UDP100G-IP reference design

Rev1.0 4-Aug-21

1 Introduction

Comparing to TCP protocol, UDP protocol provides a procedure to send data with a minimum of protocol mechanism. There is no handshake and no data recovery process for the sender to confirm that the receiver accepts all data correctly. Similar to TCP protocol, UDP protocol provides checksum for data integrity and port numbers for addressing different functions at the source and the destination in the networks.

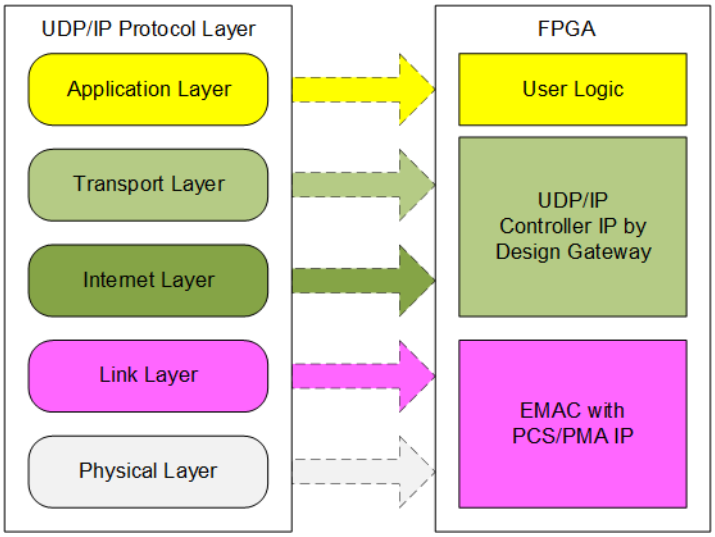

Figure 1-1 TCP/IP protocol layer

UDP100G-IP implements Transport and Internet layer of UDP/IP Protocol for building Ethernet packet from the user data (UDP payload data) to EMAC. If UDP payload data size is larger than a packet size, UDP100G-IP splits the data from the user to smaller size to fit in one packet. After that, the payload data is appended by UDP/IP header. On the other hand, the received Ethernet packet from EMAC is extracted by UDP100G-IP. The header of the packet is verified. If the header is valid, UDP payload data is forwarded to the user logic. Otherwise, the packet is rejected.

The low-layer protocols are implemented by 100G Ethernet IP, including EMAC and PCS/PMA logic. The reference design uses 100G Ethernet Subsystem, provided by Xilinx.

The reference design provides the evaluation system which includes simple user logic to transfer data by using UDP100G-IP. UDP100G-IP transfers data with PC or another UDP100G-IP on another FPGA board. To run with PC, the test application, udpdatatest, is called on PC to send and verify UDP payload data via Ethernet connection at very high-speed rate. One application is called for transferring data in one direction. To run full-duplex test, two test applications are called for sending and receiving data.

To allow the user controlling the test parameters and the operation of UDP100G-IP demo via UART/JTAGUART, the CPU system is included. It is easy for the user to set the test parameters and monitor the current status on the console. The firmware on CPU is built by using bare-metal OS. More details of the demo are described as follows.

2 Hardware overview

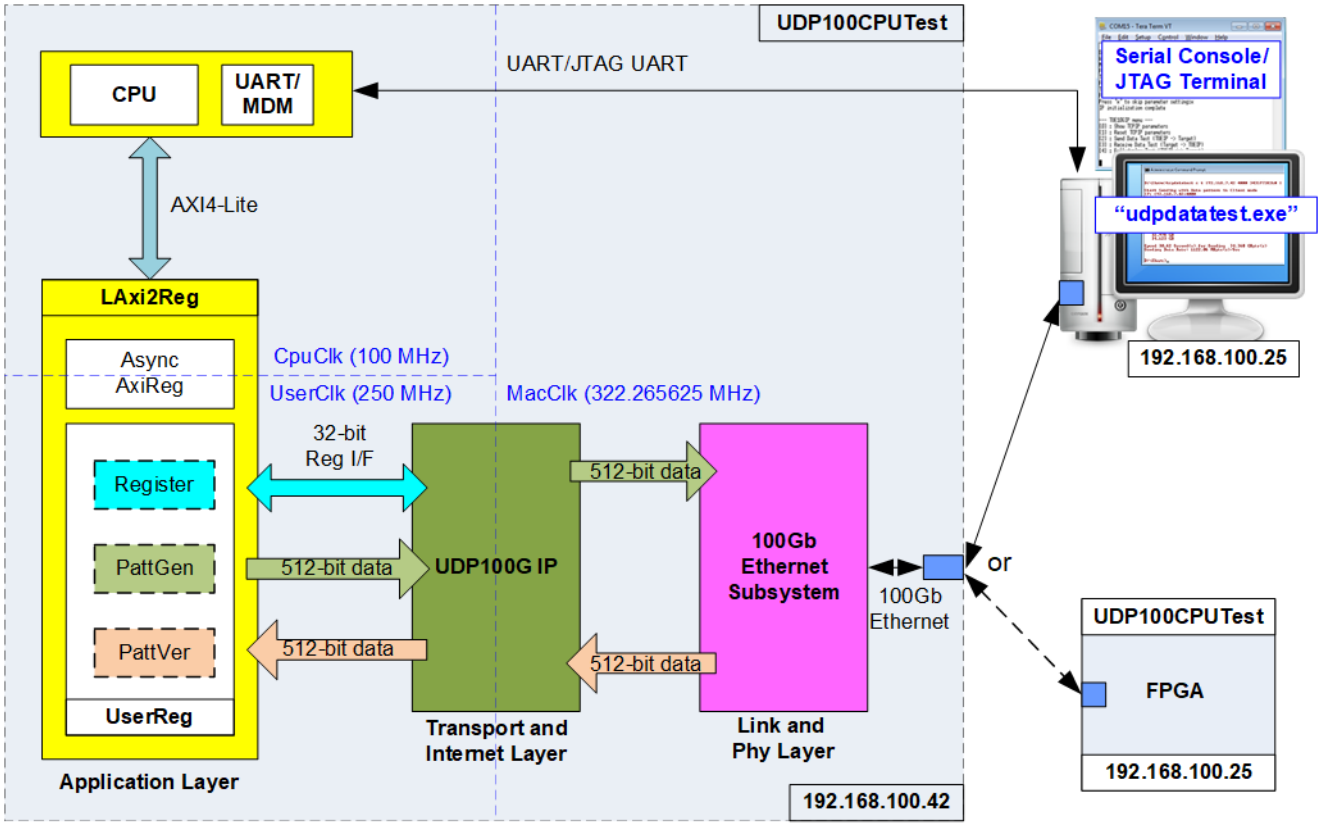

Figure 2-1 Demo block diagram

In test environment, two devices are used for 100Gb Ethernet transferring. First device is UDP100G-IP which may be initialized by Client or Fixed-MAC mode. The second device may be Test PC which runs "udpdatatest" or another FPGA including UDP100G-IP that is initialized by Server or Fixed-MAC mode, as shown in Figure 2-1.

In FPGA system, UDP100G-IP connects with 100G Ethernet Subsystem to complete all UDP/IP layer implementation. User interface of UDP100G-IP connects to UserReg within LAxi2Reg which consists of Register file for interfacing with Register interface, PattGen for sending test data via Tx FIFO interface, and PattVer for verifying test data via Rx FIFO interface. Register files of UserReg are controlled by CPU firmware through AXI4-Lite bus.

There are three clock domains in the design, i.e., CpuClk which is the clock for running the CPU system, MacClk which is the clock output from 100Gb Ethernet Subsystem to interface with 100Gb Ethernet Subsystem, and UserClk which is the clock for running user logic of UDP100G-IP. According to UDP100G-IP datasheet, clock frequency of UserClk must be more than or equal to 240 MHz.

Note: In real system, UserClk can be changed to use the same clock as CpuClk for reducing clock resource.

AsyncAxiReg is designed to support asynchronous signals between CpuClk and UserClk. More details of each module inside the UDP100CPUTest are described as follows.

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

This module implements EMAC and PCS/PMA logic of 100G Ethernet. The physical interface on FPGA board can be applied by QSFP28 or 4xSFP28 for 100Gb BASE-SR standard. The user interface for connecting with EMAC is 512-bit AXI4-stream interface running at 322.265625 MHz. This IP core can be created by using IP wizard in Vivado tools. More details of the core are described in the following link.

PG203: UltraScale+ Devices Integrated 100G Ethernet Subsystem Product Guide
https://www.xilinx.com/products/intellectual-property/cmac_usplus.html

2.2 UDP100G-IP

UDP100G-IP implements UDP/IP stack and offload engine. User interface has two signal groups - control signals and data signals. Control and status signals use Single-port RAM interface for write/read register access. Data signals use FIFO interface for transferring data stream in both directions. More details are described in datasheet. The interface with 100G EMAC is 512-bit AXI4 interface.

More details are described in datasheet.
https://dgway.com/products/IP/UDP100G-IP/dg_udp100gip_data_sheet_xilinx.pdf

2.3 CPU and Peripherals

32-bit AXI4-Lite is applied to be the bus interface for the CPU accessing the peripherals such as Timer and UART. To control and monitor the test system, the control and status signals are connected to register for CPU access as a peripheral through 32-bit AXI4-Lite bus. CPU assigns the different base address and the address range to each peripheral for accessing one peripheral at a time.

In the reference design, the test hardware is connected as a peripheral of CPU system with specified base address and range. Therefore, LAxi2Reg module that interfaces with CPU must support AXI4-Lite bus standard for supporting CPU writing and reading, as shown in Figure 2-2.

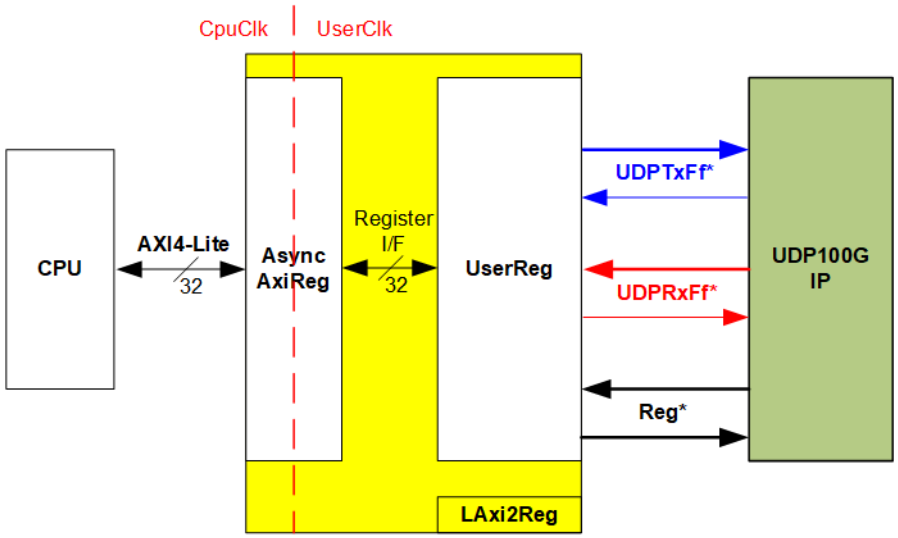

Figure 2-2 LAxi2Reg block diagram

LAxi2Reg consists of AsyncAxiReg and UserReg. AsyncAxiReg is designed to convert the AXI4-Lite signals to be the simple register interface which has 32-bit data bus size (similar to AXI4-Lite data bus size). Besides, AsyncAxiReg includes asynchronous logic to support clock domain crossing between CpuClk domain and UserClk domain.

UserReg includes the register file of the parameters and the status signals of test logics. Both data interface and control interface of UDP100G-IP are also connected to UserReg. More details of AsyncAxiReg and UserReg are described as follows.

2.3.1 AsyncAxiReg

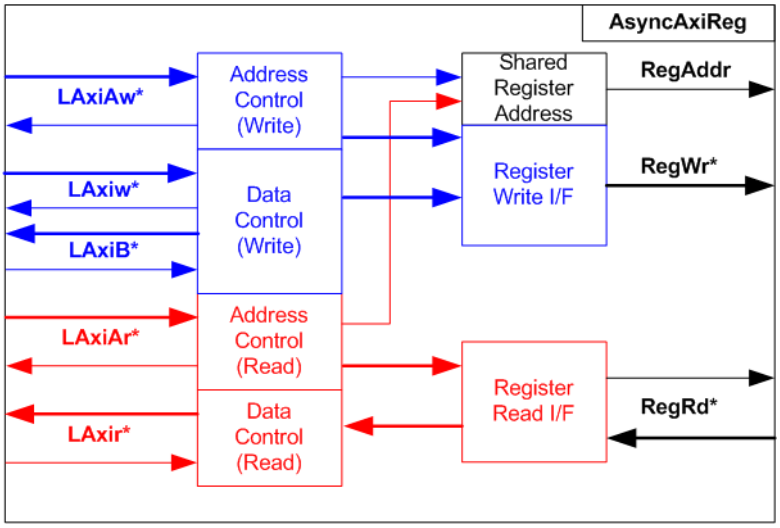

Figure 2-3 AsyncAxiReg interface

The signal on AXI4-Lite bus interface can be split into five groups, i.e., LAxiAw* (Write address channel), LAxiw* (Write data channel), LAxiB* (Write response channel), LAxiAr* (Read address channel), and LAxir* (Read data channel). More details to build custom logic for AXI4-Lite bus is described in following document.
https://forums.xilinx.com/xlnx/attachments/xlnx/NewUser/34911/1/designing_a_custom_axi_slave_rev1.pdf

According to AXI4-Lite standard, the write channel and the read channel are operated independently. Also, the control and data interface of each channel are run separately. The logic inside AsyncAxiReg to interface with AXI4-Lite bus is split into four groups, i.e., Write control logic, Write data logic, Read control logic, and Read data logic as shown in the left side of Figure 2-3. Write control I/F and Write data I/F of AXI4-Lite bus are latched and transferred to be Write register interface with clock domain crossing registers. Similarly, Read control I/F of AXI4-Lite bus are latched and transferred to be Read register interface with clock domain crossing registers. While the returned data from Register Read I/F is transferred to AXI4-Lite bus by using clock domain crossing registers. In register interface, RegAddr is shared signal for write and read access. Therefore, it loads the address from LAxiAw for write access or LAxiAr for read access.

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

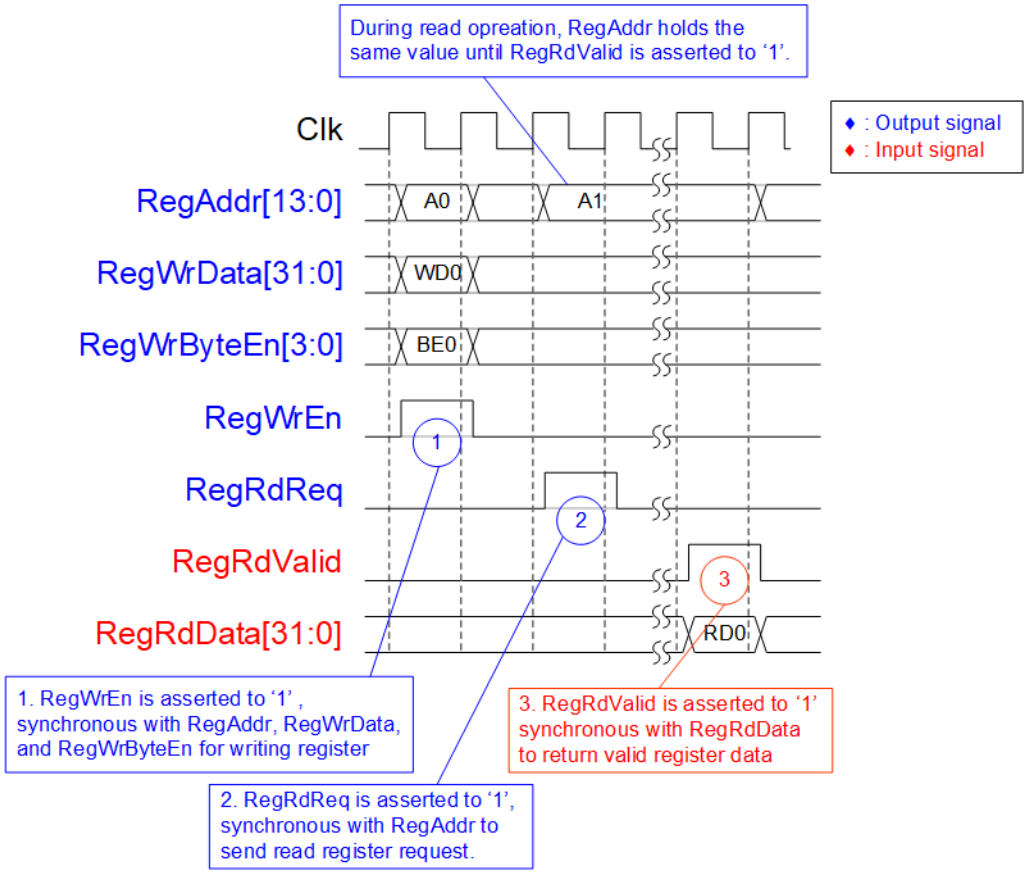

Figure 2-4 Register interface timing diagram

1) To write register, the timing diagram is similar to single-port RAM interface. RegWrEn is asserted to '1' with the valid signal of RegAddr (Register address in 32-bit unit), RegWrData (write data of the register), and RegWrByteEn (the write byte enable). Byte enable has four bits to be the byte data valid. Bit[0], [1], [2], and [3] are equal to '1' when RegWrData[7:0], [15:8], [23:16], and [31:24] are valid respectively.
2) To read register, AsyncAxiReg asserts RegRdReq to '1' with the valid value of RegAddr. 32-bit data must be returned after receiving the read request. The slave must monitor RegRdReq signal to start the read transaction. During read operation, the address value (RegAddr) does not change the value until RegRdValid is asserted to '1'. Therefore, the address can be used for selecting the returned data by using multiple layers of multiplexer.
3) The read data is returned on RegRdData bus by the slave with asserting RegRdValid to '1'. After that, AsyncAxiReg forwards the read value to LAxir* interface.

2.3.2 UserReg

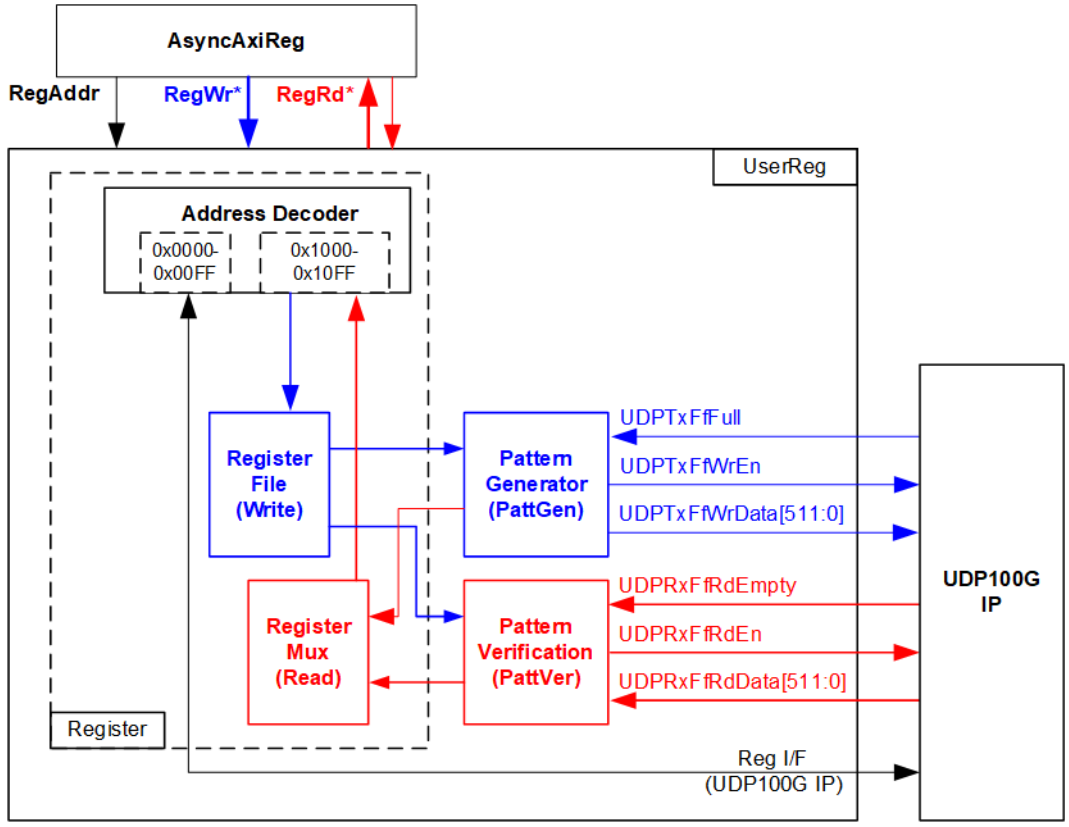

Figure 2-5 UserReg block diagram

The logic inside UserReg has three operations, i.e., Register, Pattern generator (PattGen), and Pattern verification (PattVer). Register block decodes the address requested from AsyncAxiReg and then selects the active register for write or read transaction. Pattern generator block sends 512-bit test data to UDP100G-IP following FIFO interface standard. Pattern verification block reads and verifies 512-bit data from UDP100G-IP following FIFO interface standard. More details of each block are described as follows.

Register Block

The address range, mapped to UserReg, is split into two areas, i.e., UDP100G-IP register (0x0000-0x00FF) and UserReg register (0x1000-0x10FF).

Address decoder decodes the upper bit of RegAddr for selecting the active hardware. The register file inside UserReg is 32-bit bus size. Therefore, write byte enable (RegWrByteEn) is not applied in the test system and the CPU uses 32-bit pointer to set the hardware registers.

To read register, one multiplexer is designed to select the read data within each address area. The lower bit of RegAddr is applied in each Register area to select the active data. Next, the address decoder uses the upper bit to select the read data from each area for returning to CPU. Totally, the latency of read data is equal to one clock cycle. Therefore, RegRdValid is created by RegRdReq with asserting one D Flip-flop. More details of the address mapping within UserReg module are shown in Table 2-1

Table 2-1 Register map Definition

Address Wr/Rd	Register Name (Label in the "udp100gtest.c")	Description
BA+0x0000 – BA+0x00FF: UDP100G-IP Register Area More details of each register are described in UDP100G-IP datasheet.		
BA+0x0000	UDP_RST_REG	Mapped to RST register within UDP100G-IP
BA+0x0004	UDP_CMD_REG	Mapped to CMD register within UDP100G-IP
BA+0x0008	UDP_SML_REG	Mapped to SML register within UDP100G-IP
BA+0x000C	UDP_SMH_REG	Mapped to SMH register within UDP100G-IP
BA+0x0010	UDP_DIP_REG	Mapped to DIP register within UDP100G-IP
BA+0x0014	UDP_SIP_REG	Mapped to SIP register within UDP100G-IP
BA+0x0018	UDP_DPN_REG	Mapped to DPN register within UDP100G-IP
BA+0x001C	UDP_SPN_REG	Mapped to SPN register within UDP100G-IP
BA+0x0020	UDP_TDL_REG	Mapped to TDL register within UDP100G-IP
BA+0x0024	UDP_TMO_REG	Mapped to TMO register within UDP100G-IP
BA+0x0028	UDP_PKL_REG	Mapped to PKL register within UDP100G-IP
BA+0x0034	UDP_TDH_REG	Mapped to TDH register within UDP100G-IP
BA+0x0038	UDP_SRV_REG	Mapped to SRV register within UDP100G-IP
BA+0x003C	UDP_VER_REG	Mapped to VER register within UDP100G-IP
BA+0x0040	UDP_DML_REG	Mapped to DML register within UDP100G-IP
BA+0x0044	UDP_DMH_REG	Mapped to DMH register within UDP100G-IP
BA+0x1000 – BA+0x10FF: UserReg control/status		
BA+0x1000 Wr/Rd	Total transmit length (Low) (USER_TXLENL_REG)	Wr [31:0] – 32 lower bits of 42-bit total transmit size in 512-bit unit. Valid from 1-0x3FF_FFFF_FFFF. Rd [31:0] – 32 lower bits of 42-bit current transmit size in 512-bit unit. The value is cleared to 0 when USER_CMD_REG is written by user.
BA+0x1004 Wr/Rd	Total transmit length (High) (USER_TXLENH_REG)	Wr [9:0] – 10 upper bits of 42-bit total transmit size in 512-bit unit. Rd [9:0] – 10 upper bits of 42-bit current transmit size in 512-bit unit.
BA+0x1008 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] – PattGen busy ('0': Idle, '1': PattGen is busy) [1] – Data verification error ('0': Normal, '1': Error) This bit is auto-cleared when user starts new operation or reset.
BA+0x100C 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 USER_RST_REG[8] to '0' Rd [8] – Asserted to '1' when IntOut (output from TOE10G-IP) is asserted to '1'. This flag is de-asserted by writing USER_RST_REG[8]='1' or system reset. [16] – Ethernet linkup status from Ethernet MAC ('0': Not linkup, '1': Linkup)
BA+0x1010 Rd	FIFO status (USER_FFSTS_REG)	Rd[5:0] - Mapped to UDPRxFfLastRdCnt signal of UDP100G-IP [15:6] - Mapped to UDPRxFfRdCnt signal of UDP100G-IP [24] - Mapped to UDPTxFfFull signal of UDP100G-IP
BA+0x1014 Rd	Total receive length (Low) (USER_RXLENL_REG)	Rd[31:0] – 32 lower bits of 42-bit current receive size in 512-bit unit The value is cleared to 0 when USER_CMD_REG is written by user.
BA+0x1018 Rd	Total receive length (High) (USER_RXLENH_REG)	Rd[9:0] – 10 upper bits of 42-bit current receive size in 512-bit unit
BA+0x1080 Rd	EMAC IP version (EMAC_VER_REG)	Rd[31:0] – Mapped to IPVersion output from DG EMAC-IP when the system integrates DG EMAC-IP. In this demo, it is equal to 0.

Pattern Generator

Figure 2-6 shows the details of PattGen which generates test data to UDP100G-IP while Figure 2-7 shows timing diagram of the signals inside PattGen.

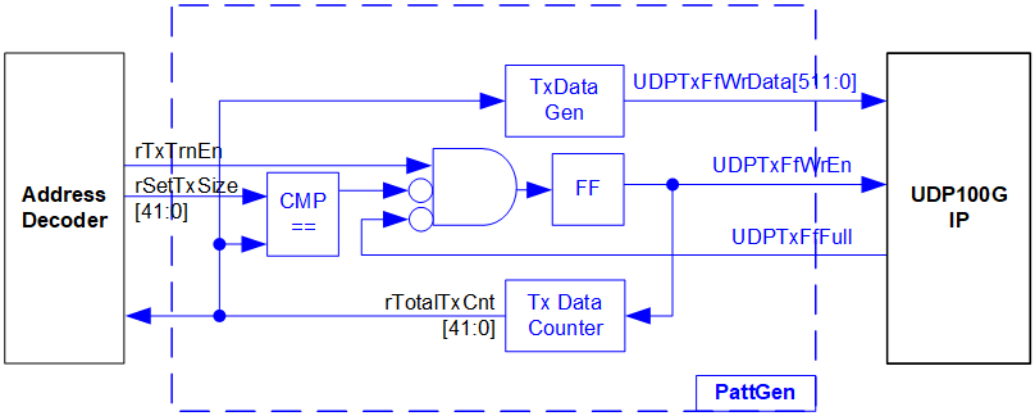

Figure 2-6 PattGen block

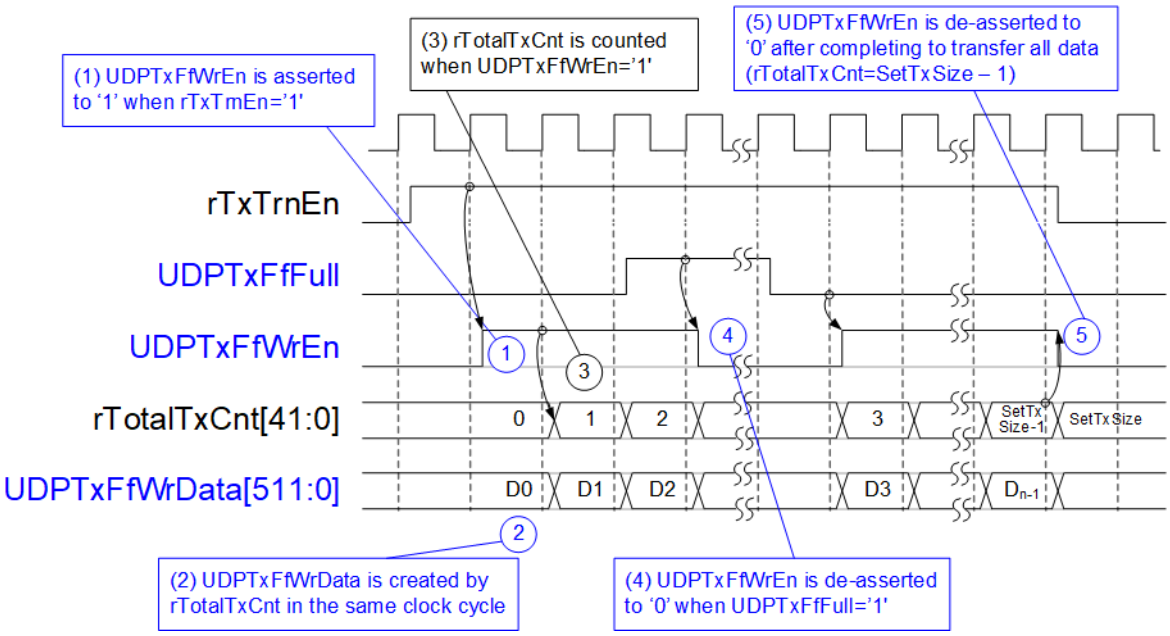

Figure 2-7 PattGen timing diagram

To start PattGen operation, the user sets USER_CMD_REG[0]='0' and then rTxTrnEn is asserted to '1'. When rTxTrnEn is '1', UDPTxFfWrEn is controlled by UDPTxFfFull. UDPTxFfWrEn is de-asserted to '0' when UDPTxFfFull is '1'. rTotalTxCnt is the data counter to check total number of transmitted data sent to UDP100G-IP. Also, the lower bits of rTotalTxCnt are used to generate 32-bit incremental data to UDPTxFfWrData signal. After all data is transferring completely, equal to rSetTxSize, rTxTrnEn is de-asserted to '0'.

Pattern Verification

Figure 2-8 shows the details of PattVer which receives test data from UDP100G-IP while Figure 2-9 shows timing diagram of the signals inside PattVer.

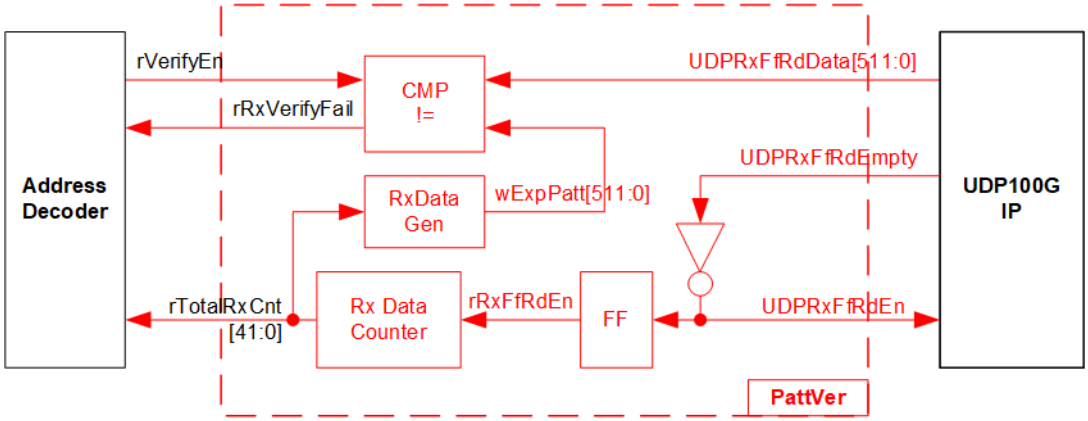

Figure 2-8 PattVer block

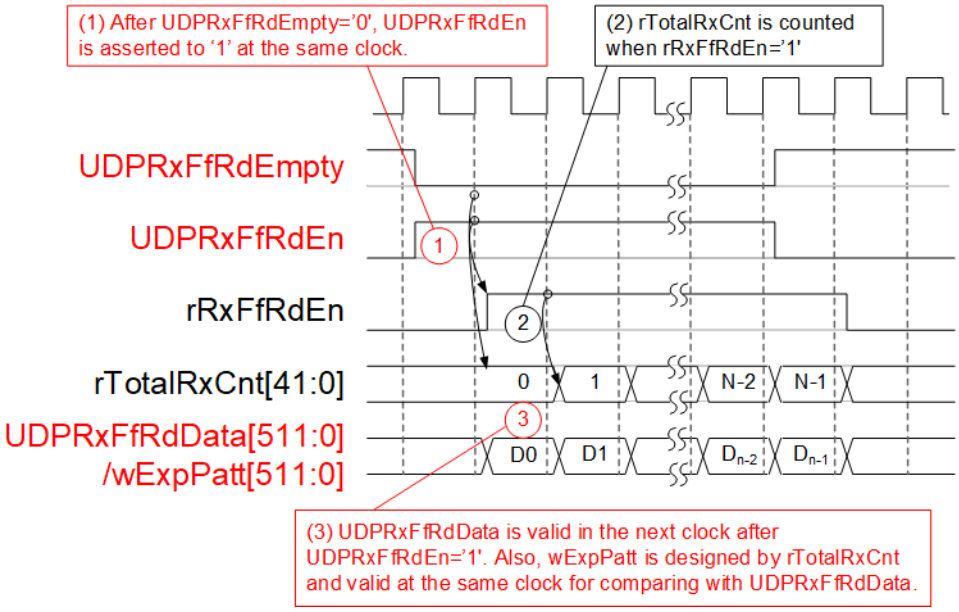

Figure 2-9 PattVer Timing diagram

When rVerifyEn is set to '1', the verification logic is run. The received data (UDPRxFfRdData) is compared with the expected data (wExpPatt). If data verification is failed, rRxVerifyFail is asserted to '1'. UDPRxFfRdEn is designed by using NOT logic of UDPRxFfRdEmpty. UDPRxFfRdData is valid for data comparison in the next clock. rRxFfRdEn, one clock latency of UDPRxFfRdEn, is applied to be counter enable of rTotalRxCnt which is designed to count total number of received data size. Also, the lower bits are applied to generate wExpPatt. Therefore, UDPRxFfRdData and wExpPatt are valid in the same clock for running data comparison, controlled by rRxFfRdEn signal.

3 CPU Firmware on FPGA

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

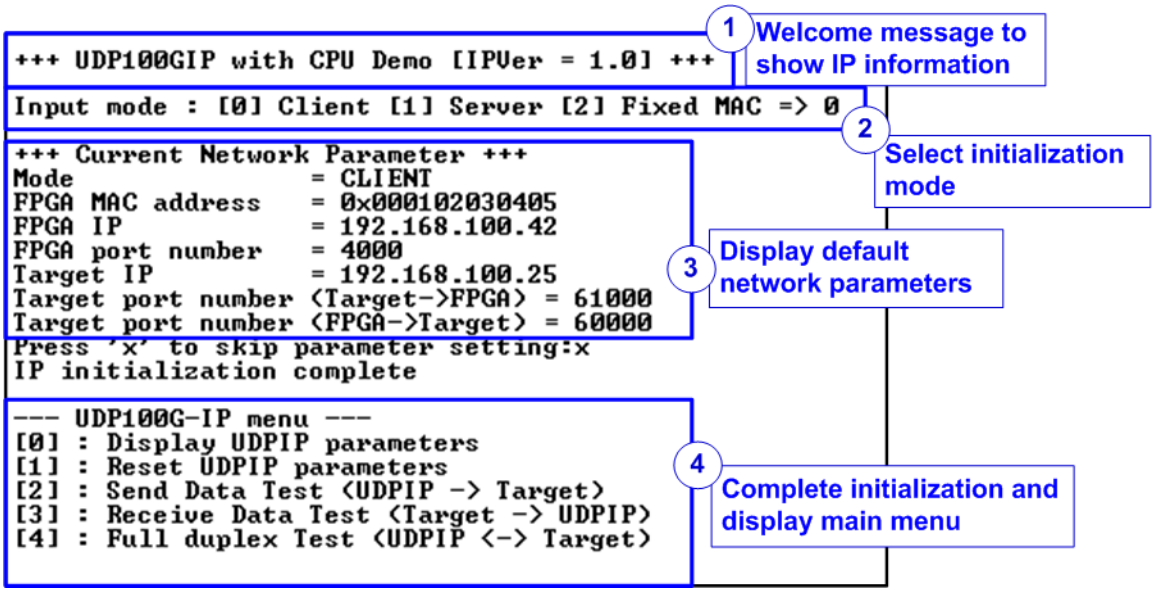

Figure 3-1 Message on the console during initialization process

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

1) After FPGA boot-up, 100G Ethernet link up status (USER_RST_REG[16]) is polling. The CPU waits until link up is detected and then displays welcome message to show IP information.
2) The menu to select the initialization mode of UDP100G-IP is displayed. The user can set as Client, Server, or Fixed-MAC mode.

   Note:
   a) *When running in Client mode, UDP100G-IP sends ARP request to get the MAC address of the target device from ARP reply. When running in Server mode, UDP100G-IP waits until ARP request is received to decode MAC address and return ARP reply. When running Fixed-MAC mode, the user needs to know MAC address of the target device because UDP100G-IP does not transfer ARP packet.*
   b) *When running the test environment by using one FPGA board and Test PC, it is recommended to set FPGA to run as Client mode.*
   c) *When the test environment uses two FPGA boards, there are three ways to initial the connection between two boards. First, one is Client and another is Server. Second, both are set to Fixed-MAC mode. Last, one is set to Fixed-MAC mode and another must be set to Client.*

3) CPU displays default value of the network parameters, i.e., initialization mode, FPGA MAC address, FPGA IP address, FPGA port number, Target IP address, and Target port number. The firmware has two default parameter sets for the operation mode, Server parameter set and Client/Fixed-MAC parameter set. For Fixed-MAC mode, there is the extra parameter, Target MAC address. The user can select to complete the initialization process by using default parameters or updating some parameters. The details to change the parameter are described in Reset parameters menu (topic 3.2).
4) CPU waits until the IP completes the initialization process by checking if busy status (UDP_CMD_REG[0]) is equal to '0'. After that, "IP initialization complete" is displayed with the main menu. There are five test operations in the main menu. More details of each menu are described as follows.

3.1 Display parameters

This menu is designed to display the current value of all UDP100G-IP parameters.

The step to display parameters is as follows.

1) Read the initialization mode.
2) Read all network parameters from each variable in firmware following the initialization mode, i.e., source (FPGA) MAC address, source (FPGA) IP address, source (FPGA) port number, target MAC address (only displayed in fixed MAC mode), target IP address, and target port number.
   Note: The source parameters are FPGA parameters set to UDP100G-IP while the target parameters are the parameters of TestPC or another FPGA.
3) Print out each variable

3.2 Reset parameters

This menu is used to change some UDP100G-IP parameters such as IP address and source port number. After setting the updated value to UDP100G-IP, the CPU resets the IP to start re-initialization process by using new parameters. Finally, the CPU waits until the initialization is completed.

The step to reset parameters is as follows.

1) Display all parameters on the console, similar to topic 3.1 (Display parameters).
2) Skip to the next step if the user uses the default value. Otherwise, the menu to set all parameters is displayed.
   i. Receive initialization mode from the user. If the initialization mode is changed, the latest parameter set of new mode is displayed on the console.
   ii. Receive remaining parameters from user and verify all inputs. If the input is invalid, the parameter is not updated.
3) Force reset to IP by setting UDP_RST_REG[0]='1'.
4) Set all parameters to UDP100G-IP register such as UDP_SML_REG and UDP_DIP_REG.
5) De-assert IP reset by setting UDP_RST_REG[0]='0' to start IP initialization process.
6) Wait until busy flag (UDP_CMD_REG[0]) is asserted to '0' after finishing the initialization process.

3.3 Send data test

This menu is designed to run sending data test. The user sets the parameters such as total transmit length. If the inputs are valid, the data is transferred by sending 32-bit incremental test data. The operation is finished when all data is completely transferred.

The step to send the data is as follows.

1) Receive two parameters from user, i.e., total transmit size and packet size. After that, CPU verifies all inputs. The operation is cancelled if some inputs are invalid.
2) Set UserReg registers, i.e., transfer size (USER_TXLENL/H_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 starts sending data to UDP100G-IP.
3) Display recommended parameters of test application on PC by reading current system parameters. Wait until user enters any keys to start IP sending data operation.
4) Set packet size to UDP100G-IP register (UDP_PKL_REG) and set total number of transmitted data to UDP_TDL/H_REG. After that, set Send command to UDP100G-IP (UDP_CMD_REG[0]='1') to run Send command.
5) Wait until operation is completed by monitoring busy flag (UDP_CMD_REG[0]='0'). During monitoring busy flag, CPU reads current number of transferred data from user logic (USER_TXLENL/H_REG) and displays the results on the console every second.
6) After the operation is completed, CPU calculates performance and displays test result on the console.

3.4 Receive data test

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

The step to receive the data is as follows.

1) Receive two parameters, i.e., total transfer size and data verification mode from user input. The operation is cancelled if some inputs are invalid.
2) Set UserReg registers, i.e., reset flag to clear the initial value of test pattern (USER_RST_REG[0]='1') and data verification mode (USER_CMD_REG[1]='0'/'1' to disable/enable).
3) Display recommended parameter (similar to Step 3 of Send data test).
4) Wait until total number of received data (USER_RXLENL/H_REG) is equal to the set value (complete condition) or the number of received data is not updated for 100 msec (timeout condition). During receiving data, CPU displays the current number of received data on the console every second.
5) Stop timer and check data verification status (USER_CMD_REG[1]). If the verification error is found, the error message will be displayed.
6) Calculate performance and then display 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 another device (PC/FPGA) in both directions at the same time. The menu receives user parameters for running the test such as total transfer length. If all inputs are valid, the data starts transferring. The operation is finished when the data in both directions are completely transferred.

Note: When running the test with PC, the transfer size on the test application (udpdatatest) must be equal to the transfer size set on FPGA. Two "udpdatatest" are run by using different port number, one for sending data and another for receiving data. When running by FPGA and FPGA, the port number for sending and receiving data are similar.

The step to run full duplex test is as follows.

1) Receive three parameters, i.e., total transfer size (the same size for both transfer directions), packet size, and data verification mode (enable or disabled) from user. The operation is cancelled if some inputs are invalid.
2) Set UserReg registers, i.e., transfer size (USER_TXLENL/H_REG), reset flag to clear the 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=0 or 2).
3) Display the recommended parameters of test application run on PC from the current system parameters.
4) Set packet size to UDP100G-IP register (UDP_PKL_REG) and set total number of transferred data to UDP_TDL/H_REG. After that, set Send command to UDP100G-IP (UDP_CMD_REG[0]='1') to run Send command. The IP starts sending data to the target device. At the same time, the IP is ready to receive data from the target.
5) CPU controls data flow of both directions at the same time. Therefore, there are two tasks running in the test, described as follows.
   a. To send data, CPU reads busy flag (UDP_CMD_REG[0]) to wait until it is de-asserted to '0'. When Send command is finished, busy flag is de-asserted to '0'.
   b. To receive data, CPU reads total number of received data. The read process is finished when total number of received data is equal to set value (no data lost). Otherwise, it is finished when total number of received data does not change for 100 msec (timeout).

   When the data is not completely transferred, the current number of transmit data size (USER_TXLENL/H_REG) and receive data size (USER_RXLENL/H_REG) are read and displayed on the console every second.
6) Stop timer and check data verification status (USER_CMD_REG[1]). If the verification error is found, the error message will be displayed.
7) Calculate performance and display test result on the console.

3.6 Function list in User application

This topic describes the function list to run UDP100G-IP operation.

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

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

void show_cursize(void)	
Parameters	None
Return value	None
Description	Read current number of transmitted data and number of received data by reading USER_TXLENL/H_REG and USER_RXLENL/H_REG. Then, display the result in Byte, KByte, or MByte unit.

void show_interrupt(void)	
Parameters	None
Return value	None
Description	Read interrupt status from UDP_TMO_REG and decode interrupt type to display the details of interrupt on the console.

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

void show_result(void)	
Parameters	None
Return value	None
Description	Read total transmit data size and total receive data size from USER_TXLENL/H_REG and USER_RXLENL/H_REG and display the results. After that, read total time usage from 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 in MB/s unit.

int udp_recv_test(void)	
Parameters	None
Return value	0: The operation is successful -1: Receive invalid input or error is found
Description	Run Receive data test following description in topic 3.4. It calls show_interrupt, show_cursize, and show_result function.

int udp_send_test(void)	
Parameters	None
Return value	0: The operation is successful -1: Receive invalid input or error is found
Description	Run Send data test following description in topic 3.3. It calls show_cursize and show_result function.

int udp_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. It calls show_interrupt, show_cursize, and show_result function.

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

4 Test Software on PC

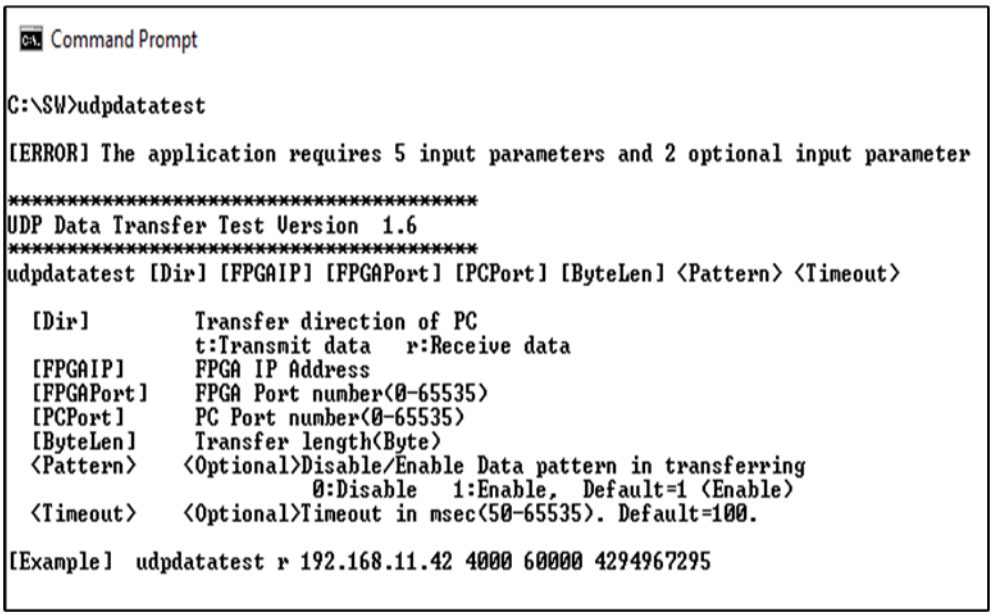

Figure 4-1 “udpdatatest” application usage

“udpdatatest” is an application on PC for sending or receiving UDP payload data. There are five parameters and two optional parameters. To run the test, the parameters must be matched to parameter set 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 192.168.100.42)
3) FPGAPort : Port number of FPGA (default value in FPGA is 4000)
4) PCPort : PC port number for sending or receiving data
   (default is 61000 for PC to FPGA and 60000 for FPGA to PC)
5) ByteLen : Transfer length for sending or receiving data in byte unit.
   This value must be aligned to 64 from UDP100G-IP limitation.
6) Pattern (optional): Default value when user does not input this parameter is equal to 1.
   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.
7) Timeout (optional): Timeout for receiving data in msec unit.
   Default value when user does not input this parameter is 100.
   100 ms is recommended value for running with UDP100G-IP.

Transmit data mode

The step when running the test application in transmit mode is described as follows.

1) Get parameters from user and verify that the input is valid.
2) Create the socket and then set properties of receive buffer.
3) Set IP address and port number from user parameters and then connect.
4) Send data to the send buffer for transmitting data. During sending data, the application prints total sent data on the console every second.
   a) When Pattern=1, the send buffer is filled by 32-bit incremental pattern.
   b) When Pattern=0, the send buffer is not filled. Dummy data is applied in the test.
5) After finishing sending all data, the application displays performance with total transmit data size as a test result.

Receive data mode

The step when running the test application in receive mode is described as follows.

1) Follow step (1)-(3) in Transmit data mode.
2) Repeat to read data until total number of received data is equal to set value. Otherwise, it is cancelled when there is no new received data until timeout. During reading data, the application prints total number of received data on the console every second.
   a) When Pattern=1, the read data is verified by 32-bit incremental pattern which is increased every 4-byte received data.
   b) When Pattern=0, the read data is not verified.
3) When the read loop is finished by timeout condition, “Timeout” message is displayed with total number of lost data and total number of received data. Also, total time usage is decreased by timeout value.
4) After finishing the operation, the application displays performance as a test result.

5 Revision History

Revision	Date	Description
1.0	4-Aug-21	Initial version release