TOE10G-IP with CPU reference design

Rev1.0 22-Jan-18

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 1-1, five layers are displayed for simple matching with hardware implementation by FPGA. Network Access Layer is split into Link and Physical Layer.

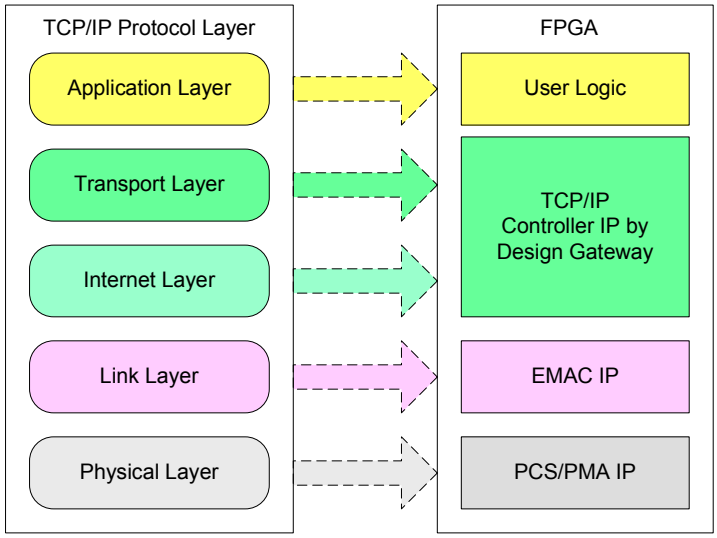

Figure 1-1 TCP/IP Protocol Layer

TOE10G-IP implements Transport and Internet layer of TCP/IP Protocol. For transmitted side, TOE10G-IP prepares TCP data from user logic, add TCP/IP header to generate Ethernet packet, and sends to EMAC. For received side, TOE10G-IP extracts TCP data and header from Ethernet packet. If TCP/IP header in the packet is valid, TCP data will be stored to the buffer and wait user logic reading.

The lower layer protocols are implemented by EMAC-IP and PCS/PMA-IP from Xilinx.

The reference design provides evaluation system which includes simple user logic to send and receive data by using TOE10G-IP. For user interface, CPU system is designed to interface with user through Serial console. The firmware is designed as bare-metal OS. Two test applications are applied in the demo, i.e. “tcpdatatest” for half-duplex test and “tcp_client_txrx_10G” for full-duplex test. The reference design is available on Xilinx development board to show ultra high-speed transfer with network reliability. More details of the demo are described as follows.

2 Hardware description

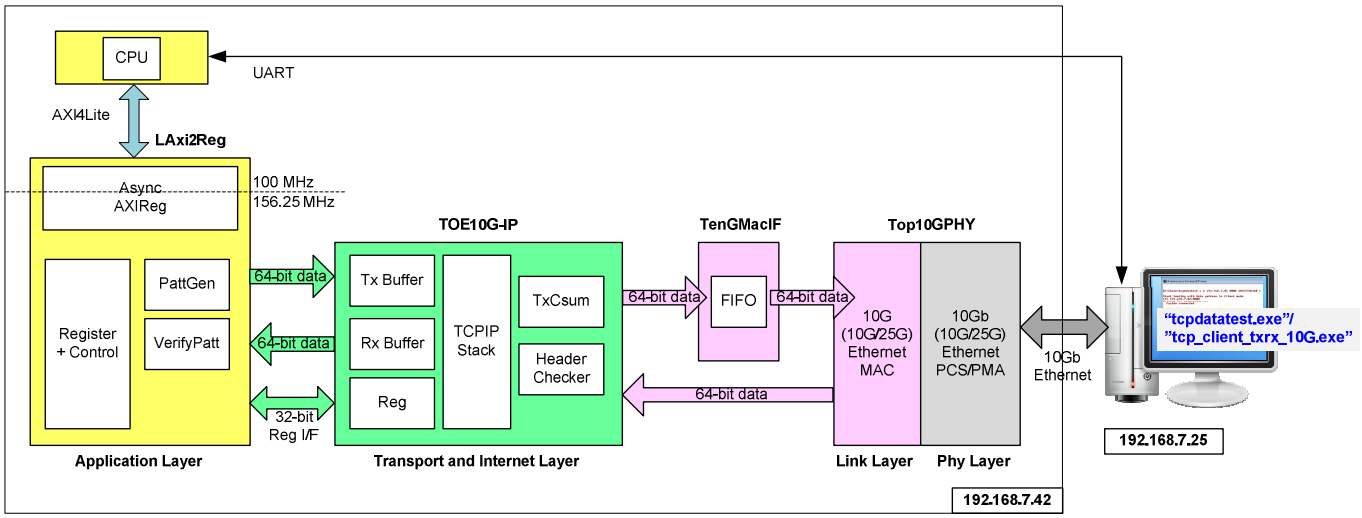

Figure 2-1 Demo Block Diagram

TOE10G-IP needs to connect to 10G Ethernet MAC and 10 Gb Ethernet PCS/PMA to complete all TCP/IP layer implementation (For Ultrascale+ device, it uses 10G/25G Ethernet Subsystem instead of 10G Ethernet). The user interfaces of TOE10G-IP are connected to LAxi2Reg for both control and data interface. For data interface, UserReg includes PattGen to generate test pattern for sending data. Also, VerifyPatt includes test pattern generator for verifying received data. Test pattern in the reference design is fixed to 32-bit increment data.

For control interface, LAxi2Reg includes register to store parameters from user such as transfer length and test mode which are received through Serial console. CPU firmware validates all parameters from user and then sets the valid value to TOE10G-IP and LAxi2Reg parameters through AXI4-Lite bus. Due to the fact that CPU system and TOE10G-IP run in different clock domain, AsyncAXIReg module is used to be asynchronous circuit to support clock-crossing function and convert AXI4-Lite bus signal which is standard bus in CPU system to be register interface. CPU in the demo runs in bare-metal OS. In CPU system, UART hardware is included for user interface. Timer is also designed in CPU system for measuring transfer performance.

Two applications on PC are applied in the demo. The first application is “tcpdatatest.exe” which is designed to send or receive Ethernet data with TOE10G-IP. Data transferring is half-duplex mode. The second application is “tcp_client_txrx_10G.exe” which is designed to send and receive Ethernet data by using same TCP port number at the same time (full-duplex mode). In full-duplex mode, PattGen and VerifyPatt module inside LAxi2Reg transfers data with TOE10G-IP in both directions at the same time.

2.1 10G (10G/25G) EMAC and PCS/PMA

Both link layer and physical layer of 10G Ethernet are implemented by using Xilinx IP core. 10G (10G/25G) EMAC implements the link layer while 10G (10G/25G) Ethernet PCS/PMA implements the physical layer. Data path of EMAC is 64-bit AXI4 stream interface.

Tx interface timing diagram of Xilinx EMAC and TOE10G-IP are different. TOE10G-IP needs to send data of one packet continuously, but Xilinx EMAC does not support this feature. Xilinx EMAC may de-assert ready signal to receive data from Tx data interface between start-of-frame and end-of-frame. TenGMacIF needs to be designed to store transmitted data from TOE10G-IP when Xilinx EMAC is not ready to receive new data.

10G/25G EMAC does not include zero padding function. TenGMacIF for connecting with 10G/25G EMAC Subsystem needs to add zero padding when transmitted packet size from TOE10G-IP is less than 60 bytes. More details of 10G (10G/25G) EMAC and PCS/PMA are described in following link.

10G Ethernet MAC and PCS/PMA
https://www.xilinx.com/products/intellectual-property/do-di-10gemac.html
https://www.xilinx.com/products/intellectual-property/10gbase-r.html

10G/25G Ethernet Subsystem
https://www.xilinx.com/products/intellectual-property/ef-di-25gemac.html

2.2 TOE10G-IP

TOE10G-IP implements TCP/IP stack and offload engine. Control and status signals for user interface are accessed through register interface. Data interface is accessed through FIFO interface. More details are described in datasheet.
http://www.dgway.com/products/IP/TOE10G-IP/dg_toe10gip_data_sheet_xilinx_en.pdf

2.3 TenGMacIF

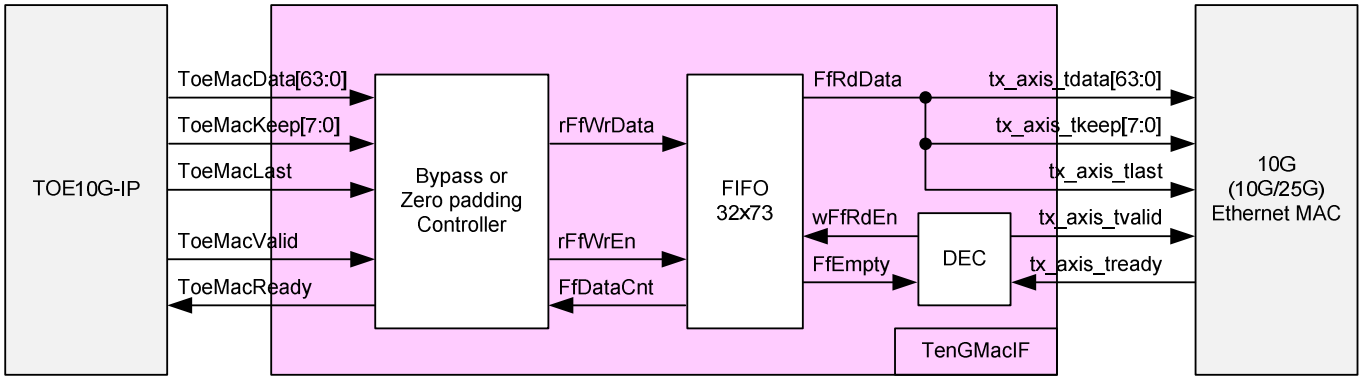

Figure 2-2 TenGMACIF Block Diagram

This module is designed to be adapter logic connecting between Tx interface of TOE10G-IP and Tx interface of 10G Ethernet MAC. ToeMacReady to TOE10G-IP must be always asserted to '1' during transferring one packet (between receiving 1st data and the last data) to send one packet continuously. But Xilinx 10G Ethernet MAC specification shows that timing diagram of tx_axis_tready may be de-asserted to '0' during transmitting the packet. TenGMacIF including small FIFO must be designed to store the data from TOE10G-IP when 10G Ethernet MAC is not ready to receive new data.

Otherwise, 10G/25G Ethernet Subsystem does not include zero padding function. So, TenGMacIF needs to add zero data to the packet when total packet size is too small (less than 60 bytes). Data flow control for normal packet which does not need to add zero padding is designed by following sequence.

ToeMacReady is asserted to '1' to receive start of new packet and FfDataCnt is less than 16. ToeMacReady is asserted to '1' until TOE10G-IP sends the last data by asserting ToeMacLast to '1'. After ToeMacLast is asserted to '1', ToeMacReady is de-asserted to '0' to block new packet from TOE10G-IP until FfDataCnt is less than 16. FIFO size in TenGMacIF is 32, so TenGMacIF supports tx_axis_tready de-asserting up to 16 clocks per one packet.

For 10G Ethernet system, input signals of Tx interface from TOE10G-IP are stored to FIFO when ToeMacValid='1' and ToeMacReady='1'. 73-bit data bus size is used to store 64-bit data, 8-bit keep signal, and 1-bit last flag. More details of bypass/zero padding controller is shown in Figure 2-3.

To forward data in FIFO to Ethernet MAC, simple logic is designed by monitoring FfEmpty. If FIFO is not empty (FfEmpty='0') and tx_axis_tready='1', 73-bit data will be read from FIFO and forwarded to Ethernet MAC.

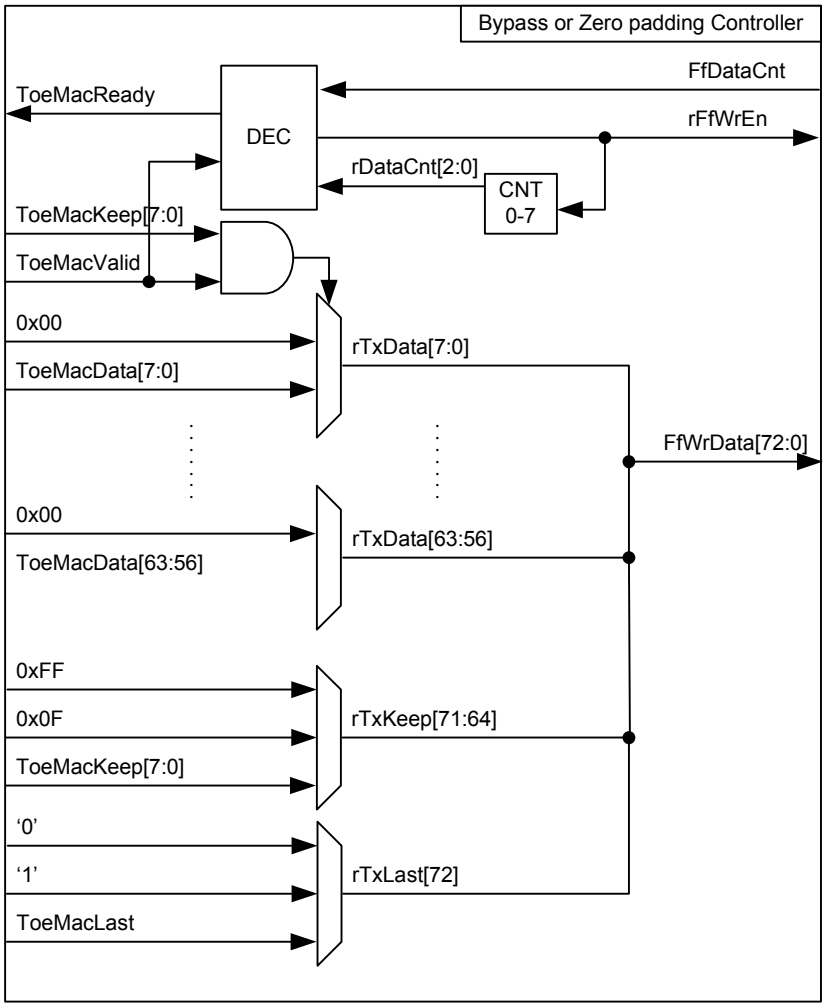

Figure 2-3 Data path of zero padding function

For 10G/25G Ethernet system, 3-bit data counter is additional designed to check the packet size. If packet size is less than 60 byte (data counter is less than 7), rFfWrEn will be asserted to fill zero padding data until data counter=7. rTxData is selected between data signal from TOE10G-IP or zero value for padding. When ToeMacKeep of which byte='1' and ToeMacValid='1' (real data is transferred from user), rTxData of that byte is fed from ToeMacData to stored to FIFO. If ToeMacValid='0' or ToeMacKeep of which byte='0', rTxData of that byte will be filled by zero.

rTxKeep is fixed to 0xFF in word 0-6. In word 7 (last word for minimum size), rTxKeep is fixed to 0x0F to pass 60 bytes for small packet or bypass from ToeMacKeep for the packet which is more than 60 byte. After that, rTxKeep is forwarded from TOE10G-IP directly until end of the packet.

Similar to rTxKeep, rTxLast can be set to three values, i.e. '0' (for word 0-6), '1' (for word 7 when packet size is not more than 64 bytes), and ToeMacLast (when packet size is more than 64 bytes).

2.4 LAxi2Reg

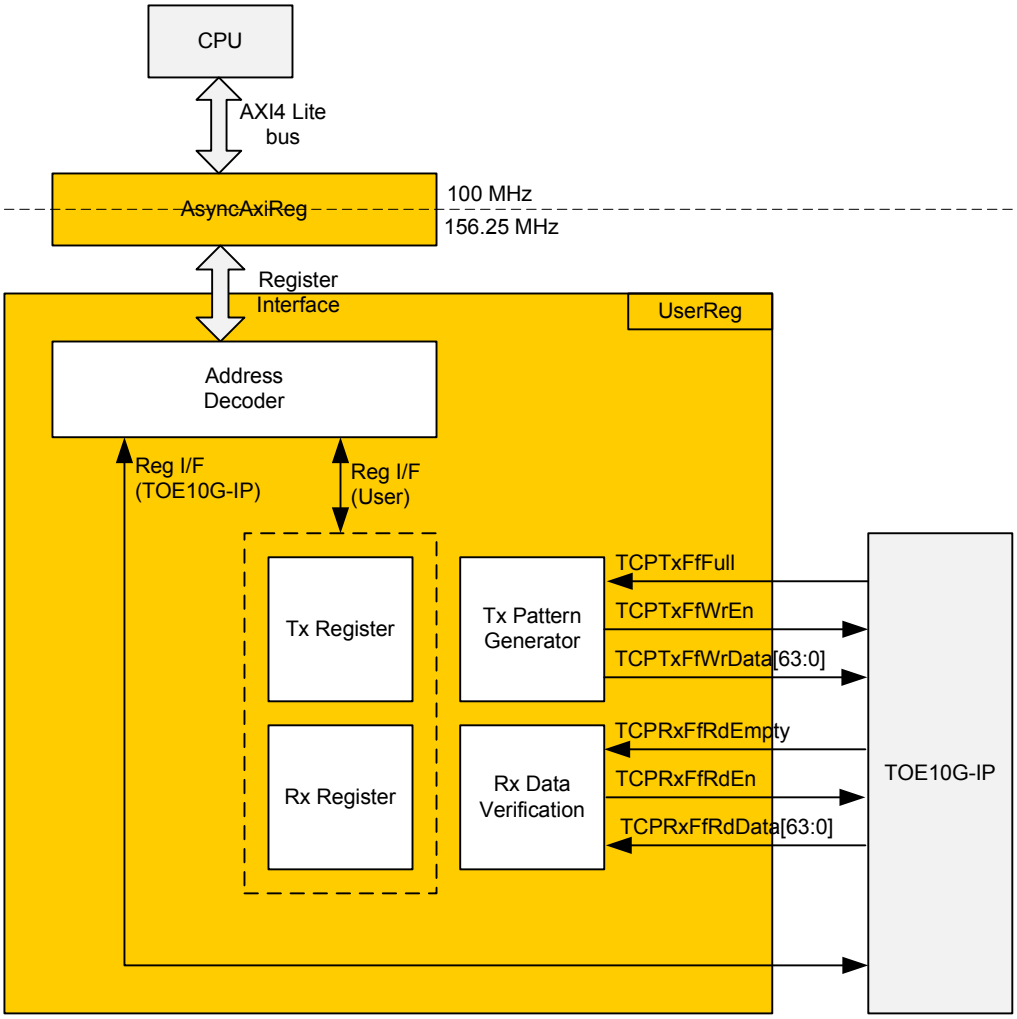

Figure 2-4 LAxi2Reg block diagram

This module consists of two submodules, i.e. AsyncAxiReg and UserReg.

AsyncAxiReg module is asynchronous circuit to support clock-crossing function. Also, it includes the adapter logic to convert AXI4 Lite bus which is CPU interface to be register interface.

As shown in Figure 2-4, UserReg module consists of four parts, i.e. Address decoder, Tx/Rx Register, Pattern Generator, and Rx Data Verification.

a) The address decoder is designed to split register map into two areas.
   0x0000 – 0x00FF is mapped to registers inside TOE10G-IP.
   0x1000 – 0x10FF is mapped to registers inside UserReg (to control Tx Pattern Generator and Rx Data Verification).
   The details of each register are shown in Table 2-1.

Table 2-1 Register map Definition

Address Wr/Rd	Register Name (Label in the "toe10cpu_demo.c")	Description
BA+0x00 – BA+0xFF: TOE10G-IP Register Area More details of each register are described in Table3 of TOE10G-IP datasheet.		
BA+0x00	TOE10_RST_REG	Mapped to RST register within TOE10G-IP
BA+0x04	TOE10_CMD_REG	Mapped to CMD register within TOE10G-IP
BA+0x08	TOE10_SML_REG	Mapped to SML register within TOE10G-IP
BA+0x0C	TOE10_SMH_REG	Mapped to SMH register within TOE10G-IP
BA+0x10	TOE10_DIP_REG	Mapped to DIP register within TOE10G-IP
BA+0x14	TOE10_SIP_REG	Mapped to SIP register within TOE10G-IP
BA+0x18	TOE10_DPN_REG	Mapped to DPN register within TOE10G-IP
BA+0x1C	TOE10_SPN_REG	Mapped to SPN register within TOE10G-IP
BA+0x20	TOE10_TDL_REG	Mapped to TDL register within TOE10G-IP
BA+0x24	TOE10_TMO_REG	Mapped to TMO register within TOE10G-IP
BA+0x28	TOE10_PKL_REG	Mapped to PKL register within TOE10G-IP
BA+0x2C	TOE10_PSH_REG	Mapped to PSH register within TOE10G-IP
BA+0x30	TOE10_WIN_REG	Mapped to WIN register within TOE10G-IP
BA+0x34	TOE10_ETL_REG	Mapped to ETL register within TOE10G-IP
BA+0x38	TOE10_SRV_REG	Mapped to SRV register within TOE10G-IP
BA+0x1000 – BA+0x10FF: UserReg control/status		
BA+0x1000 Wr/Rd	Total transmit length (USER_TXLEN_REG)	Wr [31:0] – Total transmitted size in Qword unit (64-bit). Valid from 1-0xFFFFFFFF. Rd [31:0] – Current transmitted size in Qword unit (64-bit). The value is cleared to 0 when USER_CMD_REG is written by user.
BA+0x1004 Wr/Rd	User Command (USER_CMD_REG)	Wr [0] – Start Transmitting. Set '1' to start transmitting. This bit is auto-cleared to '0' after end of total transfer. [1] – Data Verification enable ('0': Enable data verification, '1': Disable data verification) Rd [0] – Tx Busy. ('0': Idle, '1': Tx module is busy) [1] – Data verification error ('0': Normal, '1': Error) This bit is auto-cleared when user starts new operation or reset. [2] – Mapped to ConnOn signal of TOE10G-IP
BA+0x1008 Wr/Rd	User Reset (USER_RST_REG)	Wr [0] – Reset signal. Set '1' to reset the logic. This bit is auto-cleared to '0'. [8] – Set '1' to clear TimerInt latch value Rd [8] – Latch value of TimerInt output from IP ('0': Normal, '1': TimerInt='1' is detected) This flag can be cleared by system reset condition or setting USER_RST_REG[8]='1'.
BA+0x100C Rd	FIFO status (FIFO_STS_REG)	Rd [2:0]: Mapped to TCPRxFfLastRdCnt signal of TOE10G-IP [15:3]: Mapped to TCPRxFfRdCnt signal of TOE10G-IP [24]: Mapped to TCPTxFfFull signal of TOE10G-IP
BA+0x1010 Rd	Total Receive length (TRN_RXLEN_REG)	Rd [31:0] – Current received size in Qword unit (64-bit). The value is cleared to 0 when USER_CMD_REG is written by user.

b) Tx Registers are used to set total transfer size, enable data verification module, clear status flag, and reset internal logic.
Rx Registers are used to monitor current transfer size, error test pattern, interrupt flag, and fail flag.

c) Tx Pattern Generator generates 32-bit increment data to TCPTxFf interface. TCPTxFfFull is monitored during generating data. Data is transferred by asserting TPCTxFfWrEn=1' when TCPTxFfFull='0'. If TCPTxFfFull = '1', TCPTxFfWrEn will be de-asserted to '0' to pause data generation. TCPTxFfWrEn is cleared to '0' after total data are transferred completely, as shown in Figure 2-5.

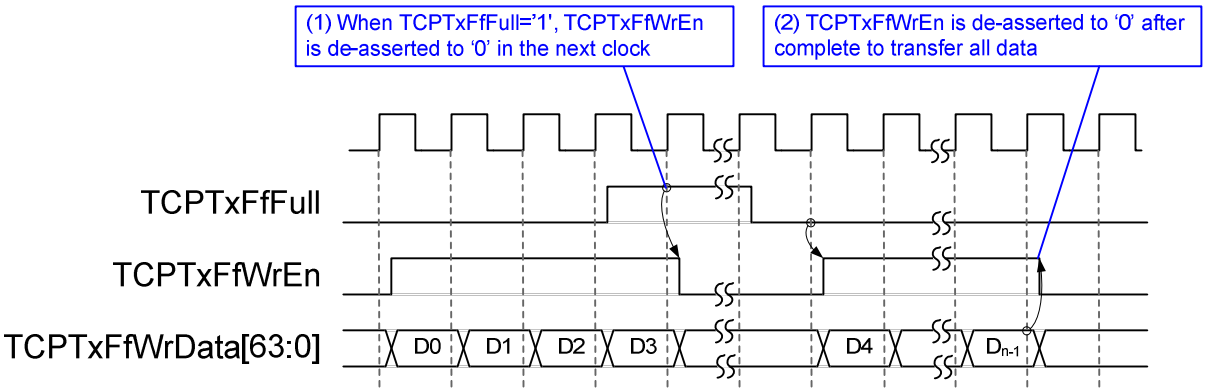

Figure 2-5 Tx Pattern Generator Timing diagram

d) Rx Data Verification monitors TCPRxFfRdEmpty status. TCPRxFfRdEn is designed by using NOT logic of TCPRxFfRdEmpty. TCPRxFfRdData is valid in the next clock after asserting TCPRxFfRdEn to '1'. Read data is compared to expected pattern when verification flag is enabled. Similar to Tx path, expected pattern is 32-bit increment pattern. Fail flag will be asserted to '1' if Read Data is not equal to expected pattern.

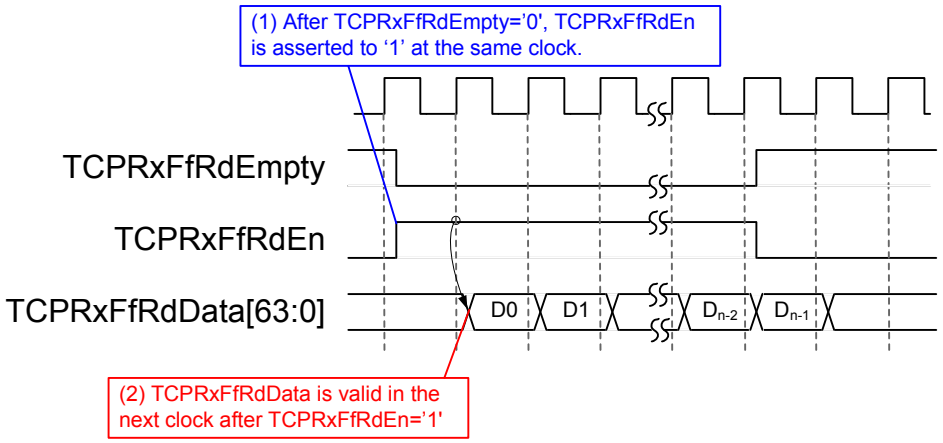

Figure 2-6 Rx Data Verification Timing diagram

3 CPU Firmware Sequence

Main menu in the firmware has four operations which are described in more details as follows.

3.1 Reset IP

This menu is used to change TOE10G-IP parameters such as IP address, source port number. After setting TOE10G-IP register, CPU resets the IP to re-initialize by using new parameters. CPU monitors busy flag to wait until the initialization is completed.

The sequence of reset sequence is shown as follows.

1) Display current parameter value to the console.
2) Receive input parameters from user and check input value whether it is in a valid range or not. If the input is invalid, the variable that input is invalid will not be changed.
3) Force reset to IP by setting TOE10_RST_REG[0]='1'.
4) Set all parameters to TOE10G-IP register such as TOE10_SML_REG, TOE10_DIP_REG.
5) De-assert IP reset by setting TOE10_RESET_REG[0]='0'.
6) Clear user logic status by sending reset to user logic (USER_RST_REG[0]='1').
7) Monitor IP busy flag (TOE10_CMD_REG[0]). Wait until busy flag is de-asserted to '0' to confirm that initialization sequence is completed.

3.2 Send data test

Two user inputs are requires to set total transmit length and packet size. The operation will be cancelled if the input is invalid. During the test, 32-bit increment data is generated from the logic and send to PC. Test application on PC verifies the received data. The operation is completed when total data are transferred from FPGA to PC completely.

The sequence of this test is shown as follows.

1) Receive transfer size and packet size from user and verify that the value is valid.
2) Set UserReg registers, i.e. transfer size (USER_TXLEN_REG), reset flag to clear initial value of test pattern (USER_RST_REG), and command register to start data pattern generator (USER_CMD_REG=0). After that, test pattern generator in UserReg transmits data to TOE10G-IP.
3) Display recommended parameter of test application running on PC by reading current parameters in the system.
4) Wait until connection is opened by PC. CPU monitors ConnOn status until it is equal to '1' (USER_CMD_REG[2]).
5) Set packet size to TOE10G-IP register (TOE10_PKL_REG) and calculate total loops from total transfer size. Maximum transfer size of each loop is 4 GB. The operation of each loop is shown as follows.
    a. Set transfer size of this loop to TOE10G-IP register (TOE10_TDL_REG). The set value is equal to remaining transfer size for the last loop or equal to 4 GB for other loops.
    b. Set send command to TOE10G-IP register (TOE10_CMD_REG=0).
    c. Wait until operation is completed by monitoring TOE10_CMD_REG[0]='0'. During waiting, CPU reads current transfer size from user logic (USER_TXLEN_REG and USER_RXLEN_REG) and displays on the console every second.
6) Set close connection command to TOE10G-IP register (TOE10_CMD_REG=3).
7) Calculate performance and show test result on the console.

3.3 Receive data test

User sets total received size and selects data verification mode (enable or disable). The operation will be cancelled if the input is invalid. During the test, 32-bit increment data is generated to verify the received data from PC when data verification mode is enabled.

The sequence of this test is shown as follows.

1) Receive total transfer size and data verification mode from user input. Verify that all inputs are valid.
2) Set UserReg registers, i.e. reset flag to clear initial value of test pattern (USER_RST_REG), and data verification mode (USER_CMD_REG[1]='0' or '1').
3) Display recommended parameter (same as Step 3 of Send data test).
4) Wait connection opened by PC (same as Step 4 of Send data test).
5) Wait until connection is closed by PC by monitoring Connon status (USER_CMD_REG[2]='0'). During waiting, CPU reads current transfer size from user logic (USER_TXLEN_REG and USER_RXLEN_REG) and displays on the console every second.
6) Check that total received length of user logic (USER_RXLEN_REG) is equal to set value from user and verification result is not failed (USER_CMD_REG[1]) = '0'). If the error is detected, error message will be displayed.
7) Calculate performance and show test result on the console.

3.4 Full duplex test

This menu is designed to run full duplex test by transferring data between FPGA and PC in both directions at the same time and same port number. Two inputs are received from user, i.e. packet size for FPGA sending logic and data verification mode for FPGA receiving logic. Transfer size for sending and receiving logic is fixed to maximum size (32 GB) which is equal to the size setting in the test application on PC. The test runs in forever loop until user cancels operation on PC (input Ctrl+C) and on Serial console (input any keys during operation).

The sequence of this test is shown as follows.

1) Receive packet size and data verification mode from user and verify that the value is valid.
2) Display recommended parameter of test application running on PC by reading current parameters in the system.
3) Set UserReg registers, i.e. transfer size (USER_TXLEN_REG), reset flag to clear initial value of test pattern (USER_RST_REG), and command register to start data pattern generator with data verification mode (USER_CMD_REG=1 or 3).
4) Wait connection opened by PC (same as Step 4 of Send data test).
5) Set TOE10G-IP registers, i.e. packet size (TOE10_PKL_REG=user input) and total length (TOE10_TDL_REG=4 GB). Run this step for 8 times to transfer 32 GB data. The operation of each loop is shown as follows.
    a. Set send command to TOE10G-IP register (TOE10_CMD_REG=0).
    b. Wait until send command is completed by monitoring TOE10_CMD_REG[0]='0'. During waiting, CPU reads current transfer size from user logic (USER_TXLEN_REG and USER_RXLEN_REG) and displays on the console every second.
6) Wait until connection is closed by PC by monitoring Connon status (USER_CMD_REG[2]='0').
7) Check received result and error (same as Step 6 of Receive data test).
8) Calculate performance and show test result on the console. Go back to step 3 to run the test in forever loop.

4 Test Software Sequence

4.1 "tcpdatatest" for half duplex test

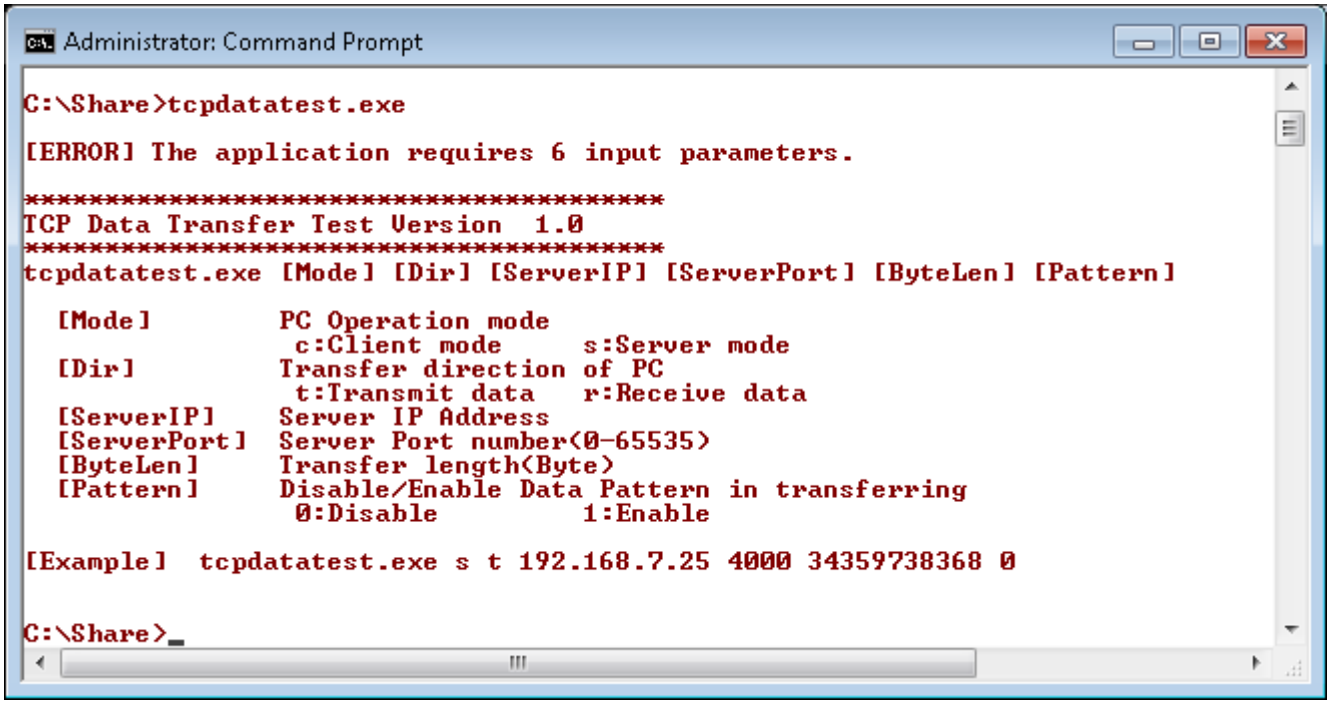

Figure 4-1 "tcpdatatest" application usage

"tcpdatatest" is designed to run on PC for sending/receiving TCP data through Ethernet for both Server and Client mode. PC of this demo runs in Client mode only. User can input parameter to select transfer direction and the mode. Six parameters are required, i.e.

1) Mode: c – when PC runs in Client mode and FPGA runs in Server mode
2) Dir: t – transmit mode (PC sends data to FPGA)
   r – receive mode (PC receives data from FPGA)
3) ServerIP: IP address of FPGA when PC runs in client mode (default is 192.168.7.42)
4) ServerPort: Port number of FPGA when PC runs in client mode (default is 4000)
5) ByteLen: Total transfer size in byte unit. This input is used in transmit mode only and is ignored in receive mode. In receive mode, application is closed when connection is destroyed. ByteLen in transmit mode must be equal to transfer size setting in Serial console (under received data test submenu).
6) Pattern:
   0 – Generate dummy data in transmit mode or disable data verification in receive mode.
   1 – Generate increment data in transmit mode or enable data verification in receive mode.

Transmit data mode

Following is the sequence when test application runs in transmit mode.

1) Allocate 1 MB memory to be send buffer.
2) Create socket and set properties of send buffer.
3) Create new connection to server by using IP address and port number from user.
4) Generate increment test pattern to send buffer when test pattern is enabled. Skip this step if dummy pattern is selected.
5) Send data out and decrease remaining transfer size.
6) Print total transfer size every second.
7) Run step 4) – 6) in the loop until remaining transfer size is 0.
8) Close socket and print total size and performance.

Receive data mode

Following is the sequence when test application runs in receive mode.

1) Allocate 1 MB memory to be received buffer.
2) Create socket and set properties of received buffer.
3) Same step as step3) in Transmit data mode.
4) Read data from received buffer and increase total received data size.
5) If verification is enabled, data will be verified with increment pattern and error message will be printed out when data is not correct. Skip this step if data verification is disabled.
6) Print total transfer size every second.
7) Run step 4) – 6) in the loop until connection status is closed.
8) Close socket and print total size and performance.

4.2 "tcp_client_txrx_10G" for full duplex test

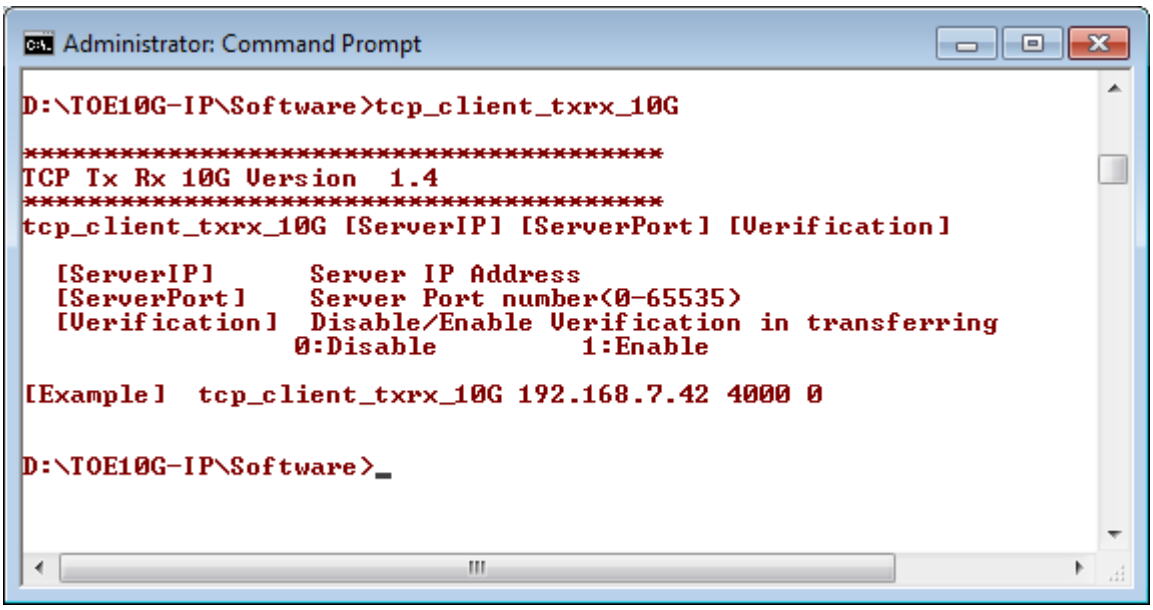

Figure 4-2 "tcp_client_txrx_10G" application usage

"tcp_client_txrx_10G" application is designed to run on PC for sending and receiving TCP data through Ethernet by using same port number at the same time. The application is run in Client mode, so user needs to input server parameters. As shown in Figure 4-2, there are three parameters to run the application, i.e.

1) ServerIP: IP address of FPGA (default is 192.168.7.42)
2) ServerPort: Port number of FPGA (default is 4000)
3) Verification:
   0 – Generate dummy data for sending function and disable data verification for receiving function. This mode is used to check the best performance of full-duplex transfer.
   1 – Generate increment data for sending function and enable data verification for receiving function.

The sequence of test application is shown as follows.
(1) Allocate 60 KB memory for send and receive buffer.
(2) Create socket and set properties.
(3) Create new connection by using IP address and port number from user.
(4) Generate increment test pattern to send buffer when test pattern is enabled. Skip this step if dummy pattern is selected.
(5) Send data out and decrease remaining transfer size.
(6) Read data from received buffer and increase total received data size.
(7) If verification is enabled, data will be verified by increment pattern and error message will be printed out when data is not correct. Skip this step if data verification is disabled.
(8) Print total transfer size every second
(9) Run step 5) – 8) until total sending/receiving data are equal to 32 GB.
(10) Print total size and performance and close socket.
(11) Sleep for 1 millisecond to wait the hardware complete current test loop.
(12) Run step 3) – 11) in forever loop. If verification is fail, the application will quit.

5 Revision History

Revision	Date	Description
1.0	22-Jan-18	Initial version release