TOE40G-IP reference design

Rev1.0 24-Dec-18

1 Introduction

TCP/IP is the core protocols of the Internet Protocol Suite for networking application. TCP/IP model has four layers, i.e. Application Layer, Transport Layer, Internet Layer, and Network Access Layer. In Figure 1-1, five layers are displayed for simple matching with hardware implementation by FPGA. Network Access Layer is split into Link and Physical Layer.

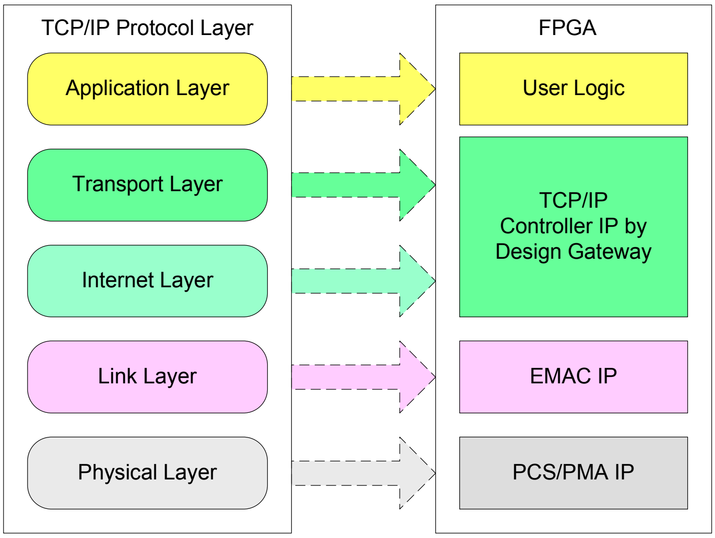


Figure 1-1 TCP/IP Protocol Layer

TOE40G-IP implements Transport and Internet layer of TCP/IP Protocol. To send data, TOE40G-IP prepares TCP data from user logic, adds TCP/IP header to generate Ethernet packet, and sends to EMAC. To receive data, TOE40G-IP extracts TCP data and header from Ethernet packet. If TCP/IP header in the packet is valid, TCP data will be stored to the buffer for user logic reading.

The lower layer protocols are implemented by 40 Ethernet and PHY IP from Intel FPGA.

The reference design provides evaluation system which includes simple user logic to send and receive data by using TOE40G-IP. For user interface, CPU system is designed to interface with user through JTAG UART. The firmware is designed as bare-metal OS. Two test applications are applied in the demo, i.e. “tcpdatatest” for half-duplex test and “tcp_client_txrx_40G” for full-duplex test. The reference design is available on Intel FPGA development board to show ultra high-speed transfer with network reliability. More details of the demo are described as follows.

2 Hardware overview

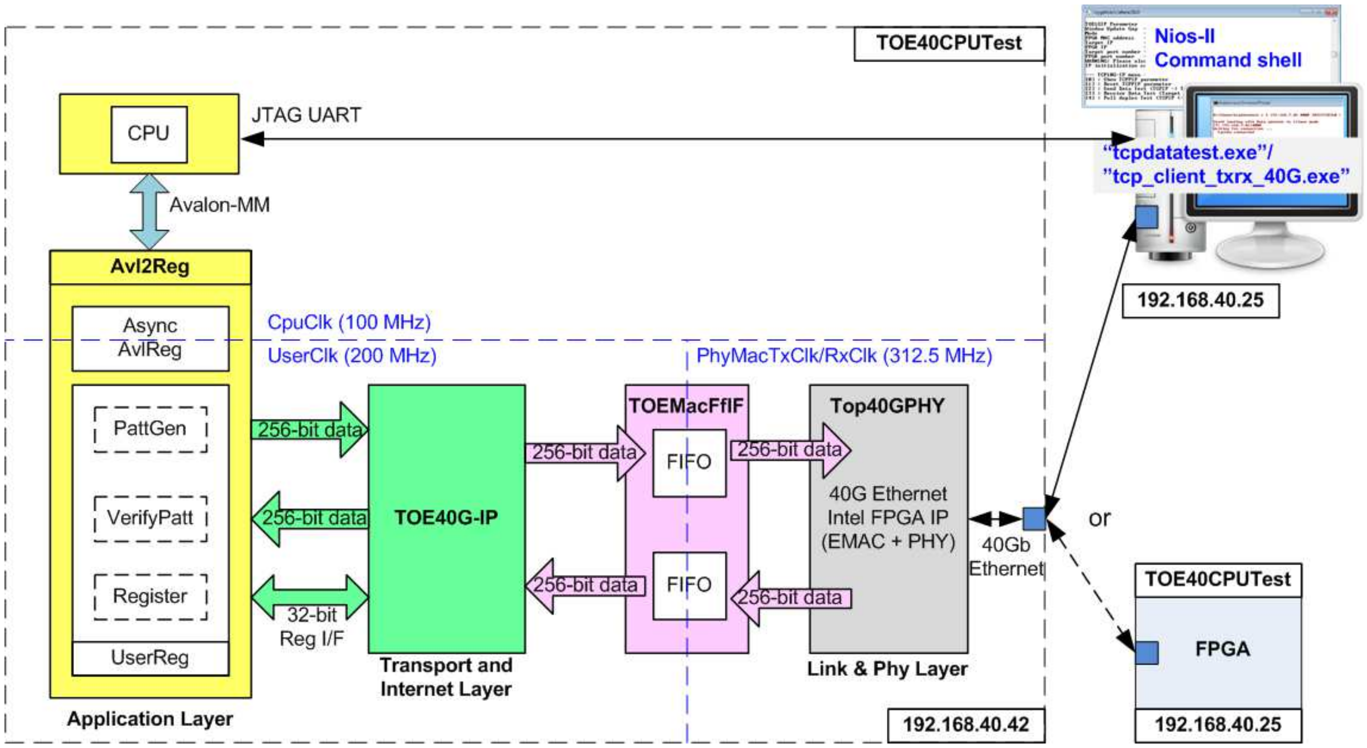


Figure 2-1 Demo block diagram

In test environment, two devices are used for 40Gb Ethernet transferring. First device runs in client mode and another runs in server mode. To confirm TOE40G-IP operation in both modes, the demo can be tested by using two test environments. First environment uses two FPGAs (one is client and another is server). Second environment uses one FPGA and one PC, as shown in Figure 2-1.

In FPGA logic, TOE40G-IP and 40G Ethernet MAC and PHY (Intel FPGA IP) are applied to complete all TCP/IP layer implementation. TOEMacFfIF module is used to convert 256-bit FIFO interface of TOE40G-IP to be 40G EMAC interface. User interface of TOE40G-IP is connected to UserReg module within Avl2Reg.

For user data interface, UserReg includes PattGen to generate test pattern to TOE40G-IP. Also, VerifyPatt is designed to verify received data from TOE40G-IP. Test pattern in the reference design is 32-bit increment data.

For user control interface, there are registers in UserReg to store test parameters from user such as transfer length and transfer direction. Input parameters are received from user through JTAG UART. CPU firmware validates all parameters from user before forwarding the set value to hardware through Avalon-MM bus.

Due to the fact that CPU system and TOE40G-IP run in different clock domain, AsyncAvlReg module inside Avl2Reg is designed as asynchronous circuit to support clock-crossing operation. Also, AsyncAvlReg converts Avalon-MM bus signal which is standard bus in CPU system to be register interface.

In CPU system, two CPU peripherals are used, i.e. JTAG UART for user interface and Timer for measuring transfer performance. CPU detects Avl2Reg module as CPU peripheral like Timer and JTAG UART.

Two applications on PC are applied in the demo. The first application is "tcpdatatest.exe" which is designed to send or receive Ethernet data with TOE40G-IP. Data transferring is half-duplex mode. The second application is "tcp_client_txrx_40G.exe" which is designed to send and receive Ethernet data by using same TCP port number at the same time (full-duplex mode). In full-duplex mode, PattGen and VerifyPatt module inside UserReg transfer data with TOE40G-IP in both directions at the same time.

2.1 Low Latency 40G Ethernet Intel FPGA IP

This is Intel FPGA IP core which provides 40 Gb Ethernet MAC integrated with PHY layer. Data path interface is defined as 256-bit Avalon-Stream. More details of Low Latency 40G Ethernet Intel FPGA IP are described in following link.
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_ll_40gbe.pdf

2.2 TOE40G-IP

TOE40G-IP implements TCP/IP stack and offload engine. Control and status signals for user interface are accessed through register interface. Data interface is accessed through FIFO interface. More details are described in datasheet.
http://www.dgway.com/products/IP/TOE40G-IP/dg_toe40gip_data_sheet_intel_en.pdf

2.3 TOEMacFfIF

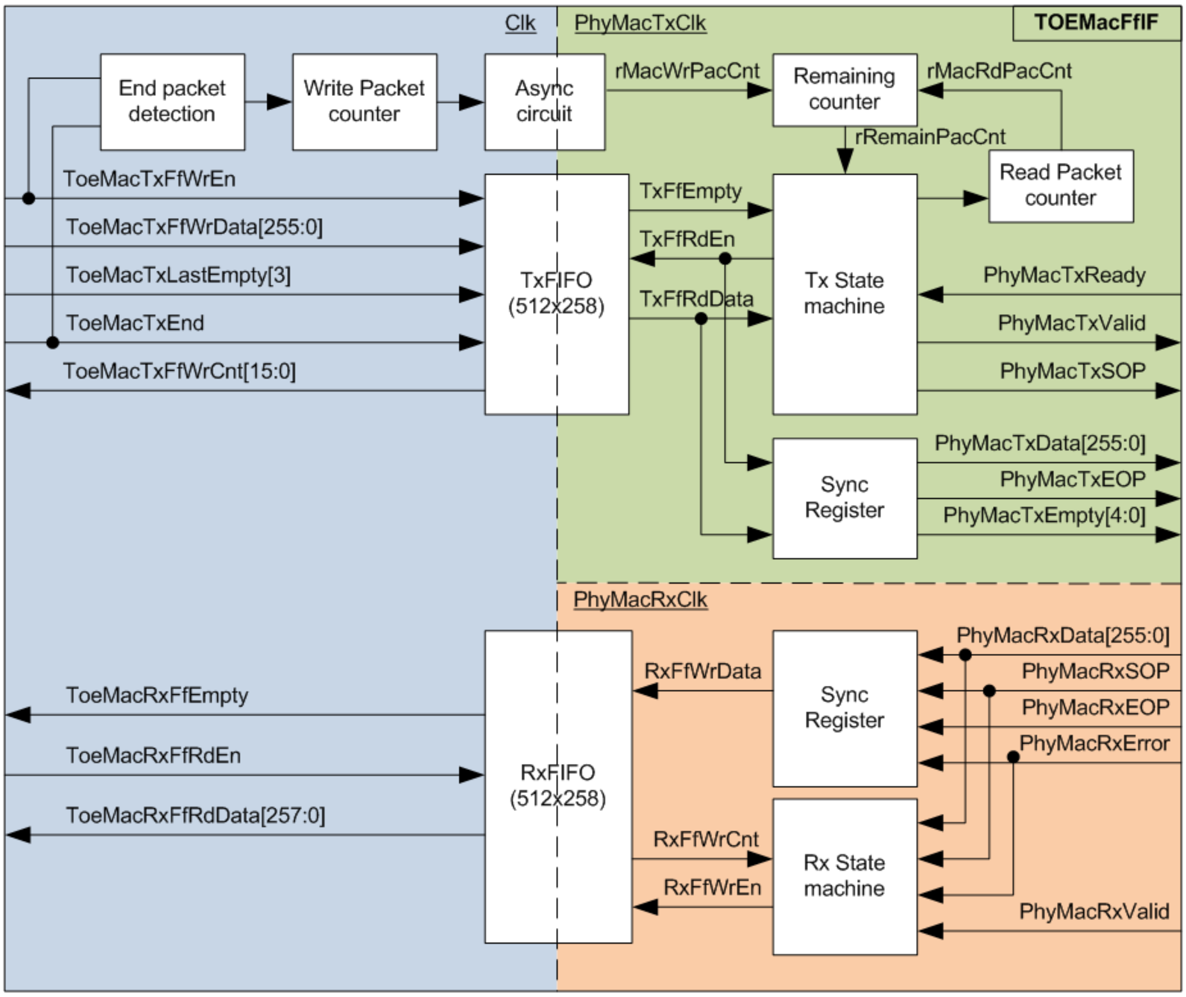


Figure 2-2 TOEMacFfIF block diagram

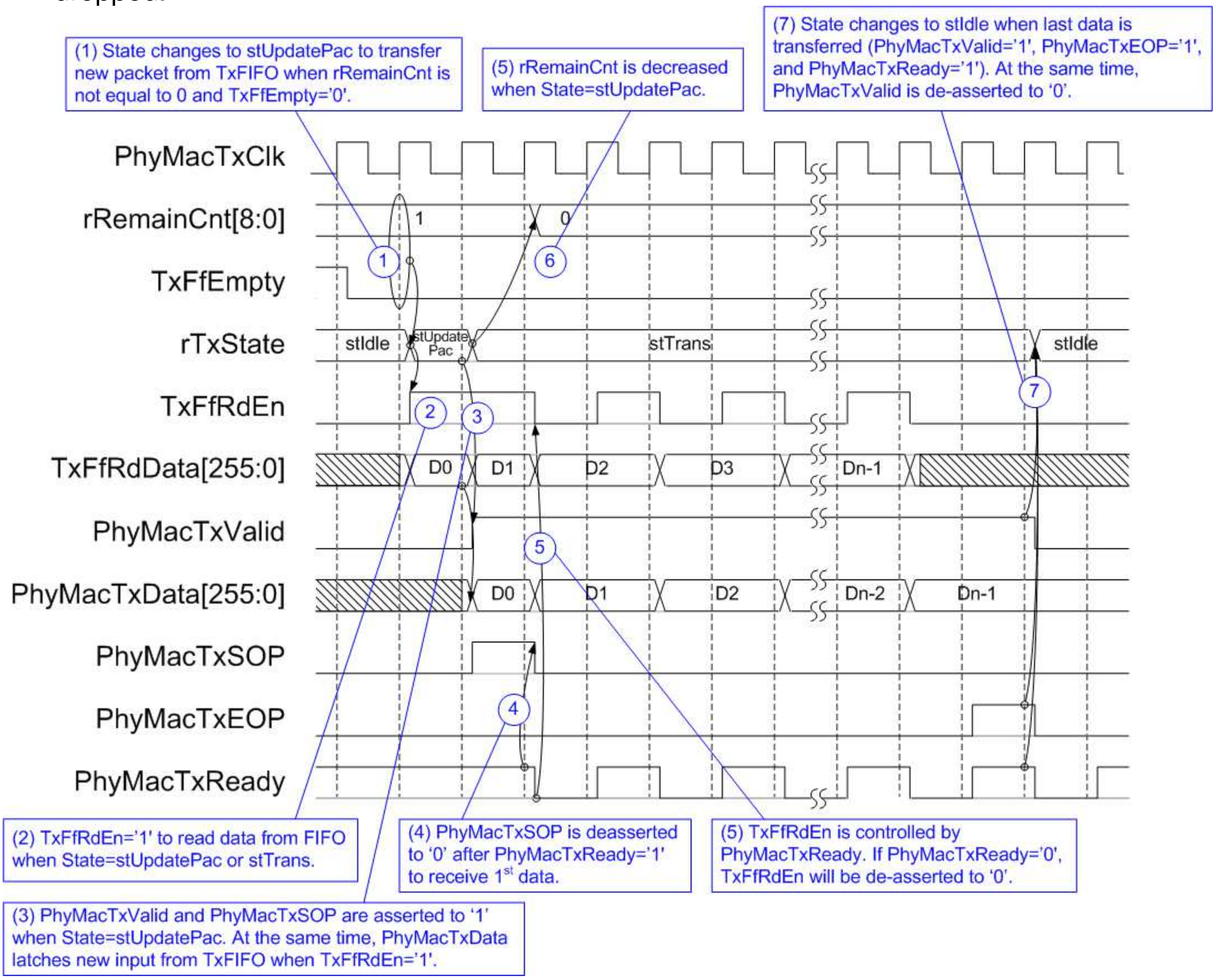
This module is designed to be adapter logic connecting between TOE40G-IP and 40G Ethernet MAC. There are three clock domains in this module, i.e. Clk which is synchronous to TOE40G-IP, PhyMacTxClk which is synchronous to Tx interface of 40G EMAC, and PhyMacRxClk which is synchronous to Rx interface of 40G EMAC. The interface type of TOE40G-IP for both Tx and Rx interface is FIFO while the interface type of 40G EMAC is Avalon-stream. So, the logic inside TOEMacFfIF is designed to convert interface type and includes asynchronous circuit.

Two asynchronous FIFO are applied to store Tx/Rx packet transferring between TOE40G-IP and 40G EMAC. FIFO size is 512x258-bit which is much enough to store 9K byte packet (maximum TCP packet size supported on 40G Ethernet card). The logic is split into two parts, i.e. Tx interface logic and Rx interface logic.

To transfer data from TOE40G-IP to 40G EMAC (Tx interface), write interface of TxFIFO is connected to TOE40G-IP directly. There is the logic to count total Tx packet inside TOEMacFfIF. The counter is increased when end of Tx packet is detected. Tx packet counter is forwarded to asynchronous circuit to run on PhyMacTxClk domain. Read interface of TxFIFO is controlled by Tx state machine. If at least one Tx packet is ready inside TxFIFO, Tx state machine will forward data from TxFIFO to 40G EMAC following Avalon-stream standard.

To check total packet inside TxFIFO, Packet counter to count total read packet from TxFIFO is designed. This counter is increased when Tx State machine forwards 1st data of each packet from TxFIFO to 40G EMAC. Remaining packet counter is calculated by Write packet counter (rMacWrPacCnt) – Read packet counter (rMacRdPacCnt).
To reduce data size of TxFIFO, only bit 3 of TxLastEmpty signal from TOE40G-IP is applied because TxLastEmpty could be equal to two values, i.e. 0x04 and 0x0A.

To receive packet from 40G EMAC, RxFIFO is used to store received packet. Read interface of RxFIFO is connected to TOE40G-IP directly. Rx state machine checks free space of RxFIFO by monitoring RxFfWrCnt. If free space is more than 9 Kbyte, the received packet from 40G EMAC will be forwarded to RxFIFO. Otherwise, the received packet will be dropped.



(1) State changes to stUpdatePac to transfer new packet from TxFIFO when rRemainCnt is not equal to 0 and TxFfEmpty='0'.

(5) rRemainCnt is decreased when State=stUpdatePac.

(7) State changes to stIdle when last data is transferred (PhyMacTxValid='1', PhyMacTxEOP='1', and PhyMacTxReady='1'). At the same time, PhyMacTxValid is de-asserted to '0'.

(2) TxFfRdEn='1' to read data from FIFO when State=stUpdatePac or stTrans.

(4) PhyMacTxSOP is deasserted to '0' after PhyMacTxReady='1' to receive 1st data.

(5) TxFfRdEn is controlled by PhyMacTxReady. If PhyMacTxReady='0', TxFfRdEn will be de-asserted to '0'.

(3) PhyMacTxValid and PhyMacTxSOP are asserted to '1' when State=stUpdatePac. At the same time, PhyMacTxData latches new input from TxFIFO when TxFfRdEn='1'.

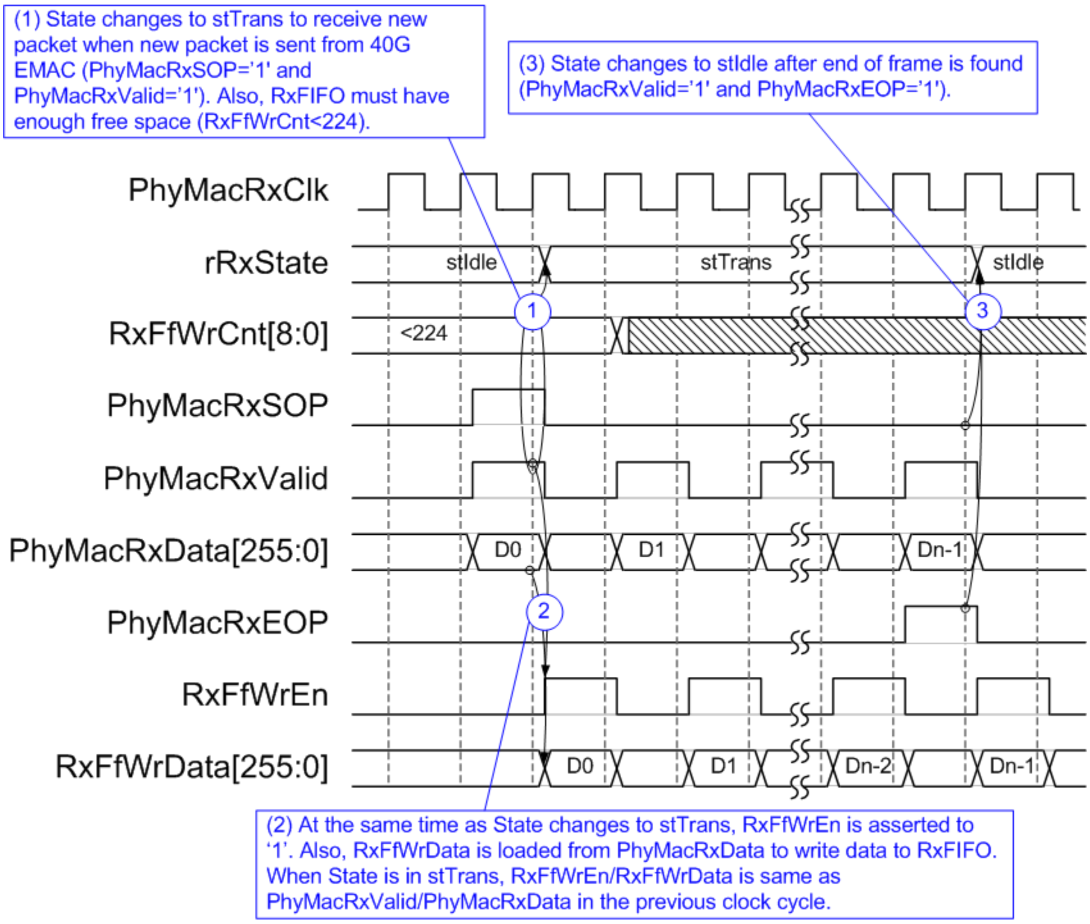
Figure 2-3 PhyMacTx interface timing diagram

Timing diagram of the logic to read data from TxFIFO and forward to 40G EMAC is shown in Figure 2-3. When at least one packet is available in TxFIFO (rRemainCnt is not equal to 0) and TxFfEmpty is equal to ‘0’ (Read interface of TxFIFO is ready), TxState forwards one packet from TxFIFO to 40G EMAC by changing state from stIdle to stUpdatePac. stUpdatePac is run only one clock cycle to decrease rRemainCnt signal. In the next clock, TxState changes to stTrans.

TxFfRdEn is asserted to ‘1’ to read data from TxFIFO when state is not equal to stIdle and 40G EMAC is ready to receive data (PhyMacTxReady=’1’). TxFfRdEn is also used to load data output of TxFIFO to latch as PhyMacTxData signal (data output to 40G EMAC). FIFO using in this module is Show-ahead type (Read data is received at the same clock as Read enable asserting to ‘1’).

At the same time, PhyMacTxValid is asserted to ‘1’ to send valid data to 40G EMAC. PhyMacTxValid is asserted to ‘1’ until end of frame. PhyMacTxSOP is asserted to ‘1’ with the 1st valid data of the packet on PhyMacTxData. After PhyMacTxReady=’1’, PhyMacTxSOP is de-asserted to ‘0’ in the next clock.

During packet transferring, PhyMacTxReady could be de-asserted to ‘0’ to pause data transmission. When 40G EMAC is not ready, TxFfRdEn is de-asserted to ‘0’ to hold same data on PhyMacTxData signal. After end of frame is transferred (PhyMacTxEOP=’1’), state changes from stTrans to stIdle. At the same time, PhyMacTxValid is de-asserted to ‘0’ to complete transferring one packet.



(1) State changes to stTrans to receive new packet when new packet is sent from 40G EMAC (PhyMacRxSOP='1' and PhyMacRxValid='1'). Also, RxFIFO must have enough free space (RxFfWrCnt<224).

(3) State changes to stIdle after end of frame is found (PhyMacRxValid='1' and PhyMacRxEOP='1').

(2) At the same time as State changes to stTrans, RxFfWrEn is asserted to '1'. Also, RxFfWrData is loaded from PhyMacRxData to write data to RxFIFO. When State is in stTrans, RxFfWrEn/RxFfWrData is same as PhyMacRxValid/PhyMacRxData in the previous clock cycle.

Figure 2-4 PhyMacRx interface timing diagram when RxFIFO is ready

When new packet is received from 40G EMAC, Rx state machine monitors RxFfWrCnt value. RxFIFO must have free space enough to store 9Kbyte packet (maximum TCP packet size supported on 40G Ethernet card). So, 224 is minimum value to confirm that at least 9Kbyte size is free in RxFIFO (Free space = (511 – 224) x 32 = 9184 byte). If new packet is received and FIFO has free space enough, Rx state machine will change to stTrans to forward packet from 40G EMAC to RxFIFO.

The value of RxFfWrEn is same as PhyMacRxValid with 1-clock latency. To synchronous with RxFfWrEn, RxFfWrData is also generated by adding one Flip Flop to PhyMacRxData to add 1-clock latency. When end of packet is found (PhyMacRxValid='1' and PhyMacRxEOP='1'), Rx state machine changes to stIdle.

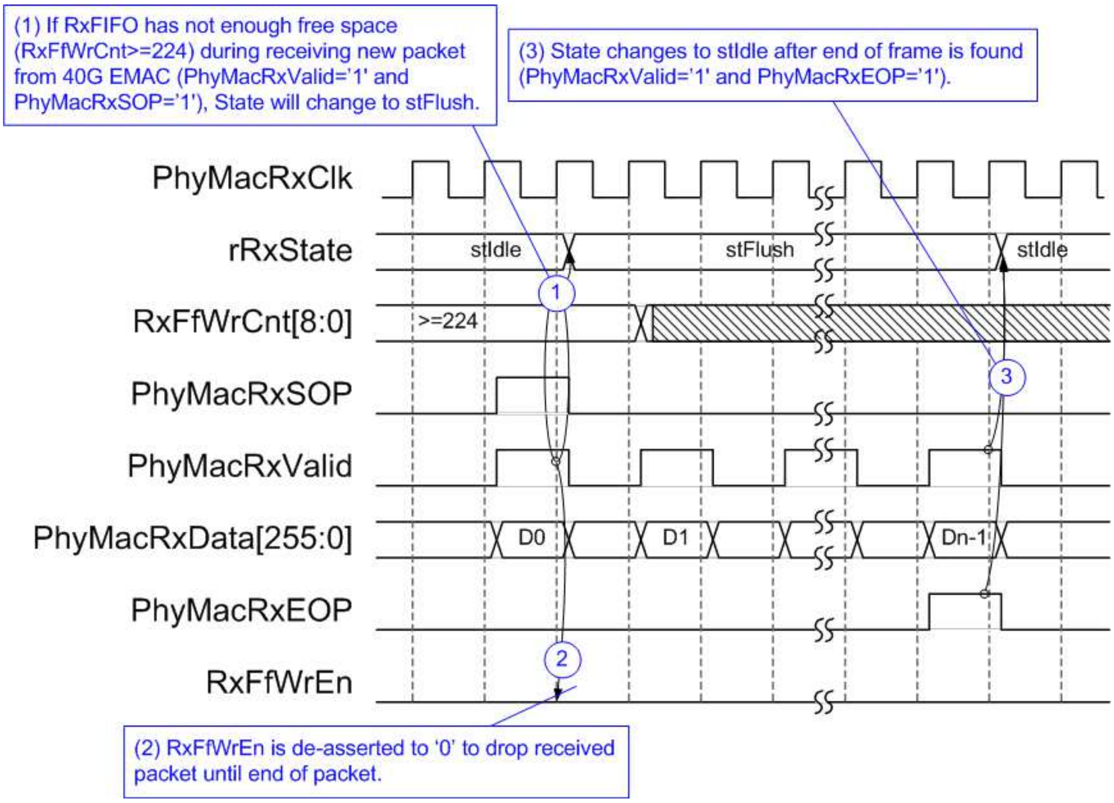

(1) If RxFIFO has not enough free space (RxFfWrCnt>=224) during receiving new packet from 40G EMAC (PhyMacRxValid='1' and PhyMacRxSOP='1'), State will change to stFlush.
(3) State changes to stIdle after end of frame is found (PhyMacRxValid='1' and PhyMacRxEOP='1').
PhyMacRxClk
rRxState
stIdle
stFlush
stIdle
RxFfWrCnt[8:0]
>=224
PhyMacRxSOP
PhyMacRxValid
PhyMacRxData[255:0]
D0
D1
Dn-1
PhyMacRxEOP
RxFfWrEn
1
2
3
(2) RxFfWrEn is de-asserted to '0' to drop received packet until end of packet.

Figure 2-5 PhyMacRx interface timing diagram when RxFIFO is not ready

Rx state machine changes to stFlush to drop received packet from 40G EMAC when new packet is received during RxFIFO full (RxFfWrCnt is more than or equal to 224). During receiving the packet, RxFfWrEn is de-asserted to ‘0’ until end of the packet. The dropped packet is not stored to RxFIFO. Rx state machine returns to stIdle after end of frame is detected (PhyMacRxValid=’1’ and PhyMacRxEOP=’1’).

2.4 Avl2Reg

The hardware is connected to CPU through Avalon-MM bus, similar to other CPU peripherals. The hardware registers are mapped to CPU memory address, as shown in Table 2-1. The control and status registers for CPU access are designed in Avl2Reg.

Avl2Reg connects to TOE40G-IP for both data path and control path. As shown in Figure 2-6, there are two clock domains applied in this block, i.e. CpuClk (100 MHz) which is used to interface with CPU through Avalon-MM bus and UserClk (200 MHz) which is user clock domain for TOE40G-IP.

AsyncAvlReg includes asynchronous circuit between CpuClk and UserClk. More details of each hardware are described as follows.

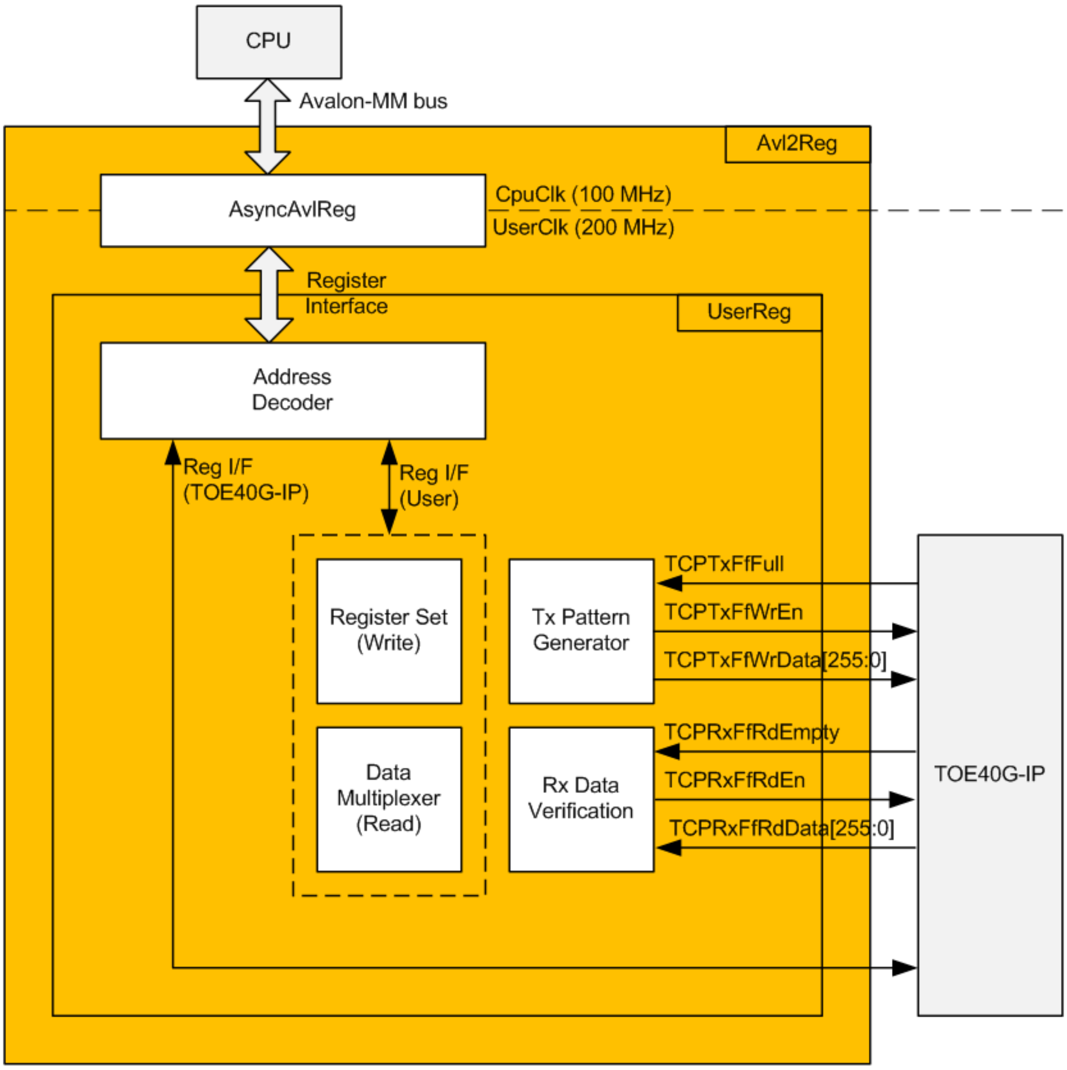


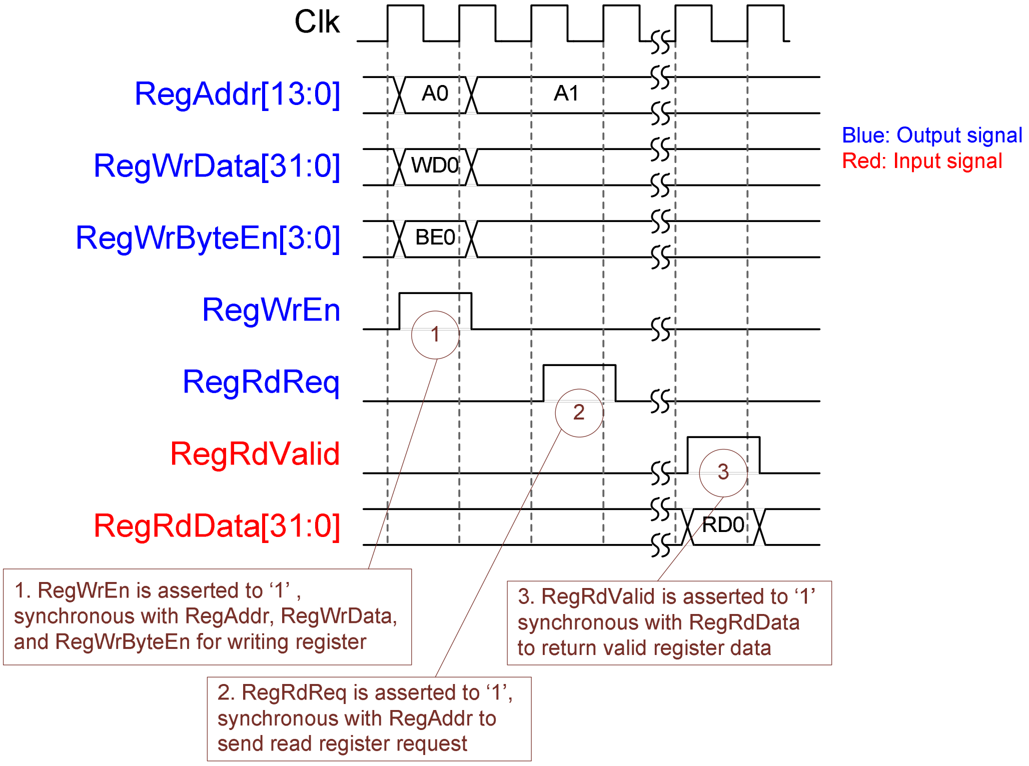
Figure 2-6 Avl2Reg block diagram

2.4.1 AsyncAvlReg

This module is designed to convert signal interface between Avalon-MM and register interface. Also, it supports to convert clock domain between CpuClk (100 MHz) and UserClk (200 MHz). Timing diagram of register interface is shown in Figure 2-7.

To write register, timing diagram is same as 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 byte enable of this access: bit[0] is write enable for RegWrData[7:0], bit[1] is used for RegWrData[15:8], …, and bit[3] is used for RegWrData[31:24]).

To read register, AsyncAvlReg asserts RegRdReq=’1’ with the valid value of RegAddr (the register address in 32-bit unit). After that, the module waits until RegRdValid is asserted to ‘1’ to get the read data through RegRdData signal at the same clock.



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

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

3. RegRdValid is asserted to ‘1’ synchronous with RegRdData to return valid register data

Figure 2-7 Register interface timing diagram

2.4.2 UserReg

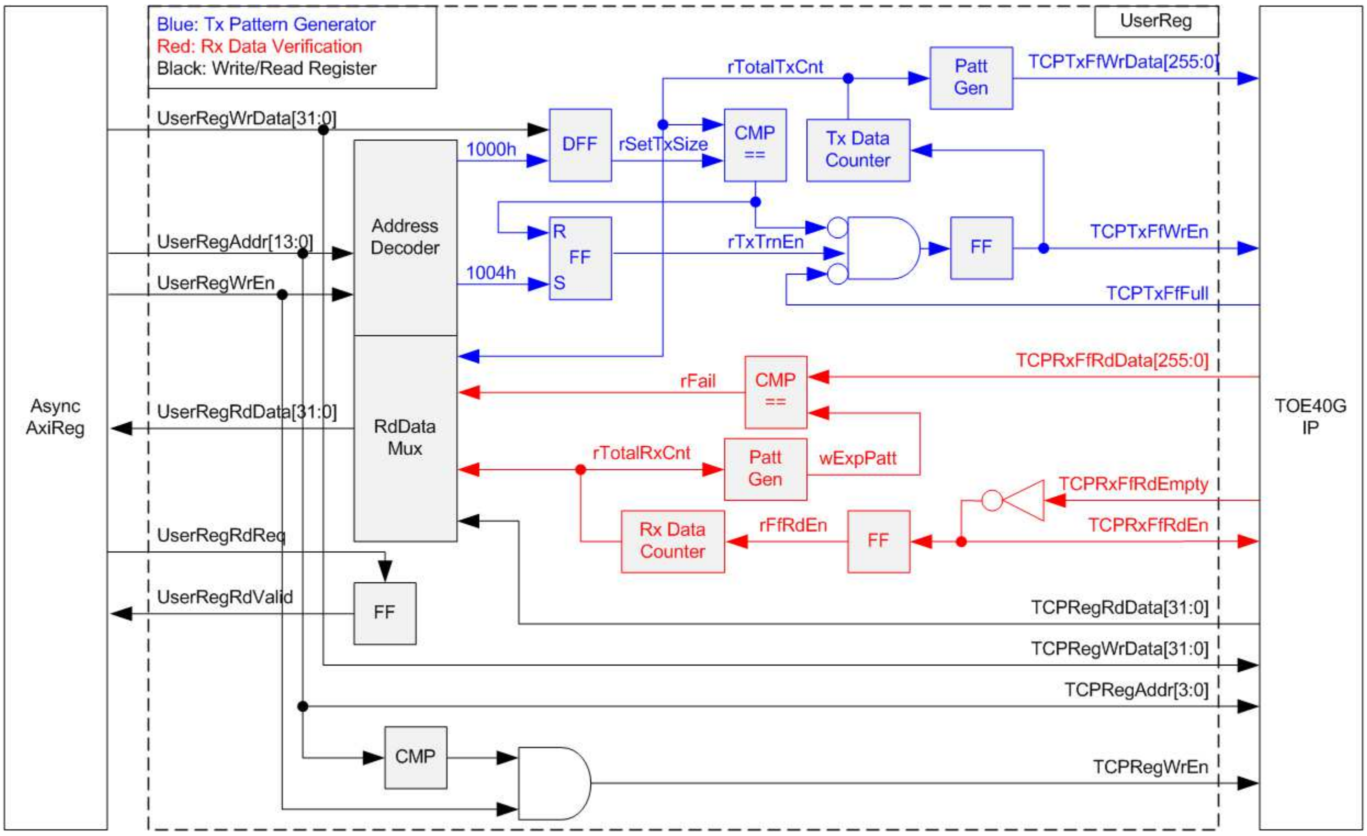


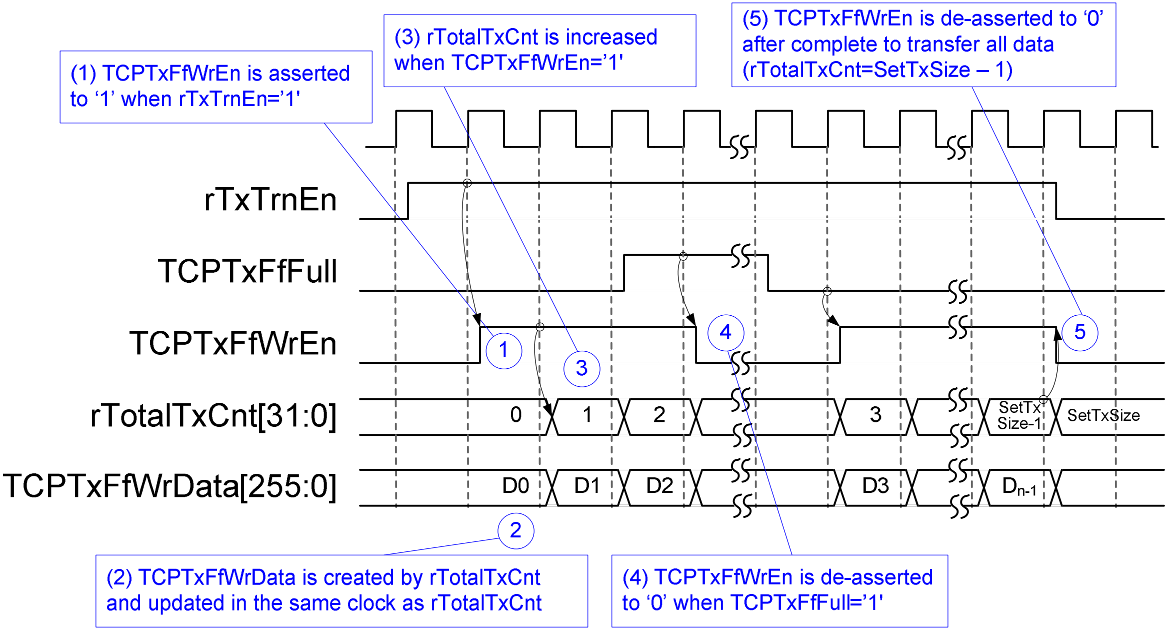
Figure 2-8 UserReg block diagram

Memory map of control and status signals inside UserReg module is shown in Table 2-1.
0x0000 – 0x00FF is mapped to registers inside TOE40G-IP.
0x1000 – 0x10FF is mapped to registers inside UserReg (to control Tx Pattern Generator and Rx Data Verification).

To request write register, UserRegWrEn is asserted to '1' with the valid of UserRegAddr. The upper bits of UserRegAddr are used to decode that CPU accesses TOE40G-IP area or internal register area. If CPU accesses TOE40G-IP area, TCPRegWrEn will be asserted to '1'. Otherwise, UserRegWrData is loaded to internal register which has matched lower bits of UserRegAddr. For example, rSetTxSize is loaded by UserRegWrData when UserRegAddr=0x1000. UserRegWrByteEn signal is not used in this module, so CPU firmware needs to access the hardware register by using 32-bit pointer only.

For read request, UserRegRdReq is asserted to '1'. RdDataMux selects status signals from internal register or TOE40G-IP, depending on the upper bits of UserRegAddr. In the next clock, the output of RdDataMux is forwarded to UserRegRdData. To synchronous with UserRegRdData, RegRdValid is designed by using one D Flip-flop, input by RegRdReq signal.

The upper logic in blue color of Figure 2-8 is designed to generate test data to TOE40G-IP. rTxTrnEn is asserted to '1' when write register address is 1004h. When rTxTrnEn is '1', TCPTxFfWrEn is controlled by TCPTxFfFull. TCPTxFfWrEn is de-asserted to '0' when TCPTxFfFull is '1'. rTotalTxCnt is data counter to check total data sending to TCPTxFf interface. rTotalTxCnt is also used to generate 32-bit increment data to TCPTxFfWrData signal. rTxTrnEn is de-asserted to '0' when total data has been transferred completely (total data is set by rSetTxSize).


(1) TCPTxFfWrEn is asserted to '1' when rTxTrnEn='1'

(3) rTotalTxCnt is increased when TCPTxFfWrEn='1'

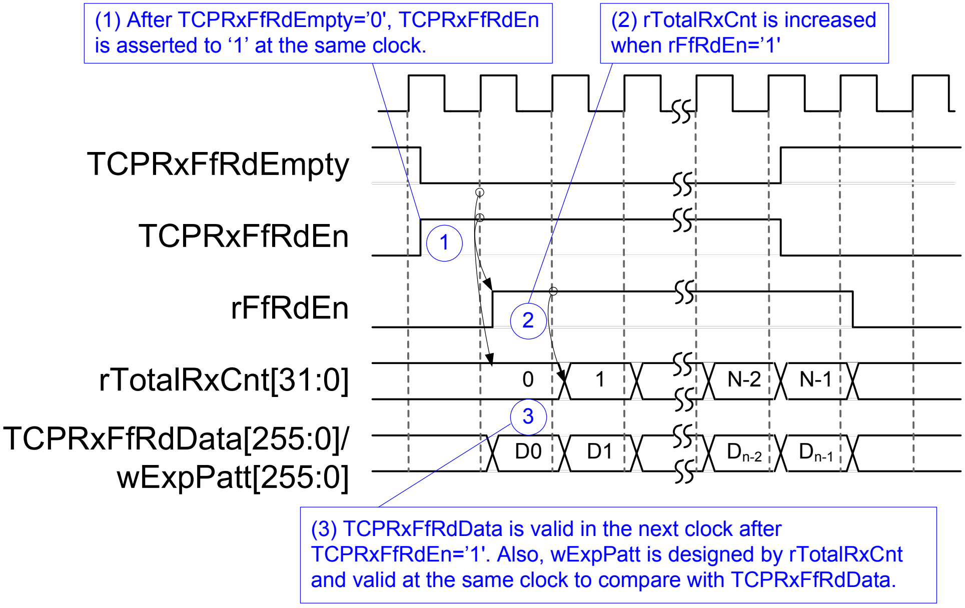
(5) TCPTxFfWrEn is de-asserted to '0' after complete to transfer all data (rTotalTxCnt=SetTxSize – 1)

(2) TCPTxFfWrData is created by rTotalTxCnt and updated in the same clock as rTotalTxCnt

(4) TCPTxFfWrEn is de-asserted to '0' when TCPTxFfFull='1'

Figure 2-9 Tx Pattern Generator timing diagram

The logic in red color of Figure 2-8 is designed to verify received data from TOE40G-IP. TCPRxFfRdEn is designed by using NOT logic of TCPRxFfRdEmpty. TCPRxFfRdData is valid in the next clock after asserting TCPRxFfRdEn to '1'. Read data (TCPRxFfRdData) is compared to expected pattern (wExpPatt) which is designed by rTotalRxCnt. rTotalRxCnt is data counter to check total data reading from TCPRxFf. Similar to Tx path, expected pattern is 32-bit increment pattern. Fail flag (rFail) will be asserted to '1' if Read Data is not equal to expected pattern.


(1) After TCPRxFfRdEmpty='0', TCPRxFfRdEn is asserted to '1' at the same clock.

(2) rTotalRxCnt is increased when rFfRdEn='1'

(3) TCPRxFfRdData is valid in the next clock after TCPRxFfRdEn='1'. Also, wExpPatt is designed by rTotalRxCnt and valid at the same clock to compare with TCPRxFfRdData.

Figure 2-10 Rx Data Verification timing diagram

Table 2-1 Register map Definition

Address Wr/Rd	Register Name (Label in the “toe40gip_demo.c”)	Description
BA+0x0000 – BA+0x00FF: TOE40G-IP Register Area More details of each register are described in Table2 of TOE40G-IP datasheet.		
BA+0x00	TOE40_RST_REG	Mapped to RST register within TOE40G-IP
BA+0x04	TOE40_CMD_REG	Mapped to CMD register within TOE40G-IP
BA+0x08	TOE40_SML_REG	Mapped to SML register within TOE40G-IP
BA+0x0C	TOE40_SMH_REG	Mapped to SMH register within TOE40G-IP
BA+0x10	TOE40_DIP_REG	Mapped to DIP register within TOE40G-IP
BA+0x14	TOE40_SIP_REG	Mapped to SIP register within TOE40G-IP
BA+0x18	TOE40_DPN_REG	Mapped to DPN register within TOE40G-IP
BA+0x1C	TOE40_SPN_REG	Mapped to SPN register within TOE40G-IP
BA+0x20	TOE40_TDL_REG	Mapped to TDL register within TOE40G-IP
BA+0x24	TOE40_TMO_REG	Mapped to TMO register within TOE40G-IP
BA+0x28	TOE40_PKL_REG	Mapped to PKL register within TOE40G-IP
BA+0x2C	TOE40_PSH_REG	Mapped to PSH register within TOE40G-IP
BA+0x30	TOE40_WIN_REG	Mapped to WIN register within TOE40G-IP
BA+0x34	TOE40_ETL_REG	Mapped to ETL register within TOE40G-IP
BA+0x38	TOE40_SRV_REG	Mapped to SRV register within TOE40G-IP
BA+0x1000 – BA+0x10FF: UserReg control/status		
BA+0x1000 Wr/Rd	Total transmit length (USER_TXLEN_REG)	Wr [31:0] – Total transmitted size in 256-bit unit. Valid from 1-0xFFFFFFFF. Rd [31:0] – Current transmitted size in 256-bit unit. The value is cleared to 0 when USER_CMD_REG is written by user.
BA+0x1004 Wr/Rd	User Command (USER_CMD_REG)	Wr [0] – Start Transmitting. Set ‘1’ to start transmitting. This bit is auto-cleared to ‘0’ after end of total transfer. [1] – Data Verification enable (‘0’: Enable data verification, ‘1’: Disable data verification) Rd [0] – Tx Busy. (‘0’: Idle, ‘1’: Tx module is busy) [1] – Data verification error (‘0’: Normal, ‘1’: Error) This bit is auto-cleared when user starts new operation or reset. [2] – Mapped to ConnOn signal of TOE40G-IP
BA+0x1008 Wr/Rd	User Reset (USER_RST_REG)	Wr [0] – Reset signal. Set ‘1’ to reset the logic. This bit is auto-cleared to ‘0’. [8] – Set ‘1’ to clear TimerInt latch value Rd [8] – Latch value of TimerInt output from IP (‘0’: Normal, ‘1’: TimerInt=’1’ is detected) This flag can be cleared by system reset condition or setting USER_RST_REG[8]=’1’.
BA+0x100C Rd	FIFO status (FIFO_STS_REG)	Rd [4:0]: Mapped to TCPRxFfLastRdCnt signal of TOE40G-IP [15:5]: Mapped to TCPRxFfRdCnt signal of TOE40G-IP [24]: Mapped to TCPTxFfFull signal of TOE40G-IP
BA+0x1010 Rd	Total Receive length (TRN_RXLEN_REG)	Rd [31:0] – Current received size in 256-bit unit. The value is cleared to 0 when USER_CMD_REG is written by user.

3 CPU Firmware Sequence

After FPGA boot-up, user must select the operation mode on FPGA to be client or server. The operation mode is set to TOE40_SRV_REG register. To initialize as client, FPGA sends ARP request to get the MAC address from the destination device. To initialize as server, FPGA waits ARP request from the destination device and then returns ARP reply.

To run the test by using two FPGAs, the operation mode on each FPGA must be set to different value (one is client and another is server). In case of running FPGA with PC, it is recommended to set FPGA to initialize as client mode. It is easier for PC to return ARP reply after receiving ARP request, comparing to setting PC to send ARP request to FPGA.

In the firmware, there are two default parameters for each operation mode. The initialization sequence after system boot-up is as follows.

1) CPU receives the operation mode from user and displays default parameters on the console.
2) User inputs 'x' to complete initialization sequence by using default parameters or inputs other keys to change some parameters. In case of changing parameters, the operation sequence is same as Reset IP menu (described in topic 3.2).
3) CPU waits until TOE40G-IP completes initialization sequence (TOE40_CMD_REG[0]='0').
4) Main menu is displayed to select one of five operations. More details of each menu are shown as follows.

3.1 Show parameters

This menu is used to show current parameters of TOE40G-IP such as operation mode, source MAC address, destination IP address, source IP address, destination port, and source port. The sequence of display parameters is as follows.

1) Read network parameters from each variable in firmware.
2) Print out each variable.

3.2 Reset IP

This menu is used to change TOE40G-IP parameters such as IP address, source port number. After setting TOE40G-IP register, CPU resets the IP to re-initialize by using new parameters. CPU monitors busy flag to wait until the initialization is completed. The sequence of reset sequence is shown as follows.

1) Display current parameter value to the console.
2) Receive input parameters from user and check input value whether it is valid or not. If the new input is invalid, the old value will be used instead of new value.
3) Force reset to IP by setting TOE40_RST_REG[0]='1'.
4) Set all parameters to TOE40G-IP register such as TOE40_SML_REG, TOE40_DIP_REG.
5) De-assert IP reset by setting TOE40_RST_REG[0]='0'.
6) Clear user logic status by sending reset to user logic (USER_RST_REG[0]='1').
7) Monitor IP busy flag (TOE40_CMD_REG[0]) until initialization sequence is completed (busy flag is de-asserted to '0').

3.3 Send data test

Three user inputs are required to set total transmit length, packet size, and connection mode (active open for client operation or passive open for server operation). The operation will be cancelled if the input is invalid. During the test, 32-bit increment data is generated from the logic and sent to PC/FPGA. Data is verified by Test application on PC (in case of PC <-> FPGA) or verification module in FPGA (in case of FPGA <-> FPGA). The operation is completed when total data are transferred from FPGA to PC/FPGA completely. The sequence of this test is as follows.

1) Receive transfer size, packet size, and connection mode from user and verify that the value is valid.
2) Set UserReg registers, i.e. transfer size (USER_TXLEN_REG), reset flag to clear initial value of test pattern (USER_RST_REG[0]='1'), and command register to start data pattern generator (USER_CMD_REG=0). After that, test pattern generator in UserReg transmits data to TOE40G-IP.
3) Display recommended parameter of test application running on PC by reading current parameters in the system.
4) Open connection following connection mode value.
   a. For active open, CPU sets TOE40_CMD_REG=2 and monitors ConnOn status (USER_CMD_REG[2]) until it is equal to '1'.
   b. For passive open, CPU waits until connection is opened by PC/FPGA by monitoring ConnOn status (USER_CMD_REG[2]) until it is equal to '1'.
5) Set packet size to TOE40G-IP register (TOE40_PKL_REG) and calculate total loops from total transfer size. Maximum transfer size of each loop is 4 GB. The operation of each loop is as follows.
   a. Set transfer size of this loop to TOE40G-IP register (TOE40_TDL_REG). The set value is equal to remaining transfer size for the last loop or equal to 4 GB for other loops.
   b. Set send command to TOE40G-IP register (TOE40_CMD_REG=0).
   c. Wait until operation is completed by monitoring busy flag (TOE40_CMD_REG[0]) until it is equal to '0'. During monitoring busy flag, CPU reads current transfer size from user logic (USER_TXLEN_REG and USER_RXLEN_REG) and displays the results on the console every second.
6) Set close connection command to TOE40G-IP register (TOE10_CMD_REG=3).
7) Calculate performance and show test result on the console.

3.4 Receive data test

User sets total received size, data verification mode (enable or disable), and connection mode (active open for client operation or passive open for server operation). The operation will be cancelled if the input is invalid. During the test, 32-bit increment data is generated to verify the received data from PC/FPGA when data verification mode is enabled. The sequence of this test is as follows.

1) Receive total transfer size, data verification mode, and connection mode from user input. Verify that all inputs are valid.
2) Set UserReg registers, i.e. reset flag to clear initial value of test pattern (USER_RST_REG[0]='1'), and data verification mode (USER_CMD_REG[1]='0' or '1').
3) Display recommended parameter (same as Step 3 of Send data test).
4) Open connection following connection mode value (same as Step 4 of Send data test).
5) Wait until connection is closed by PC/FPGA by monitoring Connon status (USER_CMD_REG[2]) until it is equal to'0'. During monitoring, CPU reads current transfer size from user logic (USER_TXLEN_REG and USER_RXLEN_REG) and displays the results on the console every second.
6) Compare total received length of user logic (USER_RXLEN_REG) to set value from user. Wait until total length is equal to set value. After that, check verification result is not failed (USER_CMD_REG[1] = '0'). If the error is detected, error message will be displayed.
7) Calculate performance and show test result on the console.

3.5 Full duplex test

This menu is designed to run full duplex test by transferring data between FPGA and PC/FPGA in both directions by using same port number at the same time. Four inputs are received from user, i.e. total size for both directions, packet size for FPGA sending logic, data verification mode for FPGA receiving logic, and connection mode (active open/close for client operation or passive open/close for server operation).

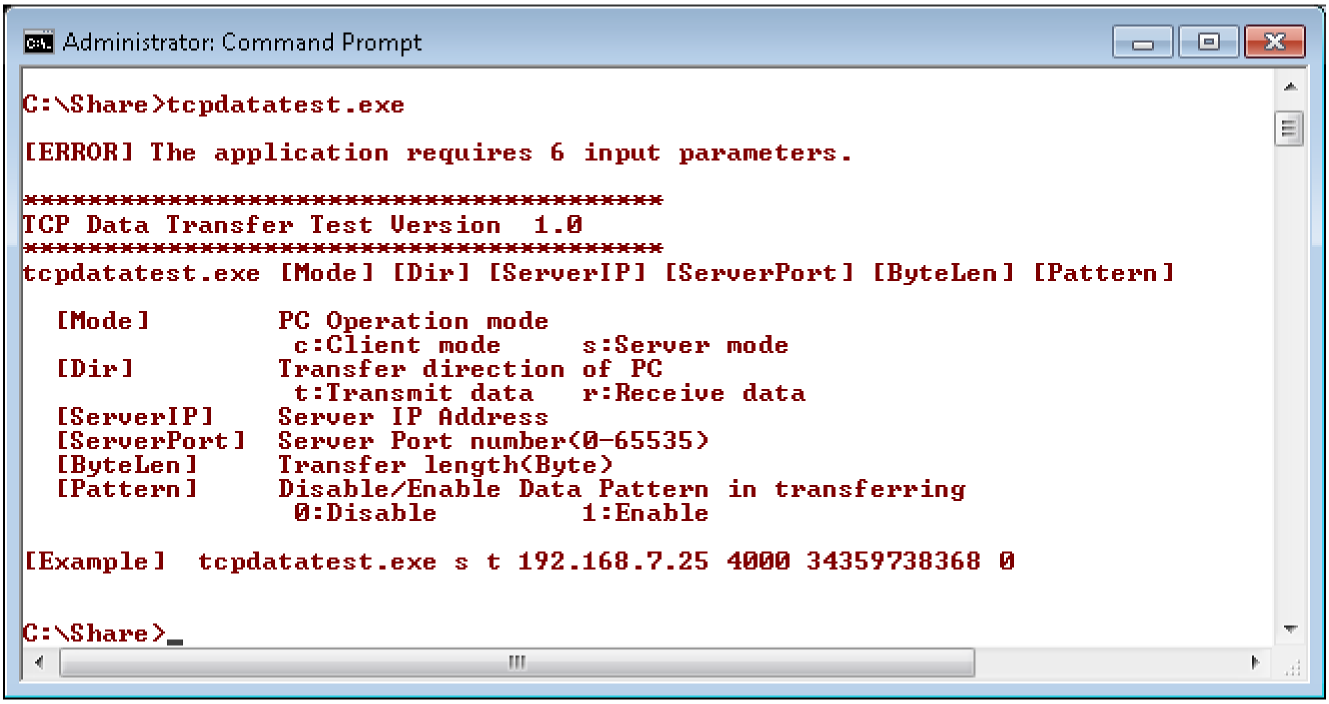
When running the test by using PC and FPGA, transfer size setting on FPGA must be matched to size setting on test application (tcp_client_txrx_40G). Connection mode on FPGA when running with PC must be set to passive (server operation).

The test runs in forever loop until user cancels operation on PC (input Ctrl+C) in case transferring data between PC and FPGA. For FPGA <-> FPGA environment, user cancels operation by input some keys to the console. The sequence of this test is as follows.

1) Receive total data size, packet size, data verification mode, and connection mode from user and verify that the value is valid.
2) Display recommended parameter of test application running on PC by reading current parameters in the system.
3) Set UserReg registers, i.e. transfer size (USER_TXLEN_REG), reset flag to clear initial value of test pattern (USER_RST_REG[0]='1'), and command register to start data pattern generator with data verification mode (USER_CMD_REG=1 or 3).
4) Open connection following connection mode value (same as Step 4 of Send data test).
5) Set TOE40G-IP registers such as packet size (TOE40_PKL_REG=user input), and then calculate total transfer size in each loop. Maximum size of one loop is 4 GB. The operation of each loop is as follows.
    a. Set transfer size of this loop to TOE40_TDL_REG. Except the last loop, transfer size in each loop is set to maximum size (4GB) which is also aligned to packet size. For the last loop, transfer size is equal to the remaining size.
    b. Set send command to TOE40G-IP register (TOE40_CMD_REG=0).
    c. Wait until send command is completed by monitoring busy flag (TOE40_CMD_REG[0]) until it is equal to '0'. During monitoring busy flag, CPU reads current transfer size from user logic (USER_TXLEN_REG and USER_RXLEN_REG) and displays the results on the console every second.
6) Close connection following connection mode value.
    a. For active close, CPU waits until received transfer size is equal to set value. Then, set USER_CMD_REG=3 to close connection. Next, CPU waits until connection is closed by monitoring ConnOn (USER_CMD_REG[2])='0'.
    b. For passive close, CPU waits until connection is closed from FPGA/PC by monitoring ConnOn (USER_CMD_REG[2])='0'.
7) Check received result and error (same as Step 6 of Receive data test).
8) Calculate performance and show test result on the console. Go back to step 3 to run the test in forever loop.

4 Test Software Sequence

4.1 “tcpdatatest” for half duplex test



```
C:\Share>tcpdatatest.exe

[ERROR] The application requires 6 input parameters.

****************************************
TCP Data Transfer Test Version  1.0
****************************************
tcpdatatest.exe [Mode] [Dir] [ServerIP] [ServerPort] [ByteLen] [Pattern]

  [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

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


C:\Share>_
```

Figure 4-1 “tcpdatatest” application usage

“tcpdatatest” is designed to run on PC for sending/receiving TCP data through Ethernet for both server and client mode. PC of this demo runs in client mode only. User inputs parameter to select transfer direction and the mode. Six parameters are required, i.e.

1) Mode: c – when 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 192.168.40.42)
4) ServerPort: Port number of FPGA when PC runs in client mode (default is 60000)
5) ByteLen: Total transfer size in byte unit. This input is used in transmit mode only and ignored in receive mode. In receive mode, application is closed when connection is destroyed. ByteLen in transmit mode must be equal to transfer size setting in NiosII command shell (under received data test submenu) and the value must be aligned to 32 (byte).
6) Pattern:
   0 – Generate dummy data in transmit mode or disable data verification in receive mode.
   1 – Generate increment data in transmit mode or enable data verification in receive mode.

Transmit data mode

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

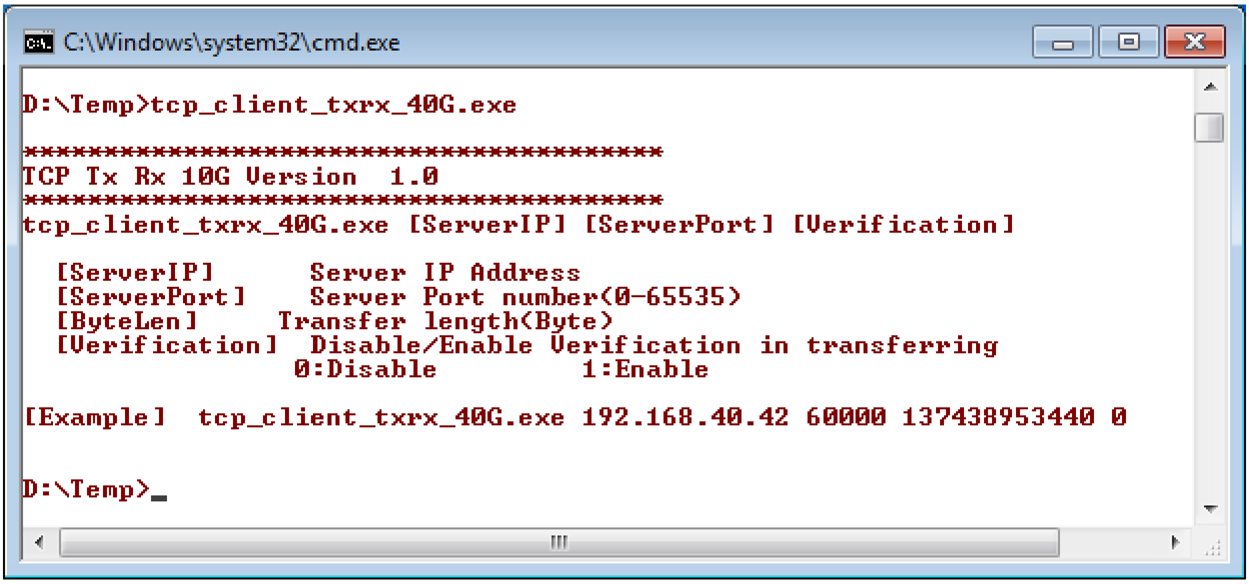
1) Allocate 1 MB memory to be send buffer.
2) Create socket and set properties of send buffer.
3) Create new connection to server by using IP address and port number from user.
4) Generate increment test pattern to send buffer when test pattern is enabled. Skip this step if dummy pattern is selected.
5) Send data out and decrease remaining transfer size.
6) Print total transfer size every second.
7) Run step 4) – 6) in the loop until remaining transfer size is 0.
8) Close socket and print total size and performance.

Receive data mode

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

1) Allocate 1 MB memory to be received buffer.
2) Create socket and set properties of received buffer.
3) Same step as step3) in Transmit data mode.
4) Read data from received buffer and increase total received data size.
5) If verification is enabled, data will be verified with increment pattern and error message will be printed out when data is not correct. Skip this step if data verification is disabled.
6) Print total transfer size every second.
7) Run step 4) – 6) in the loop until connection status is closed.
8) Close socket and print total size and performance.

4.2 “tcp_client_txrx_40G” for full duplex test



```
D:\Temp>tcp_client_txrx_40G.exe

****************************************
TCP Tx Rx 10G Version  1.0
****************************************
tcp_client_txrx_40G.exe [ServerIP] [ServerPort] [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_40G.exe 192.168.40.42 60000 137438953440 0


D:\Temp>_
```

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 same port number at the same time. The application is run in client mode, so user needs to input server parameters (network parameters of TOE40G-IP). As shown in Figure 4-2, there are three parameters to run the application, i.e.

1) ServerIP: IP address of FPGA
2) ServerPort: Port number of FPGA
3) ByteLen: Total transfer size in byte unit. This is total size to transmit and receive data. The value must be aligned to 32 (byte).
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 increment data for sending function and enable data verification for receiving function.

The sequence of test application is as follows.
(1) Allocate 60 KB memory for send and receive buffer.
(2) Create socket and set properties.
(3) Create new connection by using IP address and port number from user.
(4) Generate increment test pattern to send buffer when test pattern is enabled. Skip this step if dummy pattern is selected.
(5) Send data out and decrease remaining transfer size.
(6) Read data from received buffer and increase total received data size.
(7) If verification is enabled, data will be verified by increment pattern and error message will be printed out when data is not correct. Skip this step if data verification is disabled.
(8) Print total transfer size every second
(9) Run step 5) – 8) until total sending/receiving data are equal to ByteLen (input from user)
(10) Print total size and performance and close socket.
(11) Sleep for 1 second to wait the hardware complete current test loop.
(12) Run step 3) – 11) in forever loop. If verification is fail, the application will quit.

5 Revision History

Revision	Date	Description
1.0	24-Dec-18	Initial version release