# UDP10G-IP reference design manual

Rev1.2 22-Mar-18

## 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.

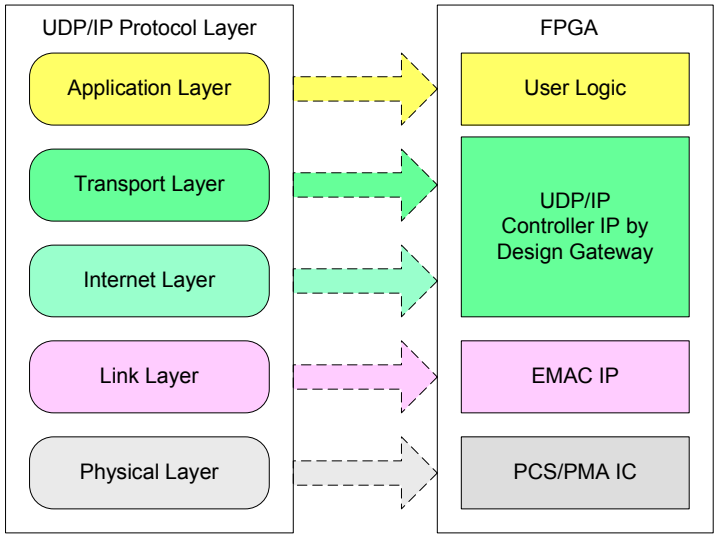


Figure 1-1 UDP/IP Protocol Layer

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-IP from Intel FPGA.

This reference design provides evaluation system which includes simple user logic to send and receive data by using UDP10G-IP. This system demonstrates on Intel FPGA 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 NiosII command shell and select transfer directions to start test operation. The test application on PC is “udpdatatest.exe”.

# 2 Hardware description

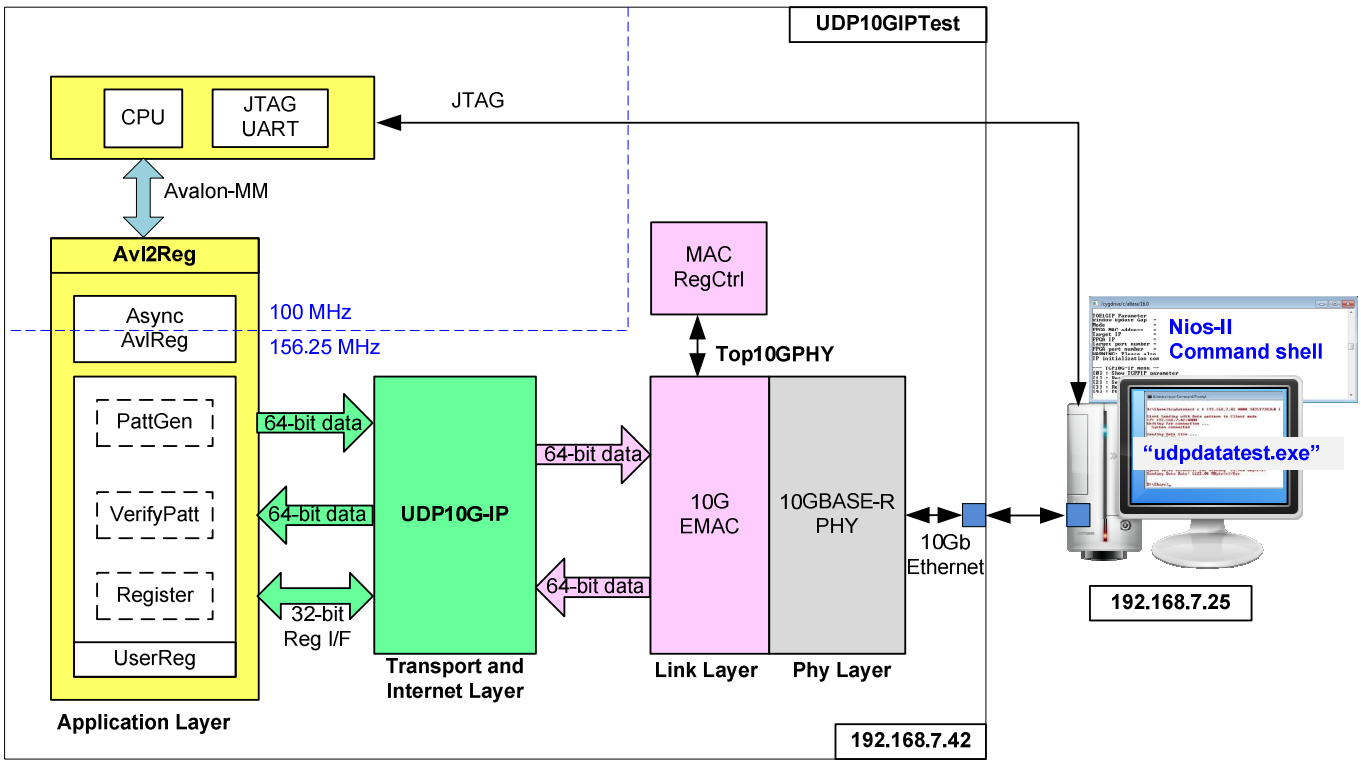


Figure 2-1 Hardware Architecture reference design

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 PHY 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 for transfer data in each direction. To avoid performance dropped by CPU resource limitation, the reference design shows only half-duplex transfer. User can modify CPU firmware to run full-duplex transfer by enable both PattGen and VerifyPatt module to send and receive data with UDP10G-IP at the same time.

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, Avl2Reg includes register to store test parameters from user such as transfer length. User inputs parameters through NiosII command shell. CPU firmware validates all parameters and sets to the hardware through Avalon-MM bus. Due to the fact that CPU system and UDP10G-IP run in different clock domain, AsyncAvlReg module is used to be asynchronous circuit to support clock-crossing function and convert Avalon-MM 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 Ethernet MAC and 10GBASE-R PHY

Link layer and physical layer in the reference design are implemented by IntelFPGA IP core. For Arria10 FPGA, Low latency 10G Ethernet MAC is applied to connect with UDP10G-IP and 10GBASE-R PHY. More details are described in following link.
https://www.altera.com/products/intellectual-property/ip/interface-protocols/m-alt-10gbps-ethernet-mac.html

10GBASE-R PHY is designed to connect with 10G SFP+ module. Another side is connected to 10G Ethernet MAC through XGMII interface running at 156.25 MHz. More details are described in following link.
https://www.altera.com/products/intellectual-property/ip/interface-protocols/m-alt-10gbase-r-pcs.html

## 2.2 UDP10G-IP

UDP10G-IP implements UDP/IP stack and offload engine. Control and status signals are accessed through register interface. Data interface is accessed through FIFO interface. More details are described in datasheet.
http://www.dgway.com/products/IP/UDP10G-IP/dg_udp10gip_data_sheet_intel_en.pdf

## 2.3 MACRegCtrl

This module is designed to set parameter to 10 Gb Ethernet MAC and monitors EMAC status through Avalon MM bus. The logic is simple designed by using state machine which runs only one time after system power up for initializing Ethernet MAC.

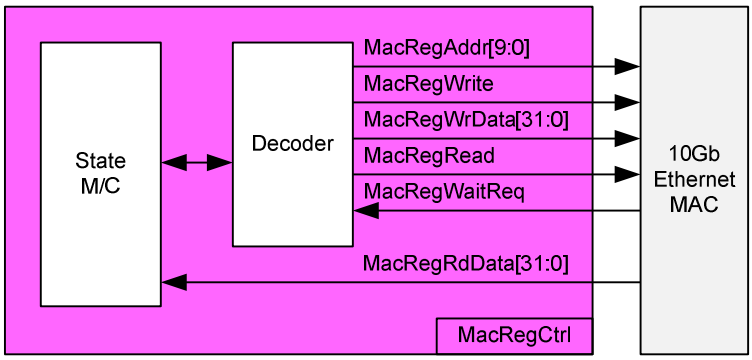


Figure 2-2 MACRegCtrl Block Diagram

The sequence of initialization sequence is below.
1) Disable receive path of EMAC.
2) Disable transmit path of EMAC.
3) Monitor that receive path is in idle status.
4) Monitor that transmit path is in idle status.
5) Set receive path of EMAC to remove CRC and padding.
6) Disable pause frame transmission in transmit path.
7) Enable receive path of EMAC.
8) Enable transmit path of EMAC.

## 2.4 Avl2Reg

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

Avl2Reg connects to UDP10G-IP for both data path and control path. As shown in Figure 2-3, there are two clock domains applied in this block, i.e. 100 MHz (CpuClk) which is used to interface with CPU through Avalon-MM bus and 156.25 (MacClk) which is user clock domain for UDP10G-IP and EMAC.

AsyncAvlReg includes asynchronous circuit between 100 MHz and 156.25 MHz. More details of each hardware are described as follows.

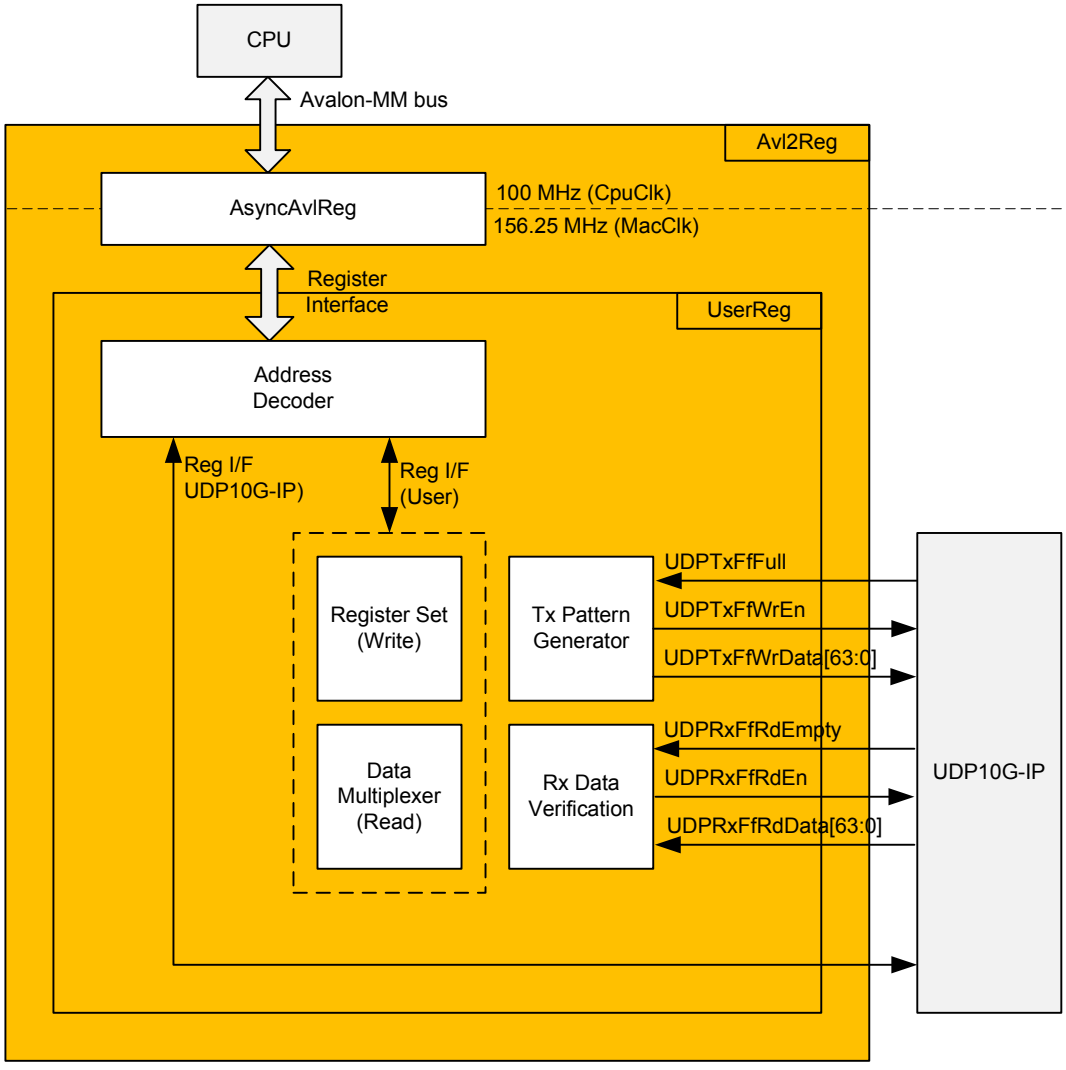


Figure 2-3 Avl2Reg block diagram

### 2.4.1 AsyncAvlReg

This module is designed to convert signal interface of Avalon-MM to be register interface. Also, it supports to convert clock domain from 100 MHz to be 156.25 MHz. Timing diagram of register interface is shown in Figure 2-4.

To write register, timing diagram is same as RAM interface. RegWrEn is asserted to '1' with the valid signal of RegAddr (Register address in 32-bit unit), RegWrData (write data of the register), and RegWrByteEn (the byte enable of this access: bit[0] is write enable for RegWrData[7:0], bit[1] is used for RegWrData[15:8], …, and bit[3] is used for RegWrData[31:24]).

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

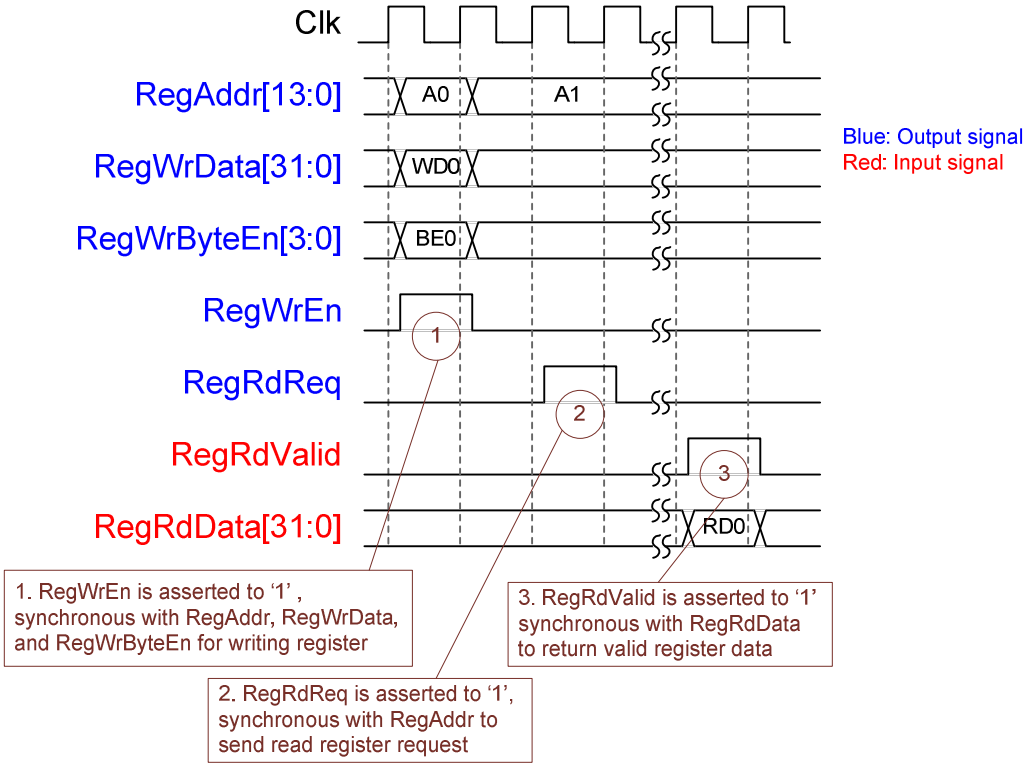


Figure 2-4 Register interface timing diagram

### 2.4.2 UserReg

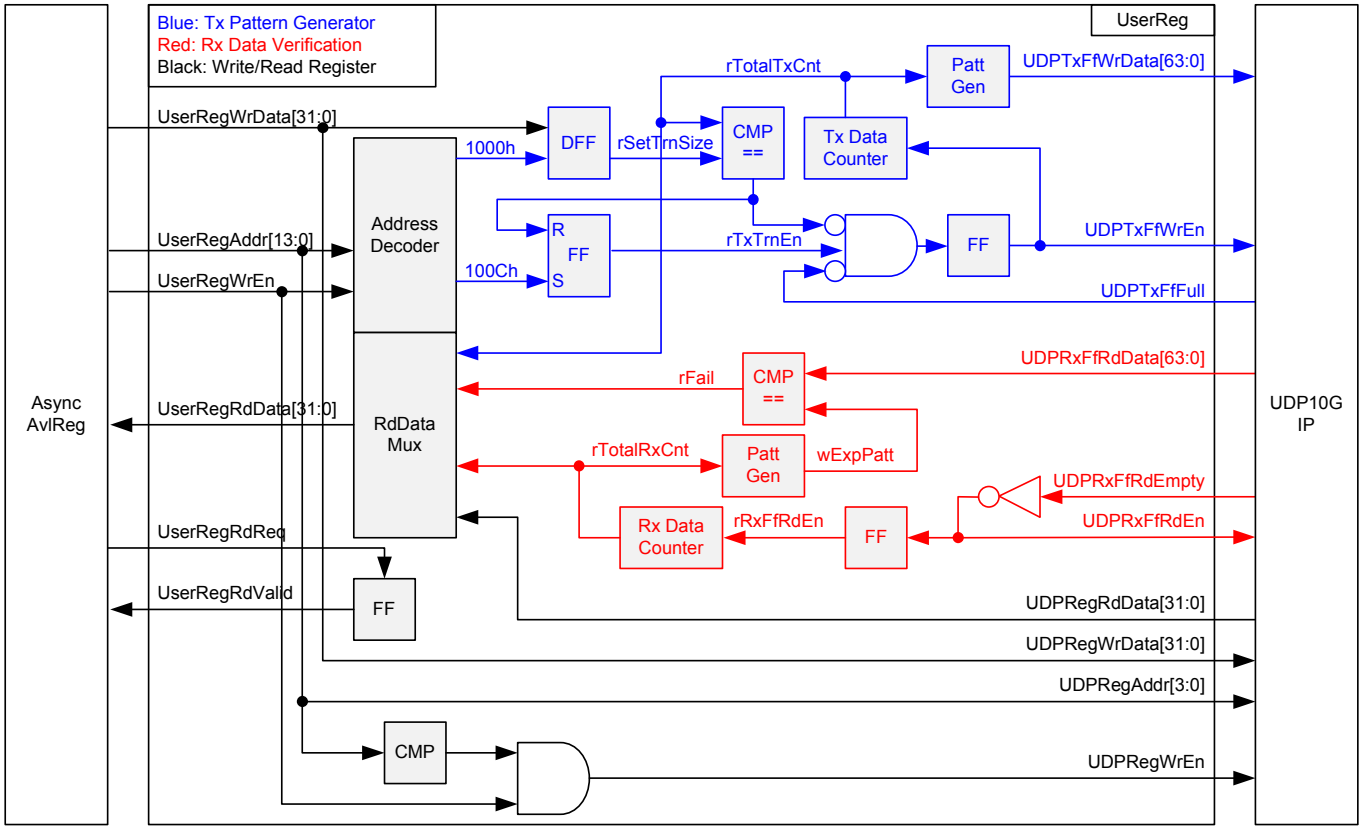


Figure 2-5 UserReg block diagram

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 inside UserReg (to control Tx Pattern Generator and Rx Data Verification).

To request write register, UserRegWrEn is asserted to '1' with the valid of UserRegAddr. The upper bits of UserRegAddr are forwarded to check whether the address is in UDP10G-IP range or not. If the address is in UDP10G-IP range, UDPRegWrEn will be asserted to '1'. For internal register of UserReg, UserRegWrData is loaded to internal register when the address is matched. For example, rSetTrnSize is loaded by UserRegWrData when UserRegAddr=0x1000. UserRegWrByteEn signal is not used in this module, so CPU firmware needs to access the hardware register by using 32-bit pointer only.

For read request, UserRegRdReq is asserted to '1'. RdDataMux selects status signals from internal register or UDP10G-IP, and forwards to UserRegRdData in the next clock. 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 2-5 is designed to generate test data to UDP10G-IP. rTxTrnEn is asserted to '1' when write register address is 100Ch. When rTxTrnEn is '1', UDPTxFfWrEn is controlled by UDPTxFfFull as shown in Figure 2-6. UDPTxFfWrEn is de-asserted to '0' when UDPTxFfFull is '1'. rTotalTxCnt is data counter to check total transfer data to UDPTxFf. rTotalTxCnt is also used to generate 32-bit increment data to UDPTxFfWrData signal. rTxTrnEn is de-asserted to '0' when total transfer size is equal to the set value (rSetTrnSize).

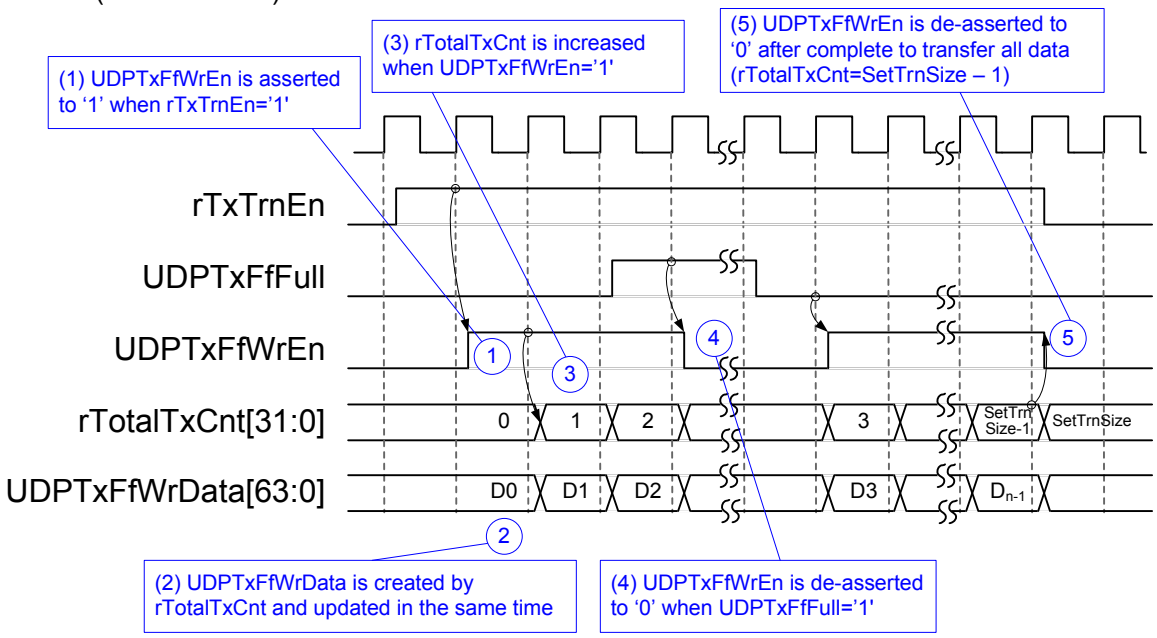


Figure 2-6 Tx Pattern Generator Timing diagram

The logic in red color of Figure 2-5 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. rTotalRxCnt is data counter to check total transfer data 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.

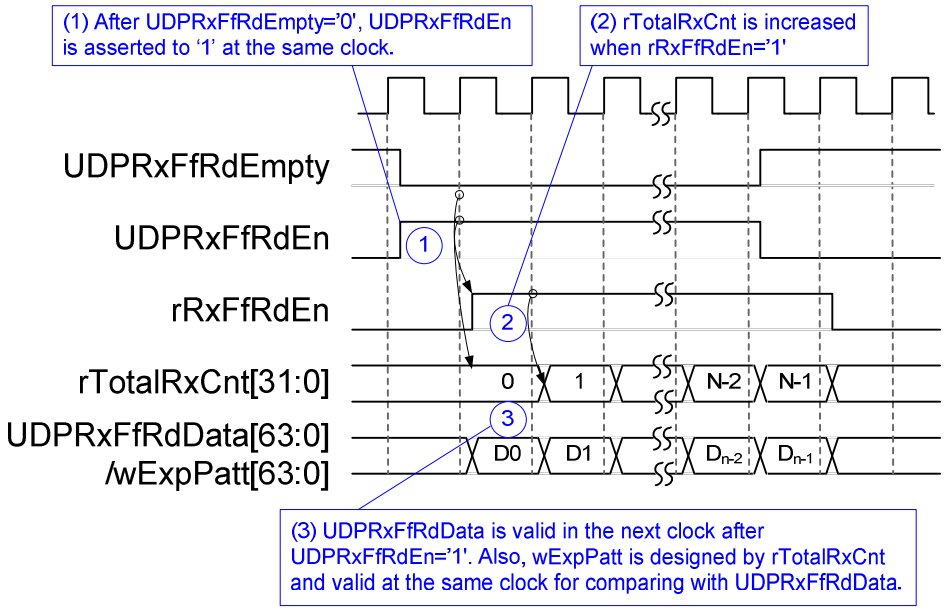


Figure 2-7 Rx Data Verification Timing diagram

## Table 2-1 Register map Definition

| Address<br>Wr/Rd | Register Name<br>(Label in the “udp10cpu_demo.c” ) | Description |
|---|---|---|
| BA+0x0000 – BA+0x00FF: UDP10G-IP Register Area (BA=0x0030_0000)<br>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=0x0030_0000) | | |
| BA+0x1000<br>Wr/Rd | Total transmit length<br>(USER_TXLEN_REG) | Wr [31:0] – Total transmit size in QWord unit (64-bit).<br>Rd [31:0] – Current transmit size in QWord unit (64-bit). |
| BA+0x1004<br>Rd | Total Receive length<br>(USER_RXLEN_REG) | Rd [31:0] – Current receive size in QWord unit (64-bit). |
| BA+0x100C<br>Wr/Rd | User Control<br>(USER_CTRL_REG) | Wr<br>[0] – Start Transmit. Set ‘1’ to start transmit.<br>This bit is auto-cleared to ‘0’.<br>[1] – Data Verification enable<br>(‘0’: Enable data verification, ‘1’: Disable data verification)<br>Rd [0] – Tx Busy. (‘0’: Idle, ‘1’: Tx module is busy) |
| BA+0x1010<br>Wr/Rd | User Error status<br>(USER_ERR_REG) | Wr [0] – Reset user logic. Set ‘1’ to force reset to this module.<br>This bit is auto-cleared to ‘0’.<br>Rd<br>[0] – Data verification error (‘0’: Normal, ‘1’: Error).<br>This bit is auto-cleared when user starts new operation or reset.<br>[1] – Latch value of TimerInt output from IP<br>(‘0’: Normal, ‘1’: TimerInt=’1’ is detected)<br>This flag can be cleared from system reset condition or user logic reset (USER_ERR_REG[0]=’1’). |

# 3 CPU Firmware Sequence

Main menu in the firmware has three operations, i.e.

1) Reset IP: This menu is used to change UDP10G-IP parameters such as FPGA Mac address, FPGA IP address, FPGA Port number, Target IP address and Target Port number. After setting all registers, the IP releases reset and starts initialization process. CPU monitors busy flag to confirm that initialization process is completed.
2) Send data test: Two user inputs are required, i.e. total transmit length which maximum size is 4 GByte and packet length which must be aligned to 8-byte unit. Then, data is transferred to PC. PC runs test application to read and verify data from FPGA. On the console, total received data is shown during transfer every 1 second. Total transfer size and performance are displayed after end of transfer.
3) Receive data test: User can select to enable or disable data verification mode within UserReg. Data verification failure from dropped packet could be found because UDP protocol is a lossy protocol. Similar to Send data test, total transfer size and performance are displayed on the console.

## 3.1 Reset IP

The sequence of this operation is as follows.

1) Display current parameter value to the console.
2) Receive input parameter from user and verify that the value is valid. If input is invalid, parameter will not change.
3) Reset IP by setting UDP10_RST_REG[0]='1' and hold it until finishing parameter setting.
4) Set parameter input from user to UDP10G-IP register such as UDP10_SML_REG, UDP10_DIP_REG, UDP10_SPN_REG and UDP10_TMO_REG.
5) Release IP reset by de-asserting UDP10_RESET_REG[0]='0'.
6) Monitor IP busy flag (UDP10_CMD_REG[0]). Wait until busy flag is de-asserted to '0' to confirm that initialization sequence is completed.

## 3.2 Send data test

The sequence of this operation is as follows.

1) Receive transmit size and packet size from user and verify that all inputs are valid.
2) Set User Module registers, i.e. transmit size (USER_TXLEN_REG), reset flag to clear initial value of test pattern (USER_ERR_REG), and USER_CTRL_REG[0]='1' to start data pattern generator.
3) CPU sets UDP register, i.e. UDP packet length (UDP10_PKL_REG), UDP total transfer length (UDP10_TDL_REG) and UDP10_CMD_REG[0]='1' to start data sending operation
4) Wait until IP busy flag (UDP10_CMD_REG[0]) is de-asserted to confirm that sending process is completed.
5) During transfer, current transfer size is monitored by USER_TXLEN_REG[31:0] and displayed every second.
6) After sending complete, stop timer and show test performance to the console.

### 3.3 Receive data test

The sequence of this operation is as follows.

1) Receive verification mode from user and verify that input is valid.
2) Set User Module registers, i.e. data verification mode (USER_CTRL_REG[1]) and reset flag to clear initial value of test pattern (USER_ERR_REG).
3) Wait until receive the first packet by monitoring current received size (USER_RXLEN_REG) is not equal to 0. Then, start timer.
4) Wait until total receive size (USER_RXLEN_REG) does not change more than 1 msec.
5) Check Timeout (USER_ERR_REG[1]) and verify flag (USER_ERR_REG[0]) register when verification mode is applied.
6) Stop timer and show test performance to the console.

# 4 Test Software description

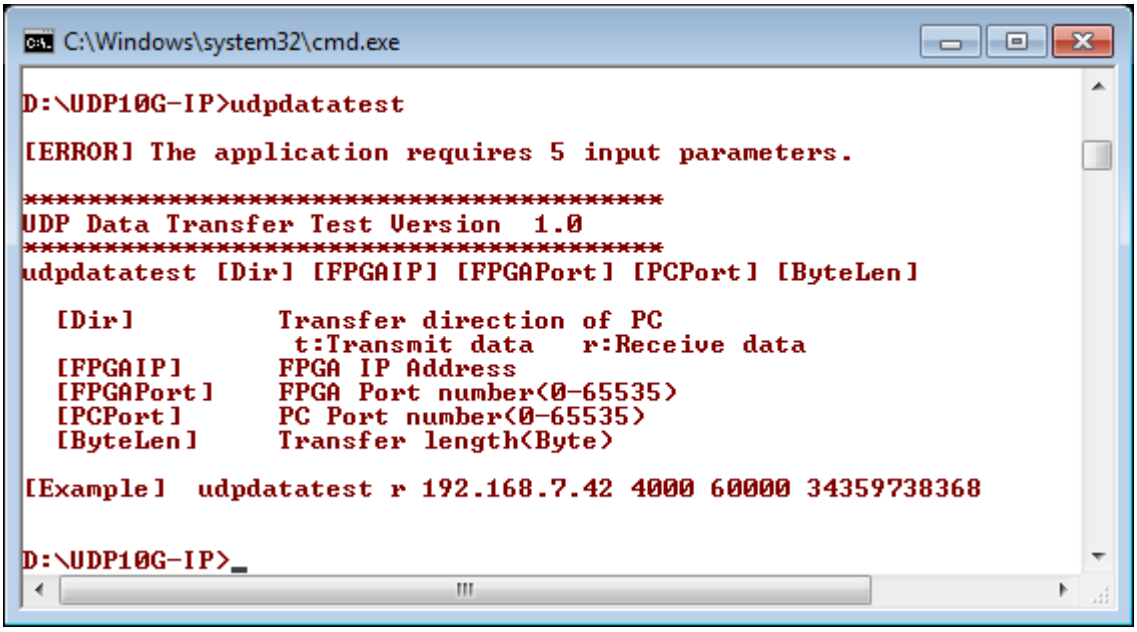


Figure 4-1 udpdatatest application parameter

"udpdatatest" is an application on PC for sending or receiving UDP data. There are five parameters to select test mode and direction. The parameter input should be matched to parameter setting on the console. More details of each parameter input are followed.

1) Dir: t – when PC sends data to FPGA
   r – when PC receives data from FPGA
2) FPGAIP : IP address setting on FPGA (default is 192.168.7.42)
3) FPGAPort : Port number of FPGA (default 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.

### 4.1 Receive data mode

The operation sequence of the application is as follows.

(1) Get parameters from user.
(2) Create socket and then set properties of receive buffer.
(3) Set IP address and Port number from user parameter and then connect.
(4) Loop to verify data until total received data is equal to set value or no more received data within 100 msec. Verification pattern is 32-bit increment starting from 0. Pattern is increased after receiving 4 bytes. During running, application prints total received size on the console every second.
(5) In case of 100 msec timeout condition, “Timeout” message is displayed with total lost size and total receive size when end of operation.

### 4.2 Transmit data mode

The operation sequence of the application is as follows.

(1) – (3) are same as Receive data mode
(4) Generate 32-bit increment pattern to buffer and then send data out. The packet size is fixed to 1472 byte.
(5) 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 | 22-Mar-18 | Add Avl2Reg details |