TOE10GLL-IP reference design

Rev1.1 5-May-21

1 Introduction

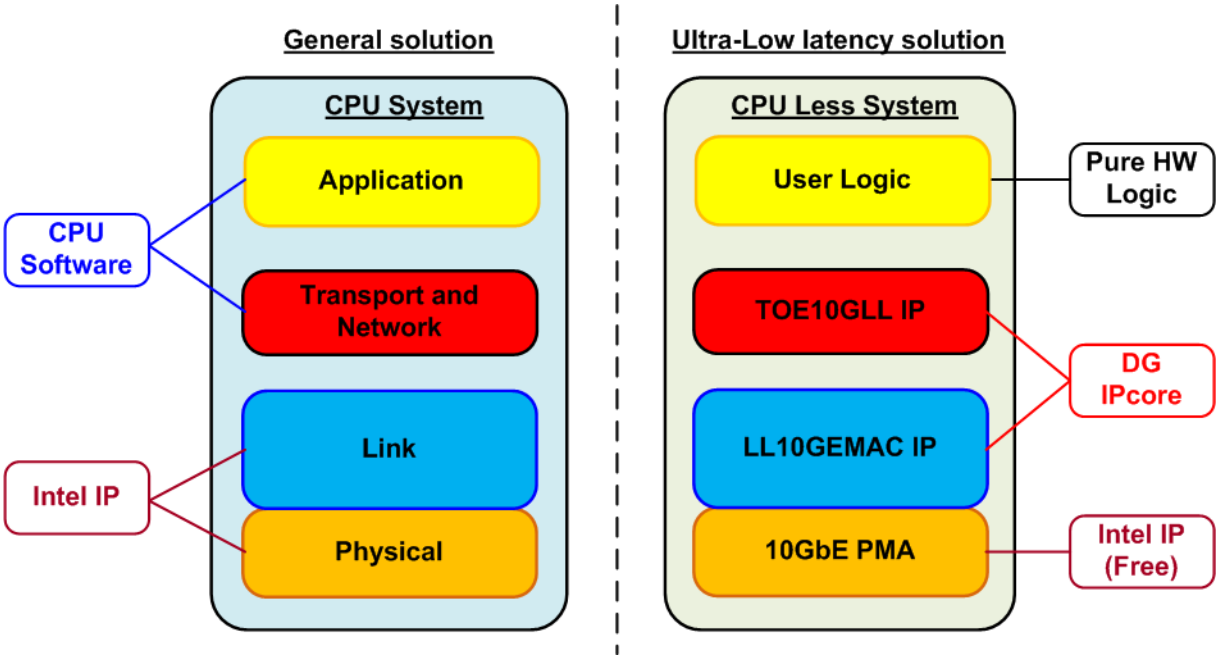

Figure 1-1 Low-latency solution

The general solution for implementing TCP/IP data processor with FPGA is mostly designed by using CPU system for running TCP/IP stack. While the lower layers - link layer and physical layer are implemented by Intel IP core, Low Latency Ethernet 10G MAC, as shown on the left side of Figure 1-1. Though this solution is flexible to design application on CPU, the result shows much latency time for processing both TCP/IP stack and the application.

To achieve the lowest latency solution, the design on the right side of Figure 1-1 is designed, the full hardware logic system for processing TCP/IP packet. This solution is fit with the time-sensitive application that needs to implement the user logic by the hardware logic for transferring TCP data with TOE10GLL-IP. TOE10GLL-IP designs TCP/IP stack with ultra-low latency. Also, it is recommended to connect with the low latency 10G Ethernet MAC IP (LL10GEMACIP) to achieve the lowest latency system. The lowest layer of hardware, PMA layer, is provided by Intel as a free PMA IP.

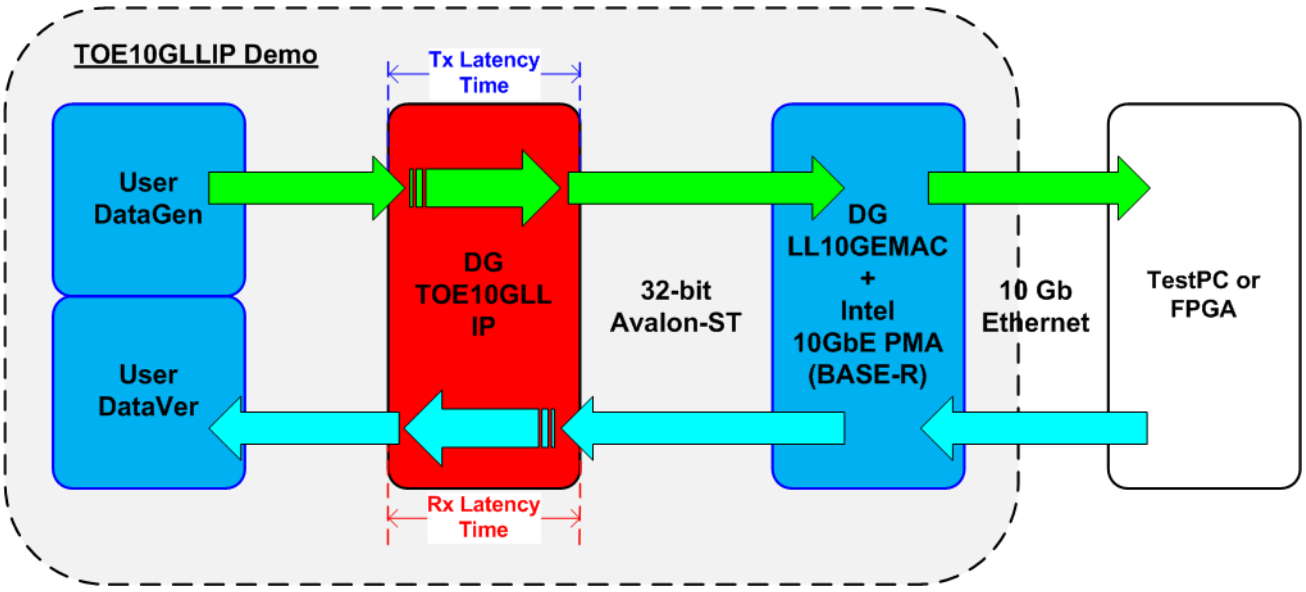

Figure 1-2 Test logic for TOE10GLL-IP

The simple test logic is designed to show TCP/IP stack implementation with achieving low latency time as shown in Figure 1-2. User logic can be separated into two parts: UserDataGen and UserDataVer. UserDataGen generates the 32-bit incremental data pattern and transfers to TOE10GLL-IP to build TCP packet. Next, TCP packet is forwarded to LL10GEMAC-IP and Intel 10GbE PMA (BASE-R). The target device can be designed by TestPC or another FPGA which runs the same hardware but verifies the received data by UserDataVer. When using TestPC, the test software is run on TestPC for data verification. Besides, the latency time of TOE10GLL-IP when transmitting data is measured by using the timer.

On the other hand, UserDataVer verifies the received TCP packet from TOE10GLL-IP, sent via 10G Ethernet from Test software on PC or another FPGA board. Similar to Tx path, the latency time of TOE10GLL-IP when receiving data is measured by the timer. The latency time for both directions is measured from start-of-frame to start-of-frame.

CPU system is included for user to interface with hardware logic via JTAG UART. Network parameters of the test system can be set by the user from the NiosII command shell as well as the parameters of the test logic such as total transfer length and packet size. Also, the test results, latency time and the progress of test operation, are returned to CPU and then displayed on the NiosII command shell. More details of the demo are described as follows.

2 Hardware overview

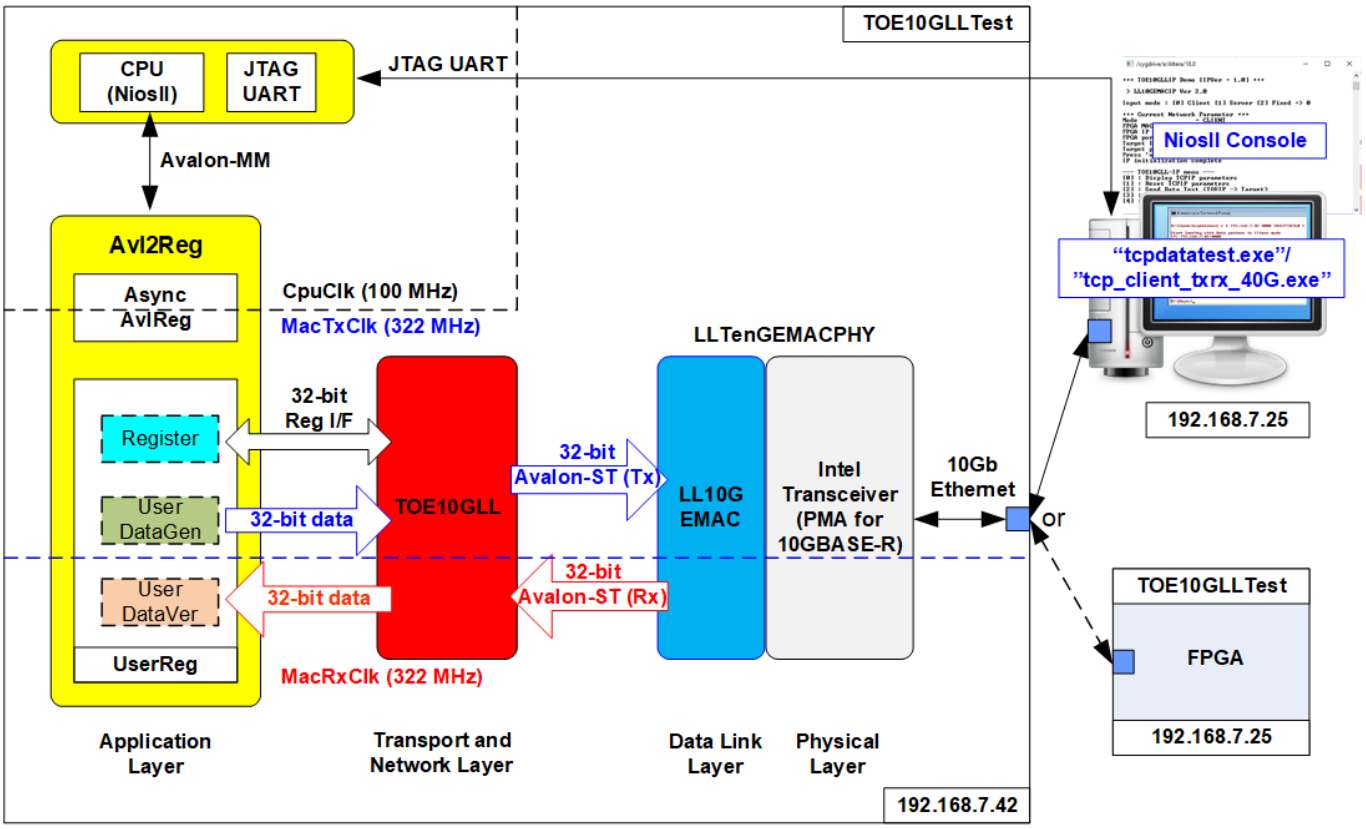

Figure 2-1 Demo Block Diagram

The test system includes CPU for the ease of user interface and flexibility of test environment. User can input the test parameters such as network parameters and transfer size. Also, the current status of the test system such as current transfer size is displayed on NiosII command shell for monitoring the test progress. To connect the hardware with CPU system, Avalon-MM bus must be implemented. Avl2Reg is the interface module to convert Avalon-MM interface to be the user interface of TOE10GLL-IP module. Avl2Reg includes AsyncAvlReg which is designed to be asynchronous module between CpuClk which is the independent clock for running the CPU system and MacTxClk which is the clock output, generated by Intel transceiver module.

The user interface of TOE10GLL-IP connects to UserReg within Avl2Reg module to control and monitor the TOE10GLL-IP. Also, UserReg consists of UserDataGen module which is the 32-bit test data generator and UserDataVer module which verifies the data, extracted from received packet. Register files of UserReg are written and read by CPU firmware through Avalon-MM bus. Another side of TOE10GLL-IP is connected to 10G Ethernet MAC controller (LL10GEMAC-IP) by using 32-bit Avalon stream interface (Avalon-ST). LL10GEMAC-IP implements the Ethernet MAC layer and PCS layer with less latency time. Tx and Rx interface of Avalon-ST are run in the different clock domains: MacTxClk and MacRxClk respectively.

LL10GEMAC-IP provided by Design Gateway requires to run with Intel Transceiver which is configured to be PMA module for 10GBASE-R interface.

Another side of 10Gb Ethernet is the target device, Test PC or another FPGA board. When using TestPC, the test application - tcpdatatest and tcp_client_txrx_40G must be run for transferring TCP data. Otherwise, another FPGA board is applied by implementing TOE10GLL-IP to transfer data in the different mode to show the best performance.

2.1 Intel Transceiver (PMA for 10GBASE-R)

PMA IP core for 10Gb Ethernet (BASE-R) can be generated by using Quartus IP catalog. In FPGA Transceivers Wizard, the user uses the following settings.

- Transceiver configuration rules : PCS Direct
- Data rate : 10312.5 Mbps
- PCS Direct interface width : 32

More details are described in the following link.
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-10/ug_arria10_xcvr_phy.pdf

2.2 LL10GEMAC

The IP core by Design Gateway implements low-latency EMAC and PCS logic for 10Gb Ethernet (BASE-R) standard. The user interface is 32-bit Avalon-stream bus. Please see more details from LL10GEMAC datasheet on our website.
https://dgway.com/products/IP/Lowlatency-IP/dg_ll10gemacip_data_sheet_intel.pdf

2.3 TOE10GLL

TOE10GLL-IP is the IP core provided by Design Gateway to implement the TCP/IP stack and offload engine for the low latency solution. User interface has two signal groups, i.e., control signals and data signals. More details are described in datasheet.
https://dgway.com/products/IP/Lowlatency-IP/dg_toe10gllip_data_sheet_intel.pdf

2.4 CPU and Peripherals

32-bit Avalon-MM is applied to be the bus interface for the CPU accessing the peripherals such as Timer and JTAG 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 Avalon-MM bus. CPU assigns the different base address and the address range to each peripheral for accessing one peripheral at a time.

In the reference design, the CPU system is built with one additional peripheral to access the test logic. The base address and the range for accessing the test logic are defined in the CPU system. Therefore, the hardware logic must be designed to support Avalon-MM bus standard for CPU write access and read access. Avl2Reg module is designed to connect the CPU system as shown in Figure 2-2.

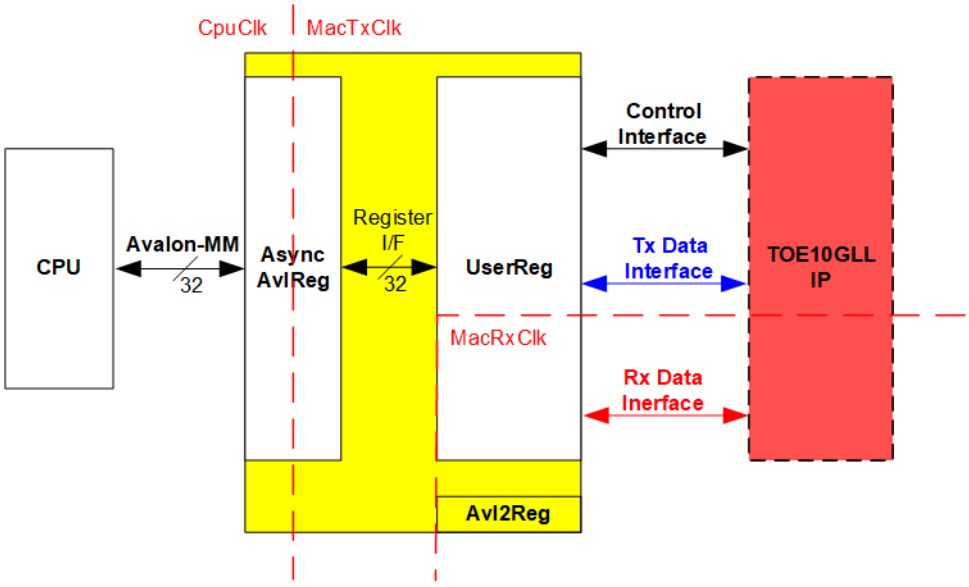

Figure 2-2 Avl2Reg block diagram

Avl2Reg consists of AsyncAvlReg and UserReg. AsyncAvlReg is designed to convert the Avalon-MM signals to be the simple register interface which has 32-bit data bus size (similar to Avalon-MM data bus size). Besides, AsyncAvReg includes asynchronous logic to support clock crossing between CpuClk domain and MacTxClk domain. Tx data interface and Rx data interface of TOE10GLL-IP are run in different clock domain, MacTxClk and MacRxClk. Therefore, UserReg module consists of the logics which run in two clock domains.

UserReg includes the register file of the parameters and the status signals of test logics, including TOE10GLL-IP. Both data interface and control interface of TOE10GLL-IP are connected to UserReg. More details of AsyncAvlReg and UserReg are described as follows.

2.4.1 AsyncAvlReg

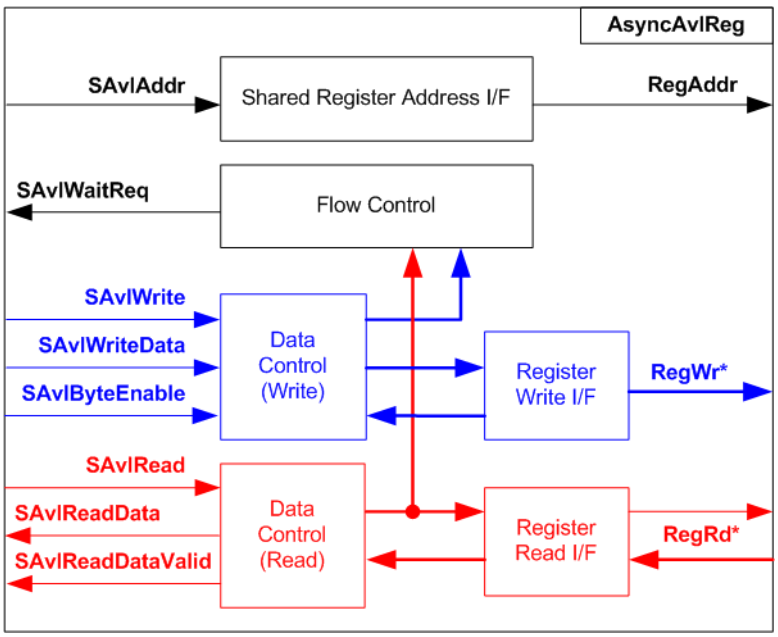

Figure 2-3 AsyncAvlReg Interface

The signal on Avalon-MM bus interface can be split into three groups, i.e., Write channel (blue color), Read channel (red color), and Shared control channel (black color). More details of Avalon-MM interface specification are described in following document.
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf

According to Avalon-MM specification, one command (write or read) can be operated at a time. The logics inside AsyncAvlReg are split into three groups, i.e., Write control logic, Read control logic, and Flow control logic. Flow control logic controls SAvlWaitReq to hold the next request from Avalon-MM interface if the current request does not finish. Write control and Write data I/F of Avalon-MM bus are latched and transferred to be Write register interface with clock-crossing registers. Similarly, Read control I/F are latched and transferred to be Read register interface with clock-crossing registers. After that, the returned data from Register Read I/F is transferred to Avalon-MM bus by using clock-crossing registers. Address I/F of Avalon-MM is latched and transferred to Address register interface as well.

The simple register interface is compatible with single-port RAM interface for write transaction. The read transaction of the register interface is slightly modified from RAM interface by adding RdReq and RdValid signals for controlling read latency time. The address of register interface is shared for write and read transaction, so user cannot write and read the register at the same time. The timing diagram of the register interface is shown in Figure 2-4.

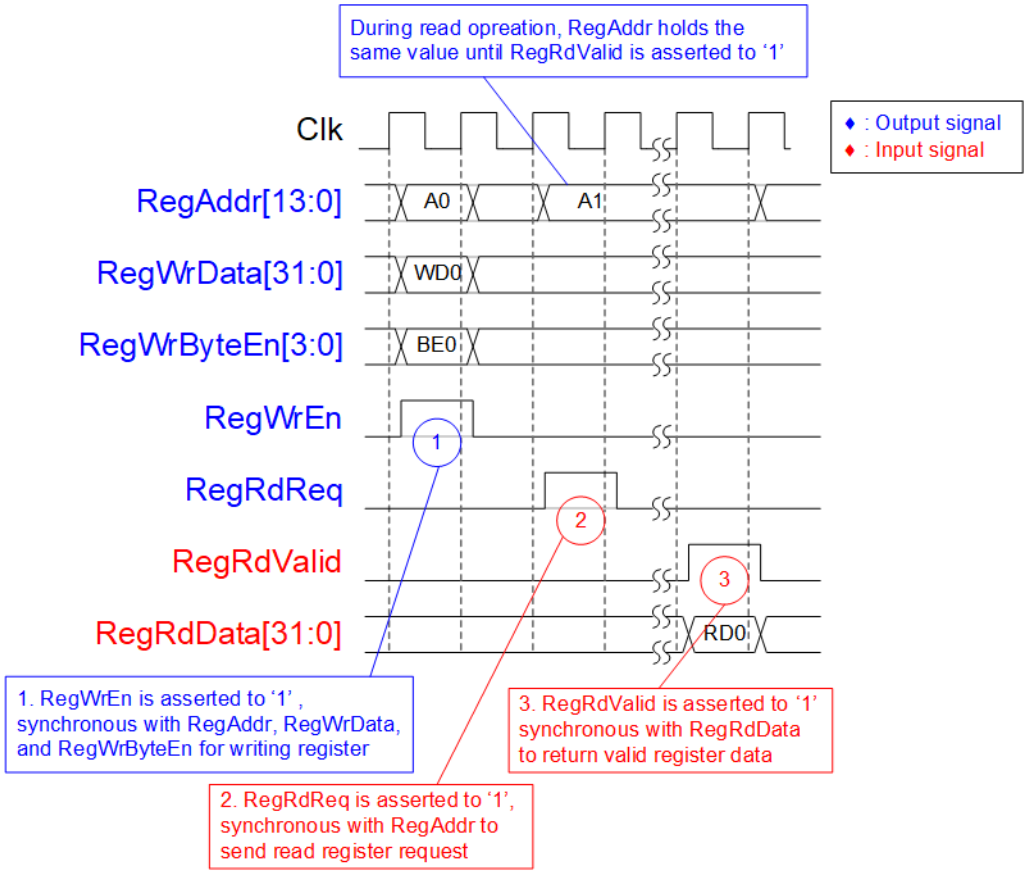

Figure 2-4 Register interface timing diagram

1) To write register, the timing diagram is similar to single-port RAM interface. RegWrEn is asserted to '1' with the valid signal of RegAddr (Register address in 32-bit unit), RegWrData (write data of the register), and RegWrByteEn (the write byte enable). Byte enable has four bits to enable 4-byte data. 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, AsyncAvlReg asserts RegRdReq to '1' with the valid value of RegAddr. 32-bit data must be returned after receiving the read request. The slave detects RegRdReq asserted to start the read transaction. During read operation, the address value (RegAddr) does not change until RegRdValid is asserted to '1'. Therefore, the address can be used for selecting the returned data by using multiple levels of multiplexer.
3) The read data is returned on RegRdData bus by the slave with asserting RegRdValid to '1'. After that, AsyncAvlReg forwards the read value to SAvlRead * interface.

2.4.2 UserReg

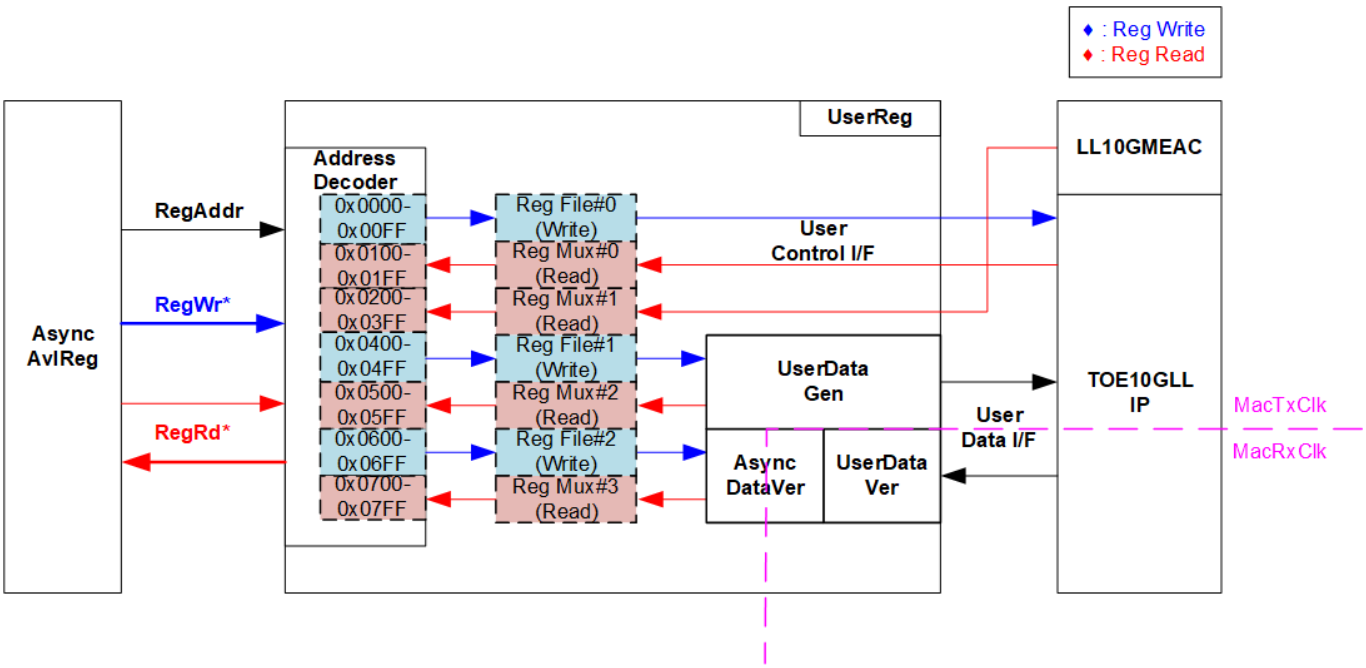

Figure 2-5 UserReg block diagram

The logic inside UserReg consists of three operations, i.e., Register interface (Address decoder, RegFile, and RegMux), Data pattern generator (UserDataGen), and Data pattern verification (UserDataVer and AsyncDataVer).

Register block decodes the address which is requested from AsyncAvlReg and then selects the active register for write or read transaction. UserDataGen block is designed to send 32-bit test data to TOE10GLL-IP. UserDataVer block is designed to read and verify 32-bit data from TOE10GLL-IP following Avalon-Stream interface standard. AsyncDataVer is asynchronous circuit of UserDataVer which is run in MacRxClk domain to interface with other modules which are run in MacTxClk domain and vice versa. More details of Register block, UserDataGen, and UserDataVer are described as follows.

Register Block

The address range assigned in UserReg is split into four areas for four hardware logics.

- 0x0000 – 0x01FF: TOE10GLL-IP
- 0x0200 – 0x03FF: LL10GEMAC-IP
- 0x0400 – 0x05FF: UserDataGen
- 0x0600 – 0x07FF: UserDataVer

Address decoder decodes the upper bit of RegAddr for selecting the active hardware that is TOE10GLL-IP, LL10GEMAC-IP, UserDataGen, or UserDataVer. The register file inside UserReg is 32-bit bus size, so write byte enable (RegWrByteEn) is ignored. The CPU must use 32-bit pointer to place 32-bit valid value on the write data bus.

To read register, multi-level multiplexers (mux) select the data to return to CPU following the address. The lower bit of RegAddr is applied to the mux in the submodule for selecting the internal signals while the upper bit is applied to the mux in UserReg to select the result from each submodule. Totally, the latency time of read data is equal to two clock cycles. Therefore, RegRdValid is created by RegRdReq with asserting two D Flip-flops. More details of the address mapping within UserReg module are shown in Table 2-1.

Table 2-1 Register map Definition

Address Wr/Rd	Register Name (Label in "toe10glltest.c")	Description
BA+0x0000 – BA+0x03FF: TOE10GLL-IP and LL10GEMAC-IP interface More details of each signal are described in TOE10GLL-IP and LL10GEMAC-IP datasheet.		
BA+0x0000 – BA+0x00FF: Input signals of TOE10GLL-IP (Write access only)		
BA+0x0000	TOE_RST_REG	[0]: Mapped to RstB of TOE10GLL-IP
BA+0x0004	TOE_DMM_REG	[1:0]: Mapped to DstMacMode of TOE10GLL-IP
BA+0x0008	TOE_SML_REG	[31:0]: Mapped to SrcMacAddr[31:0] of TOE10GLL-IP
BA+0x000C	TOE_SMH_REG	[15:0]: Mapped to SrcMacAddr[47:32] of TOE10GLL-IP
BA+0x0010	TOE_DMIL_REG	[31:0]: Mapped to DstMacAddr[31:0] of TOE10GLL-IP
BA+0x0014	TOE_DMIH_REG	[15:0]: Mapped to DstMacAddr[47:32] of TOE10GLL-IP
BA+0x0018	TOE_SIP_REG	[31:0]: Mapped to SrcIPAddr of TOE10GLL-IP
BA+0x001C	TOE_DIP_REG	[31:0]: Mapped to DstIPAddr of TOE10GLL-IP
BA+0x0020	TOE_TMO_REG	[31:0]: Mapped to TimeOutSet of TOE10GLL-IP
BA+0x0040	TOE_CMD_REG	[1:0]: Mapped to TCPCmd of TOE10GLL-IP. When this register is written, TCPCmdValid of TOE10GLL-IP is asserted to '1' for one cycle.
BA+0x0044	TOE_SPN_REG	[15:0]: Mapped to TCPSrcPort[15:0] of TOE10GLL-IP
BA+0x0048	TOE_DPN_REG	[15:0]: Mapped to TCPDstPort[15:0] of TOE10GLL-IP
BA+0x0100 – BA+0x01FF: Output signals of TOE10GLL-IP (Read access only)		
BA+0x0100	TOE_VER_REG	[31:0]: Mapped to IP version of TOE10GLL-IP
BA+0x0104	TOE_STS_REG	[0]: Mapped to InitFinish of TOE10GLL-IP [1]: Mapped to TCPConnOn of TOE10GLL-IP [2]: TOE10GLL-IP Interrupt. Asserted to '1' when IPInt of TOE10GLL-IP is asserted to '1'. This flag is cleared after TOE_INT_REG is read. [20:16]: Mapped to IPState of TOE10GLL-IP
BA+0x0108	TOE_INT_REG	[31:0]: Mapped to IntStatus of TOE10GLL-IP
BA+0x010C	TOE_DMOL_REG	[31:0]: Mapped to DstMacAddrOut[31:0] of TOE10GLL-IP
BA+0x0110	TOE_DMOH_REG	[15:0]: Mapped to DstMacAddrOut[47:32] of TOE10GLL-IP
BA+0x0200 – BA+0x03FF: Output signals of LL10GEMAC-IP (Read access only)		
BA+0x0200	EMAC_VER_REG	[31:0]: Mapped to IP version of DG LL10GEMAC-IP
BA+0x0204	EMAC_STS_REG	[0]: Mapped to Linkup of LL10GEMAC-IP

Address Wr/Rd	Register Name (Label in “toe10glltest.c”)	Description
BA+0x0400 – BA+0x07FF: UserDataGen and UserDataVer interface		
BA+0x0400 – BA+0x04FF: Input signals of UserDataGen (Write access only)		
BA+0x0400	USRTX_CMD_REG	[0]: Start flag to generate test data by UserDataGen. Set ‘1’ to start sending data. This flag is auto-cleared after running the operation. [1]: Set ‘1’ to clear USRTX_LENL/H_REG and USRTX_TMR_REG.
BA+0x0404	USRTX_PKL_REG	[10:0]: Packet length in byte unit for assigning TCPTxPkLen, input of TOE10GLL-IP. The best performance is achieved when this value is aligned to 4-byte unit (bit[1:0]=00b).
BA+0x0408	USRTX_PSH_REG	[0]: PSH flag for assigning TCPTxPSH, input of TOE10GLL-IP
BA+0x040C	USRTX_TDL_REG	[31:0]: Bit[31:0] of total size in byte unit to generate test data by UserDataGen. The value is cleared by USRTX_CMD_REG[1].
BA+0x0410	USRTX_TDH_REG	[15:0]: Bit[47:32] of total size in byte unit to generate test data by UserDataGen. The value is cleared by USRTX_CMD_REG[1].
BA+0x0500 – BA+0x05FF: Output signals of UserDataGen (Read access only)		
BA+0x0500	USRTX_STS_REG	[0]: Busy signal of UserDataGen (‘0’-Idle, ‘1’-Data is transmitting)
BA+0x0504	USRTX_TMR_REG	Timer value which shows latency time in Tx interface of TOE10GLL-IP [15:0]: Tx latency time of TOE10GLL-IP in clock cycle unit The value is cleared by USRTX_CMD_REG[1].
BA+0x0508	USRTX_LENL_REG	[31:0]: Bit[31:0] of complete transmit size in byte unit which is calculated by the sum of TCPTxCplLen, output from TOE10GLL-IP. The value is cleared by USRTX_CMD_REG[1].
BA+0x050C	USRTX_LENH_REG	[15:0]: Bit[47:32] of complete transmit size in byte unit which is calculated by the sum of TCPTxCplLen, output from TOE10GLL-IP. The value is cleared by USRTX_CMD_REG[1].
BA+0x0600 – BA+0x06FF: Input signals of UserDataVer (Write access only)		
BA+0x0600	USRRX_CMD_REG	[0]: Enable data verification of UserDataVer (‘0’: Disable data verification, ‘1’: Enable data verification) [1]: Set ‘1’ to clear USRRX_LENL/H_REG and USRRX_TMR_REG are reset.
BA+0x0700 – BA+0x07FF: Output signals of UserDataVer (Read access only)		
BA+0x0700	USRRX_STS_REG	[0]: Verify fail (‘0’-No error, ‘1’-Received data is incorrect) [1]: Receive error interrupt. Asserted to ‘1’ when TCPRxError from TOE10GLL-IP shows error status. This flag can be cleared after USRRX_ERR_REG is read.
BA+0x0704	USRRX_TMR_REG	Timer value which shows latency time in Rx interface of TOE10GLL-IP [15:0]: Rx Latency time of TOE10GLL-IP in clock cycle unit. The value is cleared by USRRX_CMD_REG[1].
BA+0x0708	USRRX_ERR_REG	[7:0]: Latch signal of TCPRxError, output of TOE10GLL-IP. The value does not change after detecting error. The value is reset after this register is read by CPU.
BA+0x0720	USRRX_LENL_REG	[31:0]: Bit[31:0] of current receive data size in byte unit which is counted when the packet received from TOE10GLL-IP is valid. The value is cleared by USRRX_CMD_REG[1].
BA+0x0724	USRRX_LENH_REG	[15:0]: Bit[47:32] of current receive data size in byte unit which is counted when the packet received from TOE10GLL-IP is valid. The value is cleared by USRRX_CMD_REG[1].
BA+0x0728	USRRX_FAILPOSL_REG	[31:0]: Bit[31:0] of the first position in byte unit that data verification detects failure. The value is cleared by USRRX_CMD_REG[1].
BA+0x072C	USRRX_FAILPOSH_REG	[15:0]: Bit[47:32] of the first position in byte unit that data verification detects failure. The value is cleared by USRRX_CMD_REG[1].
BA+0x0730	USRRX_EXPPAT_REG	[31:0]: Expected data that IP should receive when data verification is failed.
BA+0x0734	USRRX_RDPAT_REG	[31:0]: Received data which is an incorrect data when data verification is failed.

User Data Generator

UserDataGen receives the parameters for generating test pattern to TOE10GLL-IP and then start generating the test data by using 32-bit incremental pattern, starting from zero value. Total data size and packet size are set via Register interface. Busy signal is asserted to '1' when the module starts the operation. It is de-asserted to '0' when the module finishes the operation. Therefore, Busy is monitored to wait until the operation completes.

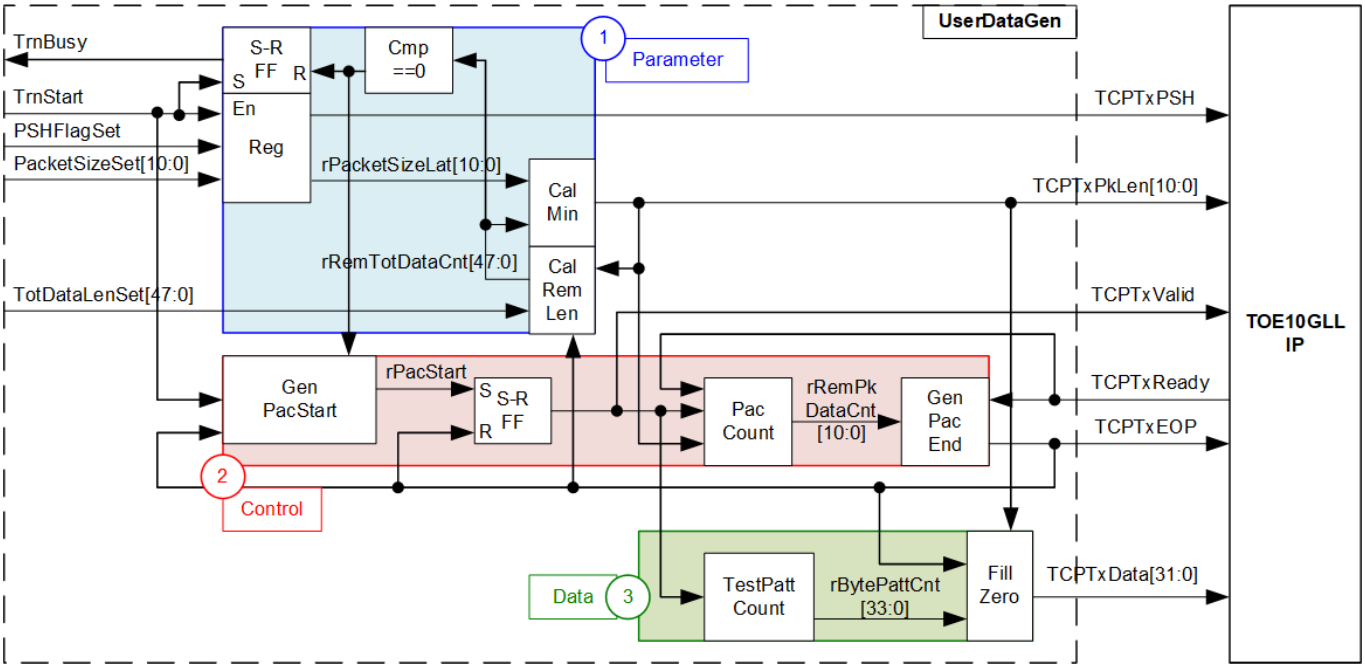

Figure 2-6 UserDataGen block

As shown in Parameter block (1), there are three parameters, input from Register interface, to set UserDataGen module.

a. PSHFlagSet: Parameter assigned to TCPTxPSH of TOE10GLL-IP. The value does not change until the test operation is finished. Therefore, all packets in the test use the same TCPTxPSH value though TOE10GLL-IP supports to set TCPTxPSH for each transmitted packet independently.
b. PacketSizeSet: Parameter assigned to TCPTxPkLen of TOE10GLL-IP. Similar to TCPTxPSH, TCPTxPkLen is set by the same value for every packet until finishing the operation, except the last packet. The last packet of the test operation is assigned by the remaining transfer length which is less than or equal to TCPTxPkLen.
c. TotDataLenSet: Total size for generating test data in byte unit. If this value is more than packet size (PacketSizeSet), many packets are generated by UserDataGen to TOE10GLL-IP until total number of transmitted data is equal to TotDataLenSet.

From above specification, Reg module loads and latches PSHFlagSet and PacketSizeSet when test operation begins (TrnStart is asserted to '1'). TCPTxPSH is the direct output from Reg module while TCPTxPkLen must be calculated by finding the minimum value (CalMin) between the packet size set by the user (rPacketSizeLat) or the remaining data size (rRemTotDataCnt). The remaining data size is designed by using calculating unit, CalRemLen, which loads the start value from TotDataLenSet and then subtracts to the packet size at the end of packet. TrnBusy is designed by S-R Flip flop which is set when test operation begins and clear when remaining data size (rRemTotDataCnt) is equal to 0.

Control block (2) shows the interface of data flow control signals, i.e., TCPTxValid, TCPTxReady, and TCPTxEOP for transferring TCPTxData. To start generating each packet, GenPacStart asserts a pulse of rPacStart which is asserted to '1' by two conditions. First, the test operation begins when TrnStart='1' for creating the first packet. Second, the previous packet is completely transferred (TCPTxEOP='1') and there are remained data to transfer (rRemTotDataCnt is not equal to 0) for creating the remained packet. After rPacStart is asserted to '1', TCPTxValid, created by S-R Flip Flop, is asserted to '1' to start sending data to TOE10GLL-IP until end of a packet (TCPTxEOP='1'). During transferring data to TOE10GLL-IP, it needs to use a down-counter, PacCount, to count the number of generated data. At the first data, rRemPkDataCnt is equal to packet size (TCPTxPkLen) and then decreased until it is less than 4-byte, the end of packet. At the same cycle that generates the last data, TCPTxEOP is asserted to '1'. TCPTxReady is applied to confirm that the transmitted data is received by TOE10GLL-IP completely. TCPTxReady is always asserted to '1' during a packet transferring. It is de-asserted after sending end of packet.

Data block (3) is designed to generate 32-bit incremental data to TOE10GLL-IP. According to TOE10GLL-IP specification, zero data must be filled as dummy byte to the last data of a packet if the packet size is not aligned to 4-byte. Therefore, TCPTxEOP is applied to fill the zero value when TCPTxPkLen[1:0] is not equal to 00b.

User Data Verification

UserDataVer receives the data from TOE10GLL-IP and then verifies the value if verification enable, VerifyEn, is set to '1' by the user. The expected pattern is 32-bit incremental pattern. The received packet from TOE10GLL-IP may have the error when the signal quality of ethernet connection is bad condition to make the packet broken. The expected data for verification after receiving the error packet must be reversed to the latest valid packet to re-verify the new packet.

Note: When the received packet is error, TOE10GLL-IP starts the data recovery process. After that, the same data packet is re-transmitted again.

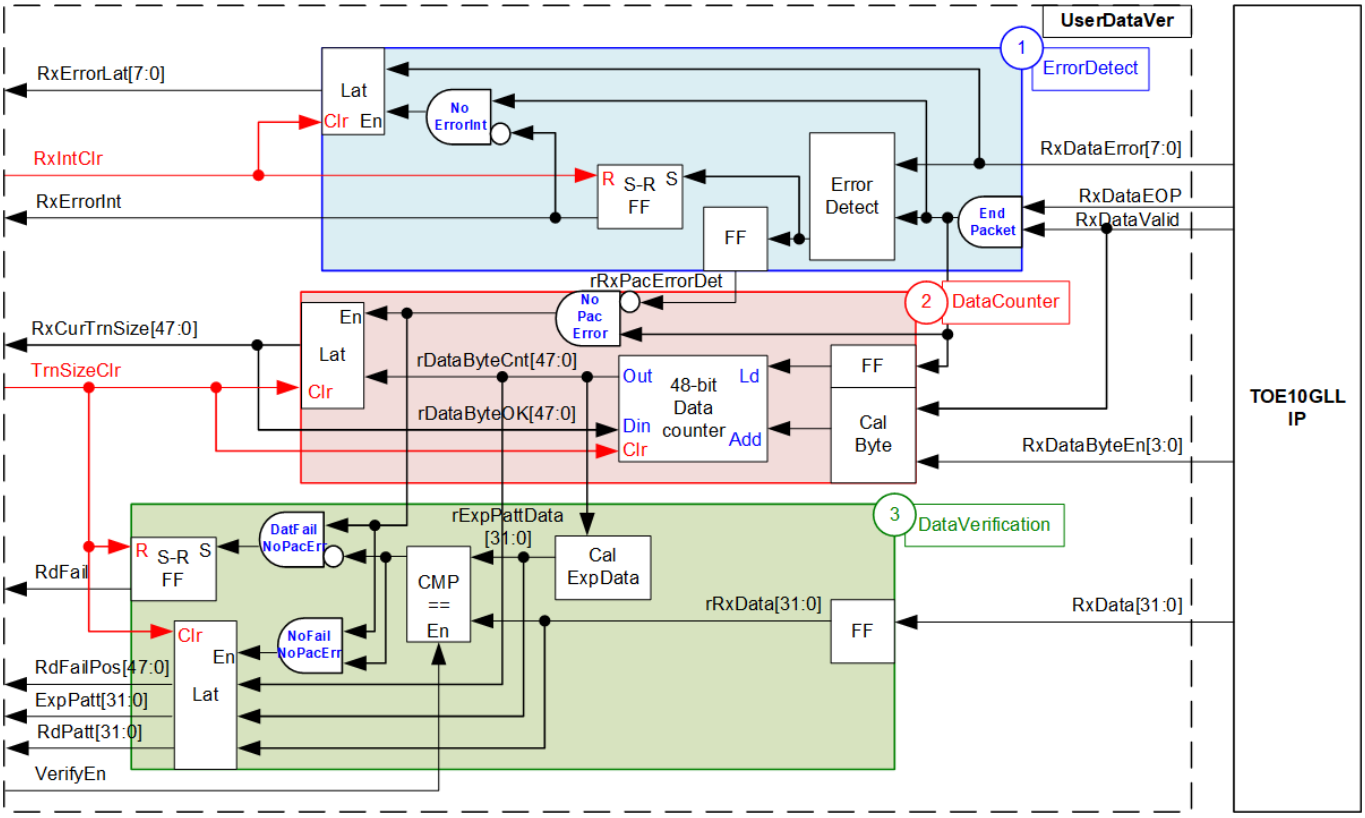

Figure 2-7 UserDataVer block

As shown in Figure 2-7, UserDataVer has three operations, i.e., Error detection to detect the error from TOE10GLL-IP, Data counter to count total number of valid data, and Data verification to verify the received data.

Error detect block (1) reads the error output from TOE10GLL-IP, RxDataError, when the end of packet is received (RxDataEOP='1' and RxDataValid='1'). If the error is detected, the interrupt flag, RxErrorInt, is asserted to '1' with the latch value of RxDataError, RxErrorLat. After the user completely reads the interrupt status, interrupt clear flag (RxIntClr) is asserted to clear the current status.

Data counter block (2) includes the counter to count total number of data, received from TOE10GLL-IP. Valid signal of received data, RxDataValid, and byte enable, RxDataByteEn, are monitored to check the number of valid bytes in each cycle that can be equal to 1 – 4 bytes. After that, the data counter, rDataByteCnt, is added by the valid received byte. To support the re-transmitted packet from error recovery process, latch register is designed to load total number of received data at the end of packet when the packet is valid. This value, rDataByteOK, is applied to be load value of 48-bit data counter before starting receiving the new packet. Therefore, it guarantees that total data count continues to count the valid data without including the error packet. After finishing the test operation, the user can assert clear flag, TrnSizeClr, to clear the data counter for preparing the next test.

Data verification block (3) has CalExpData which calculates the expected data from the current data count, generated in block (2). Only 34-bit of the data count is applied. Bit[1:0] is the byte alignment for setting byte offset of 32-bit expected data. While bit[33:2] is 32-bit expected data. The expected data (rExpPattData) is compared with the received data (rRxData) when the user enables the verification flag, VerifyEn. If data verification error is found while the packet is not error, Failed flag (RdFail) is asserted to ‘1’. The other signals to show verification error information, i.e., the error position in byte unit (RdFailPos), the expected value (ExpPatt), and the received value (RdPatt) are latched to show the details of the first data which is error. The error flag is cleared by asserting TrnSizeClr to ‘1’.

Asynchronous module for UserDataVer

Most logics in UserReg are run in the same clock as Tx interface of TOE10GLL-IP (MacTxClk), except UserDataVer module which is run in the same clock as Rx interface of TOE10GLL-IP, MacRxClk. Therefore, the signal interface with UserDataVer needs asynchronous circuit for clock-crossing domain. AsyncDataVer is designed to convert the signal from MacTxClk to MacRxClk and vice versa.
Note: TxClk of AsyncDataVer is connected to MacTxClk while RxClk is connected to MacRxClk.

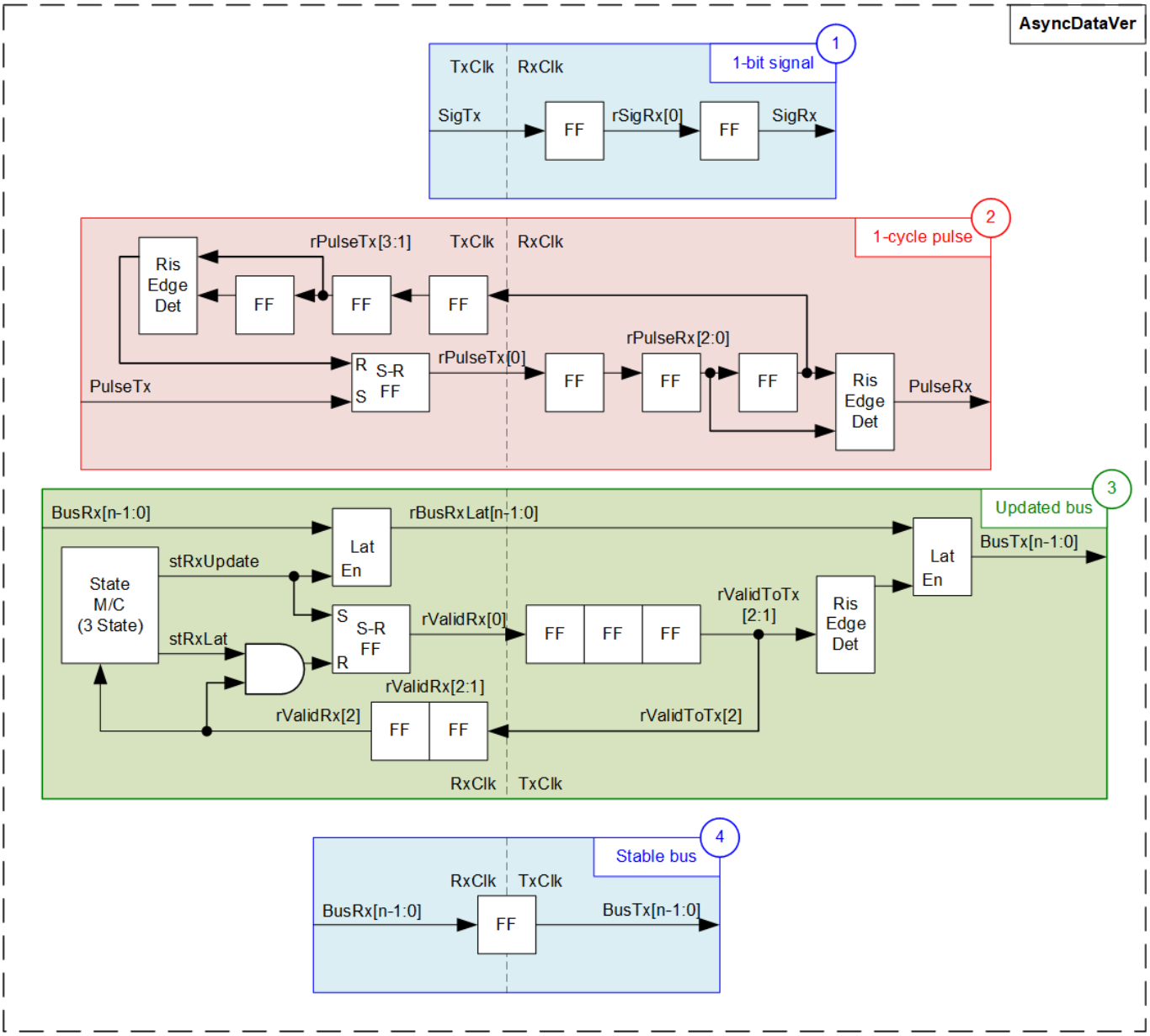

Figure 2-8 AsyncDataVer block

As shown in Figure 2-8, there are four signal types for converting clock domain, i.e., 1-bit signal, 1-cycle pulse signal, multiple-bit signal which can be updated, and multiple-bit signal which is stable value.

Block (1) shows two Flip-flops are inserted to solve metastability problem on SigTx signal which is run on input clock domain (TxClk). The result is SigRx signal which is applied on output clock domain (RxClk). When the signal is one-bit signal and the value is valid for several cycles, block (1) is applied. The signals of UserDataVer which use block (1) circuit are VerifyEn, RdFail, and RxErrorInt.

When the one-bit signal is a pulse which is asserted to '1' only one cycle, asynchronous circuit is designed as shown in block (2). First, S-R Flip flop is applied to convert the input signal (PulseTx) to be asserted to '1' for several clock cycles, assigned as rPulseTx[0]. After that, the signal is forwarded to the output clock domain (RxClk) by using three Flip-flops (rPulseRx[2:0]). The pulse on the output clock domain, PulseRx, is asserted to '1' for one cycle when rising edge of rPulseRx is found (rPulseRx[2:1]=01b). The signal on the output clock (rPulseRx[2]) is feedback to the input clock (TxClk), assigned as rPulseTx[3:1]. When detecting the rising edge of rPulseTx[3:2] which can refer that PulseRx was asserted, rPulseTx[0] is de-asserted to '0' to finish the operation. The signals of UserDataVer which use block (2) circuit are RxIntClr and TrnSizeClr.

When the signal is bus type which has multiple bits and the signal is always updated, asynchronous circuit is designed as shown in block (3). Data bus must be latched on input clock domain (RxClk) to hold the same value before transferring to the output clock domain (TxClk), assigned as rBusRxLat. There is state machine which has three states for controlling clock-crossing process, i.e., stRxUpdate which is the cycle to latch input signal, stRxLat which is the cycle to wait until the input signal is transferred to the output clock domain, and stRxWait which is the cycle to clear the signals on input clock domain to be Idle status. The state machine is run as forever loop to always update the signal from input clock domain to output clock domain.

When state machine is in stRxUpdate, the valid signal on input clock domain, rValidRx[0], is asserted to '1' on input clock domain to start data updating process. The valid signal is transferred to the output clock domain by using three Flip-flops, rValidToTx[2:0]. The signal is loaded to output clock domain, assigned as BusTx, when rising edge of valid signal (rValidToTx[2:1]) is detected. The valid signal on the output clock domain is feedback to input clock domain, assigned as rValidRx[2:1]. When rValidRx[2] is asserted to '1', it guarantees that BusTx was updated. Therefore, state machine changes to stRxWait which is designed to de-assert rValidRx and rValidToTx to '0'. After all valid signal are de-asserted, state machine returns to the first state, stRxUpdate, to restart the new update loop. The signal of UserDataVer which uses block (3) circuit is RxCurTrnSize.

The last block (4) is designed when multiple-bit signal is stable for long time before reading the value. One flip-flop is inserted to change clock synchronous from input clock domain (RxClk) to output clock domain (TxClk). The signals of UserDataVer which use block (4) circuit are RdFailPos, ExpPatt, and RdPatt.

2.5 Timer

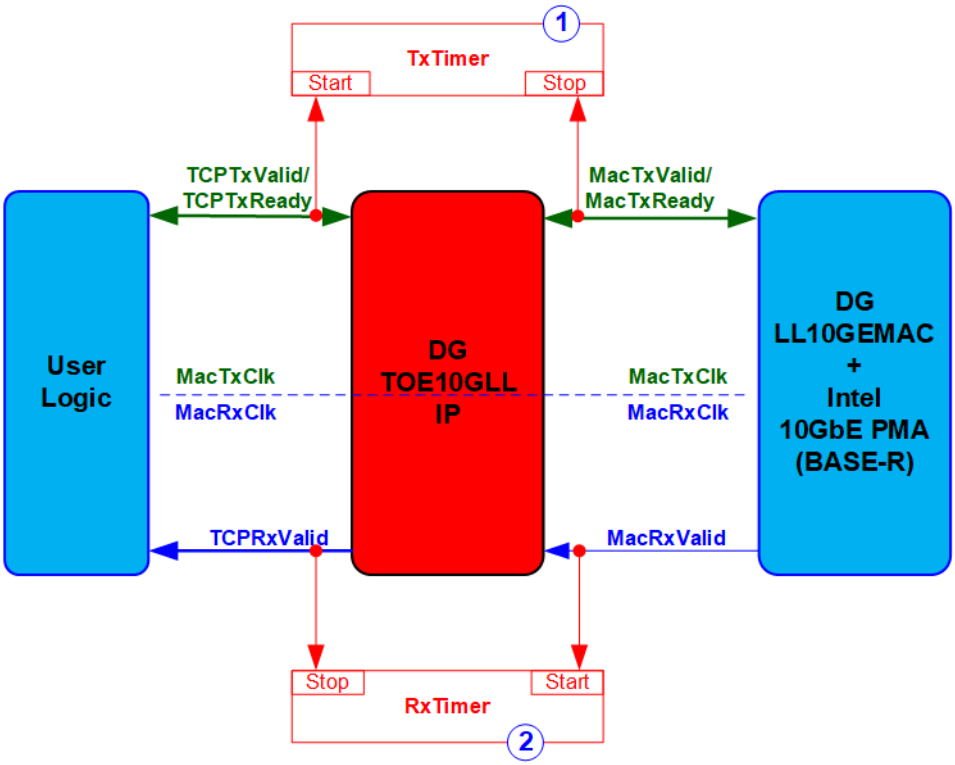

Figure 2-9 Timers in the reference design

To measure the latency time of TOE10GLL-IP for both Tx interface and Rx interface, two timers (TxTimer and RxTimer) are designed. Both timers are controlled by Start flag and Stop flag. Start flag is asserted to '1' when the first data is detected at the input of the measured module. Stop flag is asserted to '1' when the first data is detected at the output of the measured module. The timer latches the value for CPU reading to calculate the time in nanosecond unit for displaying on the console. As a result, the latency time is measured from the first packet only. Other packets are not be calculated. The timers are rerun after the user sends the new request to transfer the data. From above measurement, latency time of TOE10GLL-IP is measured by start-of-packet of one side and start-of-packet of another side.

3 CPU Firmware (FPGA)

In reference design, CPU firmware is implemented as bare-metal OS for easily handling with the hardware. After the test system is run, the first step in the firmware is hardware initialization.

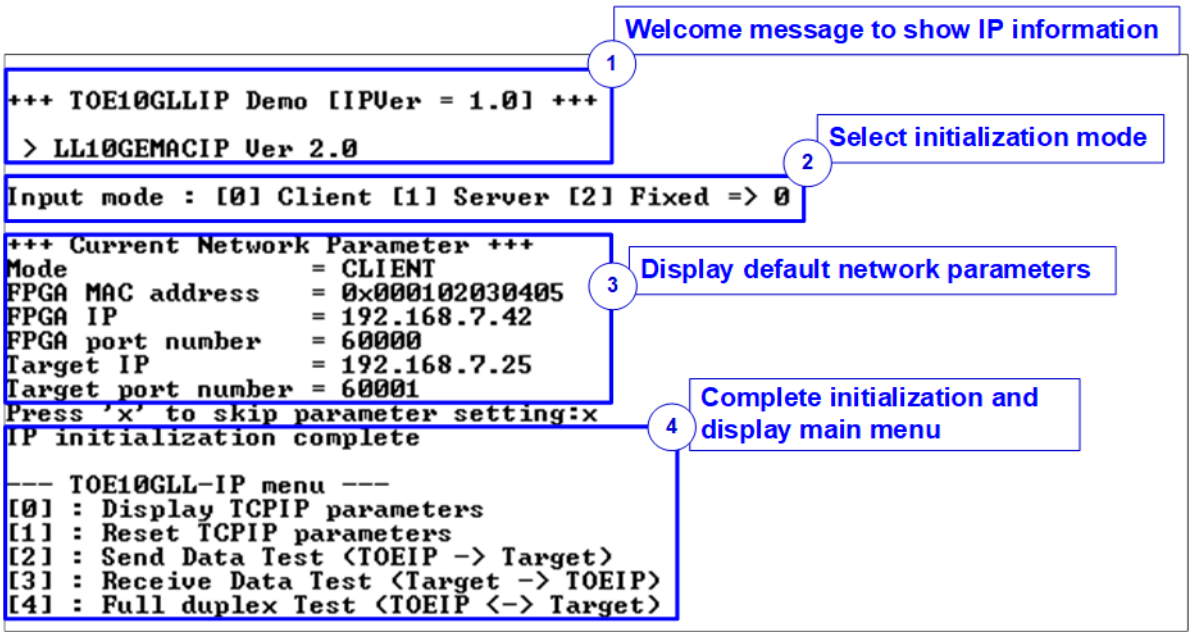

Figure 3-1 Message on the console during initialization process

As shown in Figure 3-1, there are four steps to initialize the hardware, described as follows.

1) After FPGA boot-up, 10G Ethernet link up status (EMAC_STS_REG[0]) is polling. The CPU waits until link up is detected and then displays welcome message to show IP information.
2) The menu to select the initialization mode of TOE10GLL-IP is displayed. The user can set as client, server, or fixed MAC mode.

Note:

- *When running in client mode, TOE10GLL-IP sends ARP request to get the MAC address of the target device from ARP reply. When running in server mode, TOE10GLL-IP waits until ARP request is received to decode MAC address and return ARP reply. When running fixed MAC mode, the user needs to know MAC address of the target device because TOE10GLL-IP does not transfer ARP packet.*
- *When running the test environment by using one FPGA board and Test PC, it is recommended to set FPGA run as client mode.*
- *When the test environment uses two FPGA boards, there are three solutions to initial the connection between two boards. First, one is client and another is server. Second, both are set to fixed MAC mode. Last, one is set to fixed MAC mode and another must be set to client.*

3) CPU displays default value of the network parameters, i.e., 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. First is the parameter set for server mode and another is the parameter set for client/fixed MAC mode. For fixed MAC mode, there is an extra parameter, Target MAC address. The user can select to complete the initialization process by using default parameters or updating some parameters. The details to change the parameter are described in Reset IP menu (topic 3.2).
4) CPU waits until the IP completes the initialization process by checking if InitFinish (TOE_STS_REG[0]) is equal to '1'. After that, "IP initialization complete" is displayed with the main menu. There are five test operations in the main menu. More details of each menu are described as follows.

3.1 Display parameters

This menu is designed to display the current value of all TOE10GLL-IP parameters.

The step to display parameters is as follows.
1) Read the initialization mode.
2) Read all network parameters from each variable in firmware following the initialization mode, i.e., 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 FPGA parameters set to TOE10GLL-IP while the target parameters are the parameters of TestPC or another FPGA.
3) Print out each variable.

3.2 Reset parameters

This menu is designed to change some parameters of TOE10GLL-IP such as IP address and source port number. After setting the updated parameters to TOE10GLL-IP, the CPU resets the IP to start re-initialization process by using new parameters. Finally, the CPU waits until the initialization is completed.

The step to reset parameters is as follows.
1) Display all parameters on the console.
2) Skip to the next step if the user uses the default value. Otherwise, the menu to set all parameters is displayed.
   a) Receive initialization mode from the user. If the initialization mode is changed, the latest parameter set of new mode is displayed on the console.
   b) Receive remaining parameters from user and validate all inputs. If the input is invalid, the parameter is not updated.
3) Force reset to IP by setting TOE_RST_REG[0]='1'.
4) Set all parameters to TOE10GLL-IP registers such as TOE_SML_REG and TOE_DIP_REG.
5) De-assert IP reset by setting TOE_RST_REG[0]='0' to start IP initialization process. Also, UserDataGen and UserDataVer module are reset.
6) Wait until InitFinish signal (TOE_STS_REG[0]) is asserted to '1' after finishing the initialization process.

3.3 Send data test

This menu is designed to test data sending. The user sets the parameters such as total transmit length and the open connection mode. If all inputs are valid, the port is opened. After that, 32-bit incremental test data is sent until all data is completely transferred. Finally, the port is closed in active mode.

The step to send the data is as follows.

1) Receive four parameters, i.e., transmit size, packet size, PSH flag mode, and open connection mode (active or passive) from user. After that, CPU validates all inputs. The operation is cancelled if some inputs are invalid.
2) Display recommended parameters of test application on PC by reading current parameters in the system.
3) Set UserReg parameters, i.e., transfer size (USRTX_TDL/TDH_REG), packet size (USRTX_PKL_REG), and PSH flag mode (USRTX_PSH_REG).
4) Send open connection command following the connection mode by setting TOE_CMD_REG=Active open or Passive open. After that, wait until TCPConnOn status (TOE_STS_REG[1]) is equal to '1'.
5) Set USRTX_CMD_REG=Send data to start sending data process by UserDataGen module. After that, wait until total transmit data size, read by USRTX_LENL/H_REG, is equal to set value (USRTX_TDL/H_REG). During transferring data, current number of transmitted data (USRTX_LENL/H_REG) is displayed on console every second.
6) Set close connection command to TOE10GLL-IP register (TOE_CMD_REG=Active close). After that, wait until TCPConnOn status (TOE_STS_REG[1]) is equal to '0'.
7) Calculate performance and latency time and then display a test result on the console.

3.4 Receive data test

This menu is designed to test data receiving. The user sets the parameters such as total receive length and the open connection mode. If all inputs are valid, the port is opened. After that, 32-bit incremental test data is created for verifying with the received data from PC/FPGA when the data verification is enabled.

The step to receive the data is as follows.

1) Receive three parameters, i.e., total transfer data size, data verification mode, and connection mode (active or passive) from user input. The operation is cancelled if some inputs are invalid.
2) Display recommended parameters of test application on PC by reading current parameters in the system.
3) Set UserReg parameters, data verification mode (USRRX_CMD_REG). Total transfer data size is stored as the variable for comparison when finishing the test.
4) Send open connection command following the connection mode by setting TOE_CMD_REG=Active open or Passive open. After that, wait until TCPConnOn status (TOE_STS_REG[1]) is equal to ‘1’.
5) Wait until TCPConnOn status (TOE_STS_REG[1]) is equal to ‘0’ after the target device (PC or another FPGA) closes the port when completing transferring all data. During transferring data, current number of received data (USRRX_LENL/H_REG) is displayed on console every second.
6) Compare total receive data length (USRRX_LENL/H_REG) with the variable, total size set from the user. If they are not the same value, the warning message is displayed. Next, CPU checks verification result by reading USRRX_STS_REG[8] (‘0’: normal, ‘1’: error) when user enables the verification. If the error is detected, the error message is displayed.
7) Calculate performance and latency time and then display a test result on the console.

3.5 Full duplex test

This menu is designed to run full duplex test by transferring data in both directions by using the same port number at the same time. The menu receives the user parameters for running the test such as total transfer length and packet size. If all inputs are valid, the port is opened. After that, the data starts transferring. When finishing data transferring in both directions, the port is closed by the client which may be FPGA or PC. The test is run as forever loop until the user cancels the operation.

Note: When running the test with PC, connection mode on FPGA must be set to passive (server operation). The transfer size on the test application, tcp_client_txrx_40G, must be equal to the transfer size set on FPGA. To stop the test operation, user presses some keys on FPGA console and then enters "Ctrl+C" on PC console consequently.

The step to run full duplex is as follows.

1) Receive five parameters, i.e., total transfer size (the same size for both directions), packet size, PSH flag mode, data verification mode, and open connection mode (active or passive) from user. The operation is cancelled if some inputs are invalid.
2) Display the recommended parameters of test application run on PC by reading current parameters in the system.
3) Send open connection command following the connection mode by setting TOE_CMD_REG=Active open or Passive open. After that, wait until TCPConnOn status (TOE_STS_REG[1]) is equal to '1'.
4) Set UserReg parameters, i.e., data verification mode (USRRX_CMD_REG), total transfer size (USRTX_TDL/TDH_REG), packet size (USRTX_PKL_REG), PSH flag mode (USRTX_PSH_REG), and command register of UserDataGen (USRTX_CMD_REG) to start the test operation.
5) Wait until finishing transferring all data in both directions by comparing total transmit data size and total receive data size. Total transmit data size (USRTX_LENL/H_REG) must be equal to set value (USRTX_TDL/H_REG). Also, when running active connection mode, total receive data size (USRRX_LENL/H_REG) must be equal to set value. During transferring data, current number of transmitted data and number of received data are displayed on the console every second.
6) Set close connection following connection mode value.
   a. For active close, CPU sets close command to TOE10GLL-IP register (TOE_CMD_REG=Active close). After that, CPU waits until TCPConnOn status (TOE_STS_REG[1]) is equal to '0'.
   b. For passive close, CPU waits until TCPConnOn status (TOE_STS_REG[1]) is equal to '0'.
7) Check the result and the error (similar to Step 6 of Receive data test).
8) Calculate performance and latency time and then display a test result on the console. Return to Step 3 to repeat the test in forever loop.

3.6 Function list in User application

This topic describes the function list to run TOE10GLL-IP operation.

void clr_trns_status(void)	
Parameters	None
Return value	None
Description	Set USRTX_CMD_REG[1] and USRRX_CMD_REG[1] to '1' to clear USRTX_LENL/H_REG, USRTX_TMR_REG, USRRX_LENL/H_REG, and USRRX_TMR_REG.

void exec_port(unsigned int port_ctl, unsigned int mode_active)	
Parameters	port_ctl: 1-Open port, 0-Close port mode_active: 1-Active open/close, 0-Passive open/close
Return value	None
Description	Write TOE_CMD_REG to open (active or passive) or close (active) connection. After that, call read_conon function to monitor connection status until it changes from ON to OFF or OFF to ON when running close port or open port respectively.

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

int input_param(void)	
Parameters	None
Return value	0: Valid input, -1: Invalid input
Description	Receive network parameters from user, i.e., Initialization mode, FPGA MAC address, FPGA IP address, FPGA port number, Target MAC address (when running fixed MAC mode), Target IP address, and Target port number. If the input is valid, the parameter is updated. Otherwise, the value does not change. After receiving all parameters, calling show_param function do display parameters.

unsigned int read_conon(void)	
Parameters	None
Return value	0: Connection is OFF, 1: Connection is ON.
Description	Read value from TOE_STS_REG register and return only bit1 value to show connection status.

void show_cursize(void)	
Parameters	None
Return value	None
Description	Read current number of transmitted data and number of received data from global parameters (tx_cursize and rx_cursize) and then display in byte, Kbyte, or Mbyte unit

void show_ipstate(void)	
Parameters	None
Return value	None
Description	Read current state value from TOE_STS_REG[20:16] and then display the result

void show_latency(void)	
Parameters	None
Return value	None
Description	Read USRTX_TMR_REG and USRRX_TMR_REG which are latency time of transmit interface and receive interface of TOE10GLL-IP. After that, convert the unit from clock cycle to be the time in nsec. Finally, print out the value.

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

void show_recv_error(void)	
Parameters	None
Return value	None
Description	This function is called when error is found during running receive data test. Read USRRX_ERR_REG and two errors are decoded, i.e., TCP checksum and EMAC error.

void show_result(void)	
Parameters	None
Return value	None
Description	Read total transmit data size and total receive data size from global parameters (tx_cursize and rx_cursize) and display the results. Read total time usage from global parameters (timer_val) and calculate total time usage to display in usec, msec, or sec unit. Finally, transfer performance is calculated and displayed in MB/s unit.

void show_verfail(void)	
Parameters	None
Return value	None
Description	This function is called when data verification is failed. Read USRRX_FAILPOSL/H_REG (failure position), USRRX_EXPPAT_REG (expected data), and USRRX_RDPAT_REG (received data) and print out the result on the console to show the details of the first failure data.

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 clr_trns_status, exec_port, read_conon, update_cursize, show_cursize, show_recv_error, show_verfail, show_result, and show_latency 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 clr_trn_status, exec_port, read_conon, update_cursize, show_cursize, show_result, and show latency 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 clr_trns_status, exec_port, read_conon, update_cursize, show_cursize, show_recv_error, show_verfail, show_result, and show_latency function.

void update_cursize(void)	
Parameters	None
Return value	None
Description	Read USRTX_LENL/H_REG and USRRX_LENL/H_REG and then update to global parameters (tx_cursize and rx_cursize).

void wait_ethlink(void)	
Parameters	None
Return value	None
Description	Read EMAC_STS_REG[0] and wait until ethernet connection is linked up

4 Test Software on PC

4.1 "tcpdatatest" for half duplex test

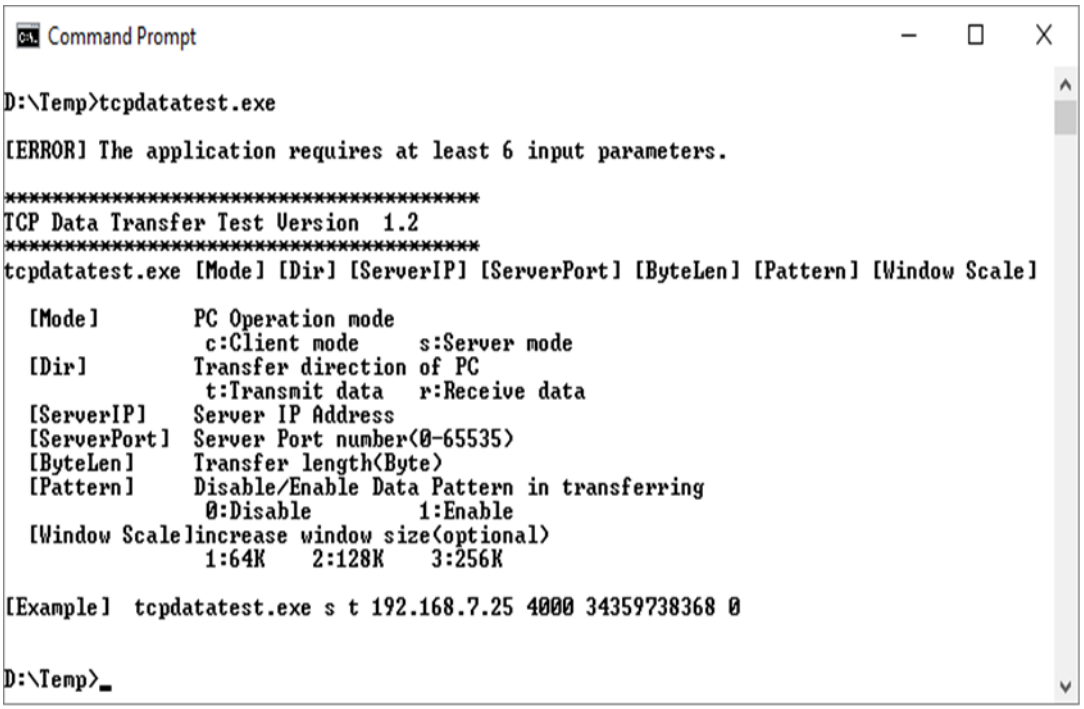

Figure 4-1 "tcpdatatest" application usage

"tcpdatatest" is designed to run on PC for sending or receiving TCP data via Ethernet as Server or Client mode. It is recommended to run PC in the demo by using Client mode. There are six parameters which must be set by the user before running the application, described as follows.

1) Mode: c –PC runs in Client mode and FPGA runs in Server mode (recommended)
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 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 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 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 number of sent data from the function.
7) Calculate remaining transfer size.
8) Print current transmit data size every second.
9) Repeat step 5) – 8) until the remaining transfer size is 0.
10) Calculate transfer 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 calculate total receive size.
4) This step is skipped if data verification is disabled. Otherwise, received data is verified by the incremental pattern. Error message is printed out when data is not correct.
5) Print current receive data size every second.
6) Repeat step 3) – 5) until the connection is closed.
7) Calculate transfer performance and print the result on the console.
8) Close socket and free the memory.

4.2 “tcp_client_txrx_40G” for full duplex test

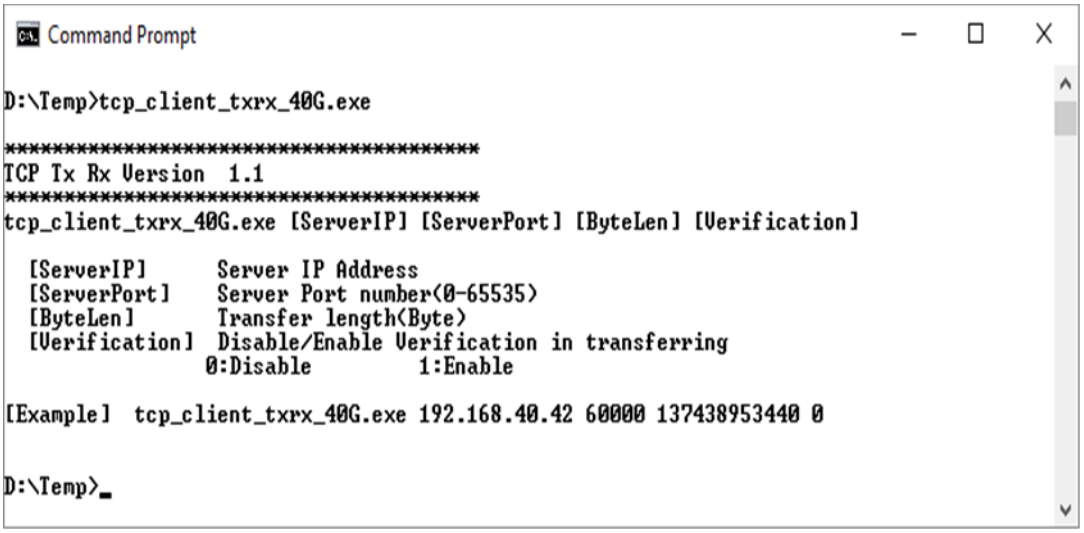

Figure 4-2 “tcp_client_txrx_40G” application usage

“tcp_client_txrx_40G” application is designed to run on PC for sending and receiving TCP data through Ethernet by using the same port number at the same time. The application is run in Client mode. Therefore, the server parameters are the network parameters of TOE10GLL-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 number of transmitted data and received data. This value must be equal to the transfer size set on FPGA for running full-duplex test.
4) Verification :
   0 – Generate dummy data for sending function and disable data verification for receiving function. When running this mode, it shows 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 sent data from the function, and calculate remaining send size.
(7) Read data from the receive buffer and calculate total receive data size.
(8) Skip this step if data verification is disabled. Otherwise, data is verified by incremental pattern. Error message is printed out when data is not correct.
(9) Print current send data size and current receive data size every second.
(10) Repeat step 5) – 9) until total number of send size and total number of receive size are equal to ByteLen, set by user.
(11) Calculate transfer performance and print the result on the console.
(12) Close the socket.
(13) Sleep for 1 second to wait until the hardware completes the current test loop.
(14) Repeat step 3) – 13) in forever loop. If verification is failed, the application is stopped.

5 Revision History

Revision	Date	Description
1.0	27-Apr-21	Initial version release
1.1	5-May-21	Update clock domain name