO_AXI
- <design name>_RX_STREAM

Suppose if you are using framing interface then streaming modules are replaced by framing related modules.

Only one instantiation of

- <Design name>_GTP_WRAPPER.

Look at the signal declarations carefully to see what should be duplicated for the second lane.

- <design name>_example_design.vhd:the changes are similar to those in <Design name>.vhd

In the process of duplicating each module, you need to duplicate the signal declarations also, which are present in each module.

Finally debug your design, if any errors were found during generation bit file then try to solve it. you are facing simple errors only.

B. SFP Transceiver:

Small Form factor Pluggable transceiver (SFP) [5] is a compact, hot-pluggable transceiver used for converting the electrical signal into light signal and vice versa and these light signals are transmitted through fiber optic cable.

C. JTAG Cable:

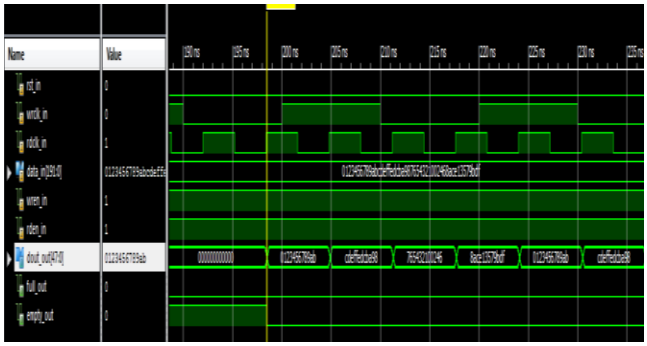
The full form of JTAG is Joint Test Action Group because it is developed by Joint Test Action Group and Sanctioned by IEEE as STD 1149.1 test access port and Boundary Scan Architecture in 1990. JTAG cable is used to downloading the BIT file or programming file from PC into the target hardware board.ie, virtex5 FPGA board.

VI. RESULTS

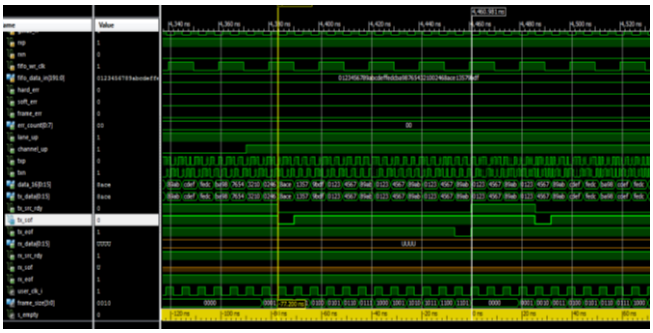
The following are the results obtained from afifo using BRAM memory generator for transmission of 192 bit parallel data, aurora core and integration of AFIFO and aurora. results for both transmission and reception are shown in final integrated project in chip-scope analyzer.

In the figure 4 the signals are channel up, lane up, Tx_data and Rx_data are shown. For the effective data transmission channel up should be one. To keep the channel link continuously active, idle sequences are sent when actual data is not present.TX_TDATA0, TX_TDATA1 are the two signals in which the data is present and transmitted over two dual independent channels. Here data sent over the Tx_data is 00B9E2391640801E0024803203E807D0, the same data is received on the Rx_data signal line (rx_data_i_0, rx_data_i_1)after some delay that is 32 clock cycles of latency which is in the permissible limit(Maximum latency for a 2-byte designs from TX_SOF_N to RX_SOF_N is approximately 53GTP/GTX transceivers clock cycles in simulation.).

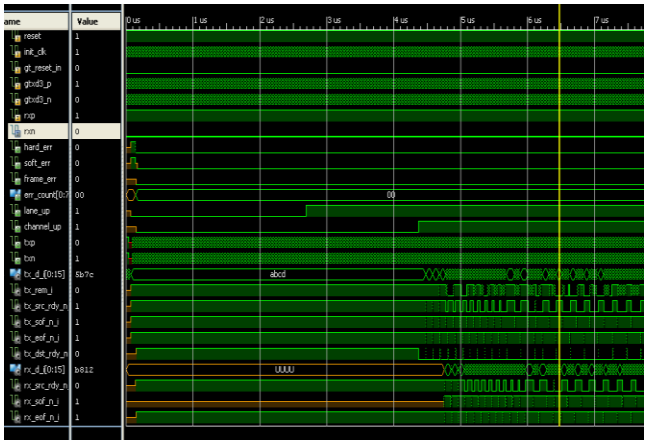
AFIFO USING BRAM:

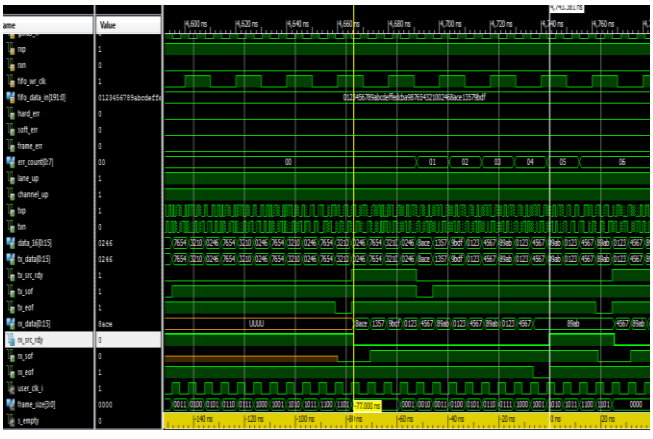


AFIFO integrated with AURORA



AURORA CORE





VII. CONCLUSION

The 192 bit parallel data's are 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 received by the multi-gigabit transceivers of the same TILE using optical fiber loopback. Finally both the transmitted data and received data's are verified by using chip-scope analyzer software.

VIII. FUTURE SCOPE

The present paper was implemented dual independent aurora links on one GTX DUAL TILE for serial data transmission at the rate of 3.125Gbps using aurora protocol. We can achieve the 6.25Gbps speed also 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 is more efficient for error free transmission and speed can be increased to higher level.

ACKNOWLEDGMENT

The authors would like to thank D.KRISHNAVENI, SC.'F', DLRL, HYDERABAD, for her support in implementation of the project and R.RAMMOHAN, SC.'E for giving his valuable suggestions.

REFERENCES

[1] S Venkat kishore, "Design and Implementation of High Speed Data Transmission over Dual Independent Aurora Channels on One GTX dual tile usingVirtex-5 FPGA", International Journal of Scientific & Engineering Research, Volume 4, Issue 12, December-2013 Clive MAX Maxfield, "The design warriors' guide to FPGAs", 2004.

[2] Volnei A. Pedroni, "Circuit Design with VHDL", 2004.

[3] Abhijit Athavale and Carl Christensen of Xilinx "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 64B/66B v6.2, available at "http://www.xilinx.com/support/documentation/ip_documentation/aurora_64b66b/v6_2/ds815_aurora_64b66b.pdf", DS815 Jan, 2012.

[6] Xilinx, Chip Scope Pro Software and Cores User Guide, available at "http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/chipscope_pro_sw_cores_ug029.pdf", UG029 (v13.1), March 1, 2011.

[7] Xilinx, ISim User Guide user guide, available at "http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/plugin_ism.pdf" ,UG660 (v 13.1) March 18, 2011.

[8] Xilinx, LogiCOREip aurora 8b/10 v6.2 user guide, available at "http://www.xilinx.com/support/documentation/ip_documentation/aurora_8b10b_ug353.pdf",UG353 (v6.2) July 23, 2010.