



TOE100G-IP 4-Session with CPU reference design

1	Introduction	1
2	Hardware overview.....	3
2.1	100Gb Ethernet (MAC) Subsystem (100G BASE-SR).....	4
2.2	TxEMACMux4to1.....	10
2.3	TOE100G-IP	11
2.4	CPU and Peripherals	12
2.4.1	AsyncAxiReg.....	13
2.4.2	UserReg.....	15
3	CPU Firmware on FPGA	20
3.1	Display parameters.....	21
3.2	Reset IP	22
3.3	Half Duplex Test.....	23
3.4	Full duplex test.....	25
3.5	Function list in User application	27
4	Test Software on PC	30
4.1	“tcpdatatest” for half duplex test.....	30
4.2	“tcp_client_txrx_single” for full duplex test	32
5	Revision History.....	34

TOE100G-IP 4-Session with CPU reference design

Rev1.1 7-Jul-23

1 Introduction

The default TOE100G-IP reference design implements one TOE100G-IP for transferring TCP payload data by using one session. It is found that the performance is limited when the test environment is FPGA and PC, not FPGA and FPGA. The maximum performance when transferring with the test application run on PC is about 3000 – 4000 Mbyte/s.

When the data is transferred by using multiple sessions shared the same 100G Ethernet channel, the total performance between FPGA and PC is much increased. For example, when using four sessions by four TOE100G-IPs, the total performance of four sessions is about 8000 – 9000 Mbyte/s.

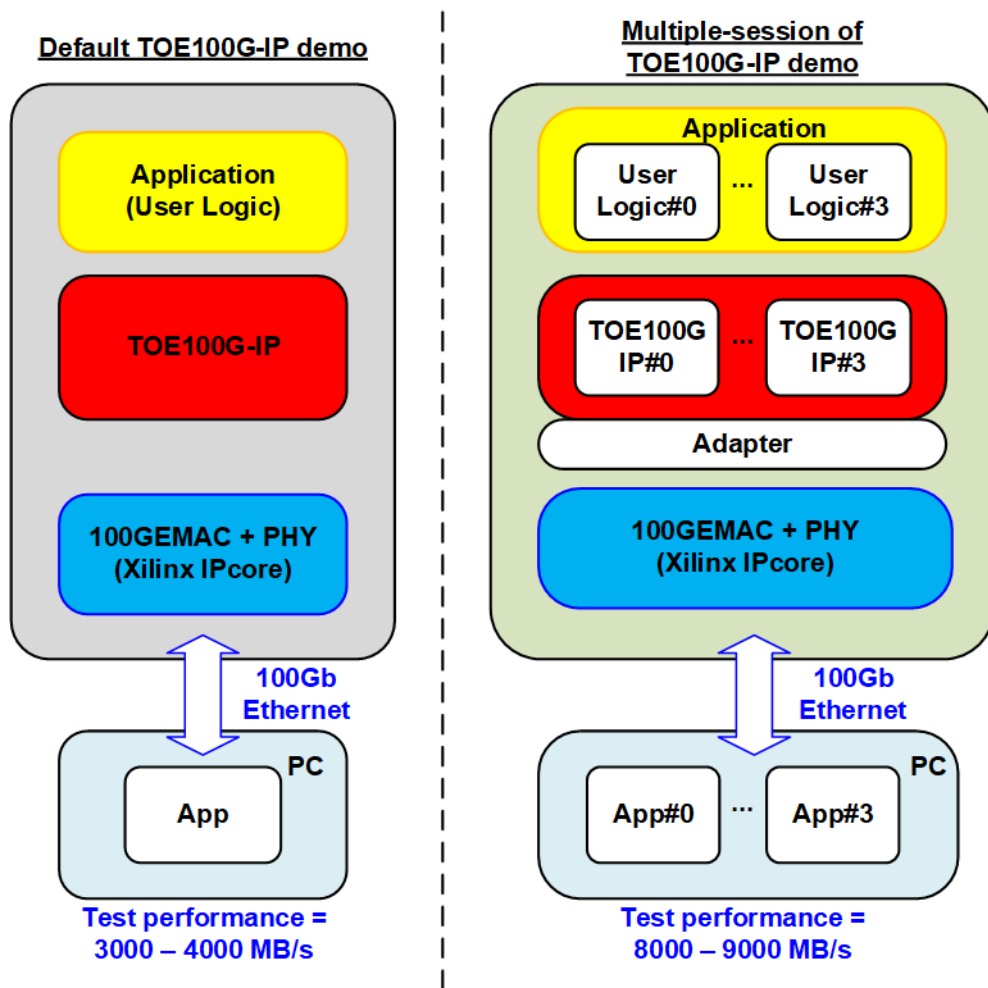


Figure 1-1 Multiple-session comparison with Uni-session

This document describes the reference design that includes four TOE100G-IPs that share the same 100GEMAC and PHY to support four sessions. The adapter is designed to interface four TOE100G-IPs and one 100GEMAC. User logic and Test application for running multiple-session demo are almost similar to one-session demo, except the test application of full-duplex demo. “tcp_client_txrx_single” is applied to run on PC instead. This application is run the test for one round, not forever loop like “tcp_client_txrx_40G”.

Though the multiple-session demo implements four sessions, the user can enable each TOE100G-IP independently to check the performance or the operation when using less than four sessions. Also, the transfer direction of each session can be configured individually. The user can modify the multiple-session reference design to increase or decrease the number of sessions to match the system requirement.

When the test environment is FPGA and FPGA, the performance of one session and four sessions is not much different. According to the test result of the default TOE100G-IP demo, using one session by FPGA and FPGA achieves the maximum performance of 100Gb Ethernet (about 12000 Mbyte/s). Therefore, using multiple sessions cannot improve the performance and may slightly reduce the performance due to the overhead time to switch the session. However, using multiple sessions may be applied when the system needs to transfer the data from many sources which are run individually.

2 Hardware overview

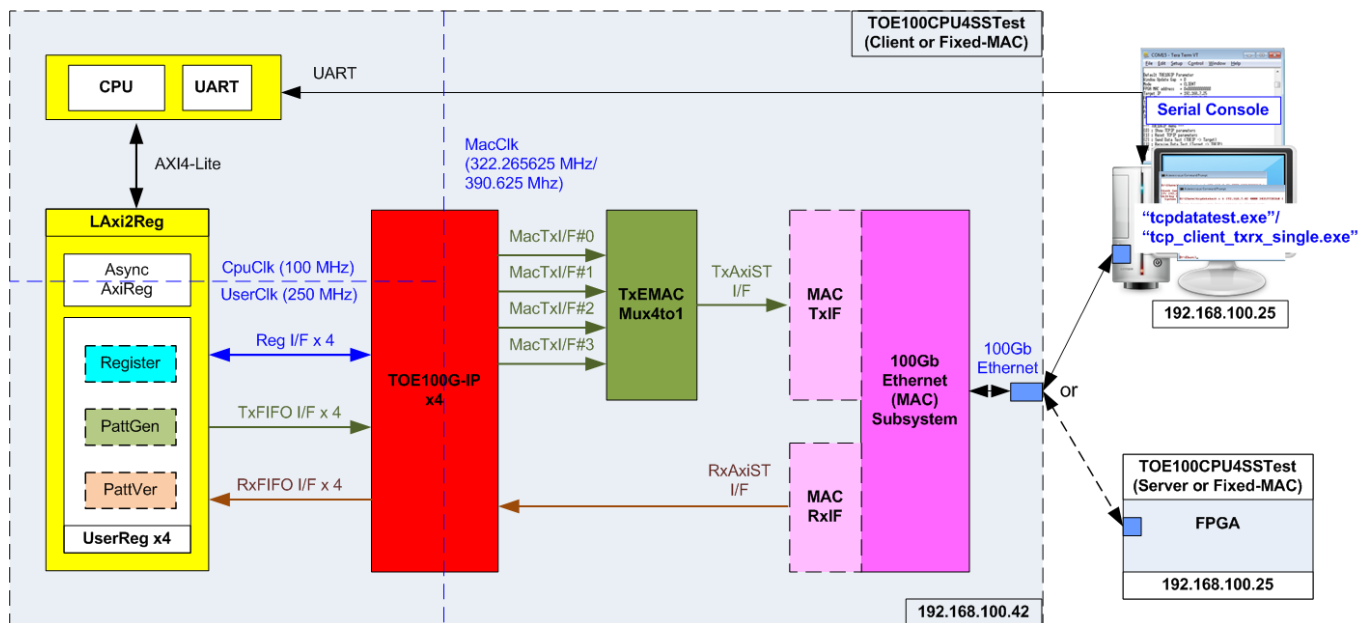


Figure 2-1 Demo block diagram

In test environment, two devices are used for 100Gb Ethernet transferring. When using FPGA and PC, FPGA is initialized by client mode and PC is initialized by server mode. On the other hand, when using two FPGAs, it may be initialized by Client <-> Server, Client <-> Fixed-MAC, or Fixed-MAC <-> Fixed-MAC, as shown in Figure 2-1. Two test applications can be called on PC for transferring the data, tcpdatatest or tcp_client_trrx_single.

In FPGA system, four TOE100G-IPs are included. Four MacTxI/F (MacTxI/F#0 - #3) which are output from TOE100G-IPs are connected to 100G Ethernet (MAC) Subsystem via TxEMACMux4to1 module. While Rx interface of 100G Ethernet (MAC) Subsystem is fed to four TOE100G-IPs directly. User interface of four TOE100G-IPs are mapped to LAXi2Reg which consists of AsyncAxiReg and four UserRegs, one UserReg connecting to one TOE100G-IP. There are three logic groups inside UserReg, i.e., 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.

For UltraScale+ device, 100G Ethernet Subsystem can connect with TOE100G-IP and TxEMACMux4to1 directly through 512-bit AXI4-Stream interface. For Versal device, 100G Ethernet MAC Subsystem uses 384-bit AXI4-Stream interface, so it needs to design MACTxIF and MacRxIF to convert data width for connecting with TOE100G-IP and TxEMAC4to1.

There are three clock domains in the design, i.e., CpuClk which is the clock for running the CPU system, MacClk which is the clock for user interface of 100Gb Ethernet (MAC) Subsystem, and UserClk which is the clock for running user logic of TOE100G-IP. In real system, the user can change the frequency of CpuClk and UserClk. According to TOE100G-IP datasheet, clock frequency of UserClk must be more than or equal to 220 MHz. AsyncAxiReg is designed to support asynchronous signals between CpuClk and UserClk. More details of each module are described as follows.

Note: When using 100G Ethernet Subsystem, MacClk is the output from 100G Ethernet subsystem and its frequency is equal to 322.266 MHz. While using 100G Ethernet MAC Subsystem, MacClk is generated by user logic and its frequency is equal to 390.625 MHz.

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

100G Ethernet (MAC) Subsystem implements the MAC layer and the low-layer protocol. To use 100G BASE-SR, physical connection of Ethernet cable is QSFP28 or 4xSFP28 connector. The IP core can be created by using IP wizard in Vivado tools.

For UltraScale+ device, 100G Ethernet Subsystem implements the MAC layer and PCS/PMA layer by integrating Transceiver module inside the IP. The user interface is 512-bit AXI4-stream at 322.266 MHz. More details of the core are described in the following link.

PG203: UltraScale+ Devices Integrated 100G Ethernet Subsystem Product Guide

https://www.xilinx.com/products/intellectual-property/cmac_usplus.html

For Versal device, 100G Ethernet MAC Subsystem implements the MAC layer and PCS logic without integrating Transceiver module. The user interface can be configured to several modes. In the reference design, Non-Segmented mode with independent clock is applied, so the user interface is 384-bit interface. The minimum clock frequency in this mode is 390.625 MHz. More details of the core are described in the following link.

PG314: Versal Devices Integrated 100G Multirate Ethernet MAC Subsystem Product Guide

<https://www.xilinx.com/products/intellectual-property/mrmac.html>

Note: 100Gb Ethernet (MAC) Subsystem is configured to not support RS-FEC feature.

The user interface of 100G Ethernet MAC Subsystem in Versal device is different from the EMAC interface of TOE100G-IP. Therefore, the adapter logic for both Tx and Rx interface must be designed, as shown Figure 2-2. More details of the adapter logic are described as follows.

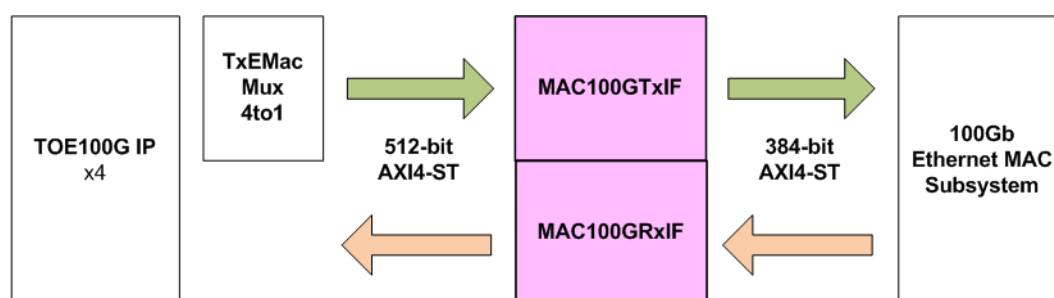


Figure 2-2 Adapter logic of EMAC interface in Versal Device

MAC100GTxIF

This module is AXI4-Stream converter from 512-bit to 384-bit to transfer data from user (TOE100G-IP) to 100G Ethernet MAC Subsystem. Therefore, it needs to have 384-bit register to store 128-bit user data that cannot be transmitted to EMAC for each cycle. rTempCnt is the control signal to show the amount of the unsent data in 128-bit unit which is stored in 384-bit internal register (rTempData). Four values are assigned to show the amount of data, i.e., 000b (No data), 001b (has one 128-bit data), 011b (has two 128-bit data), and 111b (has three 128-bit data or full). The format of data output to EMAC is mixed signal of user data (U2MACData) and 384-bit rTempData, controlled by rTempCnt. Timing diagram to show more details of MAC100GTxIF is shown in Figure 2-3.

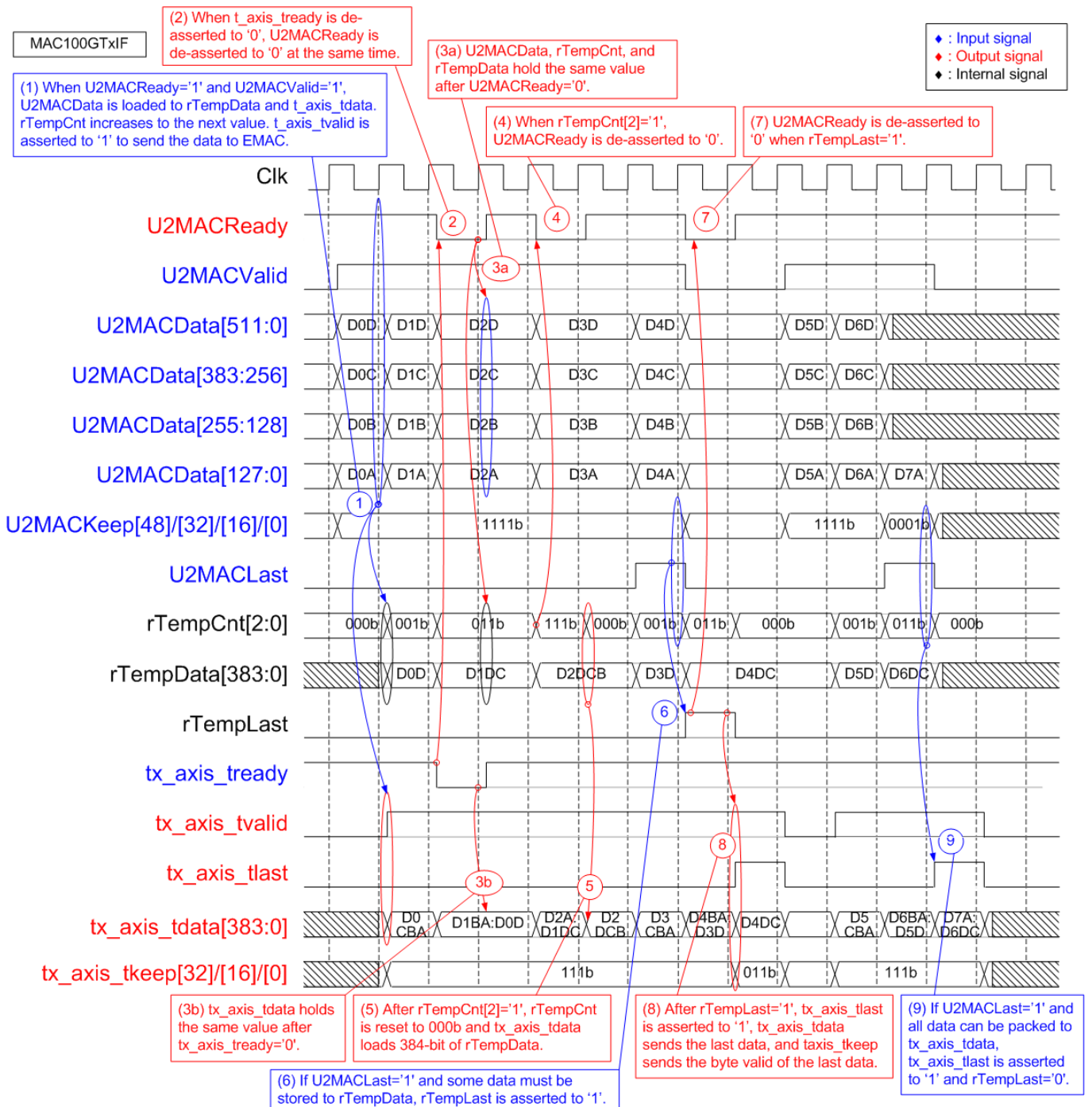


Figure 2-3 MAC100GTxIF Timing diagram

1. When the first data is received from user (U2MACValid='1' and U2MACReady='1' while rTempCnt=000b), 384-bit user data (U2MACData) is transferred to EMAC. tx_axis_tvalid is asserted to '1' and tx_axis_tdata loads 384-bit data from U2MACData. If this data is not the last data, the upper 128-bit unsent data is stored to internal register (rTempData) and rTempCnt increases the value by the sequence: 000b -> 001b -> 011b -> 111b. Also, tx_axis_tkeep are asserted to all one to transfer 384-bit data to EMAC.
2. When EMAC is not ready, tx_axis_tready is de-asserted to '0'. U2MACReady is de-asserted to '0' to pause user data transmission.
3. When tx_axis_tready and U2MACReady are de-asserted to '0', the output signals to EMAC (tx_axis_tvalid, tx_axis_tlast, tx_axis_tdata, and tx_axis_tkeep) and the input signals from user (U2MACValid, U2MACData, U2MACKeep, and U2MACLast) must hold the same value until the ready signals are re-asserted to '1' to accept the current data.
4. When 384-bit register (rTempData) stores three 128-bit data, rTempCnt is equal to 111b (full condition). At this time, U2MACReady is de-asserted to '0' to pause user data transmission. After that, 384-bit data of rTempData is flushed to EMAC.
5. "tx_axis_tdata" loads 384-bit data from rTempData and then rTempCnt is reset to 000b. There is no unsent data stored in rTempData.
6. The last user data is transmitted with asserting U2MACLast to '1'. U2MACKeep is read to check the number of valid bytes of the last data. Also, rTempCnt is read to check the amount of unsent data. In the example, one 128-bit data is stored in rTempCnt and 512-bit user data is received, so two 128-bit data must be stored to rTempCnt. In this case, rTempLast is asserted to '1' to store the unsent last data.
Note: Step 9) shows the example when the last user data is received but all data can be transferred to EMAC without storing any data in rTempData.
7. rTempLast is asserted to '1' when the last user data is stored in rTempData. At the same time, U2MACReady is de-asserted to '0' to pause user data transmission.
8. The last data from rTempData is transferred to EMAC. tx_axis_tlast is asserted to '1' and tx_axis_tkeep shows the amount of valid byte.
9. This step shows the example when there are two 128-bit data stored in rTempData and 128-bit last data is transmitted by user. Therefore, total data which has three 128-bit data can be transferred to tx_axis_tdata with asserting tx_axis_tlast to '1'. There is no data stored in rTempData and rTempLast is not asserted to '1'.

MAC100GRxIF

This module is AXI4-Stream converter from 384-bit to 512-bit to transfer data from 100G Ethernet MAC Subsystem to user (TOE100G-IP). The logic includes Latch register to store the unsent data that is not transferred to user. To support data realignment, three counters are designed to check the amount of data in 128-bit unit. First counter is `wRx128bDataCnt` which shows the amount of received data from EMAC which can be equal to 1, 2, or 3. Second counter is `rLatDataCnt` which shows the amount of unsent data that is received from EMAC. The unsent data is stored to the latch register (`rDataLat`). This counter can be equal to 0 – 3. Last counter is `wRxTotalDataCnt` which shows the sum of the amount of received data and the unsent data (`wRx128bDataCnt + rLatDataCnt`). Therefore, the value of the third counter can be equal to 1 – 6. When `wRxTotalDataCnt` is more than or equal to 4 (5 or 6), 512-bit data will be packed and transmitted to user. However, the last data transmitted to the user can be less than 512-bit data by controlling byte enable value (`MAC2UKeep`).

The second counter (`rLatDataCnt`) is updated by several conditions.

- (1) When the first data is received and there is no unsent data stored in `rDataLat` (`rLatDataCnt=0`), all bits of the first data is loaded to `rDataLat`. Therefore, `rLatDataCnt` must be equal to the amount of received data from EMAC (`wRx128bDataCnt` or `wRxTotalDataCnt` which is the same value when `rLatDataCnt=0`).
- (2) When total amount of data (`wRxTotalDataCnt`) is more than or equal to 4, one 512-bit data will be transmitted to TOE100G-IP. Therefore, the amount of unsent data (`rLatDataCnt`) is decreased by 4 (`wRxTotalDataCnt – 4`).
- (3) When the last data is transmitted and there is no new packet is received, Latch register now is empty status. Therefore, `rLatDataCnt` is reset to 0.
- (4) There is a special case that the last data is transmitted while the first data of the new packet is received. This condition is combination of (1) and (3). Therefore, the amount of unsent data is equal to the amount of received data in the new packet (`wRx128bDataCnt`).

384-bit latch register (`rDataLat`) uses `rLatDataCnt` to determine the maximum number of received data from EMAC that must be kept in the next cycle.

- (1) When `rLatDataCnt = 0`, three 128-bit new data (`rx_axis_tdata[384:0]`) must be kept.
- (2) When `rLatDataCnt = 3`, one 128-bit new data can be packed with three 128-bit previous data (`rDataLat[383:0]`). Therefore, two 128-bit new data (`rx_axis_tdata[384:128]`) must be kept to `rDataLat`.
- (3) When `rLatDataCnt = 2`, two 128-bit new data can be packed with two 128-bit previous data (`rDataLat[255:0]`). Therefore, one 128-bit data (`rx_axis_tdata[384:256]`) must be kept to `rDataLat`.
- (4) When `rLatDataCnt = 1`, all new data can be packed with one 128-bit previous data (`rDataLat[127:0]`). There is no data kept to `rDataLat`.

To transfer the last data from EMAC to the user, it has two behaviors.

- (1) If all last data from EMAC can be packed with `rDataLat` (`wRxTotalDataCnt ≤ 4`), the last data will be transferred to the user in the next cycle.
- (2) When `wRxTotalDataCnt` of the last data is more than 4, it needs two cycles to send all data – 512-bit data for the 1st cycle and the remained data for the 2nd cycle. To support this feature, `rExLast` is designed to latch the last flag of EMAC for sending the last data in the 2nd cycle.

Timing diagram to show MAC100GRxIF operation is shown in Figure 2-4.

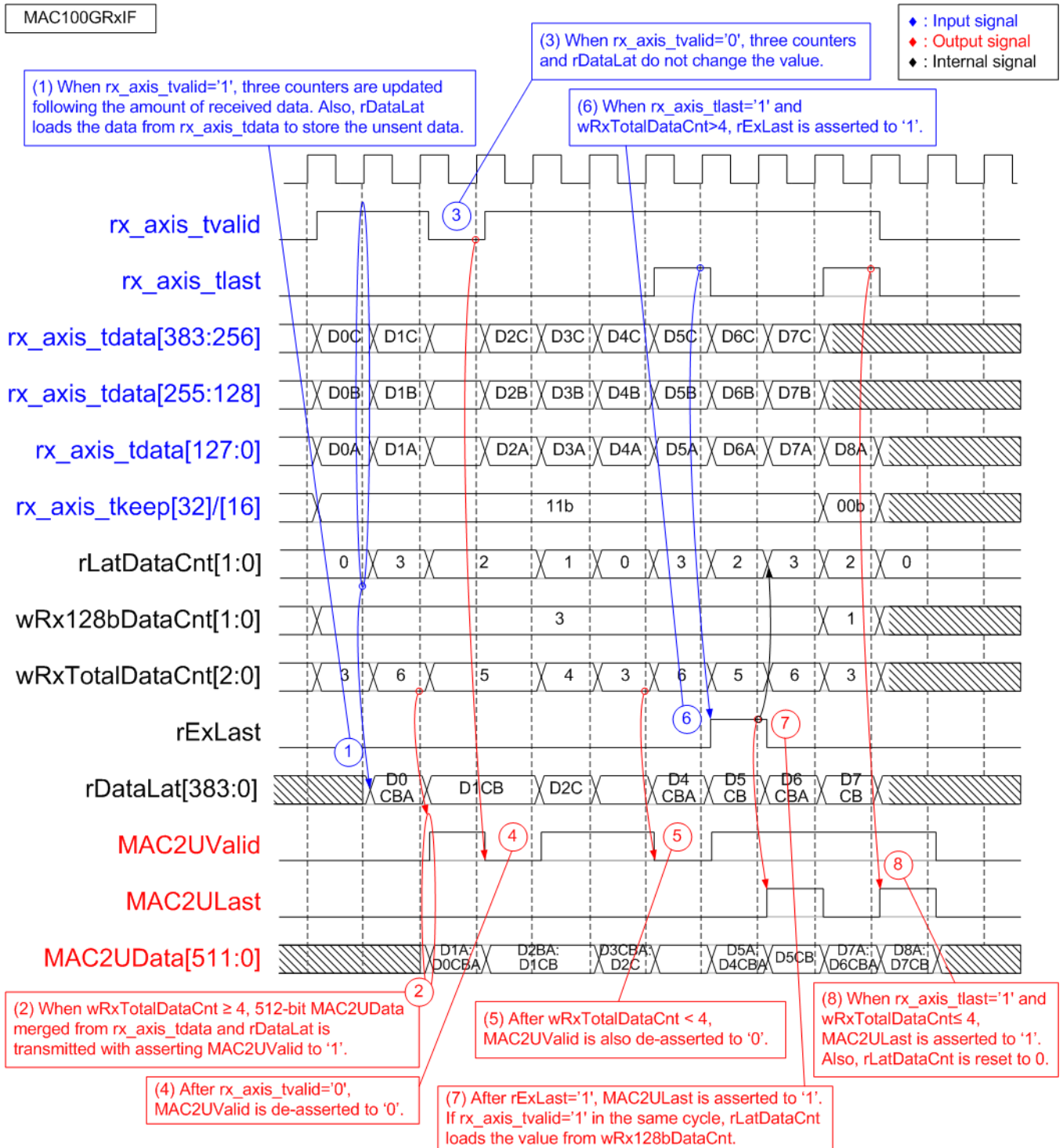


Figure 2-4 MAC100GRxIF Timing diagram

1. When the new 384-bit data is received from EMAC, wRx128bDataCnt is equal to 3. If the previous clock cycle is Idle, rLatDataCnt is equal to 0. Therefore, wRxTotalDataCnt is equal to 3 (0+3). When rLatDataCnt=0, all 384-bit data is loaded to rDataLat. As wRxTotalDataCnt is less than 4, there is no data transmitted to MAC2U I/F.
2. Next, 384-bit data are received from EMAC while rLatDataCnt is equal to 3 (the amount of data that is stored to rDataLat in the previous clock cycle). Therefore, wRxTotalDataCnt is equal to 6 (3 + 3) which is enough to transmit the data to MAC2U I/F. MAC2UValid is asserted to '1' to send 512-bit M2UData and M2UData loads three 128-bit data (D0A, D0B, and D0C) from rDataLat and one 128-bit data from rx_axis_tdata (D1A). Therefore, two 128-bit data (D1B and D1C) are unsent and stored to rDataLat. rLatDataCnt is updated to 2.
3. EMAC de-asserts rx_axis_tvalid to '0' when it is not ready to transmit the new data. Three counters (rLatDataCnt, wRx128bDataCnt, and wRxTotalDataCnt) and rDataLat hold the same value to wait more data from EMAC.
4. If EMAC pauses data transmission by de-asserting rx_axis_tvalid to '0', MAC2UValid is de-asserted to '0' in the next clock.
5. At the first cycle of every four cycle to receive 384-bit data from EMAC, rLatDataCnt is equal to 0 and wRxTotalDataCnt is less than 4. Therefore, MAC2UValid is de-asserted to '0' to pause data transmission to the user.
6. When the last data is received from EMAC (rx_axis_tlast='1' and rx_axis_tvalid='1') and wRxTotalDataCnt in that cycle is more than 4 (5 or 6), the latch flag to store last signal (rExLast) is asserted to '1'. At the same time, 512-bit data is transferred to the user while the remained last data is transferred in the next cycle.
7. After rExLast is asserted to '1', MAC2ULast is asserted to '1' to send the remained last data which is stored in rDataLat. If the first data of the new packet is transferred from EMAC immediately, the first data is loaded to rDataLat and rLatDataCnt is equal to the amount of 128-bit data in the first cycle.
8. The example to send the last data without asserting rExLast is shown in this step. When rx_axis_tlast is asserted to '1' and wRxTotalDataCnt is less than or equal to 4, the last data can be packed and transferred to the user in the next cycle. Therefore, rExLast is not asserted to '1' and rLatDataCnt is reset to 0.

2.2 TxEMACMux4to1

The system consists of four TOE100G-IPs which share the same EMAC. Therefore, TxEMACMux4to1 is designed to transfer the transmitted packet from four TOE100G-IPs to one EMAC. The core signal inside this module is rChSel that is applied to select one active TOE100G-IP module from four modules. Timing diagram of this module is displayed in Figure 2-5.

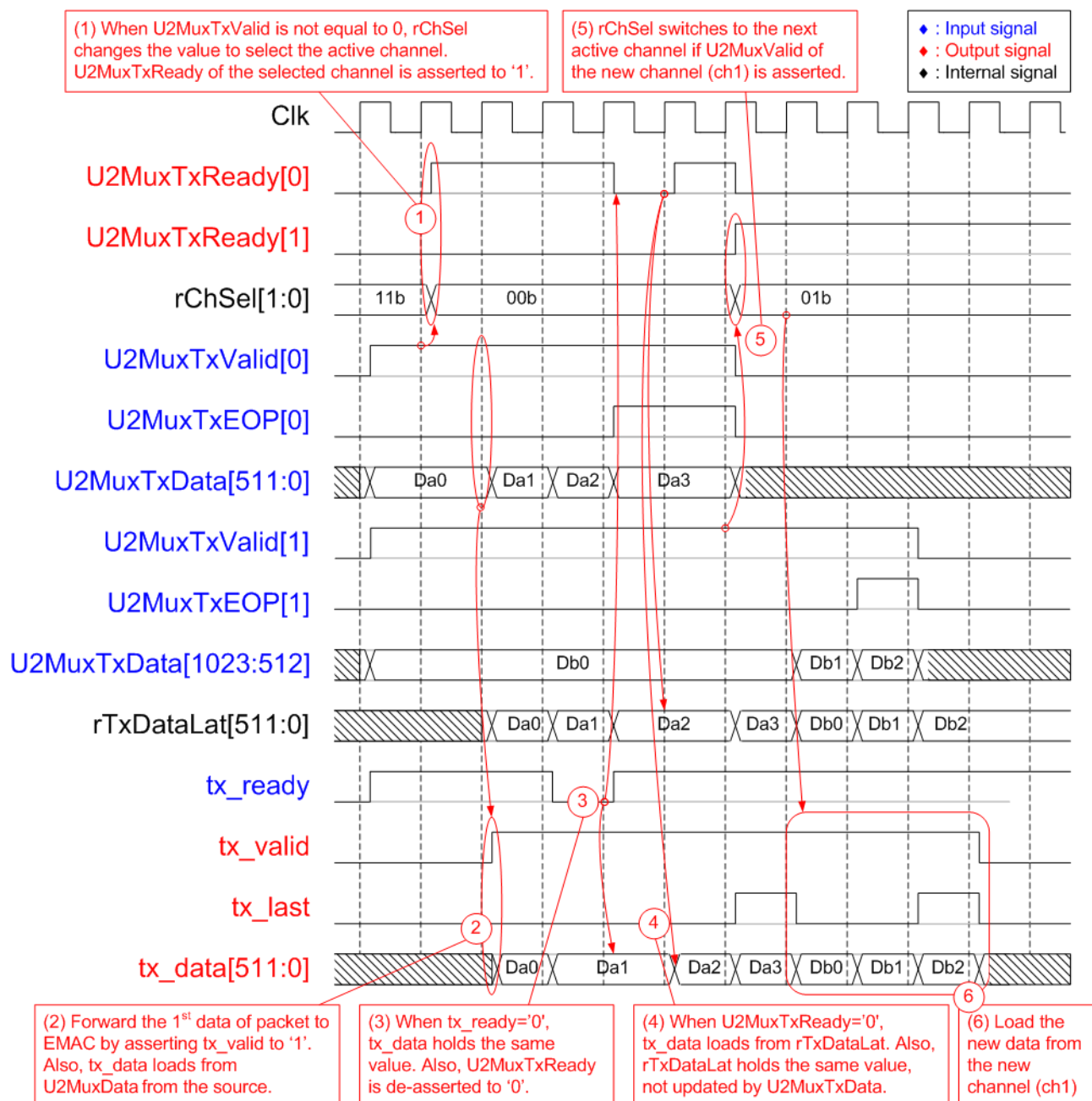


Figure 2-5 TxEMACMux4to1 timing diagram

1. If current channel (Ch#3) does not transfer the data and the new channel requests to transfer data by asserting U2MuxTxValid to '1', rChSel (the signal to indicate the active channel) will switch the value to the new channel. In Figure 2-5, there are two channels that send the request at the same time. The selected channel will be the nearest channel of the order 0 -> 1 -> 2 -> 3 -> 0. Therefore, Ch#0 is higher priority than Ch#1 when the current channel is 3. U2MuxTxReady of the selected channel (Ch#0) is asserted to '1' to accept the first data.
2. The input signals of the selected channel (Ch#0), i.e., U2MuxTxEOP (end-of-packet) and U2MuxTxData (512-bit data) are loaded to the output signals of EMAC (tx_last and tx_data respectively). Also, tx_valid is asserted to '1' to start sending the new packet to EMAC.
3. When EMAC is not ready to receive data by de-asserting tx_ready to '0', all output signals of EMAC hold the same value. Also, U2MuxTxReady of the active channel is de-asserted to '0' to hold the input signals from the source.
4. After EMAC re-asserts tx_ready to accept the data, the next output signals to EMAC will be loaded from the internal latch register (rTxDataLat). The internal latch register is designed to load the data from the active source when U2MuxTxReady is asserted to '1'. Therefore, the latch register is the temporal buffer for storing the unsent data to EMAC when EMAC pauses data transmission.
5. After accepting the final data of a packet from the active channel, the next active channel is scanned. If U2MuxTxValid of remaining channels is asserted, rChSel will update the value following the rule (0 -> 1 -> 2 -> 3 -> 0). In Figure 2-5, the next active channel is Ch#1, so rChSel is set to 01b to accept the data from Ch#1.
6. The input signals (U2MuxTxEOP and U2MuxTxData) of the active channel (Ch#1) are forwarded to be the output signals of EMAC (tx_last and tx_data) until transferring the final data of a packet.

2.3 TOE100G-IP

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

More details are described in datasheet.

https://dqway.com/products/IP/TOE100G-IP/dg_toe100gip_data_sheet_xilinx.pdf

2.4 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 2-6.

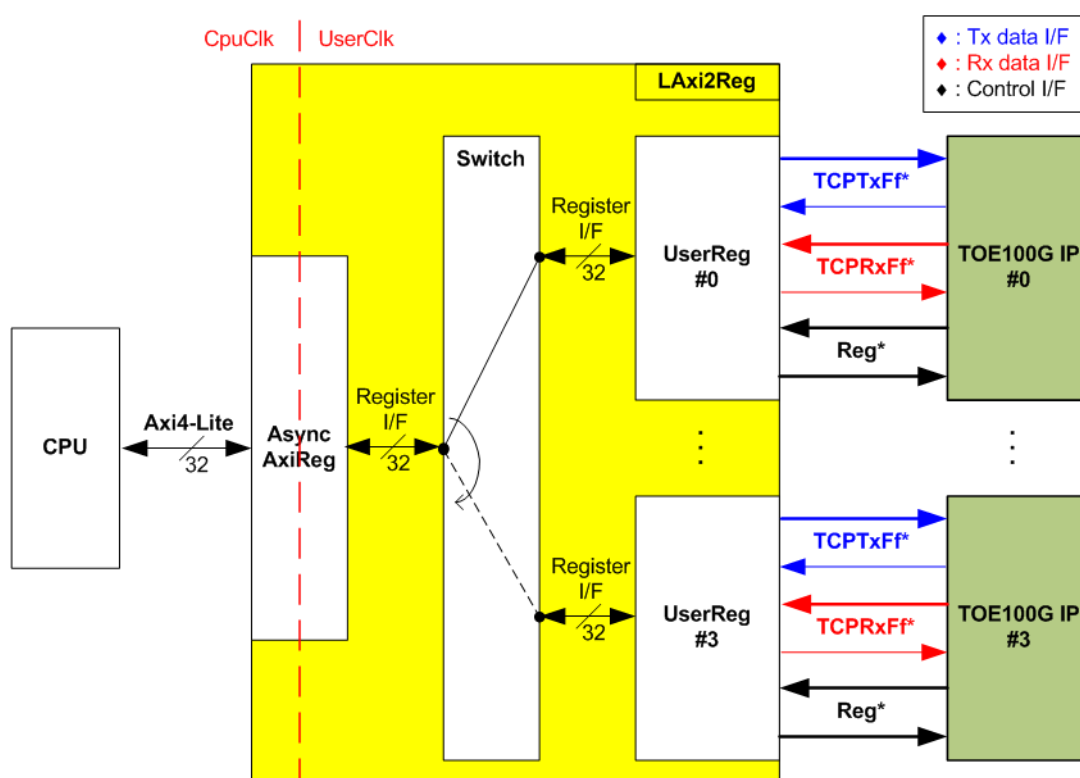


Figure 2-6 LAXi2Reg block diagram

Similar to the default TOE100G-IP reference design, LAXi2Reg consists of AsyncAxiReg and UserReg. To support multiple sessions, four UserRegs are integrated. Therefore, switch logic is designed to decode the address requested by AsyncAxiReg. Two upper bits are applied for selecting the active channel. When two upper bits of the requested address are equal to 00b, 01b, 10b, or 11b, switch logic connects Register I/F to UserReg#0, #1, #2, or #3, respectively. Switch logic includes D Flip-Flop for processing, so latency time for write and read access is increased, comparing to the default TOE100G-IP.

AsyncAxiReg and UserReg are the same module applied in the default TOE100G-IP reference design. AsyncAxiReg converts the AXI4-Lite signals to be the simple register and includes asynchronous logic to support clock domain crossing between CpuClk domain and UserClk domain. UserReg is the example of user logic to interface with TOE100G-IP. More details of AsyncAxiReg and UserReg are described as follows.

2.4.1 AsyncAxiReg

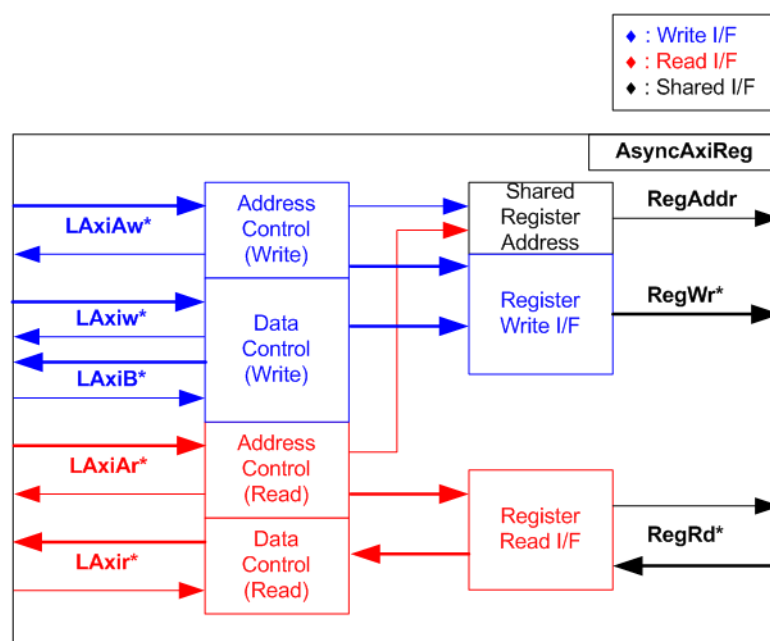


Figure 2-7 AsyncAxiReg interface

The signal on AXI4-Lite bus interface can be split into five groups, i.e., LAXiAw* (Write address channel), LAXiw* (Write data channel), LAXiB* (Write response channel), LAXiAr* (Read address channel), and LAXir* (Read data channel). More details to build custom logic for AXI4-Lite bus is described in following document.

https://forums.xilinx.com/xlnx/attachments/xlnx/NewUser/34911/1/designing_a_custom_axi_slave_rev1.pdf

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 2-7. 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 address from LAXiAw for write access or LAXiAr for read access.

The simple register interface is compatible with single-port RAM interface for write transaction. The read transaction of the register interface is slightly modified from RAM interface by adding RdReq and RdValid signals for controlling read latency time. The address of register interface is shared for write and read transaction, so user cannot write and read the register at the same time. The timing diagram of the register interface is shown in Figure 2-8.

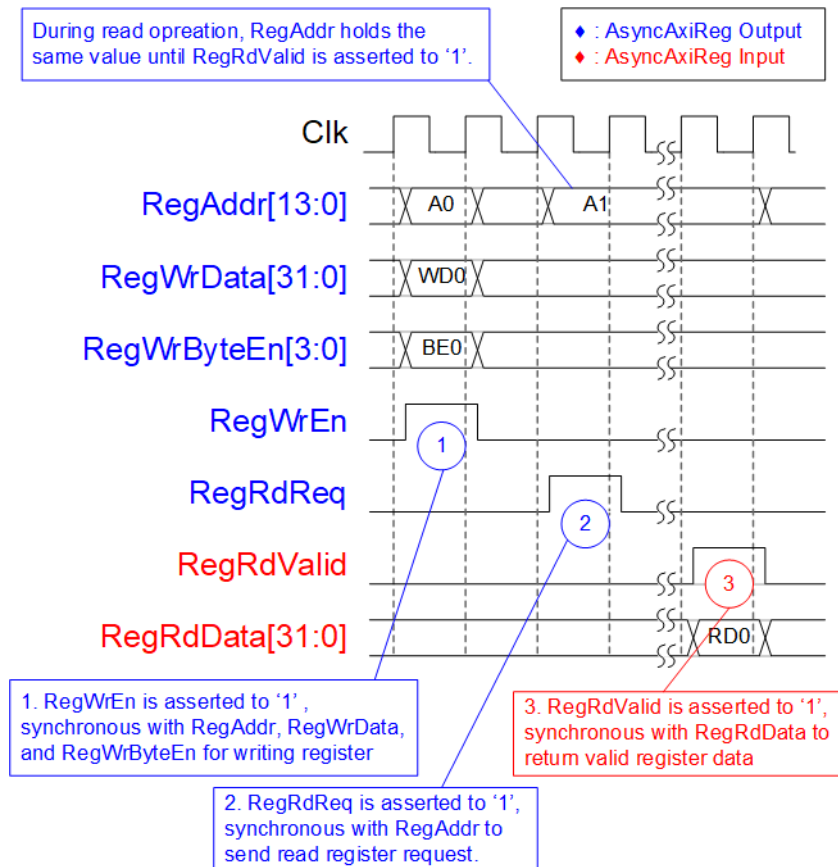


Figure 2-8 Register interface timing diagram

- 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.

2.4.2 UserReg

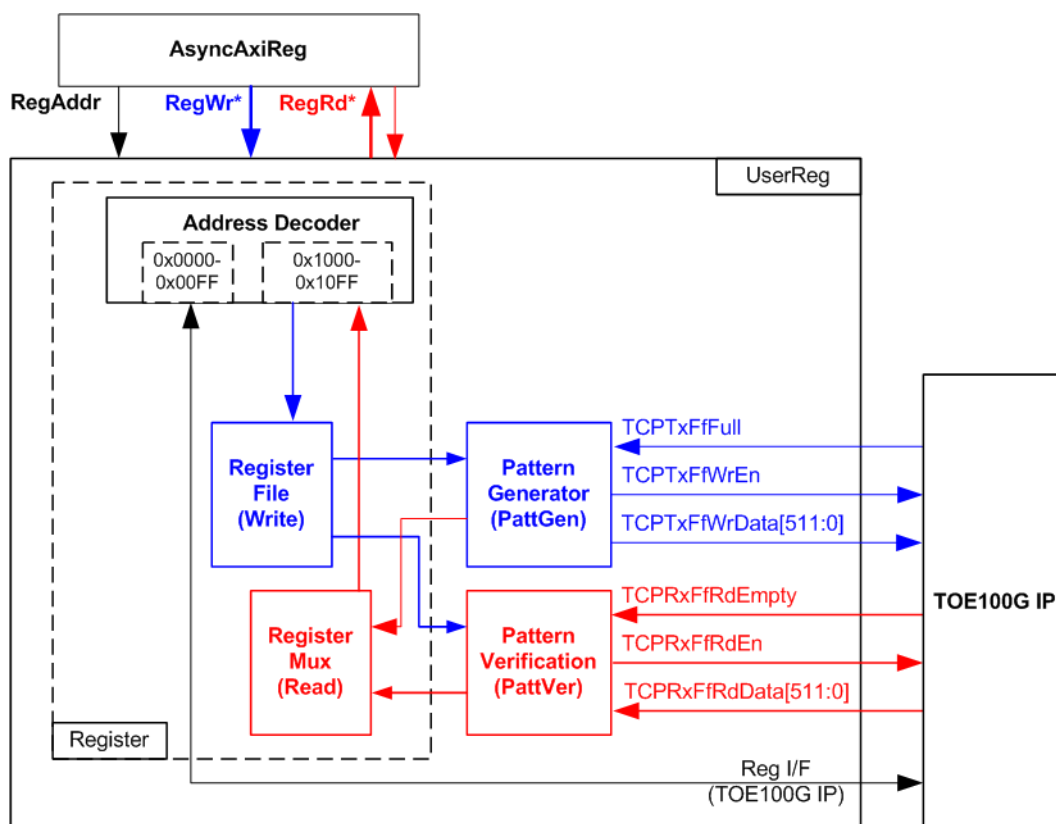


Figure 2-9 UserReg block diagram

The logic inside UserReg has three operations, i.e., Register, Pattern generator (PattGen), and Pattern verification (PattVer). Register block decodes the address requested from AsyncAxiReg and then selects the active register for write or read transaction. Pattern generator block is designed to send 512-bit test data to TOE100G-IP following FIFO interface standard. Pattern verification block is designed to read and verify 512-bit data from TOE100G-IP following FIFO interface standard. More details of each block are described as follows.

Register Block

Each UserReg uses the address area from 0x0000 – 0x1FFF. To support four sessions, four UserReg modules are mapped to 0x0000 – 0x7FFF by following assignment.

- 0x0000 – 0x1FFF: TOE100G-IP#0 and UserReg#0 (Session#0)
- 0x2000 – 0x3FFF: TOE100G-IP#1 and UserReg#1 (Session#1)
- 0x4000 – 0x5FFF: TOE100G-IP#2 and UserReg#2 (Session#2)
- 0x6000 – 0x7FFF: TOE100G-IP#3 and UserReg#3 (Session#3)

The address is split into two areas in each session, i.e., TOE100G-IP register (0x0000-0x00FF) and UserReg register (0x1000-0x10FF). Therefore, the upper bit of RegAddr is applied to select the active area between TOE100G-IP or UserReg register. While the lower bits of RegAddr are fed to TOE100G-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, one multiplexer is included within UserReg to select the read data. Totally, the latency time to read data is equal to one clock cycle, so RegRdValid is created by RegRdReq with asserting one D Flip-flop. However, there is switch logic inside LAXI2Reg which increases latency time of read access about two clock cycles. More details of the address mapping within UserReg module are shown in Table 2-1

Table 2-1 Register map Definition

Address	Register Name	Description
Wr/Rd	(Label in the "toe100g4sstest.c")	
BA+0x0000 – BA+0x00FF: TOE100G-IP#0 Register Area		
More details of each register are described in TOE100G-IP datasheet.		
BA+0x0000	TOE_RST_INTREG	Mapped to RST register within TOE100G-IP#0
BA+0x0004	TOE_CMD_INTREG	Mapped to CMD register within TOE100G-IP#0
BA+0x0008	TOE_SML_INTREG	Mapped to SML register within TOE100G-IP#0
BA+0x000C	TOE_SMH_INTREG	Mapped to SMH register within TOE100G-IP#0
BA+0x0010	TOE_DIP_INTREG	Mapped to DIP register within TOE100G-IP#0
BA+0x0014	TOE_SIP_INTREG	Mapped to SIP register within TOE100G-IP#0
BA+0x0018	TOE_DPN_INTREG	Mapped to DPN register within TOE100G-IP#0
BA+0x001C	TOE_SPN_INTREG	Mapped to SPN register within TOE100G-IP#0
BA+0x0020	TOE_TDL_INTREG	Mapped to TDL register within TOE100G-IP#0
BA+0x0024	TOE_TMO_INTREG	Mapped to TMO register within TOE100G-IP#0
BA+0x0028	TOE_PKL_INTREG	Mapped to PKL register within TOE100G-IP#0
BA+0x002C	TOE_PSH_INTREG	Mapped to PSH register within TOE100G-IP#0
BA+0x0030	TOE_WIN_INTREG	Mapped to WIN register within TOE100G-IP#0
BA+0x0034	TOE_ETL_INTREG	Mapped to ETL register within TOE100G-IP#0
BA+0x0038	TOE_SRV_INTREG	Mapped to SRV register within TOE100G-IP#0
BA+0x003C	TOE_VER_INTREG	Mapped to VER register within TOE100G-IP#0
BA+0x0040	TOE_DML_INTREG	Mapped to DML register within TOE100G-IP#0
BA+0x0044	TOE_DMH_INTREG	Mapped to DMH register within TOE100G-IP#0
BA+0x1000 – BA+0x10FF: UserReg#0 Control/Status		
BA+0x1000	Total transmit length	Wr [31:0] – Total amount of transmitted data of UserReg#0 in 512-bit unit. Valid from 1-0xFFFFFFFF.
Wr/Rd	USER_TXLEN_INTREG	Rd [31:0] – Current amount of transmitted data of UserReg#0 in 512-bit unit. The value is cleared to 0 when USER_CMD_INTREG is written by user.
BA+0x1004	User Command	Wr
Wr/Rd	USER_CMD_INTREG	[0] – Start transmitting. Set '0' to start transmitting data from UserReg#0. [1] – Data verification enable ('0': Disable data verification, '1': Enable data verification) Rd [0] – Busy of PattGen inside UserReg#0 ('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 TOE100G-IP#0
BA+0x1008	User Reset	Wr
Wr/Rd	USER_RST_INTREG	[0] – Reset signal. Set '1' to reset UserReg#0. This bit is auto-cleared to '0'. [8] – Set '1' to clear TimerInt latched value of TOE100G-IP#0 Rd [8] – Latched value of TimerInt output from TOE100G-IP#0 ('0': Normal, '1': TimerInt is asserted) This flag is cleared by system reset condition or setting USER_RST_INTREG[8]='1'. [16] – Ethernet linkup status from Ethernet MAC ('0': Not linkup, '1': Linkup)

Address	Register Name	Description
Wr/Rd	(Label in the "toe100g4sstest.c")	
BA+0x1000 – BA+0x10FF: UserReg#0 Control/Status		
BA+0x100C Rd	FIFO status USER_FFSTS_INTREG	Rd[5:0] - Mapped to TCPRxFfLastRdCnt signal of TOE100G-IP#0 [15:6] - Mapped to TCPRxFfRdCnt signal of TOE100G-IP#0 [24] - Mapped to TCPTxFfFull signal of TOE100G-IP#0
BA+0x1010 Rd	Total receive length USER_RXLEN_INTREG	Rd[31:0] – Current amount of received data from TOE100G-IP#0 in 512-bit unit. The value is cleared to 0 when USER_CMD_INTREG is written by user.
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 TOE100G-IP, '0': ConnOn does not change the value.) <i>Note: ConnOn value can be read from USER_CMD_INTREG[2].</i>
BA+0x1080 Rd	EMAC IP version EMAC_VER_INTREG	Rd[31:0] – Mapped to IPVersion output from DG EMAC-IP when the system integrates DG EMAC-IP. In this demo, it is equal to 0.
BA+0x2000 – BA+0x20FF: TOE100G-IP#1 Register Area BA+0x3000 – BA+0x30FF: UserReg#1 Control/Status BA+0x4000 – BA+0x40FF: TOE100G-IP#2 Register Area BA+0x5000 – BA+0x50FF: UserReg#2 Control/Status BA+0x6000 – BA+0x60FF: TOE100G-IP#3 Register Area BA+0x7000 – BA+0x70FF: UserReg#3 Control/Status		
BA+0x2000-BA+0x20FF	TOE100G-IP#1 register area, similar to BA+0x0000 – BA+0x00FF	
BA+0x3000-BA+0x30FF	UserReg#1 control/status, similar to BA+0x1000 – BA+0x10FF	
BA+0x4000-BA+0x40FF	TOE100G-IP#2 register area, similar to BA+0x0000 – BA+0x00FF	
BA+0x5000-BA+0x50FF	UserReg#2 control/status, similar to BA+0x1000 – BA+0x10FF	
BA+0x6000-BA+0x60FF	TOE100G-IP#3 register area, similar to BA+0x0000 – BA+0x00FF	
BA+0x7000-BA+0x70FF	UserReg#3 control/status, similar to BA+0x1000 – BA+0x10FF	

Pattern Generator

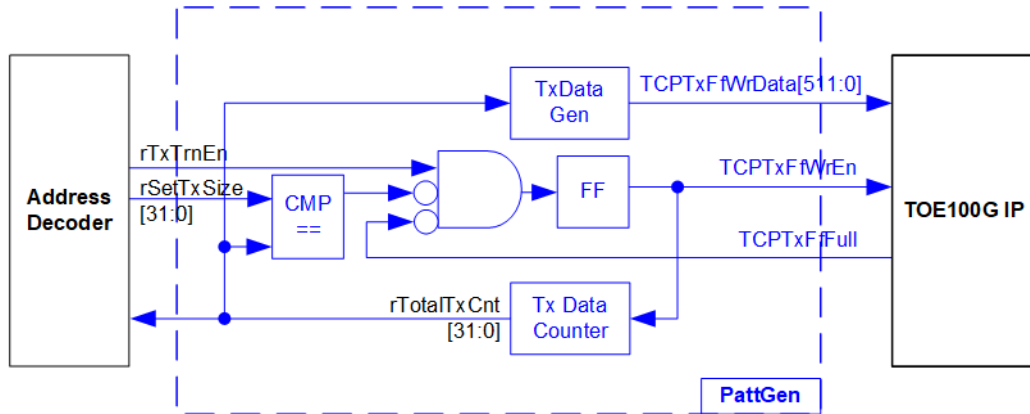


Figure 2-10 PattGen block

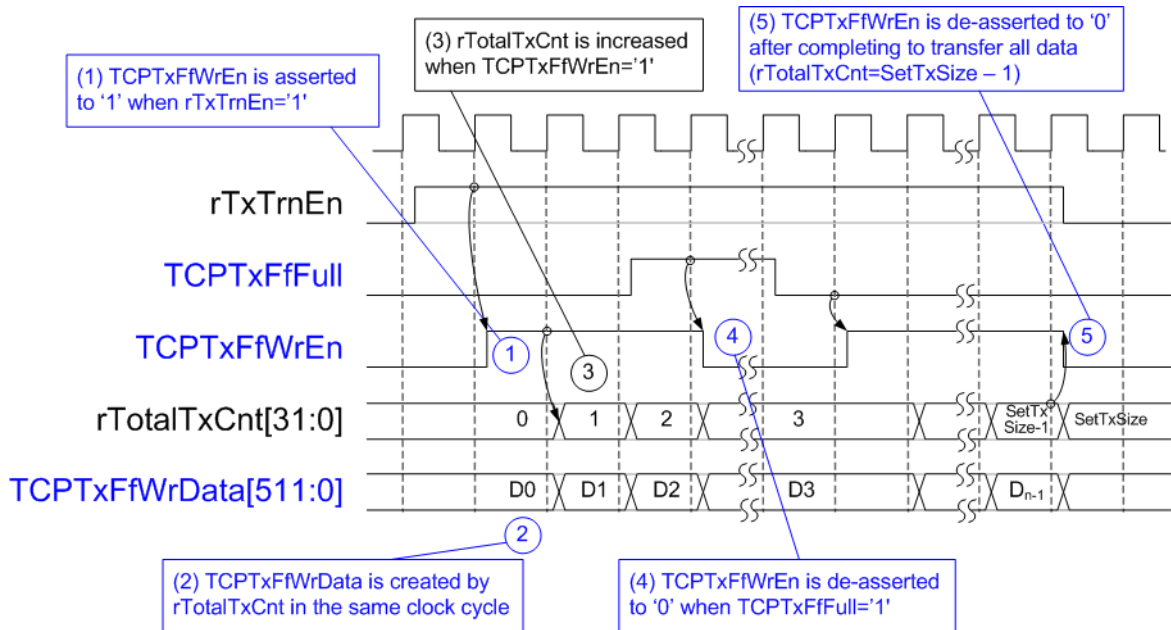


Figure 2-11 PattGen timing diagram

Figure 2-10 shows the details of PattGen which generates test data to TOE100G-IP. Timing diagram to show the relation of each logic is displayed in Figure 2-11.

To start PattGen operation, the user sets USER_CMD_REG[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 TOE100G-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

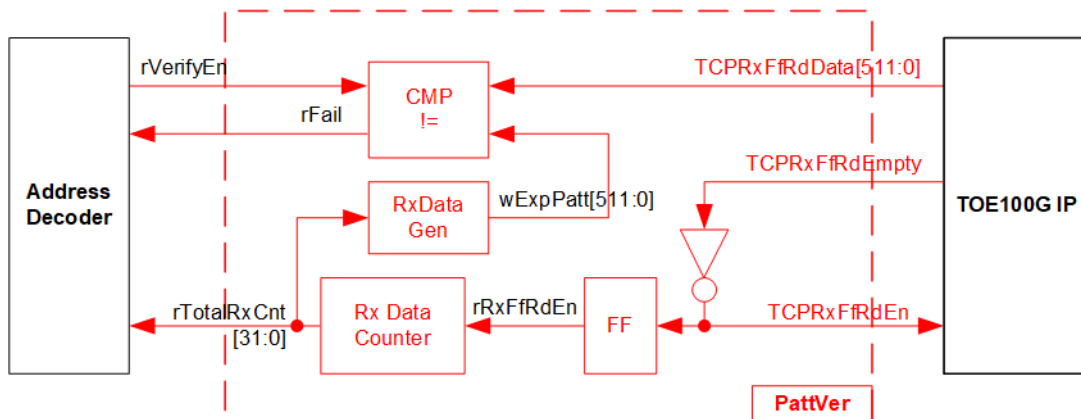


Figure 2-12 PattVer block

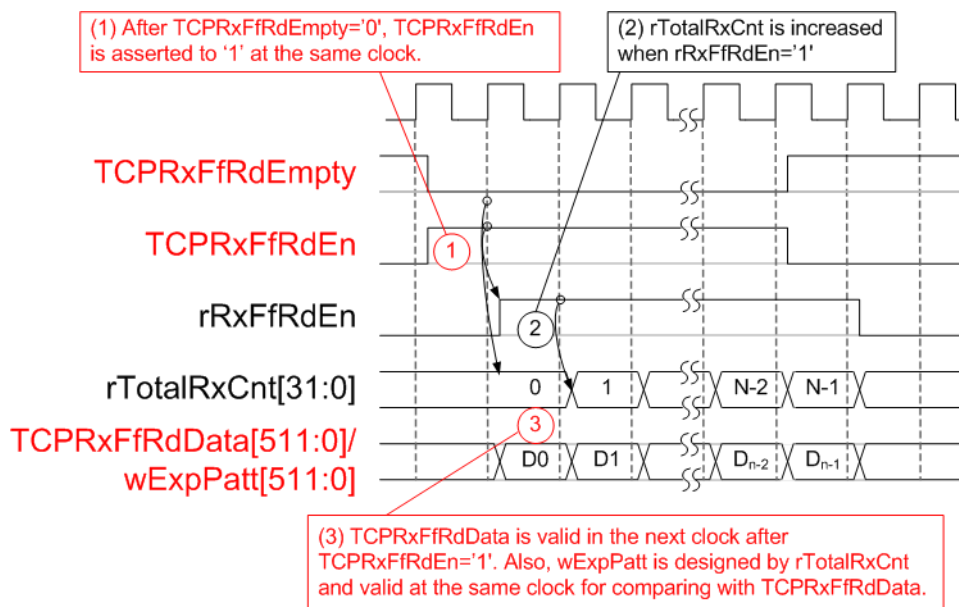


Figure 2-13 PattVer Timing diagram

Figure 2-12 shows the details of PattVer logic for reading the data from TOE100G-IP with or without data verification, controlled by rVerifyEn flag which is set by the user. Timing diagram of the logic is displayed in Figure 2-13.

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 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

For running the multisession reference design and further test modification, user should realize the information below.

- 1) Two end points of the communication are referred, i.e., FPGA#0 (left) and the target (right). The FPGA#0 always contains four TOE100G-IPs while the target can be PC with test application or FPGA#1 with TOE100G-IP(s), as shown in Figure 2-1.
- 2) In FPGA#0, up to four TCP ports/sessions are supported by using four TOE100G-IPs with the same FPGA#0 IP address. It does not need to run all four TOE100G-IPs. For example, one TOE100G-IP is not used when running three sessions in FPGA#0.
- 3) In the target, test system can be applied by using multiple targets to communicate with FPGA#0 when Ethernet switch is added to the system. Each target has unique IP address. However, the total amount of session from all target is up to four sessions.

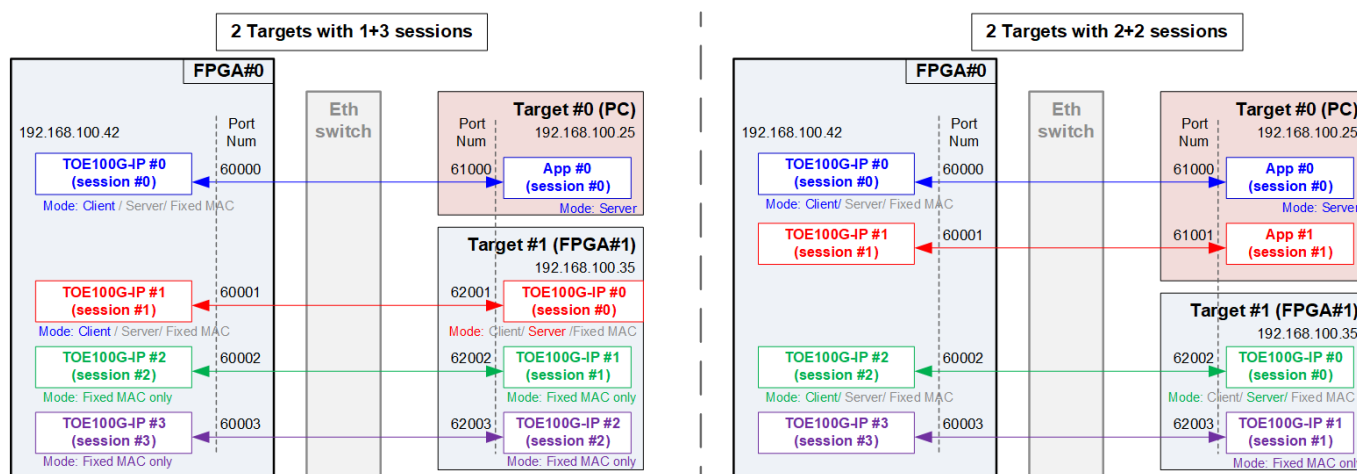


Figure 3-1 Example when running with two targets in test system

Figure 3-1 shows the example to run multisession design by using two target devices with four sessions. In the left side, the first target by PC runs one application for operating one session while the second target by FPGA#1 runs three TOE100G-IPs for operating three sessions. In the right side, the first target by PC runs two applications for operating two sessions while the second target by FPGA#1 runs two TOE100G-IPs for operating two sessions. In the real system, less than four sessions can be operated.

After FPGA (FPGA#0) boot-up, 100G Ethernet link up status (USER_RST_REG[16]) is polling. The CPU waits until link up is found. Next, welcome message is displayed and user selects the initialization mode of TOE100G-IP. There are three modes to be selection choices, i.e., Client, Server, and Fixed-MAC. Only the first session of each target can be assigned the initialization mode. The other sessions of the same target are set to be Fixed MAC.

When the target device is PC, it is recommended to configured the initialization mode on FPGA#0 to be Client mode. The IP initialization is completed when PC returns ARP reply after receiving ARP request. When two FPGAs are communicated, it is free to be initialized by several settings for the first session, i.e., Client <-> Server, Client <-> Fixed-MAC, and Fixed-MAC <-> Fixed MAC.

After receiving the initialization mode from the user, the default parameters in the selected mode are displayed on the console. The user selects to use default parameters or update the parameters before starting initialization. The example when the system is initialized in Client mode by using default parameters is shown in Figure 3-2.

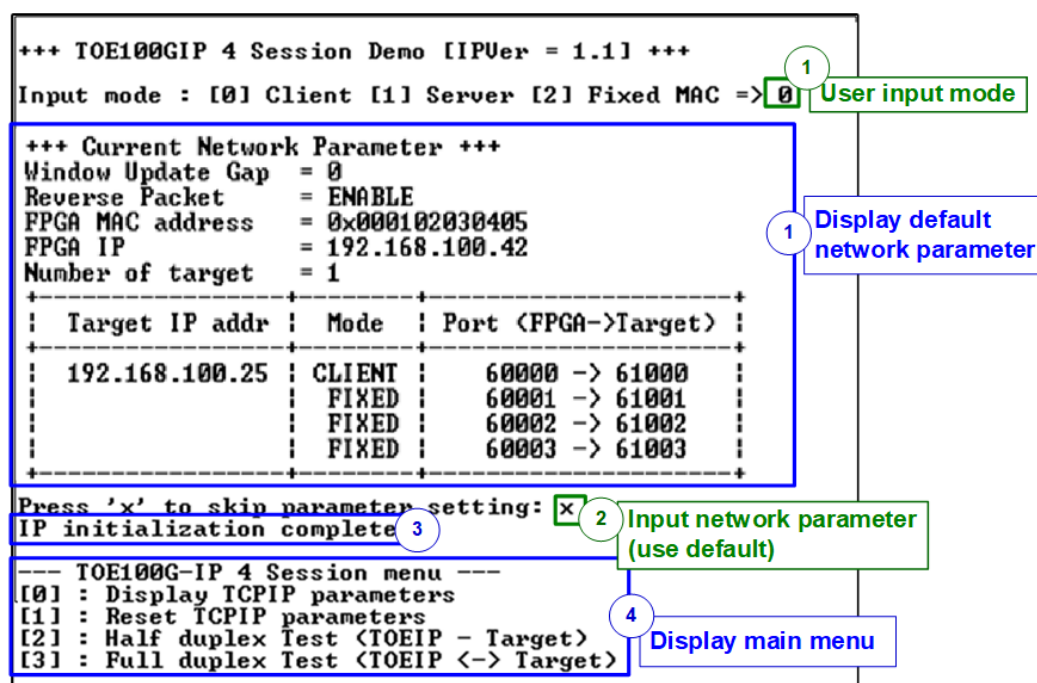


Figure 3-2 System initialization in Client mode by using default parameters

There are four steps to complete initialization sequence 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 all TOE100G-IPs finish initialization sequence (TOE_CMD_INTREG[0]='0').
- 4) Main menu is displayed. There are four 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 all active TOE100G-IPs, i.e., the initialization mode, Reverse packet enable, Windows update threshold, source MAC address, destination IP address, source IP address, destination port, source port, and destination MAC address (when using Fixed-MAC mode). The sequence to display parameters is as follows.

- 1) Read common parameters that are shared for all sessions, i.e., FPGA MAC address, FPGA IP address, Window Update Gap, and reverse packet enable from each variable in firmware.
- 2) Print out each variable.
- 3) Read number of target devices and number of sessions for each target.
- 4) Read session parameters that can be set individually for each session, i.e., Initialization mode, Target IP address, Target MAC address (when using Fixed-MAC mode), FPGA port number, and Target port number from each variable in firmware.
- 5) Print out each variable on the table.

3.2 Reset IP

This menu is used to change TOE100G-IP parameters such as IP address and source port number. After setting new parameters to TOE100G-IP registers, 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) Ask user to skip (use current parameters) or set new parameter values.
 - a. Press 'x' on keyboard to skip. The current parameters are used and continue to step 6.
 - b. Press other keys to start parameter value setting (continue to step 3).
- 3) Receive initialization mode from user and confirm that input is valid.
 - a. If input mode is invalid, mode value will not change and continue to step 4.
 - b. If input mode is valid and the value is changed from previous value, the current parameter set of new mode is displayed on the console. Next, user inputs 'x' to use the current parameters (continue to step 6). Otherwise, user inputs other keys to set the new parameters (continue to step 4).
- 4) Receive new common parameters from user, i.e., Window Update Gap, Reverse packet enable, FPGA MAC address, and FPGA IP address. If an input value is invalid, the parameter is not changed.
- 5) Receive the number of targets and the parameters of each target from user, i.e., the number of sessions, Target MAC address (only in Fixed-MAC mode), Target IP address, and port number. If an input value is invalid, the input is not changed. However, if the number of targets is updated, user needs to set all parameters by valid value.
- 6) Force reset to all IPs by setting TOE_RST_INTREG[0]='1'.
- 7) For the sessions that are enabled,
 - i. Reset PattGen and PattVer logic by sending reset to user logic (USER_RST_INTREG[0]='1').
 - ii. Set all parameters to TOE100G-IP register such as TOE_SML_INTREG and TOE_DIP_INTREG.
7. For the first session of each target device,
 - i. De-assert TOE100G-IP reset by setting TOE_RST_INTREG[0]='0' to start IP initialization process.
 - ii. Wait until the TOE100G-IP completes initialization process (TOE_CMD_INTREG[0]='0').
 - iii. Read the Target MAC address (TOE_DML_INTREG and TOE_DMH_INTREG) to set the same value to the remaining session of the same target device.
8. For the remaining session of each target device,
 - i. Set Target MAC address (TOE_DML_INTREG and TOE_DMH_INTREG) by the same value as the first session.
 - ii. Set initialization mode (TOE_SRV_INTREG) to be Fixed-MAC mode to use the same Target MAC address with the first session.
 - iii. De-assert TOE_RST_INTREG[0] to '0' to start IP initialization.
 - iv. Wait until the TOE100G-IP completes initialization process (TOE_CMD_INTREG[0]='0').

3.3 Half Duplex Test

This menu is designed to transfer data in single direction for the initialized session. The user sets transfer mode to be send data, receive data, or no operation to each session individually. Next, transfer size and connection mode (active open for client mode or passive open for server mode) are assigned by user for the session that sends data or receives data. The last parameter of Send data command is packet size while the last parameters of Receive data command is data verification flag (enable or disable). The operation is cancelled if some inputs are invalid.

To run the test, 32-bit incremental data is generated to send or verify data. The operation is finished when total data of all test session are transferred. When running the test with PC, test application (tcpdatatest) must be run.

The sequence of the test is as follows.

- 1) Display target IP address and port number of the first session.
- 2) Receive transfer mode, transfer size, packet size/data verification mode, and connection mode from user and verify if all inputs are valid.
- 3) Repeat step 2) to get the parameters of the next active session until the current session is the final active session.
- 4) Set UserReg registers following the transfer direction.
 - a. To send data, set transfer size (USER_TXLEN_INTREG), reset flag to clear initial value of PattGen (USER_RST_INTREG[0]='1'), and command register to start PattGen operation (USER_CMD_INTREG=0). After that, PattGen starts sending data to TOE100G-IP.
 - b. To receive data, set reset flag to clear initial value of PattVer (USER_RST_INTREG[0]='1') and command register to start PattVer operation with or without data verification (USER_CMD_INTREG=1/3). After that, PattVer starts verifying data from TOE100G-IP.
- 5) Display recommended parameters of test application on PC following connection mode.
 - a. For active connection mode, the parameters for running test application on PC by server mode are displayed. After that, "Press any key to proceed" is displayed to wait until user runs the test application on PC completely and enters some keys to continue the next step.
 - b. For passive connection mode, the parameters for running test application on PC by client mode are displayed.
- 6) Start open connection following connection mode setting.
 - a. For client mode (active open), CPU sets TOE_CMD_INTREG=2 (Open port) and set current state variable to WAIT_CONN.
 - b. For server mode (passive open), set current state variable to WAIT_CONN.
- 7) Wait until Connection interrupt status (USER_INT_INTREG[0]) is equal to '1'. After that, the current state variable changes to CONNECTED. If busy flag of TOE100G-IP (TOE_CMD_INTREG[0]) is de-asserted to '0' but interrupt is not asserted, the current state variable changes to ERROR. After that, the data starts transferring.

8) The steps to run depends on transfer direction.

Send data command

- i. Confirm the state variable is equal to CONNECTED and TOE100G-IP is not busy (TOE_CMD_INTREG[0]='0'). If not, skip to operate the next session.
- ii. Continue the next step only when the connection is still ON (USER_CMD_INTREG[2]='1'). Otherwise, the state variable is set to ERROR.
- iii. Check total remaining length variable.
 - o If remaining length is equal to 0, run active close command by setting TOE_CMD_REG=3. Similar to active open, the operation is successful when Connection interrupt status (USER_INT_INTREG[0]) is asserted to '1'. Otherwise, the current state variable changes to ERROR if TOE100G-IP busy flag (TOE_CMD_INTREG[0]) is de-asserted without the Connection interrupt asserted. After that, it checks if busy status of user logic (USER_CMD_INTREG[0]) is equal to 0 for changing the state variable is set to CLOSED. If not, the state variable is set to ERROR.
 - o If remaining length variable is not equal to 0, run IP send operation by setting Packet size (TOE_PKL_INTREG), total transfer size (TOE_TDL_INTREG), and Send command (TOE_CMD_INTREG=0) respectively. After that, decrease total remaining length by TOE_TDL_INTREG value.

Note: TOE_TDL_INTREG is set to maximum transfer size (0xFFFF_FFC0) for each round except the final loop that is set to the total remaining length variable.

Receive data command

- i. Confirm the state variable is equal to CONNECTED.
- ii. Read the connection status. If connection status is OFF (USER_CMD_INTREG[2]='0'), set the state variable to CLOSED.

Every second the test progress and connection status of all sessions are displayed. If the data is still transferred, total amount of transmitted data (USER_TXLEN_INTREG) and received data (USER_RXLEN_INTREG) are read and displayed on the console.

- 9) For the session that has just run receive test, compare receive length of user logic (USER_RXLEN_INTREG) with the set value from user and then read verification result (USER_CMD_INTREG[1]). Error message is displayed if receive length or data verification is error.
- 10) Calculate performance and show test result on the console.

3.4 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. Five inputs are received from user, i.e., transfer mode (full-duplex or no operation), transfer size of both directions, packet size for send test, data verification mode for receive test, and connection mode (active open/close for client mode or passive open/close for server mode).

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_single). Connection mode on FPGA when running with PC must be set to passive (server operation).

The sequence of this test is as follows.

- 1) Display target IP address and port number of the first session.
- 2) Receive transfer mode, transfer size, packet size, data verification mode, and connection mode from the user and verify that all inputs are valid.
- 3) Repeat step 2) to get the parameters of the next active session until the current session is the final active session.
- 4) For the session that is set to full-duplex and the connection is passive mode, display the recommended parameters of test application run on PC from the current system parameters.
- 5) For the active session, 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 PattGen and PattVer with data verification mode (USER_CMD_INTREG=0 or 2).
- 6) If there are some sessions setting connection mode to be active mode, "Press any key to proceed" is displayed to wait until user prepares the target device for connection creating. After that, user enters the key to continue the next step.
- 7) Start open connection following connection mode setting, similar to step 6) – 7) of Half duplex test.
- 8) Start data transferring by running following steps.
 - i. Confirm the state variable is equal to CONNECTED and TOE100G-IP is not busy (TOE_CMD_INTREG[0]='0'). If TOE100G-IP is busy, skip to operation the next session.
 - ii. Skip to the next step when the connection is still ON (USER_CMD_INTREG[2]='1'). Otherwise, confirm the connection mode is passive and remaining transmit length variable is equal to 0 before setting the state variable to CLOSED. If the session is active or the remaining transmit length is not equal to 0, the state variable changes to ERROR.
 - iii. Skip to the next step if the remaining transmit length variable is equal to 0. Otherwise, run IP send operation by setting Packet size (TOE_PKL_INTREG), total transfer size (TOE_TDL_INTREG), and Send command (TOE_CMD_INTREG=0), respectively. After that, decrease total remaining length by TOE_TDL_INTREG value.
Note: TOE_TDL_INTREG is set to maximum transfer size (0xFFFF_FFC0) which is aligned to packet size for each round except the final loop that is set to the total remaining length variable.

- iv. If total amount of received data is equal to the set value and the connection mode is active, run active close command by setting 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 current state variable changes to ERROR if TOE100G-IP busy flag (TOE_CMD_INTREG[0]) is de-asserted without the Connection interrupt asserted. After that, it checks if busy status of user logic (USER_CMD_INTREG[0]) is equal to 0 for changing the state variable is set to CLOSED. If not, the state variable is set to ERROR.

Every second the test progress and connection status of all sessions is displayed. If the data is still transferred, total amount of transmitted data (USER_TXLEN_INTREG) and received data (USER_RXLEN_INTREG) are read and displayed on the console.

- 9) Check the result and the error, similar to step 9) of Half duplex test.
- 10) Calculate performance and show the test result on the console.

3.5 Function list in User application

This topic describes the function list to run TOE100G-IP operation.

unsigned int cal_strlen(unsigned int num)	
Parameters	num: integer input to calculate the string length
Return value	ret: the length of string for displaying
Description	Receive the input and then calculate the length of string to display this value in integer style.

void get_toeindex(unsigned int *num_target, unsigned int *num_session)	
Parameters	*num_target: pointer of the number of target variable *num_session: pointer of array that stores the number of sessions of each target
Return value	None
Description	Read all target IP address variable in firmware and then calculate the number of target and the number of sessions of each target. After that, return the result to num_target and num_session.

void init_param(void)	
Parameters	None
Return value	None
Description	This function is called to set the parameters and reset the IP, following described in topic 3.2.

void input_param(void)	
Parameters	None
Return value	None
Description	Receive test parameters from user for both common parameters (Reverse packet enable, Window threshold, FPGA MAC address, and FPGA IP address) and session parameters (Initialization mode, FPGA port number, Target IP address, Target port number, and Target MAC address when running in Fixed MAC mode). After receiving all parameters, the current value of all parameter is displayed.

unsigned int read_conon(unsigned int num)	
Parameters	num: integer index number of session
Return value	0: Connection is OFF, 1: Connection is ON.
Description	Read value from USER_CMD_INTREG register of selected session and return only bit2 value to show connection status.

void show_cursize(unsigned int *test_mode, unsigned int *cur_state)	
Parameters	test_mode: pointer of array that stores test_mode which can be set to No test (0), Send test (1), Receive test (2), or Full-duplex (3) cur_state: pointer of array that stores current state variable
Return value	None
Description	Read and display connection status of all session. If the connection is active, read current amount of transmitted data and received data from USER_TXLEN_INTREG and USER_RXLEN_INTREG and then display on the console in Byte, KByte, or MByte unit.

void show_ipaddr(unsigned int ip_addr)	
Parameters	ip_addr: IP Address in hexadecimal unit
Return value	None
Description	Display IP Address in decimal unit

void show_perf_header(unsigned int *test_mode)	
Parameters	test_mode: pointer of array that stores test_mode which can be set to No test (0), Send test (1), Receive test (2), or Full-duplex (3)
Return value	None
Description	When test mode is not no test (0), read Target IP address, Target port number, and FPGA port number to display as the header of the current status table.

void show_perf_line(unsigned int *test_mode)	
Parameters	test_mode: pointer of array that stores test_mode which can be set to No test (0), Send test (1), Receive test (2), or Full-duplex (3)
Return value	None
Description	When test mode is not no test (0), display straight line to be a part of the current status table.

void show_param(void)	
Parameters	None
Return value	None
Description	Display the current test parameters which consist of common parameters (Windows threshold, Reverse packet enable, FPGA MAC address, and FPGA IP address) and session parameters (Target MAC address, Target IP address, initialization mode, and port number). Total amount of target is also displayed.

void show_result(unsigned int *test_mode, unsigned int *cur_state, unsigned int *tot_rcv, unsigned int *total_len, unsigned int *err_rcv_ver)	
Parameters	test_mode: pointer of array that stores test_mode which can be set to No test (0), Send test (1), Receive test (2), or Full-duplex (3) cur_state: pointer of array that stores current state variable tot_rcv: pointer of array that stores total amount of received data total_len: pointer of array that stores total transferred data size err_rcv_ver: pointer of array that indicates data verification failed
Return value	None
Description	Display error message of receive operation if some error is found. After that, read USER_TXLEN_INTREG and USER_RXLEN_INTREG to display total amount of transmitted data and received data. Also, read the global parameters of timers to 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_full_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 3.4.

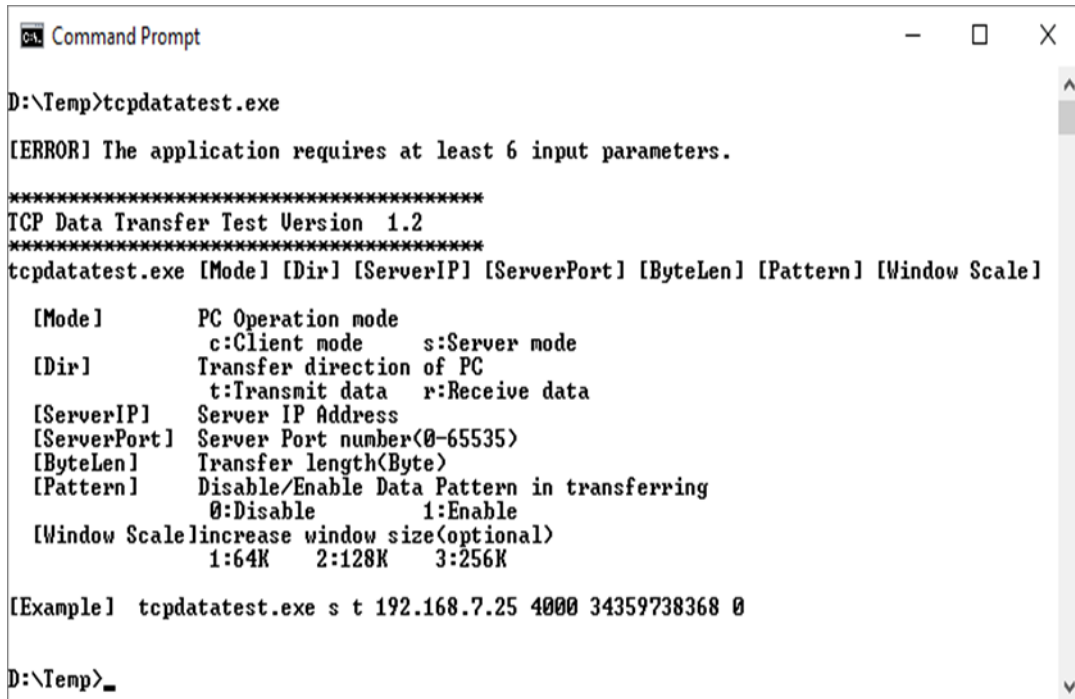
int toe_half_test(void)	
Parameters	None
Return value	0: The operation is successful -1: Receive invalid input or error is found
Description	Run half duplex data test following description in topic 3.3

void toe_tx_send(unsigned int toe_index, unsigned int *total_len, unsigned int *pac_size, unsigned int *round_size)	
Parameters	toe_index: the current number of TOE100G-IP for operating total_len: pointer of the total transfer size to send data pac_size: pointer of the packet size to send data round_size: pointer of the maximum transmit size for each test round
Return value	0: The operation is successful -1: Receive invalid input or error is found
Description	Set TOE100G-IP register to run send command by setting TOE_PKL_INTREG, TOE_TDL_INTREG, and TOE_CMD_INTREG.

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

4 Test Software on PC

4.1 “tcpdatatest” for half duplex test



```

Command Prompt
D:\Temp>tcpdatatest.exe

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

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

[Mode]      PC Operation mode
             c:Client mode    s:Server mode
[Dir]       Transfer direction of PC
             t:Transmit data  r:Receive data
[ServerIP]  Server IP Address
[ServerPort] Server Port number(0-65535)
[ByteLen]   Transfer length(Byte)
[Pattern]   Disable/Enable Data Pattern in transferring
             0:Disable      1:Enable
[Window Scale] increase window size(optional)
             1:64K    2:128K    3:256K

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

D:\Temp>_

```

Figure 4-1 “tcpdatatest” application usage

“tcpdatatest” is designed to run on PC for sending or receiving TCP data via Ethernet as server or client mode. PC of this demo should run in client mode. User sets parameters to select transfer direction and the mode. Six parameters are required as follows.

- 1) Mode: c – PC runs in Client mode and FPGA runs in Server mode
- 2) Dir: t – transmit mode (PC sends data to FPGA)
r – receive mode (PC receives data from FPGA)
- 3) ServerIP: IP address of FPGA when PC runs in Client mode (default is 192.168.7.42)
- 4) ServerPort: Port number of FPGA when PC runs in Client mode (default is 4000)
- 5) ByteLen: Total transfer size in byte unit. This input is used in transmit mode only 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, 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 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_single” for full duplex test



```

Command Prompt
D:\Temp>tcp_client_txrx_single.exe
*****
TCP Tx Rx Version 1.0
*****
tcp_client_txrx_single.exe [ServerIP] [ServerPort] [ByteLen] [Verification]

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

[Example] tcp_client_txrx_single.exe 192.168.40.42 60000 137438953440 0

D:\Temp>

```

Figure 4-2 “tcp_client_txrx_single” application usage

This application is similar to “tcp_client_txrx_40G” application but it runs for one round, not forever loop. “tcp_client_txrx_single” application is designed to run on PC for sending and receiving TCP data through Ethernet by using the same port number at the same time. The application is run in Client mode, so user needs to input server parameters (the network parameters of TOE100G-IP). As shown in Figure 4-2, there are four parameters to run the application, described as follows.

- 1) ServerIP : IP address of FPGA
- 2) ServerPort : Port number of FPGA
- 3) ByteLen : Total transfer size in byte unit. This is total size to transmitted data and received data.
- 4) Verification :
 - 0 – Generate dummy data for sending function and disable data verification for receiving function. This mode is used to check the best performance of full-duplex transfer.
 - 1 – Generate incremental data for sending function and enable data verification for receiving function.

The sequence of test application is as follows.

- 1) Get parameters from the user and verify that the input is valid.
- 2) Create the socket and set socket options.
- 3) Create the new connection by using server IP address and server port number.
- 4) Allocate 64 KB memory for send buffer 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 amount of transmitted 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.

5 Revision History

Revision	Date	Description
1.1	9-Sep-22	Support Ethernet MAC Subsystem and update firmware to use connection interrupt
1.0	21-Mar-22	Initial version release