TOE1G-IP two-port (HW) reference design

TOE1G-IP two-port (HW) reference design

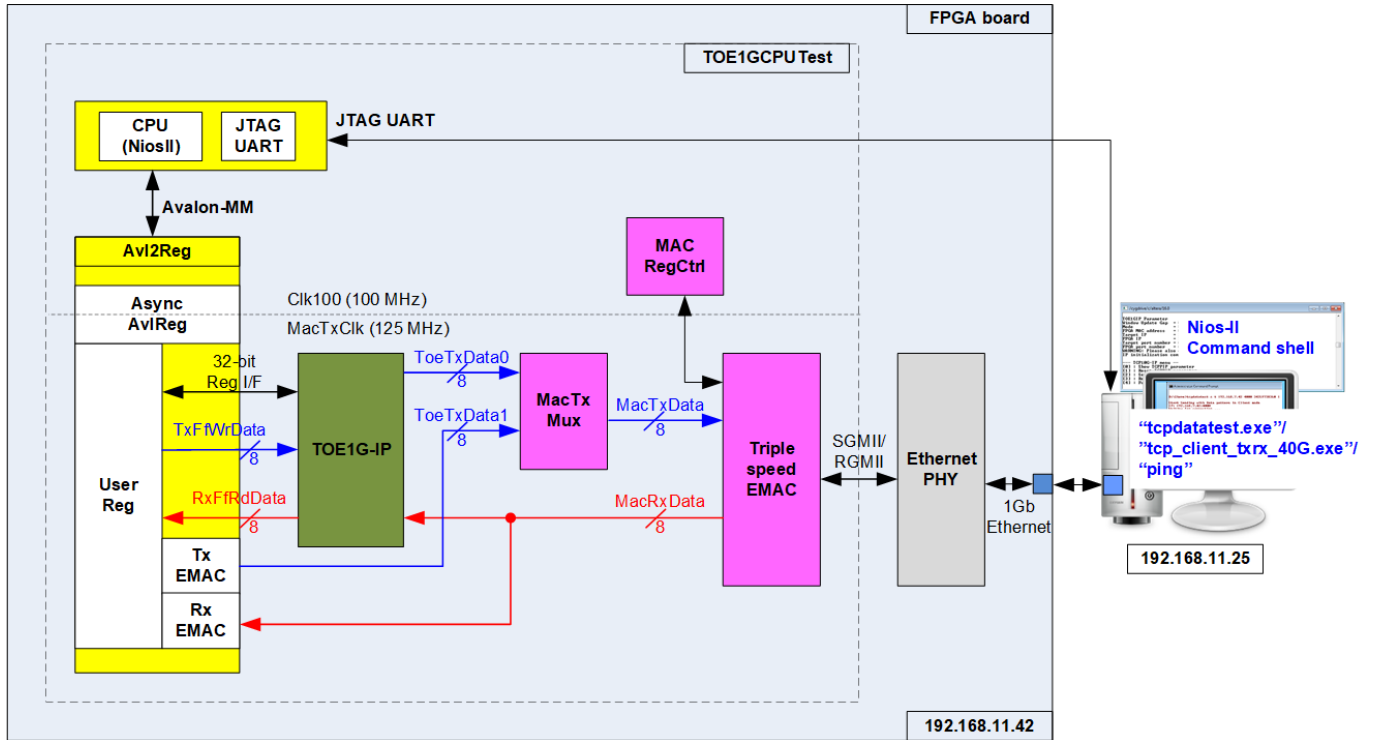Rev1.0   4-Jul-23

1   Overview

Figure 1-1 Top module block diagram

TOE1G-IP can transfer 1Gb Ethernet data by using TCP/IP protocol at high-speed rate based on one session. Some applications need to use two sessions for operating, one for data port and another for control port. Typically, data port must run at high-speed rate while control port can run at slow speed.

This reference design is designed to support the application which must use two sessions for data port and control port. Data port is designed by TOE1G-IP for transferring at high-speed while control port is designed by using simple logic (TxEMAC and RxEMAC) operating with NiosII firmware. MacTxMux is applied to select and switch sent packet between TOE1G-IP and TxEMAC. Data output from EMAC is forwarded to both TOE1G-IP and RxEMAC which include the logic to filter the received packet.

The slow port in the reference design can support only Ping command, following ICMP protocol. User can modify NiosII firmware and RxEMAC logic to support other protocol.

The high-speed port in the reference design uses the same designs as TOE1G-IP demo design. UserReg includes the logic to generate test pattern and verify test pattern with TOE1G-IP. More details of TOE1G-IP demo design are described in the following document.

https://dgway.com/products/IP/TOE1G-IP/dg_toe1gip_cpu_refdesign_intel_en.pdf
https://dgway.com/products/IP/TOE1G-IP/dg_toe1gip_cpu_instruction_intel_en.pdf

2 Hardware Structure

The hardware of two port reference design is modified from TOE1G-IP reference design, so most logics to run with TOE1G-IP is the same as TOE1G-IP reference design, i.e. EMAC, MACRegCtrl, TOE1G-IP, AsyncAvlReg, and the logic in UserReg for TOE1G-IP interface. Please see more details of these logics in the TOE1G-IP reference design document.

This document describes only the additional logics for operating slow port connection, i.e. MacTxMux, TxEMAC, RxEMAC, and some logics in UserReg for controlling TxEMAC and RxEMAC.
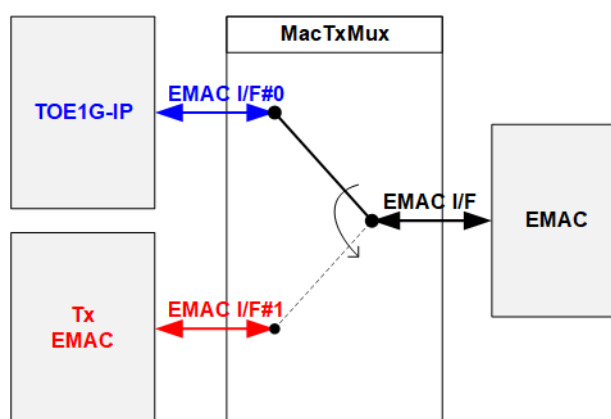
2.1 MacTxMux

Figure 2-1 MacTxMux block diagram

MacTxMux selects data source to EMAC which can be fed from TOE1G-IP on channel#0 or TxEMAC on channel#1. rChSel is designed to control the connection switch by setting to 0 for forwarding packet from TOE1G-IP or '1' for forwarding packet from TxEMAC. To hold the transmit packet from the inactive channel, MacTxReady output from MacTxMux is de-asserted to '0'.

rChSel changes the value to switch the active channel when two conditions are met.
1) Another channel (inactive) asserts MacTxValid for asking the new packet transmission.
2) The active channel finishes the packet transmission or there is no packet transmission in the active channel.
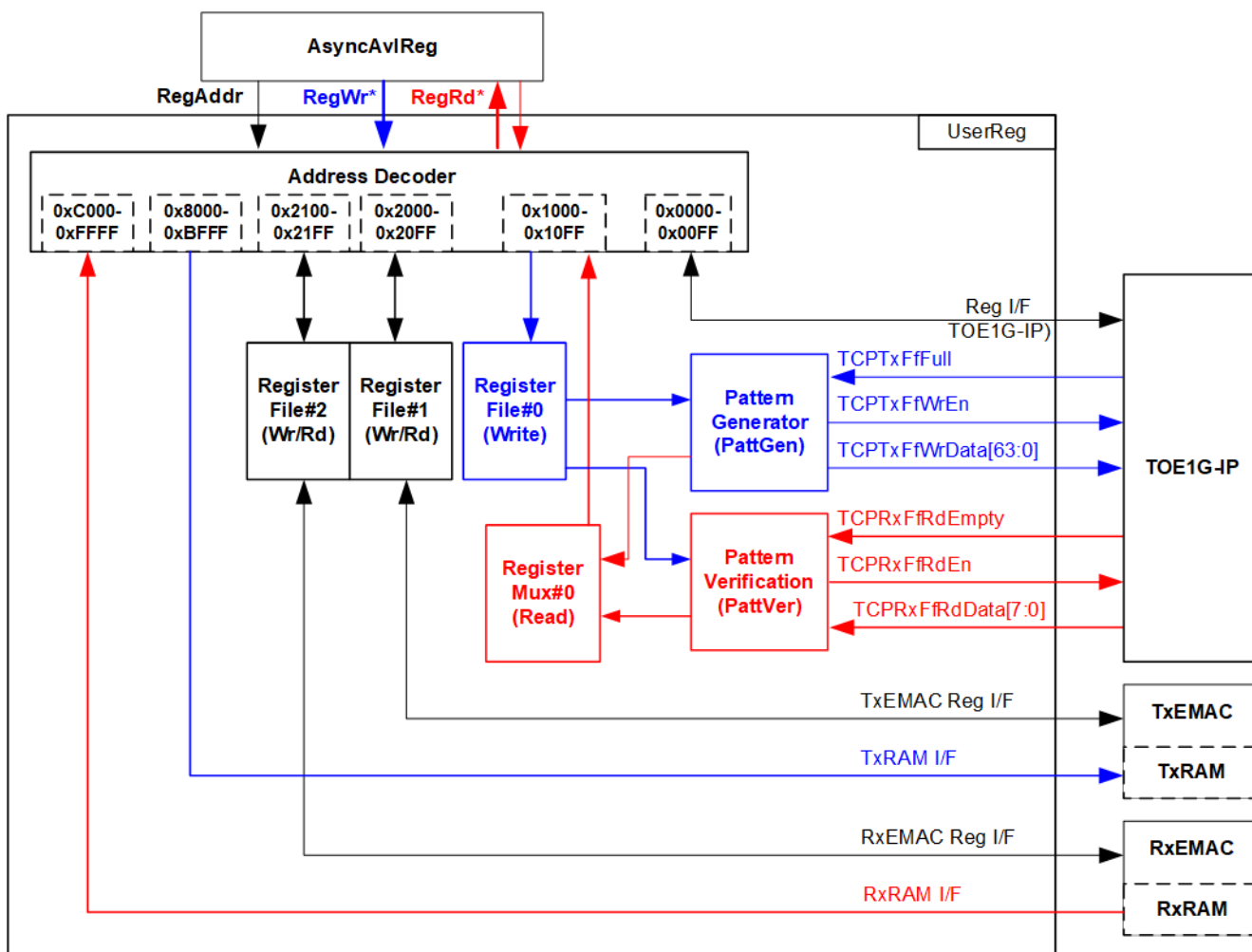
2.2   UserReg

Figure 2-2 UserReg block diagram

Address decoder uses the upper bit of address for selecting the active hardware which is split to six area as follows.

    0x0000-0x00FF: TOE1G-IP register (Write/Read)
    0x1000-0x10FF: Pattern generator and Pattern verification logic (Write/Read)
    0x2000-0x20FF: TxEMAC control/status register (Write/Read)
    0x2100-0x21FF: RxEMAC control/status register (Write/Read)
    0x8000-0xBFFF: TxRAM (Write)
    0xC000-0xFFFF: RxRAM (Read)

The lower bit of address is fed to each area to select the active register or data. For write access, all registers inside the test logic are designed as 32-bit register access, so write byte enable (RegWrByteEn) is not used for internal registers. TxRAM is designed to support byte access to allow CPU creating Ethernet packet by using byte pointer. RegWrByteEn is applied to be write byte enable of TxRAM.

For read access, the returned data from the hardware is fed to two-level multiplexers, so data latency of read access is equal to two clock cycles. RegRdValid is created by RegRdReq with asserting two D Flip-flops.

The memory map of TOE1G-IP, Pattern generator, and Pattern verification has the same description as TOE1G-IP demo reference design. So, this document describes only the remaining address.

<u>Table 2-1 UserReg memory map definition</u>

| Address	Register Name	Description
Wr/Rd	Label in "twoport_hw.c"	
BA+0x2000 – BA+0x20FF: TxEMAC control/status register		
BA+0x2000	TxEMAC command	Wr [0] – '1': Start transmit data from TxRAM to EMAC
Wr/Rd	(TXMAC_CMD_REG)	[31] – '1': Clear TxEMAC Finish flag
		Rd [0] – '0': TxEMAC is idle, '1': TxEMAC is busy
		[31] – '0': TxEMAC still not finish, '1': TxEMAC finish
BA+0x2004	Start address of TxRAM	Wr [13:0] – Start TxRAM address to transfer data to EMAC in byte unit.
Wr/Rd	(TXMAC_ADDR_REG)	Rd [13:0] – Current value of TxRAM address in byte unit.
BA+0x2008	Transfer length of TxEMAC	Wr [13:0] – Total transfer length of TxEMAC in byte unit
Wr/Rd	(TXMAC_LEN_REG)	Rd [13:0] – Remaining transfer length of TxEMAC in byte unit.
BA+0x2100 – BA+0x21FF: RxEMAC control/status register		
BA+0x2100	RxEMAC command	Wr [31] – '1': Clear RxEMAC Finish flag
Wr/Rd	(RXMAC_CMD_REG)	Rd [31] – '0': RxEMAC still not finish, '1': RxEMAC finish
BA+0x2104	RxRAM pointer	Wr [3:0] – Read pointer of RxRAM
Wr/Rd	(RXMAC_PTR_REG)	Rd [3:0] – Write pointer of RxRAM
BA+0x2110	Destination MAC address [31:0]	Wr [31:0] – Lower 32-bit MAC address value for FPGA
Wr	(RXMAC_DML_REG)	
BA+0x2114	Destination MAC address [47:32]	Wr [15:0] - Upper 16-bit MAC address value for FPGA
Wr	(RXMAC_DMH_REG)	
BA+0x8000 – BA+0xFFFF: TxRAM/RxRAM		
BA+0x8000-BA+0xBFFF	RAM in TxEMAC	16 Kbyte RAM to store transmit data of slow port connection
Wr	(TXRAM_BASE_ADDR)	
BA+0xC000-BA+0xFFFF	RAM in RxEMAC	16 Kbyte RAM to store receive data of slow port connection
Rd	(RXRAM_BASE_ADDR)	

2.3 TxEMAC
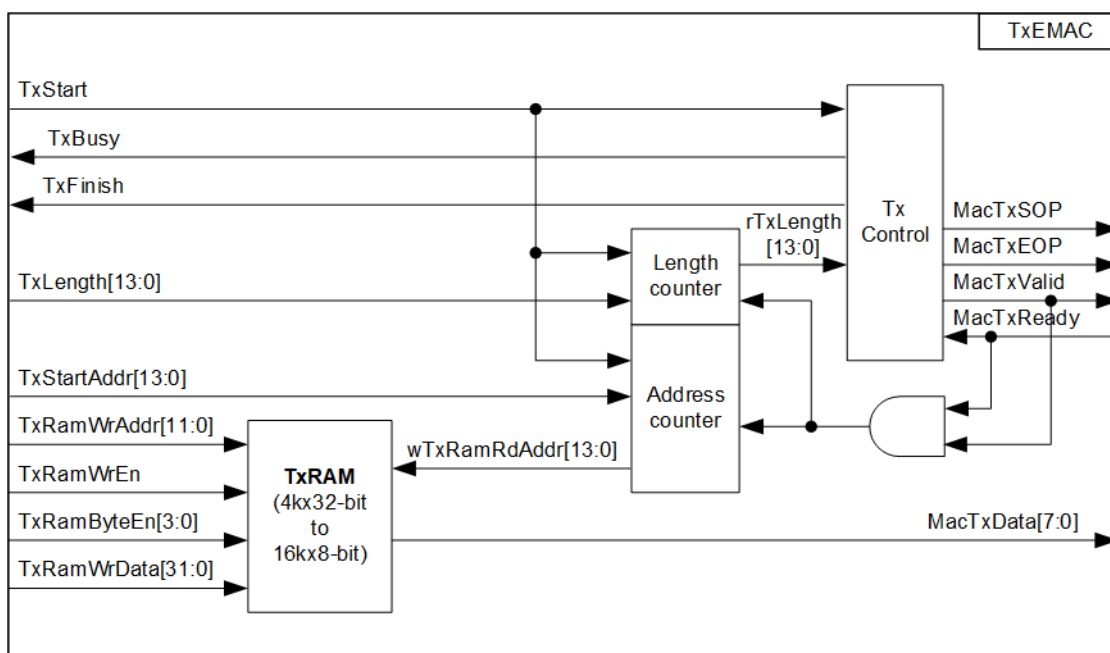
Figure 2-3 TxEMAC block diagram

TxEMAC is the module for CPU transferring Ethernet packet to EMAC. The transmit packet is prepared in TxRAM which supports to access as 8-bit, 16-bit, and 32-bit unit by CPU. TxRAM is applied to convert 32-bit interface (CPU bus size) to be 8-bit interface (EMAC bus size). TxRAM size is 16Kbyte.

TxEMAC receives start address (TxStartAddr) and transfer length (TxLength) from UserReg. After start flag (TxStart) is asserted to '1' from CPU, the data from TxRAM begins to forward to EMAC. There are two internal counters inside TxEMAC, i.e. Address counter for generating read address of TxRAM and Length counter for controlling the transmitted packet size. TxControl asserts MacTxSOP to '1' with MacTxValid at the 1st data of the packet and asserts MacTxEOP to '1' at the last data of the packet. MacTxValid and MacTxReady are applied to be counter enable for checking total data transmitting to EMAC. When all data are transmitted, TxFinish flag is asserted to '1' for one clock cycle. TxBusy is designed to show the operating time of TxEMAC. This signal is asserted to '1' when the packet is transmitting. CPU must monitor TxBusy status that is not asserted to '1' before asserting TxStart.
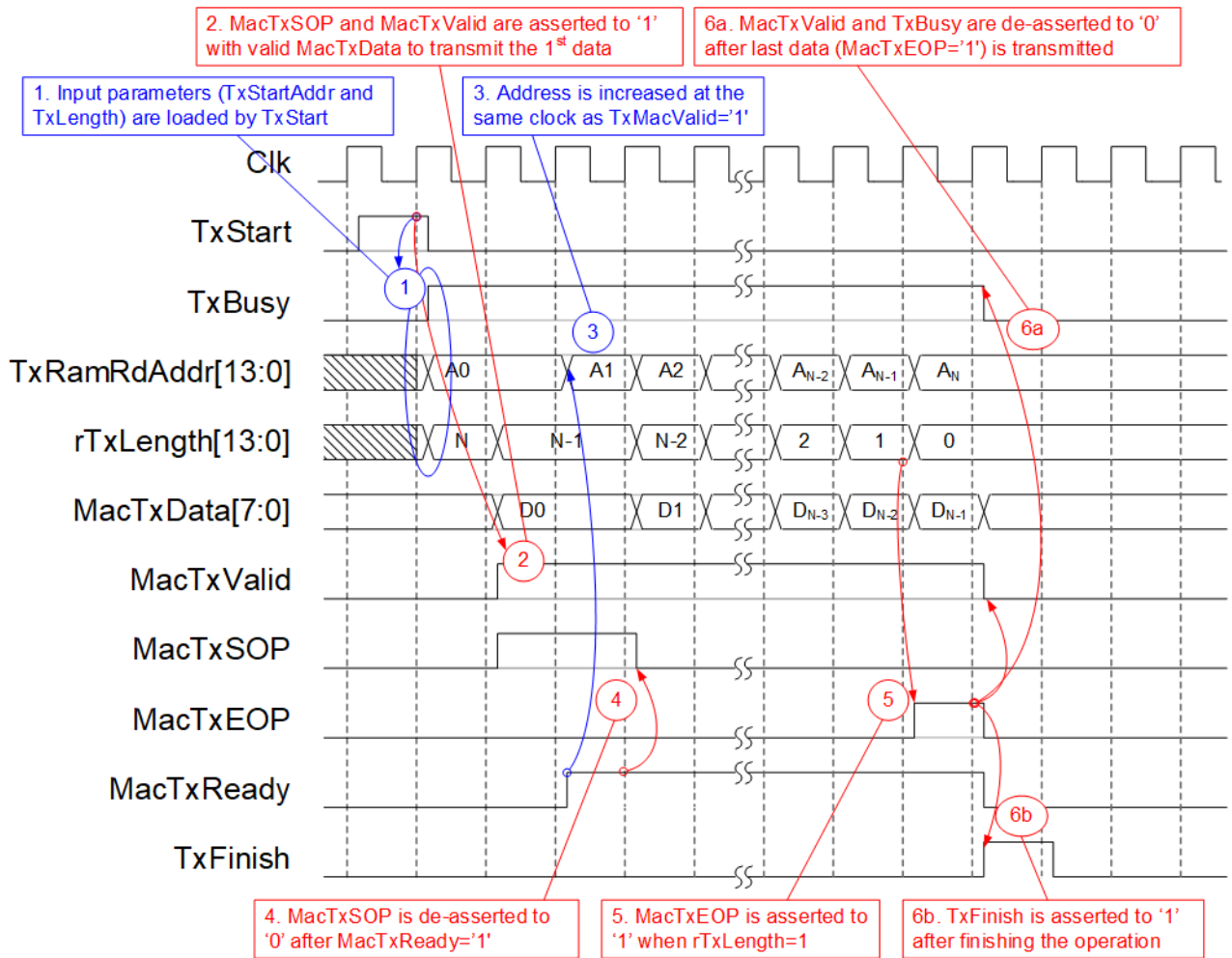
2. MacTxSOP and MacTxValid are asserted to '1' with valid MacTxData to transmit the 1st data

6a. MacTxValid and TxBusy are de-asserted to '0' after last data (MacTxEOP='1') is transmitted

1. Input parameters (TxStartAddr and TxLength) are loaded by TxStart

3. Address is increased at the same clock as TxMacValid='1'

4. MacTxSOP is de-asserted to '0' after MacTxReady='1'

5. MacTxEOP is asserted to '1' when rTxLength=1

6b. TxFinish is asserted to '1' after finishing the operation

Figure 2-4 Timing diagram of TxEMAC

Timing diagram of TxEMAC for transmitting the packet to EMAC is shown in Figure 2-4.

(1) TxStartAddr is loaded to TxRamRdAddr and TxLength is loaded to rTxLength when TxStart is asserted to '1' to begin data packet transmission.

(2) In the next two-clock cycle after TxStart is asserted to '1', MacTxData which is read data output from TxRAM is valid for transmission. In this clock, MacTxValid and MacTxSOP are asserted to '1' to send the 1st data to EMAC.

(3) At the same clock as MacTxReady asserted to '1', TxRamRdAddr is increased to the next value to read the next data from TxRAM.

(4) The next data (D1) is valid on the bus to send the 2nd data and MacTxSOP is de-asserted to '0'. Read data from TxRAM is forwarded to EMAC continuously.

(5) The last data is read from TxRAM when rTxLength is equal to '1'. So, MacTxEOP is asserted to '1' to send the end of packet signal.

(6) MacTxValid, MacTxEOP, and TxBusy are de-asserted to '0' after the last data is transmitted, detected by MacTxEOP='1' and MacTxReady='1'. In the same clock, TxFinish is asserted to '1' for one clock cycle to be interrupt signal for CPU. CPU can detect that the operation is finished.
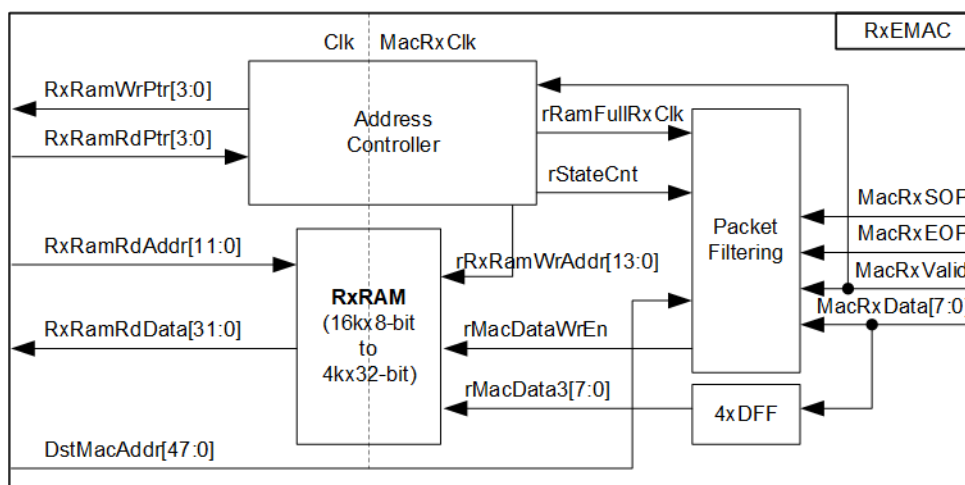
2.4 RxEMAC

Figure 2-5 RxEMAC block diagram

RxEMAC has two clock domains, i.e. MacRxClk which is clock domain for receiving EMAC packet from EMAC and Clk which is clock domain for interface with UserReg. So, RxRAM is the RAM which has different bus size and different clock domain for write interface and read interface. Packet Filtering checks the packet type and the parameter of the received packet. Only ICMP packet which has the matched destination MAC address (DstMacAddr) is stored to RxRAM. Otherwise, the packet is rejected. rMacDataWrEn is controlled by Packet Filtering to bypass or reject the received packet. The address to write RxRAM (rRxRamWrAddr) is controlled by Address controller. RxRamWrPtr is generated as the output for CPU detecting the new packet inside RxRAM. RxRamRdPtr is set by CPU for Address controller checking total remaining packet inside RxRAM. Full flag (rRamFullRxClk) is asserted to '1' when RxRAM is not free to store the new packet. More details of each submodule inside RxEMAC is described as follows.

Packet Filtering

Packet Filtering verifies the header of received packet from EMAC. Only ICMP packet which has the correct destination MAC address is stored to RxRAM. The header of ICMP is shown in Figure 2-6.

Bits 0-7	Bits 8-15	Bits 16-23	Bits 24-31	Bits 32-39	Bits 40-47	Bits 48-55	Bits 56-63
Destination MAC Address=Broadcast/Set value						Source MAC Address	
Source MAC Address				Ethernet Type=IPv4		Version/HL	Type of service
Length		Id Number		Fragment Offset		TTL	Protocol = ICMP
IP Checksum		Source IP address				Destination IP address	
Destination IP address		...					

Figure 2-6 Packet filtering

The received packet is valid when all following conditions are met.
1) Destination MAC address = RXMAC_DML/H_REG or broadcast address
2) Ethernet Type = IPv4 (0x0800)
3) IP Version = 4 and IP Header Length = 5 (Version/HL=0x45)
4) Protocol = ICMP (0x01)

Destination MAC Address is the input received from UserReg while other values are constant. User can modify the logic of the header filtering which is designed by using simple state machine to support other packet type. When the packet is valid, state machine (rStateCnt) is equal to 24 until the end of packet. Otherwise, StateCnt is reset to 0.

RxRAM

RxRAM has 16 Kbyte size to store only valid packet. 16Kbyte area is split into 8 slots to store up to 8 packets, so one packet size must be less than 2 Kbyte which is enough for normal frame, but not for jumbo frame (up to 9Kbyte size). The process to store and read packet of RxRAM is shown in Figure 2-7.
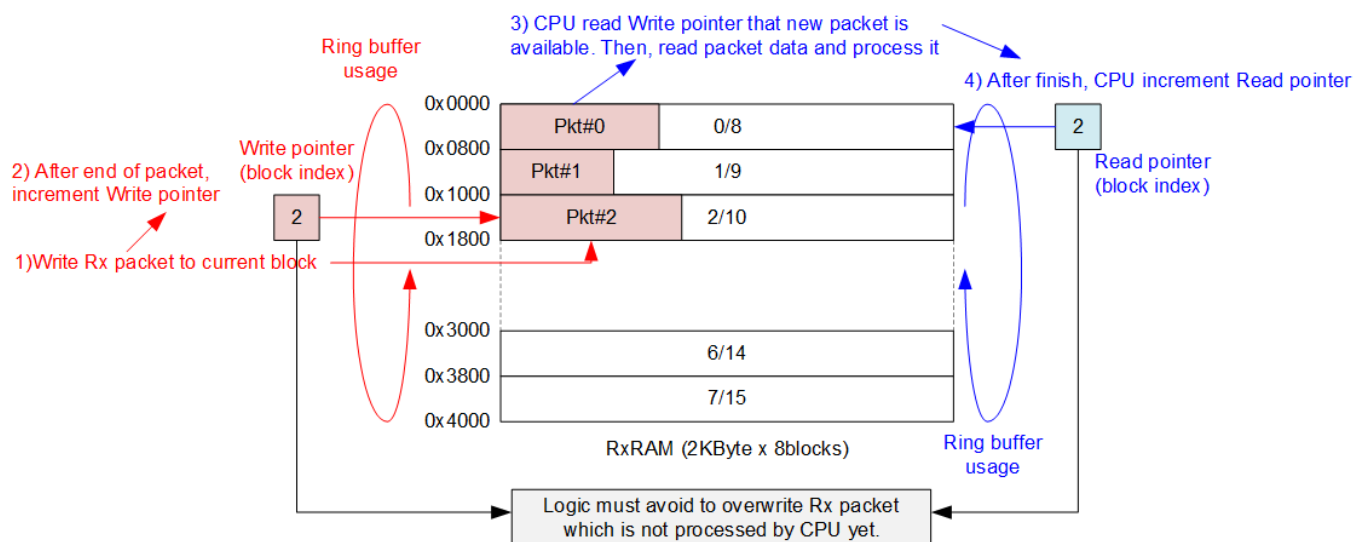

Figure 2-7 RxRAM pointer sequence

RxRAM is designed to be Ring buffer which is controlled by write pointer and read pointer. At the end of valid packet, the write pointer is increased to the next 2Kbyte address. By using 4-bit pointer, the full condition is detected when the different value between write pointer and read pointer is equal to 8. When full is found, new packet is not stored to RxRAM until CPU update the read pointer for de-asserting full condition.

3 CPU Firmware Sequence

After FPGA boot-up, welcome message is displayed. 1G Ethernet link up status (USER_RST_REG[16]) is monitored until link-up is found. Next, user selects the operation mode of TOE1G-IP to be client or server mode.

To initialize as client mode, TOE1G-IP sends ARP request to get the MAC address from the destination device. For server mode, TOE1G-IP waits ARP request to decode MAC address and returns ARP reply to complete initialization process.

If test environment uses two FPGA boards, the operation mode on two TOE1G-IPs must be different (one is client and another is server). To run with PC, it is recommended to set FPGA as client mode. When PC receives ARP request, PC always returns ARP reply. It is not simple to force PC sending ARP request to FPGA.

The software has two default parameters for each operation mode. Figure 3-1 shows the example of the initialization sequence after system boot-up.

Figure 3-1 Example of initialization sequence in client mode on NiosII terminal

There are four steps to complete initialization sequence 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. 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 TOE1G-IP finishing initialization sequence (TOE_CMD_REG[0]='0').
4) Main menu is displayed. There are five test operations for user selection. More details of each menu are described as follows.

3.1   Show parameters

This menu is used to show current parameters of TOE1G-IP, i.e. operation mode, source MAC address, destination IP address, source IP address, destination port, and source port. The step to display parameters is as follows.
1) Read all network parameters from each variable in firmware.
2) Print out each variable.

3.2   Reset IP

This menu is used to change TOE1G-IP parameters such as IP address and source port number. After setting updated parameter to TOE1G-IP register, the CPU resets the IP to re-initialize by using new parameters. Finally, the CPU monitors busy flag to wait until the initialization is completed. The step to reset IP is as follows.
1) Display current parameter value to the console.
2) Receive new input parameters from user and check input value whether valid or not. When the input is invalid, the old value is used instead.
3) Force reset to IP by setting TOE_RST_REG[0]='1'.
4) Set all parameters to TOE1G-IP register such as TOE_SML_REG and TOE_DIP_REG.
5) De-assert IP reset by setting TOE_RST_REG[0]='0'.
6) Clear PattGen and PattVer logic by sending reset to user logic (USER_RST_REG[0]='1').
7) Monitor IP busy flag (TOE_CMD_REG[0]) until initialization sequence is completed (busy flag is de-asserted to '0').

3.3 Send data test

Three user inputs are received to set total transmit length, packet size, and connection mode (active open for client operation or passive open for server operation). The operation is cancelled if some inputs are invalid. During the test, 32-bit incremental data is generated from the logic and sent to PC or FPGA. Data is verified by Test application on PC (in case of PC <-> FPGA) or verification module in FPGA (in case of FPGA <-> FPGA). The operation is finished when total data are transferred from FPGA to PC or FPGA. The step to run send data 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 sends data to TOE1G-IP.

3) Display recommended parameter of test application running on PC from the current system parameters.

4) Open connection following connection mode.
   a. For active open, CPU sets TOE_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 or FPGA. ConnOn status (USER_CMD_REG[2]) is monitored until it is equal to '1'.

5) Set packet size to TOE1G-IP register (TOE_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 TOE1G-IP register (TOE_TDL_REG). Transfer size is fixed to 4 GB except the last loop which is equal to the remaining size.
   b. Set send command to TOE1G-IP register (TOE_CMD_REG=0).
   c. Wait until operation is completed by monitoring busy flag (TOE_CMD_REG[0])='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 TOE1G-IP register (TOE_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 is cancelled when some inputs are invalid. During the test, 32-bit incremental data is generated to verify the received data from PC or FPGA when data verification mode is enabled. The step to run receive data 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 test pattern value (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 (same as Step 4 of Send data test).
5) Wait until connection is closed by PC or FPGA. Connon status (USER_CMD_REG[2]) is monitored until it is equal to'0'. During monitoring ConnOn, 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) Read total received length of user logic (USER_RXLEN_REG) and wait until total length is equal to the set value from user. After total data is received, CPU checks verification result by reading USER_CMD_REG[1] ('0': normal, '1': error). When the error is detected, the error message is 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 the 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, the transfer size setting on FPGA must be matched to the 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 the user cancels operation on PC by input Ctrl+C for PC and FPGA environment. For FPGA <-> FPGA environment, user cancels operation by pressing some keys to the console. The step to run Full duplex test is as follows.

1) Receive total data size, packet size, data verification mode, and connection mode from the user and verify that the value is valid.
2) Display the recommended parameters of test application running on PC from the current system parameters.
3) Set UserReg registers, i.e. transfer size (USER_TXLEN_REG), reset flag to clear the test pattern value (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 the connection mode value (same as Step 4 of Send data test).
5) Set packet size to TOE1G-IP register (TOE_PKL_REG=user input) and calculate total transfer size in each loop. Maximum size of one loop is 4 GB. The operation of each loop is as follows.
   a. Set transfer size of this loop to TOE_TDL_REG. Transfer size is fixed to maximum size (4GB) which is also aligned to packet size, except the last loop. The transfer size of the last loop is equal to the remaining size.
   b. Set send command to TOE1G-IP register (TOE_CMD_REG=0).
   c. Wait until send command is finished by monitoring busy flag (TOE_CMD_REG[0])='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 total received size is equal to the set value from user. 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 the result and the error (same as Step 6 of Receive data test).
8) Calculate performance and show the test result on the console. Go back to step 3 to run the test in forever loop.

3.6   Ping Command Test

The firmware creates the reply from Ping command request. When checksum of the packet is not correct, CPU returns error message on the console. The details of Ping command are follows.

IP Datagram of ICMP protocol for ping command is shown in Figure 3-2.

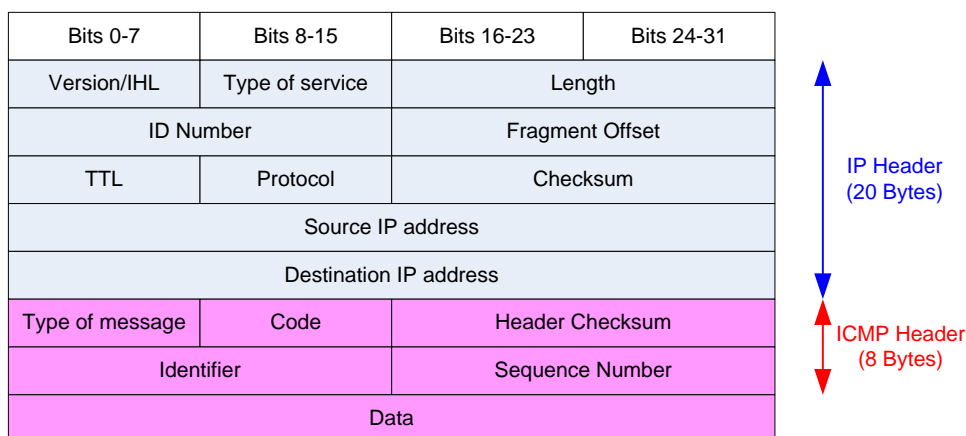| Bits 0-7	Bits 8-15	Bits 16-23	Bits 24-31
Version/IHL	Type of service	Length	
ID Number		Fragment Offset	
TTL	Protocol	Checksum	
Source IP address			
Destination IP address			
Type of message	Code	Header Checksum	
Identifier		Sequence Number	
Data			

IP Header (20 Bytes)

ICMP Header (8 Bytes)

Figure 3-2 IP Datagram for ICMP

For ping command test, TestPC sends Echo request (Type=8) to FPGA, and FPGA generates Echo reply (Type=0) to TestPC. All data of Echo reply are copied from Echo request. More information about ping command is described from following website.
http://en.wikipedia.org/wiki/Ping_(networking_utility)

The step to run this test is as follows.
1) Wait until write pointer (read value of RXMAC_PTR_REG) is not equal to the latest value of the read pointer
2) Set start read address = RXRAM_BASE_ADDR + (read_pointer[2:0])*2Kbyte.
3) Check packet header that checksum in the header is valid. Error message is displayed when the checksum is not correct.
4) Copy received packet to temp buffer in the firmware and prepare Echo reply packet.
5) Calculate IP and ICMP checksum of Echo reply packet, and then copy data from temp buffer to TxRAM.
6) Set TxEMAC register to start data sending by setting TXMAC_ADDR_REG=0, TXMAC_LEN_REG=Echo reply packet length, and TXMAC_CMD_REG[0]='1'.
7) Increase read pointer value or reset to 0 when the pointer is equal to 15.
8) Wait until TxEMAC finishing data transferring by monitoring TXMAC_CMD_REG[31]='0'.
9) Clear finish flag from TxEMAC by setting TXMAC_CMD_REG[31]='1'.
10) Clear finish flag from RxEMAC by setting RXMAC_CMD_REG[31]='1'.

3.7 Function list in User application

This topic describes the function list to run TOE1G-IP operation.

Unsigned int cal_checksum(unsigned int byte_len, unsigned char *buf)	
Parameters	byte_len: total byte for calculating checksum of the data *buf: the address of the 1st data for calculating checksum
Return value	Checksum result after finishing calculation
Description	Calculate 16-bit checksum of the data

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	For active mode, write TOE_CMD_REG to open or close connection. After that, call read_conon function to monitor connection status until it changes from ON to OFF or OFF to ON, depending on port_ctl mode.

void init_param(void)	
Parameters	None
Return value	None
Description	Set network parameters to TOE1G-IP register from global parameters. After reset is de-asserted, it waits until TOE1G-IP busy flag is de-asserted to '0'.

int input_param(void)	
Parameters	None
Return value	0: Valid input, -1: Invalid input
Description	Receive network parameters from user, i.e. mode, window threshold, FPGA MAC address, FPGA IP address, FPGA port number, Target IP address, and Target port number. When the input is valid, the parameters are updated. Otherwise, the value does not change. After receiving all parameters, the current value of each parameter is displayed.

void ping_test(void)	
Parameters	None
Return value	None
Description	Run Ping test following description in topic 3.6.

Unsigned int read_conon(void)	
Parameters	None
Return value	0: Connection is OFF, 1: Connection is ON.
Description	Read value from USER_CMD_CONNON register and return only bit2 value to show connection status.

void show_cursize(void)	
Parameters	None
Return value	None
Description	Read USER_TXLEN_REG and USER_RXLEN_REG and then display in Byte, KByte, or MByte unit

void show_param(void)	
Parameters	None
Return value	None
Description	Display the current value of the network parameters setting to TOE1G-IP such as IP address, MAC address, and port number.

void show_result(void)	
Parameters	None
Return value	None
Description	Read USER_TXLEN_REG and USER_RXLEN_REG to display total size. Read the global parameters (timer_val and timer_upper_val) and calculate total time usage to display in usec, msec, or sec unit. Finally, transfer performance is calculated and displayed on MB/s unit.

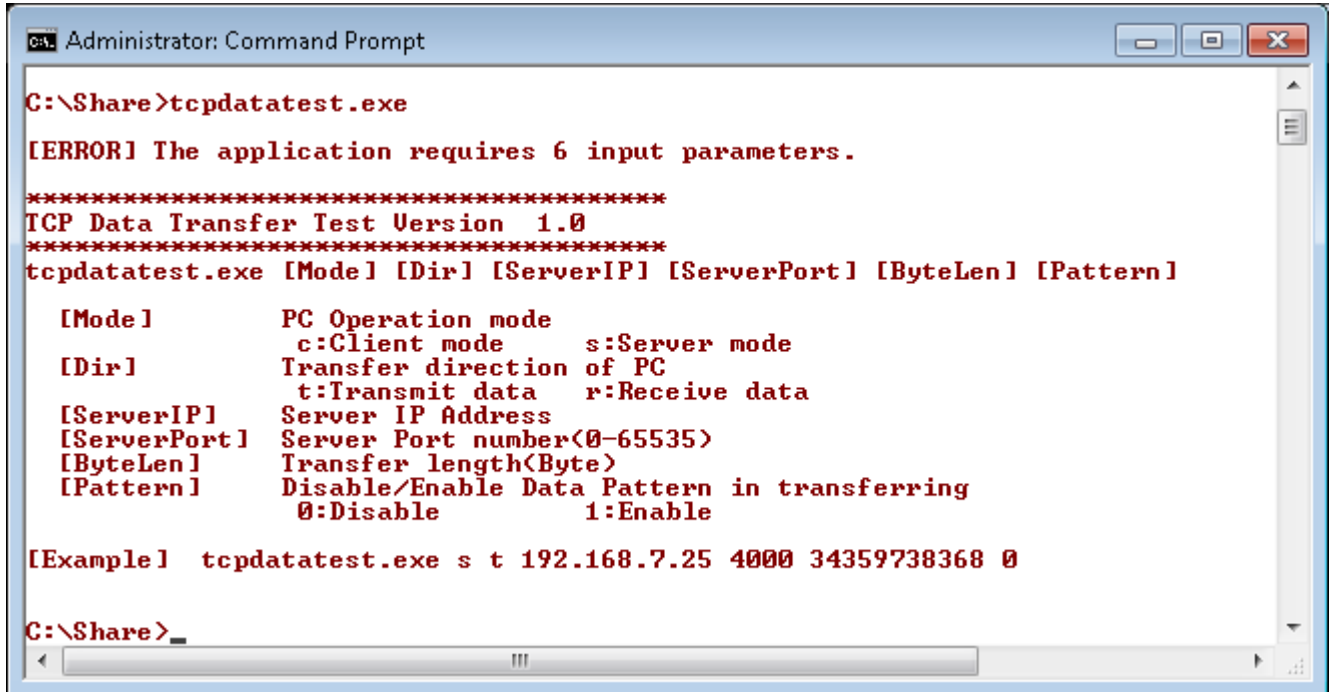| int toe_recv_test(void)	
Parameters	None
Return value	0: The operation is successful -1: Receive invalid input or error is found
Description	Run Receive data test following description in topic 3.4

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

int toe_txrx_test(void)	
Parameters	None
Return value	0: The operation is successful -1: Receive invalid input or error is found
Description	Run Full duplex test following description in topic 3.5

void wait_ethlink(void)	
Parameters	None
Return value	None
Description	Read USER_RST_REG[16] and wait until linkup status is found

4   Test Software Sequence

4.1   "tcpdatatest" for half duplex test

Figure 4-1 "tcpdatatest" application usage

"tcpdatatest" is designed to run on PC for sending/receiving TCP data through Ethernet in both server and client mode. PC of this demo should run in client mode. User inputs parameter 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 destroyed. ByteLen in transmit mode must be equal to the total transfer size on FPGA, set in received 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.

Transmit data mode

Following is the sequence when test application runs in transmit mode.
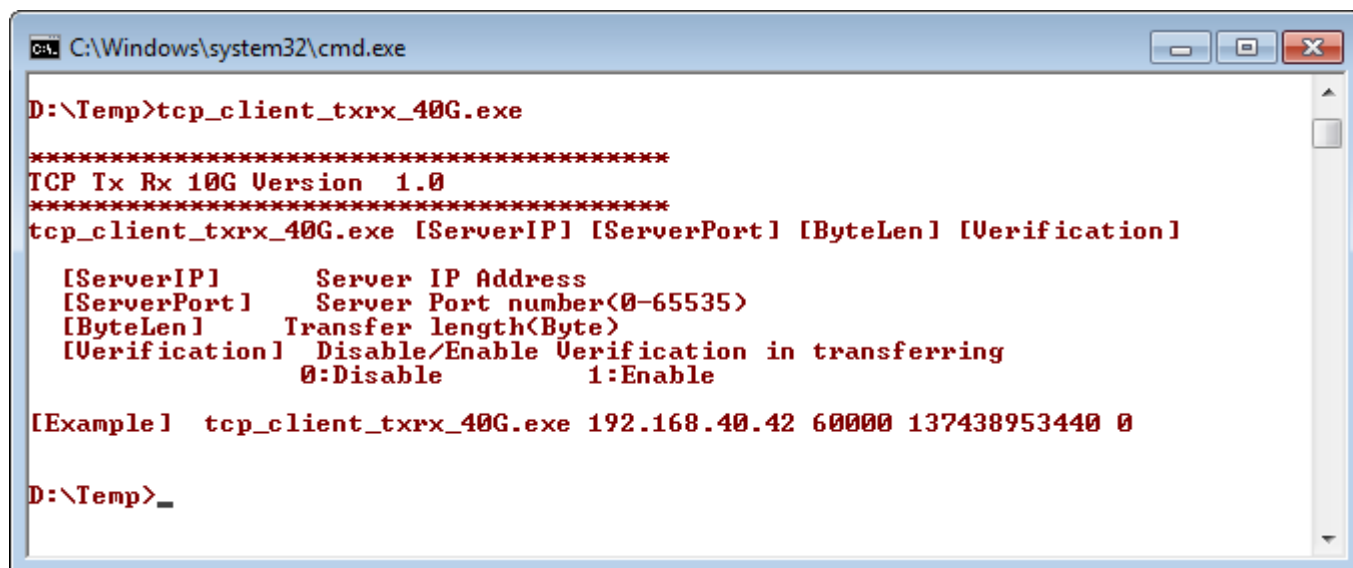
1) Get parameters from the user and verify that the input is valid.
2) Create the socket and set socket options.
3) Create the new connection by using server IP address and server port number.
4) Allocate 1 MB memory to be sent buffer.
5) Generate the incremental test pattern to send buffer when the test pattern is enabled. Skip this step if the dummy pattern is selected.
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 is the sequence when test application runs in receive mode.

1) Follow the step 1) – 3) of Transmit data mode.
2) Allocate 1 MB memory to be received buffer.
3) Read data from the received buffer and increase total received size.
4) If verification is enabled, data is verified by the incremental pattern. Error message is printed out when data is not correct. This step is skipped if data verification is disabled.
5) Print total transfer size every second.
6) Repeat step 3) – 5) until the connection is closed.
7) Calculate total performance and print the result on the console.
8) Close socket and free the memory.

4.2 "tcp_client_txrx_40G" for full duplex test

Figure 4-2 "tcp_client_txrx_40G" application usage

"tcp_client_txrx_40G" application is designed to run on PC for sending and receiving TCP data through Ethernet by using the same port number at the same time. The application is run in client mode, so user needs to input server parameters (the network parameters of TOE1G-IP). As shown in Figure 4-2, there are four parameters to run the application, i.e.

1) ServerIP   : IP address of FPGA
2) ServerPort   : Port number of FPGA
3) ByteLen   : Total transfer byte size. This is total size to transmit and receive 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 sent and received 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 sent size.
(7) Read data from the received buffer and increase total received data size.
(8) If verification is enabled, data is verified by incremental pattern. Error message is printed out when data is not correct. Skip this step if data verification is disabled.
(9) Print total sent and received size every second.
(10) Repeat step 5) – 9) until total sent and received size are equal to ByteLen (set by user).
(11) Calculate performance and print the result on the console.
(12) Close the socket.
(13) Sleep for 1 second to wait until the hardware completes the current test loop.
(14) Run step 3) – 13) in forever loop. If verification is failed, the application is stopped.

5 Revision History

Revision	Date	Description
1.0	3-Dec-19	Initial version release