# UDP1G-IP reference design manual

Rev1.1 14-Aug-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.

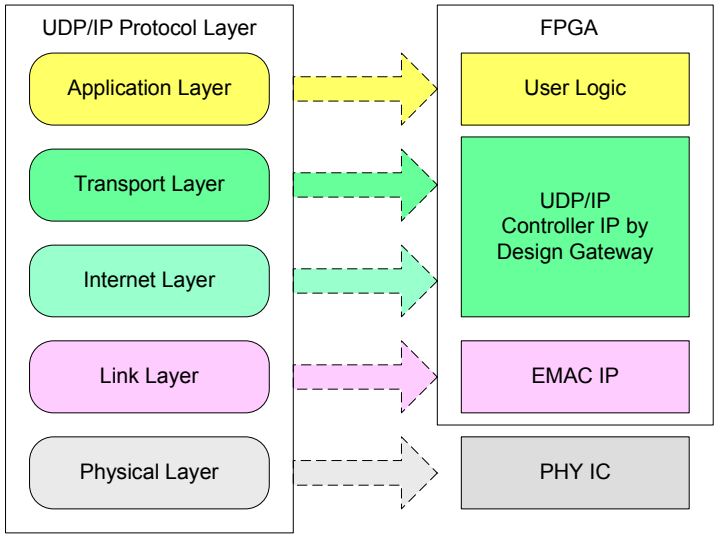


Figure 1-1 UDP/IP Protocol Layer

UDP1G-IP implements Transport and Internet layer of UDP/IP Protocol. For transmit side, UDP1G-IP prepares UDP data from user logic and adds UDP/IP header to generate Ethernet packet format before sending out to EMAC. For receive side, UDP1G-IP validates UDP/IP header of the packet and extracts only UDP data. Only UDP data of valid packet is stored to buffer for user logic reading.

The lower layer protocols are implemented by EMAC-IP from Intel and external PHY chip.

This reference design provides evaluation system which includes simple user logic to send and receive data by using UDP1G-IP. For user interface, CPU system is designed to interface with user through JTAG UART. The firmware is designed as bare-metal OS. Test application running on PC in the demo is “udp1gdatatest.exe”. The reference design is available on Intel development board to show high-speed transfer for both half-duplex and full-duplex mode. More details of the demo are described as follows.

# 2 Hardware overview

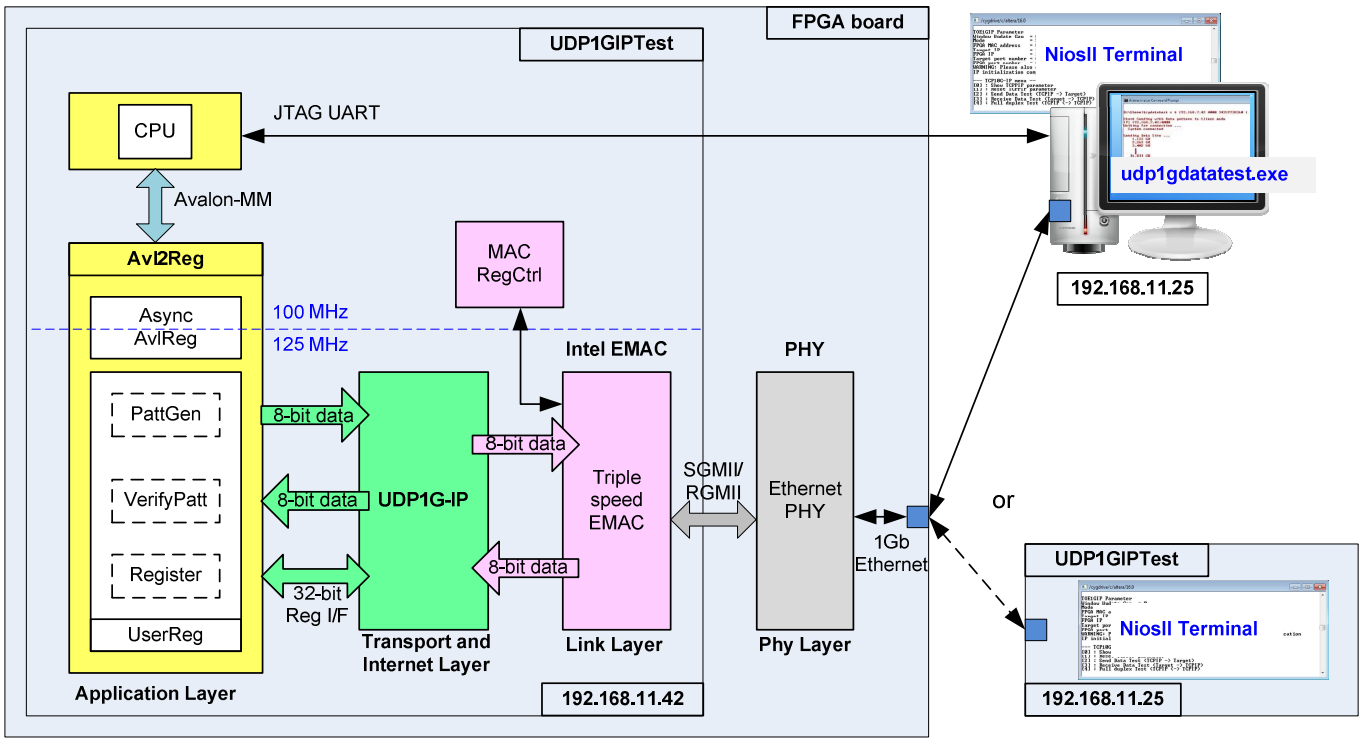


Figure 2-1 Hardware architecture reference design

In test environment, two devices are used for 1 Gb Ethernet transferring. First device runs in client mode and another runs in server mode. To confirm UDP1G-IP operation in both modes, the demo can be tested by using two test environments. First uses two FPGAs for transferring data (one is client and another is server). Second uses one FPGA transferring data with one PC, as shown in Figure 2-1.

In FPGA board, UDP1G-IP connects to Triple-speed EMAC and external Ethernet PHY to complete all UDP/IP layer implementation. User interface of UDP1G-IP is controlled by Avl2Reg. Data interface of UDP1G-IP is connected to UserReg which includes PattGen and VerifyPatt. PattGen is designed to generate test pattern to UDP1G-IP while VerifyPatt is designed to verify received data from UDP1G-IP. Test pattern in the reference design is 32-bit increment data.

Avl2Reg includes register to store test parameters (inputs from user) such as transfer length and transfer direction. User inputs parameters through NiosII terminal. CPU firmware validates all parameters and sets them to the hardware through Avalon-MM bus. Due to the fact that CPU system and UDP1G-IP run in different clock domain, AsyncAvlReg module is used to be asynchronous circuit to support clock-crossing operation and to convert Avalon-MM bus (standard bus in CPU system) to be register interface. CPU in the demo runs in bare-metal OS for simple design. Timer is also included in CPU system for measuring transfer performance.

On PC, “udp1gdatatest.exe” is the test application which is designed to run in this demo. This application sends or receives Ethernet data following UDP/IP protocol. Two applications can be called by using different port numbers to support Full-duplex test (One application is for receiving data and another is for sending data). When running full-duplex mode, both PattGen and VerifyPatt module of UserReg transfer data with UDP1G-IP at the same time.

## 2.1 External PHY

Physical layer in the reference design is implemented by using external PHY chip. For 1 Gb Ethernet, three interface types could be used, i.e. SGMII (Arria10 SoC board), RGMII (ArriaV GX Starter board/CycloneV E board), or GMII.

## 2.2 Triple speed Ethernet MAC

Link layer and PCS/PMA are implemented by Triple-speed Ethernet MAC, provided by Intel. EMAC has two user interfaces, i.e. Avalon stream for transferring data and Avalon-MM for configuration. In demo system, Avalon stream of EMAC is connected to UDP1G-IP while Avalon-MM interface is connected to MACRegCtrl module. The details about the register for EMAC configuration are described in "Configuration Register Space" topic within "Triple-Speed Ethernet MegaCore Function User Guide" document, provided by Intel.
https://www.altera.com/en_US/pdfs/literature/ug/ug_ethernet.pdf

## 2.3 MACRegCtrl

This module is designed to configure parameter of Triple Ethernet MAC and monitors EMAC status through Avalon-MM bus. The logic is simple designed by using state machine and runs only one time after system power up to initialize Ethernet MAC.

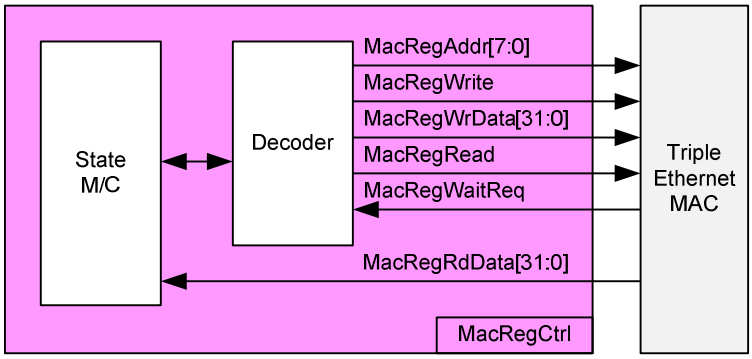


Figure 2-2 MACRegCtrl block diagram

The sequence of initialization sequence is below.
1) Disable transmit and receive path of EMAC
2) Set software reset
3) Set frame length to support jumbo frame
4) Set Tx IFG length
5) Enable transmit and receive path of EMAC

For SGMII mode, the configuration step completes in step 5. Step 6 is necessary for RGMII mode to enable Receive/Transmit timing control function through MDIO.
6) Enable RGMII timing control

## 2.4 UDP1G-IP

UDP1G-IP implements UDP/IP stack and offload engine. Control and status signals are designed by register interface. Data interface is designed by FIFO interface. More details are described in datasheet.
https://dgway.com/products/IP/UDP-IP/dg_udp1gip_data_sheet_intel_en.pdf

## 2.5 Avl2Reg

The hardware is connected to CPU through Avalon-MM bus, similar to other CPU peripherals. Table 2-1 shows CPU memory map of the register inside Avl2Reg. Register could be accessed by write or read command. For write command, it is used to set user parameters to each hardware module. For read command, it is used to monitor status signals of each hardware module.

As shown in Figure 2-3, Avl2Reg has two clock domains, i.e. 100 MHz (CpuClk) for interface with CPU through Avalon-MM bus and 125 MHz (MacTxClk) for interface with UDP1G-IP and EMAC. The module to convert signals from one clock to another clock is AsyncAvlReg.

Othewise, Avl2Reg includes test logic to send data to UDP1G-IP and verify read data from UDP1G-IP. More details of Avl2Reg are described as follows.

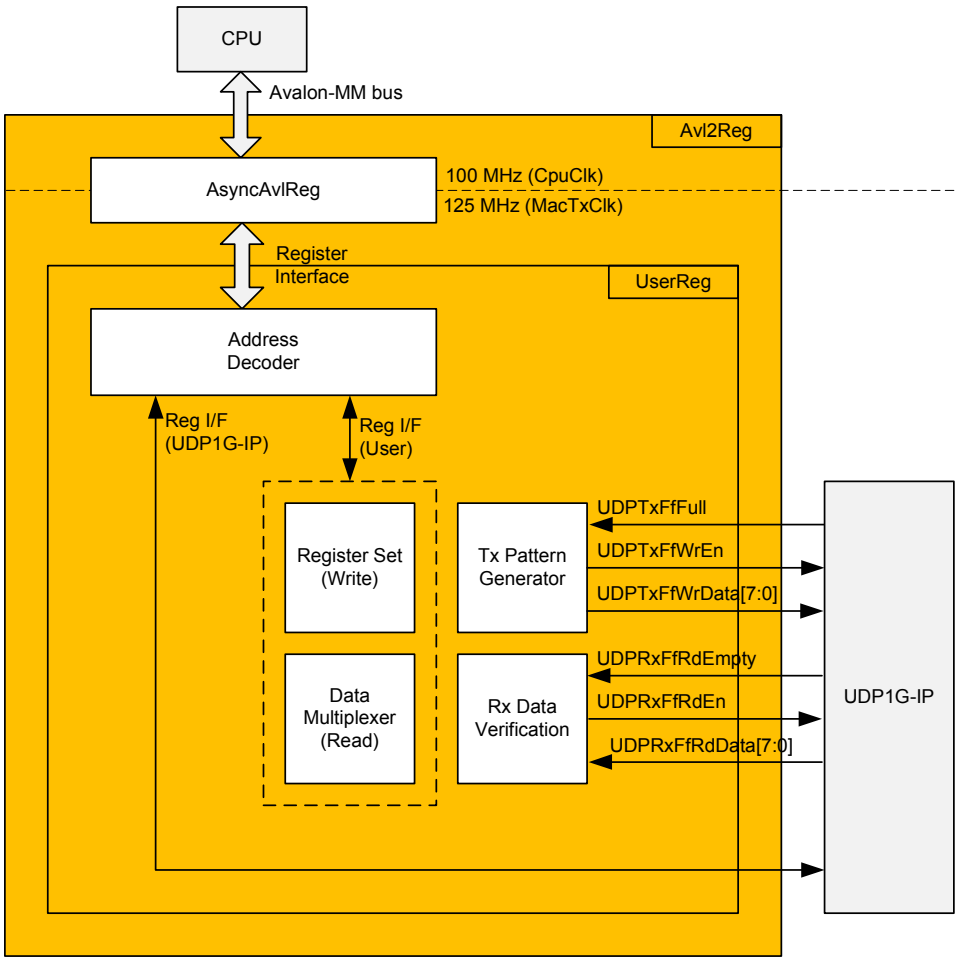


Figure 2-3 Avl2Reg block diagram

### 2.5.1 AsyncAvlReg

This module is designed to convert signal interface of Avalon-MM to be register interface. Also, it includes asynchronous circuit to transfer signal in one clock domain to another clock domain. 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.

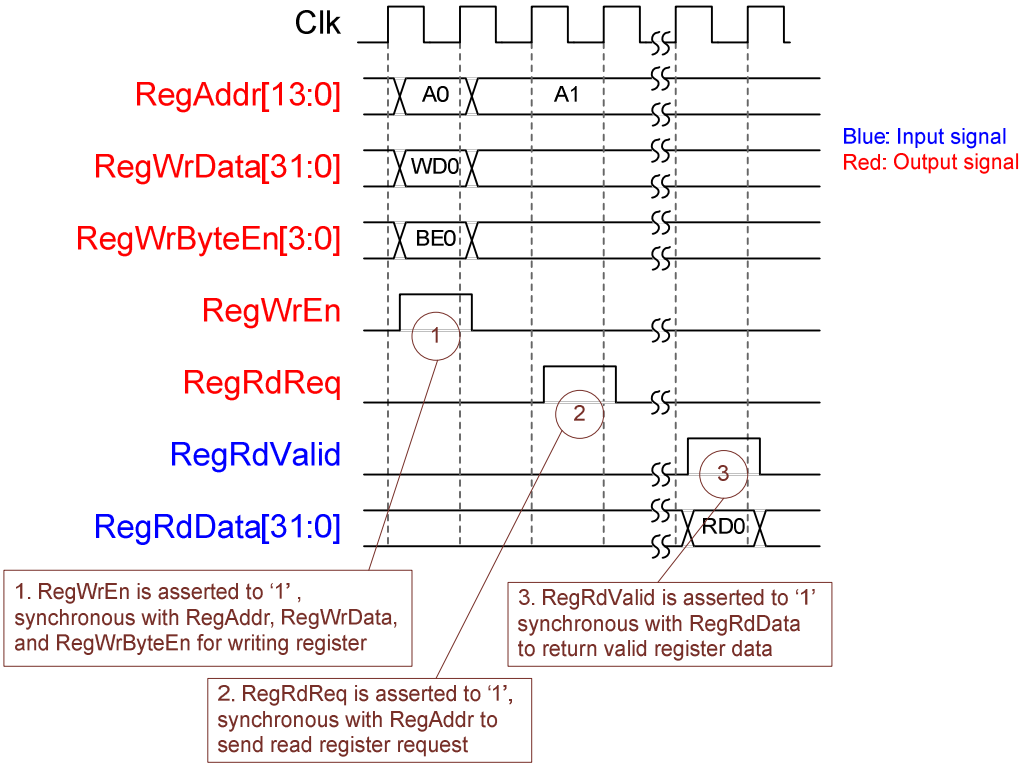


Figure 2-4 Register interface timing diagram

### 2.5.2 UserReg

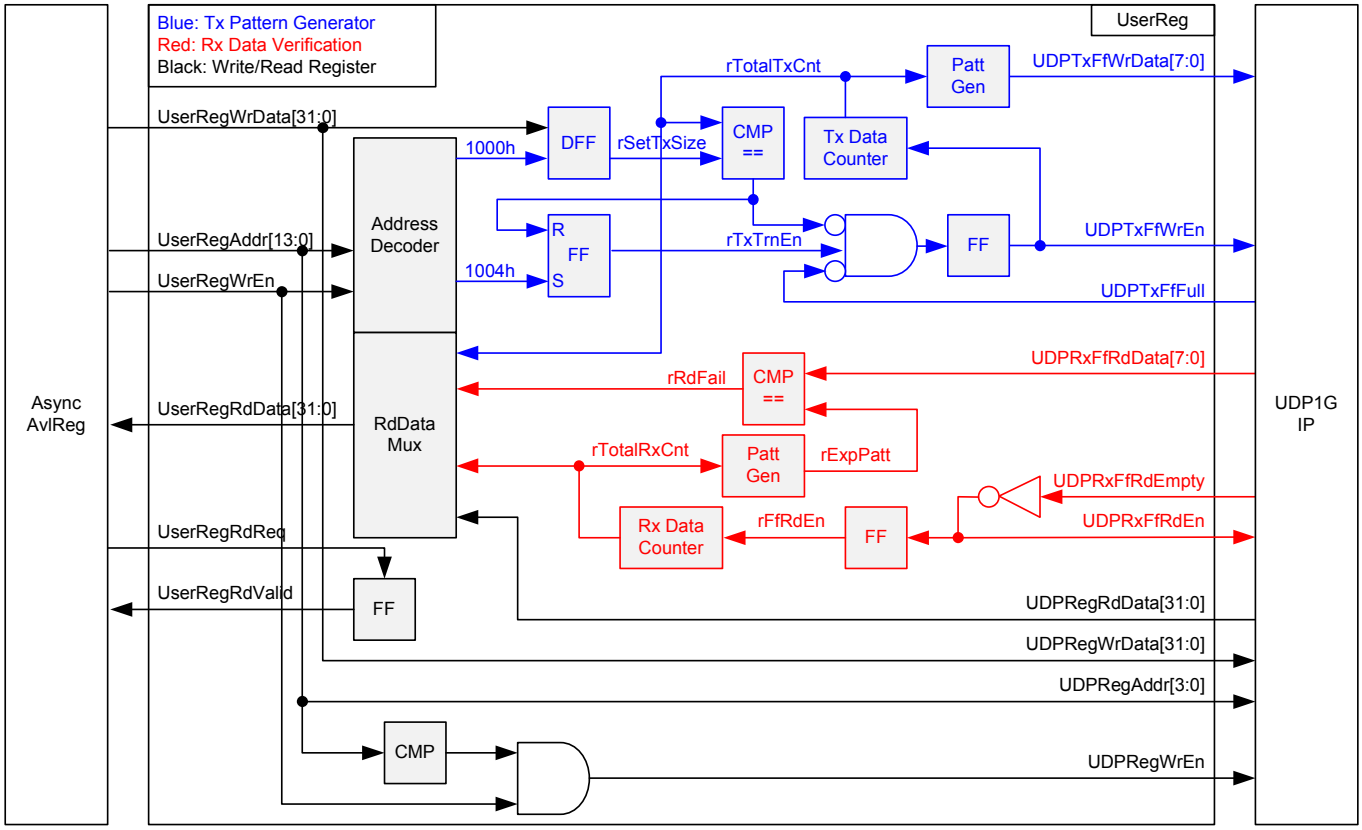


Figure 2-5 UserReg block diagram

Memory map of control and status signals inside UserReg module is shown in Table 2-1.
0x0000 – 0x00FF is mapped to registers inside UDP1G-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 UDP1G-IP range or not. If the address is in UDP1G-IP range, UDPRegWrEn will be asserted to '1'. If the address is internal register range, UserRegWrData is forwarded to set internal register. For example, UserRegAddr=0x1000, UserRegWrData is loaded to rSetTrnSize. This module does not use UserRegWrByteEn signal, 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 UDP1G-IP following address input, 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 UDP1G-IP. rTxTrnEn is asserted to '1' when write register address is 1004h. 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 (rSetTxSize-1).

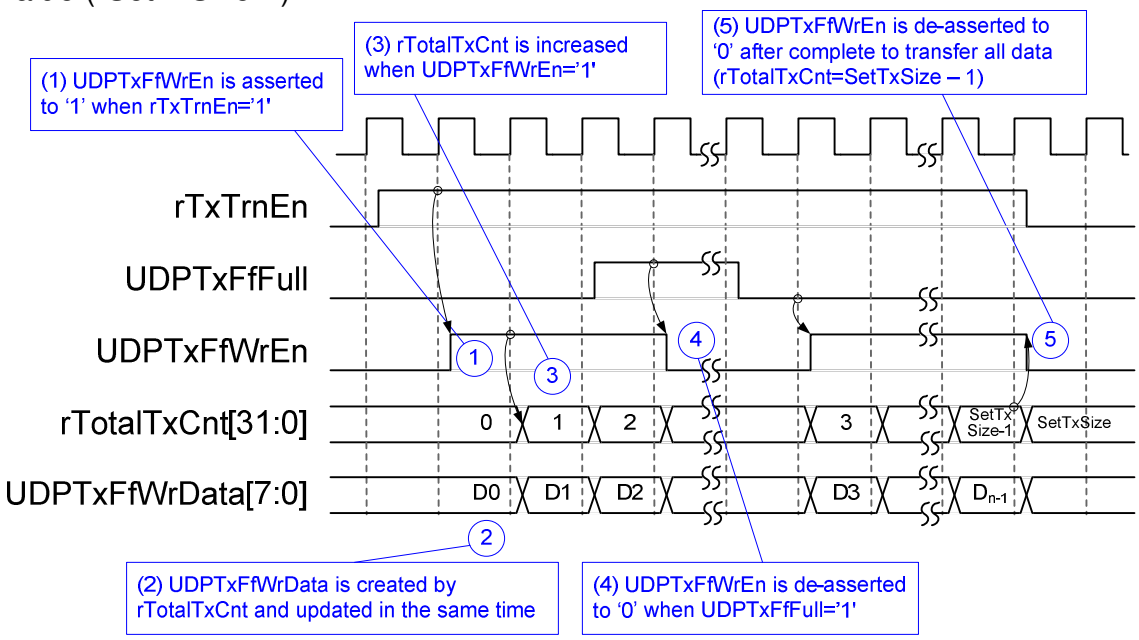


Figure 2-6 Tx Pattern Generator Timing diagram

The logic in red color of Figure 2-5 is designed to verify received data from UDP1G-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 (rExpPatt) which is designed by rTotalRxCnt. rTotalRxCnt is data counter to check total received data from UDPRxFf. Similar to Tx path, expected pattern is 32-bit increment pattern. Fail flag (rRdFail) will be asserted to '1' if Read Data is not equal to expected pattern.

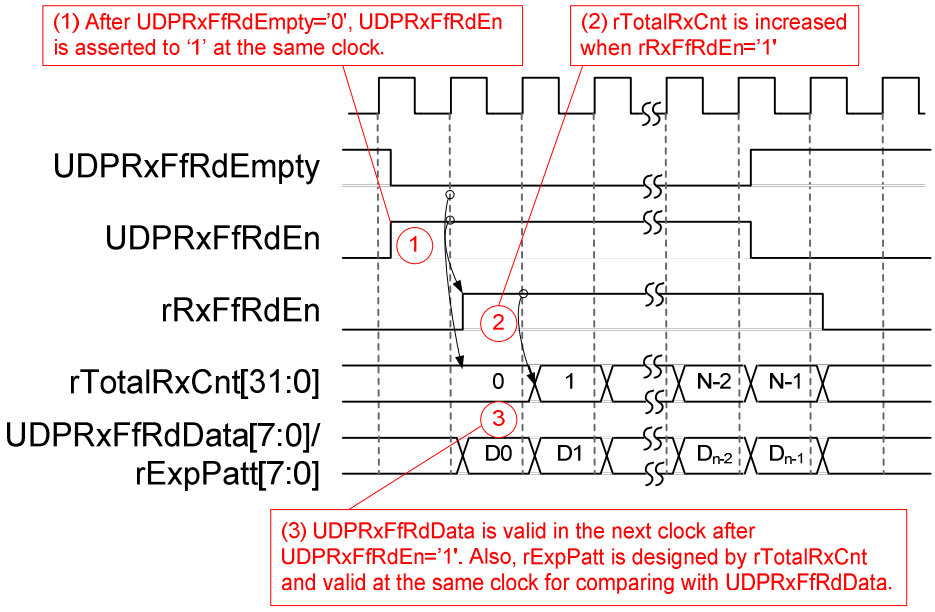


Figure 2-7 Rx Data Verification Timing diagram

## Table 2-1 Register map Definition

| Address<br>Wr/Rd | Register Name<br>(Label in the “udp1gip_demo.c” ) | Description |
|---|---|---|
| BA+0x0000 – BA+0x00FF: UDP1G-IP Register Area (BA=0x0030_0000)<br>More details of each register are described in Table2 of UDP1G-IP datasheet. | | |
| BA+0x0000 | UDP_RST_REG | Mapped to RST register within UDP1G-IP |
| BA+0x0004 | UDP_CMD_REG | Mapped to CMD register within UDP1G-IP |
| BA+0x0008 | UDP_SML_REG | Mapped to SML register within UDP1G-IP |
| BA+0x000C | UDP_SMH_REG | Mapped to SMH register within UDP1G-IP |
| BA+0x0010 | UDP_DIP_REG | Mapped to DIP register within UDP1G-IP |
| BA+0x0014 | UDP_SIP_REG | Mapped to SIP register within UDP1G-IP |
| BA+0x0018 | UDP_DPN_REG | Mapped to DPN register within UDP1G-IP |
| BA+0x001C | UDP_SPN_REG | Mapped to SPN register within UDP1G-IP |
| BA+0x0020 | UDP_TDL_REG | Mapped to TDL register within UDP1G-IP |
| BA+0x0024 | UDP_TMO_REG | Mapped to TMO register within UDP1G-IP |
| BA+0x0028 | UDP_PKL_REG | Mapped to PKL register within UDP1G-IP |
| BA+0x0038 | UDP_SRV_REG | Mapped to SRV register within UDP1G-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 byte unit<br>Rd [31:0] – Current transmit size in byte unit |
| BA+0x1004<br>Wr/Rd | User Command<br>(USER_CMD_REG) | Wr<br>[0] – Start Transmit. Set ‘0’ to start transmit data from UserReg.<br>[1] – Data Verification enable<br>(‘0’: Disable data verification, ‘1’: Enable data verification)<br>Rd<br>[0] – UserReg busy from transmitting data (‘0’: Idle, ‘1’: Busy)<br>[1] – Verification fail (‘0’: No error, ‘1’: Data verification is failed) |
| BA+0x1008<br>Wr/Rd | User Reset<br>(USER_RST_REG) | Wr<br>[0] – Reset user logic. Set ‘1’ to reset to UserReg.<br>This bit is auto-cleared to ‘0’.<br>[8] – Clear latch value of TimerInt. Set ‘1’ to clear latch of TimerInt.<br>Rd<br>[8] – Latch value of TimerInt output from IP<br>(‘0’: Normal, ‘1’: TimerInt=’1’ is detected) |
| BA+0x100C<br>Rd | FIFO status<br>(USER_FFSTS_REG) | Rd [15:0] – Mapped to UDPRxFfRdCnt<br>[24] – Mapped to UDPTxFfFull |
| BA+0x1010<br>Rd | Total receive length<br>(USER_RXLEN_REG) | Rd [31:0] – Current received size in byte unit |

# 3 CPU Firmware Sequence

After FPGA boot-up, user must select the operation mode on FPGA to be client or server. The operation mode is the set value for UDP_SRV_REG register. In client mode, FPGA sends ARP request to get the MAC address from the destination device during initialization sequence. In server mode, FPGA waits ARP request from the destination device and returns ARP reply during initialization sequence.

To run the test by using two FPGAs, the operation mode on each FPGA must be different (one is client and another is server). In case of running FPGA with PC, it is recommended to set FPGA to client mode. It is easier to force PC to return ARP reply after receiving ARP request from FPGA, comparing to find the way to make PC sending ARP request to FPGA.

In the firmware, there are two sets of default parameters, 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 UDP1G-IP completes initialization sequence (UDP_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 UDP1G-IP such as operation mode, source MAC address, destination IP address, source IP address, destination port, and source port. 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 UDP1G-IP parameters such as IP address, source port number. After setting UDP1G-IP register, CPU resets the IP to re-initialize by using new parameters. CPU monitors busy flag to wait until the initialization is completed. The sequence of reset sequence is shown as follows.

1) Display current parameter value to the console.
2) Receive input parameters from user and check input value whether it is in a valid range or not. If the input is invalid, the invalid input will not be changed.
3) Force reset to UDP1G-IP by setting UDP_RST_REG[0]='1'.
4) Set all parameters to UDP1G-IP register such as UDP_SML_REG, UDP_DIP_REG.
5) De-assert UDP1G-IP reset by setting UDP_RST_REG[0]='0'.
6) Reset UserReg by setting USER_RST_REG[0]='1'.
7) Monitor UDP1G-IP busy flag (UDP_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. During the test, 32-bit increment data is generated from the logic and sent to the target (PC or FPGA). The target verifies the received data by test application when running with PC or by verification module when running with FPGA. The operation is completed when total data are transferred from FPGA to PC/FPGA completely. The sequence of this test is as follows.

1) Receive transfer size 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 UDP1G-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 UDP1G-IP to start operation. Packet size is set to UDP_PKL_REG and total size is set to UDP_TDL_REG. Finally, UDP_CMD_REG is set to 1 to start IP sending data.
5) Wait until UDP1G-IP completes operation by monitoring IP busy flag (UDP_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

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 1 sec 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 “udp1gdatatest” application on two consoles. First application is used to receive data with FPGA and another is used to send data with FPGA. The port using in two applications must be different.

When running full duplex test by using two FPGAs, one port is applied for both sending and receiving data. 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 UDP1G-IP registers, i.e. packet size (UDP_PKL_REG), total transfer size (UDP_TDL_REG), and write command (UDP_CMD_REG=1). IP starts sending data operation after UDP_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 UDP1G-IP (UDP_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 one second (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

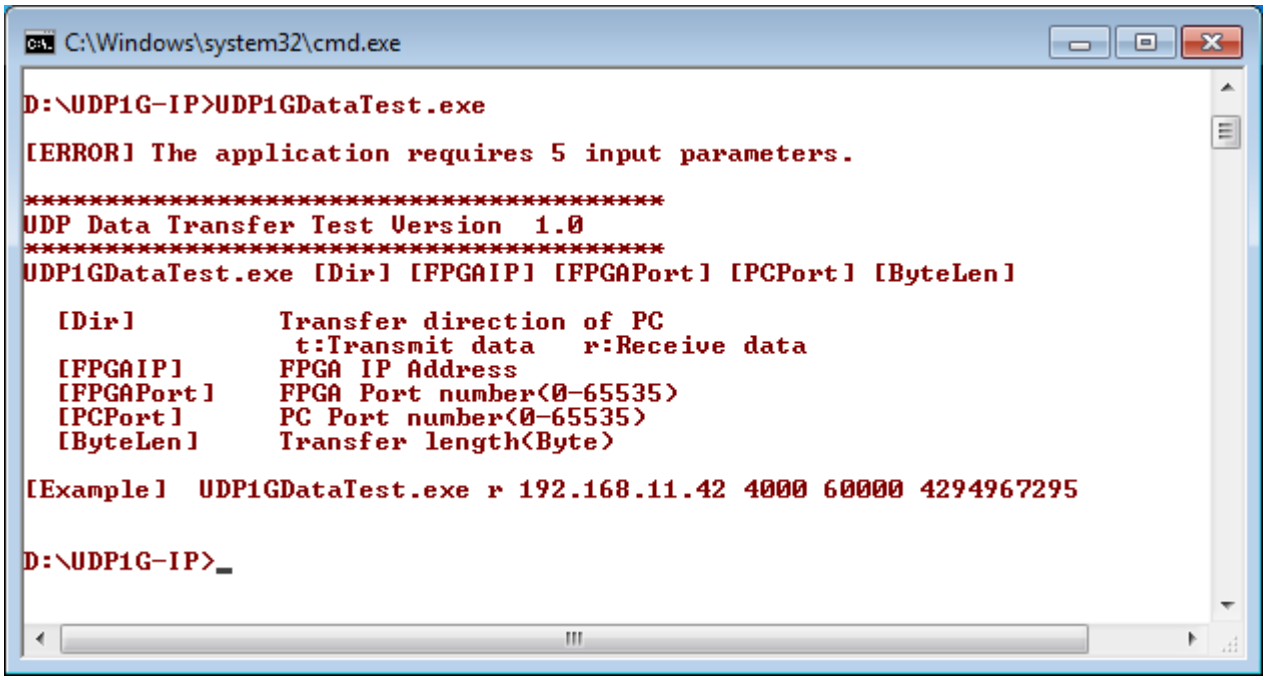


Figure 4-1 udp1gdatatest application parameter

"udp1gdatatest" is an application on PC for sending or receiving UDP data. There are five parameters. The parameter input should be matched to parameter setting on FPGA. 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 192.168.11.42)
3) FPGAPort : Port number of FPGA (default is 4000)
4) PCPort : PC port number for sending or receiving data
   (default is 60001 for transferring data from PC to FPGA
   or 60000 for transferring data from FPGA to PC)
5) ByteLen : Transfer length for sending or receiving in byte unit.

### 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 inputs and then connect.
(4) Loop to verify data until total received data is equal to set value or no more received data within 1 sec. 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 1 sec 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 | 2-Mar-17 | Initial Release |
| 1.1 | 14-Aug-18 | Update design to interface by JTAG UART |