UDP10G-IP reference design manual

Rev1.2 8-Mar-19

1 Introduction

Comparing to TCP, UDP provides a procedure to send messages with a minimum of protocol mechanism, but the data cannot guarantee to arrive destination because of no handshaking dialogues. Similar to TCP, UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.

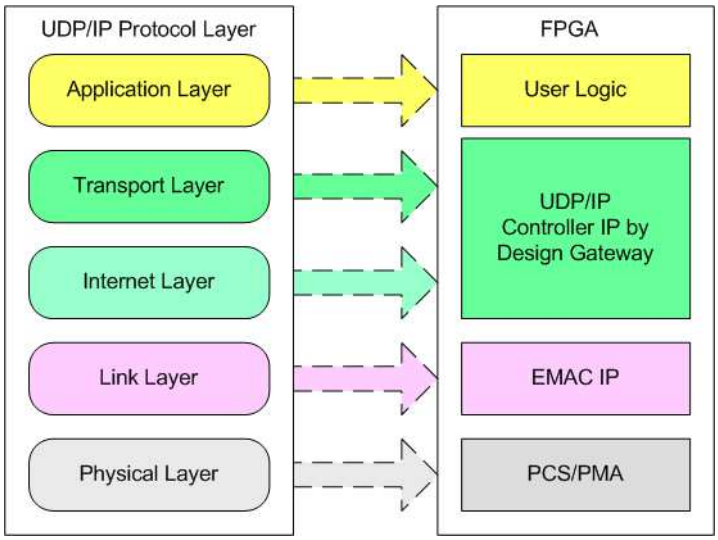

Figure 1-1 UDP/IP Protocol Layer

UDP10G-IP implements Transport and Internet layer of UDP/IP Protocol. For transmit side, UDP10G-IP prepares UDP data from user logic and add UDP/IP header to generate Ethernet packet format before sending out to EMAC. For receive side, UDP10G-IP extracts UDP data from Ethernet packet. UDP/IP header is verified to check packet valid. If packet is valid, UDP data will be extracted and stored in buffer for user logic reading.

The lower layer protocols are implemented by EMAC-IP and PCS/PMA from Xilinx.

This reference design provides evaluation system which includes simple user logic to send and receive data by using UDP10G-IP. This system demonstrates on Xilinx development board to operate with Test application on PC for transferring high speed data on network. More details are described as follows.

In the demo, CPU firmware is bare-metal. User can input the parameters through Serial port and select transfer directions to start test operation. The test application on PC is “udpdatatest.exe”.

2 Hardware description

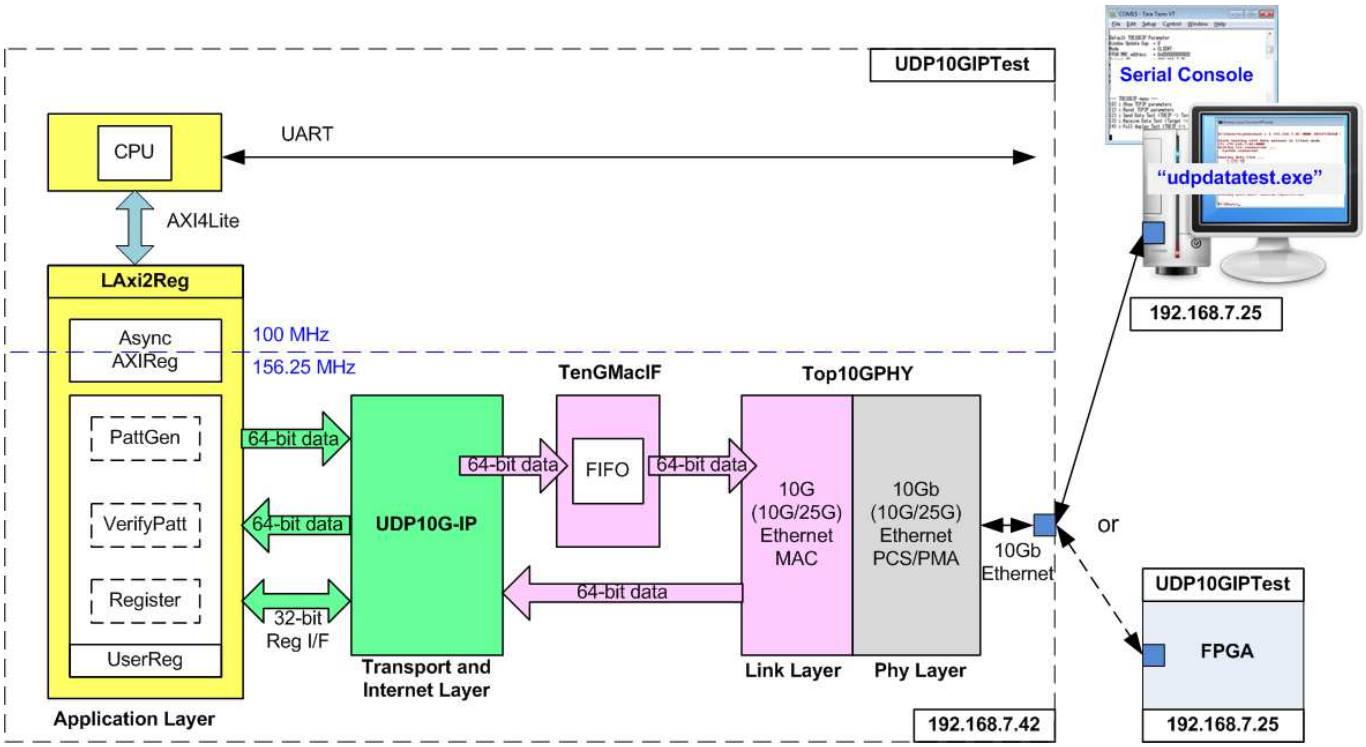

Figure 2-1 Hardware Architecture reference design

As shown in Figure 2-1, hardware architecture can be divided into 4 modules to support each UDP/IP layer protocol. UDP10G-IP operates with EMAC and PCS/PMA to implement all four lower layers of UDP/IP Protocol. UDP/IP does not have handshake packet during data transferring, so data can be transferred by UDP10G-IP as full-duplex with PC at the highest performance like half-duplex mode.

The reference design is designed to show the best performance to transfer data by running half-duplex test. When running full-duplex test by enable both PattGen and VerifyPatt module to send and receive data with UDP10G-IP at the same time, performance of each direction may drop from CPU resource limitation.

For half-duplex transfer in sending mode (FPGA to PC), UDP data is generated by PattGen inside UserReg and Test data is verified by the test application on PC. For received mode (PC to FPGA), data generated by the test application on PC is verified by VerifyPatt in UserReg module.

For control interface, LAxi2Reg includes register to store test parameters from user such as transfer length. User inputs parameters through Serial console. CPU firmware validates all parameters and sets to the hardware through AXI4-Lite bus. Due to the fact that CPU system and UDP10G-IP run in different clock domain, AsyncAXIReg module is used to be asynchronous circuit to support clock-crossing function and convert AXI4-Lite bus signal which is standard bus in CPU system to be register interface. CPU in the demo runs in bare-metal OS. Timer is also included in CPU system for measuring transfer performance.

"udpdatatest.exe" application on PC is designed to send or receive Ethernet data with UDP10G-IP through UDP/IP protocol. Data transferring is half-duplex mode.

2.1 10G (10G/25G) EMAC and PCS/PMA

Both link layer and physical layer of 10G Ethernet are implemented by using Xilinx IP core. 10G (10G/25G) EMAC implements the link layer while 10G (10G/25G) Ethernet PCS/PMA implements the physical layer. Data path of EMAC is 64-bit AXI4 stream interface.

Tx interface timing diagram of Xilinx EMAC and UDP10G-IP are different. UDP10G-IP needs to send data of one packet continuously, but Xilinx EMAC does not support this feature. Xilinx EMAC may de-assert ready signal to receive data during packet transferring (between start-of-frame and end-of-frame). TenGMacIF is designed to store transmitted data from UDP10G-IP when Xilinx EMAC is not ready to receive new data.

Furthermore, 10G/25G EMAC does not include zero padding function. TenGMacIF for connecting with 10G/25G EMAC Subsystem needs to add zero padding when transmitted packet size from UDP10G-IP is less than 60 bytes. More details of 10G (10G/25G) EMAC and PCS/PMA are described in following link.

10G Ethernet MAC and PCS/PMA
https://www.xilinx.com/products/intellectual-property/do-di-10gemac.html
https://www.xilinx.com/products/intellectual-property/10gbase-r.html

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

2.2 UDP10G-IP

UDP10G-IP implements UDP/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.
https://dgway.com/products/IP/UDP10G-IP/dg_udp10gip_data_sheet_xilinx_en.pdf

2.3 TenGMacIF

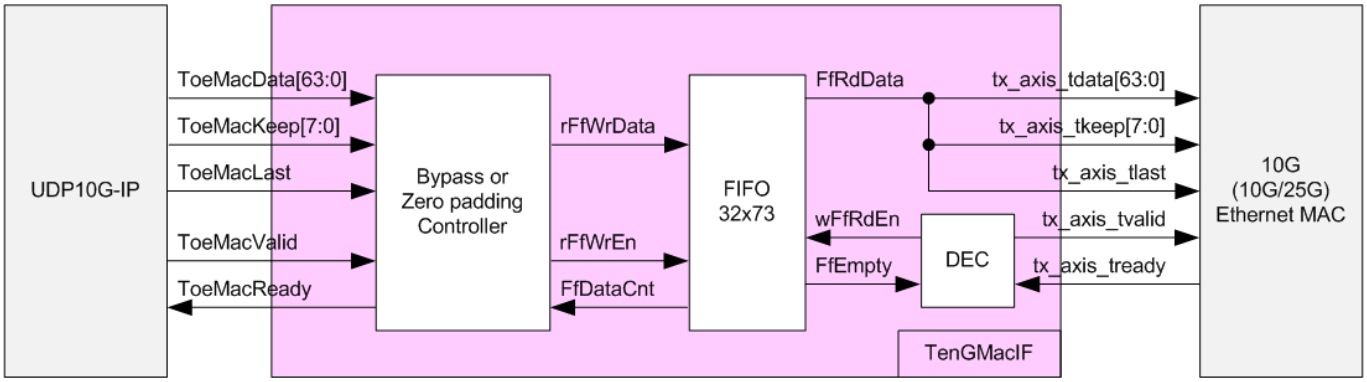

Figure 2-2 TenGMacIF Block Diagram

This module is designed to be adapter logic connecting between Tx interface of UDP10G-IP and Tx interface of 10G Ethernet MAC. ToeMacReady output to UDP10G-IP must be always asserted to '1' when transferring the packet (between start-of-frame and end-of-frame). According to Xilinx 10G Ethernet MAC specification, tx_axis_tready of Xilinx EMAC may be de-asserted to '0' during transferring the packet. So, timing diagram of UDP10G-IP and Xilinx EMAC does not match and cannot connect directly. TenGMacIF including small FIFO must be designed to store the data from UDP10G-IP when 10G Ethernet MAC is not ready to receive new data.

Otherwise, 10G/25G Ethernet Subsystem does not include zero padding function. So, TenGMacIF needs to add zero data to the packet which the size is less than 60 bytes. ToeMacReady output to UDP10G-IP for normal packet is designed by following sequence.

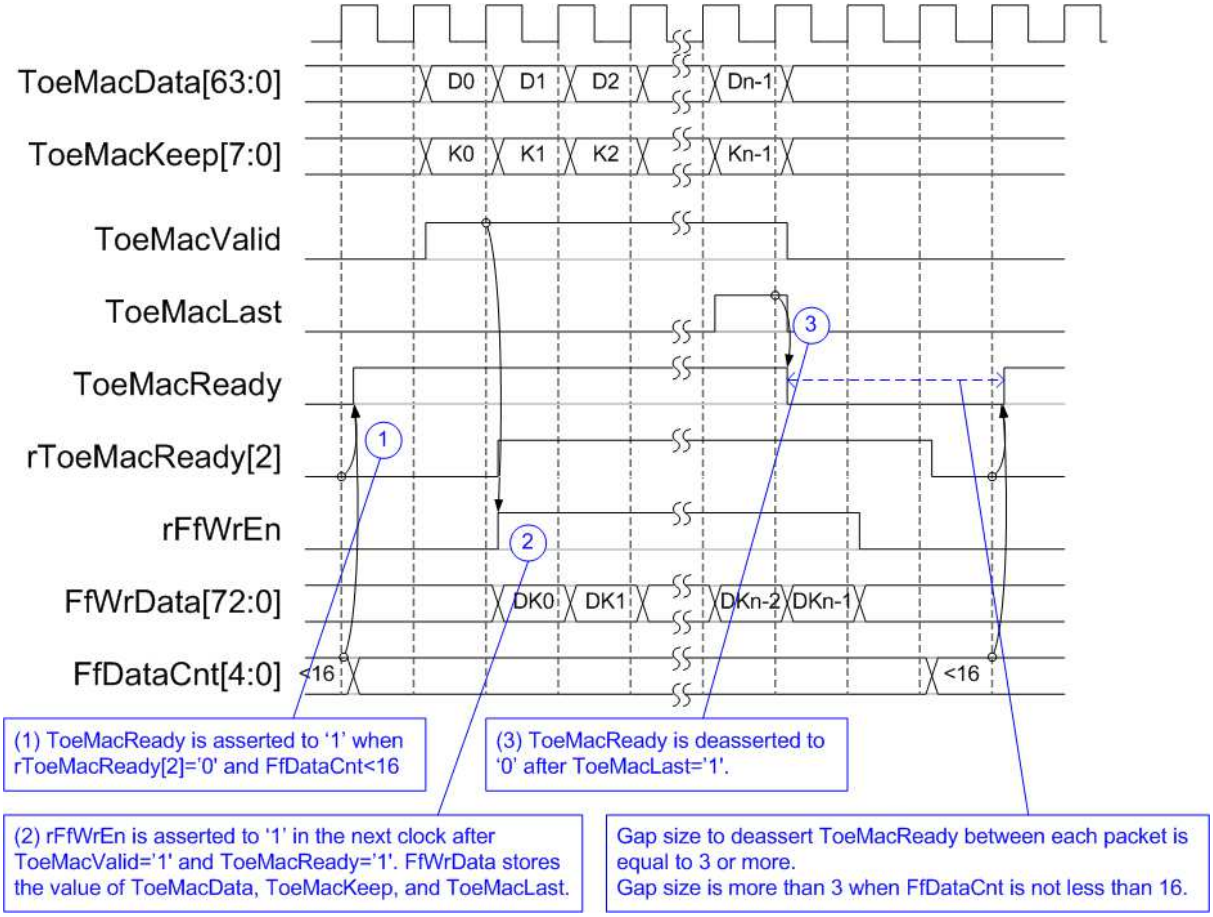

Figure 2-3 Write FIFO timing diagram of TenGMacIF

As shown in Figure 2-3, the new packet is received when two conditions are met. First condition is free space in FIFO must be much enough (FfDataCnt is less than 16). Second condition is the gap size from previous packet is more than or equal to 3, monitored by rToeMacReady[2] which is 2-clock delayed from ToeMacReady. To receive the new packet, ToeMacReady is asserted to '1'. After that, the new packet is received from UDP10G-IP continuously because ToeMacReady is always asserted to '1' until end of packet. ToeMacReady is de-asserted to '0' after last data is received (ToeMacLast is asserted to '1'). When ToeMacReady is de-asserted to '0', the new packet must pause data transmission.

Internal FIFO has 73-bit data bus to store 64-bit data (ToeMacData), 8-bit byte enable signal (ToeMacKeep), and last flag (ToeMacLast). The depth of internal FIFO is the limitation that how many clock cycles which Ethernet MAC deasserting tx_axis_tready to '0' within one packet. In the reference design, depth of FIFO is equal to 32, so maximum busy clock cycles per one packet to deassert tx_axis_tready is 32.

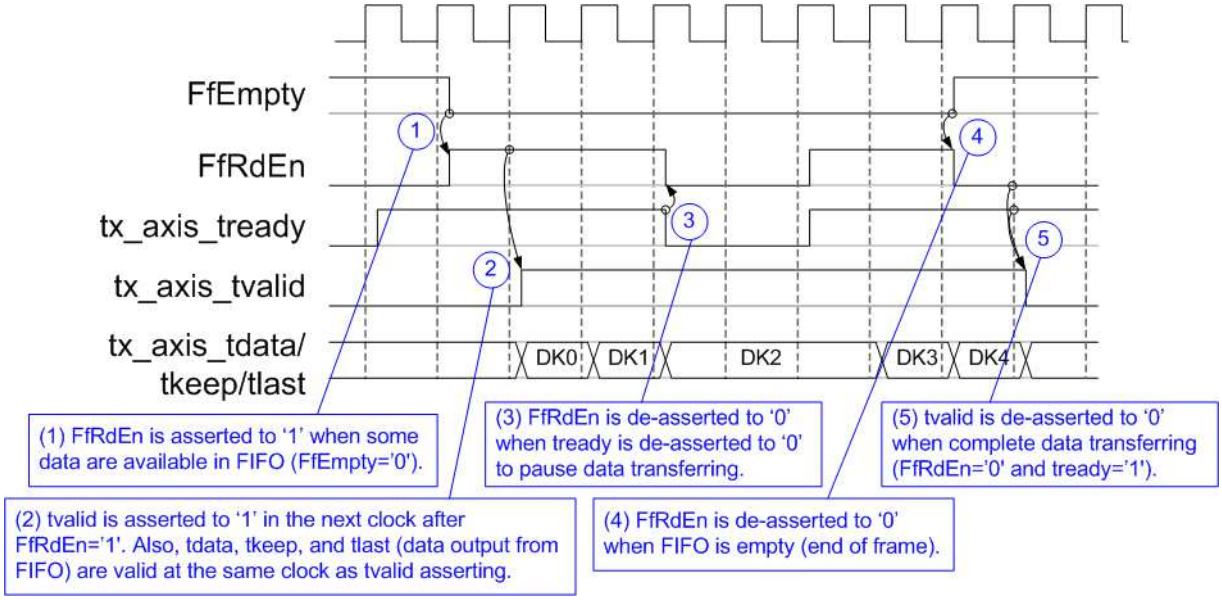

Figure 2-4 Read FIFO timing diagram of TenGMacIF

As shown in Figure 2-4, when data is available in FIFO (FfEmpty='0'), FfRdEn is asserted to '1' to transfer data from FIFO to Ethernet MAC. In the next clock, tx_axis_tvalid is asserted to '1' with the valid data (tdata, tkeep, and tlast). When EMAC is not ready to receive data (tx_axis_tready='0'), FfRdEn will be de-asserted to '0' to pause data transmission. FIFO is empty when all packet in FIFO are transferred completely (write interface is stopped transmission). FfRdEn is de-asserted to '0' when FfEmpty is equal to '1'. After that, tx_axis_tvalid is de-asserted to '0' to complete one packet transferring.

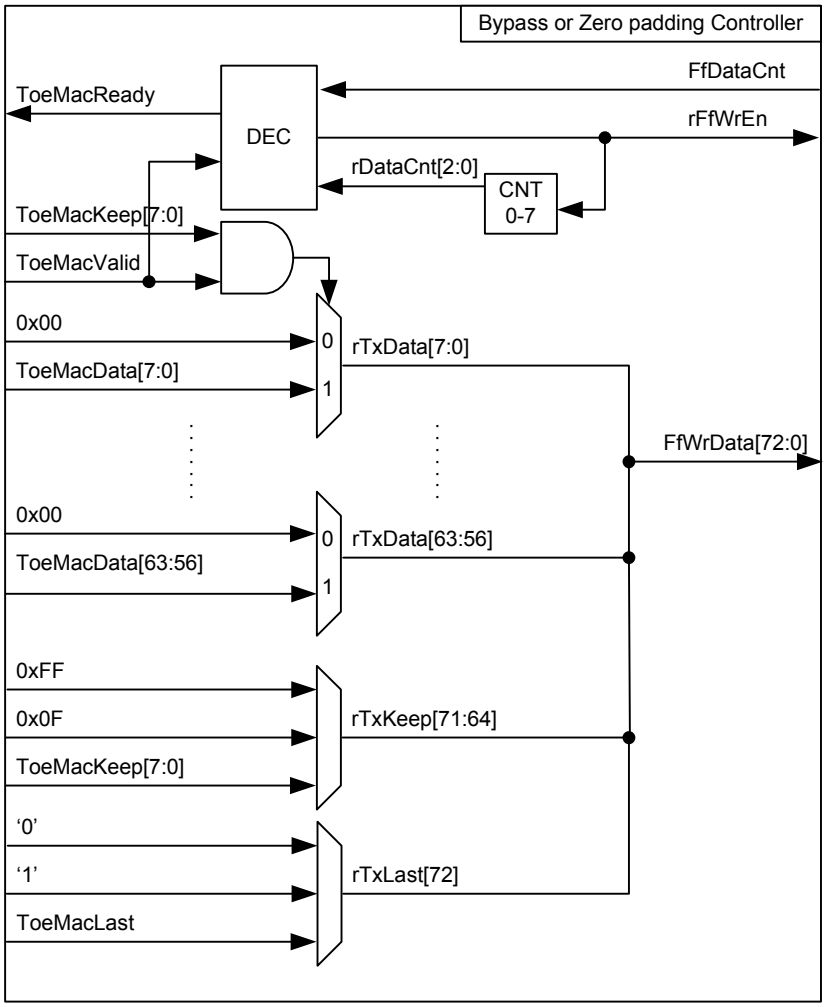

Figure 2-5 Data path of zero padding function

Figure 2-5 shows the logic to fill zero padding data when total packet size is less than 60 bytes. This feature is necessary when using 10G/25G Ethernet system. 3-bit data counter is designed to count data size whether data less than 60 bytes or not. rFfWrEn is asserted to '1' to fill zero padding data until data counter=7 (packet size is more than or equal to 60 bytes). rTxData is selected between data signal from UDP10G-IP or zero value for zero-padding. Each bit of ToeMacKeep is used to be byte valid of ToeMacData and used to select data to forward to FIFO between ToeMacData or 0x00.

rTxKeep is fixed to 0xFF for QWord 0-6. QWord 7 of the packet could be selected by two sources. First is equal to 0x0F to generate the smallest packet with zero-padding feature (60 byte data). Second is bypass signal from ToeMacKeep when packet size is more than 60 byte. The data after QWord8 is forwarded from UDP10G-IP directly to generate normal size packet.

Similar to rTxKeep, rTxLast is fixed to '0' for QWord 0-6. For small packet which requires zero-padding feature, rTxLast is asserted to '1' at QWord 7. In case of normal size packet (more than 60 byte), rTxLast is forwarded from ToeMacLast.

2.4 LAxi2Reg

The hardware is connected to CPU through AXI4-Lite bus, similar to other CPU peripherals. Table 2-1 shows CPU memory map of the register inside LAxi2Reg. Test parameters using in the system are received by register write operation. CPU monitors the status signals within the system by register read operation.

As shown in Figure 2-6, there are three parameter sets in the system which is controlled by CPU, i.e. UserReg parameters for sending/receiving test data with UDP10G-IP and UDP10G-IP parameters.

LAxi2Reg has two clock domains, i.e. CpuClk (100 MHz) for interface with CPU through AXI4-Lite bus and MacTxClk (156.25 MHz) for interface with UDP10G-IP and 10G EMAC. The module to convert signals from one clock to another clock is AsyncAxiReg.

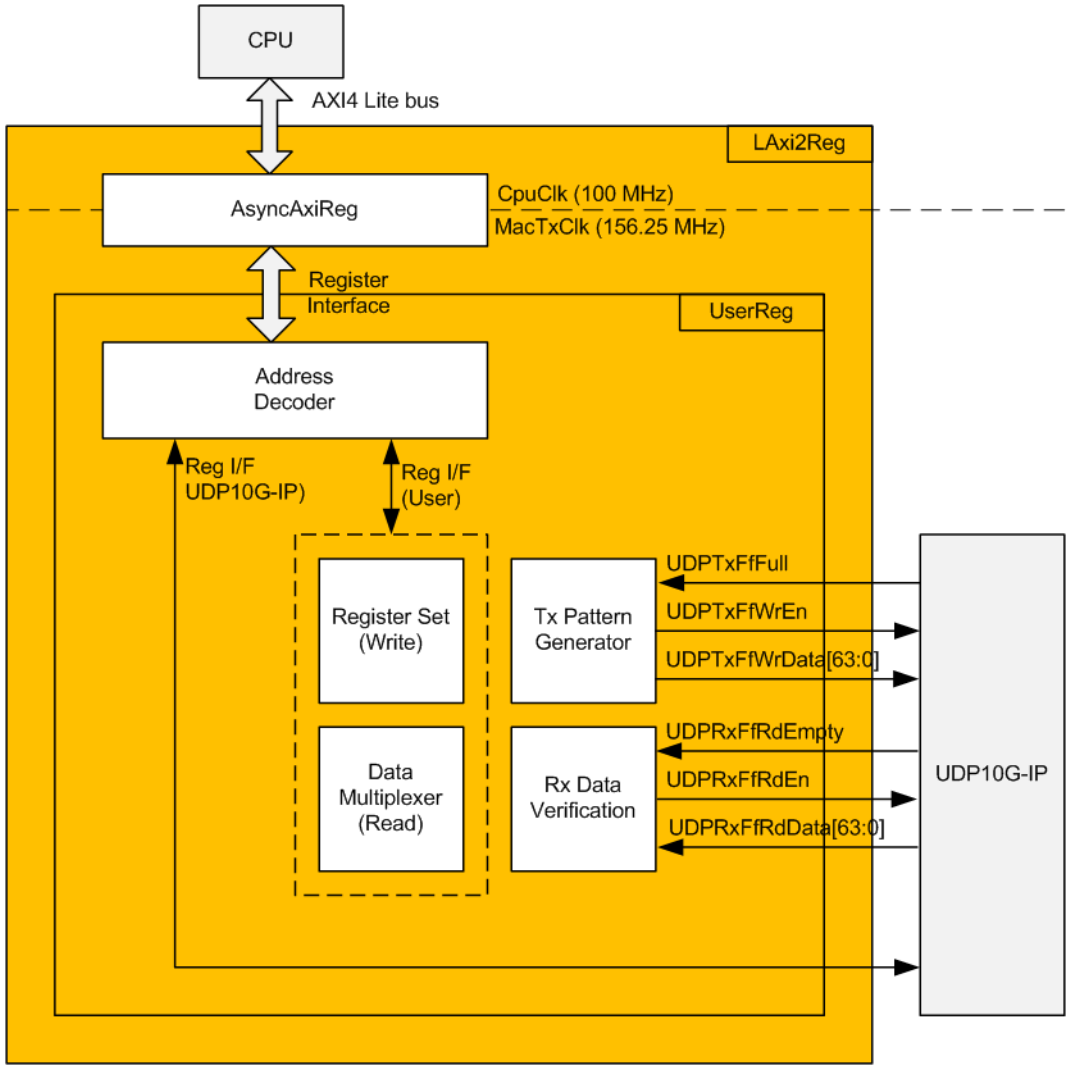

Figure 2-6 LAxi2Reg block diagram

2.4.1 AsyncAxiReg

This module is designed to convert signal interface of AXI4-Lite to be register interface. Also, it includes asynchronous circuit to transfer signal from one clock domain to another clock domain. Timing diagram of register interface is shown in Figure 2-7.

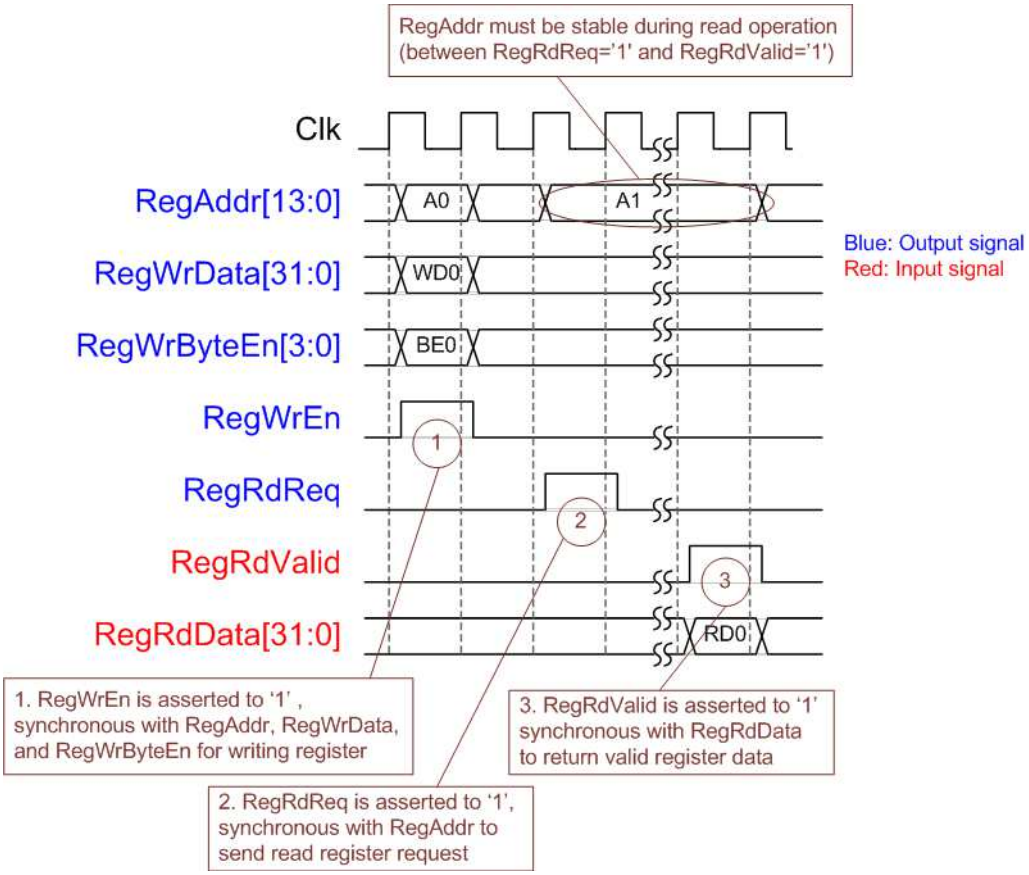

Figure 2-7 Register interface timing diagram

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, AsyncAxiReg asserts RegRdReq='1' with the valid value of RegAddr (the register address in 32-bit unit). RegAddr must hold the same value until RegRdValid is asserted to '1'. Read data is available on RegRdData with asserting RegRdValid to '1' for one clock cycle.

2.4.2 UserReg

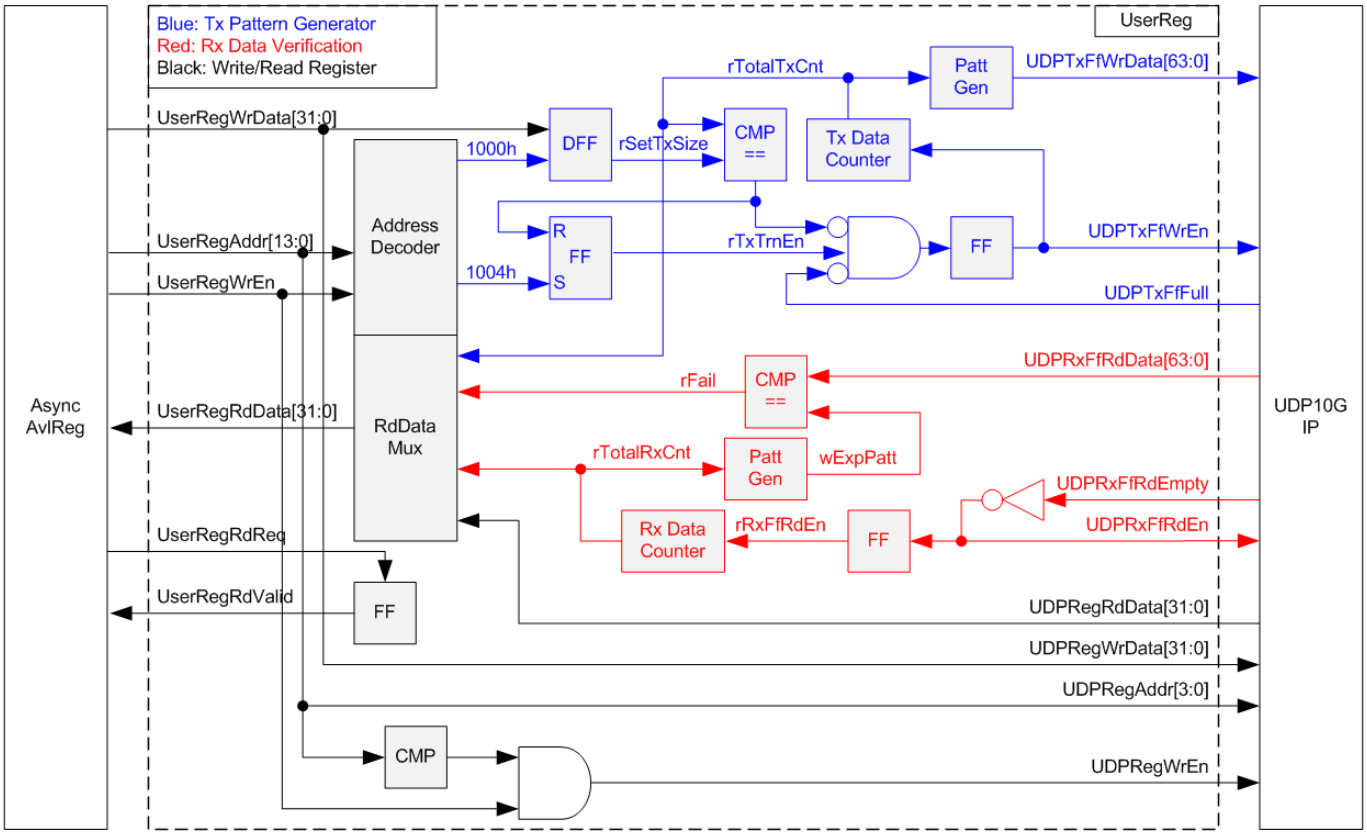

Figure 2-8 UserReg block diagram

Memory map of control and status signals inside UserReg module is shown in Table 2-1.
0x0000 – 0x00FF is mapped to registers inside UDP10G-IP.
0x1000 – 0x10FF is mapped to registers for 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 applied to decode the destination module to be UDP10G-IP or UserReg register. If CPU accesses UDP10G-IP area, UDPRegWrEn will be asserted to '1'. The lower bits of UserRegAddr are passed to all modules to decode the parameter within each module. For example, rSetTxSize is loaded by UserRegWrData when UserRegAddr=0x1000. UserRegWrByteEn signal is not used in other modules. To access any registers, CPU firmware needs to use 32-bit pointer only.

For read request, UserRegRdReq is asserted to '1'. RdDataMux selects status signals from internal register or UDP10G-IP, following 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 UDP10G-IP. rTxTrnEn is asserted to '1' when write register address is 1004h. When rTxTrnEn is '1', UDPTxFfWrEn is controlled by UDPTxFfFull. UDPTxFfWrEn is de-asserted to '0' when UDPTxFfFull is '1'. rTotalTxCnt is data counter to check data sending to UDPTxFf. rTotalTxCnt is also used to generate 32-bit increment data to UDPTxFfWrData signal. rTxTrnEn is de-asserted to '0' when total data has been transferred completely (total data is set by rSetTrnSize).

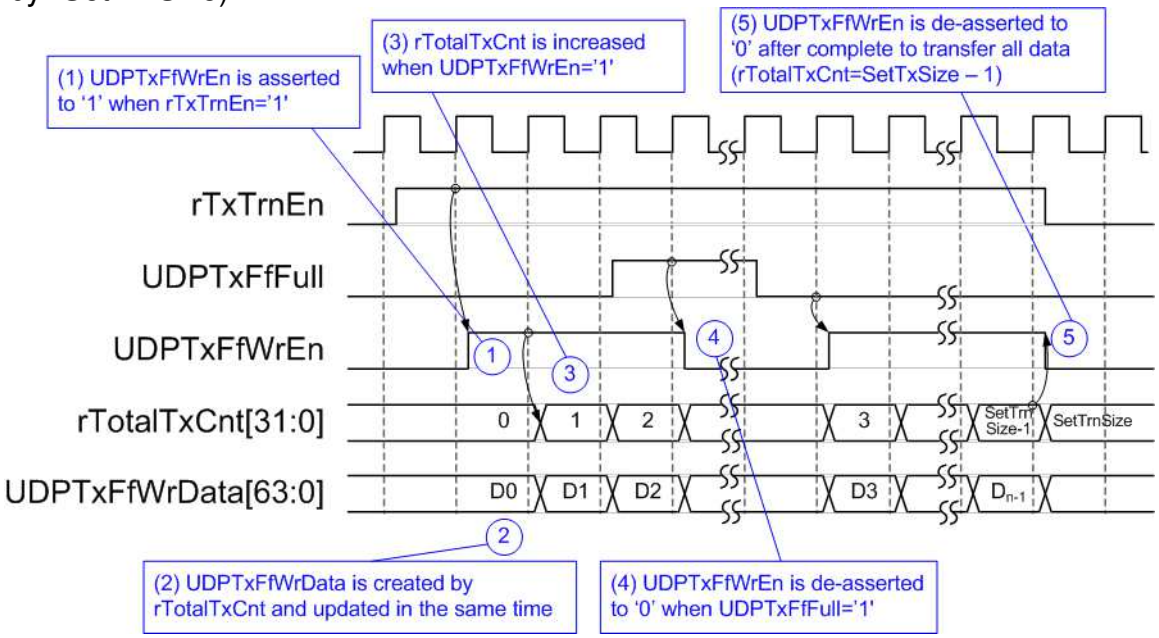

Figure 2-9 Tx Pattern Generator Timing diagram

The logic in red color of Figure 2-8 is designed to verify received data from UDP10G-IP. UDPRxFfRdEn is designed by using NOT logic of UDPRxFfRdEmpty. UDPRxFfRdData is valid in the next clock after asserting UDPRxFfRdEn to '1'. Read data (UDPRxFfRdData) is compared to expected pattern (wExpPatt) which is designed by rTotalRxCnt when rFfRdEn='1'. rTotalRxCnt is data counter to check total data transferring from UDPRxFf. 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.

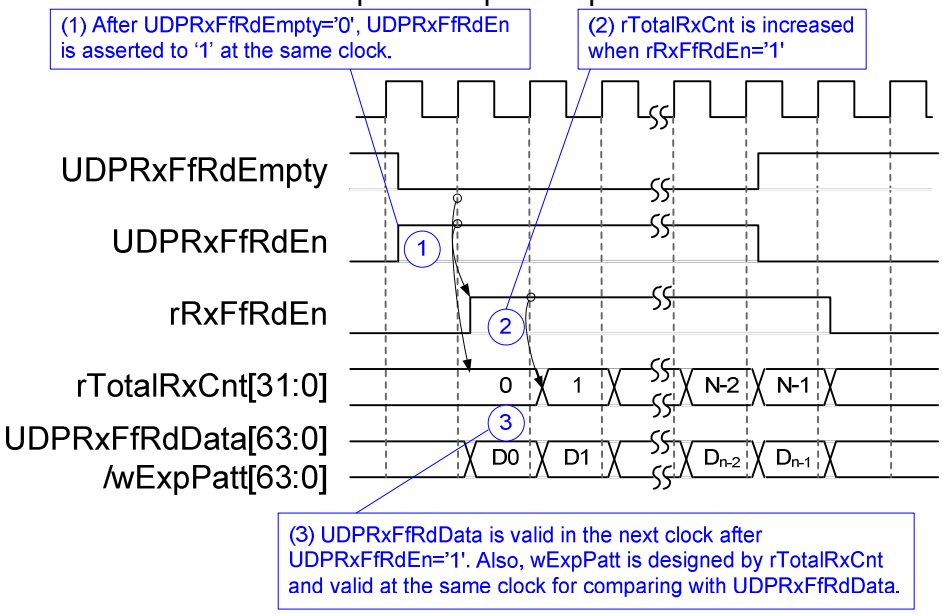

Figure 2-10 Rx Data Verification Timing diagram

Table 2-1 Register map Definition

Address Wr/Rd	Register Name (Label in the “udp10gtest.c”)	Description
BA+0x0000 – BA+0x00FF: UDP10G-IP Register Area More details of each register are described in Table2 of UDP10G-IP datasheet.		
BA+0x0000	UDP10_RST_REG	Mapped to RST register within UDP10G-IP
BA+0x0004	UDP10_CMD_REG	Mapped to CMD register within UDP10G-IP
BA+0x0008	UDP10_SML_REG	Mapped to SML register within UDP10G-IP
BA+0x000C	UDP10_SMH_REG	Mapped to SMH register within UDP10G-IP
BA+0x0010	UDP10_DIP_REG	Mapped to DIP register within UDP10G-IP
BA+0x0014	UDP10_SIP_REG	Mapped to SIP register within UDP10G-IP
BA+0x0018	UDP10_DPN_REG	Mapped to DPN register within UDP10G-IP
BA+0x001C	UDP10_SPN_REG	Mapped to SPN register within UDP10G-IP
BA+0x0020	UDP10_TDL_REG	Mapped to TDL register within UDP10G-IP
BA+0x0024	UDP10_TMO_REG	Mapped to TMO register within UDP10G-IP
BA+0x0028	UDP10_PKL_REG	Mapped to PKL register within UDP10G-IP
BA+0x0038	UDP10_SRV_REG	Mapped to SRV register within UDP10G-IP
BA+0x1000 – BA+0x10FF: UserReg control/status		
BA+0x1000 Wr/Rd	Total transmit length (USER_TXLEN_REG)	Wr [31:0] – Total transmit size in QWord unit (64-bit). Valid from 1-0xFFFFFFFF. Rd [31:0] – Current transmit size in QWord unit (64-bit). 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 ‘0’ to start transmitting. [1] – Data Verification enable (‘0’: Disable data verification, ‘1’: Enable 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.
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’. [16] – Ethernet Linkup status (‘0’: Link down, ‘1’: Link up)
BA+0x100C Rd	FIFO status (USER_FFSTS_REG)	Rd [2:0]: Mapped to UDPRxFfLastRdCnt signal of UDP10G-IP [15:3]: Mapped to UDPRxFfRdCnt signal of UDP10G-IP [24]: Mapped to UDPTxFfFull signal of UDP10G-IP
BA+0x1010 Rd	Total receive length (USER_RXLEN_REG)	Rd [31:0] – Current received size in QWord unit (64-bit). 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. Two modes could be selected, i.e. client or server. The operation mode is the set value for UDP10_SRV_REG register.

(a) In client mode, FPGA sends ARP request to get the MAC address from the target during initialization sequence.
(b) In server mode, FPGA waits ARP request from the target and returns ARP reply during initialization sequence.

When test environment consists of two FPGAs, the 1st FPGA must run as server mode and another FPGA must run as client mode. User needs to start server FPGA initialization before client FPGA initialization.

When test environment consists of FPGA and Test PC, FPGA could be set to initialized as client mode. Server mode is not recommended because it is not easy to force PC to generate ARP request to FPGA in initialization phase.

In the firmware, there are two sets of default parameters following the mode, i.e. client parameters and server parameters. 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 which is described in topic 3.2.
3) CPU waits until UDP10G-IP completes initialization sequence (UDP10_CMD_REG[0]='0').
4) Main menu is displayed with five operations. More details of each operation are described as follows.

3.1 Show parameters

This menu is used to show current parameters of UDP10G-IP such as operation mode, FPGA MAC address, FPGA IP address, and FPGA port number. The sequence to 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 UDP10G-IP parameters such as IP address, source port number. After setting UDP10G-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 initialization mode from user and confirm that input is valid. If initialization mode is changed, the latest parameter set of new mode will be displayed on the console.
3) Receive remaining input parameters from user and check input whether it is valid or not. If the input is invalid, the parameter will not be updated.
4) Force reset to UDP10G-IP by setting UDP10_RST_REG[0]='1'.
5) Set all parameters to UDP10G-IP register such as UDP10_SML_REG, UDP10_DIP_REG.
6) De-assert UDP10G-IP reset by setting UDP10_RST_REG[0]='0'.
7) Reset UserReg by setting USER_RST_REG[0]='1'.
8) Monitor UDP10G-IP busy flag (UDP10_CMD_REG[0]). Wait until busy flag is de-asserted to '0' to confirm that initialization sequence is completed.

3.3 Send data test

User needs to input two parameters, i.e. total transmit length and packet size. The operation will be cancelled if the input is invalid. 32-bit increment data is generated from the logic and sent to the target (PC or FPGA). The received data is verified by the target (PC or FPGA). After all data are transferred completely, the operation is exit. More details of this menu are described as follows.

1) Receive transfer size and packet size from user and verify that all inputs are valid.
2) Set UserReg registers, i.e. transfer size (USER_TXLEN_REG), reset flag to clear initial value of test pattern (USER_RST_REG), and command register to start data pattern generator (USER_CMD_REG=0). After that, test pattern generator in UserReg starts to generate test data to UDP10G-IP.
3) Display recommended parameter of test application running on PC by reading current parameters in the system. Wait until user press any key to start IP sending operation.
4) Set parameters to UDP10G-IP to start operation. Packet size is set to UDP10_PKL_REG and total size is set to UDP10_TDL_REG. Finally, UDP10_CMD_REG is set to 1 to start IP sending data.
5) Wait until UDP10G-IP completes operation by monitoring IP busy flag (UDP10_CMD_REG[0] ='0'). During waiting, CPU reads current transfer size from user logic (USER_TXLEN_REG) and displays on the console every second.
6) Calculate performance and show test result on the console.

3.4 Receive data test

In receive data test, user sets total received size and data verification mode (enable or disable). 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 and data verification mode from user and verify that all inputs are valid.
2) Set UserReg registers, i.e. reset flag to clear initial value of test pattern (USER_RST_REG) and data verification mode (USER_CMD_REG[1]='0' or '1').
3) Display recommended parameter of test application running on PC by reading current parameters in the system.
4) Wait until IP receives the first packet by monitoring current received size (USER_RXLEN_REG) not equal to '0'. Start timer after receiving the first packet.
5) Wait until received size (USER_RXLEN_REG) does not change more than 100 msec or total data are received. During waiting, CPU reads current received size from user logic (USER_RXLEN_REG) and displays on the console every second.
6) Stop timer. Check interrupt from timeout (USER_RST_REG[8]) and data verification flag (USER_CMD_REG[1]) register when verification mode is applied. If some errors are found, 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 at the same time. Three inputs are received from user, i.e. total size for both directions, packet size for FPGA sending logic, and data verification mode for FPGA receiving logic.

To run full duplex test by using PC, user opens “udpdatatest” application on two consoles. First application is used to receive data with FPGA and another application is used to send data with FPGA. The port using in two applications must be different. In case of two FPGAs test environment, one port is applied to set to each FPGA. The sequence of this test is as follows.

1) Receive total data size, packet size, and data verification mode from user and verify that all inputs are valid.
2) Set UserReg registers, i.e. transfer size (USER_TXLEN_REG), reset flag to clear initial value of test pattern (USER_RST_REG), and command register to start data pattern generator with data verification mode (USER_CMD_REG=1 or 3).
3) Display recommended parameter of test application running on PC by reading current parameters in the system.
4) Set UDP10G-IP registers, i.e. packet size (UDP10_PKL_REG), total transfer size (UDP10_TDL_REG), and write command (UDP10_CMD_REG=1). IP starts sending data operation after UDP10_CMD_REG is set to 1. For receiving data, IP is always ready to receive data without additional setting.
5) Wait until operation is completed for both sending and receiving direction.
   a. For sending direction, wait until busy flag of UDP10G-IP (UDP10_CMD_REG[0])=’0’.
   b. For receiving direction, wait until total received size is equal to set value or total received size does not change for 100 msec (timeout condition)

   During waiting, CPU reads current transfer size of both directions from user logic (USER_TXLEN_REG and USER_RXLEN_REG) and displays on the console every second.
6) Check interrupt from timeout (USER_RST_REG[8]) and data verification flag (USER_CMD_REG[1]) register when verification mode is applied. If some errors are found, error message will be displayed.
7) Calculate performance and show test result on the console.

4 Test Software description

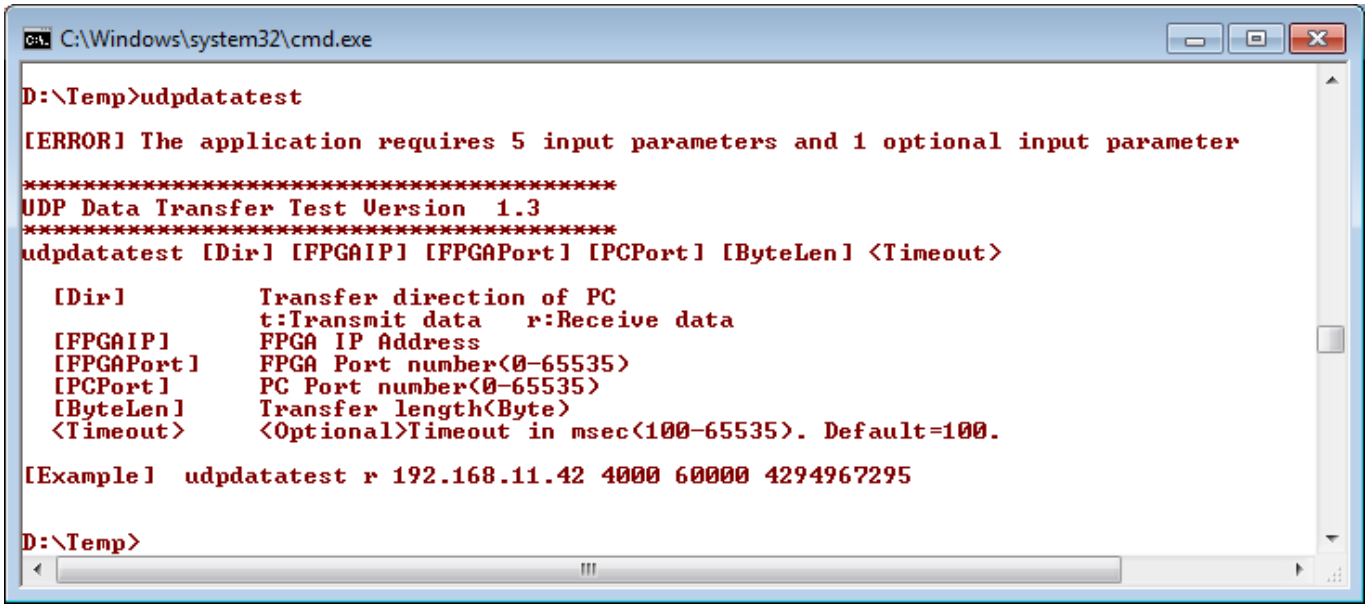

Figure 4-1 udpdatatest application parameter

"udpdatatest" is an application on PC for sending or receiving UDP data. There are five parameters and one optional parameter. The parameter input should be matched to parameter setting on FPGA. More details of each parameter input are as follows.

1) Dir: t – when PC sends data to FPGA
   r – when PC receives data from FPGA
2) FPGAIP : IP address setting on FPGA (default value in is 192.168.7.42)
3) FPGAPort : Port number of FPGA (default value in FPGA is 4000)
4) PCPort : PC port number for sending or receiving data (default is 60001 for PC to FPGA and 60000 for FPGA to PC)
5) ByteLen : Transfer length for sending or receiving in byte unit. This value must be aligned to 8 from UDP10G-IP limitation.
6) Timeout : Timeout for receiving data in msec unit. It is recommended to use 100 for running with UDP10G-IP. Default value in software when user does not input this parameter is 100.

Receive data mode

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

1) Get parameters from user.
2) Create socket and then set properties of received buffer.
3) Set IP address and port number from user parameter, and then connect.
4) Loop to verify data until total data is equal to set value or no more data is received until timeout. Verification pattern is 32-bit increment starting from 0, so test pattern is increased every 4-byte received data. During running, application prints total received size on the console every second.
5) In case of timeout condition, “Timeout” message is displayed with total lost size and total received size when end of operation.

Transmit data mode

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

1) Step (1)-(3) are same as Receive data mode.
2) Generate 32-bit increment pattern to buffer and then send data out. The packet size is fixed to 1472 byte.
3) After completing to send all data, the application displays performance with total data size as test result.

5 Revision History

Revision	Date	Description
1.0	14-Sep-17	Initial Release
1.1	17-Nov-17	Correct Receive data test sequence
1.2	8-Mar-19	Support FPGA<->FPGA connection