TOE25G-IP with CPU reference design

Rev1.4 4-Jan-23

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.

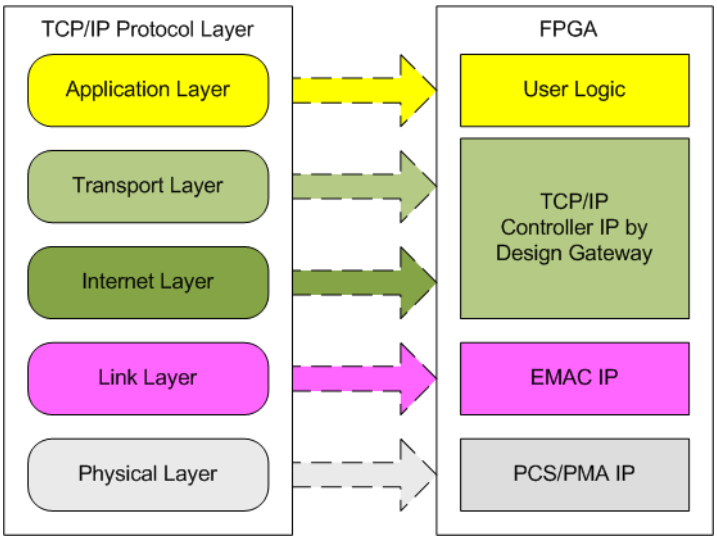

Figure 1-1 TCP/IP protocol layer

TOE25G-IP implements the Transport and Internet layer of TCP/IP Protocol for building Ethernet packet from the user data which is TCP payload data to EMAC. If TCP payload data size is larger than a packet size, TOE25G-IP splits the data to send by using multiple packets. Next, the TCP payload data is appended by TCP/IP header. On the other hand, the received Ethernet packet from EMAC is extracted by TOE25G-IP. The header of the packet is verified. If the header is valid, TCP payload data is forwarded to the user logic. Otherwise, the packet is rejected.

The lower layer protocols are implemented by EMAC-IP and PCS/PMA-IP. These low-level IPs can be implemented by several solutions. First is DG 10G25GEMAC-IP with Xilinx PCS/PMA-IP. Another is Xilinx Ethernet (MAC) Subsystem which integrates both EMAC-IP and PCS/PMA-IP.

The reference design provides the evaluation system which includes simple user logic to transfer data by using TOE25G-IP. TOE25G-IP supports to transfer data with PC or another TOE25G-IP run on another FPGA board. To run with PC, the test application is called on PC to send and verify TCP payload 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 by one session).

To allow the user setting the test parameters and controlling the operation of TOE25G-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

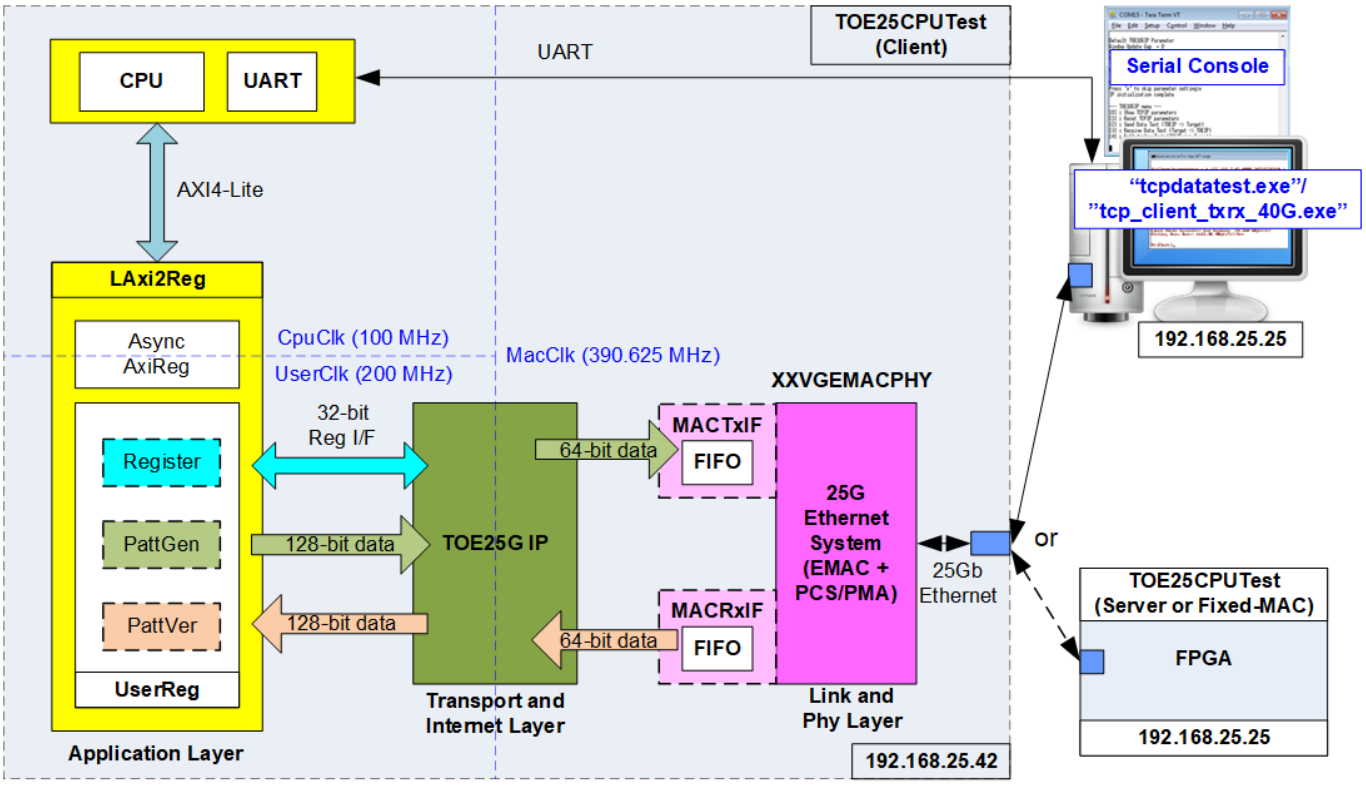

Figure 2-1 Demo block diagram

In test environment, two devices are used for 25Gb Ethernet transferring. When running by FPGA and PC, FPGA is initialized by Client mode and PC is initialized by Server mode. On the other hand, when running by two FPGAs, it may be initialized by Client <–> Server, Client <–> Fixed-MAC, or Fixed-MAC <-> Fixed-MAC, as shown in Figure 2-1. When using PC, the test applications (tcpdatatest and tcp_client_txrx_40G) must be run on PC for transferring data.

In FPGA system, TOE25G-IP connects to 25G Ethernet System to implement all TCP/IP layers. When 25G Ethernet System is implemented by DG 10G25GEMAC-IP and Xilinx PCS/PMA-IP, the 25G Ethernet System can connect to TOE25G-IP directly. While using Xilinx 25G Ethernet (MAC) Subsystem IP, the adapter logic (MACTxIF and MACRxIF) must be included to be interface module with TOE25G-IP.

User interface of TOE25G-IP connects to UserReg within LAxi2Reg. UserReg 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 user interface clock of 25Gb Ethernet System, and UserClk which is the clock for running user logic of TOE25G-IP. In real system, the user can change the frequency of CpuClk and UserClk. According to TOE25G-IP datasheet, clock frequency of UserClk should be more than or equal to 195.3125 MHz for 25Gb Ethernet to achieve the best performance. AsyncAxiReg is designed to support asynchronous signals between CpuClk and UserClk. More details of each module inside the TOE25CPUTest are described as follows.

2.1 25G Ethernet System (25G BASE-SR)

25G Ethernet System consists of the MAC layer and PCS/PMA layer. The user interface for connecting with Ethernet MAC is 64-bit AXI4-stream interface running at 390.625 MHz. The physical interface is 25G BASE-SR. There are many solutions for implementing 25G Ethernet System. Three solutions are explained in this document, as shown in Figure 2-2.

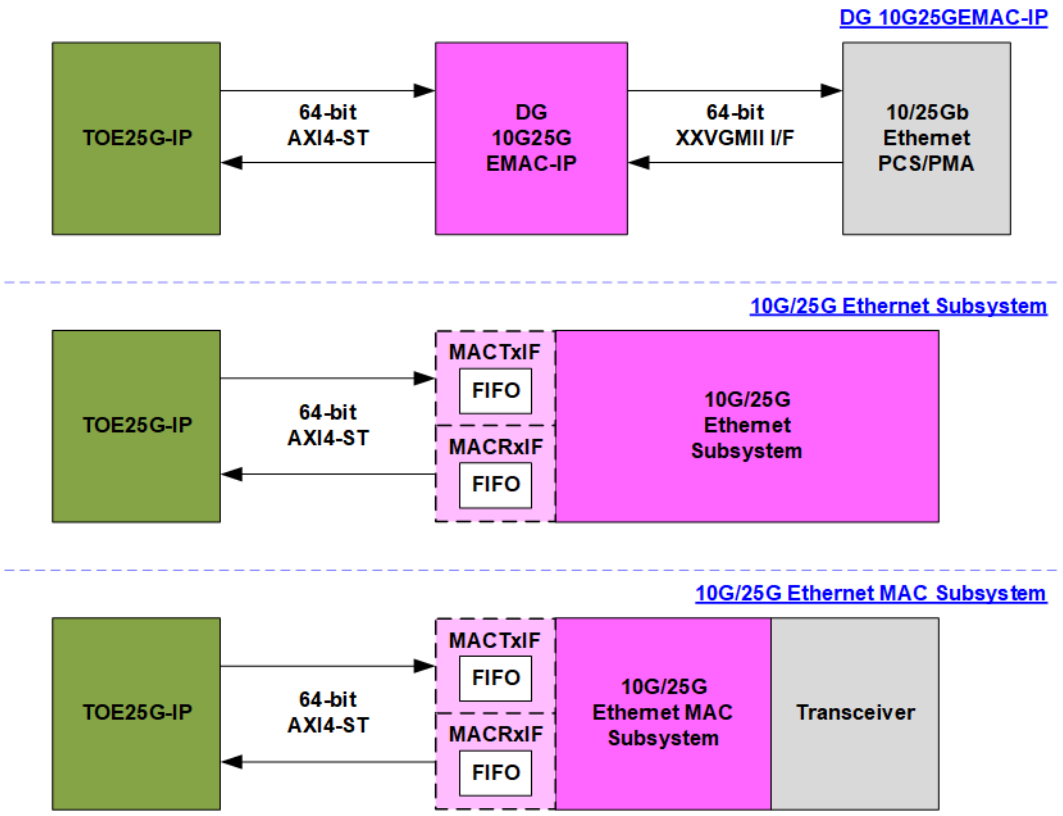

Figure 2-2 TOE25G-IP with several Ethernet system solutions

First uses DG 10G25GEMAC-IP with Xilinx PCS/PMA module. Using this solution optimizes the IP resource and the operation has less latency time. Also, the user interface of DG 10G25GEMAC-IP can connect with TOE25G-IP directly. While PCS/PMA module without supporting RS-FEC feature is the no charge IP core that can be created by using IP wizard of Xilinx tool. More details of this solution are described in the following website.
https://dgway.com/products/IP/GEMAC-IP/dg_10g25gemacip_data_sheet_xilinx.pdf

Second uses 10G/25G Ethernet Subsystem which integrates both Ethernet MAC and PCS/PMA function. While MAC interface of TOE25G-IP does not support to pause data transmission before the end of packet is transmitted, user interface of 10G/25G Ethernet Subsystem may de-assert valid/ready signal to pause data transmission. Therefore, the adapter logics (MACTxIF and MACRxIF) with small FIFO must be included to interface between TOE25G-IP and 10G/25G Ethernet Subsystem. More details of this solution are described in the following website.
https://www.xilinx.com/products/intellectual-property/ef-di-25gemac.html

Final solution uses Ethernet MAC Subsystem which is the IP on Versal device. This IP does not include Transceiver module (PMA module), so Transceiver must be generated to interface with external I/O pin. The user interface of the IP can be configured to several modes. In this reference design, 64-bit non-segmented mode with independent clock is applied. The minimum clock frequency in this mode is 390.625 MHz. Similar to 10G/25G Ethernet Subsystem, it needs to integrate the adapter logics to interface between TOE25G-IP and Ethernet MAC Subsystem. More details of the final solution are described in the following website.
https://www.xilinx.com/products/intellectual-property/mrmac.html

Note: In this demo, 25G Ethernet System enables RS-FEC feature, so please confirm the network equipment of the test environment for running this demo that it can support RS-FEC.

The adapter logics (MAC25GTxIF and MAC25GRxIF) for interface between TOE25G-IP and Ethernet system are described in more details as follows.

MAC25GTxIF

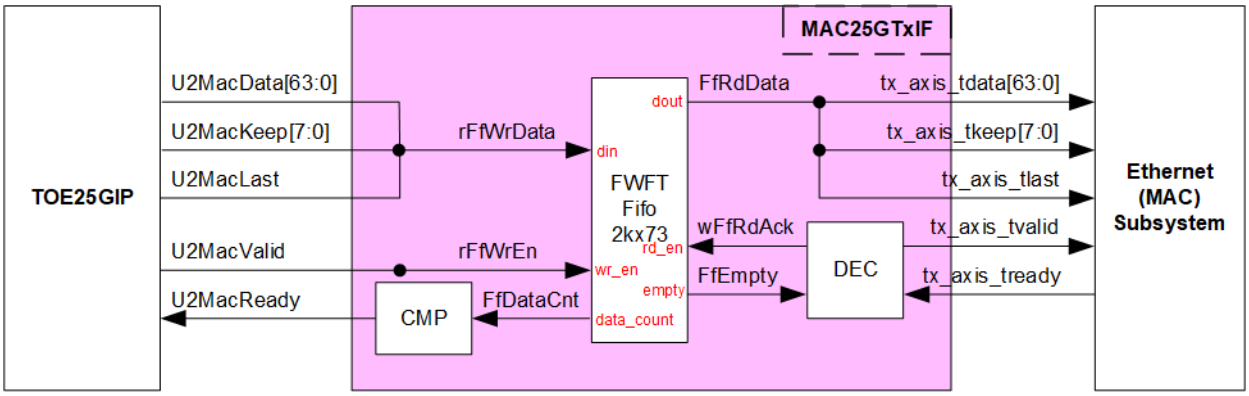

Figure 2-3 MAC25GTxIF Timing diagram

Tx interface timing diagram of Ethernet (MAC) Subsystem and TOE25G-IP are different. TOE25G-IP needs to send data of one packet continuously while Xilinx Ethernet (MAC) Subsystem does not support this feature. EMAC may de-assert ready signal to pause receiving data before end of the packet.

MAC25GTxIF is designed to store transmitted data from TOE25G-IP when EMAC is not ready to receive the new data. The FIFO depth is 2048 to store at least one data packet during pausing time. Maximum packet size of TOE25G-IP reference design is 8960 bytes or 1120 of 64-bit data. Therefore, 2048 is enough for storing one packet. The FIFO is First-Word Fall-Through (FWFT) FIFO, so the read data is valid for reading at the same time as asserting read enable to '1'.

The operation of MAC25GTxIF is split into two parts. First is the logic for transferring a packet from TOE25GIP to FIFO. Second is the logic for transferring a packet from FIFO to Ethernet (MAC) Subsystem. Timing diagrams of each part are displayed in Figure 2-4 and Figure 2-5.

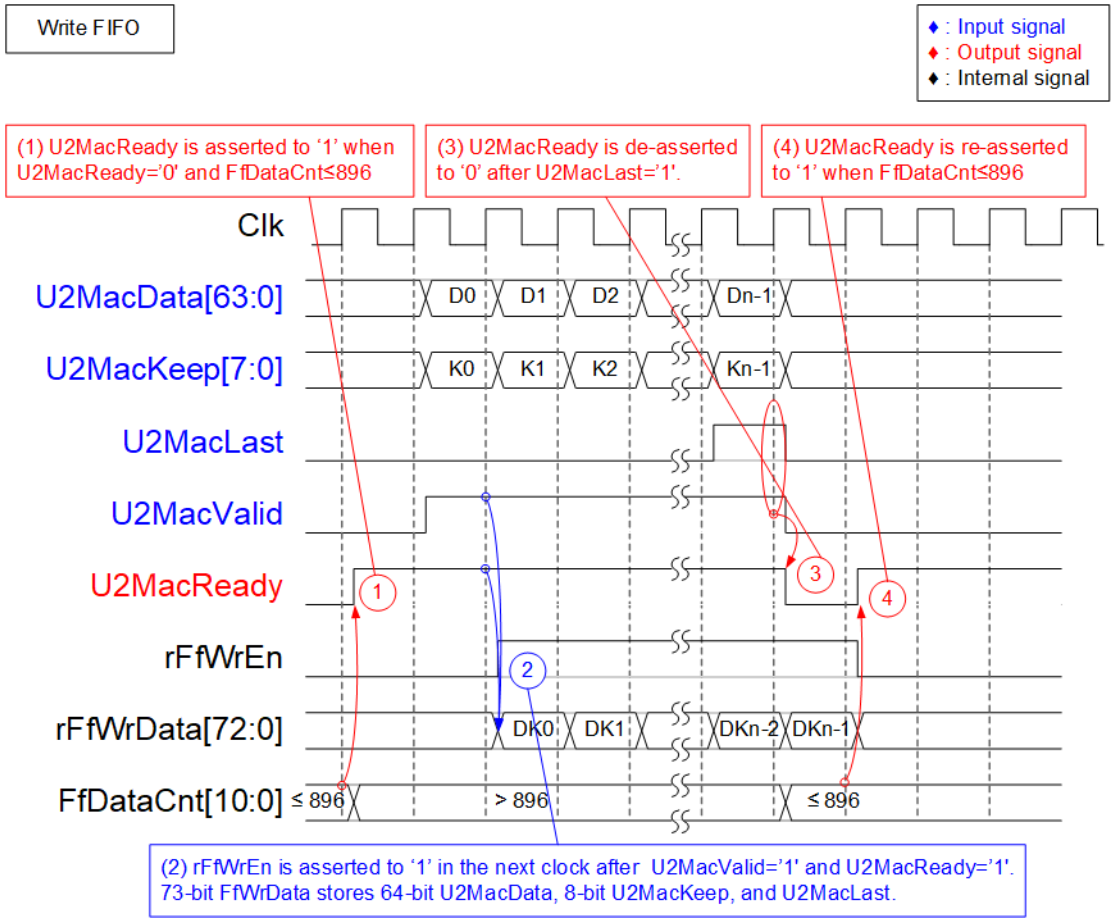

Figure 2-4 Timing diagram for transferring data from TOE25G-IP to FIFO

1) Before asserting U2MacReady to '1' for receiving the new packet from user, two conditions must be met. First, free space in FIFO is enough for storing maximum packet size, 9014 bytes. For simple monitoring logic, the upper bit of FfDataCnt is read to confirm the amount of data in FIFO is not more than 896 (free space is more than 1151 of 64-bit). Second, the previous packet is completely transferred, monitored by U2MacReady='0'.
2) User starts transmitting a packet by asserting U2MacValid to '1'. The input signals from user (U2MacData, U2MacKeep, and U2MacLast) are valid and stored to FIFO when U2MacValid and U2MacReady are asserted to '1'. After that, the inputs are stored to FIFO by asserting rFfWrEn to '1'. 73-bit Write data to FIFO consists of 64-bit data (U2MacData), 8-bit empty byte (U2MacKeep), and end flag (U2MacLast).
3) After receiving the final data of a packet (U2MacLast='1' and U2MacValid='1'), U2MacReady is de-asserted to '0' to pause data transmission for reading FfDataCnt.
4) If FfDataCnt shows free space of FIFO is enough, U2MacReady will be re-asserted to '1' in the next cycle.

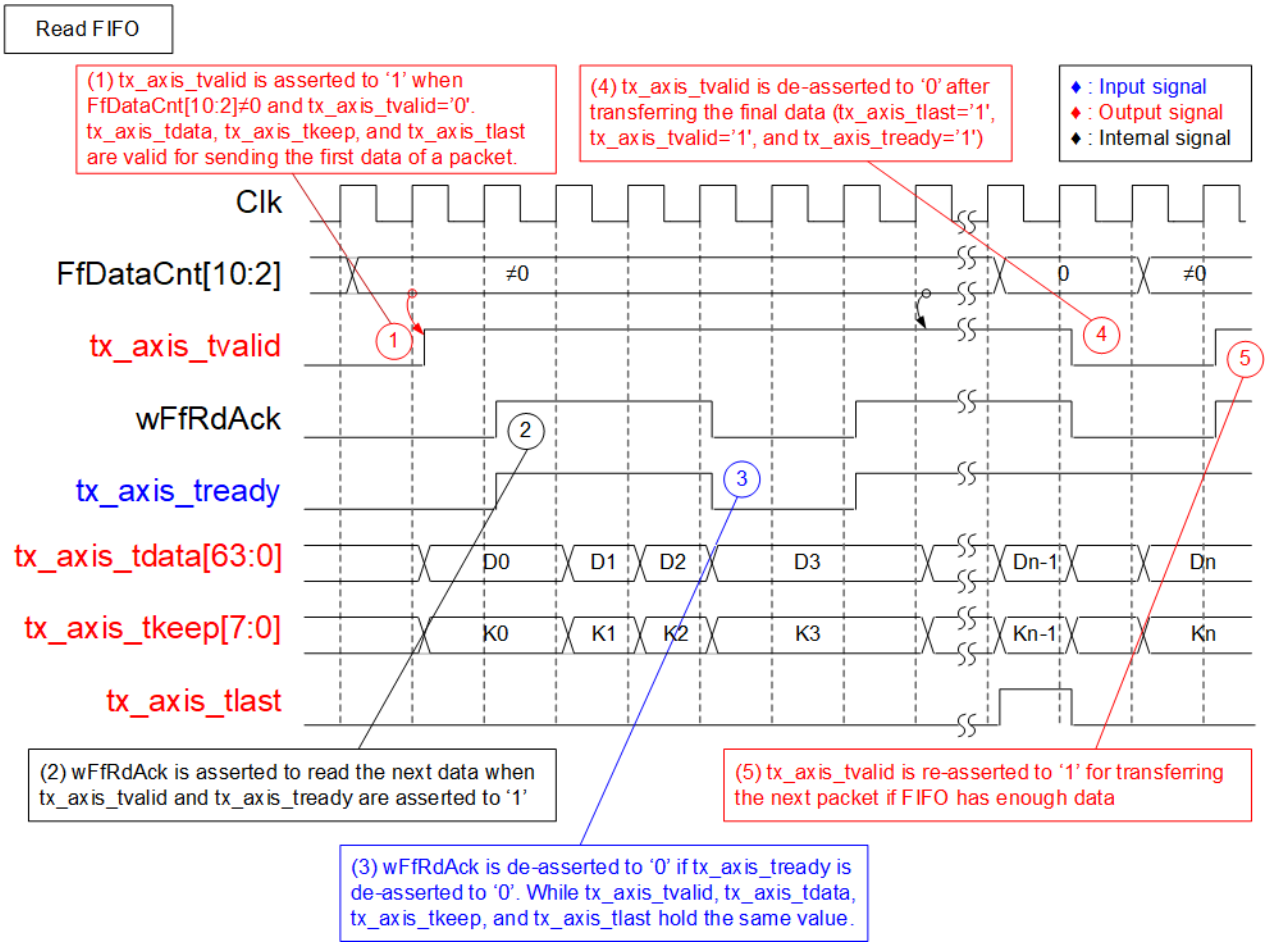

Figure 2-5 Timing diagram for transferring data from FIFO to EMAC

1) The new packet starts transmission when the FIFO stores some data (FfDataCnt[10:2] ≠ 0) and the packet is not transmitting (tx_axis_tvalid='0'). To start data transmission, tx_axis_tvalid is asserted to '1' with the valid output signals to EMAC, i.e., 64-bit tx_axis_tdata, 8-bit tx_axis_tkeep, and tx_axis_tlast.
2) If the data is transmitted to EMAC completely (tx_axis_tvalid ='1' and tx_axis_tready='1'), wFfRdAck is asserted to '1' to read the next data from FIFO.
3) If tx_axis_tready is de-asserted to '0', wFfRdAck will be de-asserted to '0' to pause reading the new data from FIFO. Therefore, all output signals sent to EMAC hold the same value until EMAC re-asserts tx_axis_tready to 1'.
4) After the final data of a packet is transferred completely (tx_axis_tlast='1' and tx_axis_tready='1'), tx_axis_tvalid is de-asserted to '0' to pause data transmission and check data size in FIFO for transferring the next packet.
5) The next packet is transmitted when FIFO has enough data. Then, it returns to step 1 to transmit the new packet.

MAC25GRxIF

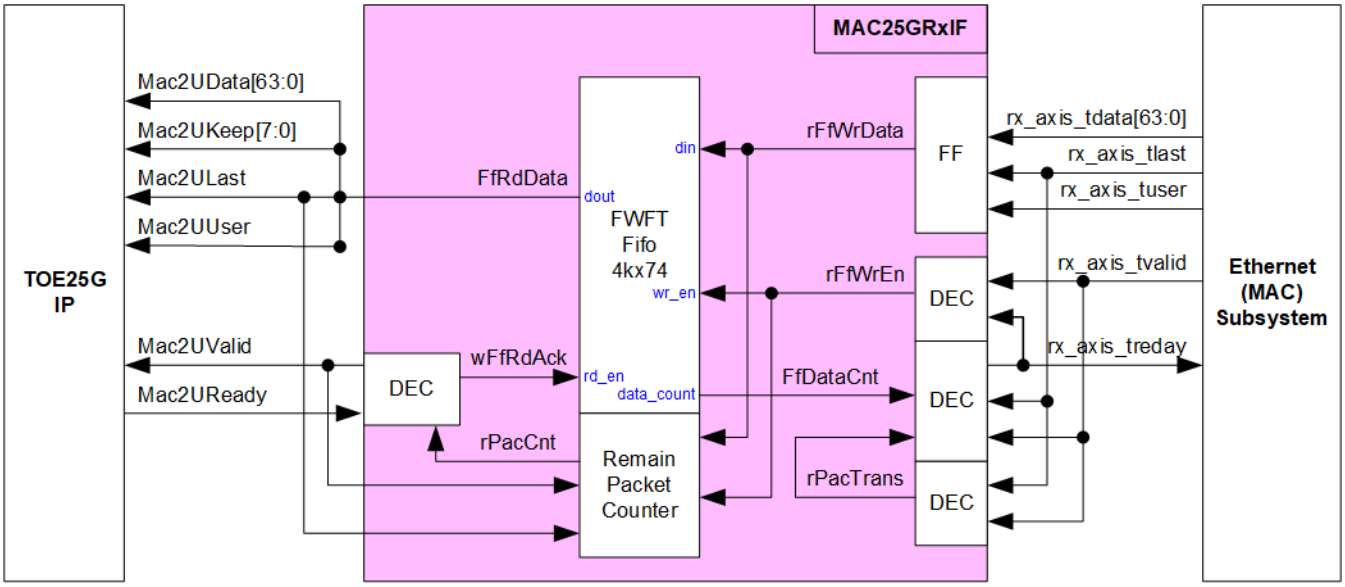

Figure 2-6 MAC25GRxIF Timing diagram

Rx interface timing diagram of Ethernet (MAC) Subsystem and TOE25G-IP are different. TOE25G-IP needs to receive data of one packet continuously, but Xilinx EMAC does not support this feature. EMAC may de-assert valid signal to pause transmitting data before the end of the packet.

MAC25GRxIF is designed to store one packet data transmitted from EMAC and then forward to TOE25G-IP without pausing data transmission. The FIFO depth is 4096 which is enough for storing several Ethernet packets. The FIFO is First-Word Fall-Through FIFO, similar to MAC25GTxIF.

Remain packet counter counts the number of packets stored in the FIFO. The counter is increased when the new packet is received from EMAC while the counter is decreased when the packet is transferred to TOE25G-IP completely.

The operation of MAC25GRxIF is split into two parts. First is the logic for transferring a packet from EMAC to FIFO. Second is the logic for transferring a packet from FIFO to TOE25G-IP. Timing diagrams of each part are displayed on Figure 2-7 and Figure 2-8.

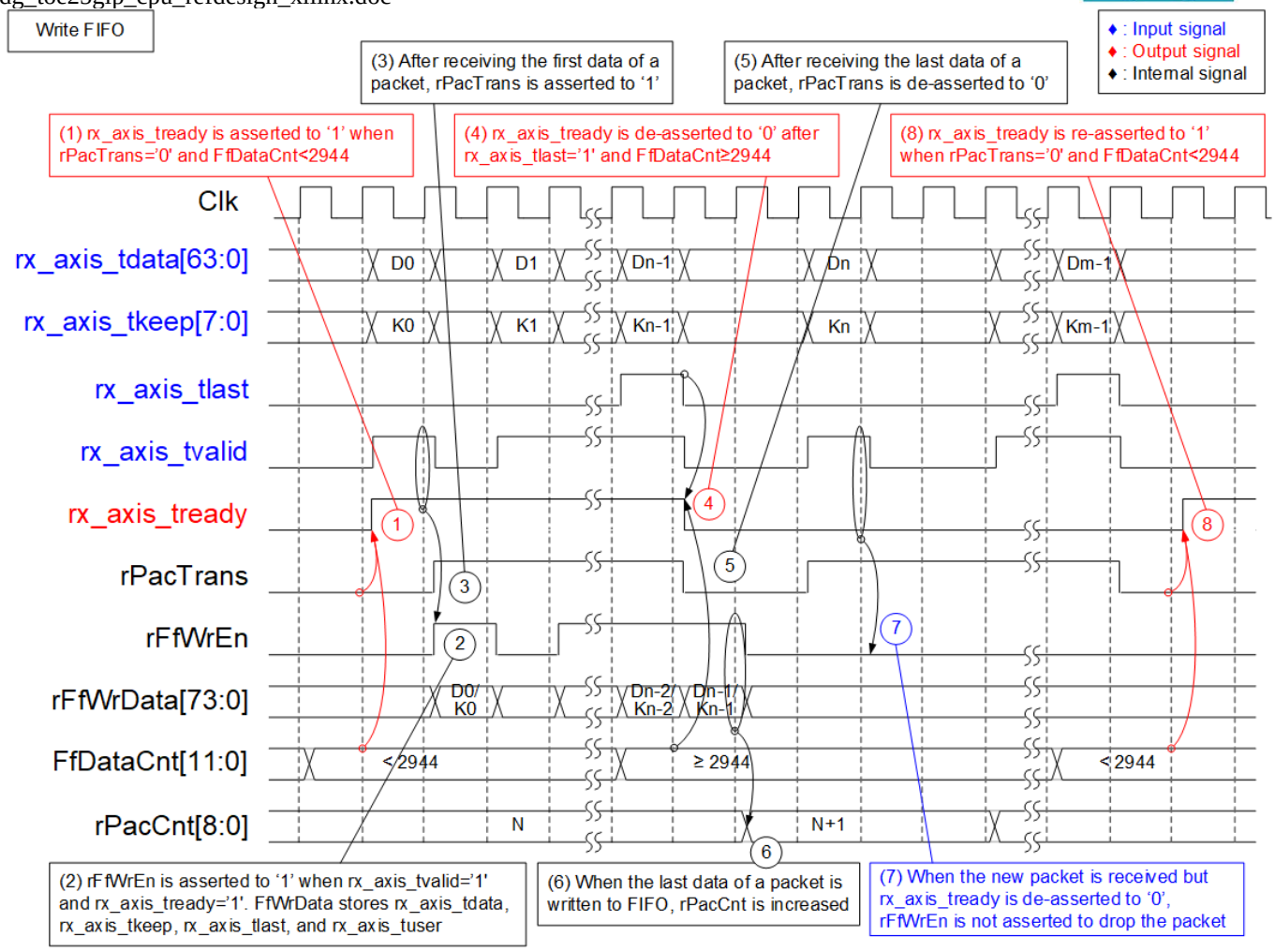

Figure 2-7 Timing diagram for transferring data from EMAC to FIFO

1) FfDataCnt is read to check the free space size. If FfDataCnt is less than 2944 (the free space in FIFO is more than 1153), it is enough for storing the maximum packet size (9014 bytes). Also, rPacTrans must be equal to 0 to confirm the packet is not transmitting. After that, rx_axis_tready is asserted to '1' to start data reception from EMAC.
2) When rx_axis_tvalid is asserted to '1' to transfer the new packet, the data and control signals from EMAC are stored to FIFO, i.e., 64-bit rx_axis_tdata, 8-bit rx_axis_tkeep, rx_axis_tlast, and rx_axis_tuser. rFfWrEn is asserted to '1' to write 74-bit data to FIFO.
3) After the first data of a packet is received, rPacTrans is asserted to '1' until receiving the end of packet. Therefore, rPacTrans can be applied to monitor the packet transmission status.
4) If the final data of a packet is received and free space size in FIFO is not enough (FFDataCnt≥2944), rx_axis_tready will be de-asserted to '0' to pause data reception.
5) After the final data of a packet is received, rPacTrans is de-asserted to '0' to change packet transmission status from Busy to Idle.
6) After storing the final data of a packet to FIFO (rFfWrEn='1' and rFfWrData[72] which is last flag ='1'), rPacCnt which is the counter to show total number of packet stored in FIFO is up-counted.
7) If the next packet is received but rx_axis_tready is still de-asserted to '0', the received packet will be dropped and not be stored to FIFO.
8) rx_axis_tready is re-asserted to '1' when there is no packet transmitted and free space size in FIFO is enough.

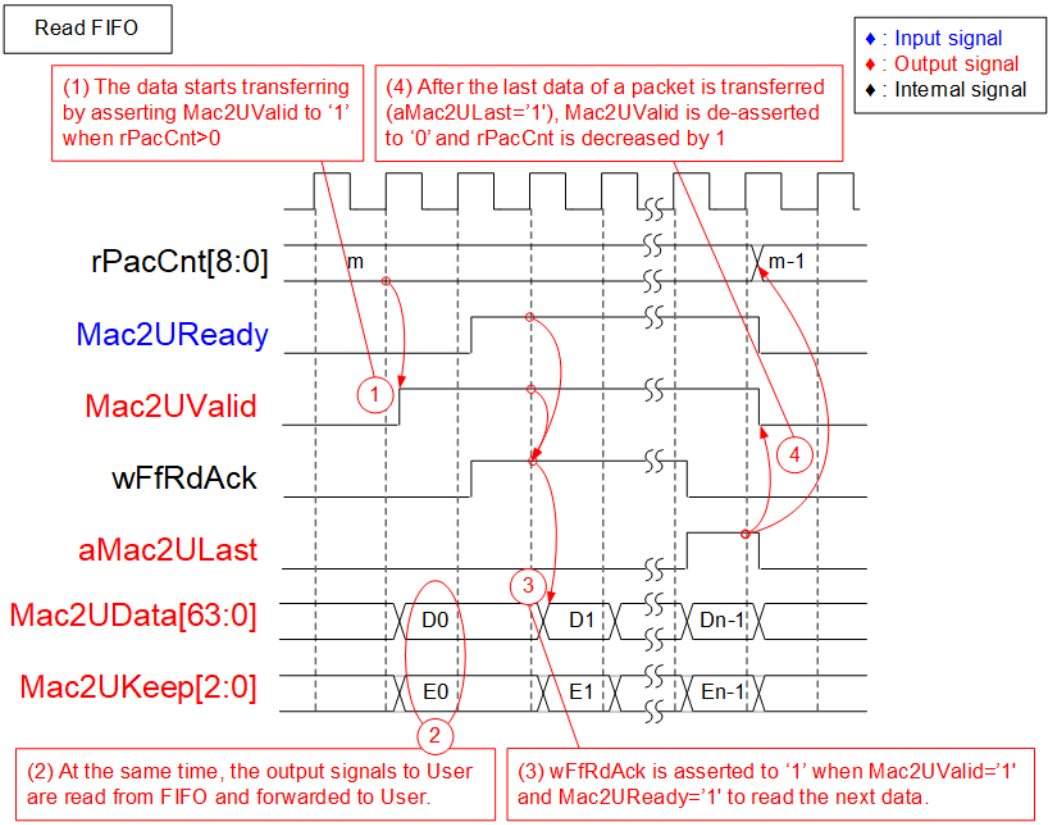

Figure 2-8 Timing diagram for transferring data from FIFO to TOE25G-IP

1) The packet forwarding from FIFO to TOE25G-IP begins when at least one packet is stored in FIFO (rPacCnt which shows the number of packets stored in FIFO is not equal to 0). Mac2UValid is asserted to '1' to start data transmission.
2) The read data output from FIFO are applied to be the data and control signals sent to TOE25G-IP, i.e., 64-bit Mac2UData, 8-bit Mac2UKeep, Mac2ULast, and Mac2UUser. Mac2UValid is asserted to '1' until the end of packet. Therefore, the data and control signals of each packet are transferred to TOE25G continuously.
3) After each data is transferred to TOE25G-IP completely (Mac2UValid='1' and Mac2UReady='1'), wFfRdAck is asserted to '1' to read the next data.
4) After the final data of a packet is transferred (aMac2ULast='1' and Mac2UValid='1'), Mac2UValid and wFfRdAck are de-asserted to '0' to pause a packet transmission. Also, rPacCnt is down-counted after completing transferring one packet.

2.2 TOE25G-IP

TOE25G-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 25G EMAC is 64-bit AXI4 interface. More details are described in datasheet.
http://www.dgway.com/products/IP/TOE25G-IP/dg_toe25gip_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. 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 2-9.

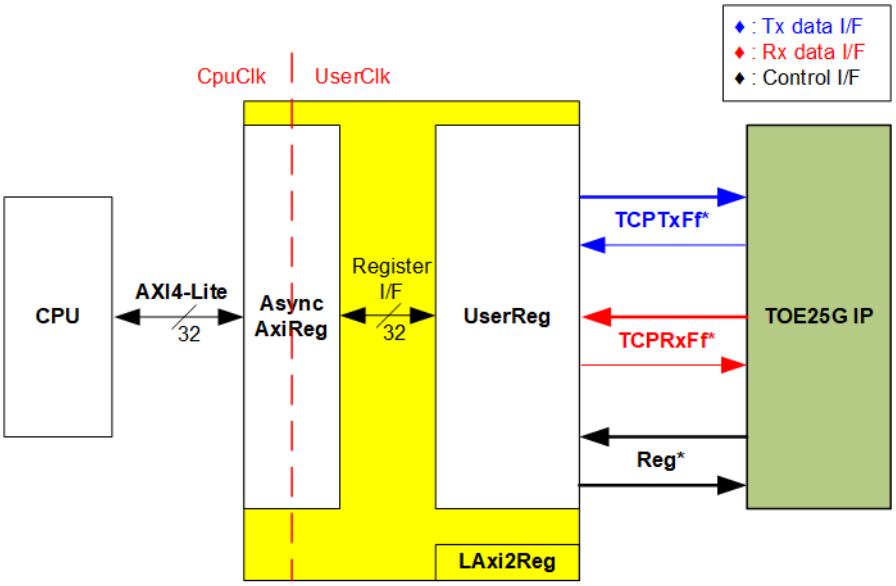

Figure 2-9 LAxi2Reg block diagram

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). Also, AsyncAxiReg includes asynchronous logic to support clock domain crossing between CpuClk and UserClk.

UserReg includes the register file of the parameters and the status signals of test logics. Both data interface and control interface of TOE25G-IP are also connected to UserReg. More details of AsyncAxiReg and UserReg are described as follows.

2.3.1 AsyncAxiReg

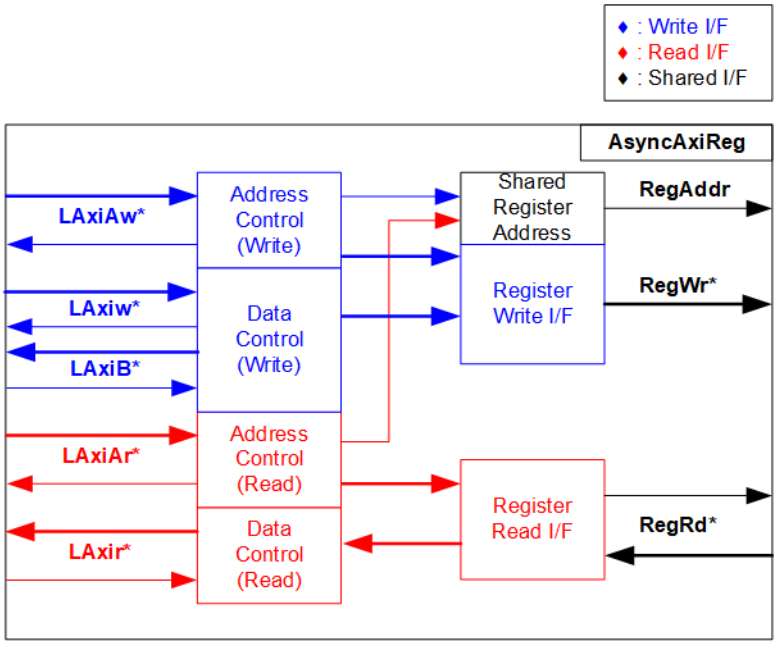

Figure 2-10 AsyncAxiReg interface

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://github.com/Architech-Silica/Designing-a-Custom-AXI-Slave-Peripheral/blob/master/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 2-10. Write control I/F and Write data I/F of AXI4-Lite bus are latched and transferred to be Write register interface with clock domain crossing registers. Similarly, Read control I/F of AXI4-Lite bus are latched and transferred to be Read register interface. While the returned data from Register Read I/F is transferred to AXI4-Lite bus by using clock domain crossing registers. In Register interface, RegAddr is shared signal for write and read access, so it loads the value 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 2-11.

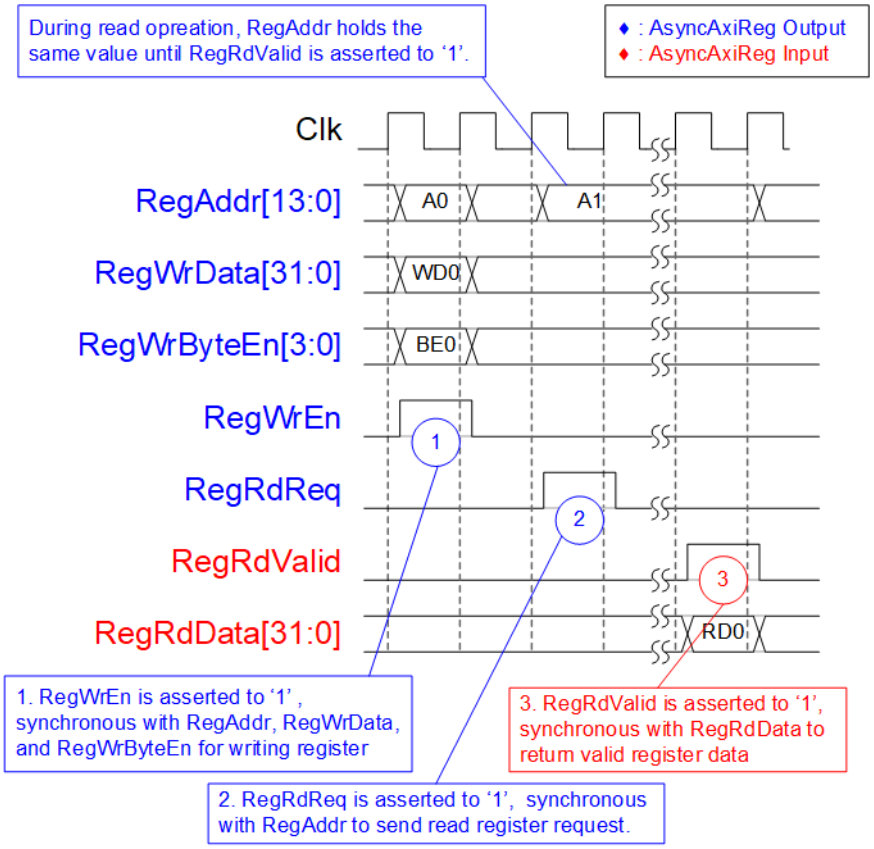

Figure 2-11 Register interface timing diagram

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

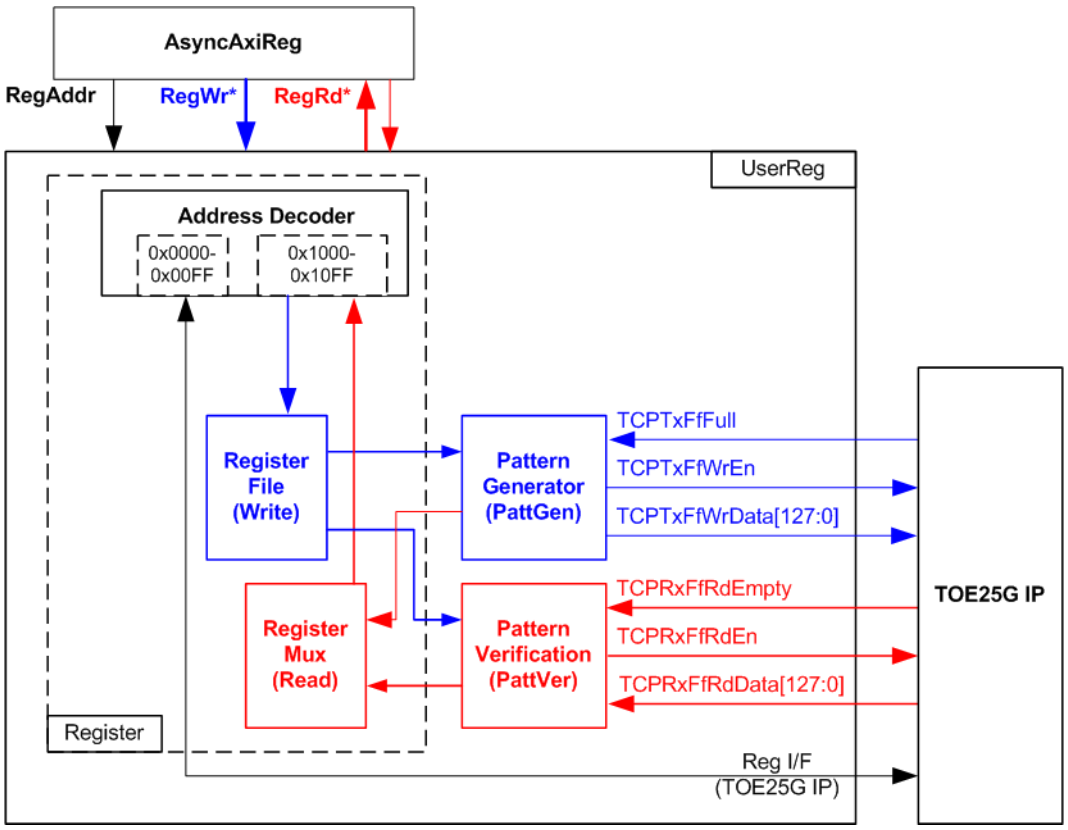

Figure 2-12 UserReg block diagram

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 128-bit test data to TOE25G-IP following FIFO interface standard. Pattern verification block is designed to read and verify 128-bit data from TOE25G-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., TOE25G-IP register (0x0000-0x00FF) and UserReg register (0x1000-0x10FF). Therefore, the upper bit of RegAddr is applied to select the active area between TOE25G-IP or UserReg register. While the lower bits of RegAddr are fed to TOE25G-IP and internal registers of UserReg to select the active register. The registers inside UserReg are 32-bit data, 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, multiplexer selects 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 Wr/Rd	Register Name (Label in the “toe25gtest.c”)	Description
BA+0x0000 – BA+0x00FF: TOE25G-IP Register Area More details of each register are described in TOE25G-IP datasheet.		
BA+0x0000	TOE_RST_INTREG	Mapped to RST register within TOE25G-IP
BA+0x0004	TOE_CMD_INTREG	Mapped to CMD register within TOE25G-IP
BA+0x0008	TOE_SML_INTREG	Mapped to SML register within TOE25G-IP
BA+0x000C	TOE_SMH_INTREG	Mapped to SMH register within TOE25G-IP
BA+0x0010	TOE_DIP_INTREG	Mapped to DIP register within TOE25G-IP
BA+0x0014	TOE_SIP_INTREG	Mapped to SIP register within TOE25G-IP
BA+0x0018	TOE_DPN_INTREG	Mapped to DPN register within TOE25G-IP
BA+0x001C	TOE_SPN_INTREG	Mapped to SPN register within TOE25G-IP
BA+0x0020	TOE_TDL_INTREG	Mapped to TDL register within TOE25G-IP
BA+0x0024	TOE_TMO_INTREG	Mapped to TMO register within TOE25G-IP
BA+0x0028	TOE_PKL_INTREG	Mapped to PKL register within TOE25G-IP
BA+0x002C	TOE_PSH_INTREG	Mapped to PSH register within TOE25G-IP
BA+0x0030	TOE_WIN_INTREG	Mapped to WIN register within TOE25G-IP
BA+0x0034	TOE_ETL_INTREG	Mapped to ETL register within TOE25G-IP
BA+0x0038	TOE_SRV_INTREG	Mapped to SRV register within TOE25G-IP
BA+0x003C	TOE_VER_INTREG	Mapped to VER register within TOE25G-IP
BA+0x0040	TOE_DML_INTREG	Mapped to DML register within TOE25G-IP
BA+0x0044	TOE_DMH_INTREG	Mapped to DMH register within TOE25G-IP
BA+0x1000 – BA+0x10FF: UserReg control/status		
BA+0x1000 Wr/Rd	Total transmit length (USER_TXLEN_INTREG)	Wr [31:0] – Total amount of transmitted data in 128-bit unit. Valid from 1-0xFFFFFFFF. Rd [31:0] – Current amount of transmitted data in 128-bit unit. The value is cleared to 0 when USER_CMD_INTREG is written by user.
BA+0x1004 Wr/Rd	User Command (USER_CMD_INTREG)	Wr [0] – Start transmitting. Set ‘0’ to start transmitting data. [1] – Data verification enable (‘0’: Disable data verification, ‘1’: Enable data verification) Rd [0] – Busy of PattGen inside UserReg (‘0’: Idle, ‘1’: PattGen 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 TOE25G-IP
BA+0x1008 Wr/Rd	User Reset (USER_RST_INTREG)	Wr [0] – Reset signal. Set ‘1’ to reset UserReg. 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_INTREG[8]=’1’. [16] – Ethernet linkup status from Ethernet MAC (‘0’: Not linkup, ‘1’: Linkup)
BA+0x100C Rd	FIFO status (USER_FFSTS_INTREG)	Rd[3:0] - Mapped to TCPRxFfLastRdCnt signal of TOE25G-IP [15:4] - Mapped to TCPRxFfRdCnt signal of TOE25G-IP [24] - Mapped to TCPTxFfFull signal of TOE25G-IP
BA+0x1010 Rd	Total receive length (USER_RXLEN_INTREG)	Rd[31:0] – Current amount of received data from TOE25G-IP in 128-bit unit. The value is cleared to 0 when USER_CMD_INTREG is written by user.
BA+0x1080 Rd	EMAC IP version (EMAC_VER_INTREG)	Rd[31:0] – Mapped to IPVersion output from DG 10G25GEMAC-IP when the system integrates DG 10G25GEMAC-IP.

Address Wr/Rd	Register Name (Label in the “toe25gtest.c”)	Description
BA+0x1000 – BA+0x10FF: UserReg control/status		
BA+0x1020 Wr/Rd	Connection interrupt (USER_INT_INTREG)	Wr[0] – Set ‘1’ to clear the connection interrupt (USER_INT_INTREG[0]) Rd[0] – Interrupt from ConnOn edge detection (‘1’: Detect edge of ConnOn signal from TOE25G-IP, ‘0’: ConnOn does not change the value.) *Note: ConnOn value can be read from USER_CMD_INTREG[2].*
BA+0x1080 Rd	EMAC IP version (EMAC_VER_INTREG)	Rd[31:0] – Mapped to IPVersion output from DG 10G25GEMAC-IP when the system integrates DG 10G25GEMAC-IP.

Pattern Generator

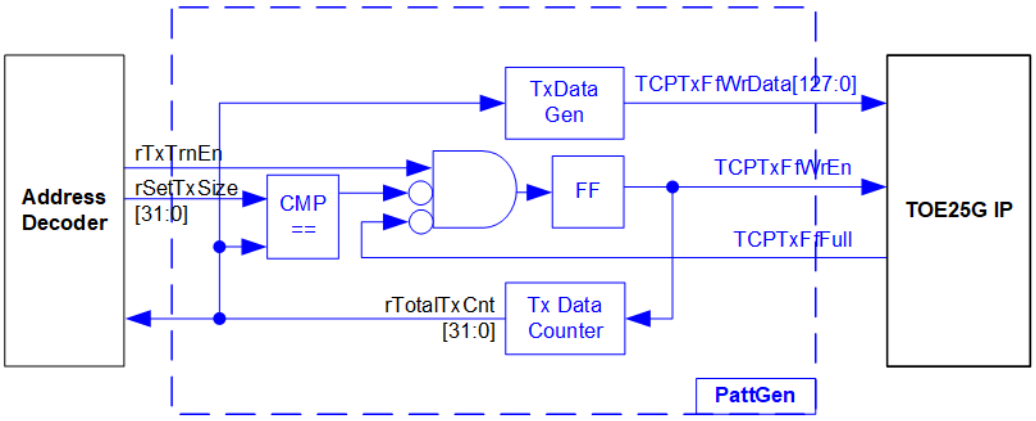

Figure 2-13 PattGen block

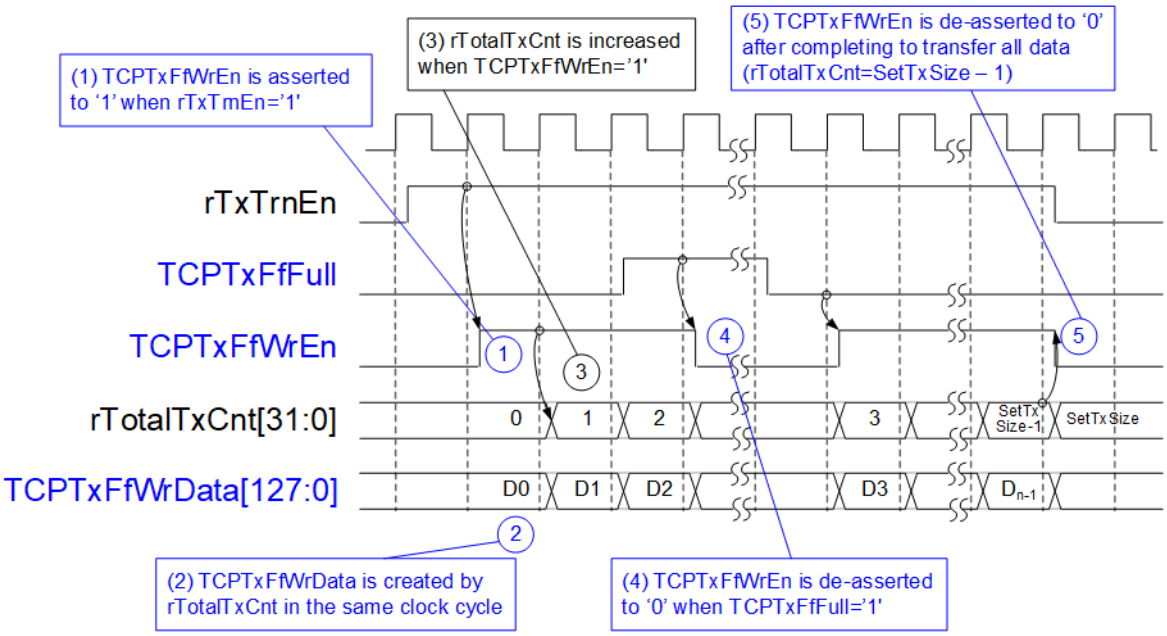

Figure 2-14 PattGen timing diagram

Figure 2-13 shows the details of PattGen which generates test data to TOE25G-IP. Timing diagram to show the relation of each logic is displayed in Figure 2-14.

To start PattGen operation, the user sets USER_CMD_INTREG[0]='0' and then rTxTrnEn is asserted to '1'. When rTxTrnEn is '1', TCPTxFfWrEn is controlled by TCPTxFfFull. TCPTxFfWrEn is de-asserted to '0' when TCPTxFfFull is '1'. rTotalTxCnt is the data counter to check total amount of transmitted data to TOE25G-IP. Also, rTotalTxCnt is used to generate 32-bit incremental data for TCPTxFfWrData signal. After all data is transferred completely (Total amount of data is equal to rSetTxSize-1), rTxTrnEn is de-asserted to '0'.

Pattern Verification

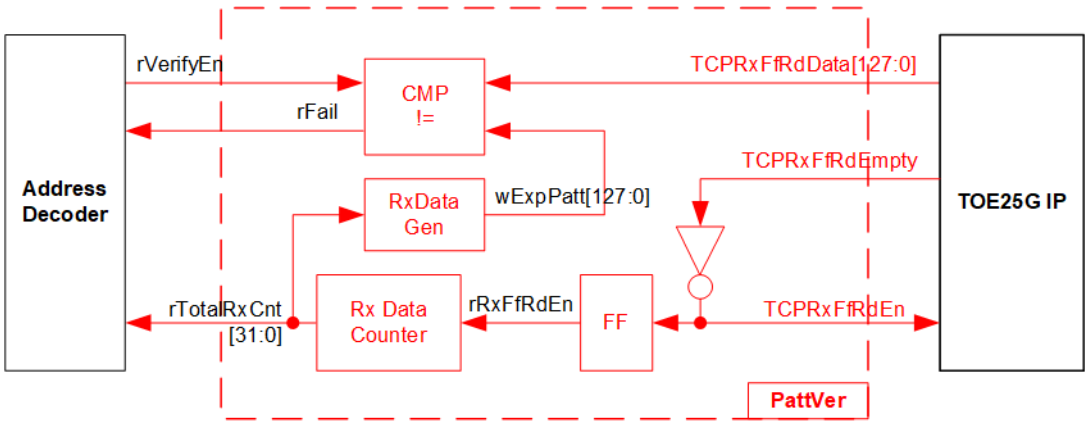

Figure 2-15 PattVer block

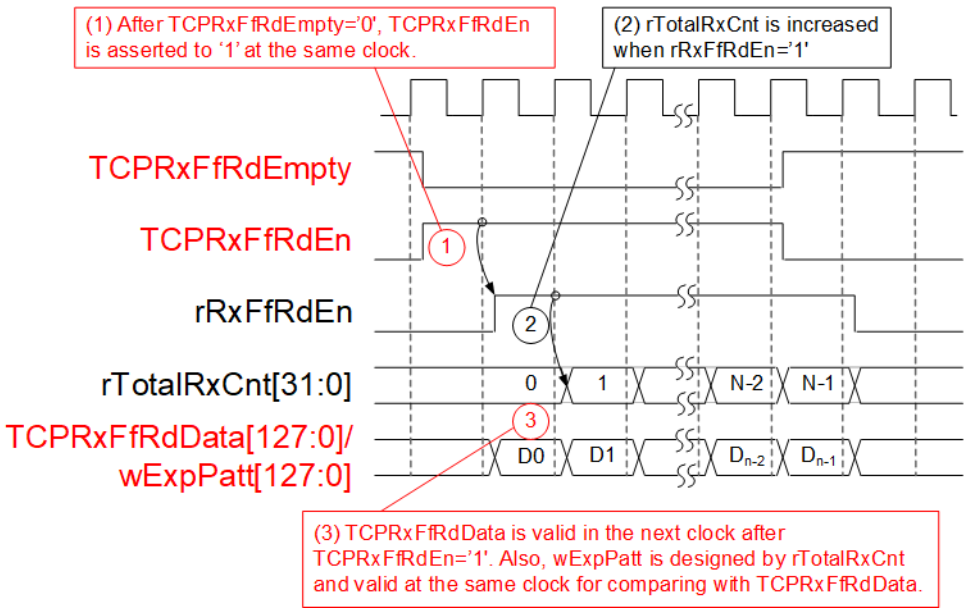

Figure 2-16 PattVer Timing diagram

Figure 2-15 shows the details of PattVer logic for reading the data from TOE25G-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 2-16.

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 amount of received data. rTotalRxCnt is used to generate wExpPatt, so wExpPatt is valid at the same time as TCPRxFfRdData valid.

3 CPU Firmware on FPGA

After FPGA boot-up, 25G Ethernet link up status (USER_RST_INTREG[16]) is polling. The CPU waits until ethernet link is established. Next, welcome message is displayed and user selects the initialization mode of TOE25G-IP to be Client, Server, or Fixed-MAC.

- When testing by FPGA and PC, it is recommended to initialize TOE25G-IP in Client mode. After PC receives ARP request from TOE25G-IP, PC returns 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 set as follows. If the first board sets Server mode, another board must be set to Client mode. Otherwise, the first board is set to Fixed-MAC mode while another board is 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 updated parameters. The example when the system is initialized in Client mode by using default parameters is shown in Figure 3-1.

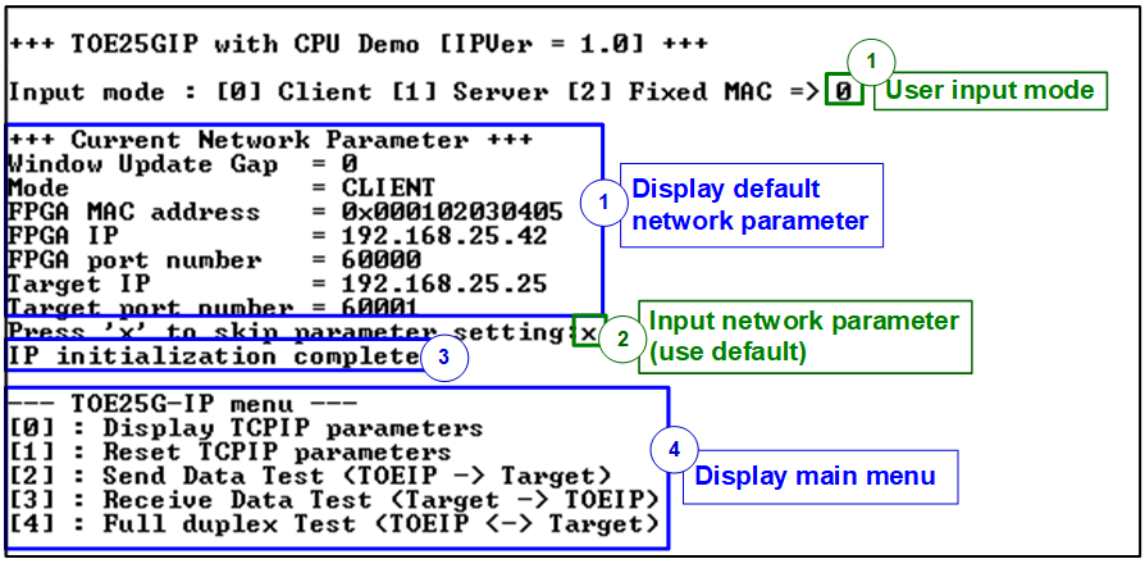

Figure 3-1 System initialization in Client mode by using default parameters

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 TOE25G-IP finishing initialization process (TOE_CMD_INTREG[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 TOE25G-IP, i.e., the initialization mode, Windows update threshold, Reverse packet enable, 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 TOE25G-IP parameters such as IP address and source port number. After setting updated parameter to TOE25G-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 the 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 if the input is valid or not. When the input is invalid, the parameter is not updated.
4) Reset PattGen and PattVer logic by sending reset to user logic (USER_RST_INTREG[0]='1').
5) Force reset to IP by setting TOE_RST_INTREG[0]='1'.
6) Set all parameters to TOE25G-IP register such as TOE_SML_INTREG and TOE_DIP_INTREG.
7) De-assert IP reset by setting TOE_RST_INTREG[0]='0'. After that, TOE25G-IP starts the initialization process.
8) Monitor IP busy flag (TOE_CMD_INTREG[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 running 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_INTREG), reset flag to clear initial value of test pattern (USER_RST_INTREG[0]='1'), and command register to start data pattern generator (USER_CMD_INTREG=0). After that, test pattern generator in UserReg starts sending data to TOE25G-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_INTREG=2 (Open port). After that, it waits until Connection interrupt status (USER_INT_INTREG[0]) is equal to '1'. If busy flag of TOE25G-IP (TOE_CMD_INTREG[0]) is de-asserted to '0' but interrupt is not asserted, the error message is displayed and then it returns to the main menu.
   b) For passive open, CPU waits until connection is opened by another device (PC or FPGA). Connection interrupt status (USER_INT_INTREG[0]) is monitored until it is equal to '1'.
5) Set packet size to TOE25G-IP register (TOE_PKL_INTREG) and calculate total number of loops from total transfer size. Maximum transfer size of each loop is 4 GB. The operation of each loop is as follows.
   i) Set transfer size of this loop to TOE25G-IP register (TOE_TDL_INTREG). Transfer size is fixed to 4 GB except the last loop which is equal to the remaining size.
   ii) Set Send command to TOE25G-IP register (TOE_CMD_INTREG=0).
   iii) Wait until operation is completed by monitoring busy flag (TOE_CMD_INTREG[0]='0'). During monitoring busy flag, CPU reads current amount of transmitted data from user logic (USER_TXLEN_INTREG) and displays the results on the console every second.
6) Set close connection command to TOE25G-IP register (TOE_CMD_INTREG=3). Similar to active open, the operation is successful when Connection interrupt status (USER_INT_INTREG[0]) is asserted to '1'. Otherwise, the error message is displayed if TOE25G-IP busy flag (TOE_CMD_INTREG[0]) is de-asserted without the Connection interrupt asserted.
7) Calculate performance and show test result on the console.

3.4 Receive data test

User sets total amount of received data, 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. While running 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_INTREG[0]='1') and data verification mode (USER_CMD_INTREG[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_INTREG[2]) is monitored until it is equal to '0'. While monitoring ConnOn, CPU reads current amount of received data from user logic (USER_RXLEN_INTREG) and displays the results on the console every second.
6) Wait until all data are read by user logic completely which is monitored by reading FIFO status (USER_FFSTS_INTREG[15:0]=0).
7) Compare receive length of user logic (USER_RXLEN_INTREG) with the set value from user. If all data is completely received, CPU checks verification result by reading USER_CMD_INTREG[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 data size for both transfer 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_INTREG), reset flag to clear the initial value of test pattern (USER_RST_INTREG[0]='1') and command register to start data pattern generator with data verification mode (USER_CMD_INTREG=1 or 3).
4) Open connection following the connection mode value (similar to Step 4 of Send data test).
5) Set packet size to TOE25G-IP registers (TOE_PKL_INTREG=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.
   i) Set transfer size of this loop to TOE_TDL_INTREG. 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.
   ii) Set Send command to TOE25G-IP register (TOE_CMD_INTREG=0).
   iii) Wait until Send command is completed by monitoring busy flag (TOE_CMD_INTREG[0] ='0'). While monitoring busy flag, CPU reads current amount of transmitted data and received data from user logic (USER_TXLEN_INTREG and USER_RXLEN_INTREG) and displays the results on the console every second.
6) Close connection following connection mode value.
   a. For active close, CPU waits until total amount of received data is equal to the set value from user. Then, set USER_CMD_INTREG=3 to close connection. Next, CPU waits until the Connection interrupt (USER_INT_INTREG[0]]) is asserted. Otherwise, the error message is displayed if TOE25G-IP busy flag (TOE_CMD_INTREG[0]) is de-asserted without the Connection interrupt asserted.
   b. For passive close, CPU waits until the connection is closed by another device (PC or FPGA). Connection interrupt status (USER_INT_INTREG[0]) is monitored until it is equal to '1'.
7) Check the result and the error (similar to Step 6-7 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 TOE25G-IP operation.

int 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	0: The open/close connection is successful -1: Fail to open/close the connection
Description	For active mode, write TOE_CMD_INTREG to open or close connection, depending on port_ctl mode. After that, monitor connection status interrupt from bit0 of USER_INT_INTREG register until it is asserted. After that, the interrupt flag is cleared.

void init_param(void)	
Parameters	None
Return value	None
Description	Ask the user if the parameters are updated. After that, set the parameters to TOE25G-IP registers from global parameters. After reset is de-asserted, it waits until TOE25G-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, Reverse packet enable, 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 parameters 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_INTREG register and return only bit2 value to show connection status.

void show_cursize(void)	
Parameters	None
Return value	None
Description	Read USER_TXLEN_INTREG and USER_RXLEN_INTREG and then display the current amount of transmitted data and received data 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 TOE25G-IP such as IP address, MAC address, and port number.

void show_result(void)	
Parameters	None
Return value	None
Description	Read USER_TXLEN_INTREG and USER_RXLEN_INTREG to display total amount of transmitted data and received data. Next, 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_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 described in topic 3.5

void wait_ethlink(void)	
Parameters	None
Return value	None
Description	Read USER_RST_INTREG[16] and wait until ethernet connection is established.

4 Test Software on PC

4.1 "tcpdatatest" for half duplex test

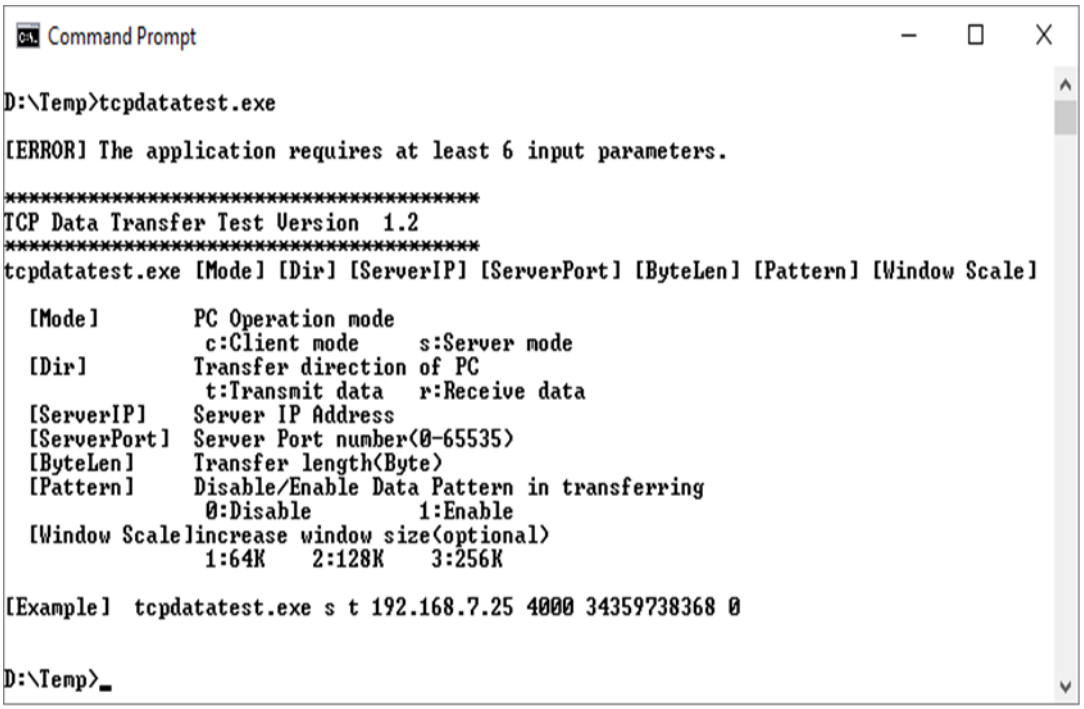

Figure 4-1 "tcpdatatest" application usage

"tcpdatatest" is designed to run on PC for sending 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 192.168.25.42)
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 terminated. In transmit mode, ByteLen must be equal to the total transfer size on FPGA that is 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.
6) Send data out and read total amount of 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 amount of received data.
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 amount of received data 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

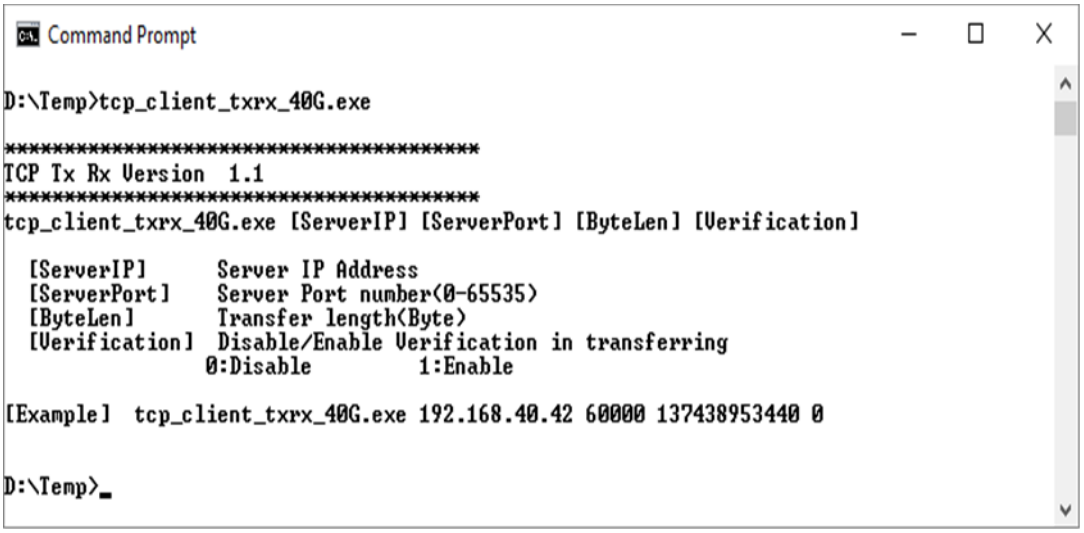

Figure 4-2 "tcp_client_txrx_40G" application usage

"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 TOE25G-IP). As shown in Figure 4-2, there are four parameters to run the application, described as follow.

1) ServerIP : IP address of FPGA
2) ServerPort : Port number of FPGA
3) ByteLen : Total transfer size in byte unit. This is total amount of transmitted data and received 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 amount of received data.
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 amount of transmitted data and received data every second.
10) Repeat step 5) – 9) until total amount of transmitted data and received data 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
1.4	4-Jan-23	- Add USER_INT_INTREG to register map and update firmware to use connection interrupt. - Enable RS-FEC on 25G Ethernet System
1.3	26-May-22	Support Ethernet MAC Subsystem
1.2	2-Sep-20	Support FPGA <-> FPGA test
1.1	24-Aug-20	Rename IP from TenGEMAC to 10G25GEMAC
1.0	6-Aug-20	Initial version release