PROJECT REPORT

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

DR. B.R.AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY

JALANDHAR

SUBMITTED BY:

SOURAV KUMAR

14104033

FPGA Based High Speed Serial Digital Interfaces in VHDL

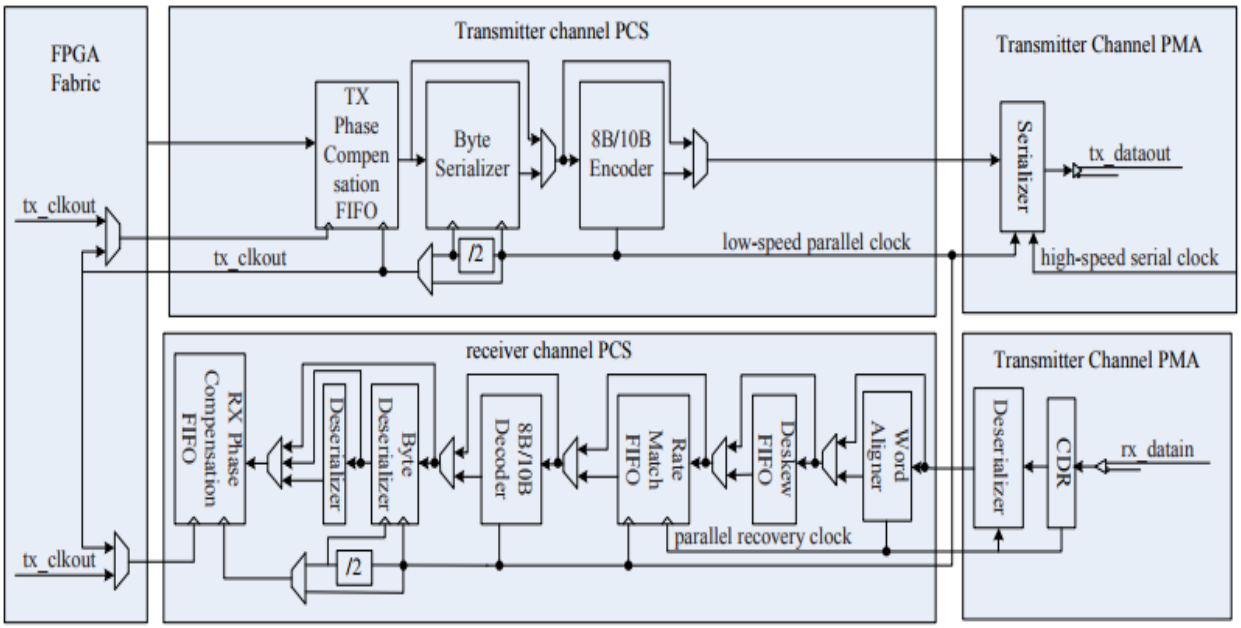


TABLE OF CONTENTS

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts success.

I am extremely grateful to respected **Dr K Maheswara Reddy Director, Defence Avionics Research Establishment (DARE)** for having agreed to allow me to train at DARE. I also would like to express my sincere gratitude to respected **Shri. R Charles Darwin (Sc-E)**, for his encouragement, overall guidance and effort in bringing out this project.

I would like to convey my heart-felt thanks to my Project guide **Mr. Pramod Kumar (Sc-C),** for his co-operation and providing me with necessary advice and support during the entire duration of the Internship. I am extremely grateful to him for his valuable suggestions and unflinching co-operation throughout project work.

Last but not the least I express my heart-felt thanks to Entire DARE Team for providing me with the necessary technical and Administrative support in completing the project.

SOURAV KUMAR(14104033)

ABSTRACT

By using the dynamic reconfigurable transceiver in high speed interface design, designer can solve critical technology problems such as ensuring signal integrity conveniently, with lower error binary rate. The following points are focused:

IP (Intellectual Property) core usage: Altera Co. offers two transceiver IP cores in Quartus II MegaWizard Plug-In Manager for XAUI design which is featured of dynamic reconfiguration performance, that is, ALTGX_RECOFIG instance and ALTGX instance.

RTL (Register Transfer Level) coding with VHDL and simulation: Create the ALTGX_RECOFIG instance and ALTGX instance, enable dynamic reconfiguration in the ALTGXB Megafunction, then connect the ALTGX_RECOFIG with the ALTGX instances. After RTL coding, the design was simulated on Quartus II v14.0 and its synthesis was done in Atlera Stratix IV GX FPGA Development kit. The validated result indicates that the packets are transferred efficiently. FPGA makes high-speed optical communication system design simplified.

1. TECHNOLOGIES USED

In this project, i have Developed High Speed SERDES for Data Processing Unit of Su-30MKI fighter aircraft by using Field Programmable Gate Array (FPGA). The code of High Speed SERDES was written in VHDL using ALTGX IP Cores and its Syntheisis was done in Atlera Stratix IV GX FPGA Development kit using Quartus II v14.0.

FPGA have been used for a wide range of applications. After the introduction of the FPGA, the field of programmable logic has expanded exponentially. Due to its ease of design and maintenance, implementation of custom made chips has shifted.The integration of FPGA and all the small devices will be integrated by using Very Large Scale Integration (VLSI).

Thus, the major technologies used in this project are:

- STRATIX IV FPGA
- VHDL
- QUARTUS II v14.0
- ALTGX IP CORES

2. STRATIX IV FPGA - ALTERA

2. 1 INTRODUCTION

Intel FPGAs are ideal for a wide variety of applications, from high-volume applications to state-of-the-art products. Each series of FPGA includes different features, such as embedded memory, digital signal processing (DSP) blocks, high-speed transceivers, or high-speed I/O pins, to cover a broad range of end products.

The Stratix® FPGA and SoC family enables you to deliver high-performance, state-of-the-art products to market faster with lower risk and higher productivity.

By combining high density, high performance and a rich feature set, Stratix series FPGAs allow you to integrate more functions and maximize system bandwidth.

2.2 FEATURE SUMMARY

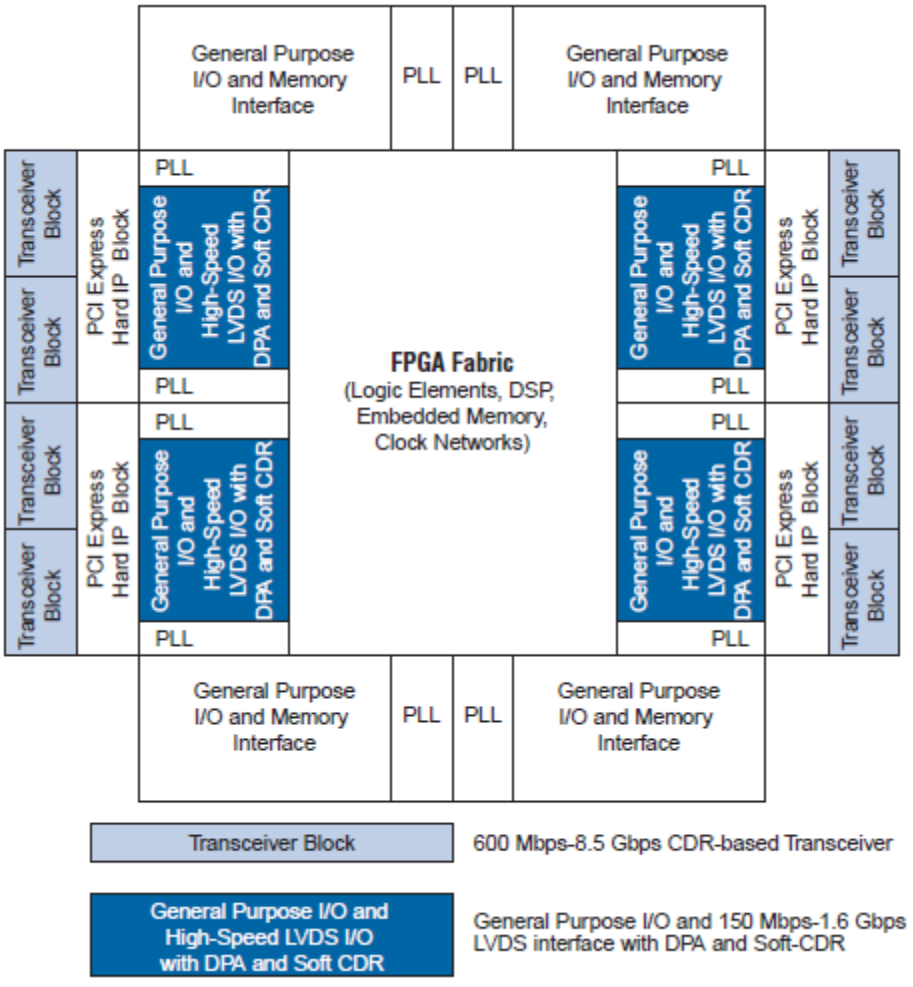
- The Stratix IV GX FPGA is fully PCI-SIG™ compliant for PCI Express® (PCIe®) Gen1 and Gen2 (x1, x4, and x8) and is on the PCI-SIG integrators list
- The Stratix IV GT FPGA is the only FPGA with integrated 11.3 Gbps transceivers
- Highest density with up to 820K logic elements (LEs), 23.1 Mb of embedded memory, and up to 1,288 18 x 18 multipliers
- Highest FPGA performance with a 2 speed grade advantage and the industry's most advanced logic and routing architecture
- Unprecedented system bandwidth with up to 48 high-speed transceivers at up to 8.5 Gbps, or up to 24 transceivers at up to 11.3 Gbps optimized for 100G applications and 1,067 Mbps (533 MHz) DDR3 memory interfaces
- Lowest power with up to 50 percent lower power than any other high-end FPGA in the market enabled by 40 nm benefits and Programmable Power Technology
- Hardened intellectual property (IP) for PCI Express Gen1 (2.5 Gbps) and Gen2 (5.0 Gbps) with up to four x8 blocks delivering a full endpoint or root-port function
- Superior signal integrity with the ability to drive a 50" backplane at 6.375 Gbps with Plug & Play Signal Integrity

- Up to 48 full-duplex CDR-based transceivers in Stratix IV GX and Stratix IV GT devices supporting data rates up to 8.5 Gbps and 10.3125 Gbps, respectively
- Typical physical medium attachment (PMA) power consumption of 100 mW at 3.125 Gbps and 135 mW at 6.375 Gbps per channel

2.3 Stratix IV GX Devices

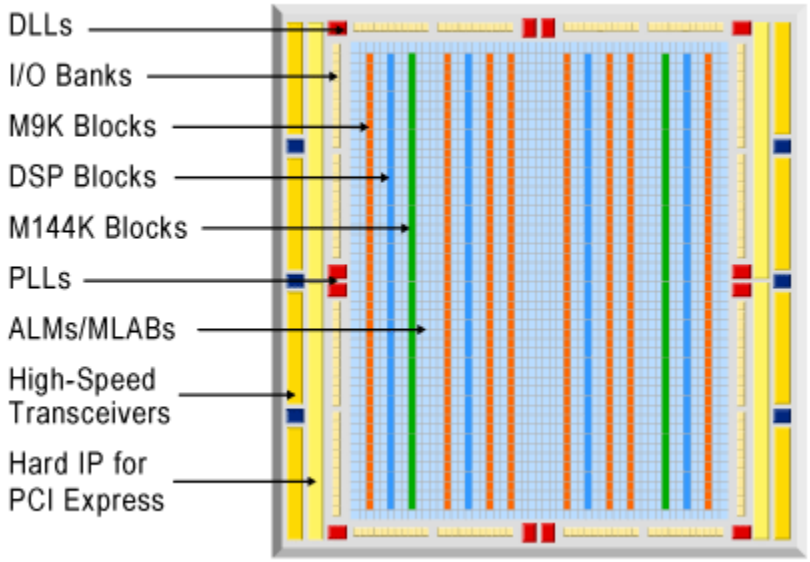
Stratix IV GX devices provide up to 48 CDR-based transceiver channels per device:

- Thirty-two out of the 48 transceiver channels have dedicated physical coding sublayer (PCS) and physical medium attachment (PMA) circuitry and support data rates between 600 Mbps and 8.5 Gbps
- The remaining 16 transceiver channels have dedicated PMA-Only circuitry and support data rates between 600 Mbps and 6.5 Gbps



STRATIX IV CHIP OVERVIEW

2.4 STRATIX IV ARCHITECTURE



High-Speed Transceiver Features

Stratix IV GX and Stratix IV GT high-speed transceiver features include:

Highest Aggregate Data Bandwidth

Up to 48 full-duplex transceiver channels supporting data rates up to 8.5 Gbps inStratix IV GX devices and up to 10.3125 Gbps in Stratix IV GT devices

Wide Range of Protocol Support

Physical layer support for the following serial protocols:

- Stratix IV GX: PCI Express (PIPE) Gen1 and Gen2, Gigabit Ethernet, Serial RapidIO, SONET/SDH, XAUI/HiGig, (OIF) CEI-6G, SD/HD/3G-SDI, Fibre Channel, SFI-5, GPON, SAS/SATA, HyperTransport 1.0 and 3.0, and Interlaken
- Extremely flexible and easy-to-configure transceiver data path to implement proprietary protocols
- PCI Express (PIPE) Support
- Complete PCI Express (PIPE) Gen1 and Gen2 protocol stack solution compliant to PCI Express Base Specification 2.0 that includes PHY-MAC, Data Link, and Transaction layer circuitry embedded in PCI Express hard IP blocks
- Root complex and end-point applications
- ×1, ×4, and ×8 lane configurations
- PIPE2.0-compliant interface
- Embedded circuitry to switch between Gen1 and Gen2 data rates
- Built-in circuitry for electrical idle generation and detection, receiver detect, power state transitions, lane reversal, and polarity inversion
- 8B/10B encoder and decoder, receiver synchronization state machine, and ± 300 parts per million (ppm) clock compensation circuitry

- Transaction layer support for up to two virtual channels (VCs)
- XAUI/HiGig Support
- Compliant to IEEEP802.3ae specification
- Embedded state machine circuitry to convert XGMII idle code groups (||I| to and from idle ordered sets (||A||, ||K||, ||R||) at the transmitter and receiver, respectively
- 8B/10B encoder and decoder, receiver synchronization state machine, lane deskew, and ± 100 ppm clock compensation circuitry
- Gigabit Ethernet Support
- Compliant to IEEE802.3-2005 specification
- Automatic idle ordered set (/I1/, /I2/) generation at the transmitter, depending on the current running disparity
- 8B/10B encoder and decoder, receiver synchronization state machine, and ± 100 ppm clock compensation circuitry
- Support for other protocol features such as MSB to LSB transmission in SONET/SDH configuration and spread-spectrum clocking in PCI Express (PIPE) configurations

2.5 AREA OF APPLICATIONS

FPGA's have gained rapid acceptance and growth over the past decade because they can be applied to a very wide range of applications. A list of typical applications includes: random logic, integrating multiple SPLDs, device controllers, communication encoding and filtering, small to medium sized systems with SRAM blocks, and many more. Other interesting applications of FPGAs are prototyping of designs later to be implemented in gate arrays, and also emulation of entire large hardware systems. Another promising area for FPGA application, which is only beginning to be developed, is the usage of FPGAs as custom computing machines. This involves using the programmable parts to "execute" software, rather than compiling the software for execution on a regular CPU. However, designs mapped into an FPGA are broken up into logic block-sized pieces and distributed through an area of the FPGA. Depending on the FPGA's interconnect structure, there may be various delays associated with the interconnections between these logic blocks. Thus, FPGA performance often depends more upon how CAD tools map circuits into the chip than is the case for CPLDs. We believe that over time programmable logic will become the dominant form of digital logic design and implementation. Their ease of access, principally through the low cost of the devices, makes them attractive to small firms and small parts of large companies. The fast manufacturing turn-around they provide is an essential element of success in the market.

3. VHDL Language and Technology

3.1 INTRODUCTION

WHAT DOES VHDL STAND FOR ?

Very **H**igh **S**peed **I**ntegrated **C**ircuits

Hardware **D**escription **L**anguage

WHAT IS A PROCESS ?

The use of processes makes your code more modular, more readable, and allows you to separate combinational logic from sequential logic.

PACKAGES

Packages offers a mechanism to globally define and share values, types, components, functions and procedures that are commonly used.package declaration and package body

STANDARD VHDL PACKAGES

The following packages should be installed along with the VHDL compiler and simulator. The packages that you need, except for "standard", must be specifically accessed by each of your source files with statements such as:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use IEEE.std_logic_arith.all;
use IEEE.numeric_bit.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
use IEEE.math_complex.all;
```

VHDL DESIGN ORGANIZATION:

Entity
the "symbol" (input/output ports)

Architecture
one of the several possible implementation of the design

Configuration
binding between the symbol and one of the many possible implementation.

VHDL MODELING TYPES:

- Structural
- Dataflow (or RTL)
- Behavioral

OBJECT TYPES

- Constants
- Signals
- Variables

3.2 PREDEFINED DATA TYPES

- **bit**: '0' , '1'
- **bit_vetor():**
- **std_logic_vector()**:
- **boolean**: false, true
-
- **integer**: from negative 2^{31}-1 to positive 2^{31}-1
- **std_ulogic**: '1','0','H','L','X','U','Z','-','W'
- **std_logic**: '1','0','H','L','X','U','Z','-','W'

3.3 VHDL OPERATORS

Logical Operators

NOT, AND, NAND, OR, NOR, XOR and XNOR.

Arithmetic Operators

- + addition
- - subtraction
- * multiplication
- / division
- ABS absolute value
- MOD modulus
- REM remainder
- ** exponent

Comparison Operators

- = equal to
- /= not equal to
- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to

Shift Operators

- sll – shift left logical
- srl – shift right logical
- sla – shift left arithmetic
- sra – shift right arithmetic
- rol – rotate left
- ror – rotate right

3.4 ATTRIBUTES

Info attached to VHDL objects
Some predefined attributes:

‘left ‘right	the leftmost value of a type
‘high	the greatest value of a type
‘low	
‘length	the number of elements in an array
‘event	a change on a signal or variable
‘range	the range of the elements of an array object

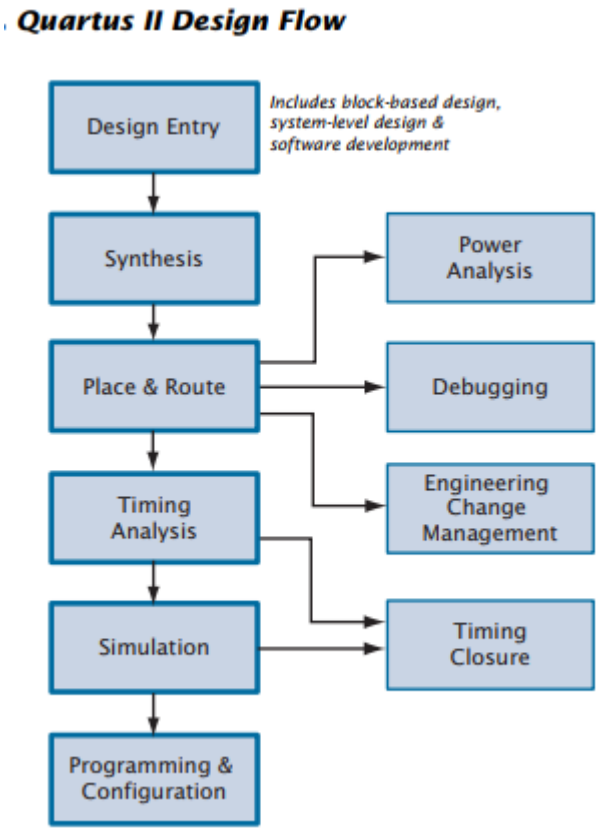
3.5 COMPLEX TYPES

ENUMERATED TYPES
TYPE color is (red, blue, yellow, green)
ARRAY
TYPE dbus is ARRAY (31 downto 0) of std_logic
RECORD
TYPE instruction is RECORD
opcode: integer; src: integer; dest: integer;
END RECORD
FILE
TYPE ram_data_file_t IS FILE OF INTEGER; FILE ram_data_file : ram_data_file_t IS IN “/claudio/vhdl/tb/ram.txt”

4 QUARTUS II SOFTWARE

4.1 INTRODUCTION

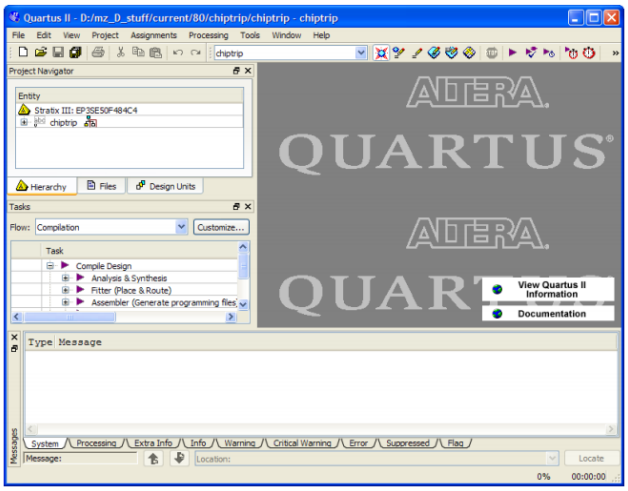
The Altera Quartus II design software provides a complete, multiplatform design environment that easily adapts to your specific design needs. It is a comprehensive environment for system-on-a-programmable-chip (SOPC) design. The Quartus II software includes solutions for all phases of FPGA and CPLD design



In addition, the Quartus II software allows you to use the Quartus II graphical user interface and command-line interface for each phase of the design flow. You can use one of these interfaces for the entire flow, or you can use different options at different phases.

4.2 Graphical User Interface Design Flow

The Quartus II software includes a modular Compiler. The Compiler includes the following modules (modules marked with an asterisk are optional during a compilation, depending on your settings):



QUARTUS II GRAPHICAL USER INTERFACE

- Analysis & Synthesis
- Partition Merge
- Fitter
- Assembler
- TimeQuest Timing Analyzer
- Design Assistant
- EDA Netlist Writer
- HardCopy® Netlist Writer

To run all Compiler modules as part of a full compilation, on the Processing menu, click Start Compilation. You can also run each module individually by pointing to Start on the Processing menu, and then clicking the command for the module you want to start.

In addition, you can use the Tasks window to start Compiler modules individually (Figure 3). The Tasks window also allows you to change settings or view the report file for the module, or to start other tools related to each stage in a flow.

The following steps describe the basic design flow for using the Quartus II GUI:

1. To create a new project and specify a target device or device family, on the File menu, click New Project Wizard.

2. Use the Text Editor to create a Verilog HDL, VHDL, or Altera Hardware Description Language (AHDL) design.

3. Use the Block Editor to create a block diagram with symbols that represent other design files, or to create a schematic.

4. Use the MegaWizard® Plug-In Manager to generate custom variations of megafunctions and IP functions to instantiate in your design, or create a system-level design by using SOPC Builder or DSP Builder.

5. Specify any initial design constraints using the Assignment Editor, the Pin Planner, the Settings dialog box, the Device dialog box, the Chip Planner, the Design Partitions window, or the Design Partition Planner.

6. (Optional) Perform an early timing estimate to generate early estimates of timing results before fitting.

7. Synthesize the design with Analysis & Synthesis.

8. (Optional) If your design contains partitions and you are not performing a full compilation, merge the partitions with partition merge.

9. (Optional) Generate a functional simulation netlist for your design and perform a functional simulation with an EDA simulation tool.

10. Place and route the design with the Fitter.

11. Perform a power estimation and analysis with the PowerPlay Power Analyzer.

12. Use an EDA simulation tool to perform timing simulation for the design.

13. Use the TimeQuest Timing Analyzer to analyze the timing of your design.

14. (Optional) Use physical synthesis, the Chip Planner, LogicLock™ regions, and the Assignment Editor to correct timing problems.

15. Create programming files for your design with the Assembler, and then program the device with the Programmer and Altera programming hardware.

16. (Optional) Debug the design with the SignalTap® II Logic Analyzer, an external logic analyzer, the SignalProbe feature, or the Chip Planner.

17. (Optional) Manage engineering changes with the Chip Planner, the Resource Property Editor, or the Change Manager.

4.3 Creating a Project

You can create a new project by clicking New Project Wizard on the File menu. When creating a new project, you specify the working directory for the project, assign the project name, and designate the name of the top-level design entity. You can also specify which design files, other source files, user libraries, and EDA tools you want to use in the project, as well as the target device.

The Project Navigator provides a graphical representation of the project hierarchy, files, and design units, and shortcuts to various menu commands.

Creating a Design Using VHDL

You can use the Quartus II Text Editor or another text editor to create Text Design Files, Verilog Design Files, and VHDL Design Files, and combine them with other types of design files in a hierarchical design.
VHDL Design Files can contain any combination of Quartus II–supported constructs. They can also contain Altera-provided logic functions, including primitives and megafunctions, and user-defined logic functions.
For VHDL designs, you can specify the name of a VHDL library for a design in the Properties dialog box, which is available from the Files page of the Settings dialog box on the Assignments menu.

Using Altera Megafunctions

Altera megafunctions are complex or high-level building blocks that can be used together with gate and flipflop primitives in Quartus II design files. The parameterizable megafunctions and LPM functions provided by Altera are optimized for Altera device architectures. You must use megafunctions to access some Altera device-specific features, such as memory, DSP blocks, LVDS drivers, PLLs, and SERDES and DDIO circuitry.
You can use the MegaWizard Plug-In Manager on the Tools menu to create Altera megafunctions, LPM functions, and IP functions for use in designs in the Quartus II software and EDA design entry and synthesis tools. Table 1 shows the types of Altera-provided megafunctions and LPM functions that you can create with the MegaWizard Plug-In Manager.

4.4 Synthesis Using Quartus II VHDL Integrated

You can use Analysis & Synthesis to analyze and synthesize VHDL designs. Analysis & Synthesis includes Quartus II Integrated Synthesis, which fully supports the Verilog HDL and VHDL languages and provides options to control the synthesis process.

Analysis & Synthesis supports the Verilog-1995 and Verilog-2001standards, a subset of features of the SystemVerilog-2005 standard, and also supports the VHDL 1987 and 1993 standards. You can select which standard to use; Analysis & Synthesis uses Verilog-2001 and VHDL 1993 by default. If you are using another EDA synthesis tool, you can also specify a Library Mapping File (.lmf) that the Quartus II software should use to map non–Quartus II functions to Quartus II functions. You can specify these and other options in the Verilog HDL Input and VHDL Input pages, which are under Analysis & Synthesis Settings in the Settings dialog box.

4.5 PROGRAMMING & CONFIGURATION

Once you have successfully compiled a project with the Quartus II software, you can program or configure an Altera device. The Assembler module of the Quartus II Compiler generates programming files that the Quartus II Programmer can use to program or configure a device with Altera programming hardware. You can also use a stand-alone version of the Quartus II Programmer to program and configure devices.

The Assembler automatically converts the Fitter's device, logic cell, and pin assignments into a programming image for the device, in the form of one or more Programmer Object Files (.pof) or SRAM Object Files (.sof) for the target device.

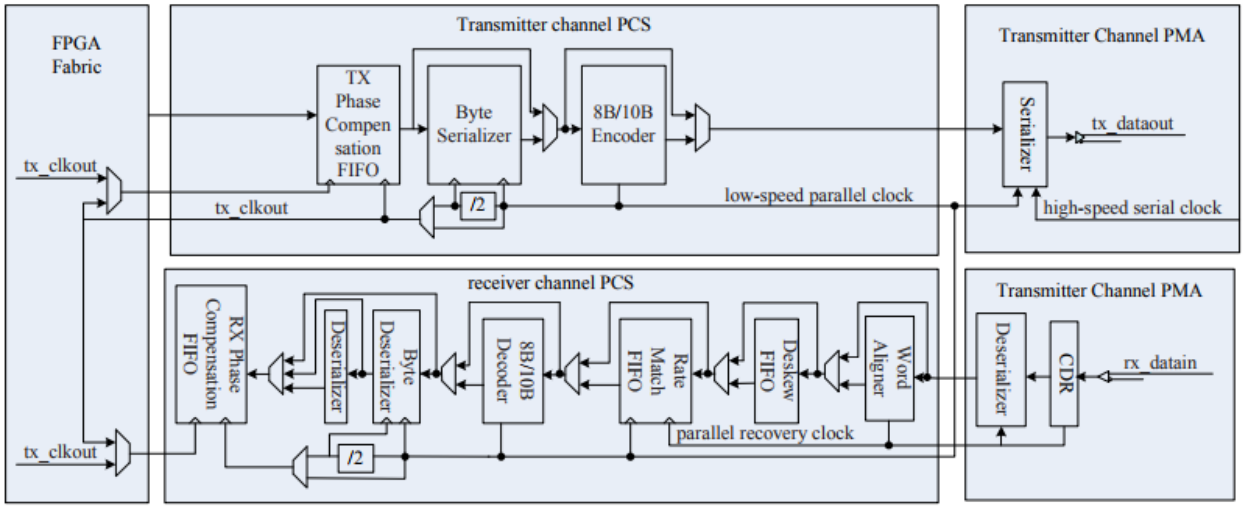
The Programmer uses the Programmer Object Files and SRAM Object Files generated by the Assembler to program or configure all Altera devices supported by the Quartus II software. You use the Programmer with Altera programming hardware, such as the MasterBlaster™, ByteBlasterMV™, ByteBlaster™ II, USB-Blaster™, or EthernetBlaster download cable; or the Altera Programming Unit (APU).

The Programmer allows you to create a Chain Description File (.cdf) that contains the name and options of devices used for a design. You can also open a JTAG Chain File (.jcf) or FLEX Chain File (.fcf) and save it in the Quartus II Programmer as a Chain Description File.

5. Design of a High Speed XAUI Based on Dynamic Reconfigurable Transceiver

5.1 Dynamic reconfigurable transceiver introduction

Figure shows the channel architecture of Stratix IV GX transceiver. Each transceiver channel consists of a transmitter channel and a receiver channel. Each transmitter or receiver channel is composed of one channel PCS block and one channel PMA block . Some modules in PCS block are optional, which can be bypassed.



Transceiver channel architecture for Stratix IV GX

Transmitter PCS consists of the TX phase compensation FIFO, byte serializer, and 8B/10B encoders. While the receiver PCS consists of the word aligner, deskew FIFO, rate-match FIFO, 8B/10B decoder, byte paralyzer, byte ordering, and RX phase compensation FIFO. Transmitter PMA consists of the serializer and the transmitter output buffer. Receiver PMA has the receiver input buffer, CDR, and paralyzer. The PMA module supports analog interface from the transceiver to medium outside.

Name	Single-Width	Double-Width
PMA-PCS interface widths	8/10 bit	16/20 bit
FPGA fabric-transceiver interface width	8/10 bit 16/20 bit	16/20 bit 32/40 bit

Name	Single-Width	Double-Width
Supported functional modes	■ PCI Express (PIPE) Gen1 and Gen2 ■ XAUI ■ GIGE ■ Serial RapidIO ■ SONET/SDH OC12 and OC48 ■ SDI ■ Basic single width	■ (OIF) CEI PHY Interface ■ SONET/SDH OC96 ■ Basic double-width
Data rate range in Basic functional mode	0.6 Gbps to 3.75 Gbps	1 Gbps to 8.5 Gbps

5.2 Dynamic reconfiguration architecture

IP cores are used in this design and a technology of high-performance IP core multiplex is chosen. These IP cores must be verified to make sure that they work well. Therefore, design efficiency can be improved if higher reliability and lower design risk is achieved. Stratix IV GX device offers two instances to support dynamic reconfiguration: ALTGX_ RECONFIG and ALTGX. Figure illustrates the connection relationships between them.

The ALTGX_RECONFIG instance generated by the ALTGX_RECONFIG MegaWizard Plug-In Manager represents the dynamic reconfiguration controller. It provides simple way to change transceiver PMA settings dynamically.

The ALTGX instance generated by the ALTGX MegaWizard Plug-In Manager represents the transceiver. This term is used as the functional module.

5.3 High speed XAUI design

As the demand for bandwidth and transfer rate increases, high speed transceiver plays a more important role in high speed interface design. The design based on dynamic reconfigurable transceiver of FPGA can fit the bill to some extent . Altera and Xilinx have already launched their own FPGA devices which meet the requirements above. With these devices, design of high speed interface becomes more simple, nevertheless, more realizable. And all this will make further promotion in the development of optical fiber communication and wireless devices.

5.4 Signal description

In the work of RTL coding, we should connect the ALTGX_RECONFIG instance with the ALTGX instances. The most important work is that signals must be connected to the related ports.

Port Name	Input/Output	Description	Scope
Clock Multiplier Unit (CMU)			
pll_inclk	Input	Input reference clock for the CMU phase-locked loop (PLL).	Transceiver block
pll_locked	Output	CMU PLL lock indicator. A high level indicates that the CMU PLL is locked to the input reference clock; a low level indicates that the CMU PLL is not locked to the input reference clock. Asynchronous signal.	Transceiver block
pll_powerdown	Input	CMU PLL power down. When asserted high, the CMU PLL is powered down. When de-asserted low, the CMU PLL is active and locks to the input reference clock. Asynchronous signal. The minimum pulse-width is 1 µs	Transceiver block

gxb_powerdown	Input	Transceiver block power down. When asserted high, all digital and analog circuitry within the PCS, PMA, CMU channel, and the CCU of the transceiver block, is powered down. Asserting the gxb_ powerdown signal does not power down the refclk buffers. Asynchronous signal. The mini Pulse width is 1 µs .	Transceiver block
tx_datain	Input	Parallel data input from the FPGA fabric to the Trans mitter. The bus width depends on thechannel width multiplied by the number of channels per instance.	Channel
tx_dataout	Output	Transmitter serial data output port.	Channel
tx_digitalreset	Input	Transmitter PCS reset. When asserted high,Trans- mitter PCS blocks are reset.	Channel
tx_coreclk	Input	Optional write clock port for the transmitter phase compensation FIFO. If not selected, Quartus II software automatically selects tx_clkout/coreclkout as the write clock for transmitter phase compen sation FIFO.	Channel
tx_ctrlenable	Input	8B/10B encoder /Kx.y/ or /Dx.y/ control. When asserted high, the 8B/10B encoder encodes the data on the tx_datain port as a /Kx.y/ control code group. When de-asserted low, it encodes the data on the tx_datain port as a /Dx.y/ data code group. The width of this signal depends on the channelwidth.	Channel
tx_clkout	Output	FPGA fabric-transceiver interface clock. Eachchannel has a tx_clkout signal in non-bonded channel configurations. Use this clock signal to clock the parallel data tx_datain from the FPGA fabric into the transmitter. This signal is not available in bonded channel configurations.	Channel

rx_datain	Input	Receiver serial data input port.	Channel
rx_dataout	Output	Parallel data output from the receiver to the FPGA fabric. The bus width depends on the channel width multiplied by the number of channels per instance.	Channel
rx_cruclk	Input	Input reference clock for the receiver clock and data recovery.	Channel
rx_clkout	Output	Recovered clock from the receiver channel. This feature is available only when the rate match FIFO is not used in the receiver datapath.	Channel
rx_digitalreset	Input	Receiver PCS reset. When asserted high, the receiver PCS blocks are reset.	Channel
rx_analogreset	Input	Receiver PMA reset. When asserted high, analog circuitry within the receiver PMA gets reset.	Channel
rx_pll_locked	Output	Receiver CDR lock-to-ref indicator. A high level indicates that the receiver CDR is locked to the input reference clock. A low level indicates that the receiver CDR is not locked to the input reference clock. Asynchronous signal.	Channel
rx_locktodata	Input	CDR lock-to-data mode control signal. When asserted high, the receiver CDR is forced to lock-to-data mode. When de-asserted low,the receiver CDR lock mode depends on the rx_locktorefclk signal level.	Channel
rx_locktorefclk	Input	CDR lock-to-ref mode control signal. The rx_locktorefclk signal along with the rx_locktodata signal controls whether the receiver CDR is in lock-to-reference or lock-to-data mode, as follows: rx_locktodata/rx_locktorefclk 0/0–receiver CDR is in automatic mode 0/1–receiver CDR is in LTR mode 1/x–receiver CDR is in LTD mode Asynchronous signal.	Channel
rx_freqlocked	Output	Receiver CDR lock mode indicator. A high level indicates that the receiver CDR is in lock-to-data (LTD) mode. A low level indicates that the receiver CDR is in lock-to-reference mode. Asynchronous signal.	Channel
rx_syncstatus	Output	Word alignment synchronization status indicator. For the word aligner in automatic synchronization state machine mode, this signal is driven high if the condi tions required to remain in synchronization are met. It is driven low if the conditions required to lose synchronization are met.	Channel

rx_enabyteord	Input	Enable byte ordering control. This feature is available in configurations with the byte ordering block enabled. The byte ordering blockis rising-edge sensitive to this signal. A low-to-high transition triggers the byte ordering block to restart the byte ordering operation. Asynchronous signal.	Channel
rx_byteorderalignstatus	output	Byte ordering status indicator. This feature is available in configurations with the byte ordering block enabled. A high level indicates that the byte ordering block has detected the programmed byte ordering pattern in the LSByte of the received data from the byte deserializer.	Channel
busy	output	Indicates the status of the dynamic reconfiguration controller. Assertion on this signal indicates that the offset cancellation process is being executed on the receiver buffer as well as the receiver CDR. When this signal is de-asserted, it indicates that offset cancellation is complete	Channel

cal_blk_clk	Input	Clock for transceiver calibration blocks.	Device
reconfig_clk	Input	Dynamic reconfiguration clock. This clock is also used for offset cancellation in all modesexcept PCI Express (PIPE) mode. Thefrequency range of this clock is 2.5 MHz to 50 MHz when the transceiver channel isconfigured in **Transmitter only** mode. The frequency range of this clock is 37.5 MHz to50 MHz when the transceiver channel is configured in **Receiver only** or **Receiver and Transceiver** mode.	
reconfig_togxb	Input	From the dynamic reconfiguration controller.	
reconfig_fromgxb	Output	To the dynamic reconfiguration controller.	

6. TRANCEIVER VHDL CODE

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity tx_rx is
port
 (
  clk                      : in std_logic:='0';
  rst                      : in std_logic:='0';
  data_received            :out   std_logic_vector(15 downto 0);
  rx_enabyteord_sig  : buffer STD_LOGIC_VECTOR (0 DOWNTO 0);
  rx_byteorderalignstatus_sig : buffer std_logic_vector(0 downto 0);
  sync_status              : buffer std_logic_vector(1 downto 0)
 );
end tx_rx;


architecture behavioral of tx_rx is

-------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------- COMPONENT OF  RECONFIGURATION BLOCK
component reconfig
port
(
 reconfig_clk                    : IN STD_LOGIC ;
 reconfig_fromgxb                : IN STD_LOGIC_VECTOR (16 DOWNTO 0);
 busy                            : OUT STD_LOGIC ;
 reconfig_togxb                  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
);
end component;

-------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------- COMPONENT OF  TRANCEIVER BLOCK
component tx_rx_trx
port
(
 cal_blk_clk                : IN STD_LOGIC ;
 gxb_powerdown              : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
 pll_inclk                  : IN STD_LOGIC ;
 pll_powerdown : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
 reconfig_clk               : IN STD_LOGIC ;
 reconfig_togxb : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
 rx_analogreset             : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
```

```
 rx_datain                    : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
 rx_digitalreset              : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
 rx_enabyteord  : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
 rx_locktodata                : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
 rx_locktorefclk  : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
 tx_ctrlenable                : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
 tx_datain                    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
 tx_digitalreset   : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
 pll_locked                   : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
 reconfig_fromgxb         : OUT STD_LOGIC_VECTOR (16 DOWNTO 0);
 rx_byteorderalignstatus: OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
 rx_clkout                    : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
 rx_dataout                   : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
 rx_freqlocked    : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
 rx_pll_locked                : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
 rx_syncstatus              : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
 tx_clkout                    : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
 tx_dataout                   : OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
);
 end component;

-------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------TRANCEIVER SIGNALS

 signal   cal_blk_clk_sig         : std_logic;
 signal   gxb_powerdown_sig : STD_LOGIC_VECTOR (0 DOWNTO 0):="0";
 signal   pll_inclk_sig           : STD_LOGIC :='0';
 signal pll_powerdown_sig    : STD_LOGIC_VECTOR (0 DOWNTO 0):="0";
 signal reconfig_clk_sig         : STD_LOGIC :='0';
 signal reconfig_togxb_sig     : STD_LOGIC_VECTOR (3 DOWNTO 0);
 signal reconfig_fromgxb_sig : STD_LOGIC_VECTOR (16 DOWNTO 0);
 signal rx_analogreset_sig     : STD_LOGIC_VECTOR (0 DOWNTO 0);
 signal rx_datain_sig             :  STD_LOGIC_VECTOR (0 DOWNTO 0);
 signal rx_digitalreset_sig      : STD_LOGIC_VECTOR (0 DOWNTO 0);
 signal rx_locktodata_sig       : STD_LOGIC_VECTOR (0 DOWNTO 0):="0";
 signal rx_locktorefclk_sig     : STD_LOGIC_VECTOR (0 DOWNTO 0):="0";
 signal tx_ctrlenable_sig       : STD_LOGIC_VECTOR (1 DOWNTO 0):="11";
 signal tx_datain_sig             : STD_LOGIC_VECTOR (15 DOWNTO 0);
 signal tx_digitalreset_sig      : STD_LOGIC_VECTOR (0 DOWNTO 0):="1";
 signal tx_dataout_sig           :  std_logic_vector(0 downto 0);
 signal rx_dataout_sig           :  std_logic_vector(15 downto 0);
 signal   rx_cruclk_sig           : STD_LOGIC_VECTOR (0 DOWNTO 0) :=(OTHERS => '0');
 signal pll_locked_sig          : STD_LOGIC_VECTOR (0 DOWNTO 0):="0";
 signal rx_clkout_sig            : STD_LOGIC_VECTOR (0 DOWNTO 0);
 signal rx_freqlocked_sig       : STD_LOGIC_VECTOR (0 DOWNTO 0);
 signal rx_pll_locked_sig       : STD_LOGIC_VECTOR (0 DOWNTO 0);
 signal rx_syncstatus_sig       : STD_LOGIC_VECTOR (1 DOWNTO 0);
 signal tx_clkout_sig            : STD_LOGIC_VECTOR (0 DOWNTO 0):="0";
 signal busy_sig                  :  STD_LOGIC ;
```

```
-------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------STATIC DATA SIGNALS

 signal CLK50MHZ: std_logic:='0';
 signal count         : integer := 0;
 signal cnt           : integer := 0;
 signal txd           : std_logic_vector(15 downto 0);
 signal i0            : std_logic_vector(15 downto 0):="1010101010101010";
 signal i1            : std_logic_vector(15 downto 0):="0101010101010101";
 signal i2            : std_logic_vector(15 downto 0):="1010101110101011";
 signal i3            : std_logic_vector(15 downto 0):="0101110001011100";
 signal i4      : std_logic_vector(15 downto 0):="1010110110101101";
 signal i5      : std_logic_vector(15 downto 0):="0101111001011110";
 signal i6      : std_logic_vector(15 downto 0):="1010100110101001";
 signal i7      : std_logic_vector(15 downto 0):="1111010111110101";

begin

-------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------PORT MAPPING OF TRANCEIVER BLOCK

 tx_rx_trx_inst : tx_rx_trx PORT MAP (
  cal_blk_clk              => cal_blk_clk_sig,
   gxb_powerdown       => gxb_powerdown_sig,
   pll_inclk                 => pll_inclk_sig,
   pll_powerdown         => pll_powerdown_sig,
   reconfig_clk           => reconfig_clk_sig,
   reconfig_togxb         => reconfig_togxb_sig,
   rx_analogreset         => rx_analogreset_sig,
   rx_datain                 => rx_datain_sig,
   rx_digitalreset          => rx_digitalreset_sig,
   rx_enabyteord          => rx_enabyteord_sig,
   rx_locktodata          => rx_locktodata_sig,
   rx_locktorefclk         => rx_locktorefclk_sig,
   tx_ctrlenable           => tx_ctrlenable_sig,
   tx_datain                 => tx_datain_sig,
   tx_digitalreset          => tx_digitalreset_sig,
   pll_locked                => pll_locked_sig,
   reconfig_fromgxb     => reconfig_fromgxb_sig,
   rx_byteorderalignstatus       => rx_byteorderalignstatus_sig,
   rx_clkout                 => rx_clkout_sig,
   rx_dataout                => rx_dataout_sig,
   rx_freqlocked          => rx_freqlocked_sig,
   rx_pll_locked          => rx_pll_locked_sig,
   rx_syncstatus          => rx_syncstatus_sig,
   tx_clkout                 => tx_clkout_sig,
   tx_dataout                => tx_dataout_sig
);
```

```
-------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------PORT MAPPING OF RECONFIGURATION BLOCK

reconfig_inst : reconfig PORT MAP
(
  reconfig_clk            => reconfig_clk_sig,
  reconfig_fromgxb        => reconfig_fromgxb_sig,
  busy                    => busy_sig,
  reconfig_togxb          => reconfig_togxb_sig
);


-------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------SIGNAL SIGNAL ASSIGNMENT

  pll_inclk_sig           <= clk;
  rx_cruclk_sig(0)        <= clk;
 cal_blk_clk_sig          <= CLK50MHZ;
 reconfig_clk_sig         <= CLK50MHZ;
  gxb_powerdown_sig(0)    <= not rst;
  rx_locktodata_sig       <="0";
  rx_locktorefclk_sig     <="0";
  rx_datain_sig           <=  tx_dataout_sig ;


-------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------PLL POWERDOWN SIGNAL

p0: process(clk, rst)
variable cnt1 : integer := 0;
begin
    if rst = '0' then
      pll_powerdown_sig         <= "1";
      cnt1          := 0;
    elsif rising_edge(clk) then
       if cnt1 <101 then
          pll_powerdown_sig     <= "1";
          cnt1 := cnt1 + 1;
       else
          pll_powerdown_sig     <= "0";
       end if;
    end if;
end process;
```

```
-------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------TRANSMITTER DIGITAL RESET SIGNAL

p12: process(clk, rst, pll_powerdown_sig, pll_locked_sig)
variable cnt1 : integer := 0;
   begin
       if rst = '0' then
          tx_digitalreset_sig      <= "1";
          cnt1 := 0;
        elsif rising_edge(clk) then
            if pll_powerdown_sig = "0" then
              if pll_locked_sig = "1" then
                 if cnt1 > 104 then
                    tx_digitalreset_sig       <= "0";
                 else
                    tx_digitalreset_sig      <= "1";
                    cnt1 := cnt1 + 1;
                 end if;
              end if;
           end if;
        end if;
 end process;


-------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------RECEIVER ANALOG RESET SIGNAL

p13: process(clk, rst, pll_powerdown_sig, busy_sig)
variable cnt1 : integer := 0;
begin
   if rst = '0' then
      rx_analogreset_sig  <= "1";
      cnt1 := 0;
   elsif rising_edge(clk) then
       if pll_powerdown_sig = "0" then
          if busy_sig = '0' then
             if cnt1 > 110 then
                rx_analogreset_sig  <= "0";
             else
                rx_analogreset_sig  <= "1";
                cnt1 := cnt1 + 1;
             end if;
          end if;
     end if;
  end if;
end process;
```

```
-------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------- RECEIVER DIGITAL RESET SIGNAL

p14: process(clk, rst, pll_powerdown_sig, rx_freqlocked_sig)
variable cnt1 : integer := 0;
begin
  if rst = '0' then
    rx_digitalreset_sig <= "1";
    cnt1 := 0;
  elsif rising_edge(clk) then
     if pll_powerdown_sig = "0" then
       if rx_freqlocked_sig = "1" then
         if cnt1 > 400 then
           rx_digitalreset_sig <= "0";
         else
           rx_digitalreset_sig <= "1";
           cnt1 := cnt1 + 1;
         end if;
       end if;
    end if;
  end if;
end process;

-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------50MHZ CLOCK

P1: PROCESS(CLK, rst)
begin
   IF RST = '0' THEN
      CLK50MHZ <= '0';
   ELSIF RISING_EDGE(CLK) THEN
       CLK50MHZ <= NOT CLK50MHZ;
   END IF;
END PROCESS;

-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------DATA TO BE SEND

P2: PROCESS( tx_clkout_sig,tx_digitalreset_sig)
variable count_value:integer:=0;
begin
    IF tx_digitalreset_sig = "1" THEN
        txd<= "0000000000000000";
   elsif RISING_EDGE(tx_clkout_sig(0)) then
       if tx_ctrlenable_sig = "00" then
         if(count_value=0)then
            txd<=i0;
```

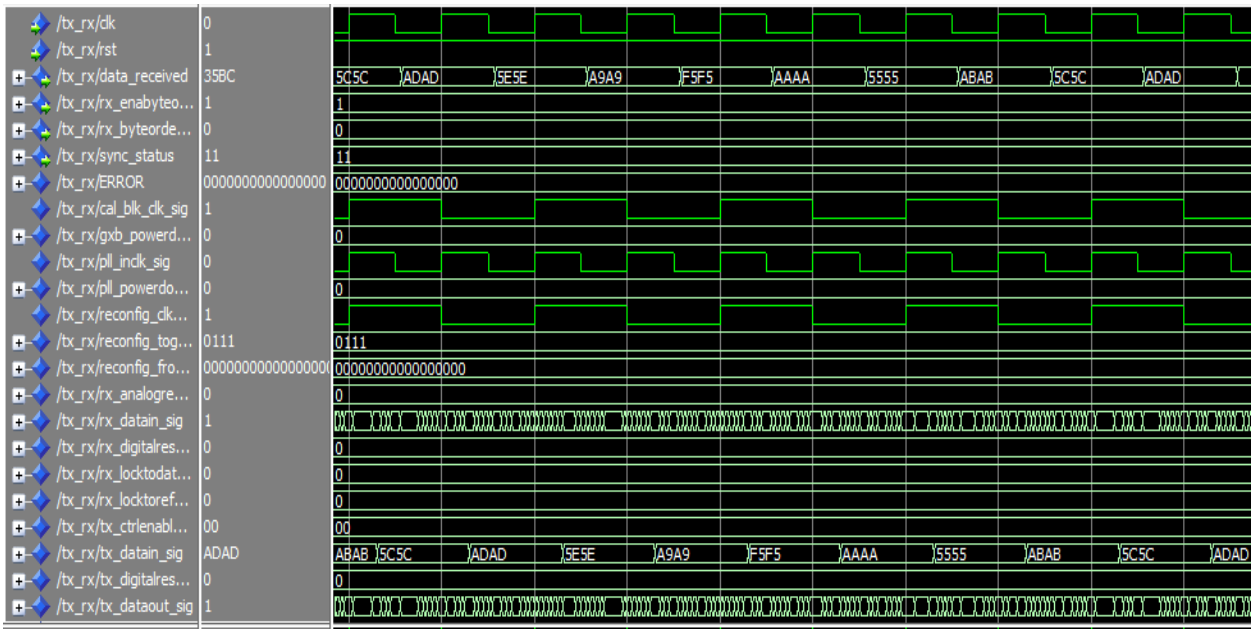
```
      elsif(count_value=1)then
            txd<=i1;
          elsif(count_value=2)then
            txd<=i2;
          elsif(count_value=3)then
            txd<=i3;
          elsif(count_value=4)then
            txd<=i4;
          elsif(count_value=5)then
            txd<=i5;
          elsif(count_value=6)then
            txd<=i6;
          elsif(count_value=7)then
            txd<=i7;
            count_value:=-1;
          end if;
        count_value:=count_value+1;
      end if;
   end if;
END PROCESS;

-------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------DATA TRANSMISSION

process(tx_clkout_sig, tx_digitalreset_sig, sync_status)
variable count : natural:=0;
begin
   if tx_digitalreset_sig = "1" then
      count := 0;
      tx_ctrlenable_sig <= "11";
   elsif rising_edge(tx_clkout_sig(0)) then
       if (count< 100) then
          count := count + 1;
          tx_ctrlenable_sig <= "11";
          tx_datain_sig <= x"BC35";
   elsif sync_status     = "11" then
          tx_ctrlenable_sig <= "00";
          tx_datain_sig <= txd;
   else
           tx_ctrlenable_sig <= "11";
           tx_datain_sig <= x"BC35";
    end if;
  end if;
end process;
```

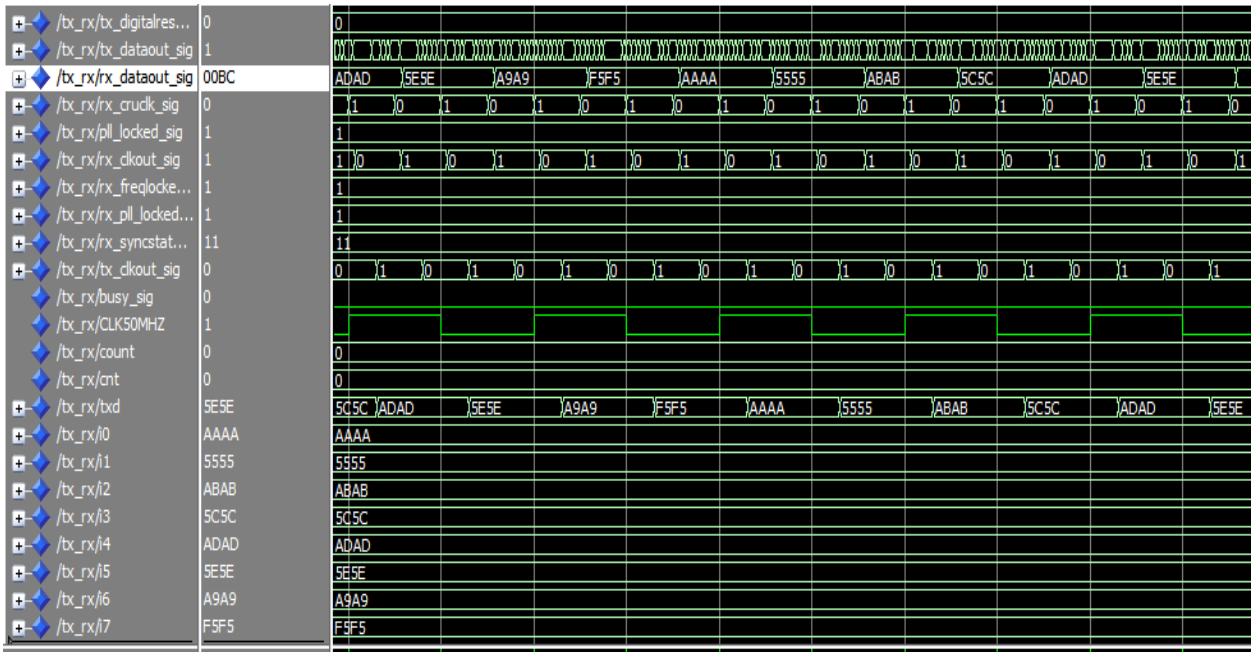
```
-------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------DATA RECEPTION
P16: PROCESS (rx_clkout_sig, rx_digitalreset_sig)
BEGIN
    if (rx_digitalreset_sig = "1") then
        data_received  <= (others => '0');
    elsif rising_edge(rx_clkout_sig(0)) then
        data_received  <= rx_dataout_sig;
    end if;
end process;


-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------SYNCHRONOUS STATUS SIGNAL

P17: PROCESS (rx_clkout_sig, rx_digitalreset_sig)
BEGIN
   if (rx_digitalreset_sig = "1") then
       sync_status <= "00";
   elsif rising_edge(rx_clkout_sig(0)) then
       sync_status <= rx_syncstatus_sig;
   end if;
end process;


-------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------RECEIVER ENABLE BYTE ORDERING SIGNAL

P21: process(rx_clkout_sig, rx_digitalreset_sig, rx_syncstatus_sig)
begin
    if (rx_digitalreset_sig = "1") then
        rx_enabyteord_sig <= "0";
     elsif rising_edge(rx_clkout_sig(0)) then
        if rx_syncstatus_sig = "11" then
          rx_enabyteord_sig <= "1";
        else
          rx_enabyteord_sig <= "0";
        end if;
     end if;
end process;
end behavioral;
```

7. RESULTS ANALYSIS

Result analysis After the design of high speed XAUI based on dynamic transceiver,I did functional simulation in Modelsim Altera v10.1e software. Then further work was done, such as synthesis, placement and routing in Atlera Stratix IV GX FPGA Development kit using Quartus II v14.0.

Transmitted and received packets





8. CONCLUSION

In the design of the proposed high speed XAUI, dynamic reconfigurable transceiver allows us to set related parameters from outside registers, which makes the best eye diagram practicable. By virtue of IP core usage, RTL coding in FPGA and SOC design becomes easy and reliable in many research areas . However, there are several parameters to set, so it's really an arduous task which increased difficulties. In the design, two authenticated XAUI cores are adopted which work as 1+1 protection. Though great reliability can be achieved in this method, disadvantages followed. It will reduce the channel utilization. After the process of synthesis with Quartus IV v14.0. As a result, the design method based on IP cores is benefit to cut down design cycle, shorten press time to market, improve design efficiency and increase design reliability.

9. REFERENCES

- Altera Inc., "Innovating With a Full Spectrum of 40-nm FPGAs and ASICs with Transceivers". http://www.altera.com.cn/literature/wp/wp-01078-stratix-iv-gt-40nm-transceivers.pdf
- Hou Hui, Cao Wei, Wang Jian, "Overview of Dynamic Reconfiguration", Semiconductor Technology, vol. 33, no. 7, pp.553-557, 2008.
- https://www.altera.com/products/fpga/stratix-series/stratix-iv/overview.html
- Altera Inc., "Stratix IV Device Handbook Volume 1: Transceivers" https://www.altera.com/content/d/en_US/pdfs/stratix-iv/stratix4_handbook.pdf
- Altera Inc., "Introduction to Quartus II Software Handbook" https://www.altera.com/content/dam/en_US/pdfs/literature/manual/intro_to_quartus2.pdf
- Wikipedia, "VHDL" https://en.wikipedia.org/wiki/VHDL
- "Circuit Design with VHDL" By Volnei A. Pedroni