

## TOE100G-IP with CPU reference design

Rev1.0 25-Feb-21

### 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.

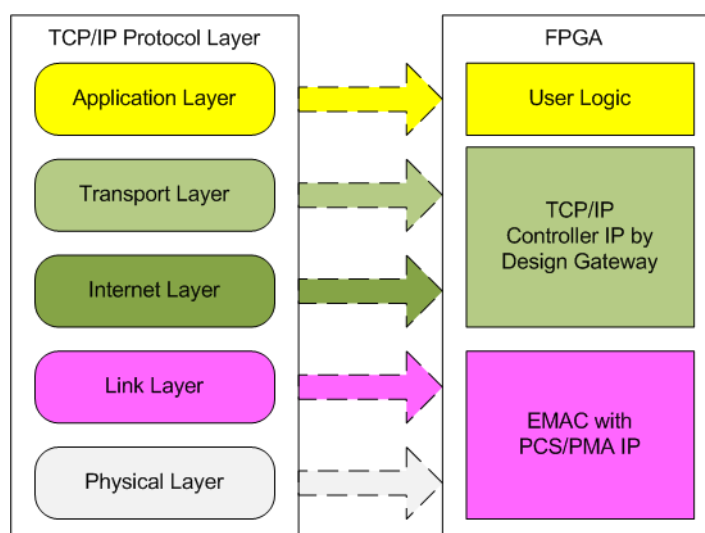


Figure 1-1 TCP/IP protocol layer

TOE100G-IP implements Transport and Internet layer of TCP/IP Protocol for building Ethernet packet from the user data which is TCP data to EMAC. If TCP data size is larger than a packet size, TOE100G-IP splits the data from the user to smaller size to fit in one packet. Then, the data is appended by TCP/IP header. On the other hand, the received Ethernet packet from EMAC is extracted by TOE100G-IP. The header of the packet is verified. If the header is valid, TCP data is forwarded to the user logic. Otherwise, the packet is rejected.

The lower layer protocols are implemented by 100G Ethernet IP, including EMAC and PCS/PMA logic. The reference design uses 100G Ethernet Subsystem, provided by Xilinx.

The reference design provides the evaluation system which includes simple user logic to transfer data by using TOE100G-IP. TOE100G-IP is designed to transfer data with PC or another TOE100G-IP run on another FPGA board. To run with PC, the test application is called on PC to send and verify TCP data via Ethernet connection at very high-speed rate. Two test applications are specially designed, "tcpdatatest" for running half-duplex test (send or receive data test) and "tcp\_client\_txrx\_40G" for running full-duplex test (send and receive data at the same time).

To allow the user controlling the test parameters and the operation of TOE100G-IP demo via UART, the CPU system is included. It is easy for the user to set the test parameters and monitor the current status on the console. The firmware on CPU is built by using bare-metal OS. More details of the demo are described as follows.

## 2 Hardware overview

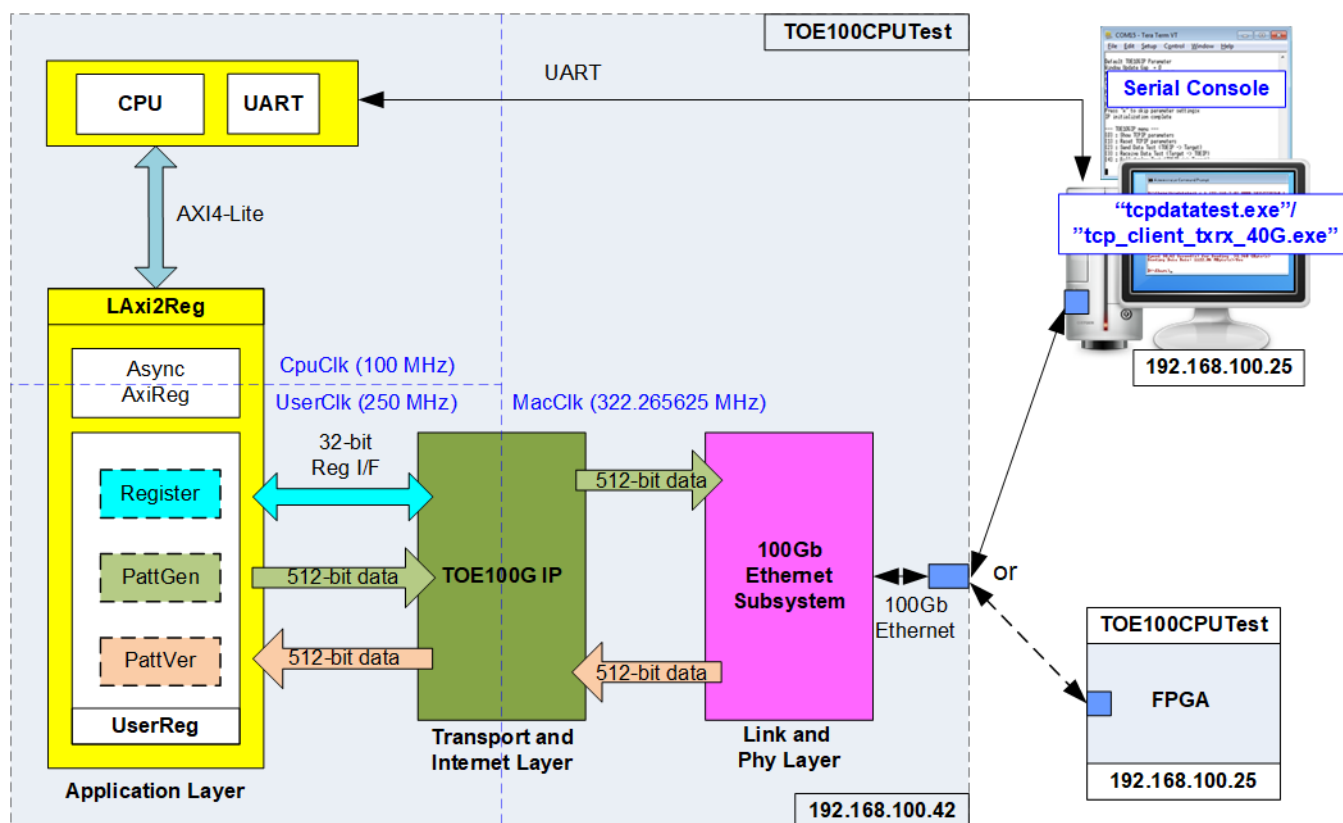


Figure 2-1 Demo block diagram

In test environment, two devices are used for 100Gb Ethernet transferring. First device runs in client mode and another device runs in server mode. In the demo, the client device is TOE100G-IP on FPGA board while the server device can be TOE100G-IP on FPGA board or PC, as shown in Figure 2-1. When using PC, the test applications (tcpdatatest and tcp\_client\_trrx\_40G) must be run on PC to transfer data with TOE100G-IP within FPGA.

In FPGA system, TOE100G-IP connects with 100G Ethernet Subsystem to implement all TCP/IP layers. User interface of TOE100G-IP connects to UserReg within AsyncAxiReg which consists of Register file for interfacing with Register interface, PattGen for sending test data via Tx FIFO interface, and PattVer for verifying test data via Rx FIFO interface. Register files of UserReg are controlled by CPU firmware through AXI4-Lite bus.

There are three clock domains in the design, i.e., CpuClk which is the clock for running the CPU system, MacClk which is the clock output from 100Gb Ethernet Subsystem to interface with 100Gb Ethernet Subsystem, and UserClk which is the clock for running user logic of TOE100G-IP. In real system, the user can change the frequency of CpuClk and UserClk. According to TOE100G-IP datasheet, clock frequency of UserClk must be more than or equal to 240 MHz.

AsyncAxiReg is designed to support asynchronous signals between CpuClk and UserClk. More details of each module inside the TOE100CPUtest are described as follows.

## 2.1 100Gb Ethernet Subsystem (100G BASE-SR)

This module implements EMAC and PCS/PMA logic of 100G Ethernet. The physical interface on FPGA board can be applied by QSFP28 or 4xSFP28 for 100Gb BASE-SR standard. The user interface for connecting with EMAC is 512-bit AXI4-stream interface running at 322.265625 MHz. This IP core can be created by using IP wizard in Vivado tools. More details of the core are described in the following link.

PG203: UltraScale+ Devices Integrated 100G Ethernet Subsystem Product Guide  
[https://www.xilinx.com/products/intellectual-property/cmac\\_usplus.html](https://www.xilinx.com/products/intellectual-property/cmac_usplus.html)

## 2.2 TOE100G-IP

TOE100G-IP implements TCP/IP stack and offload engine. User interface has two signal groups, i.e., control signals and data signals. Register interface is applied to set control registers and monitor status signals. Data signals are accessed by using FIFO interface. The interface with 100G EMAC is 512-bit AXI4 interface.

More details are described in datasheet.  
[https://dgway.com/products/IP/TOE100G-IP/dg\\_toe100gip\\_data\\_sheet\\_xilinx.pdf](https://dgway.com/products/IP/TOE100G-IP/dg_toe100gip_data_sheet_xilinx.pdf)

### 2.3 CPU and Peripherals

32-bit AXI4-Lite is applied to be the bus interface for the 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 to each peripheral for accessing one peripheral at a time.

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 supporting CPU writing and reading. LAXi2Reg module is designed to connect the CPU system as shown in Figure 2-2.

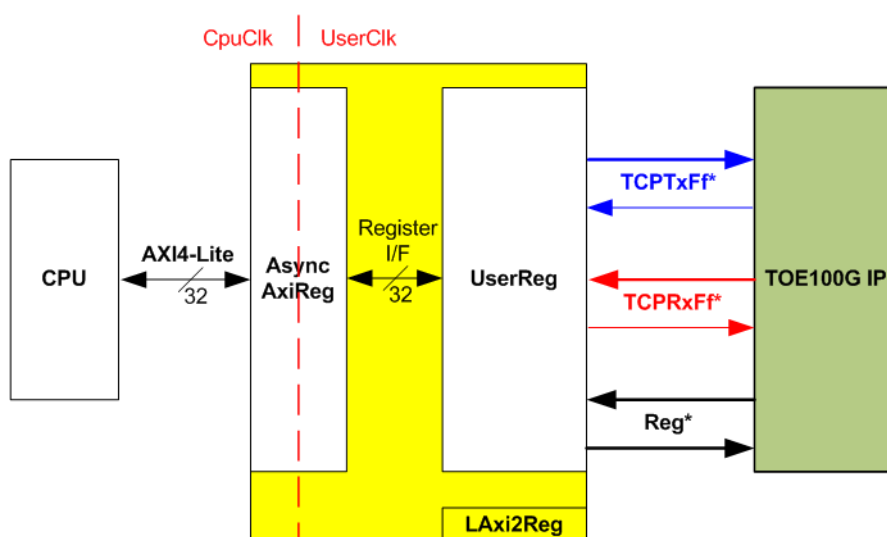


Figure 2-2 LAXi2Reg block diagram

LAXi2Reg consists of AsyncAxiReg and UserReg. AsyncAxiReg is designed to convert the AXI4-Lite signals to be the simple register interface which has 32-bit data bus size (similar to AXI4-Lite data bus size). Additionally, AsyncAxiReg includes asynchronous logic to support clock domain crossing between CpuClk domain and UserClk domain.

UserReg includes the register file of the parameters and the status signals of test logics. Both data interface and control interface of TOE100G-IP are also connected to UserReg. More details of AsyncAxiReg and UserReg are described as follows.

### 2.3.1 AsyncAxiReg

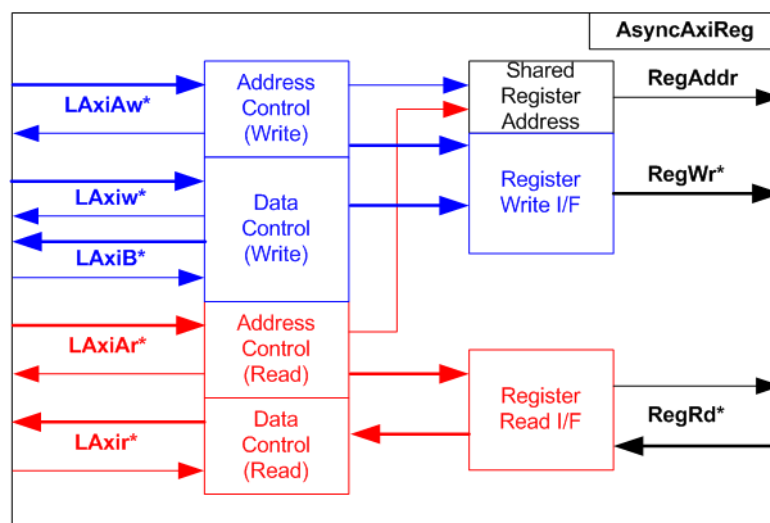


Figure 2-3 AsyncAxiReg interface

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](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 separately. 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 2-3. Write control I/F and Write data I/F of AXI4-Lite bus are latched and transferred to be Write register interface with clock-crossing registers. Similarly, Read control I/F of AXI4-Lite bus are latched and transferred to be Read register interface with clock-crossing registers. While the returned data from Register Read I/F is transferred to AXI4-Lite bus by using clock-crossing registers. In register interface, RegAddr is shared signal for write and read access, so it loads the address from LAXiAw for write access or LAXiAr for read access.

The simple register interface is compatible with single-port RAM interface for write transaction. The read transaction of the register interface is slightly modified from RAM interface by adding RdReq and RdValid signals for controlling read latency time. 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 2-4.

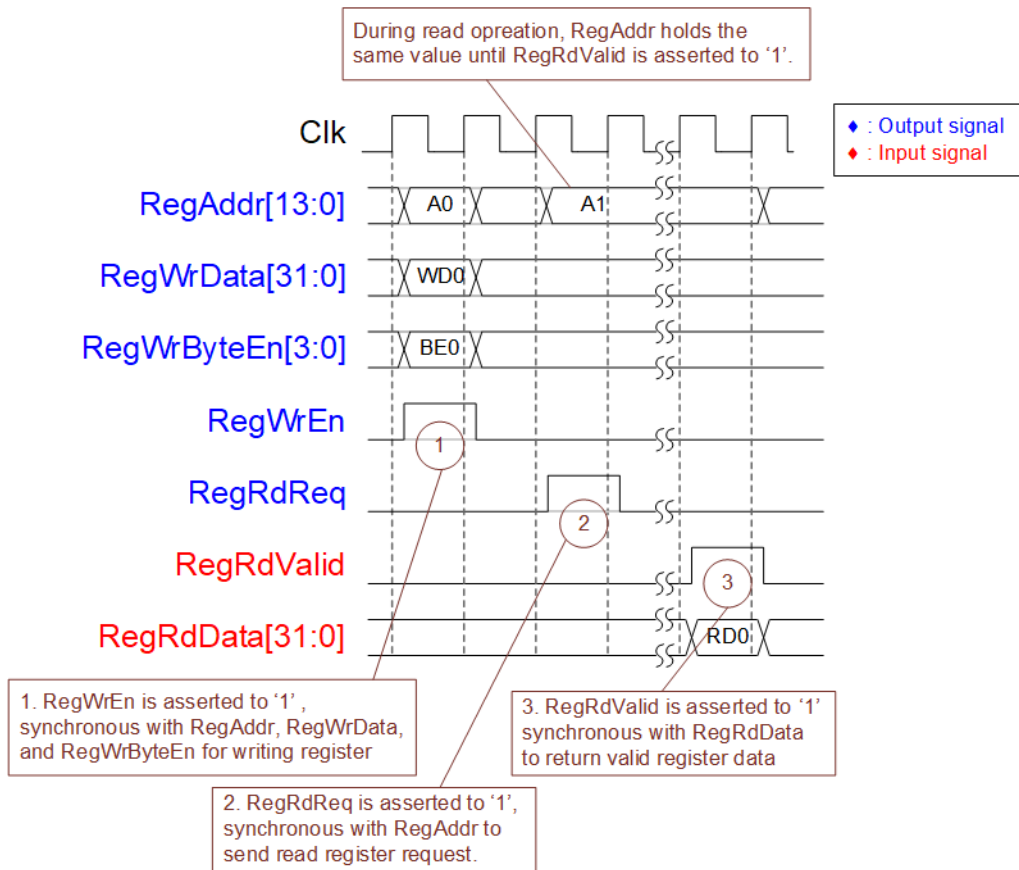


Figure 2-4 Register interface timing diagram

- 1) To write register, the timing diagram is similar to single-port 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 bits to be the byte data valid. Bit[0], [1], [2], and [3] are equal to '1' when RegWrData[7:0], [15:8], [23:16], and [31:24] are valid respectively.
- 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. During read operation, the address value (RegAddr) does not change the value until RegRdValid is asserted to '1'. So, the address can be used for selecting the returned data by using multiple layers of multiplexer.
- 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.

### 2.3.2 UserReg

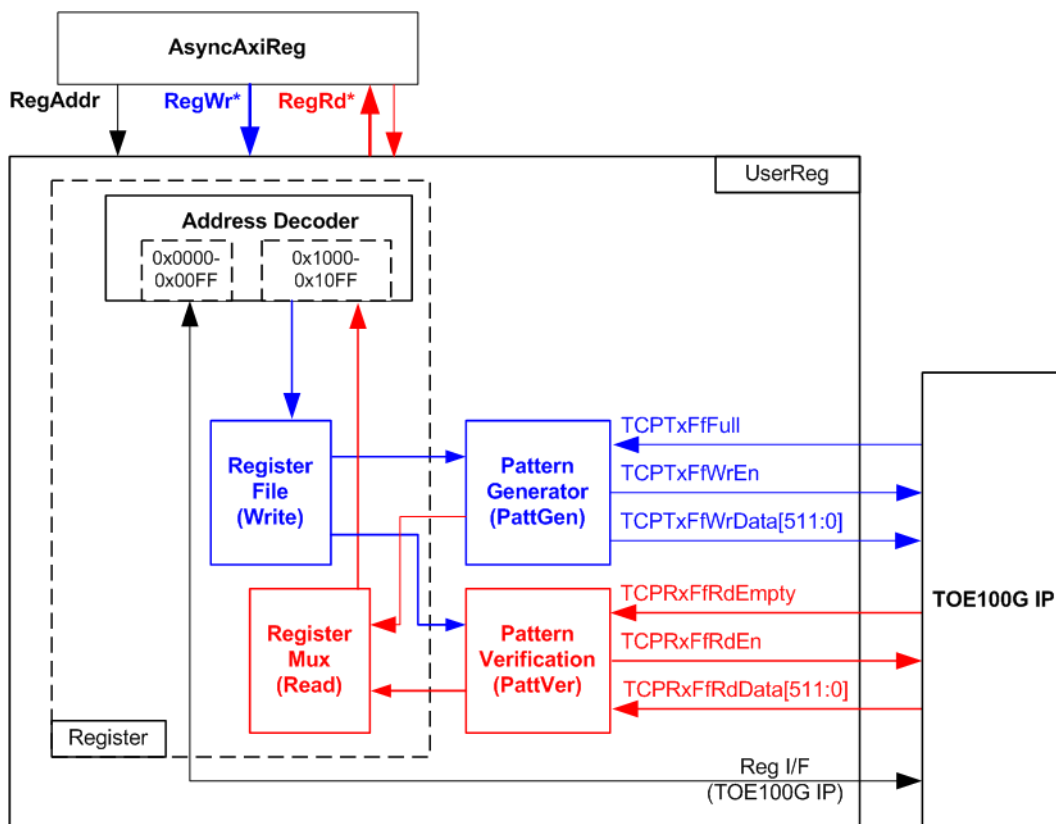


Figure 2-5 UserReg block diagram

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 512-bit test data to TOE100G-IP following FIFO interface standard. Pattern verification block is designed to read and verify 512-bit data from TOE100G-IP following FIFO interface standard. More details of each block are described as follows.

#### Register Block

The address range, mapped to UserReg, is split into two areas, i.e., TOE100G-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 write hardware registers, the CPU must use 32-bit pointer to place 32-bit valid value on the write data bus.

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 are shown in Table 2-1



**Table 2-1 Register map Definition**

Address	Register Name	Description
Wr/Rd	(Label in the "toe100gtest.c")	
BA+0x0000 – BA+0x00FF: TOE100G-IP Register Area More details of each register are described in TOE100G-IP datasheet.		
BA+0x0000	TOE_RST_REG	Mapped to RST register within TOE100G-IP
BA+0x0004	TOE_CMD_REG	Mapped to CMD register within TOE100G-IP
BA+0x0008	TOE_SML_REG	Mapped to SML register within TOE100G-IP
BA+0x000C	TOE_SMH_REG	Mapped to SMH register within TOE100G-IP
BA+0x0010	TOE_DIP_REG	Mapped to DIP register within TOE100G-IP
BA+0x0014	TOE_SIP_REG	Mapped to SIP register within TOE100G-IP
BA+0x0018	TOE_DPN_REG	Mapped to DPN register within TOE100G-IP
BA+0x001C	TOE_SPN_REG	Mapped to SPN register within TOE100G-IP
BA+0x0020	TOE_TDL_REG	Mapped to TDL register within TOE100G-IP
BA+0x0024	TOE_TMO_REG	Mapped to TMO register within TOE100G-IP
BA+0x0028	TOE_PKL_REG	Mapped to PKL register within TOE100G-IP
BA+0x002C	TOE_PSH_REG	Mapped to PSH register within TOE100G-IP
BA+0x0030	TOE_WIN_REG	Mapped to WIN register within TOE100G-IP
BA+0x0034	TOE_ETL_REG	Mapped to ETL register within TOE100G-IP
BA+0x0038	TOE_SRV_REG	Mapped to SRV register within TOE100G-IP
BA+0x003C	TOE_VER_REG	Mapped to VER register within TOE100G-IP
BA+0x0040	TOE_DML_REG	Mapped to DML register within TOE100G-IP
BA+0x0044	TOE_DMH_REG	Mapped to DMH register within TOE100G-IP
BA+0x1000 – BA+0x10FF: UserReg control/status		
BA+0x1000	Total transmit length	Wr [31:0] – Total transmit size in 512-bit unit.
Wr/Rd	(USER_TXLEN_REG)	Valid from 1-0xFFFFFFFF. Rd [31:0] – Current transmit size in 512-bit unit. The value is cleared to 0 when USER_CMD_REG is written by user.
BA+0x1004	User Command	Wr
Wr/Rd	(USER_CMD_REG)	[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 TOE100G-IP
BA+0x1008	User Reset	Wr
Wr/Rd	(USER_RST_REG)	[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	FIFO status	Rd[5:0] - Mapped to TCPRxFfLastRdCnt signal of TOE100G-IP
Rd	(USER_FFSTS_REG)	[15:6] - Mapped to TCPRxFfRdCnt signal of TOE100G-IP [24] - Mapped to TCPTxFfFull signal of TOE100G-IP
BA+0x1010	Total receive length	Rd[31:0] – Current receive size in 512-bit unit
Rd	(USER_RXLEN_REG)	The value is cleared to 0 when USER_CMD_REG is written by user.
BA+0x1080	EMAC IP version	Rd[31:0] – Mapped to IPVersion output from DG EMAC-IP when the system
Rd	(EMAC_VER_REG)	integrates DG EMAC-IP. In this demo, it is equal to 0.



### Pattern Generator

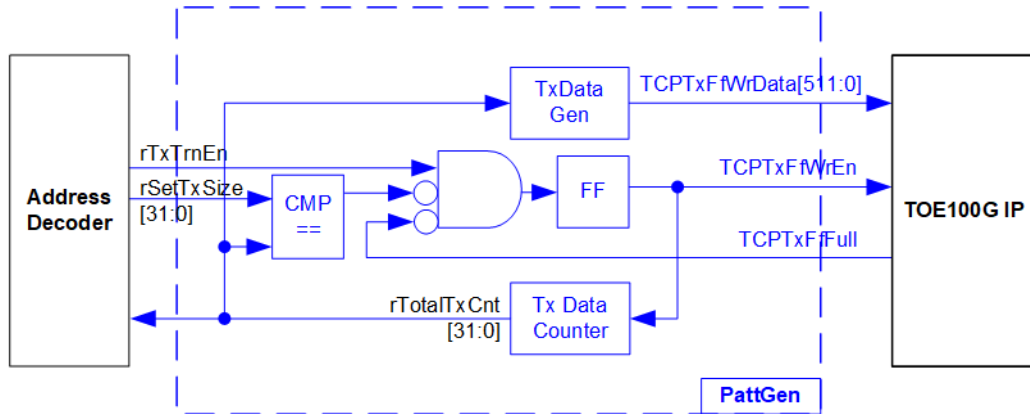


Figure 2-6 PattGen block

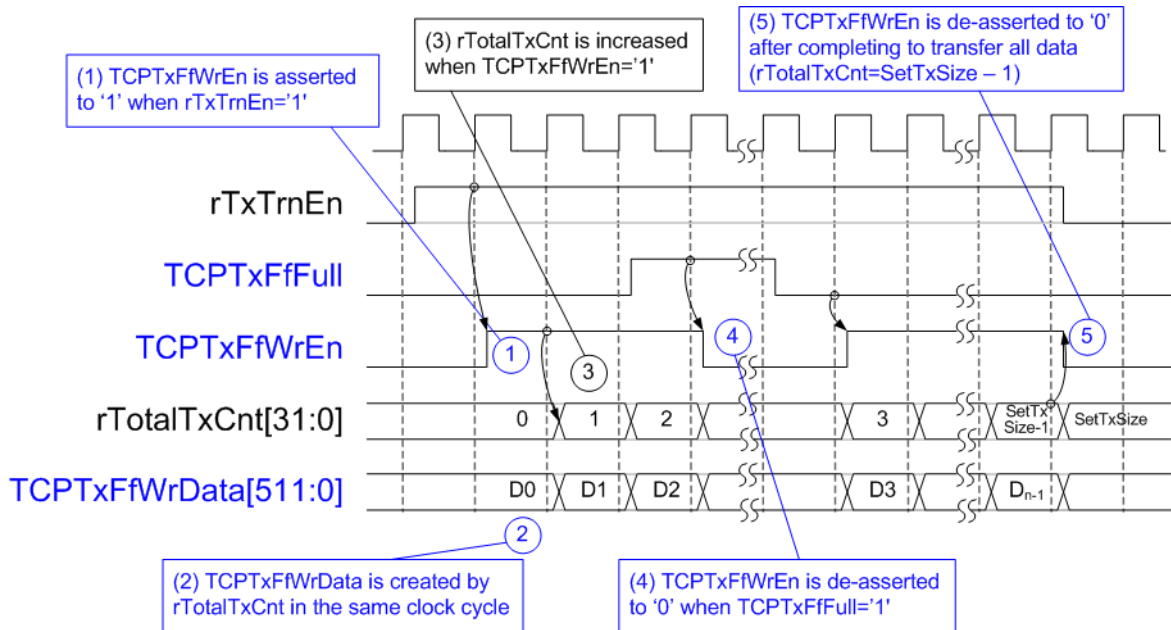
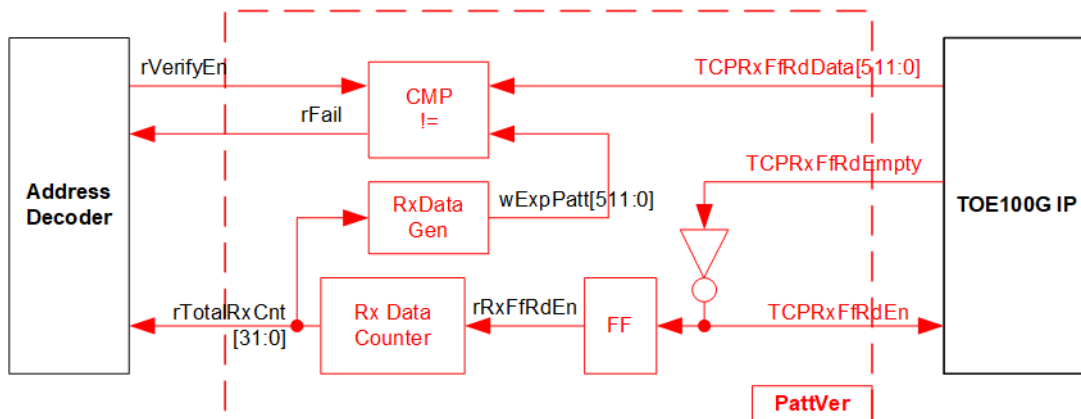


Figure 2-7 PattGen timing diagram

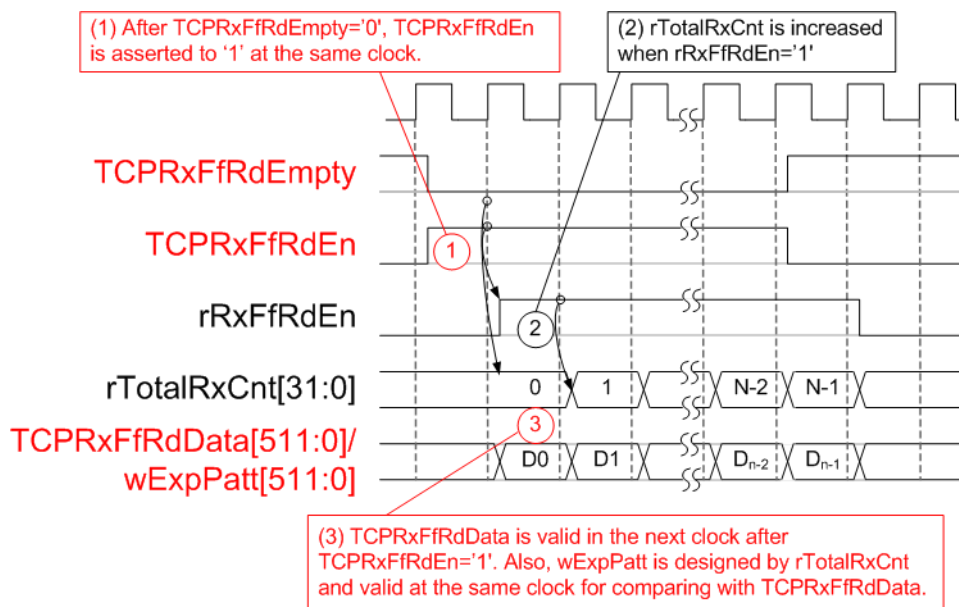
Figure 2-6 shows the details of PattGen which generates test data to TOE100G-IP. Timing diagram to show the relation of each logic is displayed in Figure 2-7.

To start PattGen operation, the user sets USER\_CMD\_REG[0]='0' and then rTxTrnEn is asserted to '1'. When rTxTrnEn is '1', TCPTxFWrEn is controlled by TCPTxFfFull. TCPTxFWrEn is de-asserted to '0' when TCPTxFfFull is '1'. rTotalTxCnt is the data counter to check total transmitted data to TOE100G-IP. Also, rTotalTxCnt is used to generate 32-bit incremental data to TCPTxFWrData signal. After all data is transferring completely, equal to rSetTxSize, rTxTrnEn is de-asserted to '0'.

Pattern Verification



**Figure 2-8 PattVer block**



**Figure 2-9 PattVer Timing diagram**

Figure 2-8 shows the details of PattVer logic for reading the data from TOE100G-IP with or without data verification, controlled by rVerifyEn flag which is set by the user. Timing diagram of the logic is displayed in Figure 2-9.

When rVerifyEn is set to '1', data comparison is enabled to compare read data (TCPRxFfRdData) with the expected pattern (wExpPatt). If 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, one clock latency of TCPRxFfRdEn, is applied to be counter enable of rTotalRxCnt, counting total received size. Also, rTotalRxCnt is used to generate wExpPatt.

### 3 CPU Firmware on FPGA

After FPGA boot-up, 100G 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 initialization mode of TOE100G-IP to be Client, Server, or Fixed-MAC.

- When testing by FPGA and PC, it is recommended to run TOE100G-IP in Client mode. After PC receives ARP request from TOE100G-IP, PC can return ARP reply. It is not simple to force PC sending ARP request to complete FPGA initialization in Server mode.
- When testing by two FPGAs, the initialization mode on two FPGAs must be following rules. If the first board sets Server mode, another board must be set to Client mode. Otherwise, the board can be set to Client or Fixed-MAC mode.

After receiving the mode from the user, the default parameters in the selected mode are displayed on the console. The user can select to complete the initialization by using default parameters or updating parameters. The example when the system is initialized in Client mode by using default parameters is shown in Figure 3-1.

```

+++ TOE100GIP with CPU Demo [IPVer = 1.0] +++
Input mode : [0] Client [1] Server [2] Fixed MAC => 0
+++ Current Network Parameter +++
Window Update Gap = 0
Mode = CLIENT
FPGA MAC address = 0x000102030405
FPGA IP = 192.168.100.42
FPGA port number = 60000
Target IP = 192.168.100.25
Target port number = 60001
Press 'x' to skip parameter setting: x
IP initialization complete
--- TOE100G-IP menu ---
[0] : Display TCPIP parameters
[1] : Reset TCPIP parameters
[2] : Send Data Test <TOEIP -> Target>
[3] : Receive Data Test <Target -> TOEIP>
[4] : Full duplex Test <TOEIP <-> Target>

```

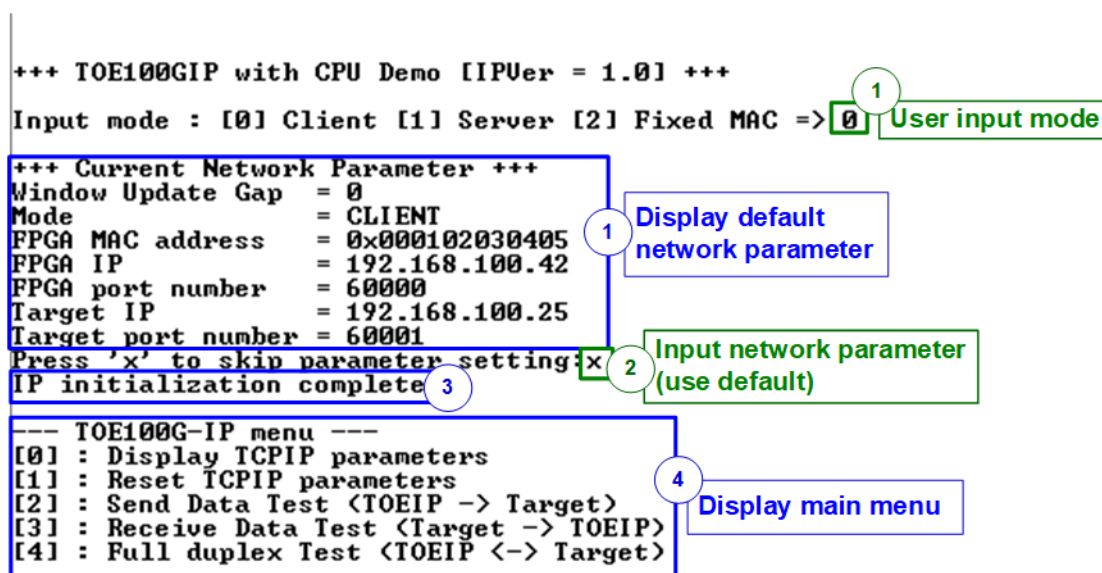


Figure 3-1 System initialization in Client mode by using default parameters

There are four steps to complete initialization process as follows.

- 1) CPU receives the initialization mode and then displays default parameters of the selected mode on the console.
- 2) User inputs 'x' to complete initialization process 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 TOE100G-IP finishing initialization process (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 Display parameters

This menu is used to show current parameters of TOE100G-IP, i.e., the initialization mode, Windows update threshold, source MAC address, destination IP address, source IP address, destination port, source port, and destination MAC address (when using Fixed MAC mode). 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 TOE100G-IP parameters such as IP address and source port number. After setting updated parameter to TOE100G-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 initialization mode from user and confirm that input is valid. If initialization mode is changed, the latest parameter set of new mode is displayed on the console.
- 3) Receive remaining input parameters from user and check input whether valid or not. If the input is invalid, the parameter is not updated.
- 4) Reset PattGen and PattVer logic by sending reset to user logic (USER\_RST\_REG[0]='1').
- 5) Force reset to IP by setting TOE\_RST\_REG[0]='1'.
- 6) Set all parameters to TOE100G-IP register such as TOE\_SML\_REG and TOE\_DIP\_REG.
- 7) De-assert IP reset by setting TOE\_RST\_REG[0]='0'. After that, TOE100G-IP starts the initialization process.
- 8) 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/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/FPGA. The sequence of the test is as follows.

- 1) Receive transfer size, packet size, and connection mode from user and verify if all inputs are 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 starts sending data to TOE100G-IP.
- 3) Display recommended parameters of test application on PC by reading current system parameters.
- 4) Open connection following connection mode value.
  - a. For active open, CPU sets TOE\_CMD\_REG=2 (Open port) and waits until ConnOn status (USER\_CMD\_REG[2]) is equal to '1'.
  - b. For passive open, CPU waits until connection is opened by another device (PC or FPGA). ConnOn status (USER\_CMD\_REG[2]) is monitored until it is equal to '1'.
- 5) Set packet size to TOE100G-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 TOE100G-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 TOE100G-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 transmit size from user logic (USER\_TXLEN\_REG) and displays the results on the console every second.
- 6) Set close connection command to TOE100G-IP register (TOE\_CMD\_REG=3).
- 7) Calculate performance and show test result on the console.

### 3.4 Receive data test

User sets total receive 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 if some inputs are invalid. During the test, 32-bit incremental data is generated to verify the received data from another device (PC or 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 the initial value of test pattern (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 (similar to Step 4 of Send data test).
- 5) Wait until connection is closed by another device (PC or FPGA). Connon status (USER\_CMD\_REG[2]) is monitored until it is equal to '0'. During monitoring Connon, CPU reads current receive size from user logic (USER\_RXLEN\_REG) and displays the results on the console every second.
- 6) Wait until all data are read by user logic completely by checking FIFO status (USER\_FFSTS\_REG[15:0]=0).
- 7) Compare receive length of user logic (USER\_RXLEN\_REG) with the set value from user. If all data is completely received, CPU checks verification result by reading USER\_CMD\_REG[1] ('0': normal, '1': error). If some errors are detected, the error message is displayed.
- 8) 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 another device (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, the transfer size set on FPGA must be matched to the size set 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. The operation can be cancelled by entering any keys on FPGA console and then entering Ctrl+C on PC 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 all inputs are valid.
- 2) Display the recommended parameters of test application run on PC from the current system parameters.
- 3) Set UserReg registers, i.e., transfer size (USER\_TXLEN\_REG), reset flag to clear the initial value of test pattern (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 (similar to Step 4 of Send data test).
- 5) Set packet size to TOE100G-IP registers (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 TOE100G-IP register (TOE\_CMD\_REG=0).
  - c. Wait until send command is completed by monitoring busy flag (TOE\_CMD\_REG[0]='0'). During monitoring busy flag, CPU reads current transmit size and receive 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 receive 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 the connection is closed by another device (PC or FPGA). ConnOn status (USER\_CMD\_REG[2]) is monitored until it is equal to '0'.
- 7) Check the result and the error (similar to 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 TOE100G-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	Ask the user if the parameters are updated. After that, set the parameters to TOE100G-IP registers from global parameters. After reset is de-asserted, it waits until TOE100G-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, Target port number, and Target MAC address (when run in Fixed MAC mode). If the input is valid, the parameter is updated. Otherwise, the value does not change. After receiving all parameters, the current value of all 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 transmit size and receive 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 set to TOE100G-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 transmit size and total receive 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 in MB/s unit.

int toe_rcv_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

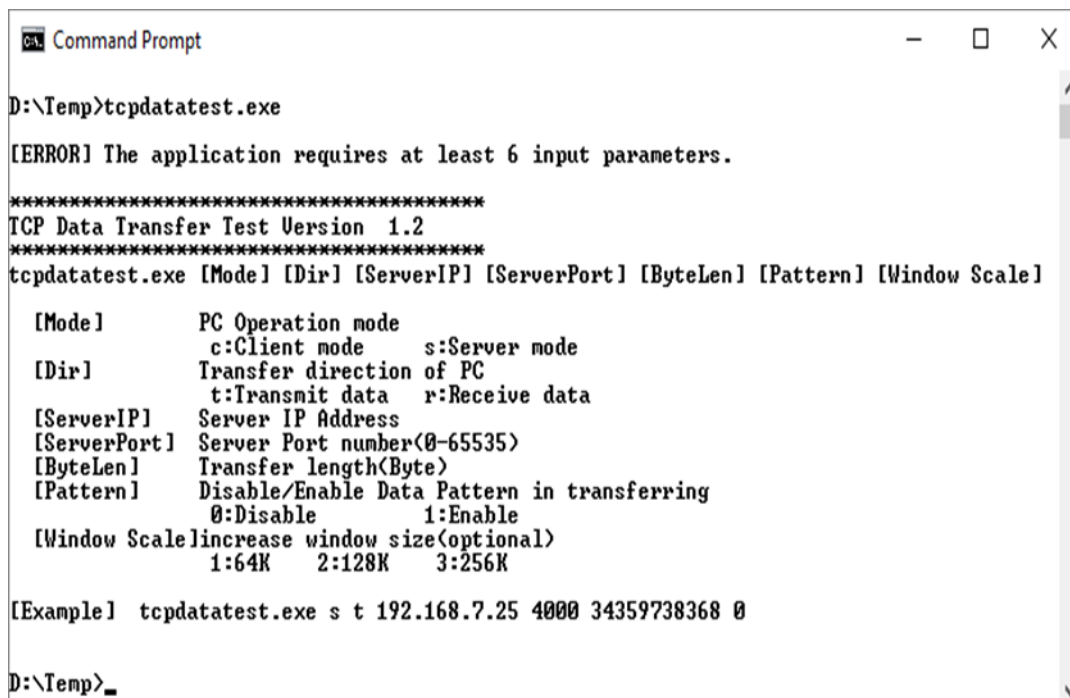
int toe_send_test(void)	
Parameters	None
Return value	0: The operation is successful -1: Receive invalid input or error is found
Description	Run Send data test following description in topic 3.3

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 described in topic 3.5

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

## 4 Test Software on PC

### 4.1 “tcpdatatest” for half duplex test



```

Command Prompt
D:\Temp>tcpdatatest.exe

[ERROR] The application requires at least 6 input parameters.

*****
TCP Data Transfer Test Version 1.2
*****
tcpdatatest.exe [Mode] [Dir] [ServerIP] [ServerPort] [ByteLen] [Pattern] [Window Scale]

[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
[Window Scale] increase window size(optional)
             1:64K    2:128K    3:256K

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

D:\Temp>_

```

Figure 4-1 “tcpdatatest” application usage

“tcpdatatest” is designed to run on PC for sending or receiving TCP data via Ethernet as server or client mode. PC of this demo should run in client mode. User sets parameters 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, so it is ignored in receive mode. In receive mode, the application is closed when the connection is destroyed. In transmit mode, ByteLen must be equal to the total transfer size on FPGA, set in receive 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.

Note: Window Scale: Optional parameter which is not used in the demo.

### Transmit data mode

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

- 1) Get parameters from the user and verify that all inputs are 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 send buffer.
- 5) Skip this step if the dummy pattern is selected. Otherwise, generate the incremental test pattern to send buffer when the test pattern is enabled.
- 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 sequence is the sequence when test application runs in receive mode.

- 1) Follow the step 1) – 3) of Transmit data mode.
- 2) Allocate memory to be receive buffer.
- 3) Read data from the receive buffer and increase total receive size.
- 4) This step is skipped if data verification is disabled. Otherwise, received data is verified by the incremental pattern. Error message is printed out when data is not correct.
- 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\_trx\_40G” for full duplex test



```

Command Prompt
D:\Temp>tcp_client_trx_40G.exe
*****
TCP Tx Rx Version 1.1
*****
tcp_client_trx_40G.exe [ServerIP] [ServerPort] [ByteLen] [Verification]

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

[Example] tcp_client_trx_40G.exe 192.168.40.42 60000 137438953440 0

D:\Temp>_

```

Figure 4-2 “tcp\_client\_trx\_40G” application usage

“tcp\_client\_trx\_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 TOE100G-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. This value must be equal to the transfer size set on FPGA for running full-duplex test.
- 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 send and receive 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 send data from the function, and calculate remaining send size.
- (7) Read data from the receive buffer and increase total receive data size.
- (8) Skip this step if data verification is disabled. Otherwise, data is verified by incremental pattern. Error message is printed out when data is not correct.
- (9) Print total send size and receive size every second.
- (10) Repeat step 5) – 9) until total send size and receive 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) Repeat step 3) – 13) in forever loop. If verification is failed, the application is stopped.

## 5 Revision History

Revision	Date	Description
1.0	25-Jan-21	Initial version release