TOE1G-IP with CPU reference design

Rev1.1 6-Dec-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 layer and Physical layer.

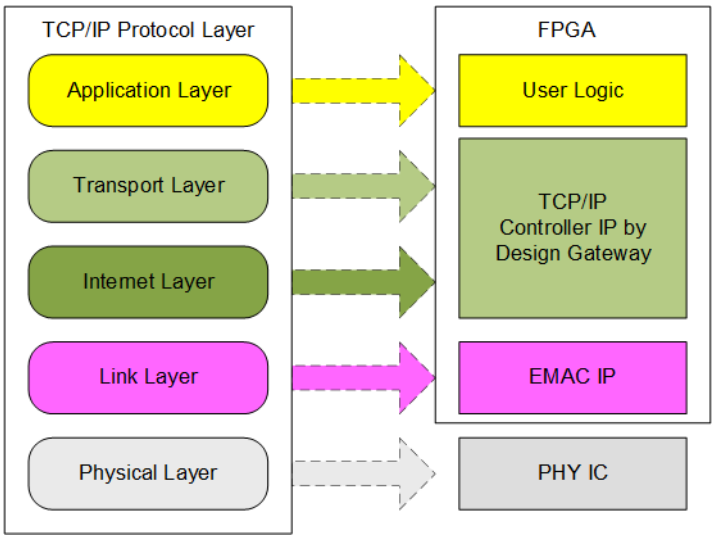

Figure 1-1 TCP/IP Protocol Layer

TOE1G-IP implements Transport and Internet layer of TCP/IP Protocol. Building Ethernet packet from the user data which is TCP data to EMAC, TOE1G-IP splits 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 TOE1G-IP. The header is applied to verify the packet and TCP data is forwarded to the user logic. The packet is rejected when TCP/IP header in the packet is invalid. The lower layer protocols are implemented by EMAC-IP from Intel FPGA and external PHY chip.

The reference design provides the evaluation system which includes simple user logic to send and receive data by using TOE1G-IP. TOE1G-IP is designed to transfer data with PC or another TOE1G-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 high speed rate. Two test applications are specially designed for running 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 allow the user controlling the test parameters and the operation of TOE1G-IP demo through JTAG UART, the CPU system is included. It is easy for the user to set and monitor the test parameters and the system status on the console through JTAG UART. The firmware on CPU is built by using bare-metal OS. More details of the demo are described as follows.

2 Hardware overview

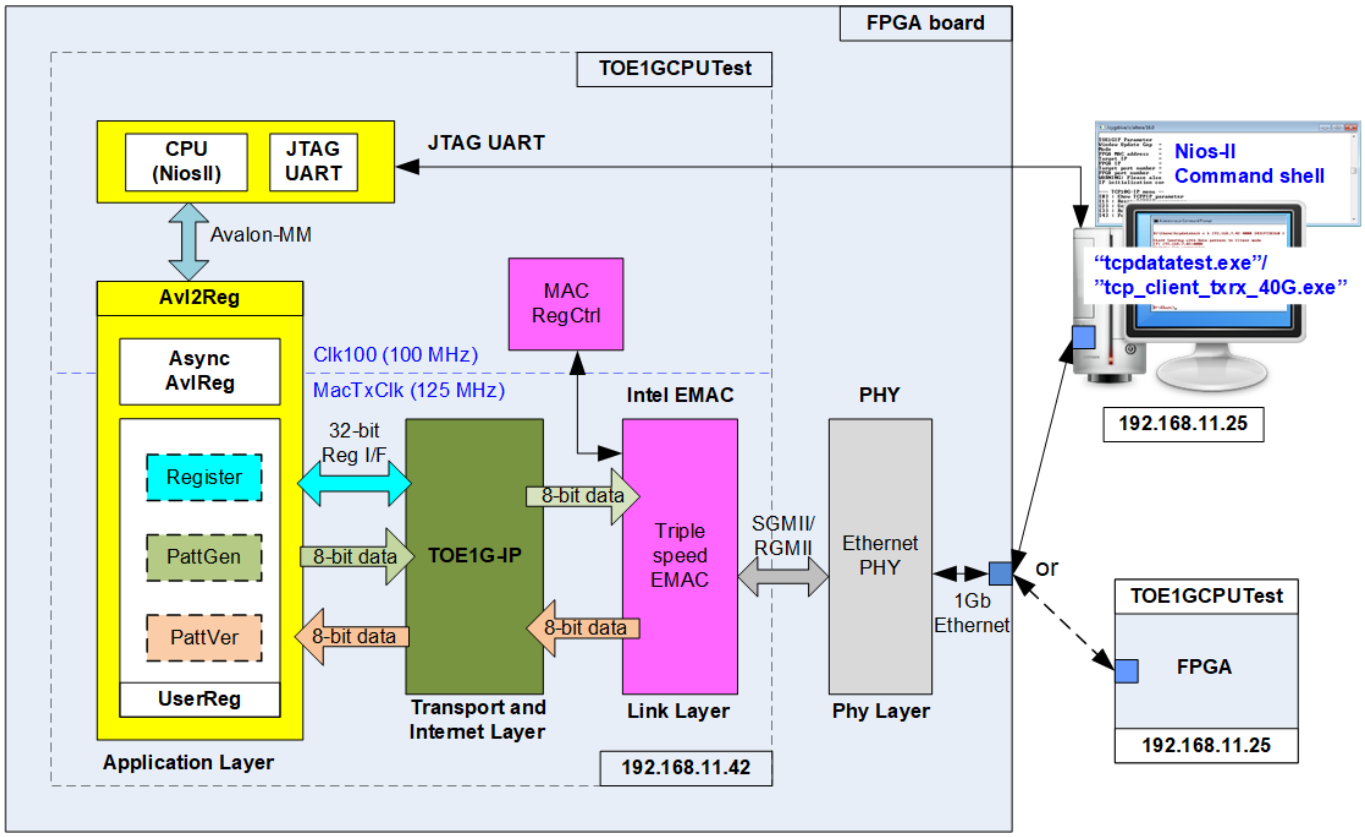

Figure 2-1 Demo Block Diagram

TOE1G-IP needs to connect to Triple-Speed Ethernet MAC and Ethernet PHY which implement lower-layer protocol. The user interfaces of TOE1G-IP are connected to UserReg for both control and data interface. For data interface, UserReg includes PattGen logic to generate test pattern for sending data and VerifyPatt logic to generate test pattern for verifying received data. Test pattern generated in UserReg is 32-bit incremental pattern.

For control interface, UserReg includes register to store parameters from user such as transfer length, transfer direction, and transfer mode which are set through JTAG UART. CPU converts the user parameters to be value for setting to hardware register through Avalon bus. Due to the fact that CPU system and TOE1G-IP run in different clock domain, AsyncAvlReg module is applied to be asynchronous circuit to support clock-crossing function and then convert Avalon bus signal which is standard bus in CPU system to be register interface. CPU in the demo is NiosII and runs on bare-metal OS. CPU system includes JTAG UART hardware for user interface and timer to measure transfer performance.

2.1 Ethernet PHY

Ethernet PHY is implemented by external PHY chip. The interface of external PHY chip must be matched to Ethernet MAC. PHY Interface could be selected to be GMII, RGMII (CycloneV E board and ArriaV GX Starter board), or SGMII (Cyclone10GX and Arria10 SoC board). Ethernet speed is fixed to 1 Gbps mode.

2.2 Triple speed Ethernet MAC

Link layer and PCS/PMA are implemented by Triple-speed Ethernet MAC, provided by Intel FPGA. EMAC has two user interfaces, i.e. Avalon stream for transferring data and Avalon-MM for configuration. In reference design, Avalon stream of EMAC is connected to TOE1G-IP while Avalon-MM interface is connected to MACRegCtrl module. The details about the register for EMAC configuration are described in “Configuration Register Space” topic within “Triple-Speed Ethernet MegaCore Function User Guide” document, provided by Intel. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_ethernet.pdf

2.3 MACRegCtrl

This module is designed to configure parameter of Triple Ethernet MAC and monitor EMAC status through Avalon-MM bus. The logic is simply designed by using state machine. This module runs once after system power up to initialize Ethernet MAC.

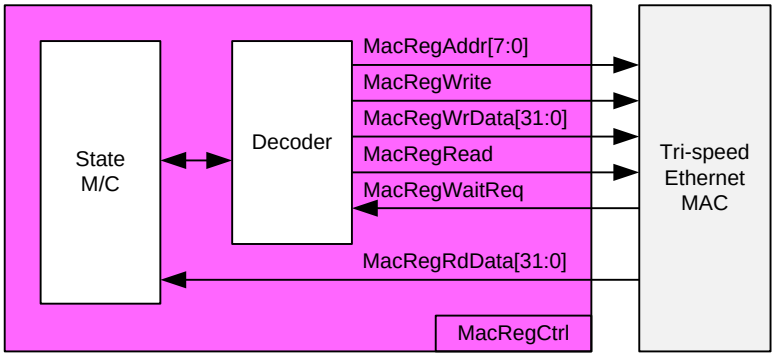

Figure 2-2 MACRegCtrl block diagram

The sequence of initialization sequence is below.

1) Disable transmit and receive path of EMAC
2) Set software reset
3) Set frame length to support jumbo frame
4) Set Tx IFG length
5) Enable transmit and receive path of EMAC

For SGMII mode, the configuration step completes in step 5. Step 6 is necessary for RGMII mode to enable Receive/Transmit timing control function through MDIO.

6) Enable RGMII timing control

2.4 TOE1G-IP

TOE1G-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.
https://dgway.com/products/IP/TOE1G-IP/dg_toe1gip_data_sheet_intel_en.pdf

2.5 CPU and Peripherals

32-bit Avalon-MM is applied to be the bus interface for the CPU accessing the peripherals such as Timer and JTAG 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 Avalon-MM bus. CPU assigns the different base address and the address range to each peripheral for accessing the internal registers of the 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 Avalon-MM bus standard for writing and reading the register. Avl2Reg module is designed to connect the CPU system as shown in Figure 2-3.

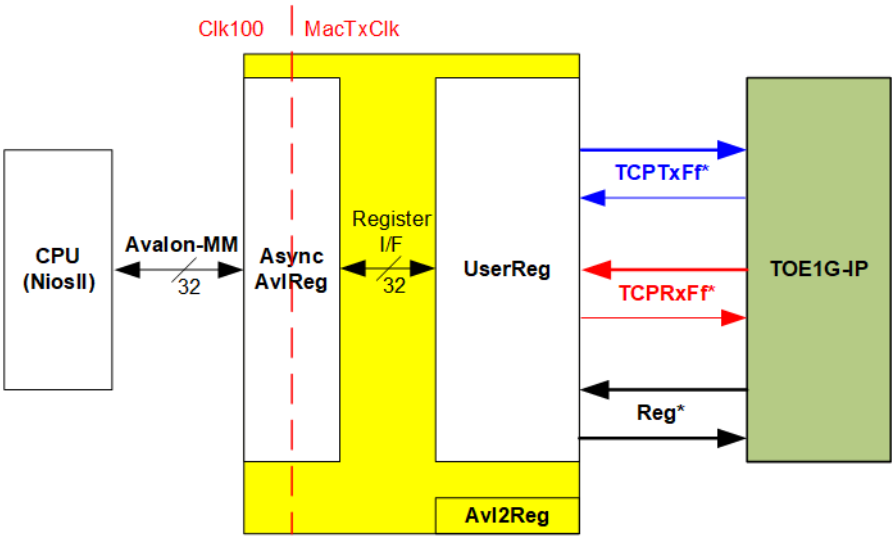

Figure 2-3 Avl2Reg block diagram

Avl2Reg consists of AsyncAvlReg and UserReg. AsyncAvlReg is designed to convert the Avalon-MM signals to be the simple register interface which has 32-bit data bus size (similar to Avalon-MM data bus size). Otherwise, AsyncAvlReg includes asynchronous logic to support clock crossing between Clk100 domain and MacTxClk domain.

UserReg includes the register file of the parameters and the status signals. Also, data interface and control interface of TOE1G-IP are connected to UserReg. More details of AsyncAvlReg and UserReg are described as follows.

2.5.1 AsyncAvlReg

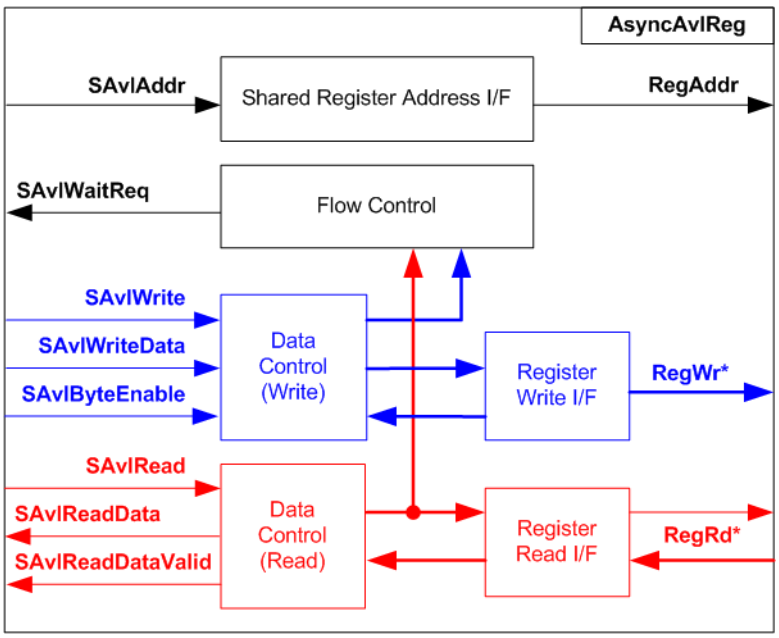

Figure 2-4 AsyncAvlReg Interface

The signal on Avalon-MM bus interface can be split into three groups, i.e. Write channel (blue color), Read channel (red color) and Shared control channel (black color). More details of Avalon-MM interface specification are described in following document.
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf

According to Avalon-MM specification, only one command (write or read) can be operated at a time. The logics inside AsyncAvlReg are split into three groups, i.e. Write control logic, Read control logic, and Flow control logic. Flow control logic to control SAvlWaitReq is applied to hold the next request from Avalon-MM interface while the current request is operating. Write control I/F and Write data I/F of Avalon-MM bus are latched and transferred as Write register. On the other hand, Read control I/F and Read data I/F of Avalon-MM bus are latched and transferred as Read register. Address I/F of Avalon-MM is latched and transferred to Address register interface as well.

The simple register interface is designed to be compatible to general RAM interface for write transaction. The read transaction of the register interface is slightly modified from RAM interface by adding RegRdReq and RegRdValid 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 2-5.

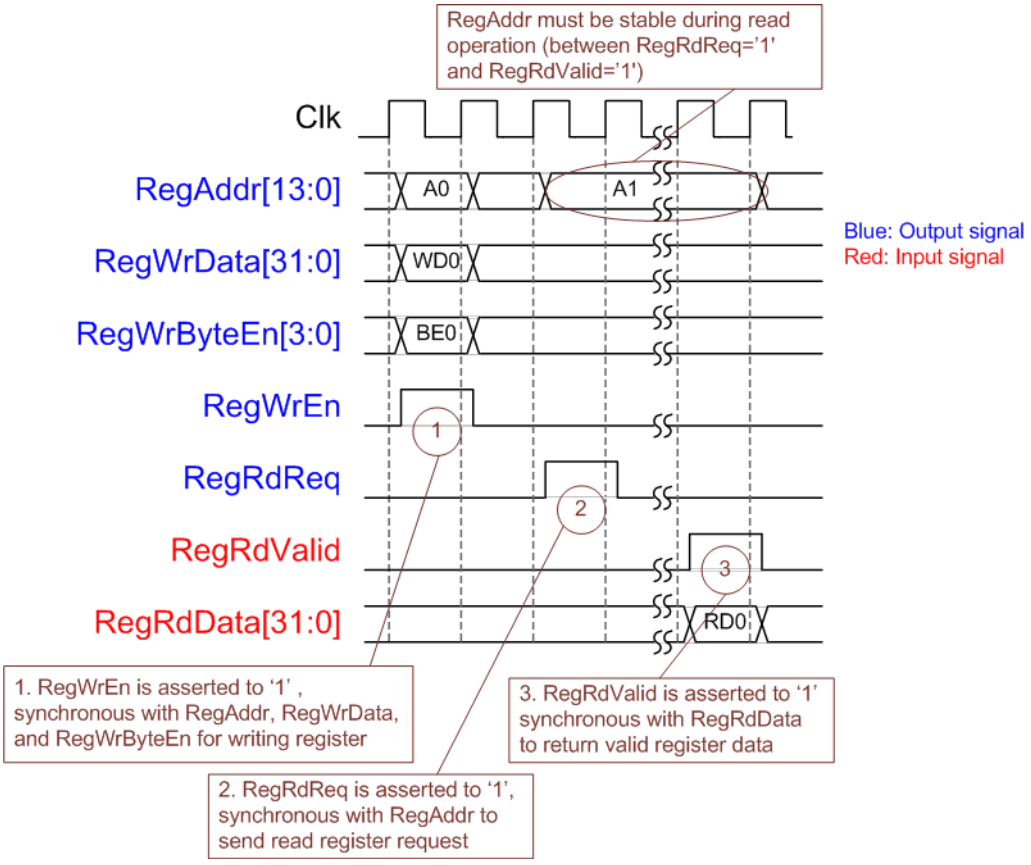

Figure 2-5 Register interface timing diagram

1) To write register, the timing diagram is the same as general RAM interface. RegWrEn is asserted to '1' with the valid RegAddr (Register address in 32 bit unit), RegWrData (write data of the register), and RegWrByteEn (the write byte enable). Byte enable has four bits 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, AsyncAvlReg 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, AsyncAvlReg forwards the read value to SAvlRead interface.

2.5.2 UserReg

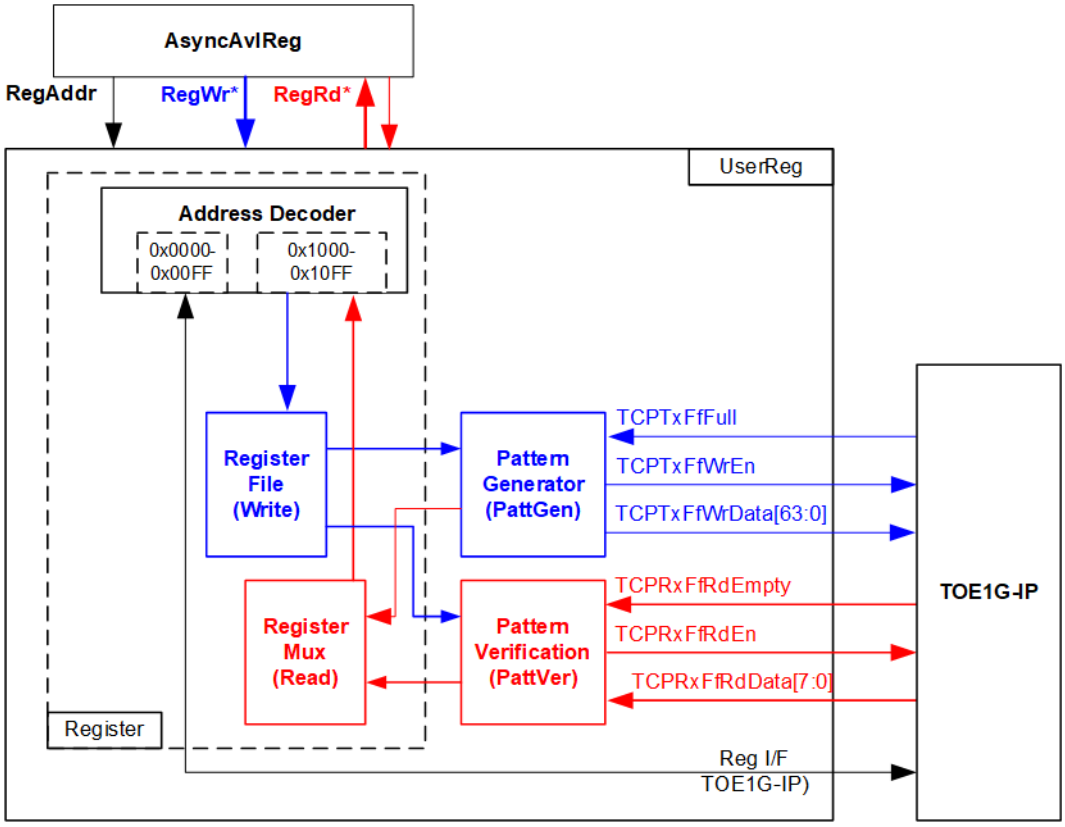

Figure 2-6 UserReg block diagram

The logic inside UserReg has three operations, i.e. Register, Pattern generator (PattGen), and Pattern verification (PattVer). Register block decodes the address requested from AsyncAvlReg and then selects the active register for write or read transaction. Pattern generator block is designed to send 8-bit test data to TOE1G-IP following FIFO interface standard. Pattern verification block is designed to read and verify 8-bit data from TOE1G-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. TOE1G-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 RegRdReq with asserting one D Flip-flop. More details of the address mapping within UserReg module is shown in Table 2-1.

Table 2-1 Register map Definition

Address Wr/Rd	Register Name (Label in “toe1gip_demo.c”)	Description
BA+0x0000 – BA+0x00FF: TOE1G-IP Register Area More details of each register are described in TOE1G-IP datasheet.		
BA+0x00	TOE_RST_REG	Mapped to RST register within TOE1G-IP
BA+0x04	TOE_CMD_REG	Mapped to CMD register within TOE1G-IP
BA+0x08	TOE_SML_REG	Mapped to SML register within TOE1G-IP
BA+0x0C	TOE_SMH_REG	Mapped to SMH register within TOE1G-IP
BA+0x10	TOE_DIP_REG	Mapped to DIP register within TOE1G-IP
BA+0x14	TOE_SIP_REG	Mapped to SIP register within TOE1G-IP
BA+0x18	TOE_DPN_REG	Mapped to DPN register within TOE1G-IP
BA+0x1C	TOE_SPN_REG	Mapped to SPN register within TOE1G-IP
BA+0x20	TOE_TDL_REG	Mapped to TDL register within TOE1G-IP
BA+0x24	TOE_TMO_REG	Mapped to TMO register within TOE1G-IP
BA+0x28	TOE_PKL_REG	Mapped to PKL register within TOE1G-IP
BA+0x2C	TOE_PSH_REG	Mapped to PSH register within TOE1G-IP
BA+0x30	TOE_WIN_REG	Mapped to WIN register within TOE1G-IP
BA+0x34	TOE_SRV_REG	Mapped to SRV register within TOE1G-IP
BA+0x38	TOE_RST_REG	Mapped to RST register within TOE1G-IP
BA+0x3C	UDP_VER_REG	Mapped to VER register within TOE1G-IP
BA+0x1000 – BA+0x10FF: UserReg control/status		
BA+0x1000 Wr/Rd	Total transmit length (USER_TXLEN_REG)	Wr [31:0] – Total transmitted byte size. Valid from 1-0xFFFFFFFF. Rd [31:0] – Current transmitted byte size. 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 (USER_FFSTS_REG)	Rd [15:0] - Mapped to TCPRxFfRdCnt signal of TOE1G-IP [24] - Mapped to TCPTxFfFull signal of TOE1G-IP
BA+0x1010 Rd	Total receive length (USER_RXLEN_REG)	Rd [31:0] – Current received byte size. The value is cleared to 0 when USER_CMD_REG is written by user.

Pattern Generator

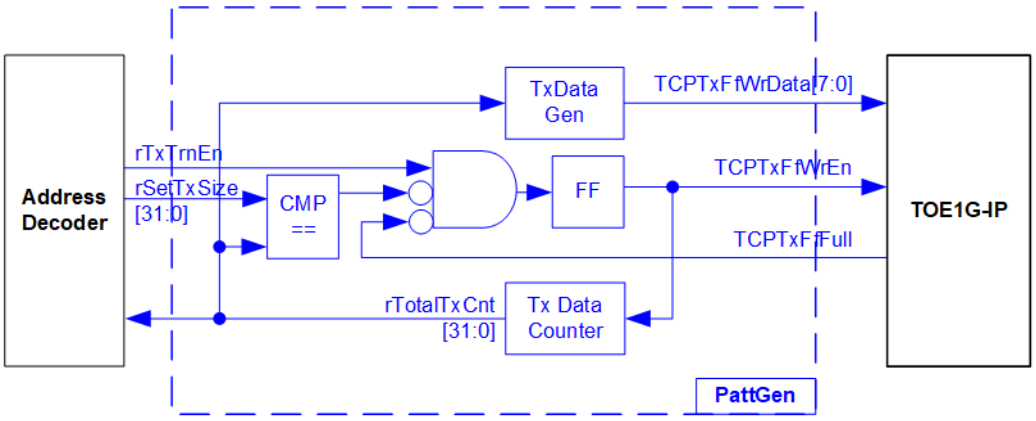

Figure 2-7 PattGen block

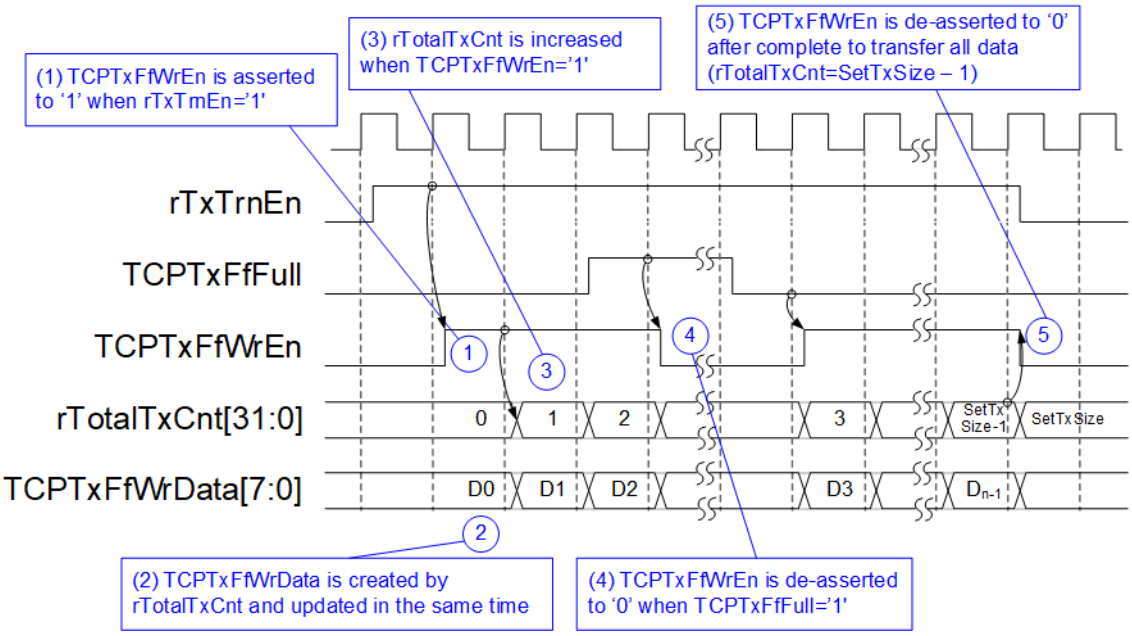

Figure 2-8 PattGen Timing diagram

PattGen is designed to generate test data to TOE1G-IP. rTxTrnEn is asserted to '1' when USER_CMD_REG[0] is set to '0'. 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 TOE1G-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

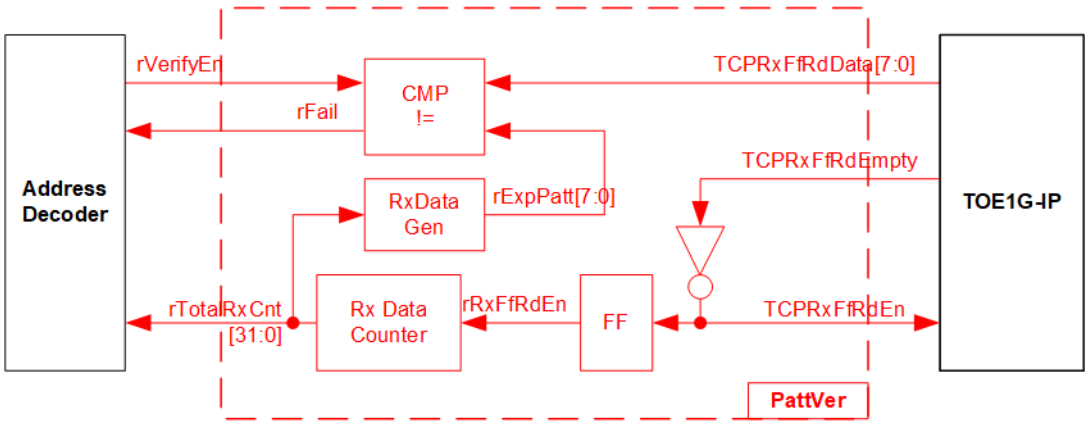

Figure 2-9 PattVer block

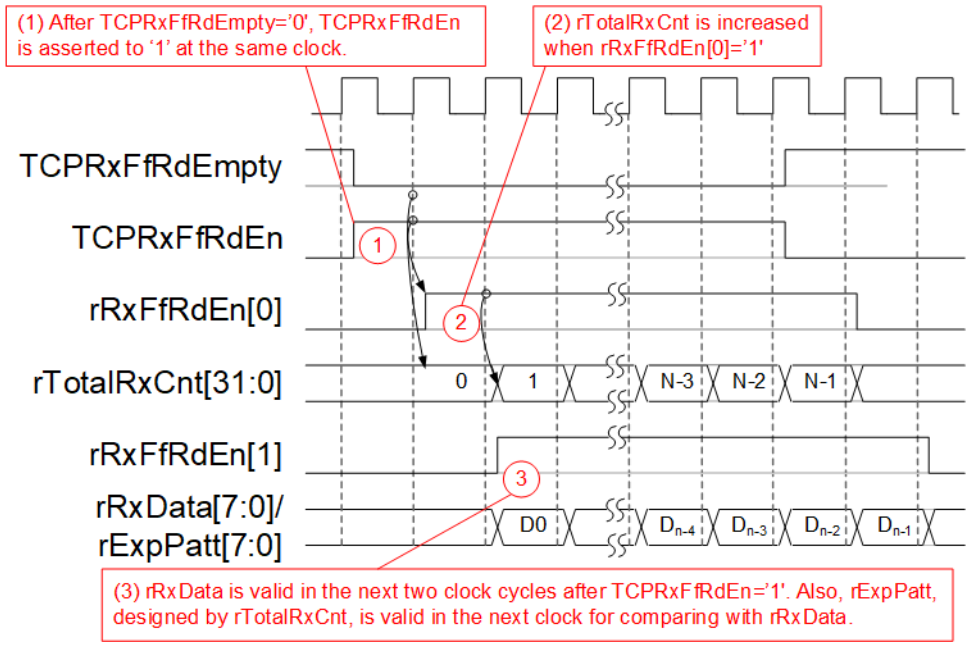

Figure 2-10 PattVer Timing diagram

PattVer is designed to read test data from TOE1G-IP with or without data verification, depending on rVerifyEn flag. When rVerifyEn is set to '1', data comparison is enabled to compare read data (rRxData which is one clock delay of TCPRxFfRdData) to the expected pattern (rExpPatt). When data verification is failed, rFail is asserted to '1'. TCPRxFfRdEn is designed by using NOT logic of TCPRxFfRdEmpty. TCPRxFfRdData is valid for data comparison in the next clock. rRxFfRdEn[0] 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 rExpPatt.

3 CPU Firmware Sequence on FPGA

After FPGA boot-up, welcome message is displayed. 1G Ethernet link up status (USER_RST_REG[16]) is monitored until link-up is found. Next, user selects the operation mode of TOE1G-IP to be client or server mode.

To initialize as client mode, TOE1G-IP sends ARP request to get the MAC address from the destination device. For server mode, TOE1G-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 TOE1G-IPs must be different (one is client and another is server). To run with PC, it is recommended to set FPGA as client mode. When PC receives ARP request, PC always returns ARP reply. It is not simple 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.

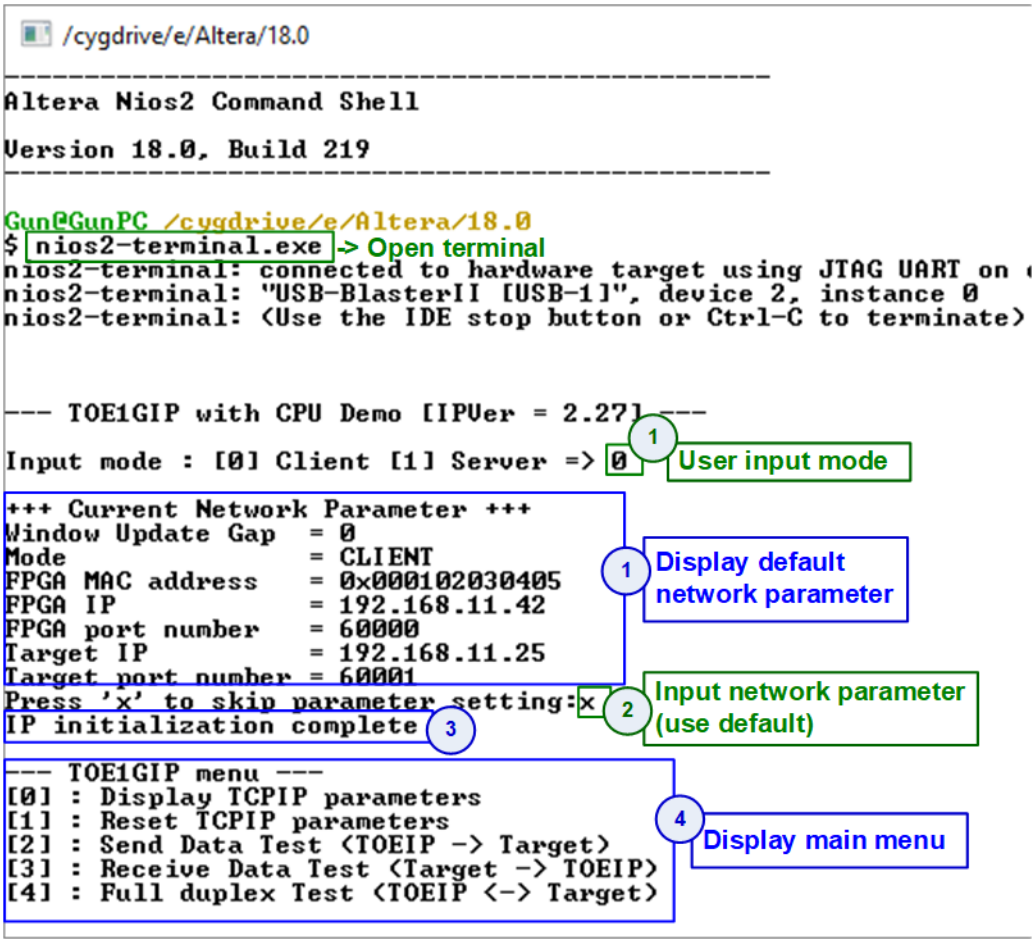

Figure 3-1 Example of initialization sequence in client mode on NiosII terminal

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 TOE1G-IP finishing initialization sequence (TOE_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 TOE1G-IP, i.e. operation mode, source MAC address, destination IP address, source IP address, destination port, and source port. The step to 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 TOE1G-IP parameters such as IP address and source port number. After setting updated parameter to TOE1G-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 step 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. When the input is invalid, the old value is used instead.
3) Force reset to IP by setting TOE_RST_REG[0]='1'.
4) Set all parameters to TOE1G-IP register such as TOE_SML_REG and TOE_DIP_REG.
5) De-assert IP reset by setting TOE_RST_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 (TOE_CMD_REG[0]) until the initialization process is completed (busy flag is de-asserted to '0').

3.3 Send data test

Three user inputs are received to set total transmit length, packet size, and connection mode (active open for client operation or passive open for server operation). The operation is 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. The step to run send data 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 TOE1G-IP.
3) Display recommended parameter of test application running on PC from the current system parameters.
4) Open connection following connection mode.
   a. For active open, CPU sets TOE_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 TOE1G-IP register (TOE_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 TOE1G-IP register (TOE_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 TOE1G-IP register (TOE_CMD_REG=0).
   c. Wait until operation is completed by monitoring busy flag (TOE_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 TOE1G-IP register (TOE_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 is cancelled when some inputs are invalid. During the test, 32-bit incremental data is generated to verify the received data from PC or FPGA when data verification mode is enabled. The step to run receive data 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 test pattern value (USER_RST_REG[0]='1'), and data verification mode (USER_CMD_REG[1]='0' or '1').
3) Display recommended parameter (similar to Step 3 of Send data test).
4) Open connection following connection mode (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). When the error is detected, the error message is 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 for PC and FPGA environment. For FPGA <-> FPGA environment, user cancels operation by pressing some keys to the console. The step to run Full duplex 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 TOE1G-IP register (TOE_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 TOE_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 TOE1G-IP register (TOE_CMD_REG=0).
   c. Wait until send command is finished by monitoring busy flag (TOE_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) Close connection following connection mode value.
   a. For active close, CPU waits until total received size is equal to the 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 run the test in forever loop.

3.6 Function list in User application

This topic describes the function list to run TOE1G-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 TOE_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 TOE1G-IP register from global parameters. After reset is de-asserted, it waits until TOE1G-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. When the input is valid, the parameters are updated. Otherwise, the value does not change. After receiving all parameters, the current value of each parameter is 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 the current transmitted and received size 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 TOE1G-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 transmitted size and total received size. Read the global parameters (timer_val and timer_upper_val) 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 toe_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 toe_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 toe_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 description in topic 3.5

void wait_ethlink(void)	
Parameters	None
Return value	None
Description	Read USER_RST_REG[16] and wait until linkup status is found

4 Test Software on PC

4.1 "tcpdatatest" for half duplex test

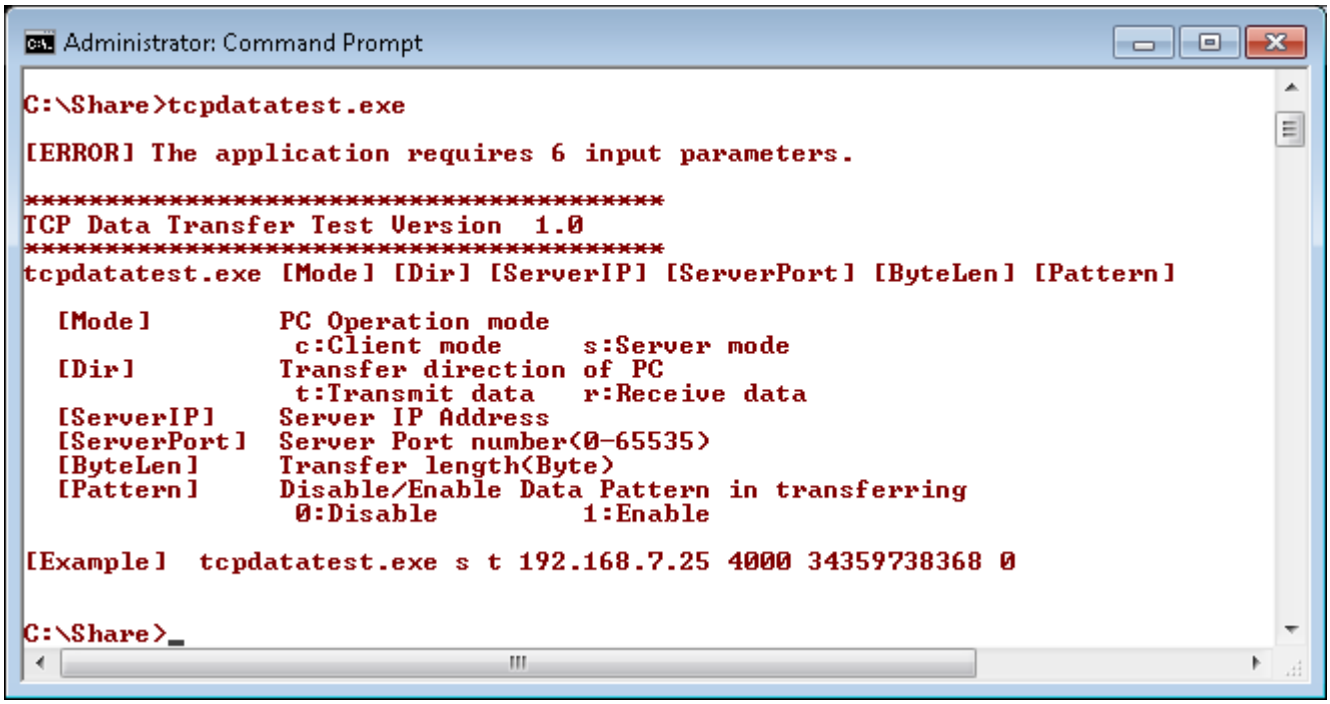

Figure 4-1 "tcpdatatest" application usage

"tcpdatatest" is designed to run on PC for sending/receiving TCP data through Ethernet in 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 in transmit mode only and ignored in receive mode. In receive mode, the application is closed when the connection is destroyed. ByteLen in transmit mode must be equal to the total 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 is verified by the incremental pattern. Error message is printed out when data is not correct. This step is 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

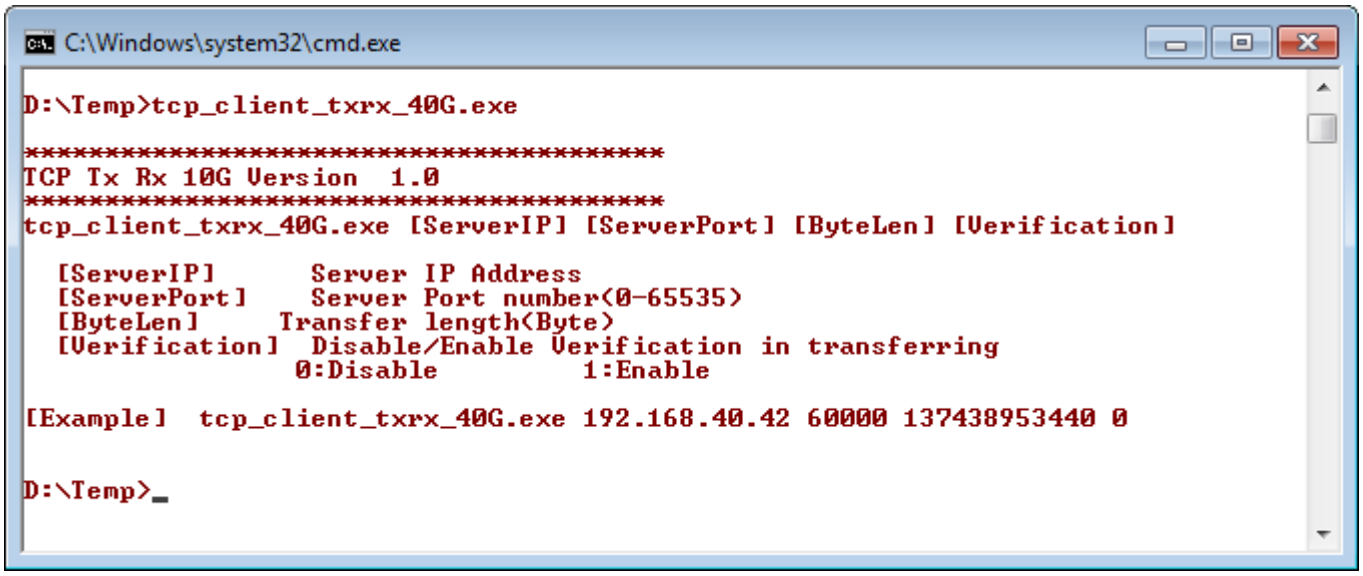

Figure 4-2 "tcp_client_txrx_40G" application usage

"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 TOE1G-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 byte size. 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) Send data out, read total sent data from the function, and calculate remaining sent size.
(7) Read data from the received buffer and increase total received data size.
(8) If verification is enabled, data is 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 is stopped.

5 Revision History

Revision	Date	Description
1.0	17-Jan-19	Initial version release
1.1	6-Dec-19	Add Function list