TOE10G-IP with CPU reference design

Rev1.2 23-Aug-19

1 Introduction

TCP/IP is the core protocol 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. As shown in Figure 1-1, five layers are displayed for simply matching with the hardware implementation on FPGA. Network Access Layer is split into Link and Physical Layer.

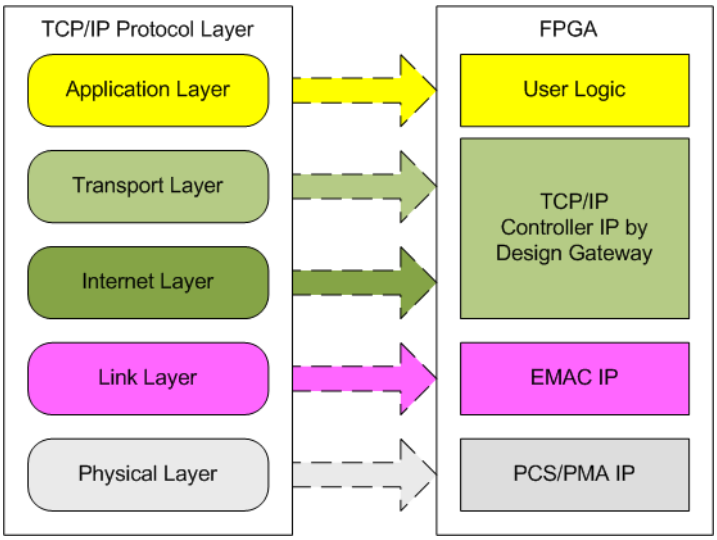

Figure 1-1 TCP/IP Protocol Layer

TOE10G IP implements Transport and Internet layer of TCP/IP Protocol. Building Ethernet packet from the user data which is TCP data to EMAC, TOE10G IP split TCP data from the user to small packet and then inserts TCP/IP header. On the other hand, the received Ethernet packet from EMAC is extracted by TOE10G IP for verifying the header and the packet. Only the extracted TCP data is forwarded to the user logic. The packet will be rejected if TCP/IP header in the packet is invalid.

The lower layer protocols are implemented by EMAC IP and PCS/PMA IP. PCS/PMA IP is provided by Xilinx while DG EMAC IP or Xilinx EMAC IP can be selected to run EMAC function.

The reference design provides the evaluation system which includes simple user logic to send and receive data by using TOE10G IP. TOE10G IP is designed to transfer data with PC or another TOE10G IP running on another FPGA board. To run with PC, the test application is called on PC to send and verify TCP data from Ethernet connection at very high speed rate. Two test applications are specially designed for the demo, i.e. “tcpdatatest” for half-duplex test (send or receive data test) and “tcp_client_txrx_40G” for full-duplex test (send and receive data at the same time).

To use UART to interface with the user for controlling the test parameters and the operation of TOE10G IP, the CPU system is included in the demo. Using Serial console is easy for the user to set and monitor the test parameters and the system status. The firmware on CPU is built by using bare-metal OS. More details of the demo are described as follows.

2 Hardware overview

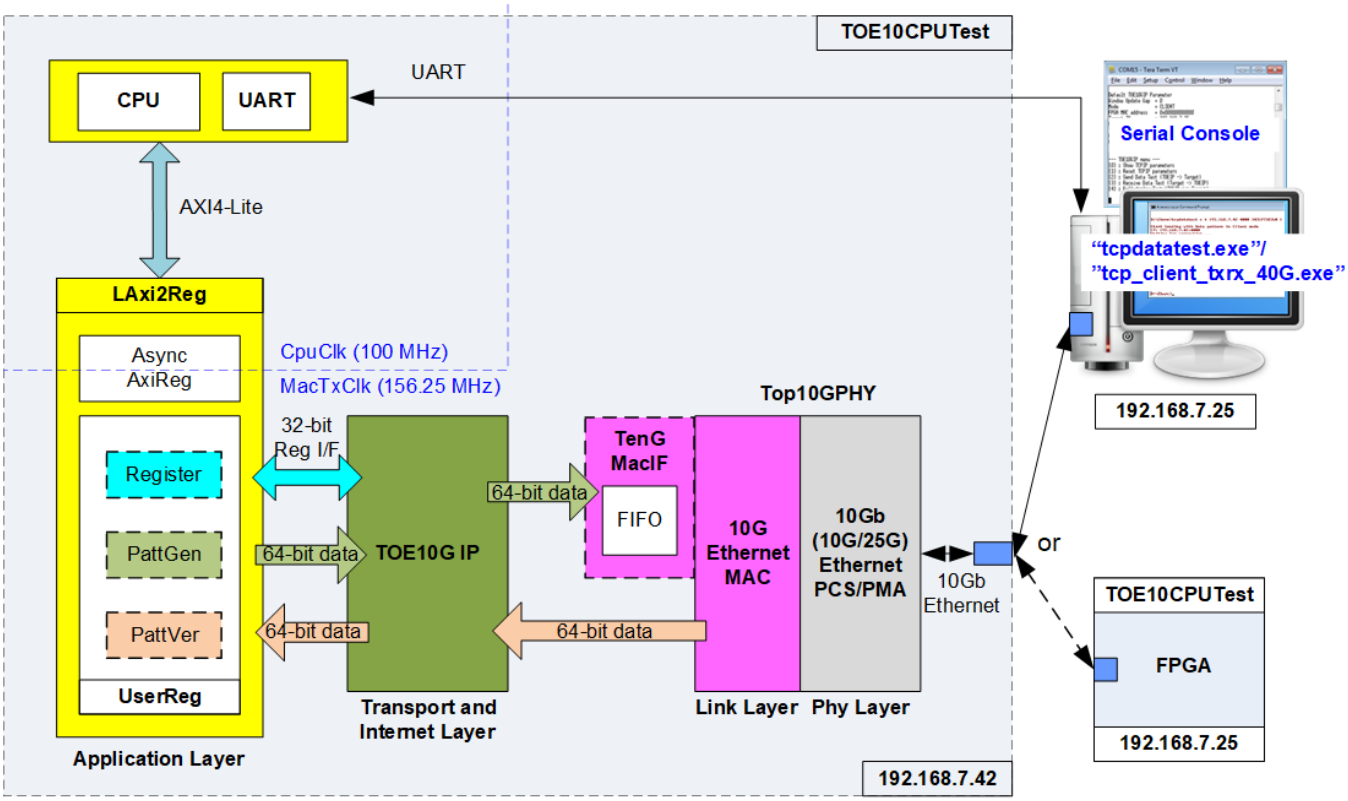

Figure 2-1 Demo Block Diagram

In test environment, two devices are used for 10Gb Ethernet transferring. First device runs in client mode and another device runs in server mode. In the demo, the client device is always TOE10G IP on FPGA board while the server device could be TOE10G IP on FPGA board or PC, as shown in Figure 2-1. The test applications (tcpdatatest and tcp_client_txrx_40G) are run on PC to transfer data with TOE10G IP within FPGA.

In FPGA logics, TOE10G IP connects to 10G Ethernet MAC and 10G Ethernet PCS/PMA to complete all TCP/IP layer implementation. For Ultrascale+ device, it uses 10G/25G Ethernet Subsystem instead of 10G Ethernet. User interface of TOE10G IP connects to UserReg within AsyncAXIReg for both data interface and register interface. Register files of UserReg are split into two regions, i.e. TOE10G IP region and internal test logic region (PattGen and PattVer). Register files of UserReg are controlled by the firmware operating on the CPU through AXI4-Lite interface. User controls the operation of TOE10G IP, PattGen, and PattVer by modifying the firmware run on the CPU.

Otherwise, the special hardwares are designed in the reference design which uses Xilinx 10G EMAC IP. TenGMacIF is designed to be the data buffer between TOE10G IP and Xilinx 10G EMAC IP when Xilinx 10G EMAC IP is not ready to receive the packet during one packet transmission.

Two clock domains are applied in the test design, i.e. CpuClk which is the independent clock for running the CPU system and MacTxClk which is the clock output from 10G Ethernet PCS/PMA. So, AsyncAXIReg is designed to support asynchronous transfer between CpuClk and MacTxClk. More details of each module inside the TOE10GCPUTest are described as follows.

2.1 10G (10G/25G) PCS/PMA (BASE-R)

10G PCS/PMA implements physical layer of 10G Ethernet. It connects to the external hardware which is 10G BASE-R SFP+ module. The user interface is 64-bit XGMII interface running at 156.25 MHz which is designed to connect with 10G Ethernet MAC. This IP core is provided by Xilinx without the charge. More details of the core are described in the following link.

10G (10G/25G) Ethernet PCS/PMA (BASE-R)
https://www.xilinx.com/products/intellectual-property/10gbase-r.html
https://www.xilinx.com/products/intellectual-property/ef-di-25gemac.html

2.2 10G EMAC

10G EMAC connects between TOE10G IP and 10G PCS/PMA module. The interface with TOE10G IP is 64-bit AXI4 stream while the interface with 10G PCS/PMA module is 64-bit XGMII interface at 156.25 MHz. When using DG 10G EMAC IP, TOE10G IP can connect to EMAC IP directly. But small buffer must be used to interface between TOE10G IP and Xilinx 10G EMAC IP. More details of each EMAC are described in the following link.

DG 10G EMAC IP
https://dgway.com/products/IP/10GEMAC-IP/dg_tengemacip_data_sheet_xilinx_en.pdf

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

Tx interface timing diagram of Xilinx 10G 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 during packet transferring (between start-of-frame and end-of-frame). TenGMacIF needs to be designed to store transmitted data from TOE10G-IP when Xilinx 10G EMAC is not ready to receive new data. The example interface module including the small buffer for connecting between TOE10G IP and Xilinx 10G EMAC IP is described as follows.

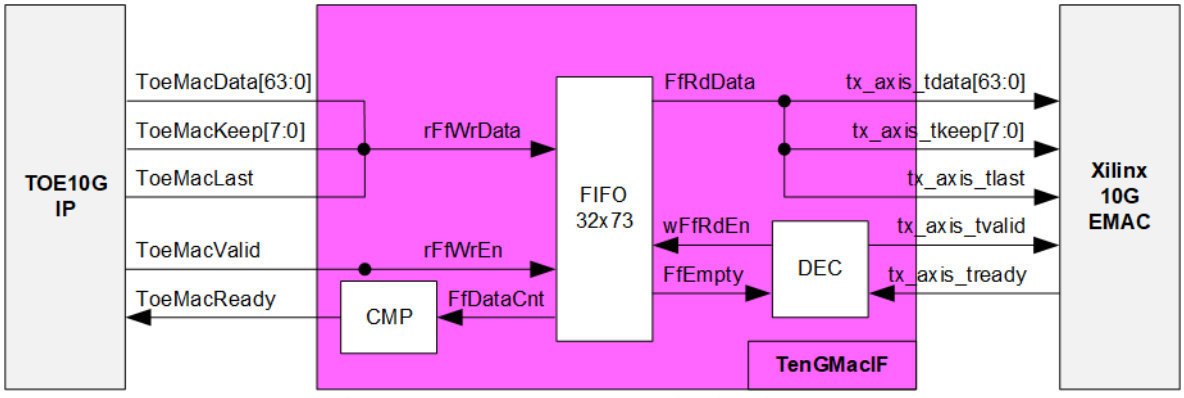

Figure 2-2 TenGMacIF Block Diagram

This module is designed to be adapter logic connecting between Tx interface of TOE10G IP and Tx interface of Xilinx 10G EMAC. ToeMacReady output to TOE10G IP must be always asserted to '1' between start-of-frame and end-of-frame, but tx_axis_tready of Xilinx EMAC may be de-asserted to '0' during transferring start-of-frame and end-of-frame. The small FIFO (FIFO32x73) is designed to store the data from TOE10G IP when Xilinx EMAC is not ready to receive new data. The logic uses 32-depth FIFO to support tx_axis_tready de-asserted in one packet less than or equal to 16 clock cycles. Timing diagram to store data from TOE10G IP to FIFO is shown in Figure 2-3.

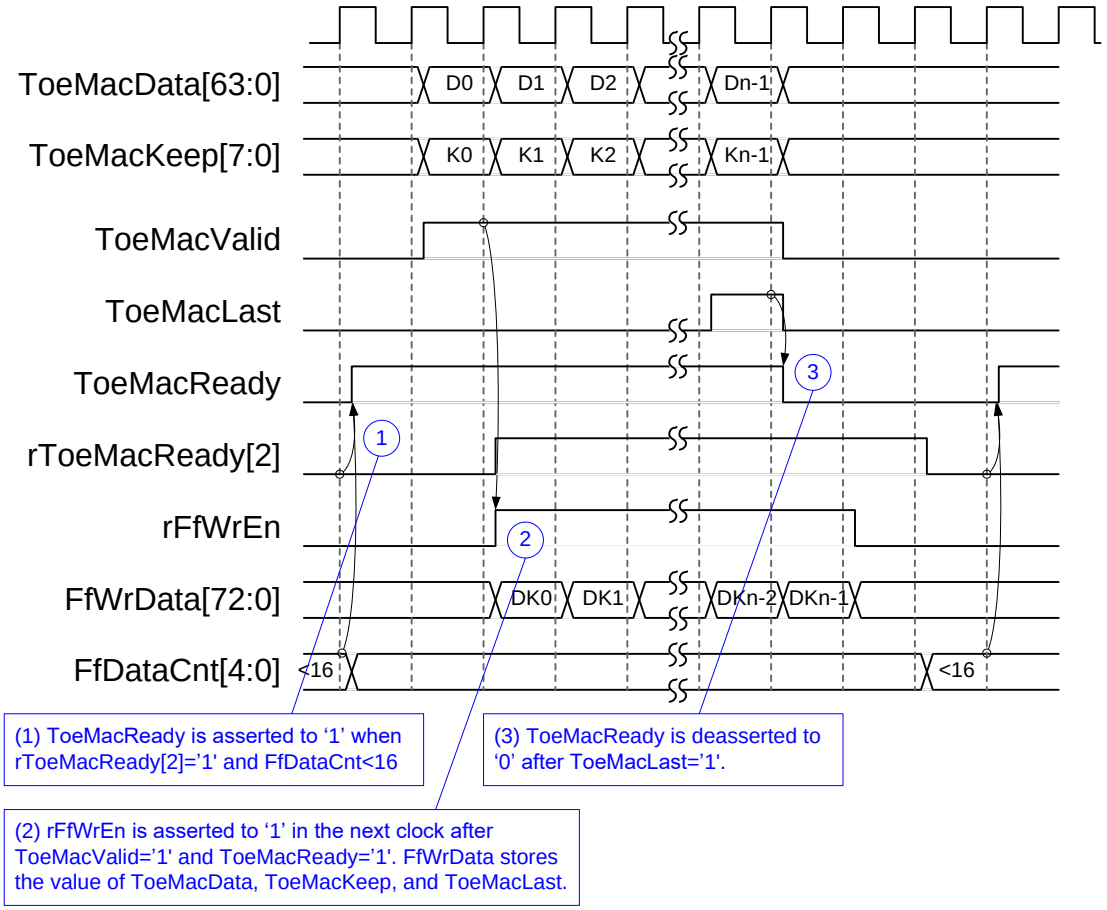

Figure 2-3 Write FIFO timing diagram of TenGMacIF

(1) When free space in FIFO is much enough (FfDataCnt is less than 16) and the previous transmit packet is completed more than two clock cycles (monitoring that rToeMacReady[2] which is 2-clock cycle delayed from ToeMacReady is de-asserted to '0'), ToeMacReady is asserted to '1' for receiving the new packet.

(2) The new packet from TOE10G IP is transferred to TenGMacIF continuously because ToeMacReady is always asserted to '1'. rFfWrEn is asserted to '1' to store all packets until end of frame. 73-bit data is designed to store 64-bit data (ToeMacData), 8-bit byte enable signal (ToeMacKeep), and last flag (ToeMacLast) into the FIFO.

(3) ToeMacReady is de-asserted to '0' after the last data is received (ToeMacLast is asserted to '1'). So, new packet from TOE10G IP is not transferred until ToeMacReady is re-asserted to '1'. In this module, ToeMacReady is de-asserted to '0' at least 3 clock cycles by monitoring rToeMacReady[2] status, as described in step (1).

Timing diagram to forward data from FIFO to Xilinx 10G EMAC is shown in Figure 2-4.

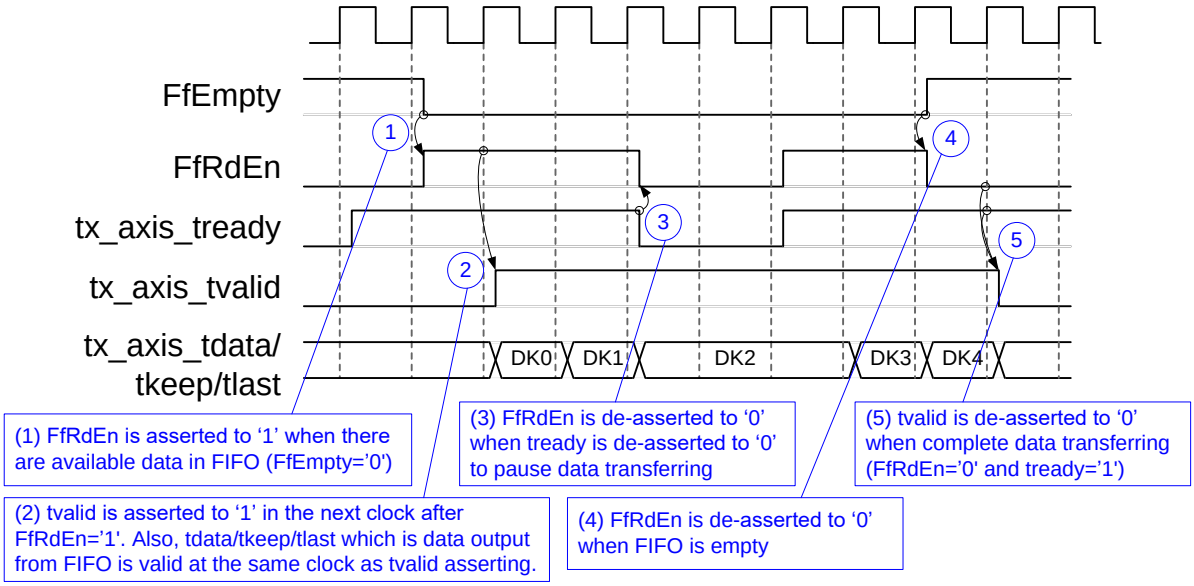

Figure 2-4 Read FIFO timing diagram of TenGMacIF

(1) When data is available in FIFO (FfEmpty='0'), FfRdEn is asserted to '1' to transfer data from FIFO to Ethernet MAC.
(2) In the next clock cycle, tx_axis_tvalid is asserted to '1' with the valid data for transferring to EMAC.
(3) When EMAC is not ready to receive data (tx_axis_tready='0'), FfRdEn is de-asserted to '0' to pause data transmission.
(4) When FIFO is empty, FfRdEn is de-asserted to '0'.
(5) After that, tx_axis_tvalid is de-asserted to '0' to complete data transferring.

Note: Because data of one packet from TOE10G IP is transferred continuously, FfEmpty is not asserted to '1' before the end of packet.

2.3 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

2.4 CPU and Peripherals

32-bit AXI4-Lite bus is applied to be the bus interface for CPU accessing the peripherals such as Timer and UART. To control and monitor the test system, the control and status signals are connected to register for CPU access as a peripheral through 32-bit AXI4-Lite bus. CPU assigns the different base address and the address range for each peripheral.

In the reference design, the CPU system is built with one additional peripheral to access the test logic. The base address and the range for accessing the test logic are defined in the CPU system. So, the hardware logic must be designed to support AXI4-Lite bus standard for writing and reading the register. LAxi2Reg module is designed to connect the CPU system as shown in Figure 2-5.

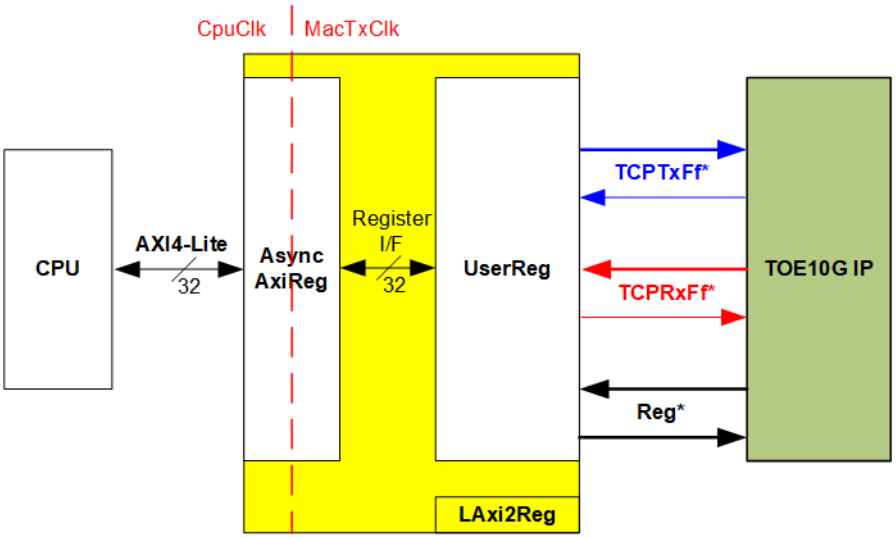

Figure 2-5 LAxi2Reg block diagram

LAxi2Reg consists of AsyncAxiReg and UserReg. AsyncAxiReg is designed to convert the AXI4-Lite signals to be the simple register interface which is 32-bit data bus size (similar to AXI4-Lite data bus size). Additionally, AsyncAxiReg includes asynchronous logic to support clock crossing between CpuClk domain and MacTxClk domain.

UserReg includes the register file of the parameters and the status signals. Also, data interface and control interface of TOE10G IP are connected to UserReg. More details of AsyncAxiReg and UserReg are described as follows.

2.4.1 AsyncAxiReg

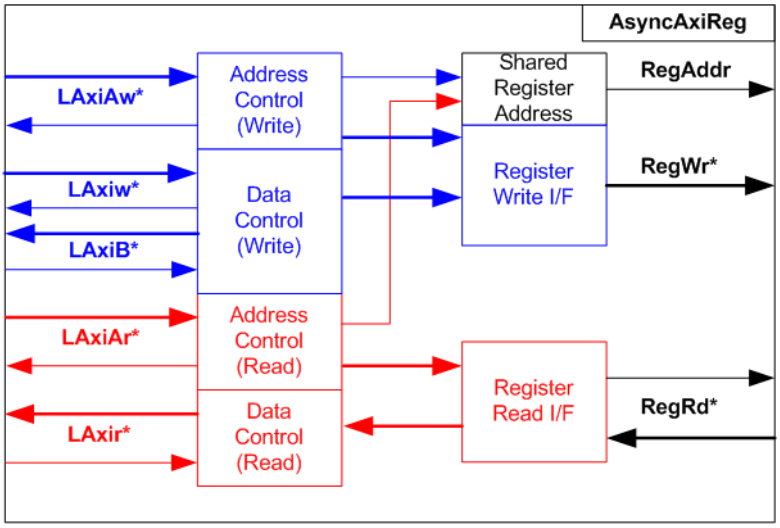

Figure 2-6 AsyncAxiReg Interface

The signal on AXI4-Lite bus interface can be split into five groups, i.e. LAxiAw* (Write address channel), LAxiw* (Write data channel), LAxiB* (Write response channel), LAxiAr* (Read address channel), and LAxir* (Read data channel). More details to build custom logic for AXI4-Lite bus is described in following document.
https://forums.xilinx.com/xlnx/attachments/xlnx/NewUser/34911/1/designing_a_custom_axi_slave_rev1.pdf

According to AXI4-Lite standard, the write channel and the read channel are operated independently. Also, the control and data interface of each channel are run in parallel. So, the logic inside AsyncAxiReg to interface with AXI4-Lite bus is split into four groups, i.e. Write control logic, Write data logic, Read control logic, and Read data logic as shown in the left side of Figure 2-6. Write control I/F and Write data I/F of AXI4-Lite bus are latched and transferred to be Write register interface with the shared register address. For read access, Read control I/F and Read data I/F of AXI4-Lite bus are latched and transferred to be Read register interface with the shared register address.

The simple register interface is designed to compatible to general RAM interface for write transaction. The read transaction of the register interface is slightly modified from RAM interface by adding RdReq signal. The address of register interface is shared for write and read transaction. So, user cannot write and read the register at the same time. The timing diagram of the register interface is shown in Figure 2-7.

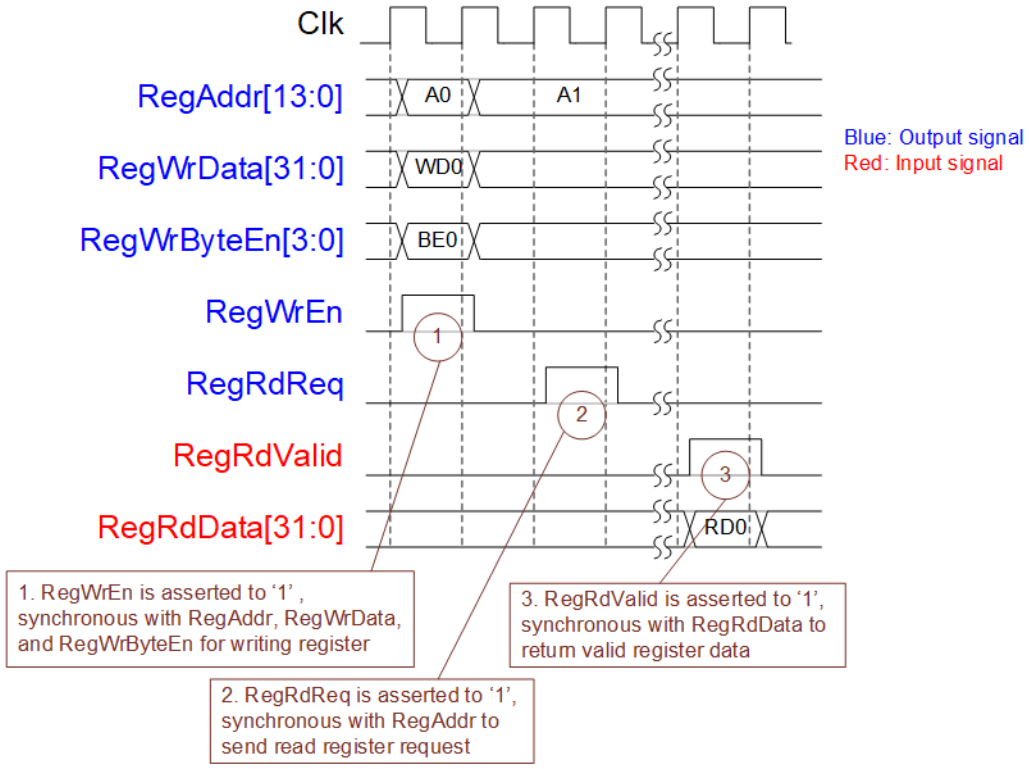

Figure 2-7 Register interface timing diagram

1) To write register, the timing diagram is same as general RAM interface. RegWrEn is asserted to '1' with the valid signal of RegAddr (Register address in 32-bit unit), RegWrData (write data of the register), and RegWrByteEn (the write byte enable). Byte enable has four bit to be the byte data valid, i.e. bit[0] for RegWrData[7:0], bit[1] for RegWrData[15:8], and so on.
2) To read register, AsyncAxiReg asserts RegRdReq to '1' with the valid value of RegAddr. 32-bit data must be returned after receiving the read request. The slave must monitor RegRdReq signal to start the read transaction.
3) The read data is returned on RegRdData bus by the slave with asserting RegRdValid to '1'. After that, AsyncAxiReg forwards the read value to LAxir* interface.

2.4.2 UserReg

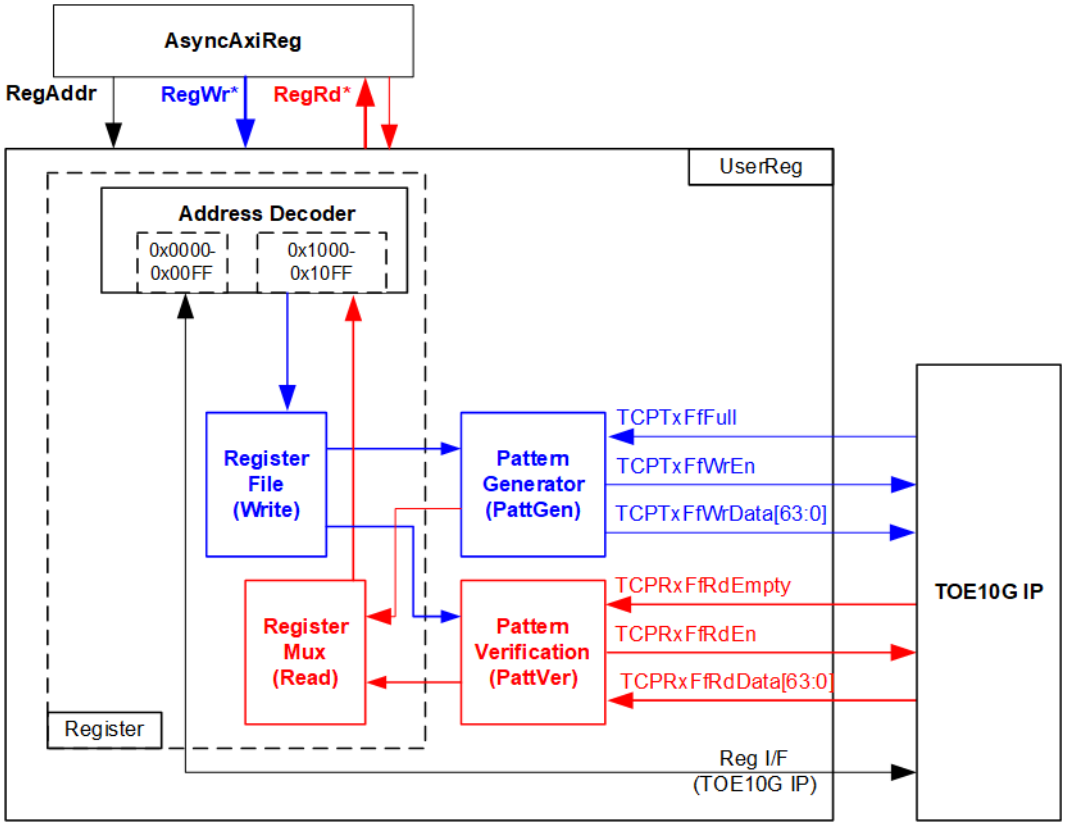

Figure 2-8 UserReg block diagram

The logic inside UserReg has three operations, i.e. Register, Pattern generator (PattGen), and Pattern verification (PattVer). Register block decodes the address requested from AsyncAxiReg and then selects the active register for write or read transaction. Pattern generator block is designed to send 64-bit test data to TOE10G IP following FIFO interface standard. Pattern verification block is designed to read and verify 64-bit data from TOE10G IP following FIFO interface standard. More details of each block are described as follows.

Register Block

The address range to map to UserReg is split into two areas, i.e. TOE10G IP register (0x0000-0x00FF) and UserReg register (0x1000-0x10FF).

Address decoder decodes the upper bit of RegAddr for selecting the active hardware. The register file inside UserReg is 32-bit bus size, so write byte enable (RegWrByteEn) is not used. To set the parameters in the hardware, the CPU must use 32-bit pointer to force 32-bit valid value of the write data.

To read register, one multiplexer is designed to select the read data within each address area. The lower bit of RegAddr is applied in each Register area to select the data. Next, the address decoder uses the upper bit to select the read data from each area for returning to CPU. Totally, the latency of read data is equal to one clock cycle, so RegRdValid is created by RegRdValid with asserting one D Flip-flop. More details of the address mapping within UserReg module are shown in Table 2-1

Table 2-1 Register map Definition

Address Wr/Rd	Register Name (Label in the “toe10gtest.c”)	Description
BA+0x0000 – BA+0x00FF: TOE10G-IP Register Area More details of each register are described in 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 ‘0’ to start transmitting. [1] – Data Verification enable (‘0’: Disable data verification, ‘1’: Enable 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 latched value Rd [8] – Latched 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’. [16] – Ethernet linkup status from Ethernet MAC (‘0’: Not linkup, ‘1’: Linkup)
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.

Pattern Generator

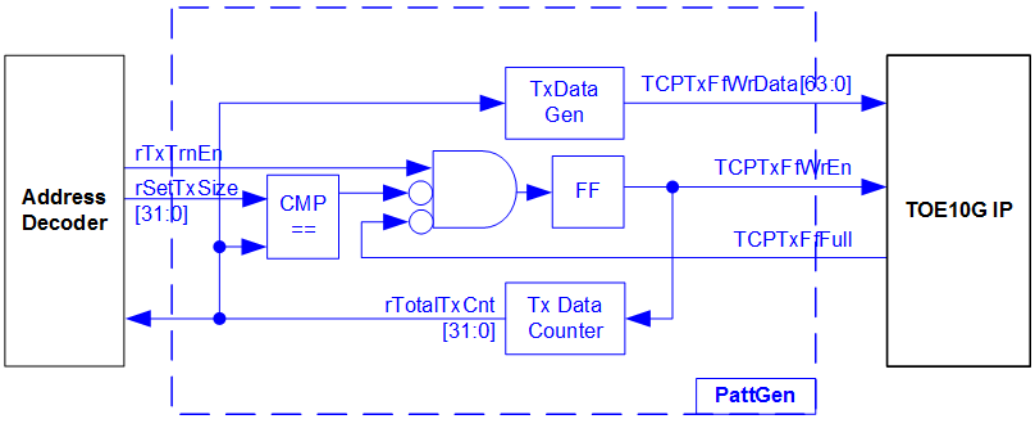

Figure 2-9 PattGen block

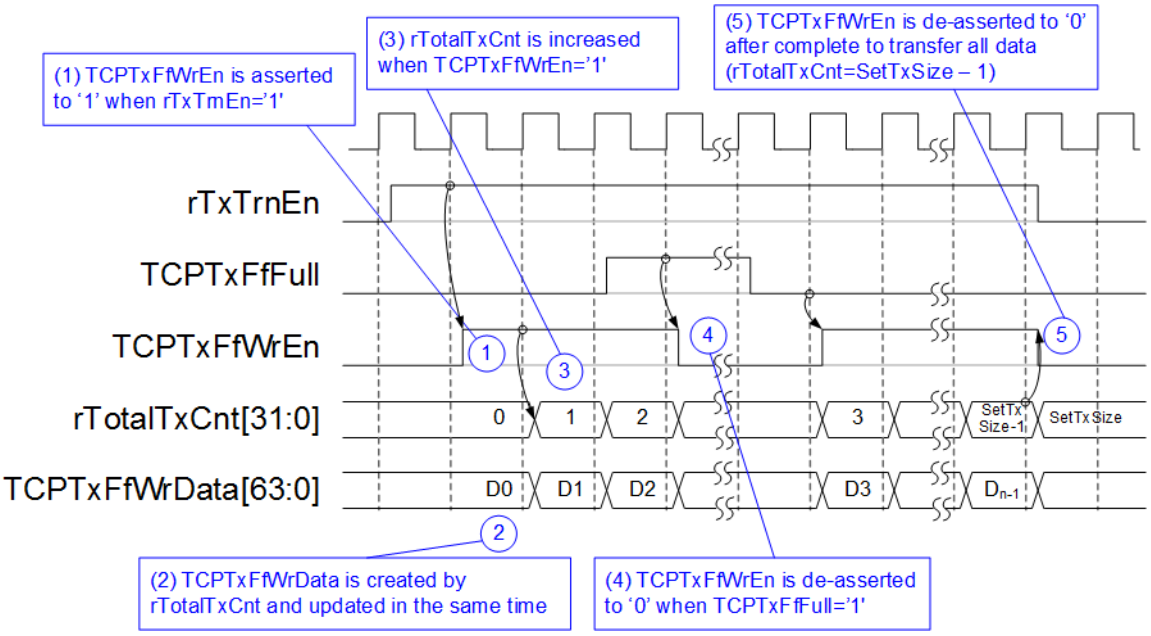

Figure 2-10 PattGen Timing diagram

PattGen is designed to generate test data to TOE10G IP. rTxTrnEn is asserted to '1' when USER_CMD_REG[0] is set to '1'. When rTxTrnEn is '1', TCPTxFfWrEn is controlled by TCPTxFfFull. TCPTxFfWrEn is de-asserted to '0' when TCPTxFfFull is '1'. rTotalTxCnt is the data counter to check total data sent to TOE10G IP. rTotalTxCnt is also used to generate 32-bit incremental data to TCPTxFfWrData signal. rTxTrnEn is de-asserted to '0' when finishing transferring total data (total data is set by rSetTxSize).

Pattern Verification

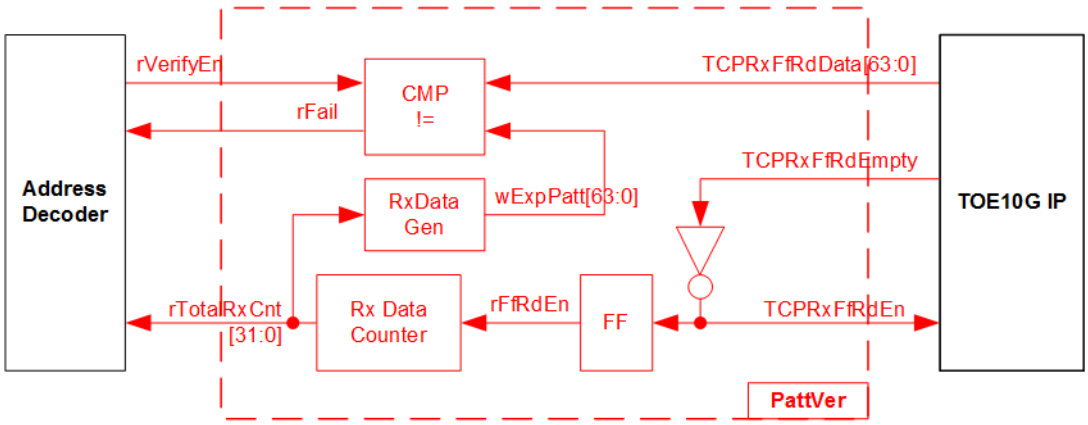

Figure 2-11 PattVer block

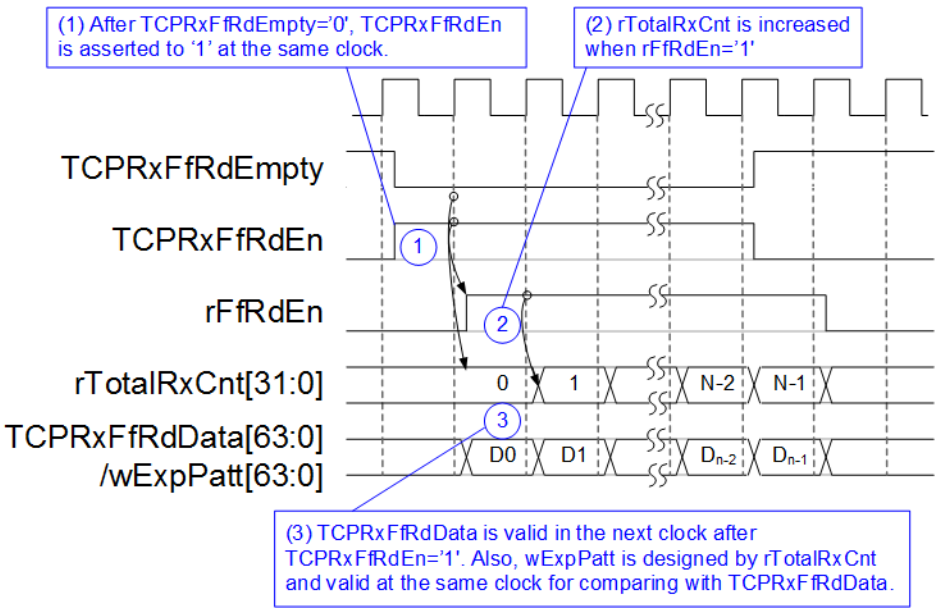

Figure 2-12 PattVer Timing diagram

PattVer is designed to read test data from TOE10G IP with or without data verification, depending on rVerifyEn flag. When rVerifyEn is set to '1', data comparison is enabled to compare read data (TCPRxFfRdData) to the expected pattern (wExpPatt). If data verification is failed, rFail will be asserted to '1'. TCPRxFfRdEn is designed by using NOT logic of TCPRxFfRdEmpty. TCPRxFfRdData is valid for data comparison in the next clock after asserting TCPRxFfRdEn to '1'. rRxFfRdEn which is one clock latency of TCPRxFfRdEn is applied to be counter enable of rTotalRxCnt to count total transfer size. rTotalRxCnt is used to generate wExpPatt.

3 CPU Firmware Sequence (FPGA)

After FPGA boot-up, 10G Ethernet link up status (USER_RST_REG[16]) is polling. The CPU waits until link up is found. Next, welcome message is displayed and user selects the operation mode of TOE10G IP to be client or server mode.

To initialize as client mode, TOE10G IP sends ARP request to get the MAC address from the destination device. For server mode, TOE10G IP waits ARP request to decode MAC address and returns ARP reply to complete initialization process.

If test environment uses two FPGA boards, the operation mode on two TOE10G IPs must be different (one is client and another is server). To run with PC, it is recommended to set FPGA to as client mode. When PC receives ARP request, PC always returns ARP reply. It is not easy to force PC sending ARP request to FPGA.

The software has two default parameters for each operation mode. Figure 3-1 shows the example of the initialization sequence after system boot-up.

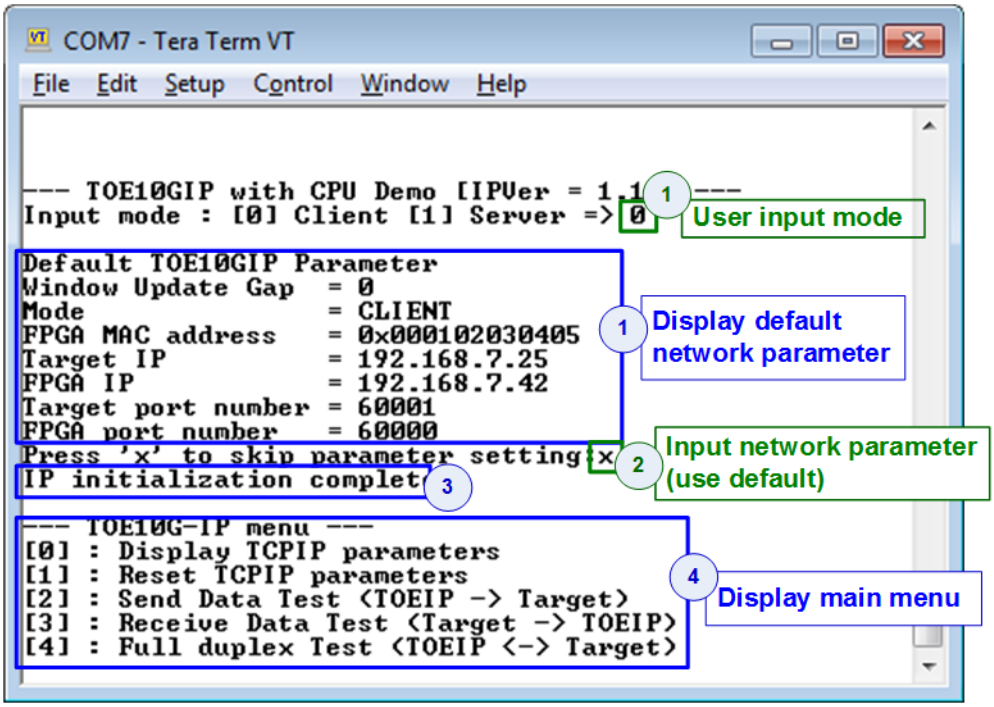

Figure 3-1 Example of initialization sequence in client mode on Serial Console

There are four steps to complete initialization sequence as follows.

1) CPU receives the operation mode from user and displays default parameters on the console.
2) User inputs 'x' to complete initialization sequence by using default parameters. Other keys are set for changing some parameters. More details for changing some parameters are described in Reset IP menu (topic 3.2).
3) CPU waits until TOE10G IP finishing initialization sequence (TOE10_CMD_REG[0]='0').
4) Main menu is displayed. There are five test operations for user selection. More details of each menu are described as follows.

3.1 Show parameters

This menu is used to show current parameters of TOE10G-IP, i.e. operation mode, Windows update threshold, source MAC address, destination IP address, source IP address, destination port, and source port. The sequence of display parameters is as follows.

1) Read all network parameters from each variable in firmware.
2) Print out each variable.

3.2 Reset IP

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

1) Display current parameter value to the console.
2) Receive new input parameters from user and check input value whether valid or not. If the input is invalid, the old value will be used instead.
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 and TOE10_DIP_REG.
5) De-assert IP reset by setting TOE10_RESET_REG[0]='0'.
6) Clear PattGen and PattVer logic by sending reset to user logic (USER_RST_REG[0]='1').
7) Monitor IP busy flag (TOE10_CMD_REG[0]) until the initialization sequence is completed (busy flag is de-asserted to '0').

3.3 Send data test

Three user inputs are received from user, i.e. total transmit length, packet size, and connection mode (active open for client operation or passive open for server operation). The operation will be cancelled if some inputs are invalid. During the test, 32-bit incremental data is generated from the logic and sent to PC or FPGA. Data is verified by Test application on PC (in case of PC <-> FPGA) or verification module in FPGA (in case of FPGA <-> FPGA). The operation is finished when total data are transferred from FPGA to PC or FPGA completely. The sequence of this test is as follows.

1) Receive transfer size, packet size, and connection mode 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[0]='1'), and command register to start data pattern generator (USER_CMD_REG=0). After that, test pattern generator in UserReg sends data to TOE10G-IP.
3) Display recommended parameter of test application running on PC from the current system parameters.
4) Open connection following connection mode value.
   a. For active open, CPU sets TOE10_CMD_REG=2 and monitors ConnOn status (USER_CMD_REG[2]) until it is equal to '1'.
   b. For passive open, CPU waits until connection is opened by PC or FPGA. ConnOn status (USER_CMD_REG[2]) is monitored until it is equal to '1'.
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 as follows.
   a. Set transfer size of this loop to TOE10G IP register (TOE10_TDL_REG). Transfer size is fixed to 4 GB except the last loop which is equal to the remaining size.
   b. Set send command to TOE10G IP register (TOE10_CMD_REG=0).
   c. Wait until operation is completed by monitoring busy flag (TOE10_CMD_REG[0]='0'). During monitoring busy flag, CPU reads current transfer size from user logic (USER_TXLEN_REG and USER_RXLEN_REG) and displays the results 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.

3.4 Receive data test

User sets total received size, data verification mode (enable or disable), and connection mode (active open for client operation or passive open for server operation). 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/FPGA when data verification mode is enabled. The sequence of this test is as follows.

1) Receive total transfer size, data verification mode, and connection 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) Open connection following connection mode value (same as Step 4 of Send data test).
5) Wait until connection is closed by PC or FPGA. Connon status (USER_CMD_REG[2]) is monitored until it is equal to '0'. During monitoring Connon, CPU reads current transfer size from user logic (USER_TXLEN_REG and USER_RXLEN_REG) and displays the results on the console every second.
6) Read total received length of user logic (USER_RXLEN_REG) and wait until total length is equal to the set value from user. After total data is received, CPU checks verification result by reading USER_CMD_REG[1] ('0': normal, '1': error). If the error is detected, the error message will be displayed.
7) Calculate performance and show test result on the console.

3.5 Full duplex test

This menu is designed to run full duplex test by transferring data between FPGA and PC/FPGA in both directions by using the same port number at the same time. Four inputs are received from user, i.e. total size for both directions, packet size for FPGA sending logic, data verification mode for FPGA receiving logic, and connection mode (active open/close for client operation or passive open/close for server operation).

When running the test by using PC and FPGA, the transfer size setting on FPGA must be matched to the size setting on test application (tcp_client_txrx_40G). Connection mode on FPGA when running with PC must be set to passive (server operation).

The test runs in forever loop until the user cancels operation on PC by input Ctrl+C when test environment is PC and FPGA. For FPGA <-> FPGA environment, user cancels operation by pressing some keys on the console. The sequence of this test is as follows.

1) Receive total data size, packet size, data verification mode, and connection mode from the user and verify that the value is valid.
2) Display the recommended parameters of test application running on PC from the current system parameters.
3) Set UserReg registers, i.e. transfer size (USER_TXLEN_REG), reset flag to clear the test pattern value (USER_RST_REG[0]='1'), and command register to start data pattern generator with data verification mode (USER_CMD_REG=1 or 3).
4) Open connection following the connection mode value (same as Step 4 of Send data test).
5) Set packet size to TOE10G IP registers (TOE10_PKL_REG=user input) and calculate total transfer size in each loop. Maximum size of one loop is 4 GB. The operation of each loop is as follows.
   a. Set transfer size of this loop to TOE10_TDL_REG. Transfer size is fixed to maximum size (4GB) which is also aligned to packet size, except the last loop. The transfer size of the last loop is equal to the remaining size.
   b. Set send command to TOE10G-IP register (TOE10_CMD_REG=0).
   c. Wait until send command is completed by monitoring busy flag (TOE10_CMD_REG[0] ='0'). During monitoring busy flag, CPU reads current transfer size from user logic (USER_TXLEN_REG and USER_RXLEN_REG) and displays the result on the console every second.
6) Close connection following connection mode value.
   a. For active close, CPU waits until total received size is equal to set value from user. Then, set USER_CMD_REG=3 to close connection. Next, CPU waits until connection is closed by monitoring ConnOn (USER_CMD_REG[2]='0').
   b. For passive close, CPU waits until connection is closed from FPGA/PC by monitoring ConnOn (USER_CMD_REG[2]='0').
7) Check the result and the error (same as Step 6 of Receive data test).
8) Calculate performance and show the test result on the console. Go back to step 3 to repeat the test in forever loop.

3.6 Function list in User application

This topic describes the function list to run TOE10G IP operation.

void exec_port(unsigned int port_ctl, unsigned int mode_active)	
Parameters	port_ctl: 1-Open port, 0-Close port mode_active: 1-Active open/close, 0-Passive open/close
Return value	None
Description	For active mode, write TOE10_CMD_REG to open or close connection. After that, call read_conon function to monitor connection status until it changes from ON to OFF or OFF to ON, depending on port_ctl mode.

void init_param(void)	
Parameters	None
Return value	None
Description	Set network parameters to TOE10G IP register from global parameters. After reset is de-asserted, it waits until TOE10G IP busy flag is de-asserted to '0'.

int input_param(void)	
Parameters	None
Return value	0: Valid input, -1: Invalid input
Description	Receive network parameters from user, i.e. Mode, Window threshold, FPGA MAC address, FPGA IP address, FPGA port number, Target IP address, and Target port number. If the input is valid, the parameters will be updated. Otherwise, the value does not change. After receiving all parameters, the current value of each parameter will be displayed.

Unsigned int read_conon(void)	
Parameters	None
Return value	0: Connection is OFF, 1: Connection is ON.
Description	Read value from USER_CMD_CONNON register and return only bit2 value to show connection status.

void show_cursize(void)	
Parameters	None
Return value	None
Description	Read USER_TXLEN_REG and USER_RXLEN_REG, and then display in Byte, KByte, or MByte unit

void show_param(void)	
Parameters	None
Return value	None
Description	Display the current value of the network parameters setting to TOE10G IP such as IP address, MAC address, and port number.

void show_result(void)	
Parameters	None
Return value	None
Description	Read USER_TXLEN_REG and USER_RXLEN_REG to display total size. Read the global parameters (timer_val_end and timer_val_start) and calculate total time usage to display in usec, msec, or sec unit. Finally, transfer performance is calculated and displayed on MB/s unit.

int toe10g_recv_test(void)	
Parameters	None
Return value	0: The operation is successful -1: Receive invalid input or error is found
Description	Run Receive data test following description in topic 3.4

int toe10g_send_test(void)	
Parameters	None
Return value	0: The operation is successful -1: Receive invalid input or error is found
Description	Run Send data test following description in topic 3.3

int toe10g_txrx_test(void)	
Parameters	None
Return value	0: The operation is successful -1: Receive invalid input or error is found
Description	Run Full duplex test following described in topic 3.5

void wait_ethlink(void)	
Parameters	None
Return value	None
Description	Read USER_RST_REG[16] and wait until ethernet is linked up

4 Test Software (PC)

4.1 "tcpdatatest" for half duplex test

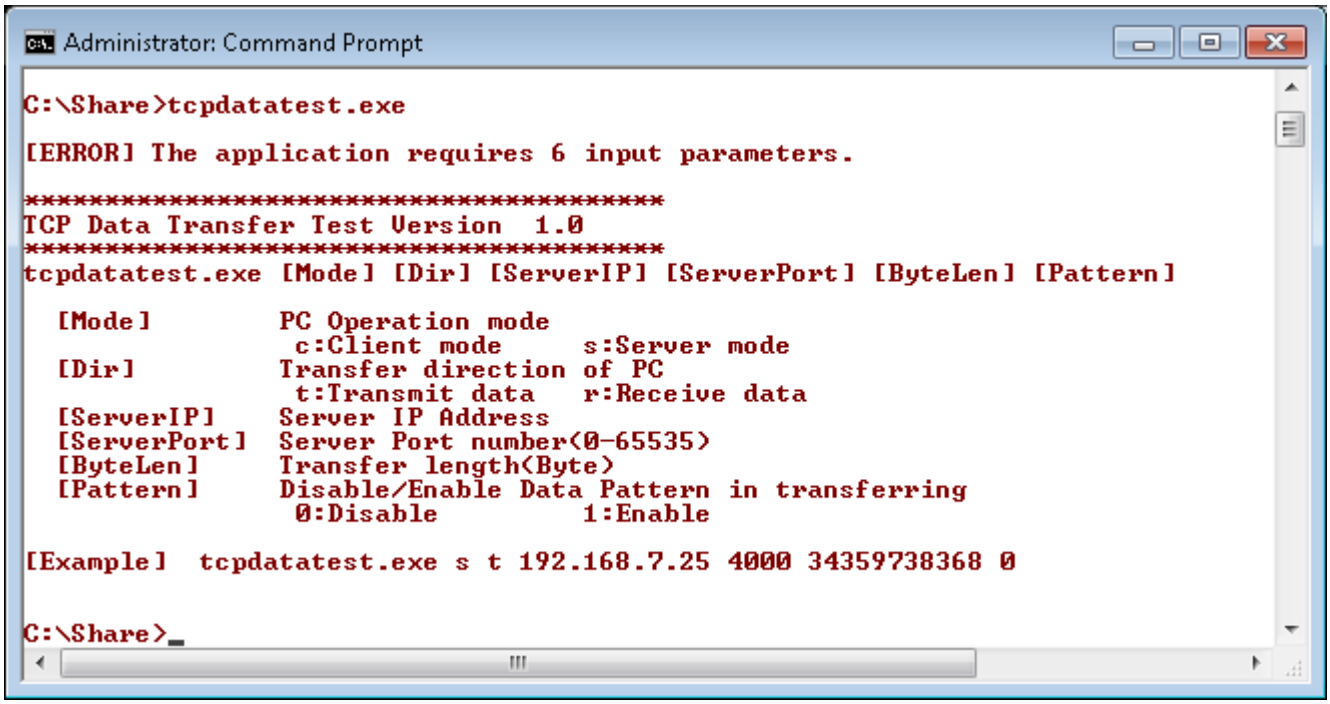

Figure 4-1 "tcpdatatest" application usage

"tcpdatatest" is designed to run on PC for sending/receiving TCP data through Ethernet for both server and client mode. PC of this demo should run in client mode. User inputs parameter to select transfer direction and the mode. Six parameters are required as follows.

1) Mode: c –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 only in transmit mode, but ignored in receive mode. In receive mode, the application is closed when the connection is destroyed. ByteLen in transmit mode must be equal to transfer size on FPGA, set in received data test menu.
6) Pattern:
   0 – Generate dummy data in transmit mode or disable data verification in receive mode.
   1 – Generate incremental 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) Get parameters from the user and verify that the input is valid.
2) Create the socket and set socket options.
3) Create the new connection by using server IP address and server port number.
4) Allocate 1 MB memory to be sent buffer.
5) Generate the incremental test pattern to send buffer when the test pattern is enabled. Skip this step if the dummy pattern is selected.
6) Send data out and read total sent data from the function.
7) Calculate remaining transfer size.
8) Print total transfer size every second.
9) Repeat step 5) – 8) until the remaining transfer size is 0.
10) Calculate total performance and print the result on the console.
11) Close the socket and free the memory.

Receive data mode

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

1) Follow the step 1) – 3) of Transmit data mode.
2) Allocate 1 MB memory to be received buffer.
3) Read data from the received buffer and increase total received size.
4) If verification is enabled, data will be verified by the incremental pattern. Error message is printed out when data is not correct. This step will be skipped if data verification is disabled.
5) Print total transfer size every second.
6) Repeat step 3) – 5) until the connection is closed.
7) Calculate total performance and print the result on the console.
8) Close socket and free the memory.

4.2 "tcp_client_txrx_40G" for full duplex test

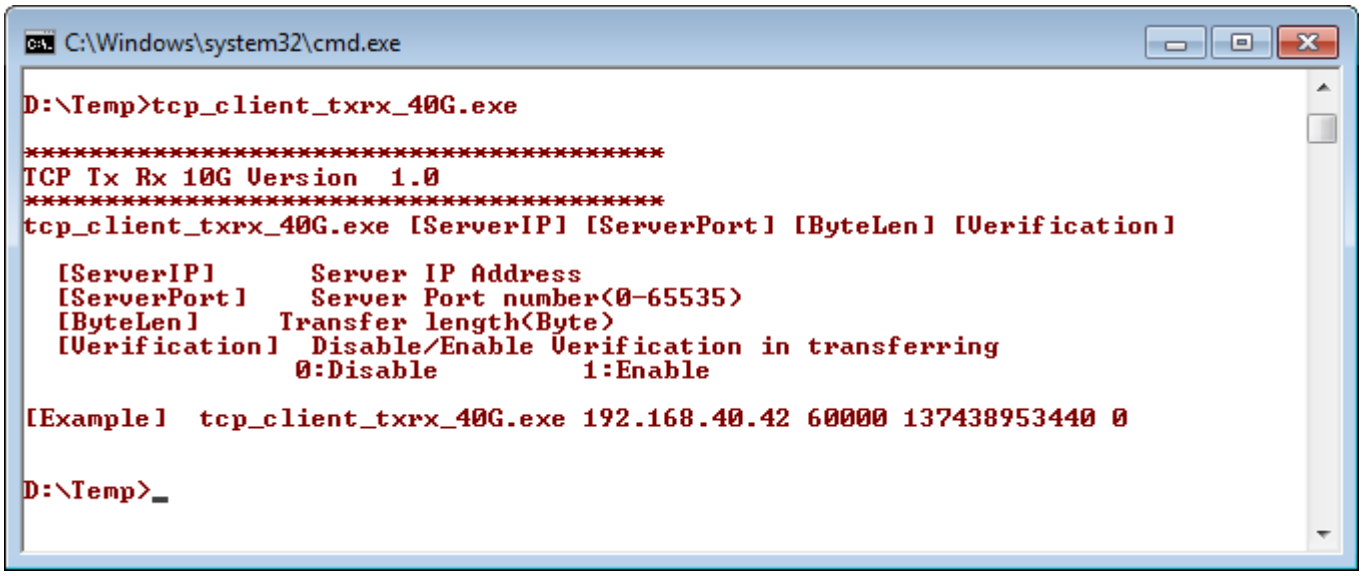

Figure 4-2 "tcp_client_txrx_40G" application usage

"tcp_client_txrx_40G" application is designed to run on PC for sending and receiving TCP data through Ethernet by using the same port number at the same time. The application is run in client mode, so user needs to input server parameters (the network parameters of TOE10G IP). As shown in Figure 4-2, there are four parameters to run the application, i.e.

1) ServerIP: IP address of FPGA
2) ServerPort: Port number of FPGA
3) ByteLen: Total transfer size in byte unit. This is total size to transmit and receive data.
4) 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 incremental data for sending function and enable data verification for receiving function.

The sequence of test application is as follows.

(1) Get parameters from the user and verify that the input is valid.
(2) Create the socket and set socket options.
(3) Create the new connection by using server IP address and server port number.
(4) Allocate 64 KB memory for sent and received buffer.
(5) Generate incremental test pattern to send buffer when the test pattern is enabled. Skip this step if dummy pattern is selected.
(6) The step for sending data is calling function to send data, reading returned parameter of send function to check the number of completely sent data, and calculating the remaining sent size.
(7) The step for reading data is calling function to read data from the received buffer and increasing total received data size.
(8) If verification is enabled, data will be verified by incremental pattern. Error message is printed out when data is not correct. Skip this step if data verification is disabled.
(9) Print total sent and received size every second.
(10) Repeat step 5) – 9) until total sent and received size are equal to ByteLen (set by user).
(11) Calculate performance and print the result on the console.
(12) Close the socket.
(13) Sleep for 1 second to wait until the hardware completes the current test loop.
(14) Run step 3) – 13) in forever loop. If verification is failed, the application will be stopped.

5 Revision History

Revision	Date	Description
1.0	22-Jan-18	Initial version release
1.1	2-Apr-18	Support FPGA<->FPGA connection
1.2	23-Aug-19	Add function list and use tcp_client_txrx_40G software