

# TOE10G-IP with CPU reference design

Rev1.1 2-Apr-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.

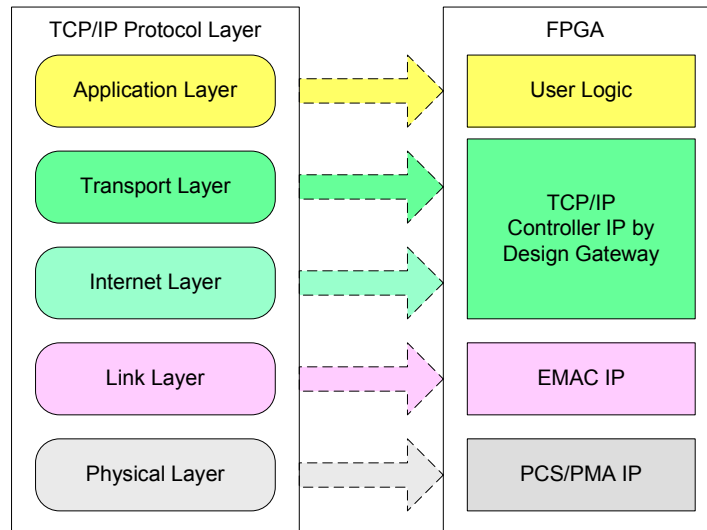


Figure 1-1 TCP/IP Protocol Layer

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\_txx\_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 overview

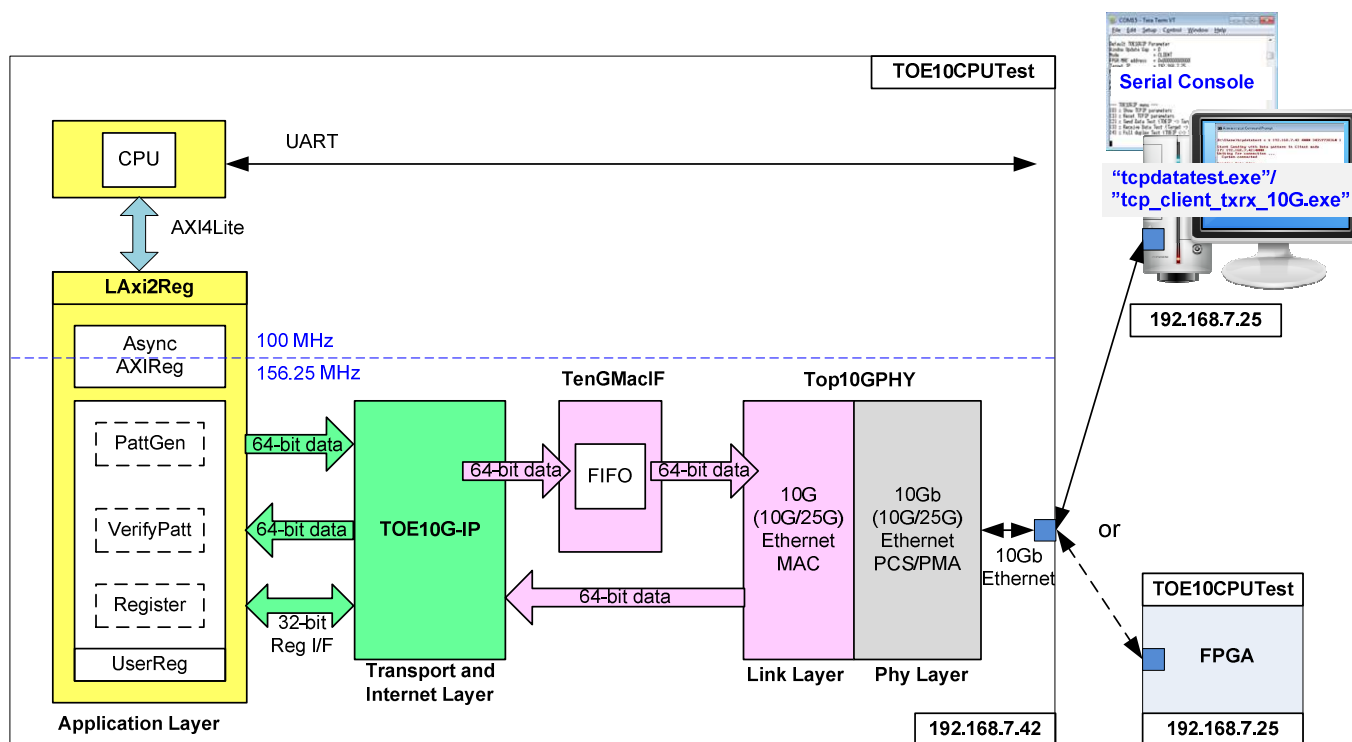


Figure 2-1 Demo Block Diagram

In test environment, two devices are used for 10Gb Ethernet transferring. First device runs in client mode and another runs in server mode. To confirm TOE10G-IP operation in both modes, the demo can be tested by using two test environments. First is using two FPGAs (one is client and one is server). Second is using one FPGA and one PC, as shown in Figure 2-1.

In FPGA logic, 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. The user interfaces of TOE10G-IP are connected to LAXi2Reg. For data interface, UserReg includes PattGen to generate test pattern to TOE10G-IP. Also, VerifyPatt is designed to verify received data from TOE10G-IP. Test pattern in the reference design is 32-bit increment data.

For control interface, LAXi2Reg includes register to store test parameters from user such as transfer length and transfer direction. User inputs parameters through Serial console. CPU firmware validates all parameters and sets to the hardware 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. Timer is also included 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\_trrx\_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 UserReg transfer 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 during packet transferring (between start-of-frame and end-of-frame). TenGMaClF 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. TenGMaClF 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](http://www.dgway.com/products/IP/TOE10G-IP/dg_toe10gip_data_sheet_xilinx_en.pdf)

## 2.3 TenGMaClF

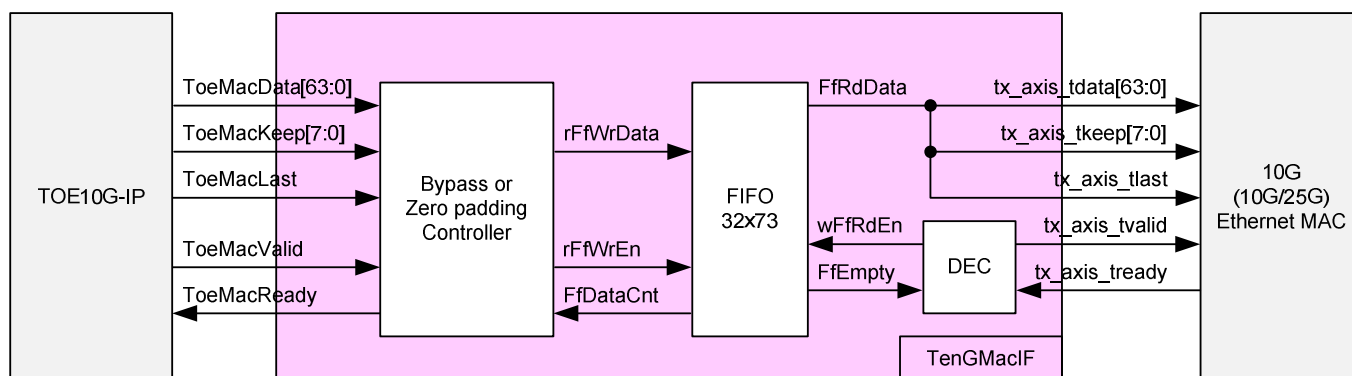


Figure 2-2 TenGMaClF Block Diagram

This module is designed to be adapter logic connecting between Tx interface of TOE10G-IP and Tx interface of 10G Ethernet MAC. ToeMacReady output to TOE10G-IP must be always asserted to '1' during transferring the 1<sup>st</sup> data and the last data. Following Xilinx 10G Ethernet MAC specification, tx\_axis\_tready of Xilinx EMAC may be de-asserted to '0' during transferring start-of-frame and end-of-frame. So, timing diagram of TOE10G-IP and Xilinx EMAC does not match. TenGMaClF 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, TenGMaClF needs to add zero data to the packet which the size is less than 60 bytes. ToeMacReady output to TOE10G-IP for normal packet is designed by following sequence.

As shown in Figure 2-3, when free space in FIFO is much enough (FfDataCnt is less than 16) and previous transmit packet is completed more than two clock cycles (rToeMacReady[2] which is 2-clock delayed from ToeMacReady is de-asserted to '0'), ToeMacReady is asserted to '1' to show ready status for receiving the new packet. The new packet from TOE10G-IP is transferred to TenGMaClF continuously because ToeMacReady is always asserted to '1'. ToeMacReady is de-asserted to '0' after 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'.

Internal FIFO is designed to store 73-bit inputs, i.e. 64-bit data (ToeMacData), 8-bit byte enable signal (ToeMacKeep), and last flag (ToeMacLast).

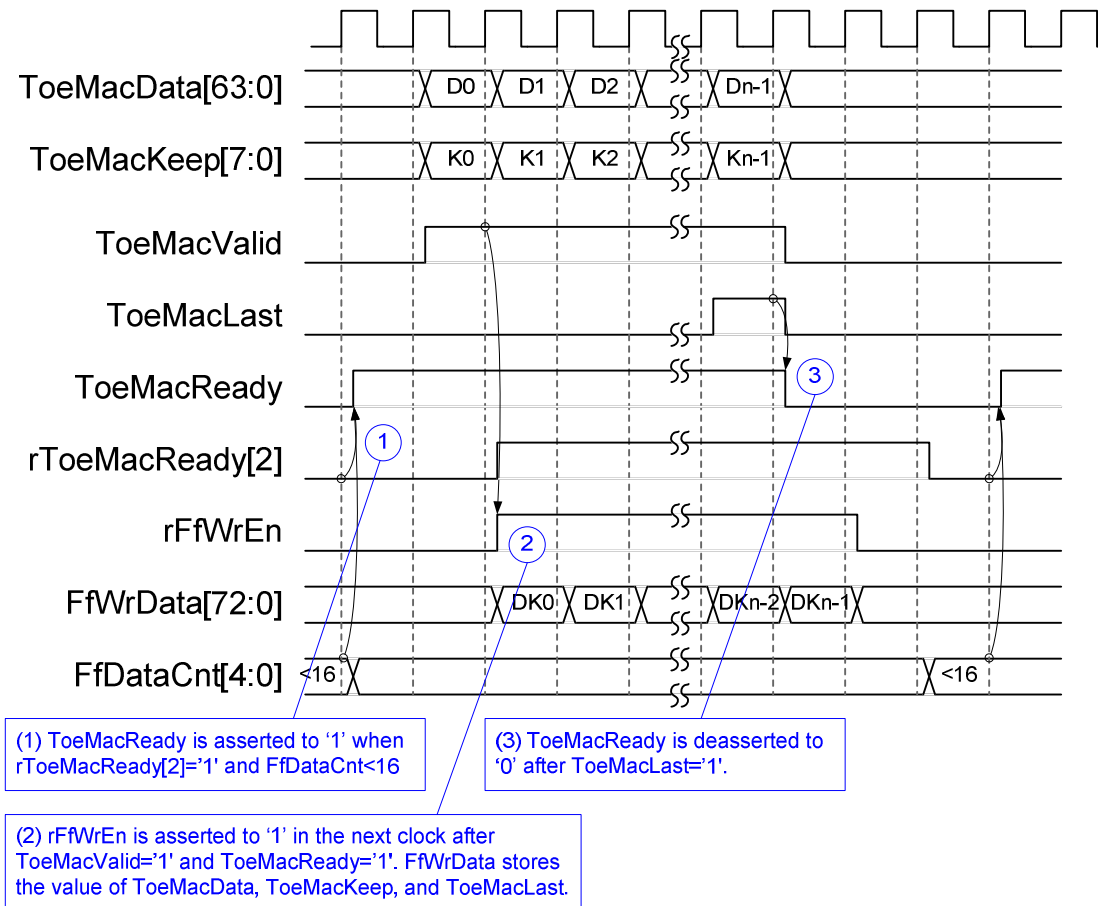


Figure 2-3 Write FIFO timing diagram of TenGMacIF

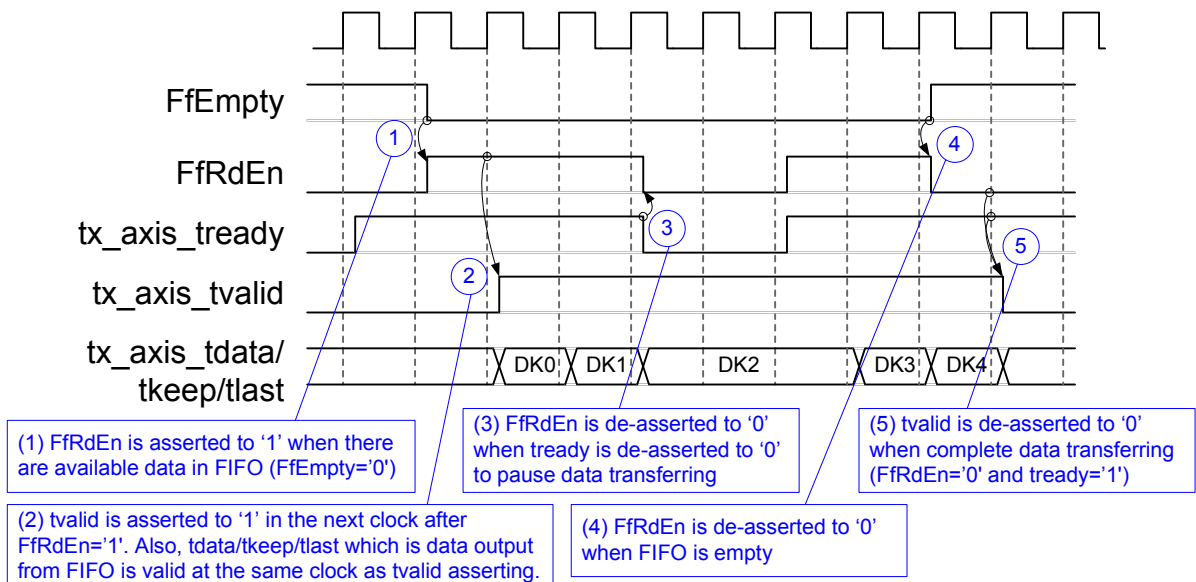


Figure 2-4 Read FIFO timing diagram of TenGMacIF

When data is available in FIFO (FfEmpty='0'), FfRdEn is asserted to '1' to transfer data from FIFO to Ethernet MAC. In the next clock, tx\_axis\_tvalid is asserted to '1' with the valid data for transferring to EMAC. When EMAC is not ready to receive data (tx\_axis\_tready='0'), FfRdEn is de-asserted to '0' to pause data transmission. When FIFO is empty, FfRdEn is de-asserted to '0'. After that, tx\_axis\_tvalid is de-asserted to '0' to complete data transferring.

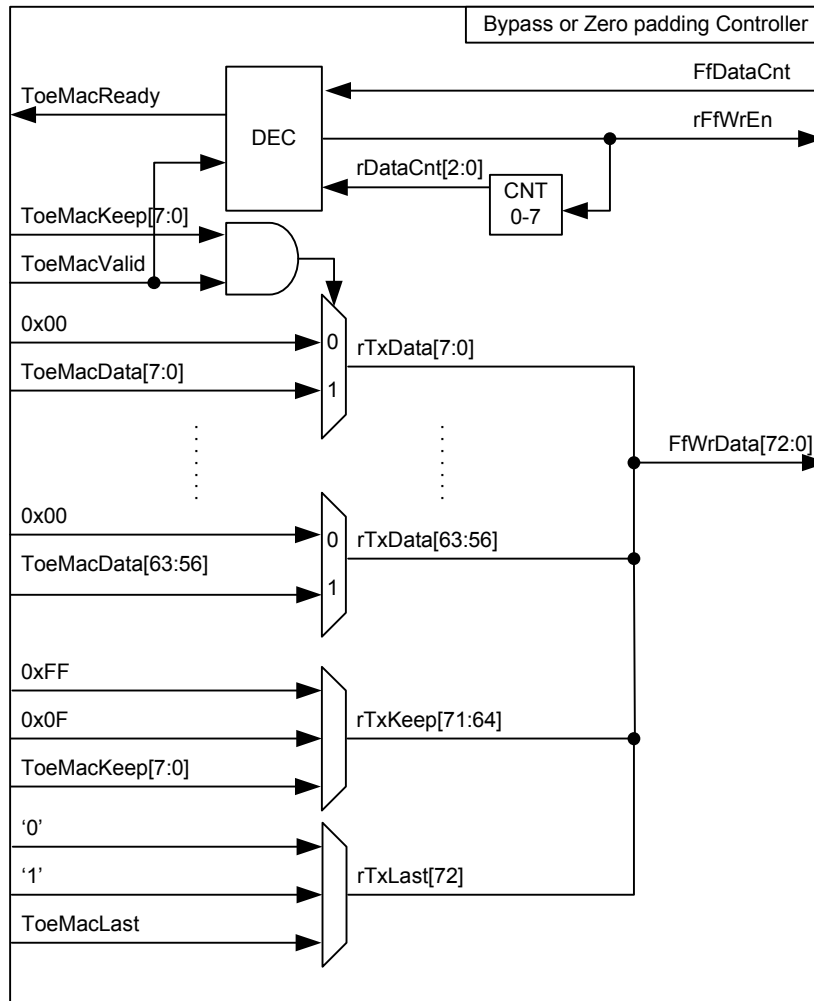


Figure 2-5 Data path of zero padding function

Figure 2-5 shows the logic to fill zero padding data when total packet size is less than 60 bytes when using 10G/25G Ethernet system. 3-bit data counter is designed to count data size whether data less than 60 bytes or not. rFfWrEn is asserted to '1' to fill zero padding data until data counter=7 (packet size is more than or equal to 60 bytes). rTxData is selected between data signal from TOE10G-IP or zero value for zero-padding. Each bit of ToeMacKeep is used to be byte valid of ToeMacData and used to select data to forward to FIFO to between ToeMacData or 0x00.

rTxKeep is fixed to 0xFF for QWord 0-6. For QWord 7 (last QWord for minimum size), rTxKeep is fixed to 0x0F to pass 60 bytes for zero-padding feature or bypass from ToeMacKeep for normal packet (the packet size is more than 60 byte). For QWord 8 or more, rTxKeep is forwarded from TOE10G-IP directly.

Similar to rTxKeep, rTxLast is fixed to '0' for QWord 0-6. rTxLast is asserted to '1' at QWord 7 for zero-padding feature. If packet size is more than 60 bytes, rTxLast will be forwarded from ToeMacLast.

## 2.4 LAXi2Reg

The hardware is connected to CPU through AXI4-Lite bus, similar to other CPU peripherals. The hardware registers are mapped to CPU memory address, as shown in Table 2-1. The control and status registers for CPU access are designed in LAXi2Reg.

LAXi2Reg connects to TOE10G-IP for both data path and control path. As shown in Figure 2-6, there are two clock domains applied in this block, i.e. 100 MHz (CpuClk) which is used to interface with CPU through AXI4-Lite bus and 156.25 MHz (MacClk) which is user clock domain for TOE10G-IP and EMAC.

AsyncAxiReg includes asynchronous circuit between 100 MHz and 156.25 MHz. More details of each hardware are described as follows.

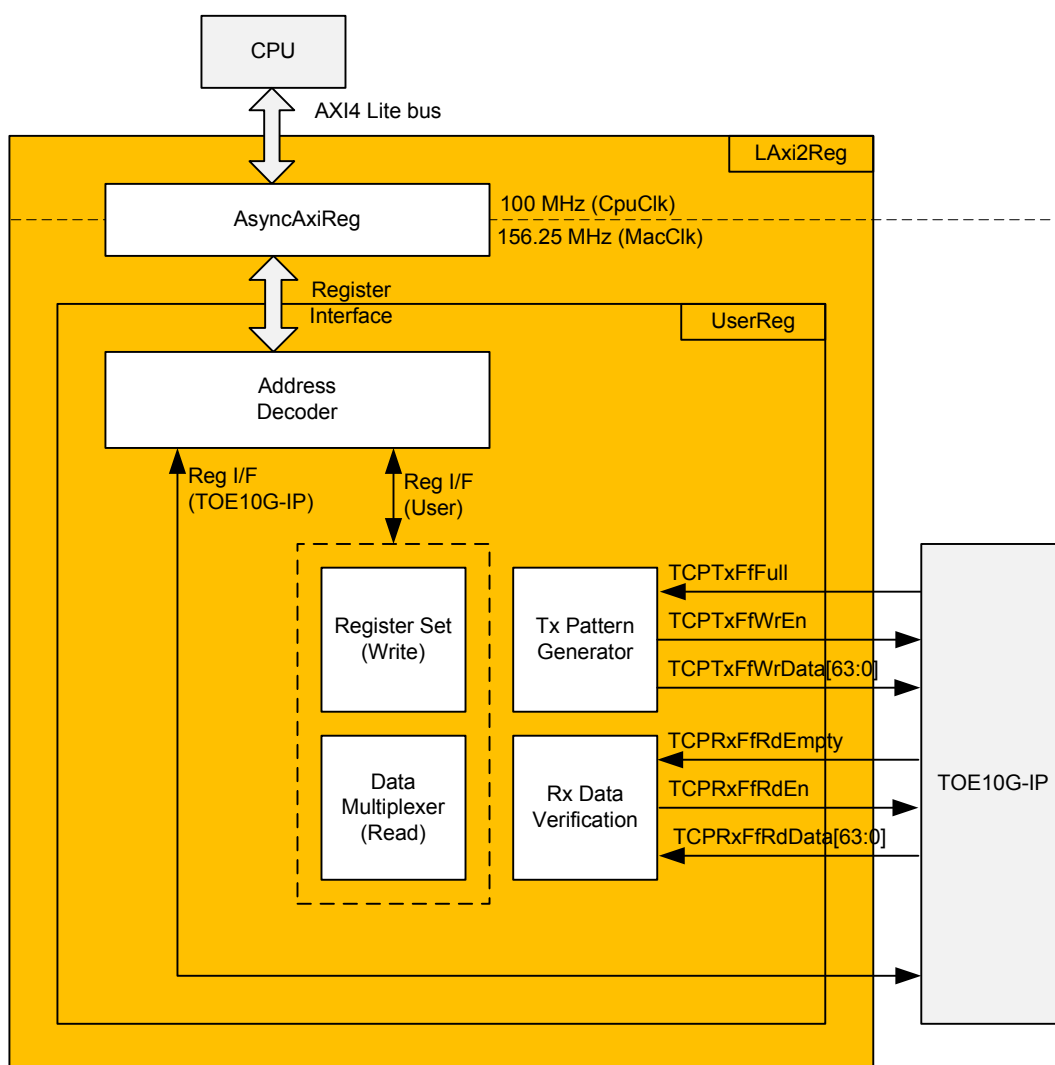


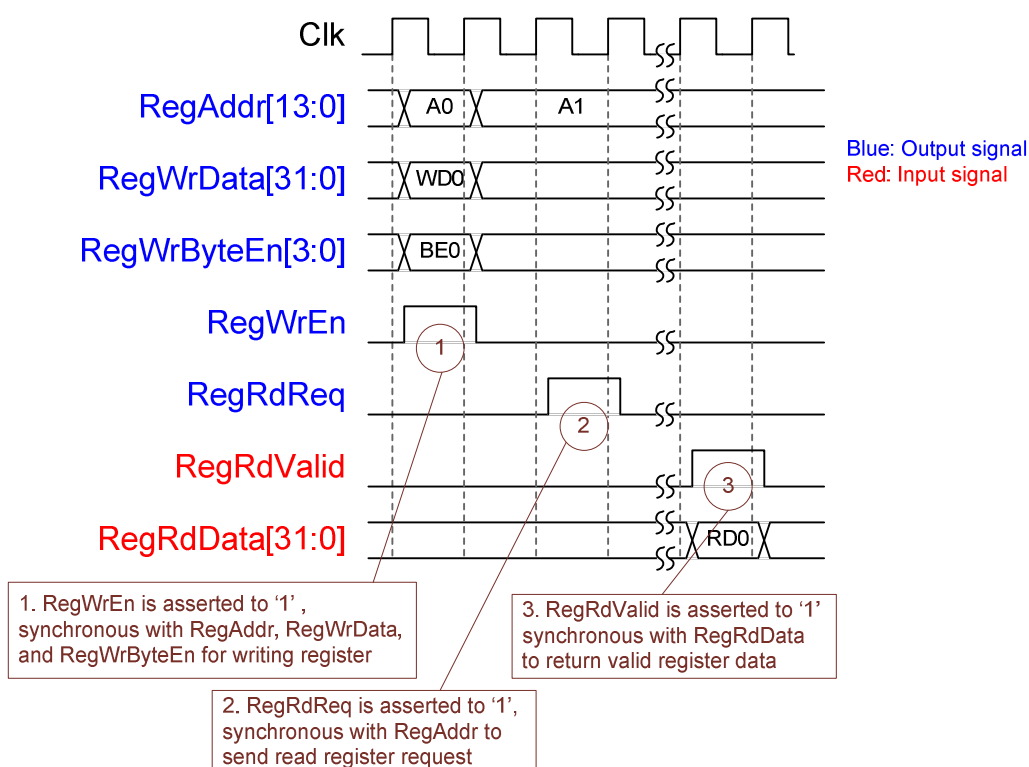
Figure 2-6 LAXi2Reg block diagram

### 2.4.1 AsyncAxiReg

This module is designed to convert signal interface of AXI4-Lite to be register interface. Also, it supports to convert clock domain from 100 MHz to be 156.25 MHz. Timing diagram of register interface is shown in Figure 2-7.

To write register, timing diagram is same as 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 byte enable of this access: bit[0] is write enable for RegWrData[7:0], bit[1] is used for RegWrData[15:8], ..., and bit[3] is used for RegWrData[31:24]).

To read register, AsyncAxiReg asserts RegRdReq='1' with the valid value of RegAddr (the register address in 32-bit unit). After that, the module waits until RegRdValid is asserted to '1' to get the read data through RegRdData signal.



**Figure 2-7 Register interface timing diagram**



## 2.4.2 UserReg

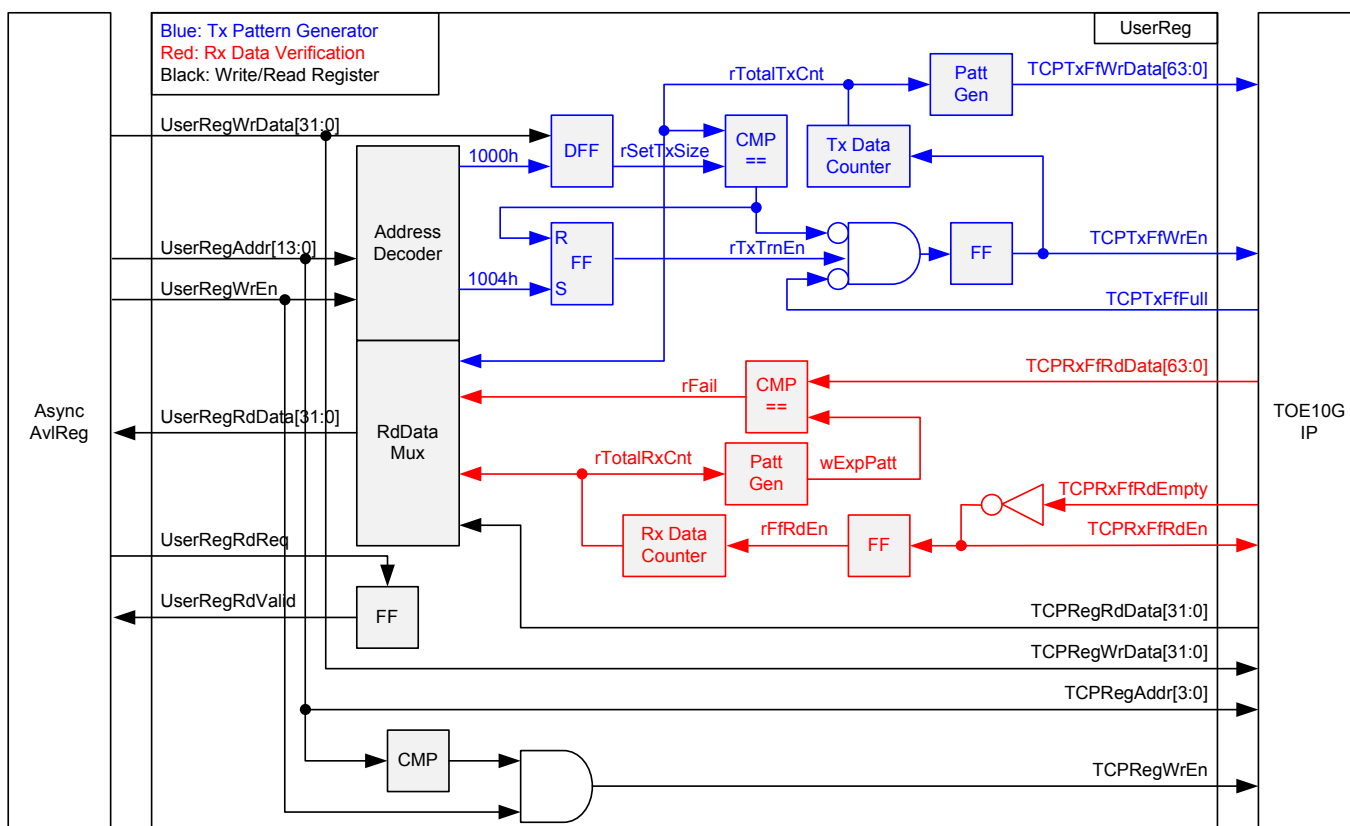


Figure 2-8 UserReg block diagram

Memory map of control and status signals inside UserReg module is shown in Table 2-1. 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).

To request write register, UserRegWrEn is asserted to '1' with the valid of UserRegAddr. The upper bits of UserRegAddr are forwarded to check whether the address is in TOE10G-IP range or not. If the address is in TOE10G-IP range, TCPRegWrEn will be asserted to '1'. For internal register of UserReg, UserRegWrData is loaded to internal register when the address is matched. For example, rSetTxSize is loaded by UserRegWrData when UserRegAddr=0x1000. UserRegWrByteEn signal is not used in this module, so CPU firmware needs to access the hardware register by using 32-bit pointer only.

For read request, UserRegRdReq is asserted to '1'. RdDataMux selects status signals from internal register or TOE10G-IP, and forwards to UserRegRdData in the next clock. To synchronous with UserRegRdData, RegRdValid is designed by using one D Flip-flop, input by RegRdReq signal.

The upper logic in blue color of Figure 2-8 is designed to generate test data to TOE10G-IP. rTxTrnEn is asserted to '1' when write register address is 1004h. When rTxTrnEn is '1', TCPTxFfWrEn is controlled by TCPTxFfFull. TCPTxFfWrEn is de-asserted to '0' when TCPTxFfFull is '1'. rTotalTxCnt is data counter to check total transfer data to TCPTxFf. rTotalTxCnt is also used to generate 32-bit increment data to TCPTxFfWrData signal. rTxTrnEn is de-asserted to '0' when total transfer size is equal to the set value (rSetTxSize).

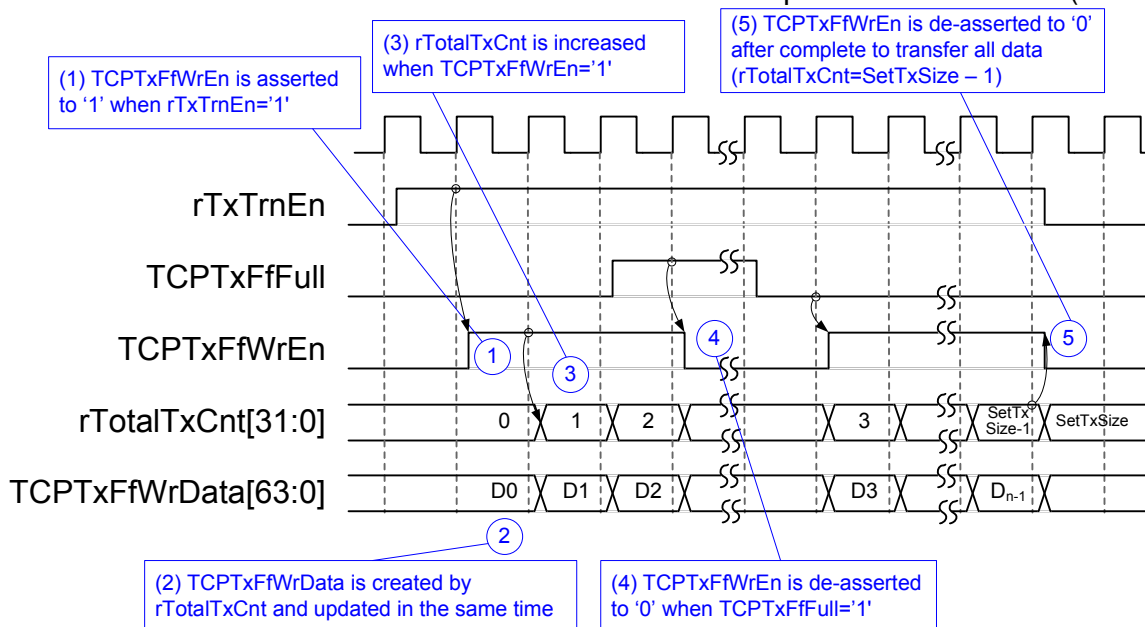


Figure 2-9 Tx Pattern Generator Timing diagram

The logic in red color of Figure 2-8 is designed to verify received data from TOE10G-IP. TCPRxFfRdEn is designed by using NOT logic of TCPRxFfRdEmpty. TCPRxFfRdData is valid in the next clock after asserting TCPRxFfRdEn to '1'. Read data (TCPRxFfRdData) is compared to expected pattern (wExpPatt) which is designed by rTotalRxCnt. rTotalRxCnt is data counter to check total transfer data from TCPRxFf. Similar to Tx path, expected pattern is 32-bit increment pattern. Fail flag (rFail) will be asserted to '1' if Read Data is not equal to expected pattern.

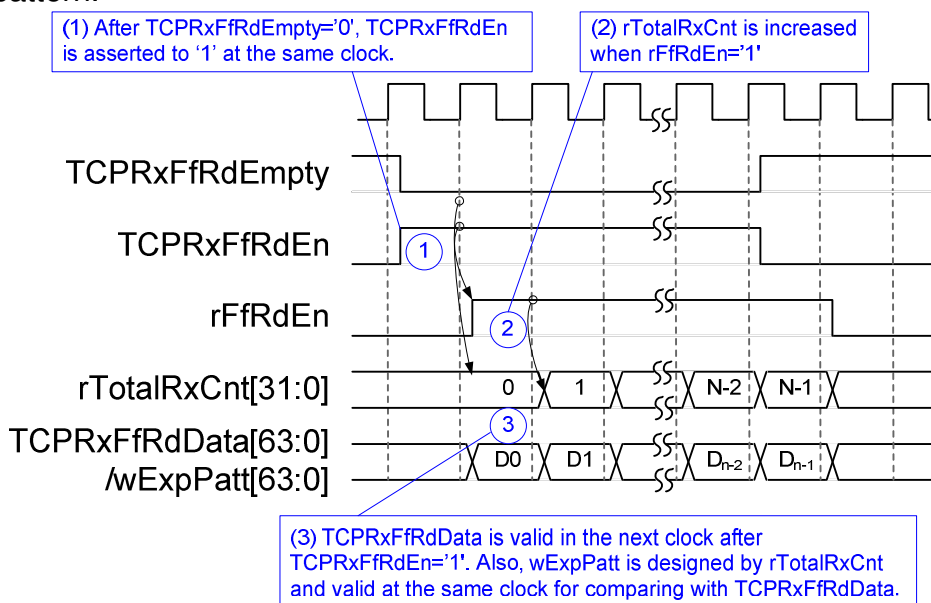


Figure 2-10 Rx Data Verification Timing diagram

**Table 2-1 Register map Definition**

Address Wr/Rd	Register Name (Label in the "toe10gip_demo.c")	Description
BA+0x0000 – BA+0x00FF: 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.

### 3 CPU Firmware Sequence

After FPGA boot-up, user must select the operation mode on FPGA to be client or server. The operation mode is the set value for TOE10\_SRV\_REG register. In client mode, FPGA sends ARP request to get the MAC address from the destination device during initialization sequence. In server mode, FPGA waits ARP request from the destination device and returns ARP reply during initialization sequence.

To run the test by using two FPGAs, the operation mode on each FPGA must be set to different value (one is client and another is server). In case of running FPGA with PC, it is recommended to set FPGA to client mode. It is easier for PC to return ARP reply after receiving ARP request than forcing PC to send ARP request.

In the firmware, there are two default parameters for each operation mode. The initialization sequence after system boot-up is 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 or inputs other keys to change some parameters. In case of changing parameters, the operation sequence is same as Reset IP which is described in topic 3.2.
- 3) CPU waits until TOE10G-IP completes initialization sequence (TOE10\_CMD\_REG[0]='0').
- 4) Main menu is displayed with five operations which are described in more details as follows.

#### 3.1 Show parameters

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

- 1) Read network parameter 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, 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 invalid input 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.

### 3.3 Send data test

Three user inputs are required to set 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 the input is invalid. During the test, 32-bit increment data is generated from the logic and sent to PC/FPGA. Test application on PC or verification module in FPGA verifies the received data. The operation is completed when total data are transferred from FPGA to PC/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), 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) Open connection following connection mode value.
  - a. For active open, CPU sets TOE10\_CMD\_REG=2 and monitors ConnOn status until it is equal to '1' (USER\_CMD\_REG[2]).
  - b. For passive open, CPU waits until connection is opened by PC/FPGA by monitoring ConnOn status = '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 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.

### 3.4 Receive data test

User sets total received size, selects 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/FPGA 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.

### 3.5 Full duplex test

This menu is designed to run full duplex test by transferring data between FPGA and PC/FPGA in both directions at the same time and same port number. 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).

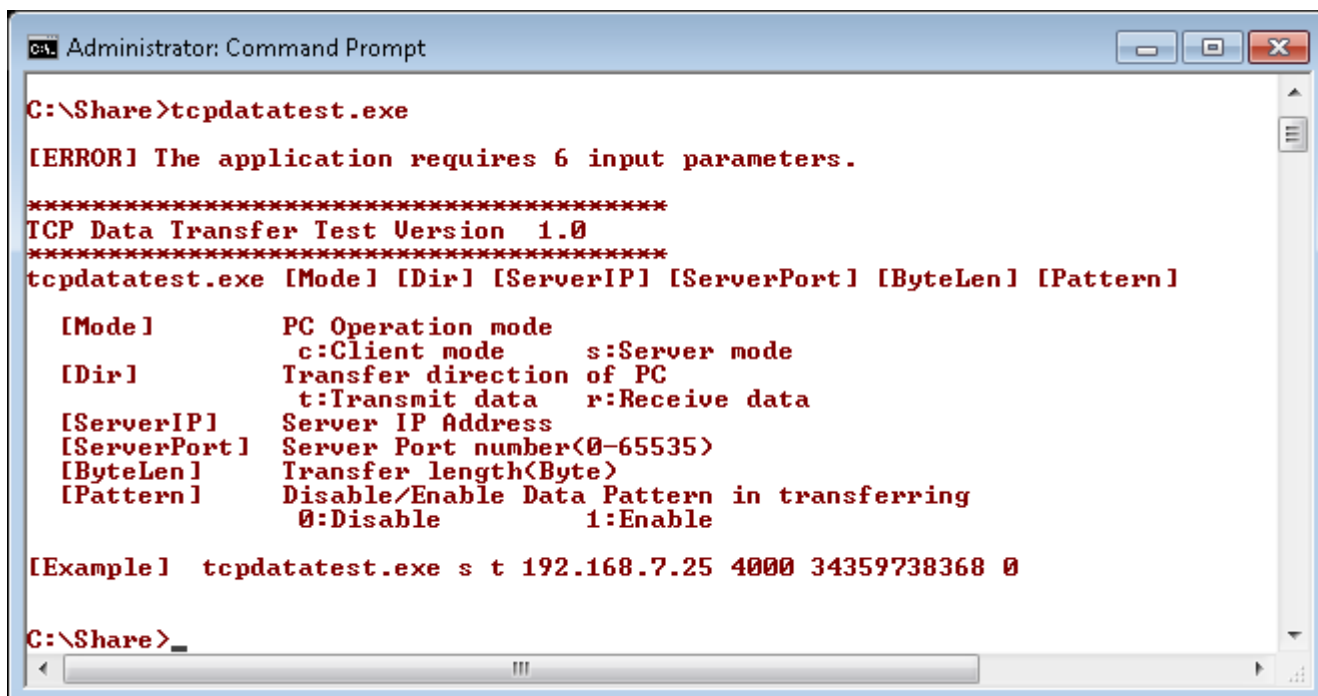
To test this mode by using PC and FPGA, transfer size for sending and receiving logic is fixed to maximum size (32 GB) which is equal to the size setting in “tcp\_client\_trx\_10G” application. Also, connection mode must be set to passive (server operation) to run with “tcp\_client\_trx\_10G” application.

The test runs in forever loop until user cancels operation on PC (input Ctrl+C) in case transferring data between PC and FPGA or user cancels operation on Serial console in case transferring data between FPGA and FPGA. The sequence of this test is as follows.

- 1) Receive total data size, packet size, data verification mode, and connection 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) Open connection following connection mode value (same as Step 4 of Send data test).
- 5) Set TOE10G-IP registers, i.e. packet size (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 in each loop except the last loop is set to align the packet size to achieve the best performance. Transfer size in the last loop is equal to the remaining size which may not be aligned to packet size.
  - b. Set send command to TOE10G-IP register (TOE10\_CMD\_REG=0).
  - c. 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) Close connection following connection mode value.
  - a. For active close, CPU waits until received transfer size is equal to set value. Then, set USER\_CMD\_REG=3 to close connection and wait until connection is closed (USER\_CMD\_REG[2]='0').
  - b. For passive close, CPU waits until connection is closed from FPGA/PC by monitoring ConnOn signal (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



```

Administrator: Command Prompt

C:\Share>tcpdatatest.exe

[ERROR] The application requires 6 input parameters.

*****
TCP Data Transfer Test Version 1.0
*****
tcpdatatest.exe [Mode] [Dir] [ServerIP] [ServerPort] [ByteLen] [Pattern]

[Mode]      PC Operation mode
             c:Client mode      s:Server mode
[Dir]       Transfer direction of PC
             t:Transmit data    r:Receive data
[ServerIP]  Server IP Address
[ServerPort] Server Port number<0-65535>
[ByteLen]   Transfer length(Byte)
[Pattern]   Disable/Enable Data Pattern in transferring
             0:Disable         1:Enable

[Example] tcpdatatest.exe s t 192.168.7.25 4000 34359738368 0

C:\Share>_

```

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 inputs 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.

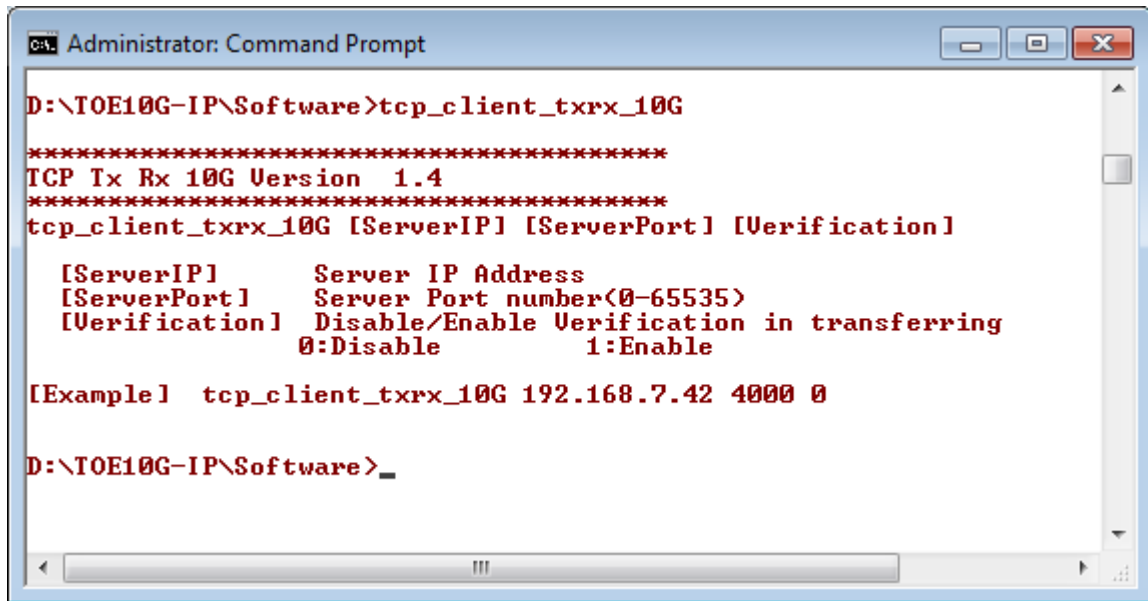
- 1) Allocate 1 MB memory to be send buffer.
- 2) Create socket and set properties of send buffer.
- 3) Create new connection to server by using IP address and port number from user.
- 4) Generate increment test pattern to send buffer when test pattern is enabled. Skip this step if dummy pattern is selected.
- 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



```

Administrator: Command Prompt

D:\TOE10G-IP\Software>tcp_client_txrx_10G

*****
TCP Tx Rx 10G Version 1.4
*****
tcp_client_txrx_10G [ServerIP] [ServerPort] [Verification]

[ServerIP]      Server IP Address
[ServerPort]    Server Port number(0-65535)
[Verification] Disable/Enable Verification in transferring
                0:Disable          1:Enable

[Example] tcp_client_txrx_10G 192.168.7.42 4000 0

D:\TOE10G-IP\Software>_

```

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 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
1.1	2-Apr-18	Support FPGA<->FPGA connection