TOE25G-IP with CPU reference design

TOE25G-IP with CPU reference design

1 Introduction

The TCP/IP is the Internet Protocol Suite for networking applications, consisting of four layers: Application, Transport, Internet, and Network Access. Figure 1-1 shows how the Network Access layer is split into two sublayers, Link and Physical, to link them with the hardware implementation using an FPGA.
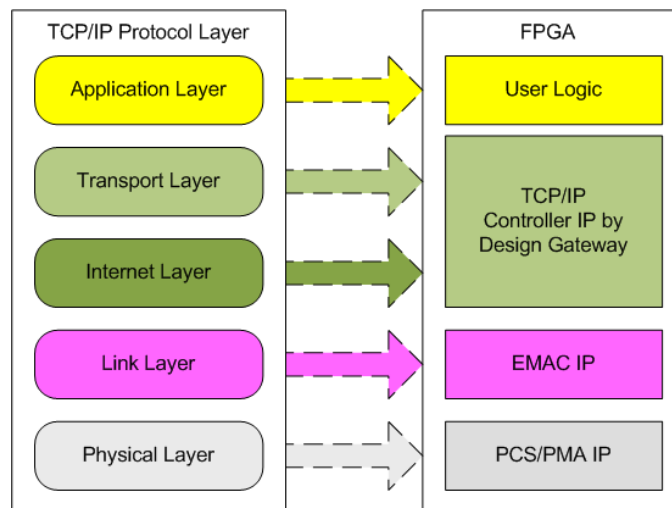

Figure 1-1 TCP/IP protocol layer

The Link and Physical layers are implemented by EMAC-IP and PCS/PMA-IP which may be built as separated IP or one IP. There are several IP providers for these features and this document shows two solutions, the IP core by Design Gateway and the IP core by Xilinx. More information of these solutions is described later in this document.

The TOE25G-IP implements the Transport and Internet layer of TCP/IP Protocol using full hardwire logic, without the need for a CPU or DDR. This allows the user logic to be designed for processing the TCP payload data at the user interface of TOE25G-IP. The TOE25G-IP is responsible for building an Ethernet packet that contains the TCP payload data form the user and transmitting it to the Ethernet MAC (EMAC). If the user data size is too large to fit in one Ethernet packet, the TOE25G-IP will split the data into multiple packets to send it. To construct a complete Ethernet packet, the TOE25G-IP must process and append the TCP/IP header to the packet before transmitting. On the other hand, when the TOE25G-IP receives an Ethernet packet from EMAC, it extracts and verifies the packet. If the packet is valid, the TOE25G-IP extracts TCP payload data from the packet and forwards it to the user logic. Otherwise, the packet is rejected.

The reference design includes a simple user logic, TOE25G-IP, and 25G Ethernet for data transfer using TCP/IP protocol. The data can be transferred with the target, which may be a PC or another FPGA with integrating TOE25G-IP. Design Gateway provides two test applications on the PC for testing the demo using one TCP session, "tcpdatatest" for half-duplex (send or receive data test) and "tcp_client_txrx_xg" for full-duplex (send and receive data simultaneously) using one TCP session.

To provide flexibility in testing, the user can set test parameters and control the operation of the TOE25G-IP demo via UART, which is integrated with a CPU system. The user can monitor the current status and set test parameters through the console. The CPU firmware is built using a simple bare-metal OS. More details of the demo are described below.
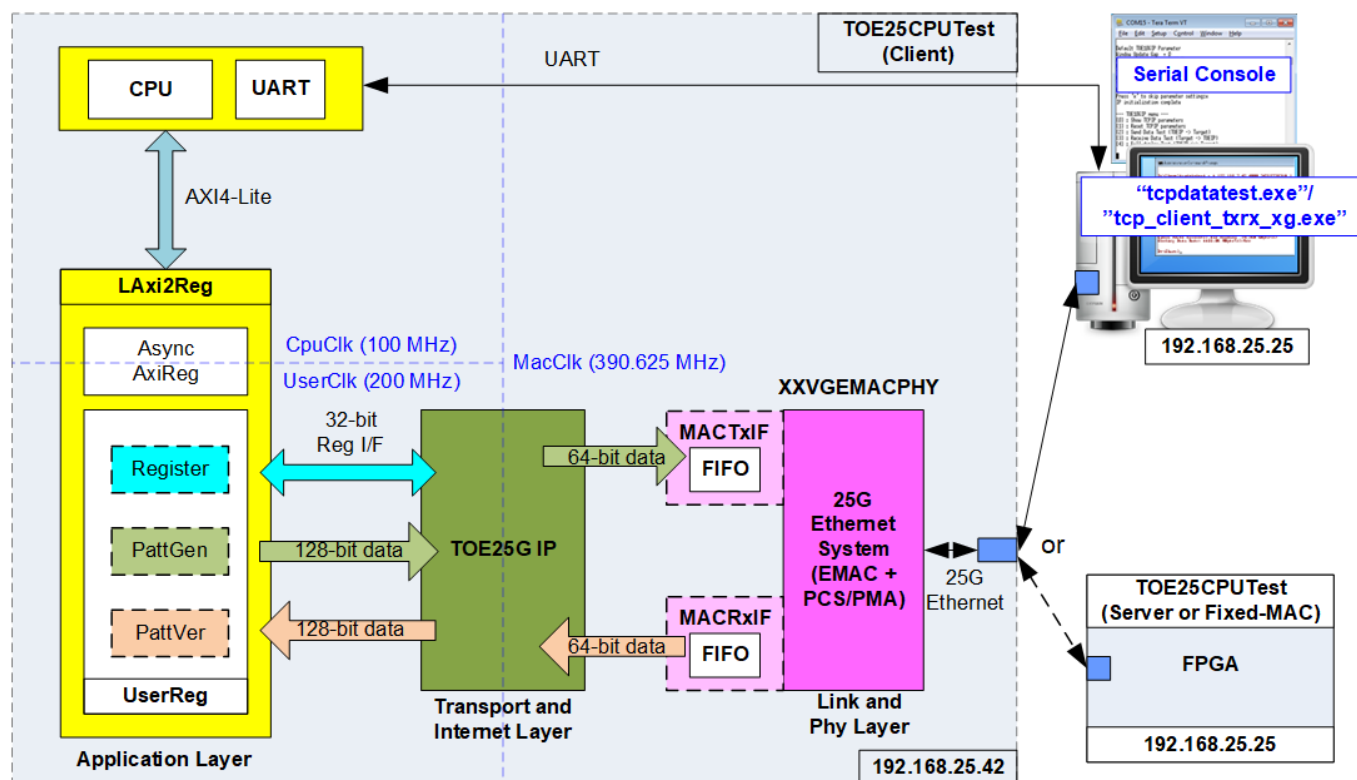
2 Hardware overview

Figure 2-1 Demo block diagram

In the test environment, two devices are used to transfer data over a 25G Ethernet connection. The first device is an FPGA which is initialized in Client mode, and the second device can be a PC or another FPGA which is initialized in Server mode. When using two FPGAs, there are additional initialization options available, i.e., Client <–> Fixed-MAC or Fixed-MAC <-> Fixed-MAC. Two test applications (tcpdatatest and tcp_client_txrx_xg) are provided for transferring data between the PC and FPGA.

In the FPGA system, the TOE25G-IP is connected to the 25G Ethernet System to implement all TCP/IP layers. Figure 2-1 shows two options for connecting the TOE25G-IP with 25G Ethernet System. In the first option, Design Gateway IP core, the 10G25G EMAC-IP or the 25GEMAC/PCS+RS-FEC IP directly connects to the TOE25G-IP via a 64-bit AXI4-Stream interface. In the second option, Xilinx IP core, the 25G Ethernet (MAC) Subsystem connects to MACTxIF and MACRxIF to include small buffer for transferring data stream between TOE25G-IP and the 25G Ethernet (MAC) Subsystem.

The user interface of the TOE25G-IP is connected to UserReg within AsyncAxiReg, which includes a Register file for interfacing with the Register interface, PattGen for sending test data via Tx FIFO interface, and PattVer for verifying test data via Rx FIFO interface. The Register file of UserReg is controlled by CPU firmware through the AXI4-Lite bus.

The test design uses three clock domains: CpuClk for the CPU system, MacClk for interfacing with the 25G Ethernet System, and UserClk for the user logic of the TOE25G-IP. Therefore, AsyncAxiReg is designed to support asynchronous signal transferring between CpuClk and UserClk. More information about each module inside the TOE25CPUTest is described below.

Note:
1. UserClk can be modified to use the same clock as CpuClk to reduce clock resource.
2. The UserClk frequency of TOE25G-IP is recommended to be 195.3125 MHz or higher.

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

The 25G Ethernet System comprises the MAC layer, PCS layer, and PMA layer to interface with the external device by 25G BASE-SR. To connect with TOE25G-IP, the user interface of 25G Ethernet System is 64-bit AXI4-stream, operated at 390.625 MHz. Figure 2-2 shows the 25G Ethernet System solutions using the IP core from Design Gateway and Xilinx. The default reference design enables the RS-FEC feature, so the network equipment of the test environment must enable RS-FEC feature. Please contact us for the demo system that disables RS-FEC feature.
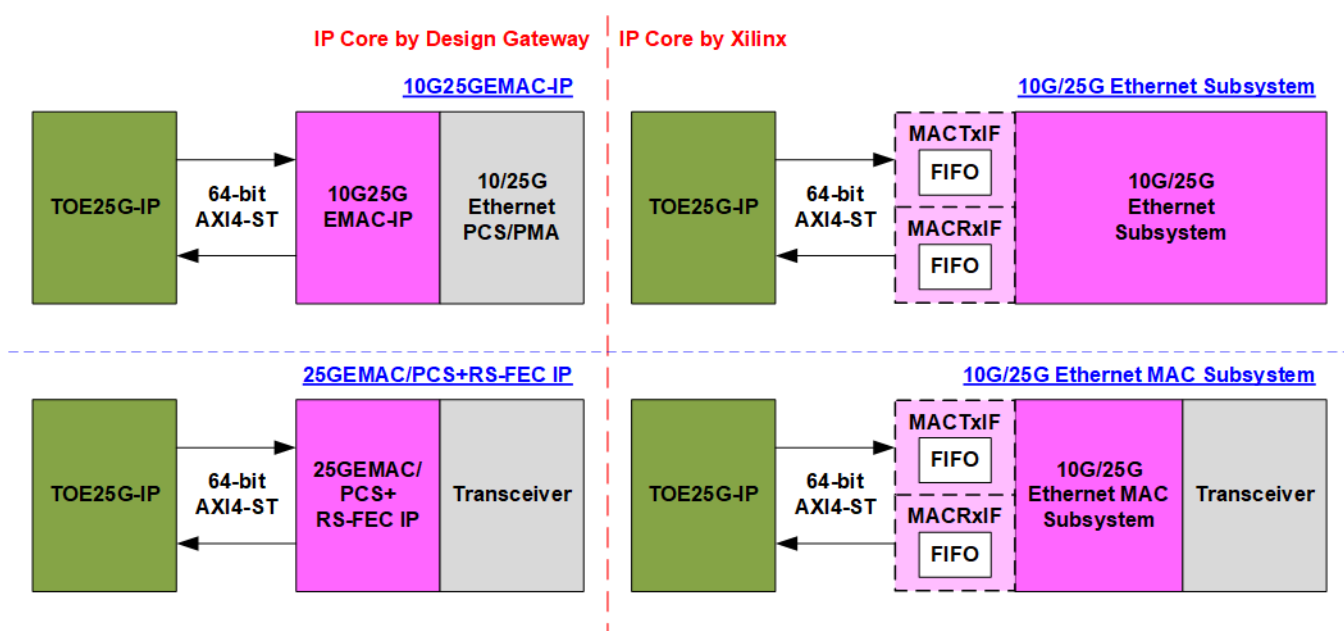

Figure 2-2 TOE25G-IP with several Ethernet system solutions

25G Ethernet System by Design Gateway

Design Gateway offers two IP cores as solutions for the 25G Ethernet system, which can directly connect to TOE25G IP. The first solution is the 10G25GEMAC IP Core, which implements Ethernet MAC features with minimal IP resource usage and low latency time. The PCS/PMA module can be created using the Xilinx's IP wizard. The Ethernet speed, whether 10G or 25G, can be configured from the PCS/PMA module, provided by from Xilinx. The second solution is the 25GEMAC/PCS + RS-FEC IP core, which implements Ethernet MAC, PCS, and RS-FEC features for 25G Ethernet with optimized resource utilization and low latency. Additional information about each IP core can be found on the following website.

- 10G25GEMAC IP Core
https://dgway.com/products/IP/10GEMAC-IP/dg_10g25gemacip_data_sheet_xilinx_en.pdf
- 10G/25G Ethernet PCS/PMA (BASE-R)
https://www.xilinx.com/products/intellectual-property/ef-di-25gemac.html
- 25GEMAC/PCS + RS-FEC IP Core
https://dgway.com/products/IP/GEMAC-IP/dg_xxvgmacrsfecip_data_sheet_xilinx.pdf

25G Ethernet System by Xilinx

Xilinx also provides two types of 25G Ethernet system: a hard IP core and a soft IP core. The soft IP core is the 10G/25G Ethernet Subsystem, which includes Ethernet MAC, PCS, and PMA features. On the other hand, the hard IP core is the Ethernet MAC Subsystem, implementing Ethernet MC and PCS features. The PMA must be generated using the Transceiver IP wizard. The hard IP core is available on the Versal devices. To connect the TOE25G IP with Xilinx Ethernet system IP, an adapter logic with a small FIFO is required due to the different characteristics of the ready/valid signals. More information regarding these characteristics is described in the EMAC interface topic. For further details about Xilinx's Ethernet solutions, please visit the following link.

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

The details of the adapter logics, MAC25GTxIF and MAC25GRxIF, are described below.
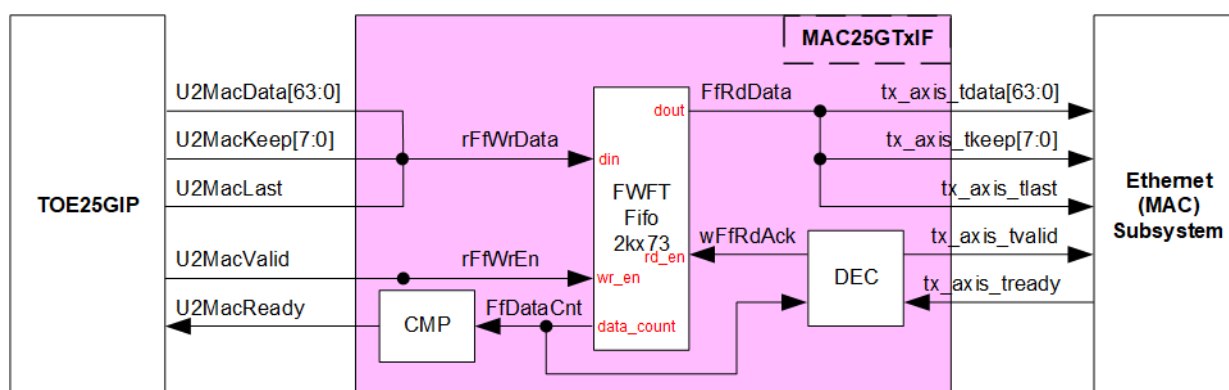
MAC25GTxIF

Figure 2-3 MAC25GTxIF Block diagram

Tx interface timing diagram of Ethernet (MAC) Subsystem and TOE25G-IP are different. TOE25G-IP needs to send each packet continuously, but the Xilinx Ethernet (MAC) Subsystem does not support this. The EMAC may pause receiving data by de-asserting tx_axis_tready signal before the end of the packet.

To solve this issue, the MAC25GTxIF is designed to store transmitted data from TOE25G-IP when EMAC is not ready to receive new data. The FIFO depth is 2048 to store at least one data packet during pausing time. Since the maximum packet size of TOE25G-IP reference design is 8960 bytes or 1120 of 64-bit data, the FIFO depth of 2048 is sufficient. 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 1b.

The operation of MAC25GTxIF is divided into two parts: Writing FIFO and Reading FIFO. Timing diagrams of each part are illustrated in Figure 2-4 and Figure 2-5.
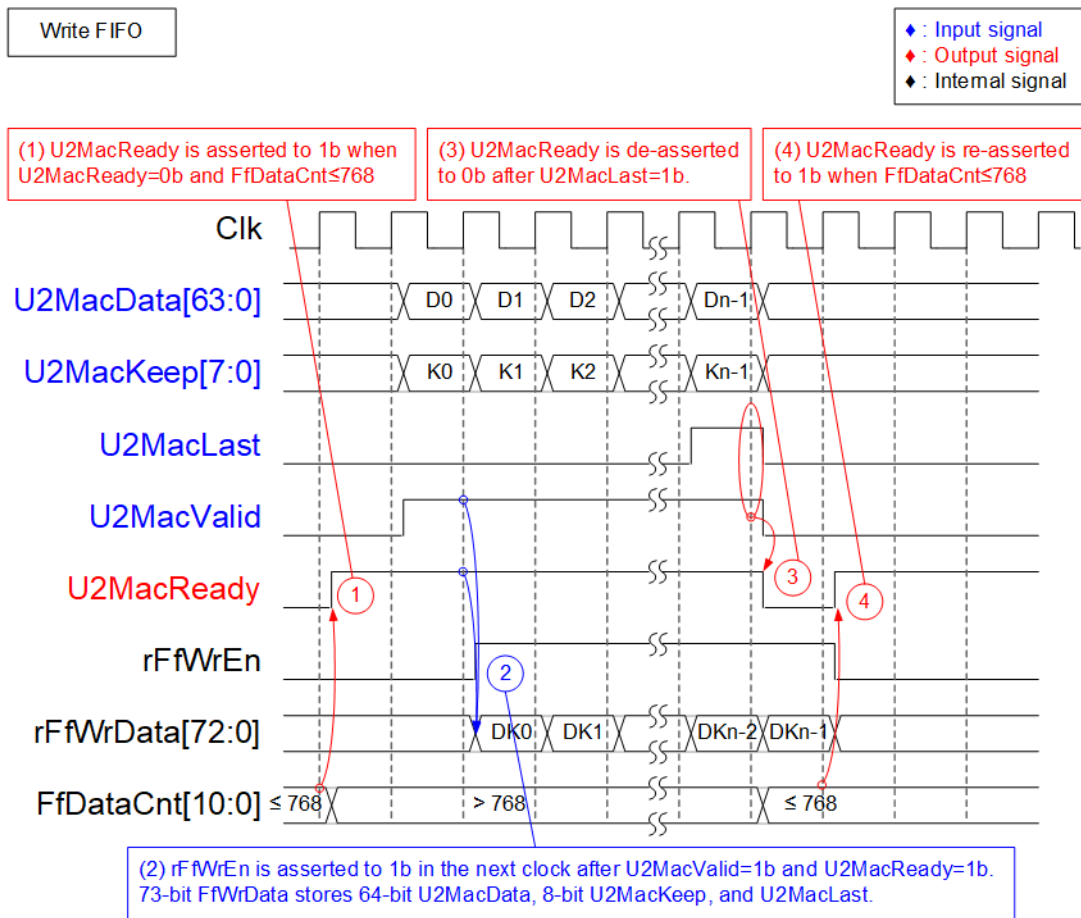
Write FIFO

♦ : Input signal
♦ : Output signal
♦ : Internal signal

(1) U2MacReady is asserted to 1b when U2MacReady=0b and FfDataCnt≤768

(3) U2MacReady is de-asserted to 0b after U2MacLast=1b.

(4) U2MacReady is re-asserted to 1b when FfDataCnt≤768

(2) rFfWrEn is asserted to 1b in the next clock after U2MacValid=1b and U2MacReady=1b. 73-bit FfWrData stores 64-bit U2MacData, 8-bit U2MacKeep, and U2MacLast.

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

1) Before asserting U2MacReady to 1b to receive a new packet from user, two conditions must be satisfied. First, it must be enough free space in FIFO to store a maximum packet size, 9014 bytes. To verify this, the upper bit of FfDataCnt is read to ensure that the amount of data in FIFO is not greater than 768 (indicating that there is at least 1279 of 64-bit free space). Second, the previous packet must have been completely transferred, as monitored by U2MacReady being set to 0b.

2) The user begins transmitting a packet by asserting U2MacValid to 1b. The input signals from user (U2MacData, U2MacKeep, and U2MacLast) are considered valid and stored in the FIFO when both U2MacValid and U2MacReady are asserted to 1b. Following this, the inputs are stored in the FIFO by asserting rFfWrEn to 1b. The 73-bit Write data to the FIFO includes 64-bit data (U2MacData), 8-bit byte enable (U2MacKeep), and the end flag (U2MacLast).

3) Once the final data of a packet is received (U2MacLast=1b and U2MacValid=1b), U2MacReady is de-asserted to 0b to pause data transmission, allowing for FfDataCnt to be read.

4) If FfDataCnt indicates that there is sufficient free space in the FIFO, U2MacReady will be re-asserted to 1b in the next cycle.

Read FIFO

(1) tx_axis_tvalid is asserted to 1b when FfDataCnt[10:2]≠0 and tx_axis_tvalid=0b. tx_axis_tdata, tx_axis_tkeep, and tx_axis_tlast are valid for sending the first data of a packet.

(4) tx_axis_tvalid is de-asserted to 0b after transferring the final data (tx_axis_tlast=1b, tx_axis_tvalid=1b, and tx_axis_tready=1b)

♦ : Input signal
♦ : Output signal
♦ : Internal signal

(2) wFfRdAck is asserted to read the next data when tx_axis_tvalid and tx_axis_tready are asserted to 1b.

(5) tx_axis_tvalid is re-asserted to 1b for transferring the next packet if FIFO has enough data.

(3) wFfRdAck is de-asserted to 0b if tx_axis_tready is de-asserted to 0b. While tx_axis_tvalid, tx_axis_tdata, tx_axis_tkeep, and tx_axis_tlast hold the same value.
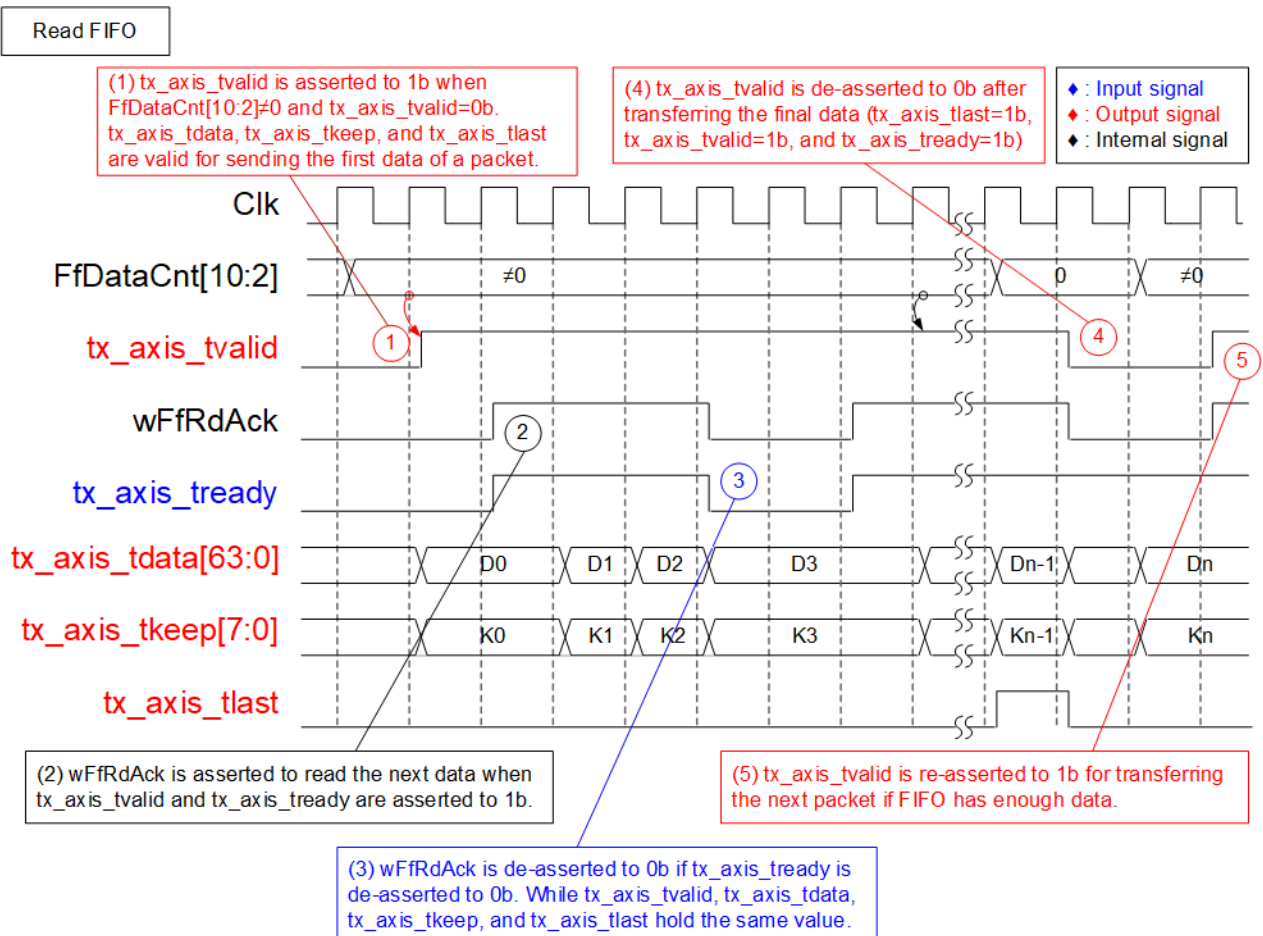
Figure 2-5 Timing diagram for transferring data from FIFO to EMAC

1) The transmission of a new packet begins only when the FIFO has data stored (FfDataCnt[10:2] ≠ 0) and there is no packet transmitting (tx_axis_tvalid=0b). To initiate the transmission, the tx_axis_tvalid signal is asserted to 1b, along with the valid output signals to EMAC which include 64-bit tx_axis_tdata, 8-bit tx_axis_tkeep, and tx_axis_tlast.

2) Upon complete transmission of data to EMAC (tx_axis_tvalid=1b and tx_axis_tready=1b), wFfRdAck is asserted to 1b to read the next set of data from FIFO.

3) If tx_axis_tready is de-asserted to 0b, wFfRdAck is de-asserted to 0b to pause the reading of new data from FIFO. This means that all output signals sent to EMAC will hold the same value until EMAC re-asserts tx_axis_tready to 1b.

4) After the final data of a packet is fully transferred (tx_axis_tlast=1b and tx_axis_tready=1b), tx_axis_tvalid is de-asserted to 0b to pause the data transmission, and allowing the checking of data size in FIFO for transferring the next packet.

5) The transmission of the next packet begins when the FIFO has sufficient data. The process returns to step 1) to transmit the new packet.
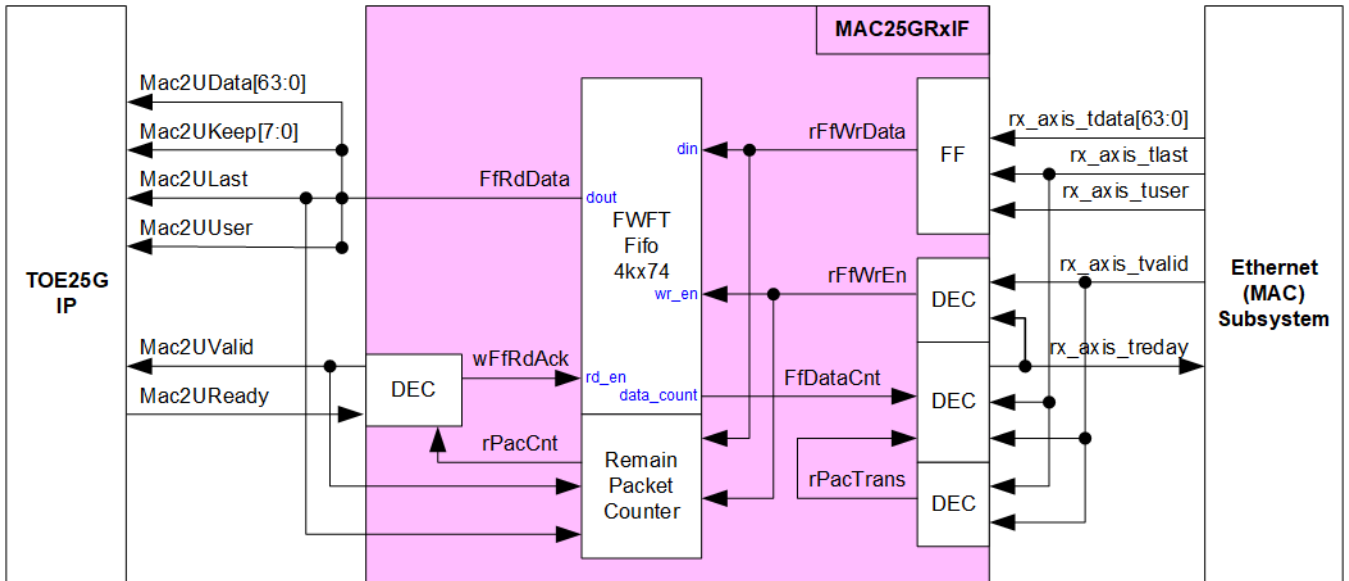
MAC25GRxIF

Figure 2-6 MAC25GRxIF Block diagram

The EMAC interface of TOE25G-IP requires to receive the packets continuously, so Mac2UValid must remain set to 1b from the beginning to the end of each packet. However, the Xilinx Ethernet (MAC) Subsystem may pause data transmission by de-asserting the valid signal (rx_axis_tvalid) before the packet is complete.

To solve this issue, the MAC25GRxIF is designed to store all packet data transmitted from EMAC before forwarding it to TOE25G-IP. This guarantees that all data is available for transfer until the end of the packet. The FIFO has a depth of 4096, which can store multiple Ethernet packets, and operates as a First-Word Fall-Through FIFO.

The Remain packet counter keeps track of the number of packets stored in the FIFO. When a new packet is received from EMAC, the counter is increased, and it decreased when the packet is completely forwarded to TOE25G-IP.

The logic of MAC25GRxIF is split into two groups: Writing FIFO and Reading FIFO. The timing diagrams for each group are displayed in Figure 2-7 and Figure 2-8.
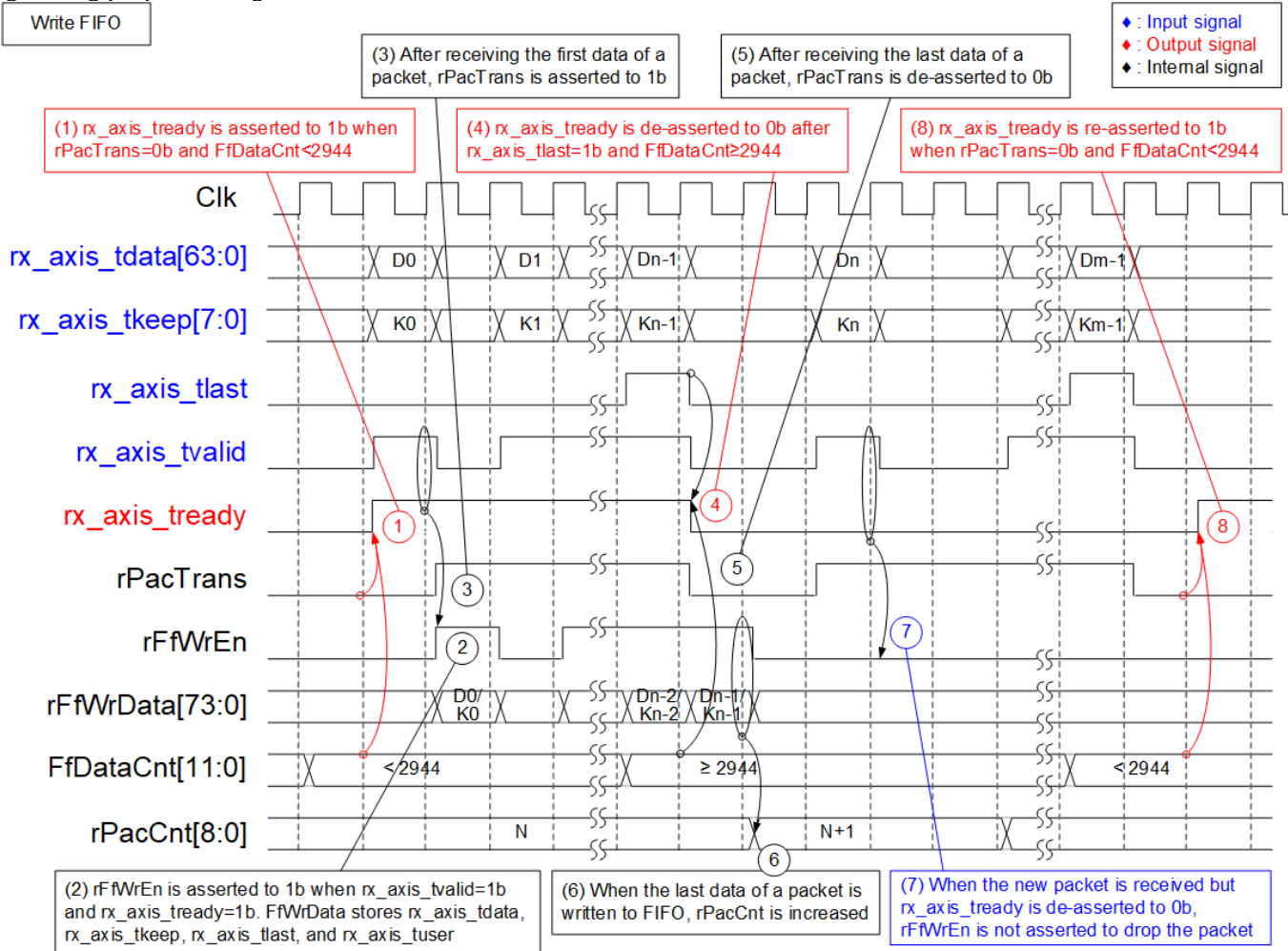
Write FIFO

(3) After receiving the first data of a packet, rPacTrans is asserted to 1b

(5) After receiving the last data of a packet, rPacTrans is de-asserted to 0b

♦ : Input signal
♦ : Output signal
♦ : Internal signal

(1) rx_axis_tready is asserted to 1b when rPacTrans=0b and FfDataCnt<2944

(4) rx_axis_tready is de-asserted to 0b after rx_axis_tlast=1b and FfDataCnt≥2944

(8) rx_axis_tready is re-asserted to 1b when rPacTrans=0b and FfDataCnt<2944

Clk

rx_axis_tdata[63:0] D0 D1 Dn-1 Dn Dm-1

rx_axis_tkeep[7:0] K0 K1 Kn-1 Kn Km-1

rx_axis_tlast

rx_axis_tvalid

rx_axis_tready ① ④ ⑧

rPacTrans ③ ⑤

rFfWrEn ② ⑦

rFfWrData[73:0] D0/K0 Dn-2/Kn-2 Dn-1/Kn-1

FfDataCnt[11:0] <2944 ≥2944 <2944

rPacCnt[8:0] N N+1 ⑥

(2) rFfWrEn is asserted to 1b when rx_axis_tvalid=1b and rx_axis_tready=1b. FfWrData stores rx_axis_tdata, rx_axis_tkeep, rx_axis_tlast, and rx_axis_tuser

(6) When the last data of a packet is written to FIFO, rPacCnt is increased

(7) When the new packet is received but rx_axis_tready is de-asserted to 0b, rFfWrEn is not asserted to drop the packet

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

1) The free space size in FIFO is checked by reading FfDataCnt. If it is less than 2944 (the free space in FIFO is more than 1153), this is sufficient to store the maximum packet size (9014 bytes). Also, rPacTrans must be equal to 0 to confirm that the packet is not currently being transmitted. Once these conditions are met, rx_axis_tready is asserted to 1b to begin receiving data from EMAC.

2) When a new packet is ready to be transferred (rx_axis_tvalid is set to 1b), the data and control signals from EMAC are stored in the FIFO, including 64-bit rx_axis_tdata, 8-bit rx_axis_tkeep, rx_axis_tlast, and rx_axis_tuser. rFfWrEn is set to 1b to write the 74-bit data to the FIFO.

3) After the first data of a packet is received, rPacTrans is set to 1b and remains until the end of packet is received. This allows to use rPacTrans for monitoring the packet transmission status.

4) If the final data of a packet is received and there is not enough free space in the FIFO (FFDataCnt≥2944), rx_axis_tready is de-asserted to 0b to pause data reception.

5) After the final data of a packet is received, rPacTrans is de-asserted to 0b to change the packet transmission status from Busy to Idle.

6) Once the final data of a packet has been stored in the FIFO (rFfWrEn=1b and rFfWrData[72] which is last flag =1b), the packet counter (rPacCnt) is increased by 1 to indicate the total number of packets stored in the FIFO.

7) If a new packet is received but rx_axis_tready is still de-asserted to 0b, the received packet will be dropped and not be stored in the FIFO.
8) When there is no packet currently being transmitted and there is enough free space in the FIFO, rx_axis_tready is re-asserted to 1b to resume data reception.
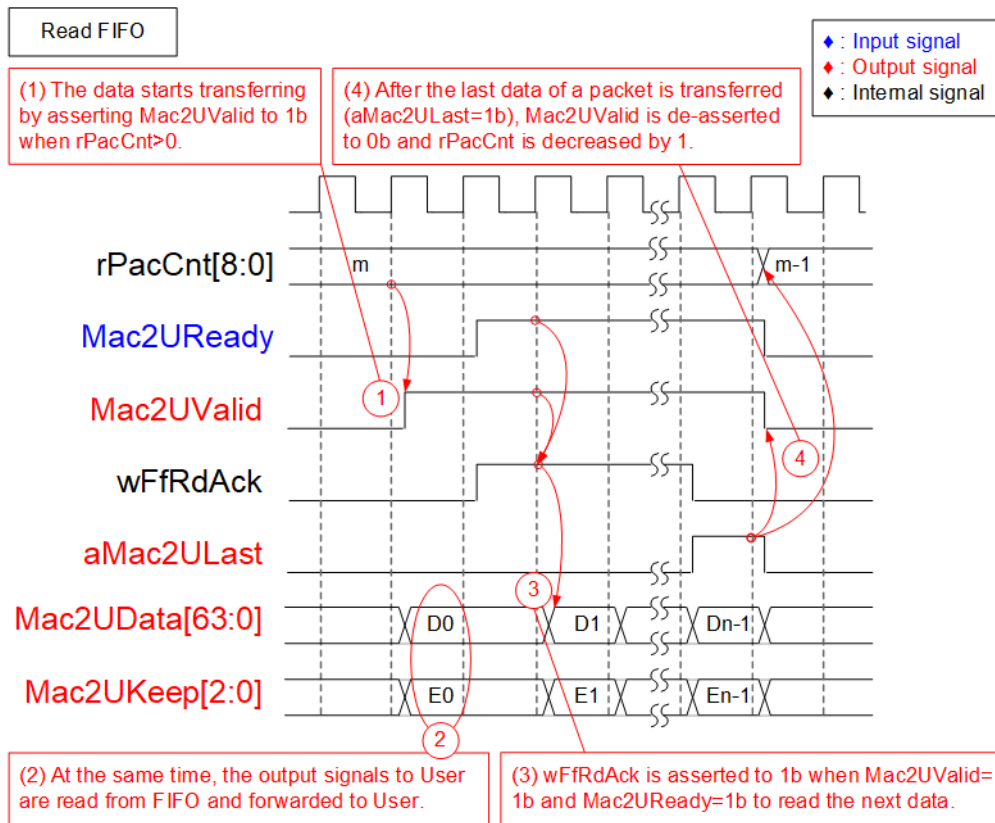

Figure 2-8 Timing diagram for transferring data from FIFO to TOE25G-IP

1) Data transmission from FIFO to TOE25G-IP begins only when there is at least one packet stored in FIFO (indicated by rPacCnt not being equal to 0). To start the transmission, Mac2UValid is asserted to 1b.
2) The data and control signals read from FIFO are sent to TOE25G-IP, including 64-bit Mac2UData, 8-bit Mac2UKeep, Mac2ULast, and Mac2UUser. Mac2UValid remains asserted to 1b until the end of packet to ensure continuous data transmission.
3) Once the data is completely transferred to TOE25G-IP (Mac2UValid=1b and Mac2UReady=1b), wFfRdAck is asserted to 1b to read the next data.
4) After the final data of a packet is transferred (aMac2ULast=1b and Mac2UValid=1b), Mac2UValid and wFfRdAck are de-asserted to 0b to pause a packet transmission. Also, rPacCnt is decreased by 1 after the completion of one packet transfer.

2.2 TOE25G-IP

TOE25G-IP implements TCP/IP stack and fully offload engine without requiring the CPU and the external memory. User interface has two signal groups - control signals and data signals. Control and status signals use Single-port RAM interface for write/read register access. Data signals use FIFO interface for transferring data stream in both directions. More information can be found from the datasheet.
http://www.dgway.com/products/IP/TOE25G-IP/dg_toe25gip_data_sheet_xilinx.pdf

2.3   CPU and Peripherals

The CPU system uses a 32-bit AXI4-Lite bus as the interface to access peripherals such as the Timer and UART. The system also integrates an additional peripheral to access the test logic by assigning a unique base address and address range. To support CPU read and write operations, the hardware logic must comply with the AXI4-Lite bus standard. LAxi2Reg module, as shown in Figure 2-9, is designed to connect the CPU system via the AXI4-Lite interface, in compliance with the standard.
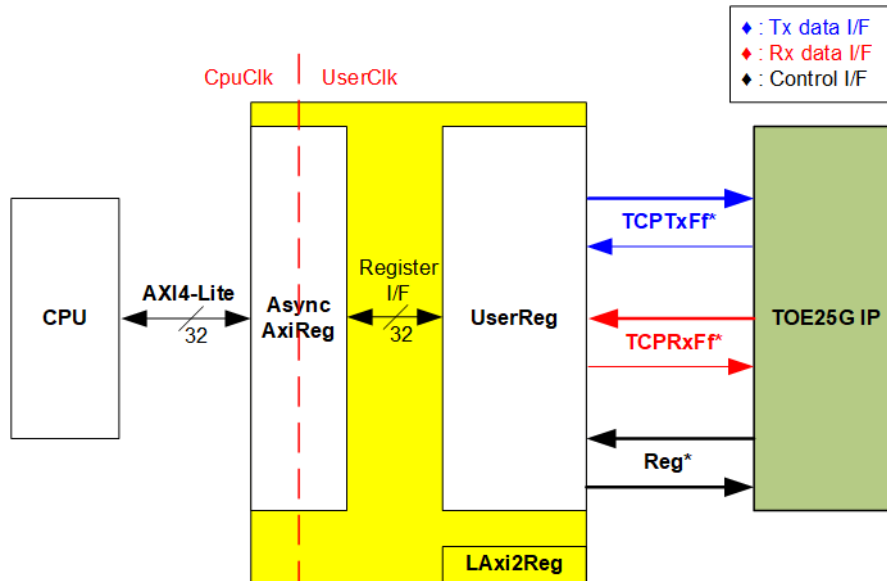
Figure 2-9 LAxi2Reg block diagram

LAxi2Reg consists of AsyncAxiReg and UserReg. AsyncAxiReg converts AXI4-Lite signals into a simple Register interface with a 32-bit data bus size, similar to the AXI4-Lite data bus size. It also includes asynchronous logic to handle clock domain crossing between the CpuClk and UserClk domains.

UserReg includes the Register file of the parameters and the status signals of test logics. Both the data interface and control interface of TOE25G-IP are connected to UserReg. Further details of AsyncAxiReg and UserReg are described below.
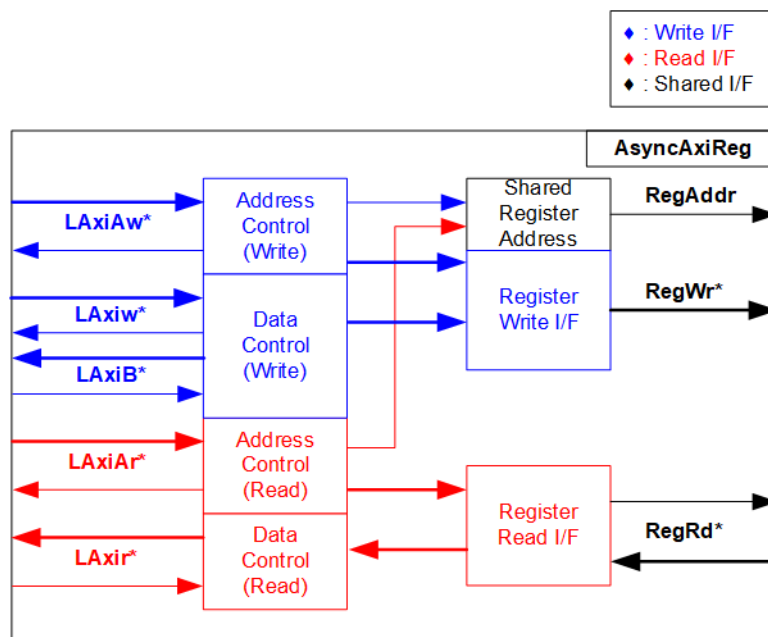
2.3.1 AsyncAxiReg

Figure 2-10 AsyncAxiReg interface

The signal on AXI4-Lite bus interface can be grouped 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 read channel operate independently for both control and data interfaces. Therefore, the logic within AsyncAxiReg to interface with AXI4-Lite bus is divided 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. The Write control I/F and Write data I/F of the AXI4-Lite bus are latched and transferred to become the Write register interface with clock domain crossing registers. Similarly, the Read control I/F of AXI4-Lite bus is latched and transferred to the Read register interface, while Read data is returned from Register interface to AXI4-Lite via clock domain crossing registers. In the Register interface, RegAddr is a shared signal for write and read access, so it loads the value from LAxiAw for write access or LAxiAr for read access.

The Register interface is compatible with single-port RAM interface for write transaction. The read transaction of the Register interface has been slightly modified from RAM interface by adding the RdReq and RdValid signals to control read latency time. The address of Register interface is shared for both write and read transactions, 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.
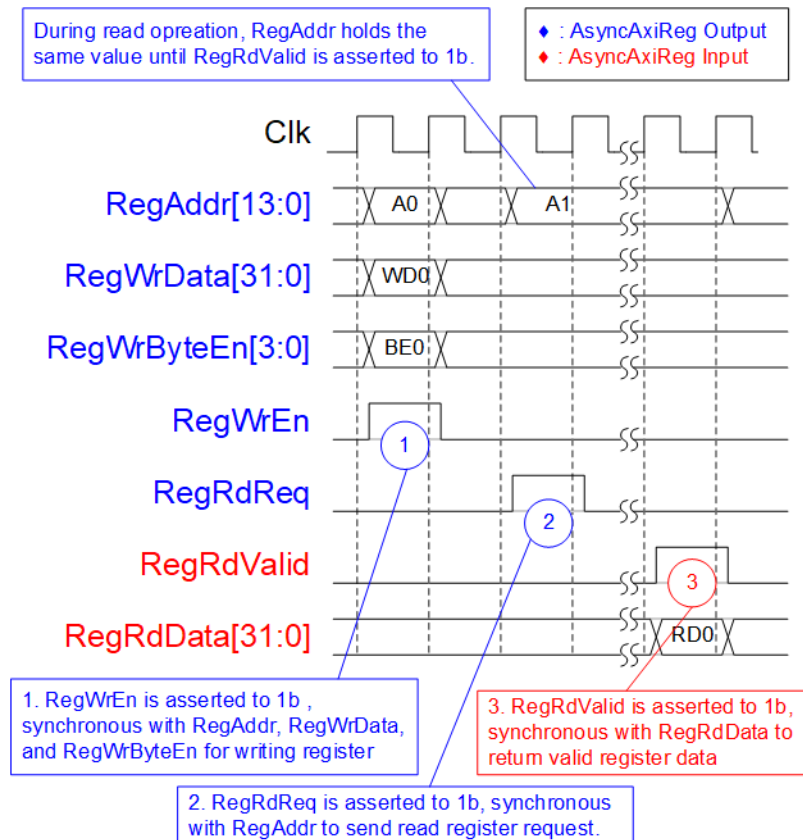
During read opreation, RegAddr holds the same value until RegRdValid is asserted to 1b.

♦ : AsyncAxiReg Output
♦ : AsyncAxiReg Input

1. RegWrEn is asserted to 1b , synchronous with RegAddr, RegWrData, and RegWrByteEn for writing register

2. RegRdReq is asserted to 1b, synchronous with RegAddr to send read register request.

3. RegRdValid is asserted to 1b, synchronous with RegRdData to return valid register data

Figure 2-11 Register interface timing diagram

1) Timing diagram to write register is similar to that of a single-port RAM. The RegWrEn signal is set to 1b, along with a valid RegAddr (Register address in 32-bit units), RegWrData (write data for the register), and RegWrByteEn (write byte enable). The byte enable consists of four bits that indicate the validity of the byte data. For example, bit[0], [1], [2], and [3] are set to 1b when RegWrData[7:0], [15:8], [23:16], and [31:24] are valid, respectively.

2) To read register, AsyncAxiReg sets the RegRdReq signal to 1b with a valid value for RegAddr. The 32-bit data is returned after the read request is received. The slave detects the RegRdReq signal being set to start the read transaction. In the read operation, the address value (RegAddr) remains unchanged until RegRdValid is set to 1b. The address can then be used to select the returned data using multiple layers of multiplexers.

3) The slave returns the read data on RegRdData bus by setting the RegRdValid signal to 1b. After that, AsyncAxiReg forwards the read value to the LAxir* interface.
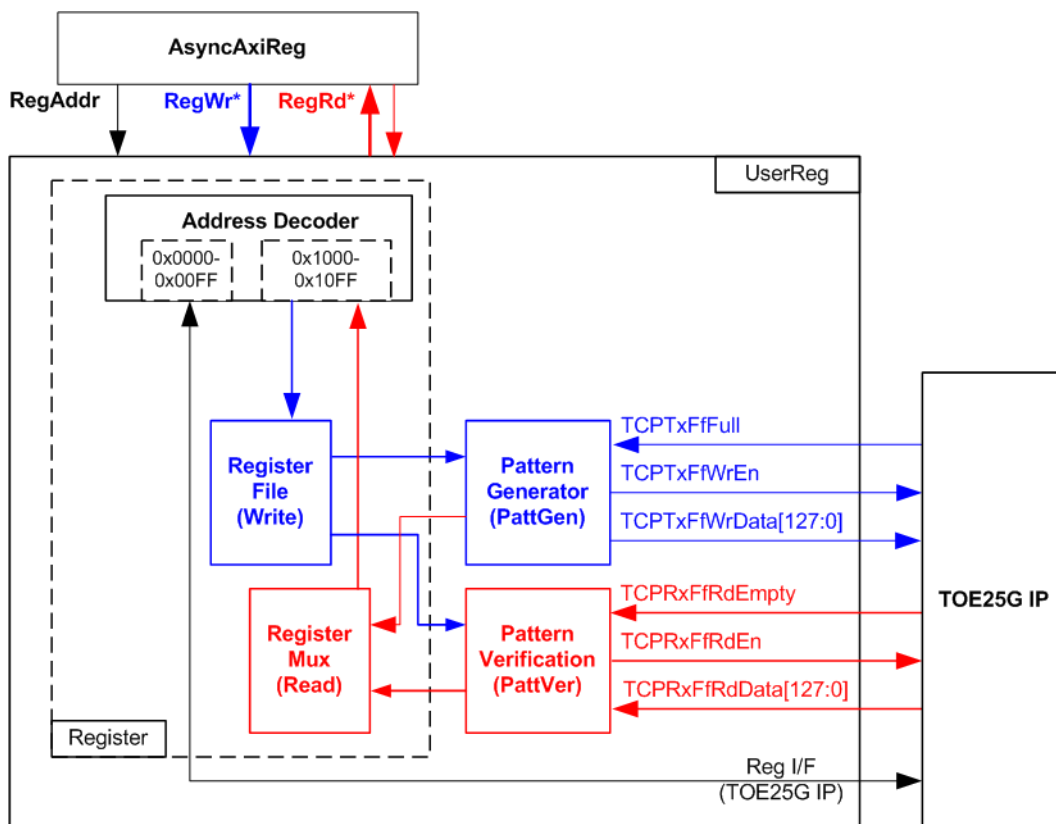
2.3.2 UserReg

Figure 2-12 UserReg block diagram

The UserReg module includes three functions: Register, Pattern generator (PattGen), and Pattern verification (PattVer). The Register block decodes the requested address from AsyncAxiReg and selects the active register for a write or read transaction. The PattGen block is designed to send 128-bit test data to TOE25G-IP following FIFO interface standard, while the PattVer block is designed to read and verify 128-bit data from TOE25G-IP following FIFO interface standard.

Register Block
The address range, mapped to UserReg, is split into two areas: TOE25G-IP register (0x0000-0x00FF) and UserReg register (0x1000-0x10FF). The Address decoder decodes the upper bits of RegAddr to select the active hardware. Since the Register file inside UserReg is 32-bit bus size, Write byte enable (RegWrByteEn) is not used. To write hardware registers, the CPU must use a 32-bit pointer to place a 32-bit valid value on the write data bus.

For reading a register, a multiplexer selects the data to return to CPU by using the address. The lower bits of RegAddr are applied to select the active data within each Register area. While the upper bits are used to select the returned data from each Register area. The total latency time of read data is equal to one clock cycle, and RegRdValid is created by RegRdReq by asserting a D Flip-flop. More details of the address mapping within the 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 0b to start transmitting data. [1] – Data verification enable (0b: Disable data verification, 1b: Enable data verification) Rd [0] – Busy of PattGen inside UserReg (0b: Idle, 1b: PattGen is busy) [1] – Data verification error (0b: Normal, 1b: 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 1b to reset UserReg. This bit is auto-cleared to 0b. [8] – Set 1b to clear TimerInt latched value Rd [8] – Latched value of TimerInt, the output from IP (0b: Normal, 1b: TimerInt=1b has been asserted) This flag can be cleared by system reset condition or setting USER_RST_INTREG[8]=1b. [16] – Ethernet linkup status from Ethernet MAC (0b: Not linkup, 1b: 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.

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

Pattern Generator

The logic diagram and timing diagram of Pattern Generator (PattGen) are illustrated in Figure 2-13 and Figure 2-14, respectively.
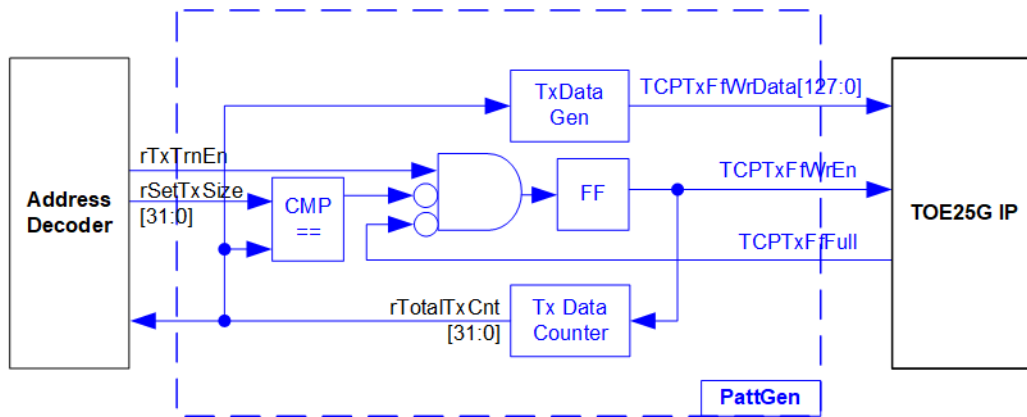
Figure 2-13 PattGen block
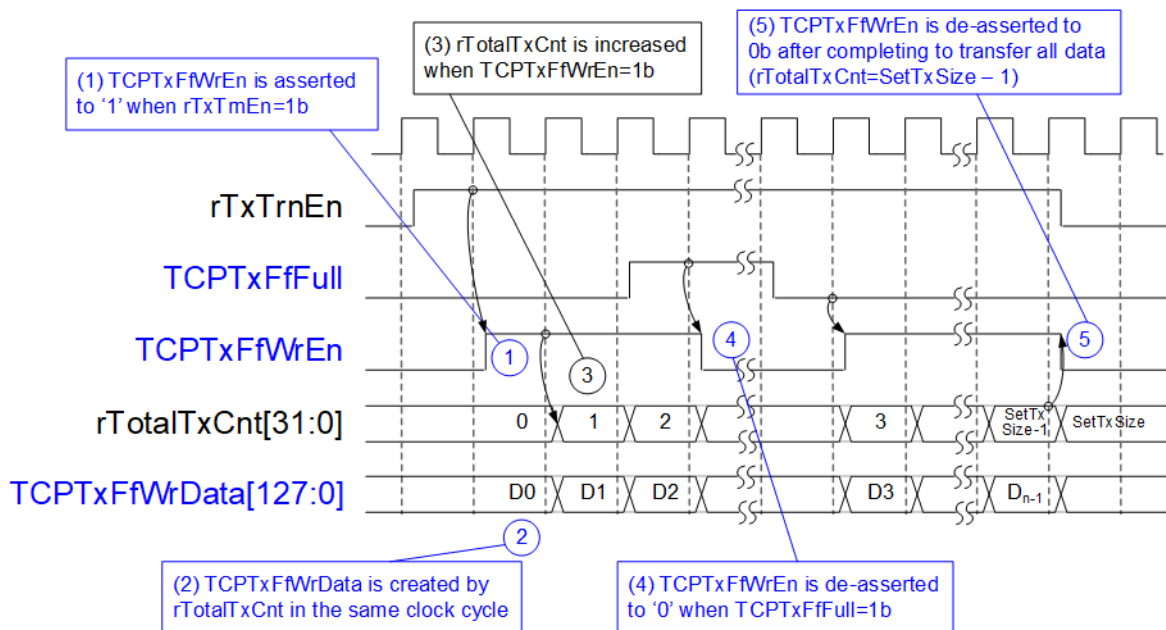
Figure 2-14 PattGen timing diagram

When USER_CMD_INTREG[0] is set to 0b, PattGen initiates the operation of generating test data by setting rTxTrnEn to 1b. While rTxTrnEn remains set to 1b, TCPTxFfWrEn is controlled by TCPTxFfFull. If TCPTxFfFull is 1b, TCPTxFfWrEn is de-asserted to 0b. The data counter, rTotalTxCnt, checks the total amount of data sent to TOE25G-IP. The lower bits of rTotalTxCnt generate 32-bit incremental data for the TOETxFfWrData signal. Once all data has been transferred, equal to rSetTxSize, rTxTrnEn is de-asserted to 0b.

Pattern Verification

The logic diagram and timing diagram of Pattern Verification (PattVer) are illustrated in Figure 2-15 and Figure 2-16, respectively. The verification feature is executed when the verification flag (rVerifyEn) is enabled.
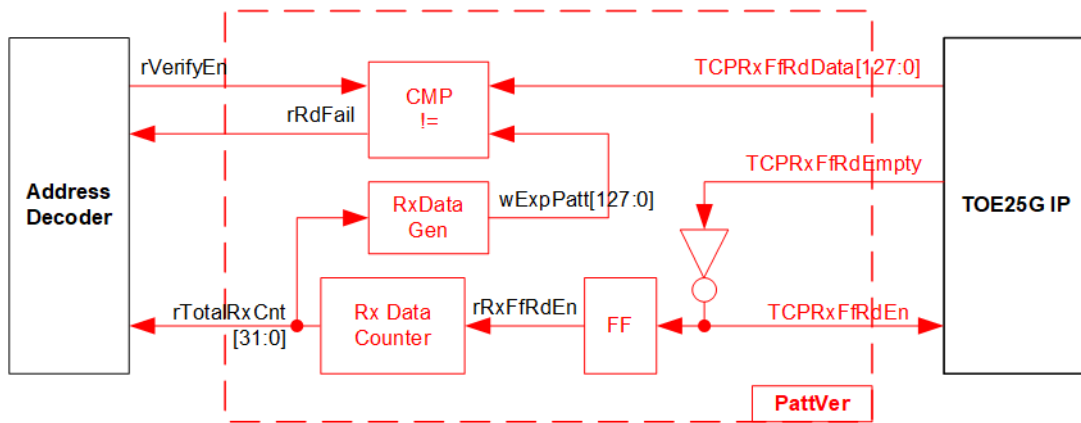
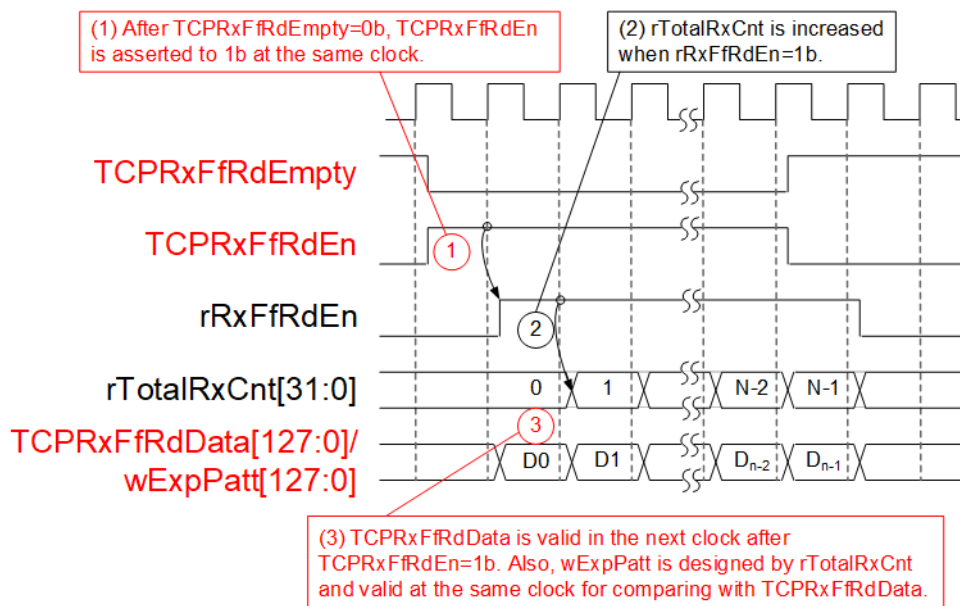

Figure 2-15 PattVer block

Figure 2-16 PattVer Timing diagram

When rVerifyEn is set to 1b, the verification logic is processed. It compares the received data (TCPRxFfRdData) with the expected data (wExpPatt). If comparison fails, rRdFail is asserted to 1b. The TCPRxFfRdEn signal is created by applying NOT logic to TCPRxFfRdEmpty. The data for comparison, TCPRxFfRdData, becomes valid in the next clock cycle. To count the total size of received data, rTotalRxCnt is enabled by rRxFfRdEn, which is delayed by one clock cycle from TCPRxFfRdEn. rTotalRxCnt are applied to generate wExpPatt for comparison with TCPRxFfRdData. Therefore, TCPRxFfRdData and wExpPatt are valid in the same clock cycle and can be compared using rRxFfRdEn signal.

3 CPU Firmware on FPGA

The reference design uses a bare-metal OS for the CPU firmware operating, which facilitates hardware handling. When executing the test system, the first step is to initialize the hardware, described in more details below.
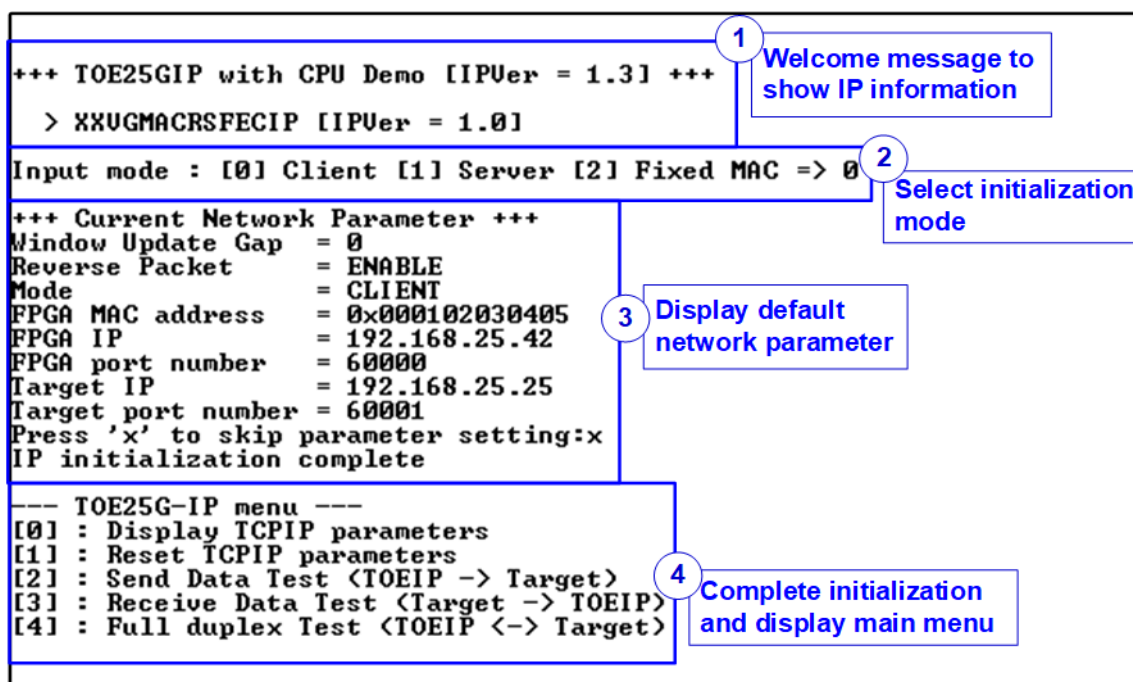

Figure 3-1 System initialization in Client mode by using default parameters

Figure 3-1 illustrates the four-step process for hardware initialization, which is described below.

1) Upon FPGA boot-up, the firmware polls the status of the 25G Ethernet link (USER_RST_INTREG[16]). The CPU waits until the link is up, and then displays a welcome message to show IP information.
2) The menu to select the initialization mode of TOE25G-IP is displayed, allowing the user to choose the Client, Server, or Fixed-MAC mode.

   Note:
   - *When running in Client mode, TOE25G-IP sends an ARP request to obtain the MAC address of the target device from the ARP reply. When running in Server mode, TOE25G-IP waits until an ARP request is received to decode the MAC address and return an ARP reply. When running in Fixed-MAC mode, the user needs to know MAC address of the target device for setting to TOE25G-IP.*
   - *When running the test environment with an FPGA board and a PC, it is recommended to set the FPGA to run as Client mode.*
   - *When using two FPGA boards in a test environment, there are three options to establish the connection between them. The first option is to set one board as the Client and the other as the Server. The second option is to configure both boards in Fixed-MAC mode. The third option is to set one board to Fixed-MAC mode and the other board to act as the Client.*

3) The CPU displays the default values of the network parameters including Window update gap value, Reverse packet enable flag, the initialization mode, FPGA MAC address, FPGA IP address, FPGA port number, Target IP address, and Target port number. The firmware has two default parameter sets for the operation mode: Server parameter set (used for Server mode only) and Client parameter set (used for both Client and Fixed-MAC mode). When setting to Fixed-MAC mode, an extra parameter, Target MAC address, is also displayed. The user can select to complete the initialization process using the default parameters or by updating some parameters. The details of how to change the parameter are provided in Reset parameters menu (topic 3.2).

4) The CPU waits until the IP completes the initialization process by checking if busy status (TOE_CMD_INTREG[0]) is equal to 0b. After that, "IP initialization complete" is displayed with the main menu. There are five test operations in the main menu, and more details of each menu are described below.

3.1 Display parameters

This menu displays the current value of all TOE25G-IP parameters. The following steps are executed to display parameters.

1) Read the initialization mode.

2) Read all network parameters from each variable in the firmware following the initialization mode, i.e., Window update threshold, Reverse packet enable, source (FPGA) MAC address, source (FPGA) IP address, source (FPGA) port number, Target MAC address (only displayed in fixed MAC mode), Target IP address, and Target port number.
   Note: The source parameters are the FPGA parameters set to TOE25G-IP, while the Target parameters are the parameters of a PC or another FPGA.

3) Print out each variable.

3.2 Reset parameters

This menu is used to change some TOE25G-IP parameters, such as IP address and source port number. After setting the updated values to TOE25G-IP registers, the CPU resets to re-initialize the IP using new parameters. Finally, the CPU waits until the initialization is completed, monitored by busy flag. The following steps are executed to reset the parameters.

1) Display all parameters on the console, similar to topic 3.1 (Display parameters).

2) If the user uses the default value, skip to the next step. Otherwise, display the menu to set all parameters.
   i) Receive initialization mode from the user. If the initialization is changed, display the latest parameter set of new mode on the console.
   ii) Receive the remaining parameters from the user and validate each input separately. If an input is found to be invalid, that particular parameter will not be updated.

3) Force reset to PattGen and PattVer logic by setting USER_RST_INTREG[0]=1b.

4) Force reset to TOE25G-IP by setting TOE_RST_INTREG[0]=1b.

5) Set all parameters to TOE25G-IP registers, such as TOE_SML_INTREG and TOE_DIP_INTREG.

6) De-assert TOE25G-IP reset by setting TOE_RST_INTREG[0]=0b to initiate the initialization process of TOE25G-IP.

7) Monitor the TOE25G-IP busy flag (TOE_CMD_INTREG[0]) until the initialization process is completed (busy flag is de-asserted to 0b).

3.3 Send data test

This test process consists of three user inputs to set total transmit length, packet size, and connection mode (active open for Client connection mode or passive open for Server connection mode). If any of the inputs are invalid, the operation is cancelled. During the test, 32-bit incremental data is generated from the logic and sent to either PC or FPGA. The data is then verified by the test application on PC or by the verification module in FPGA. The operation is considered complete when all the 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, including transfer size (USER_TXLEN_INTREG), reset flag to clear initial value of the test pattern (USER_RST_INTREG[0]=1b), 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 the current system parameters.
4) Open connection following connection mode setting.
   i) For active open, CPU sets TOE_CMD_INTREG=2 (Open port). After that, it waits until the Connection interrupt status (USER_INT_INTREG[0]) is equal to 1b. If the busy flag of TOE25G-IP (TOE_CMD_INTREG[0]) is de-asserted to 0b but the interrupt is not asserted, the error message is displayed, and it returns to the main menu.
   ii) For passive open, CPU waits until the connection is opened by another device (PC or FPGA). Connection interrupt status (USER_INT_INTREG[0]) is monitored until it is equal to 1b.
5) Set the packet size to the TOE25G-IP register (TOE_PKL_INTREG) and calculate the total number of loops from the total transfer size. The maximum transfer size of each loop is 4 GB. The operation of each loop is as follows.
   i) Set the transfer size of this loop to TOE25G-IP register (TOE_TDL_INTREG). The transfer size is fixed to 4 GB except the last loop which is equal to the remaining size.
   ii) Set the send command to TOE25G-IP register (TOE_CMD_INTREG=0).
   iii) Wait until operation is completed by monitoring busy flag (TOE_CMD_INTREG[0]=0b). While monitoring the busy flag, the CPU reads the current amount of transmitted data from user logic (USER_TXLEN_INTREG) and displays the results on the console every second.
6) Set the close connection command to the 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 1b. Otherwise, the error message is displayed if TOE25G-IP busy flag (TOE_CMD_INTREG[0]) is de-asserted without the asserting of Connection interrupt status.
7) Calculate performance and show test result on the console.

3.4 Receive data test

The user specifies the total amount of data to be received, the data verification mode (enabled or disabled), and the connection mode (active open for Client connection mode or passive open for Server connection mode). If any of the inputs are invalid, the operation is cancelled. During the test, 32-bit incremental data is generated to verify the received data from either PC or FPGA when data verification mode is enabled. The sequence of this test is as follows.

1) Receive the total transfer size, data verification mode, and connection mode from the user input. Verify that all inputs are valid.
2) Set the UserReg registers.
    i) Reset flag to clear the initial value of test pattern (USER_RST_INTREG[0]=1b)
    ii) Data verification mode (USER_CMD_INTREG[1]=0b to only read the received data or 1b to read and verify the received data).
3) Display the recommended parameter (similar to Step 3 of Send data test).
4) Open the connection following the connection mode (similar to Step 4 of Send data test).
5) Wait until the connection is closed by the other device (PC or FPGA). ConnOn status (USER_CMD_INTREG[2]) is monitored until it is equal to 0b. While monitoring Connon, the CPU reads the current amount of received data from the user logic (USER_RXLEN_INTREG) and displays the results on the console every second.
6) Wait until all the data has been completely read by the user logic, as indicated by the FIFO status (USER_FFSTS_INTREG[15:4]=0).
   Note: USER_FFSTS_INTREG[15:4] is mapped from the TCPRxFfRdCnt signal of TOE25G-IP, which represents the number of data in the FIFO that can be read in 128-bit unit. Therefore, the amount of data that cannot yet be read from the FIFO, indicated by USER_FFSTS_INTREG[3:0] (or TCPRxFfLastRdCnt), should be neglected.
7) Compare the received length of user logic (USER_RXLEN_INTREG) with the set value from the user. If all data is completely received, the CPU checks verification result by reading USER_CMD_INTREG[1] (0b: normal, 1b: error). If some errors are detected, the error message is displayed.
8) Calculate performance and show the test result on the console.

3.5 Full duplex test

This menu enables full duplex testing by simultaneously transferring data between the FPGA and another device (PC/FPGA) in both directions using the same port number. The user provides four inputs: 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 with a PC, the transfer size set on the FPGA must match the size set on the test application (tcp_client_txrx_xg). The connection mode on the FPGA must be set to passive (server operation) when running with a PC.

The test runs in a forever loop until the user cancels the operation by entering any keys on the FPGA console and then entering Ctrl+C on the PC console. The test sequence is as follows.

1) Receive the 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 the test application that runs on the PC from the current system parameters.
3) Set UserReg registers, including the transfer size (USER_TXLEN_INTREG), reset flag to clear the initial value of the test pattern (USER_RST_INTREG[0]=1b), and command register to start the data pattern generator with data verification mode (USER_CMD_INTREG=1 or 3).
4) Open the connection following the connection mode (similar to Step 4 of Send data test).
5) Set the packet size to TOE25G-IP registers (TOE_PKL_INTREG=user input) and calculate total transfer size in each loop. The maximum size of each loop is 4 GB. The operation of each loop is as follows.
    i)   Set the transfer size of this loop to TOE_TDL_INTREG. The transfer size is fixed to maximum size (4GB) which is also aligned to the packet size, except the last loop. The transfer size of the last loop is equal to the remaining size.
    ii) Set the send command to the TOE25G-IP register (TOE_CMD_INTREG=0).
    iii) Wait until the send command is completed by monitoring the busy flag (TOE_CMD_INTREG[0] =0b). While monitoring the busy flag, the CPU reads the current amount of transmitted data and received data from the user logic (USER_TXLEN_INTREG and USER_RXLEN_INTREG) and displays the results on the console every second.
6) Close the connection following the connection mode value.
    a) For active close, the CPU waits until the total amount of received data is equal to the set value from the user. Then, set USER_CMD_INTREG=3 to close the connection. Next, the CPU waits until the Connection interrupt status (USER_INT_INTREG[0]) is asserted to 1b. Otherwise, an error message is displayed if the TOE25G-IP busy flag (TOE_CMD_INTREG[0]) is de-asserted without the asserting of the Connection interrupt status.
    b) For passive close, the CPU waits until the connection is closed by the other device (PC or FPGA). The Connection interrupt status (USER_INT_INTREG[0]) is monitored until it is equal to 1b.
7) Check the result and any error (similar to Step 6-7 of Receive data test).
8) Calculate the performance and show the test result on the console. Go back to step 3 to repeat the test in a 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	To initiate an active mode connection, set the TOE_CMD_INTREG register to either open or close the connection based on the port_ctl mode. Then, monitor the bit0 of USER_INT_INTREG register, which indicates the connection status interrupt, until it is asserted. Once the interrupt flag is set, it will be cleared.

void init_param(void)	
Parameters	None
Return value	None
Description	Reset parameter following the description in topic 3.2. In the function, show_param and input_param function are called to display parameters and get parameters from user.

int input_param(void)	
Parameters	None
Return value	0: Valid input, -1: Invalid input
Description	Receive network parameters from user, i.e., the initialization 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). Each input is validated separately. It will be updated when the input is valid. If it is invalid, that particular parameter will not be updated. After receiving all parameters, calling show_param function to display parameters.

Unsigned int read_conon(void)	
Parameters	None
Return value	0: Connection is OFF, 1: Connection is ON.
Description	Read the bit2 of USER_CMD_INTREG register to retrieve the connection status.

void show_cursize(void)	
Parameters	None
Return value	None
Description	To display the amount of data transmitted and received on the console, first read the USER_TXLEN_INTREG and USER_RXLEN_INTREG to obtain the current values. Then, convert the values to appropriate units, i.e., bytes, Kbytes, or Mbytes and display them on the console.

void show_param(void)	
Parameters	None
Return value	None
Description	Display the parameters following the description in topic 3.1.

void show_result(void)	
Parameters	None
Return value	None
Description	To display the total amount of data transmitted and received, first read USER_TXLEN_INTREG and USER_RXLEN_INTREG registers. Next, read the global parameters, timer_val and timer_upper_val, to calculate the total time usage in usec, msec, or sec units. Finally, calculate the transfer performance and display it in MB/s.

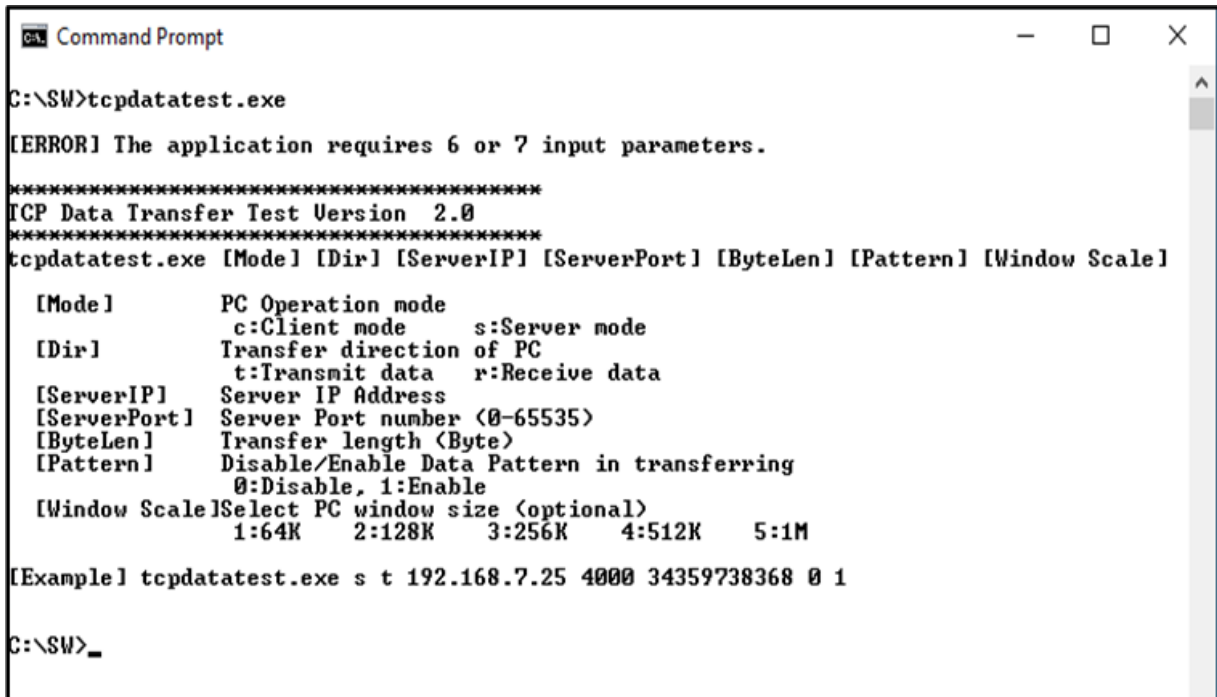| int toe_recv_test(void)	
Parameters	None
Return value	0: The operation is successful -1: Receive invalid input or error is found
Description	Run Receive data test following description in topic 3.4. It calls show_cursize and show_result function.

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. It calls show_cursize and show_result function.

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. It calls show_cursize and show_result function.

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

The "tcpdatatest" application is executed to send or receive TCP data on a PC. It requires six mandatory parameters and one optional parameter. It is important to ensure that the parameter inputs match those set on the FPGA. The details of each parameter are as follows.

Mandatory parameters
1) Mode   : c – The PC runs in Client mode and the FPGA runs in Server mode
                         s – The PC runs in Server mode and the FPGA runs in Client mode
2) Dir   : t – transmit mode (the PC sends data to the FPGA)
                         r – receive mode (the PC receives data from the FPGA)
3) ServerIP   : The IP address of the FPGA when the PC runs in Client mode
                         (Default is 192.168.25.42)
4) ServerPort   : The port number of the FPGA when the PC runs in Client mode
                         (Default is 60000)
5) ByteLen   : The total size of data to be transferred in bytes. This parameter is used only in transmit mode only and is ignored in receive mode. In transmit mode, the ByteLen value must match the total transfer size set in the receive data test menu of the FPGA. In receive mode, the application is closed when the connection is terminated.
6) Pattern   : 0 – Generate dummy data in transmit mode and disable data verification in receive mode.
                         1 – Generate incremental data in transmit mode and enable data verification in receive mode.

Optional parameter
1) Window Scale : Indicate the size of the allocated buffer for the TCP socket on PC.
It is also applied for TCP Window scaling feature. The valid range is 1-5.
1 – Allocated buffer size of 64 Kbyte
2 – Allocated buffer size of 128 Kbyte
3 – Allocated buffer size of 256 Kbyte
4 – Allocated buffer size of 512 Kbyte
5 – Allocated buffer size of 1 Mbyte

Note: The Window Scale parameter is an optional setting. If the user does not provide this parameter, it is automatically set to 1. The default TOE25G-IP does not have Window Scale features enabled. If you require the use of this feature, please contact our sales team for assistance.

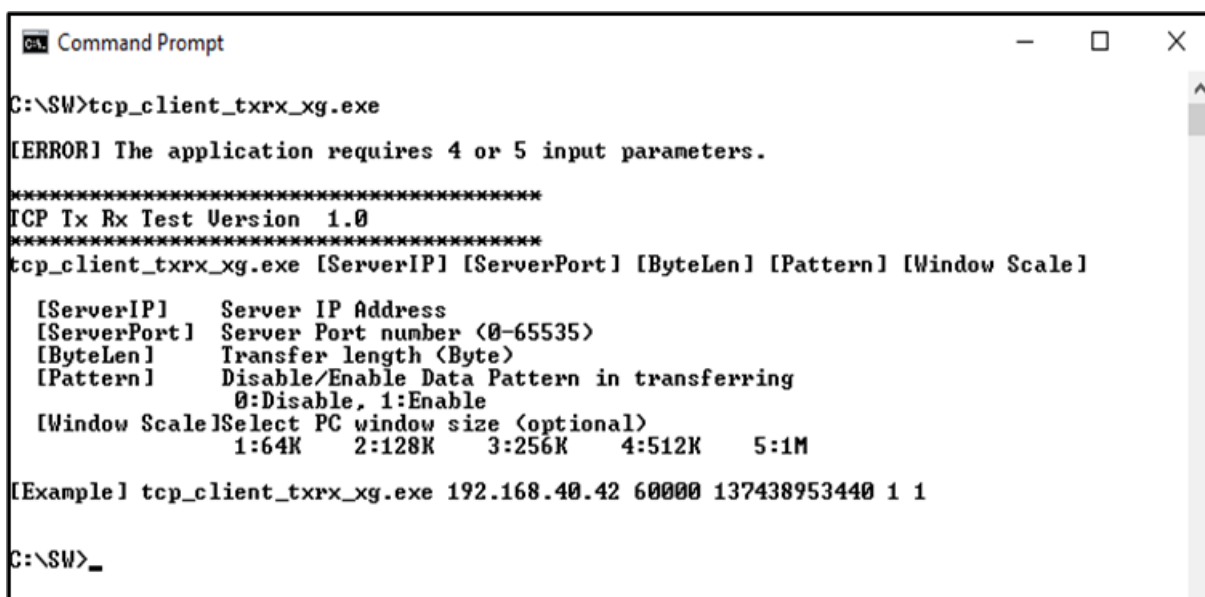The sequence of the test application when running in transmit mode and receive mode are described as follows.

Transmit data mode
1) Obtain and verify the user's input parameters, excluding the optional one.
2) Create a socket, set the socket options, and specify the socket memory size.
3) Establish a new connection using the server IP address and server port number.
4) Allocate 2 MB memory for the send buffer.
5) Generate the incremental test pattern to the send buffer if the dummy pattern is not selected.
6) Send the data out and read the total amount of sent data through the socket function.
7) Calculate the remaining transfer size.
8) Print the total transfer size every second.
9) Repeat step 5) – 8) until the remaining transfer size is 0.
10) Calculate the total performance and print the result on the console.
11) Close the socket and free the memory.

Receive data mode
1) Follow the step 1) – 3) of the Transmit data mode.
2) Allocate 2 MB memory for the receive buffer.
3) Wait for the data to be stored in the receive buffer and read it, and increase the total amount of received data.
4) Verify the received data by the incremental pattern if the data verification is enabled. Otherwise, skip this step. Print an error message if the data is incorrect.
5) Print the total amount of received data every second.
6) Repeat steps 3) – 5) until the connection is closed by the other device.
7) Calculate the total performance and print the result on the console.
8) Close the socket and free the memory.

4.2   "tcp_client_txrx_xg" for full duplex test

Figure 4-2 "tcp_client_txrx_xg" application usage

The "tcp_client_txrx_xg" application enables PC to send and receive TCP data through Ethernet using the same port number simultaneously. It runs only in Client mode and requires server parameters (network parameters of TOE25G-IP) to be input by the user. The application uses five parameters, described as follows.

Mandatory parameters
1) ServerIP   : The IP address of the FPGA
2) ServerPort   : The port number of FPGA
3) ByteLen   : The total transfer size in byte units, which is the total amount of
                     transmitted data and received data. This value must be equal to the
                     transfer size set on the FPGA for running full-duplex test.
4) Pattern   : 0 – Generate dummy data for the sending function and disable data
                     verification for the receiving function. This mode is used to check the
                     best performance of full-duplex transfer.
                     1 – Generate incremental data for the sending function and enable data
                     verification for the receiving function.

Optional parameter
1) Window Scale   : Indicate the size of the allocated buffer for the TCP socket on PC.
                     It is also applied for TCP Window scaling feature. The valid range is 1-5.
                     1 – Allocated buffer size of 64 Kbyte
                     2 – Allocated buffer size of 128 Kbyte
                     3 – Allocated buffer size of 256 Kbyte
                     4 – Allocated buffer size of 512 Kbyte
                     5 – Allocated buffer size of 1 Mbyte
Note: The Window Scale parameter is an optional setting. If the user does not provide this parameter, it is automatically set to 1. The default TOE25G-IP does not have Window Scale features enabled. If you require the use of this feature, please contact our sales team for assistance.

The sequence of the test application is as follows.

1) Obtain and verify the user's input parameters, excluding the optional one.
2) Allocate 2 MB memory for the send and receive buffers, separately.
3) Create the socket, set socket options, and set the socket memory size.
4) Create a new connection using the server IP address and server port number.
5) If the test pattern is enabled, generate the incremental test pattern in the send buffer; otherwise, skip this step for dummy data.
6) If the send function is not ready for operating, skip this step; otherwise, continue the following steps.
    i)   If the test pattern is enabled, generate the incremental test pattern in the send buffer; otherwise, skip this step for dummy data.
    ii) Send the data out and read the amount of sent data through socket function.
    iii) Calculate the remaining send size.
7) If the receive function is not ready for operating, skip this step; otherwise, continue the following steps.
    i)   Read the data from the receive buffer and increase the total amount of received data.
    ii) If the test pattern is enabled, verify the received data using the incremental pattern, and print an error message if verification fails; otherwise, skip this step.
8) Print the total amount of transmitted data and received data every second.
9) Repeat steps 5) – 8) until the total amount of transmitted and received data equals ByteLen, set by the user.
10) Calculate the performance and print the result on the console.
11) Close the socket.
12) Sleep for 1 second to wait until the hardware to complete the current test loop.
13) Start a new test by repeating steps 3) – 12) in forever loop. However, if the data verification fails, stop the application.

5 Revision History

Revision	Date	Description
1.4	23-May-23	1) Change full-duplex test application to "tcp_client_txrx_xg" 2) Add USER_INT_INTREG to register map and update firmware to use connection interrupt. 3) 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