TOE1G-IP reference design manual

Rev1.3 19-Aug-16

1. Introduction

TCP/IP is the core protocols of the Internet Protocol Suite for networking application. TCP/IP model has four layers, i.e. Application Layer, Transport Layer, Internet Layer, and Network Access Layer. In Figure 1, five layers are displayed for simple matching with hardware implementation by FPGA. Network Access Layer is split into Link and Physical Layer.

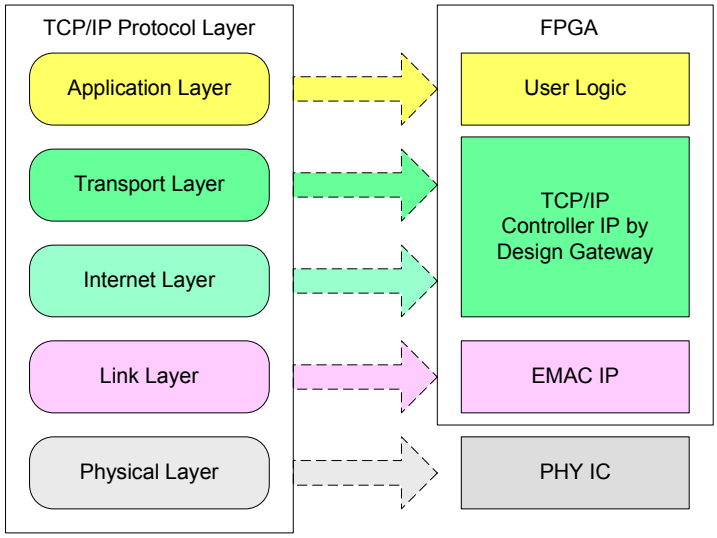

Figure 1 TCP/IP Protocol Layer

TOE1G-IP implements Transport and Internet layer of TCP/IP Protocol. To transmit data, TOE1G-IP will split long TCP data stream from user logic into many packets following TCP/IP standard. Then, combine TCP data with TCP/IP header which is generated within IP before sending out to EMAC. To receive data, TOE1G-IP will check TCP packet that data and header are valid. Then, extract TCP data and header from valid packet. Only TCP data will be stored in buffer for user logic reading.

Using TCP protocol can guarantee data reliability by monitoring acknowledge packet. During transmitting data, TOE1G-IP will retransmit data if acknowledge packet from receiver is lost. During receiving data, TOE1G-IP will monitor the sequence number of received packet. If lost packet is detected, it will send duplicate acknowledge packet to request the lost packet.

The lower layer protocols are implemented by EMAC-IP from Altera and external PHY chip.

This reference design provides evaluation system which includes simple user logic to send and receive data with TOE1G-IP. This system demonstrates on Altera Development board to operate with Test application on PC for transferring high speed data on network. More details are described as follows.

2. Environment

To operate this reference design, following environment must be setup.

- FPGA Development board
- QuartusII Programmer
- Ethernet cable (Cat5e or Cat6)
- PC with Gigabit Ethernet
- USB A-B cable for FPGA configuration
- Test Application, i.e. “send_tcp_client” and “recv_tcp_client”, provided by Design Gateway

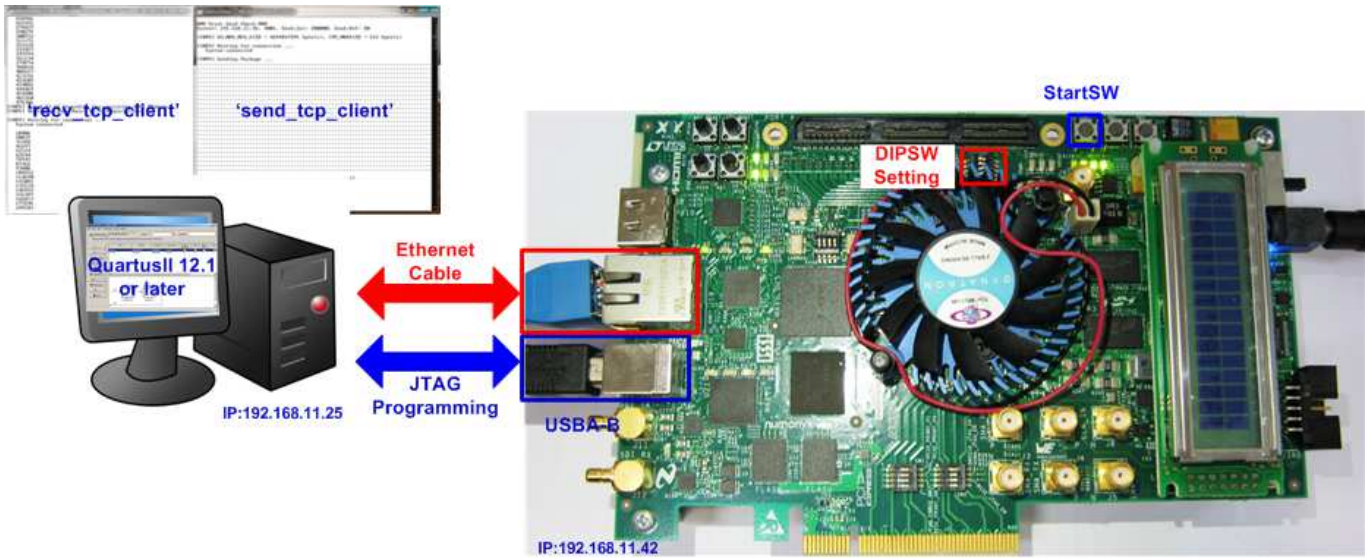

Figure 2 TOE1GIP Demo on ArriaV GX board

3. Hardware description

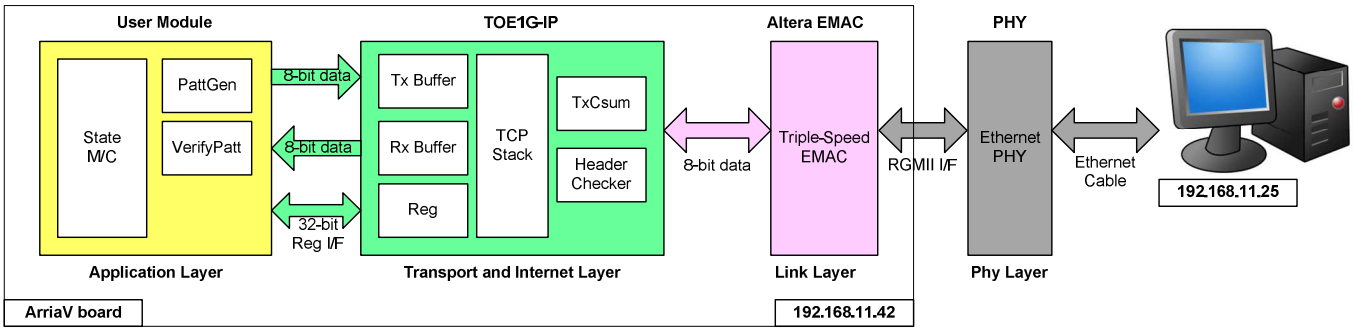

Figure 3 Hardware Architecture in ArriaV GX reference design (RGMII)

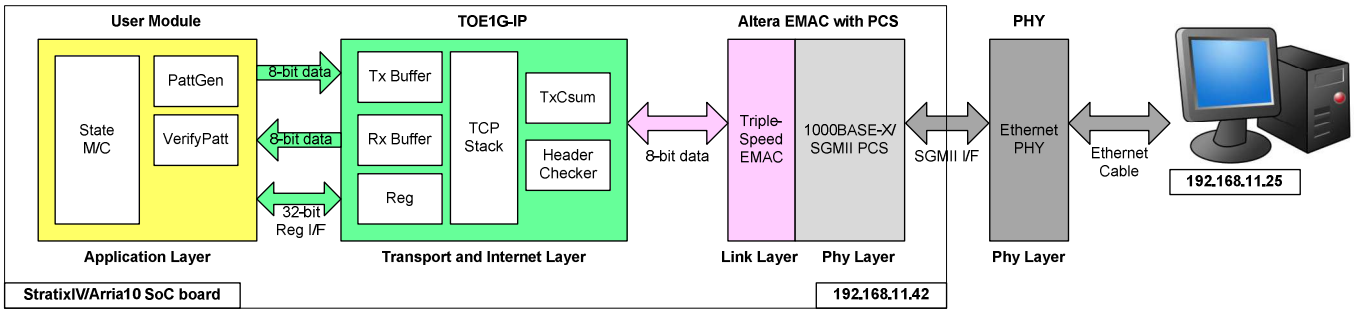

Figure 4 Hardware Architecture in StratixIV GX/Arria10 SoC reference design (SGMII)

As shown in Figure 3, hardware architecture can be divided into 4 modules to support each TCP/IP layer protocol. TOE1G-IP operates with EMAC and external PHY to implement all four lower layers of TCP/IP Protocol. User module can transfer TCP data by using FIFO interface and write/read control signal with TOE1G-IP by using Register interface. TCP data will be transferred to PC which will run test application to generate or verify data.

This reference design includes the example of User Module to generate Test pattern in transmitting mode or verify Test pattern in receiving mode on ArriaV/StratixIV GX/Arria10 SoC development board.

- External PHY

Physical layer is implemented by external PHY chip. The interface type of PHY chip can be three formats, i.e. SGMII (like StartixIV GX/Arria10 SoC board), RGMII (like ArriaV GX Starter board), or GMII.

- EMAC

Link layer and PCS/PMA are implemented by Triple-speed Ethernet MAC, provided by Altera. Internal FIFO and all options of EMAC are disabled to reduce resource. TOE1G-IP can connect to user interface of EMAC directly.

In the demo system, Avalon interface of EMAC is controlled by EMAC State machine. The state machine runs only one time to initialize basic parameters of EMAC and external PHY. The details about the register in EMAC are described in “Configuration Register Space” topic within “Triple-Speed Ethernet MegaCore Function User Guide” document, provided by Altera. EMAC State machine of SGMII and RGMII is different design.

For SGMII mode, EMAC state machine will program only Base Configuration area to configure MAC function such as disable/enable transmit and receive paths, frame length, Tx IPG length, and software reset.

For RGMII mode, EMAC state machine will program both Base Configuration and MDIO Space1 area. MDIO Space1 area is used to access external PHY register through MDIO interface for enable RGMII Receive/Transmit timing control function. When enable timing control bit, transmit and receive clock will include the delay to shift clock phase for synchronous with transmit/receive data.

- TOE1G-IP

More details about both modules are described in "dg_toe1gip_data_sheet_altera_en.pdf" document.

- User Module

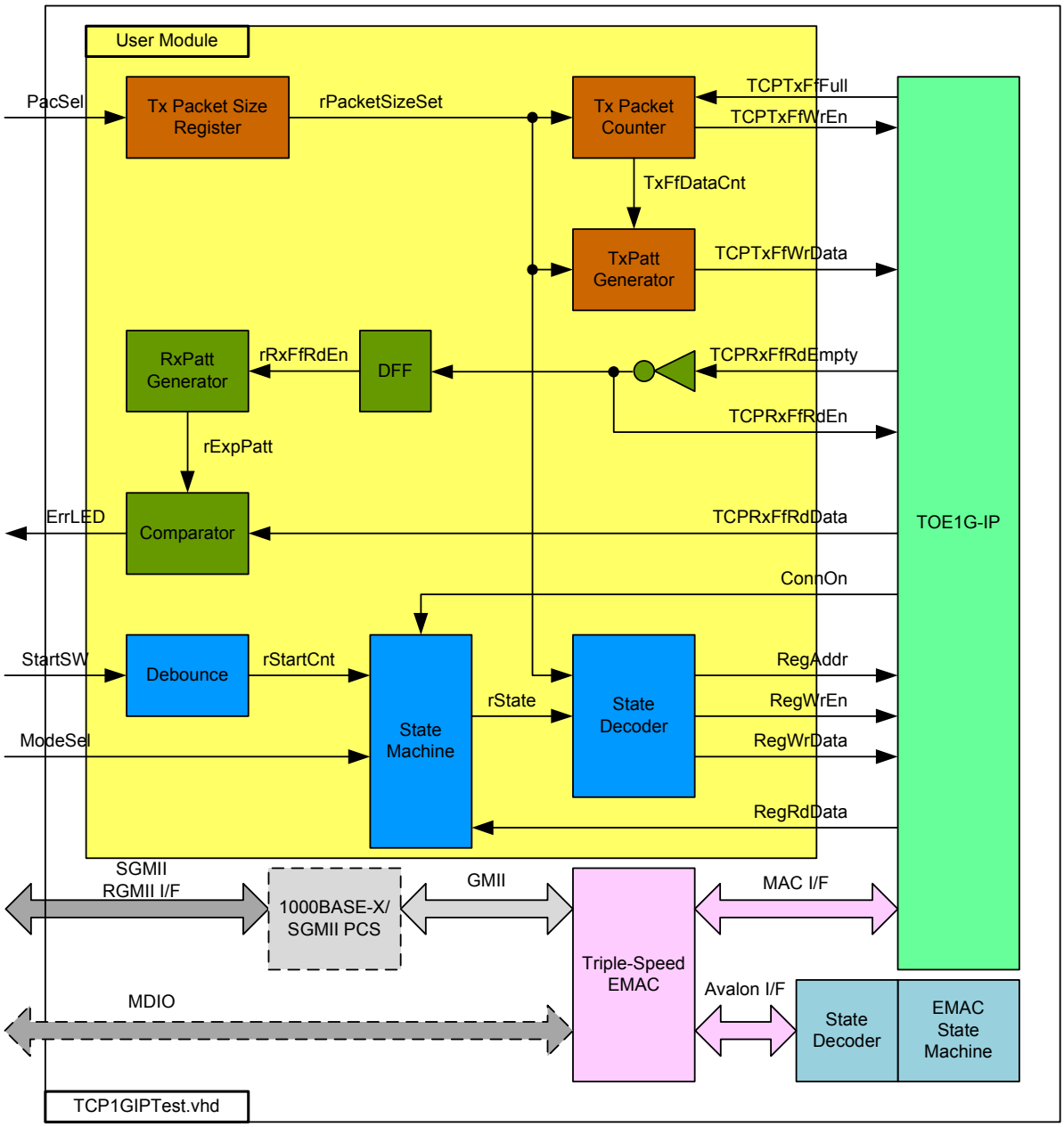

Figure 5 User Module and Test System block diagram

User Module can split into three parts, i.e. TxFIFO interface, RxFIFO interface, and Register Control interface. 32-bit increment test data is increased when end of sending each Tx packet. Tx Packet counter is designed to count data size in each Tx packet which can be set by external PacSel DIPSW. There are two supported sizes in this demo, i.e. 1460 bytes for non-Jumbo frame, and 8960 bytes for Jumbo frame.

32-bit increment data is also generated by RxPatt Generator to verify data output from RxFIFO interface of TOE1G-IP. Simple logic is designed to read out data from FIFO by monitoring Empty flag. ErrLED is blinking when data verification is error.

Register Control interface is designed by using State Machine. Register address and write value signals are decoded from State Machine. Transfer data direction is selected by ModeSel DIPSW, and data starts transferring when StartSW is pressed by user. State Machine diagram is displayed as shown in Figure 6.

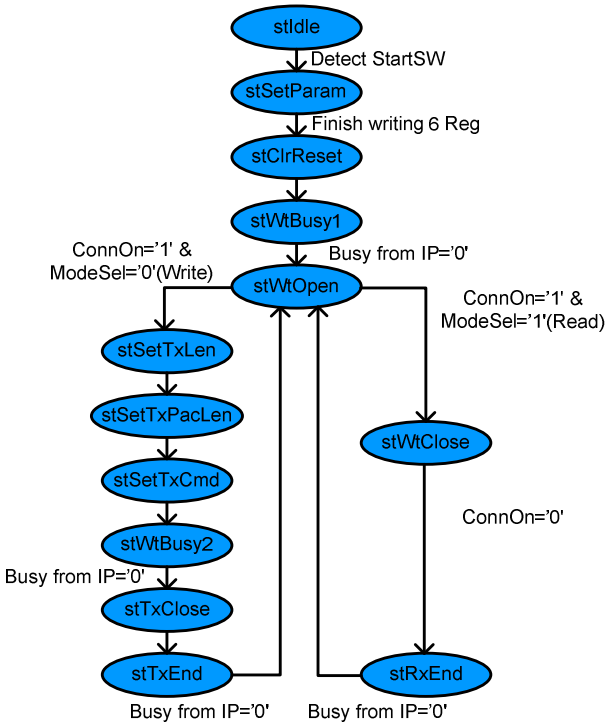

Figure 6 State Machine Diagram within User Module

State machine changes to stSetParam state after user press StartSW button. In stSetParam state, it will set parameters to register within TOE1G-IP, i.e.

- Source MAC address (SML/SMH Reg) = 00:01:02:03:04:05
- Source IP address (SIP Reg) = 192.168.11.42
- Source Port number (SPN Reg) = 4000
- Destination IP address (DIP Reg) = 192.168.11.25

Next State, stClrReset, is applied to release Reset signal (RST Reg=0) within TOE1G-IP and then starting parameter initialization. State machine waits initialization complete by monitoring Busy flag (bit 0 of CMD Reg) through Register interface. State will pause in stWtOpen until test application on PC starts running.

On this demo, FPGA runs as Server mode and Test application on PC runs as Client mode, so the connection is opened by Test application on PC. ConnOn output from TOE1G-IP will change to '1' and then state machine will change to stSetTxLen for transmitting mode or to stWtClose for receiving mode.

On transmitting mode, more three states, i.e. stSetTxLen, stSetTxPacLen, and stSetTxCmd are designed for setting total transfer size (TDL Reg), packet size (PKL Reg), and data sending to command register (CMD Reg=0) within TOE1G-IP. Then, Busy signal is monitored to wait transfer complete in stWtBusy2 state. After all data are transferred completely, State sends command to TOE1G-IP for sending packet to close connection (CMD Reg=0x3) in stTxClose state. After connection is closed and Busy='0', state will run in stWtOpen for waiting next transfer.

On receiving mode, state will stay in stWtClose to wait data transfer from PC complete and follow with connection closed command from PC. So, ConnOn value from TOE1G-IP will change from '1' to '0' after connection has already closed. Similar to transmitting mode, state will go back to stWtOpen for waiting next transfer.

In conclusion, the demo uses passive open mode for both transmitting and receiving data, but uses different mode to close connection. Active close connection is operated for data transmitting mode while passive close connection is operated for data receiving mode.

4. Test Software description

Two test applications are applied within this demo, i.e. “recv_tcp_client” and “send_tcp_client”. Both application runs as client mode.

- recv_tcp_client

This test application runs to test sending operation of TOE1G-IP, so data sending to PC will be verified by test application. This application requires three input parameters from user, i.e.

- FPGA IP address: This demo sets IP address to “192.168.11.42”. User can modify HDL code of User module to change this value.
- FPGA Port number: This demo sets Port number to “4000”. User can modify HDL code of User module to change this value.
- Packet size: This demo can set as two values, i.e. 1460 for non-Jumbo frame mode, and 8960 for Jumbo frame mode. If setting with wrong value, verified error message will be displayed on Test application and operation will be stopped.

The operation sequence of the application is follows.

(1) Get three parameters from user.
(2) Create socket and then set properties of received buffer.
(3) Set IP address and Port number from user parameter and then connect.
(4) Loop run for receiving data and verify data. Data format will be 32-bit increment value which will increase every end of packet. Thus, all data in same packet will be similar. Two errors can be printed out from verification process, i.e. “Drop Expect” printed out when 1st data of packet is not expect value, and “Error Expect” printed out when data within each packet is not expect value. Total number of packet is printed out on console every second.
(5) Socket is closed by FPGA side and then Performance with total number of transferred data will be printed out as test result.
(6) Go back to step (3) in loop to continue test operation until operation cancel.

- send_tcp_client

This test application sends data out to TOE1G-IP to test receiving operation. This application requires four input parameters from user, i.e.

- FPGA IP address: This demo sets IP address to “192.168.11.42”. User can modify HDL code of User module to change this value.
- FPGA Port number: This demo sets Port number to “4000”. User can modify HDL code of User module to change this value.
- Packet Count: This value is set transfer size in 16kByte unit. Total transfer size is equal to this value x 16kByte. Valid range is 1-262143.
- Verification On/Off: Select ‘0’ to transfer dummy data and ‘1’ to transfer 32-bit increment data out. This setting value is effect to output performance from PC. In some PC, performance in dummy data mode is better than increment data mode.

The operation sequence of the application is follows.

(1) Get three parameters from user.
(2) Create socket and then set properties of transmit buffer.
(3) Set IP address and Port number from user parameter and then connect.
(4) Fill test pattern with dummy (all ‘0’) or increment pattern to buffer and then send data out. Transfer size is set from user.
(5) Close socket and print out performance with total number of transferred data as test result.

5. Revision History

Revision	Date	Description
1.0	24-Jul-14	Initial Release
1.1	27-Nov-14	Update to support both StratixIV/ArriaV GX boards
1.2	12-Dec-14	Add EMAC register programming for RGMII
1.3	19-Aug-16	Change IP name and support Arria10 SoC