UDP10GRx IP reference design

Rev1.0 22-Apr-21

1 Introduction

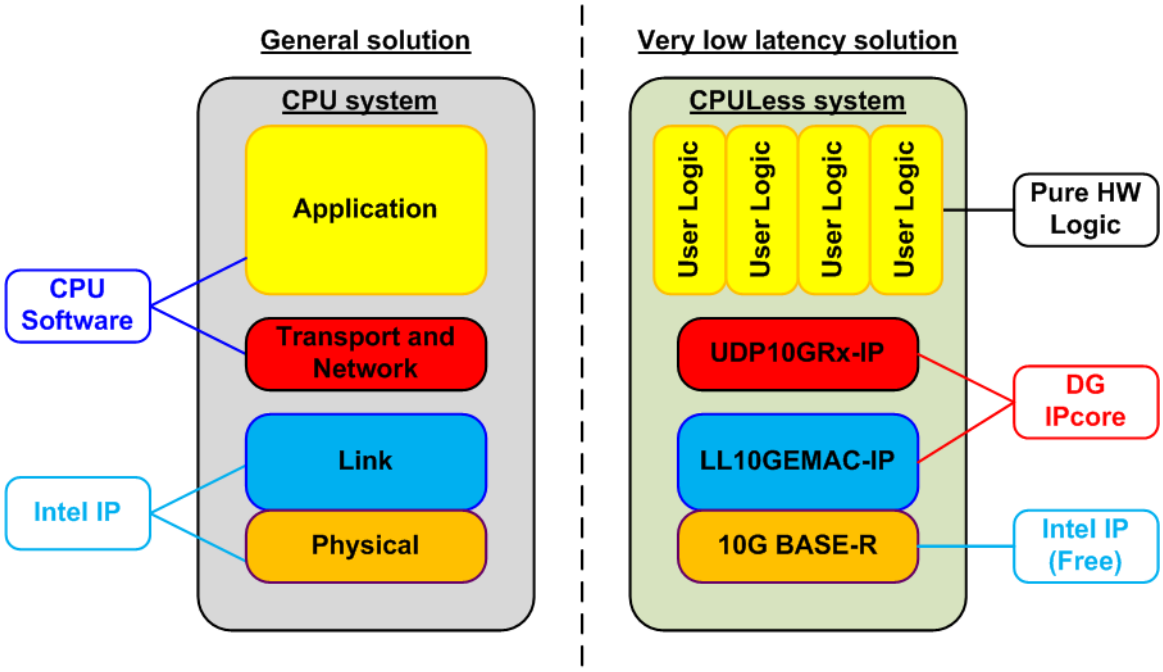

Figure 1-1 Low latency solution

When FPGA is applied for implementing UDP/IP data processor, the general solution is designed by using CPU system running UDP/IP stack as shown in the left side of Figure 1-1. Link layer and physical layer are implemented by Intel IP core, Low Latency Ethernet 10G MAC, as shown on the left side of Figure 1-1.Though this solution is flexible for many applications on CPU, the result shows much latency time for processing both UDP/IP stack and the application.

To achieve the lowest latency solution, the design on the right side of Figure 1-1 is designed, the full hardware logic system for processing UDP/IP stack. This solution is fit with the time-sensitive application that needs to implement the user logic by the hardware logic for receiving UDP data with UDP10GRx-IP. UDP10GRx-IP designs UDP/IP stack with ultra-low latency. Also, it is recommended to connect with the low latency 10G Ethernet MAC IP (LL10GEMACIP) to achieve the lowest latency system. The lowest layer of hardware, PMA layer, is provided by Intel as a free PMA IP.

The UDP data, extracted from the valid packet, is forwarded to the user logic via data interface. There are four data control signals for transferring up to four-session data in UDP10GRx-IP. Therefore, the user logic can be designed by using four independent modules.

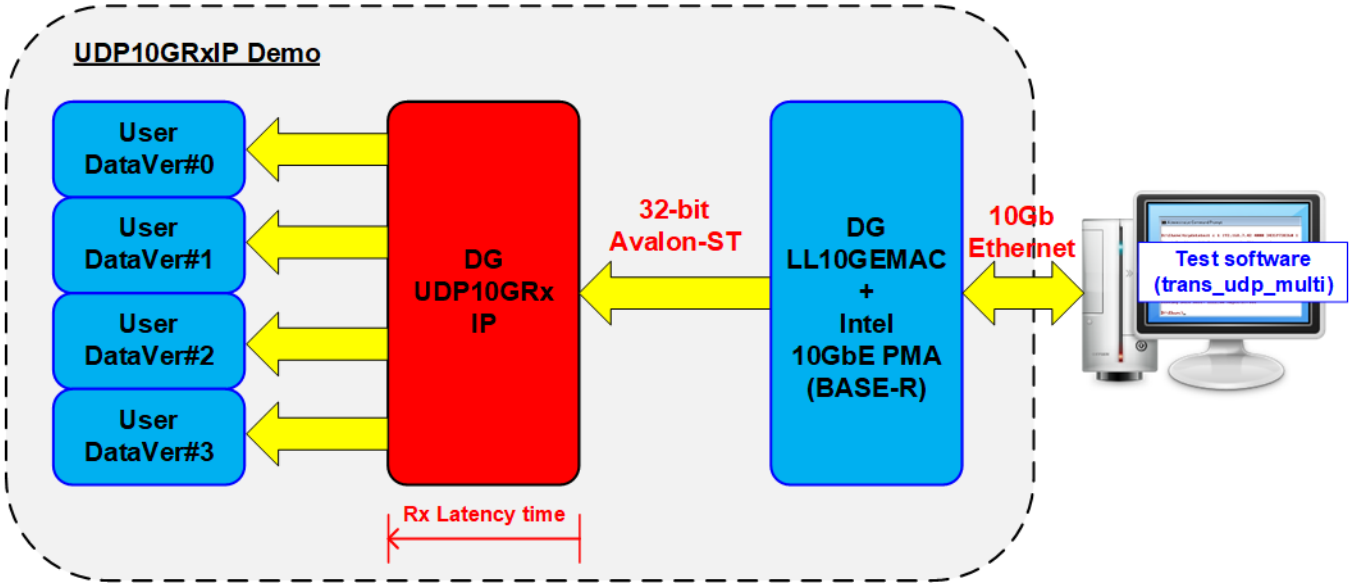

Figure 1-2 Test logic for UDP10GRx-IP

To show UDP/IP stack implementation by UDP10GRx-IP with achieving low latency time, the simple test logic is designed, as shown in Figure 1-2. Four data verification modules (UserDataVer) are connected with the user interface of UDP10GRx-IP for verifying the data of four sessions. Test software, trans_udp_multi.exe, running on the PC are the data source to send UDP/IP packet of four sessions via 10Gb Ethernet. DG LL10GEMAC-IP and Intel 10GbE PMA for BASE-R (Transceiver PHY IP) implement the low-level interface module to transfer Ethernet packet with UDP10GRx-IP. The latency time of data path, decoded in UDP10GRx-IP, is measured by using the timer.

CPU system is included for user interface via JTAG UART. Network parameters of the test system can be set by the user from the NiosII command shell. Also, the test result and the progress of test operation are returned to CPU for displaying on the NiosII command shell. More details of the demo are described as follows.

2 Hardware overview

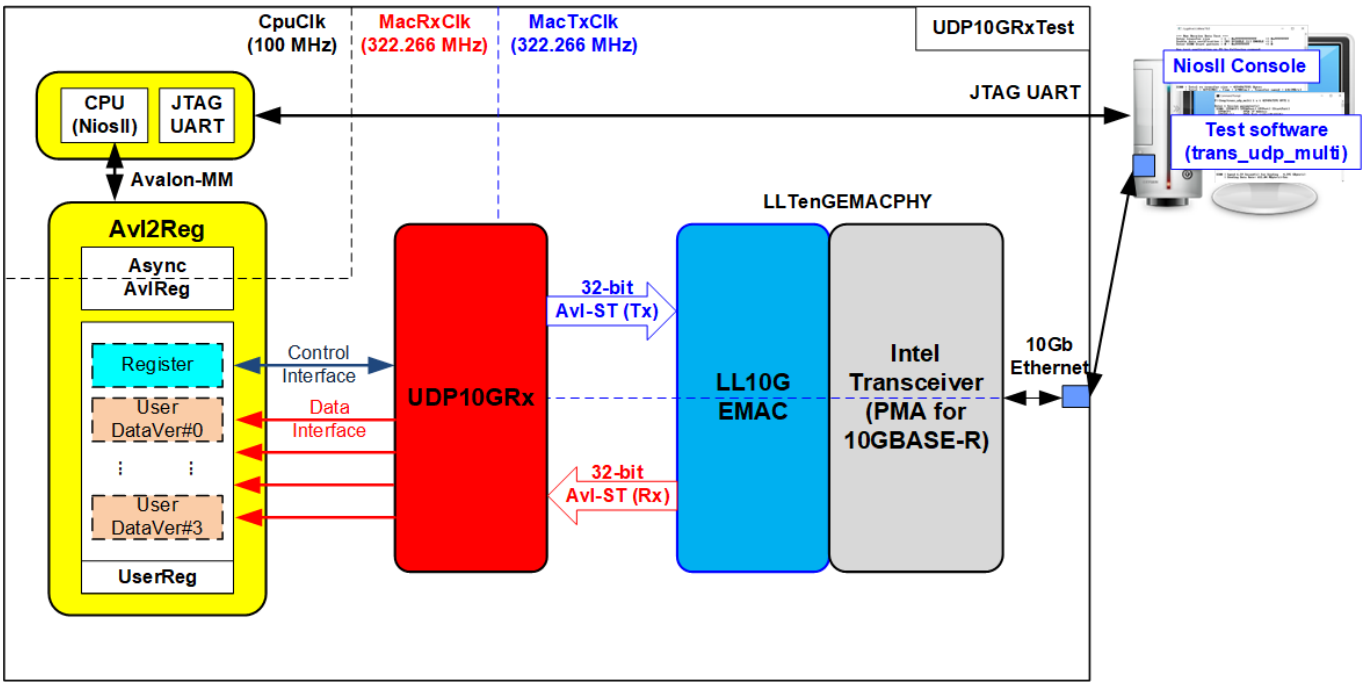

Figure 2-1 UDP10GRxTest Block Diagram

The test system includes CPU for easy user interface and flexible test. Test parameters such as network parameters and transfer size are controlled by the user inputs. Also, the current status of the test system such as current transfer size is displayed on NiosII command shell. To connect the hardware with CPU system, Avalon-MM bus standard must be implemented. Avl2Reg is the interface module to convert Avalon-MM interface to be the user interface of UDP10GRx module. Avl2Reg includes AsyncAvlReg which is designed to be asynchronous module between CPU system clock (CpuClk) and UDP10GRx user interface clock (MacRxClk).

The user interface of UDP10GRx is divided to two interfaces - control interface and data interface. The control interface is controlled by CPU system while the data interface is connected to UserDataVer module for verifying the received data. Four UserDataVer modules are applied for verifying four session data, outputs from UDP10GRx module. Another side of UDP10GRx-IP is connected to Low-Latency Ethernet MAC (LL10GEMACIP) by using 32-bit Avalon stream interface (Avalon-ST). LL10GEMAC-IP implements the Ethernet MAC layer and PCS layer with less latency time. Tx and Rx interface of Avalon-ST are run in the different clock domains, MacTxClk and MacRxClk respectively.

LL10GEMAC-IP provided by Design Gateway requires to run with Intel Transceiver which is configured to be PMA module for 10GBASE-R interface.

Another side of 10Gb Ethernet is designed to connect with TestPC which runs Test software, trans_udp_multi.exe, provided by Design Gateway. “trans_udp_multi.exe” is designed to send UDP data up to four sessions at the same time by using Multicast mode or Unicast mode.

2.1 Intel Transceiver (PMA for 10GBASE-R)

PMA IP core for 10Gb Ethernet (BASE-R) can be generated by using Quartus IP catalog. In FPGA Transceivers Wizard, the user uses the following settings.

- Transceiver configuration rules : PCS Direct
- Data rate : 10312.5 Mbps
- PCS Direct interface width : 32

More details are described in the following link.
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-10/ug_arria10_xcvr_phy.pdf

2.2 LL10GEMAC

The IP core by Design Gateway implements low-latency EMAC and PCS logic for 10Gb Ethernet (BASE-R) standard. The user interface is 32-bit Avalon-stream bus. Please see more details from LL10GEMAC datasheet on our website.
https://dgway.com/products/IP/Lowlatency-IP/dg_ll10gemacip_data_sheet_intel.pdf

2.3 UDP10GRx

The IP core by Design Gateway is designed to receive and decode UDP/IP packet from EMAC. UDP data is extracted from the received packet and then forwarded to the user with very low latency time. Up to four sessions are supported by the IP. More details of UDP10GRx are described in UDP10GRx datasheet, provided on our website.
https://dgway.com/products/IP/Lowlatency-IP/dg_udp10grxip_data_sheet_intel.pdf

2.4 CPU and Peripherals

32-bit Avalon-MM is applied to be the bus interface for the CPU accessing the peripherals such as Timer and JTAG 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 Avalon-MM 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 CPU system is built with one additional peripheral to access the test logic. The base address and the range for accessing the test logic are defined in the CPU system. Therefore, the hardware logic must be designed to support Avalon-MM bus standard for supporting CPU writing and reading. Avl2Reg module is designed to connect the CPU system as shown in Figure 2-2.

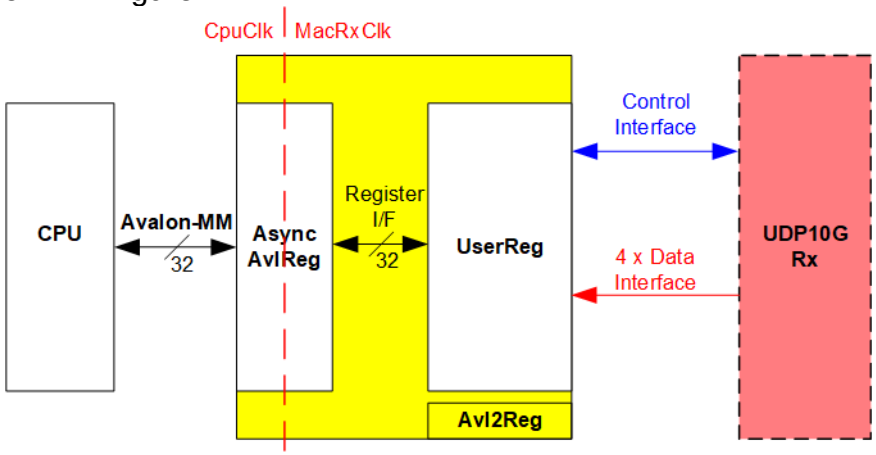

Figure 2-2 CPU and peripheral hardware

Avl2Reg consists of AsyncAvlReg and UserReg. AsyncAvlReg is designed to convert the Avalon-MM signals to be the simple register interface which has 32-bit data bus size (similar to Avalon-MM data bus size). Also, AsyncAvlReg includes asynchronous logic to support clock crossing between CpuClk domain and MacRxClk domain.

UserReg includes the register file of the input signals for UDP10GRx-IP. The output signals from UDP10GRx-IP are also mapped to UserReg for CPU checking the current status. More details of AsyncAvlReg and UserReg are described as follows.

2.4.1 AsyncAvlReg

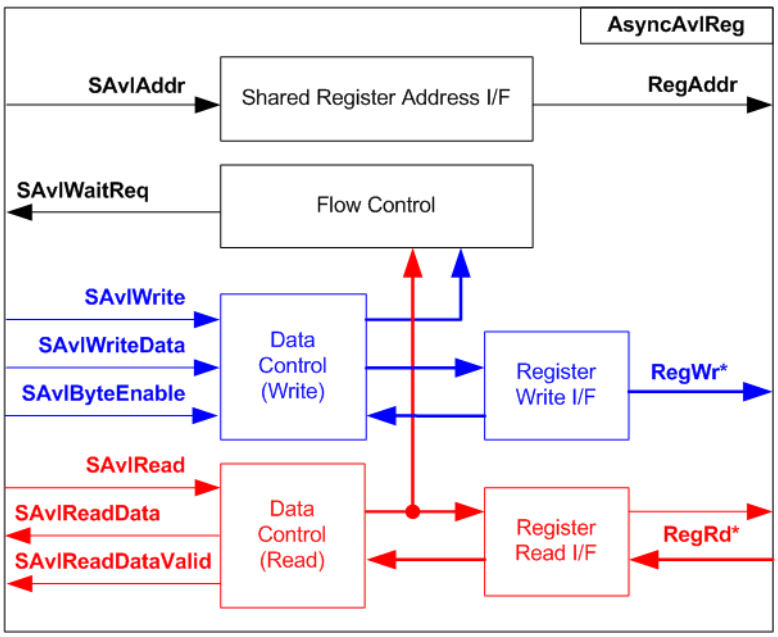

Figure 2-3 AsyncAvlReg Interface

The signal on Avalon-MM bus interface can be split into three groups, i.e., Write channel (blue color), Read channel (red color), and Shared control channel (black color). More details of Avalon-MM interface specification are described in following document.
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf

According to Avalon-MM specification, one command (write or read) can be operated at a time. The logics inside AsyncAvlReg are split into three groups, i.e., Write control logic, Read control logic, and Flow control logic. Flow control logic controls SAvlWaitReq to hold the next request from Avalon-MM interface if the current request does not finish. Write control and Write data I/F of Avalon-MM bus are latched and transferred to be Write register interface with clock-crossing registers. Similarly, Read control I/F are latched and transferred to be Read register interface with clock-crossing registers. After that, the returned data from Register Read I/F is transferred to Avalon-MM bus by using clock-crossing registers. Address I/F of Avalon-MM is latched and transferred to Address register interface as well.

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.

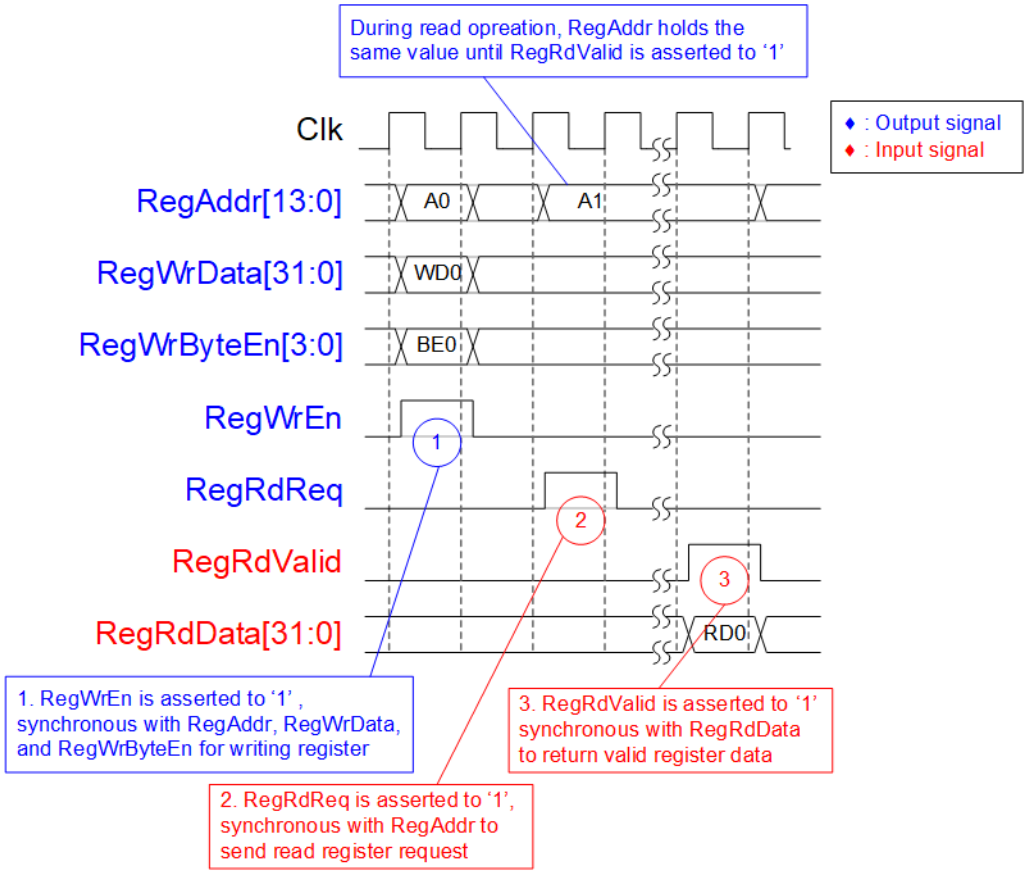

Figure 2-4 Register interface timing diagram

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 4-byte data enable. 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, AsyncAvlReg 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, AsyncAvlReg forwards the read value to SAvlRead interface.

2.4.2 UserReg

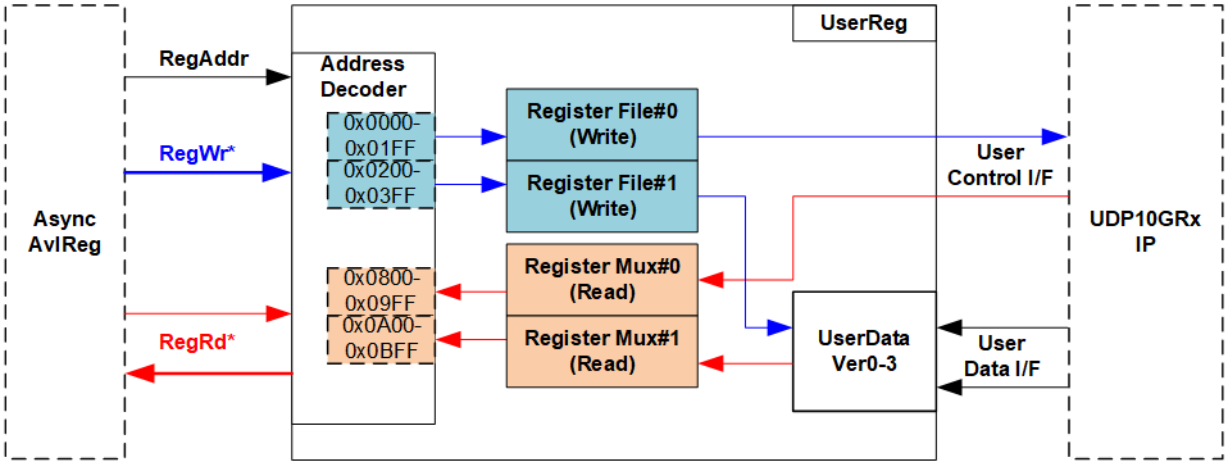

Figure 2-5 UserReg block diagram

UserReg consists of many registers for interfacing with user control interface of UDP10GRx-IP module. The address for write or read access is decoded by Address decoder to select the active register. There are four regions mapping in UserReg, two for write only and two for read only. Also, UserReg includes four UserDataVer modules for interfacing with User data interface of UDP10GRx.

As shown in Figure 2-5, the address is split into four areas.

1) 0x0000 – 0x01FF: Write only area for setting the input signals of UDP10GRx such as network parameters.
2) 0x0200 – 0x03FF: Write only area for setting the input signals of four UserDataVer modules such as start value of test pattern.
3) 0x0800 – 0x09FF: Read only area for reading the output signals of UDP10GRx such as Error flags and active channel.
4) 0x0A00 – 0x0BFF: Read only area for reading the output signals of four UserDataVer modules such as current transfer size.

Address decoder decodes the upper bit of RegAddr for selecting the active hardware. The register file inside UserReg is 32-bit bus size, so write byte enable (RegWrByteEn) is not used. To write hardware registers, the CPU must use 32-bit pointer to place 32-bit valid value on the write data bus.

To read register, 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 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 three clock cycles, so RegRdValid is created by RegRdReq with asserting three D Flip-flops. 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 udp10grxtest.c")	Description
BA+0x0000 – BA+0x01FF: Input signals for UDP10GRx-IP (Write access only)		
BA+0x0000	UDP_SSEN_REG	[3:0]: Mapped to SSEnable of UDP10GRx-IP
BA+0x0004	UDP_MCEN_REG	[0]: Mapped to McastEn of UDP10GRx-IP
BA+0x0040	UDP_SML_REG	[31:0]: Mapped to SrcMacAddr[31:0] of UDP10GRx-IP
BA+0x0044	UDP_SMH_REG	[15:0]: Mapped to SrcMacAddr[47:32] of UDP10GRx-IP
BA+0x0048	UDP_SIP_REG	[31:0]: Mapped to SrcIPAddr of UDP10GRx-IP
BA+0x0100	UDP_SPN0_REG	[15:0]: Mapped to SrcPort0 of UDP10GRx-IP
BA+0x0104	UDP_DPN0_REG	[15:0]: Mapped to DstPort0 of UDP10GRx-IP
BA+0x0108	UDP_DIP0_REG	[31:0]: Mapped to DstIPAddr0 of UDP10GRx-IP
BA+0x010C	UDP_MIP0_REG	[31:0]: Mapped to McastIPAddr0 of UDP10GRx-IP
BA+0x0120	UDP_SPN1_REG	[15:0]: Mapped to SrcPort1 of UDP10GRx-IP
BA+0x0124	UDP_DPN1_REG	[15:0]: Mapped to DstPort1 of UDP10GRx-IP
BA+0x0128	UDP_DIP1_REG	[31:0]: Mapped to DstIPAddr1 of UDP10GRx-IP
BA+0x012C	UDP_MIP1_REG	[31:0]: Mapped to McastIPAddr1 of UDP10GRx-IP
BA+0x0140	UDP_SPN2_REG	[15:0]: Mapped to SrcPort2 of UDP10GRx-IP
BA+0x0144	UDP_DPN2_REG	[15:0]: Mapped to DstPort2 of UDP10GRx-IP
BA+0x0148	UDP_DIP2_REG	[31:0]: Mapped to DstIPAddr2 of UDP10GRx-IP
BA+0x014C	UDP_MIP2_REG	[31:0]: Mapped to McastIPAddr2 of UDP10GRx-IP
BA+0x0160	UDP_SPN3_REG	[15:0]: Mapped to SrcPort3 of UDP10GRx-IP
BA+0x0164	UDP_DPN3_REG	[15:0]: Mapped to DstPort3 of UDP10GRx-IP
BA+0x0168	UDP_DIP3_REG	[31:0]: Mapped to DstIPAddr3 of UDP10GRx-IP
BA+0x016C	UDP_MIP3_REG	[31:0]: Mapped to McastIPAddr3 of UDP10GRx-IP
BA+0x0200 – BA+0x07FF: Input signals for UserReg (Write access only)		
BA+0x0200	User Command Reg (USER_CMD_REG)	[0]: Start flag of UserDataVer. Set to '1' to start the operation. This flag is auto-cleared. This signal is also applied to clear the timer to check latency time of UDP10GRx-IP. [1]: Verification enable. '0': Disable data verification, '1': Enable data verification
BA+0x0204	User Clear Reg (USER_CLR_REG)	[0]: Reset flag to clear error signal. Set to '1' to clear error signals inside UserDataVer. This flag is auto-cleared.
BA+0x0210	User Start Pattern0 Reg (USER_PAT0_REG)	[31:0]: Start value of 32-bit incremental pattern for verifying data in UserDataVer0.
BA+0x0214	User Start Pattern1 Reg (USER_PAT1_REG)	[31:0]: Start value of 32-bit incremental pattern for verifying data in UserDataVer1.
BA+0x0218	User Start Pattern2 Reg (USER_PAT2_REG)	[31:0]: Start value of 32-bit incremental pattern for verifying data in UserDataVer2.
BA+0x021C	User Start Pattern3 Reg (USER_PAT3_REG)	[31:0]: Start value of 32-bit incremental pattern for verifying data in UserDataVer3.

Address Wr/Rd	Register Name (Label in the udp10grxtest.c")	Description
BA+0x0800 – BA+0x09FF: Output signals of UDP10GRx-IP and UserReg (Read access only)		
BA+0x0800	EMAC_STS_REG	[0]: Mapped to Linkup of LL10GEMAC-IP ('0'-Link down, '1'-Link up).
BA+0x0804	EMAC_VER_REG	[31:0]: Mapped to IPVersion of LL10GEMAC-IP
BA+0x0808	UDP_VER_REG	[31:0]: Mapped to IPVersion of UDP10GRx-IP
BA+0x080C	UDP_TESTPIN_REG	[31:0]: Mapped to TestPin of UDP10GRx-IP
BA+0x0810	UDP_SSAC_REG	[3:0]: Mapped to SSActive of UDP10GRx-IP
BA+0x0814	User Status Reg (USER_STS_REG)	[0]: Data verification fail flag from UserDataVer0 ('0'-No error, '1'-Error) [1]: UDPRxError from UDP10GRx-IP shows error in session#0 ('0'-No error, '1'-Error) [5:4]: Similar to USER_STS_REG[1:0], the flag is applied for session#1. [9:8]: Similar to USER_STS_REG[1:0], the flag is applied for session#2. [13:12]: Similar to USER_STS_REG[1:0], the flag is applied for session#3.
BA+0x0818	UDP Error Reg (UDP_ERR_REG)	[7:0]: Latch value of UDPRxError when error is found in session#0. [15:8]: Latch value of UDPRxError when error is found in session#1. [23:16]: Latch value of UDPRxError when error is found in session#2. [31:24]: Latch value of UDPRxError when error is found in session#3.
BA+0x081C	Rx Time Reg (USER_RXTIM_REG)	[31:0]: Latency time of received data in UDP10GRx-IP. Time unit is 3.1 ns.
BA+0x0A00 – BA+0x0BFF: Output signals of UserDataVer (Read access only)		
BA+0x0A00	Current Received Size0 (Low) (USER_RXLENL0_REG)	[31:0]: The lower 32-bit of current received data size in byte unit when receiving data from session#0.
BA+0x0A04	Current Received Size0 (High) (USER_RXLENH0_REG)	[15:0]: The upper 16-bit of current received data size in byte unit when receiving data from session#0.
BA+0x0A08	Data Failure Position0 (Low) (USER_RDFAILL0_REG)	[31:0]: The lower 32-bit of failure position of the 1st failure data in byte unit when receiving data from session#0.
BA+0x0A0C	Data Failure Position0 (High) (USER_RDFAILH0_REG)	[15:0]: The upper 16-bit of failure position of the 1st failure data in byte unit when receiving data from session#0.
BA+0x0A10	Expect Data0 (USER_EXPPAT0_REG)	[31:0]: Expected value of the 1st failure data when receiving data from session#0.
BA+0x0A14	Read Data0 (USER_RDPAT0_REG)	[31:0]: Read value of the 1st failure data when receiving data from session#0.
BA+0x0A20- BA+0x0A34	USER_RXLENL1_REG – USER_RDPAT1_REG	Similar to BA+0x0A00 – BA+0A14, the registers are applied for receiving data from session#1.
BA+0x0A40- BA+0x0A54	USER_RXLENL2_REG – USER_RDPAT2_REG	Similar to BA+0x0A00 – BA+0A14, the registers are applied for receiving data from session#2.
BA+0x0A60- BA+0x0A74	USER_RXLENL3_REG – USER_RDPAT3_REG	Similar to BA+0x0A00 – BA+0A14, the registers are applied for receiving data from session#3.

2.4.3 UserDataVer

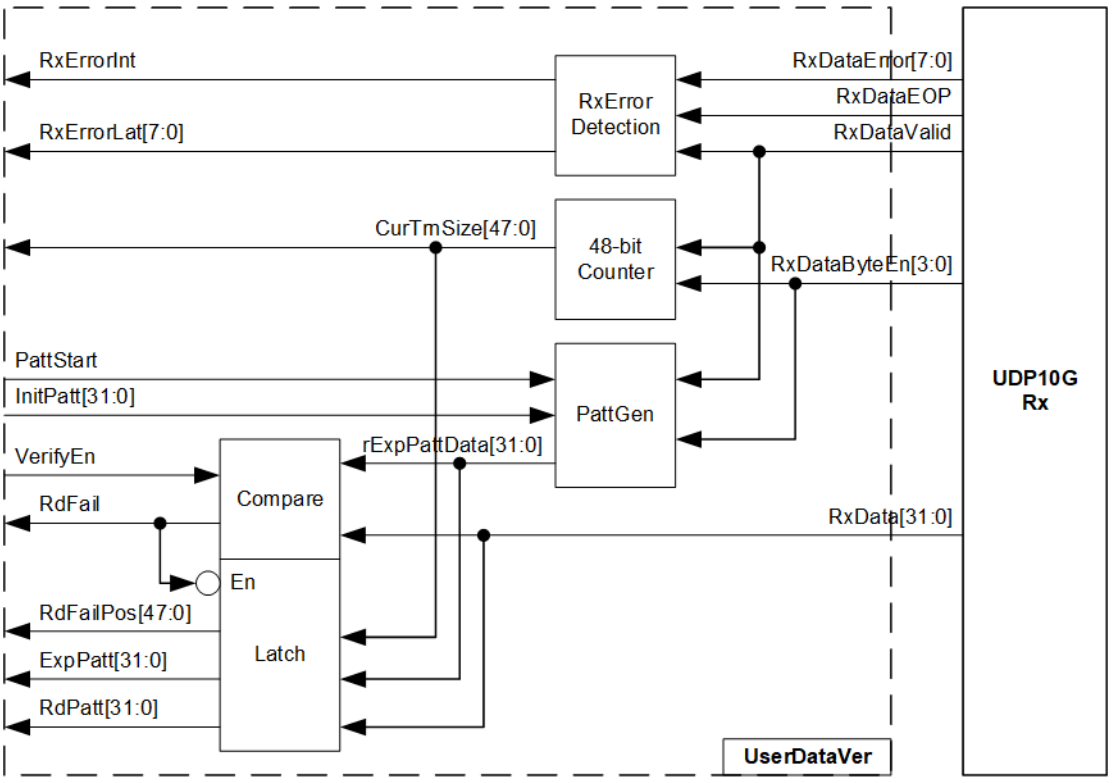

Figure 2-6 UserDataVer

UserDataVer is designed to verify the received packet output from UDP10GRx-IP. Two logic functions are implemented, i.e., error monitoring and data verifying. The error flag of UDP10GRx-IP is valid at the end of received packet. Therefore, RxError Detection module is designed to read Error signal (RxDataError) at the end of frame (RxDataEOP='1' and RxDataValid='1'). The interrupt (RxErrorInt) is asserted to '1' with latch value of Error flag (RxErrorLat) when some errors are found (RxDataError is not equal to 0).

When data is transferred, current number of received data is returned for user monitoring. It is designed by using 48-bit counter (CurTrnSize), increased by data valid (RxDataValid='1'). Byte enable (RxDataByteEn) is also monitored to check the number of valid bytes in each clock cycle. It may be equal to 1-4 bytes.

To verify data, PattGen is designed to create the expected value (rExpPattData) of the received data. Test pattern is 32-bit incremental data which can set the start value (InitPatt) from the user. The start value is loaded when the test begins (PattStart='1'). Fail flag (RdFail) is asserted to '1' when the received data (RxData) is not equal to the expected value and data verification is enabled (VerifyEn='1'). To display verification error details when the error is found, RdFail is also applied to latch the information of the first data which is failure, i.e., data position (RdFailPos), expected value (ExpPatt), and read value (RdPatt). The latch registers are designed by updating the information until the failure is detected.

3 CPU Firmware (FPGA)

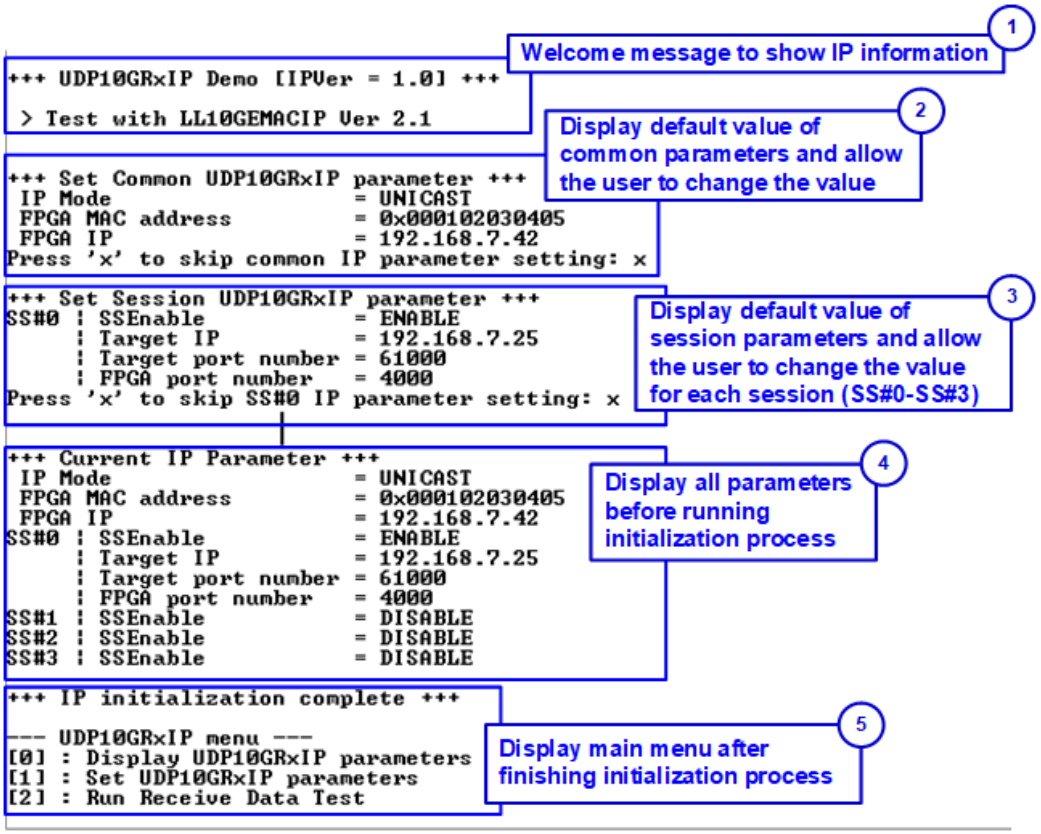

Figure 3-1 Message on the console in initialization process

In reference design, CPU firmware is implemented as bare-metal OS for easily handling with the hardware. After the test system is run, the hardware starts the initialization process, as shown in Figure 3-1. There are five steps to set up the hardware before starting initialization process, described in more details as follows.

1) CPU reads UDP10GRx-IP and LL10GEMACIP information and displays on the NiosII command shell. After that, CPU waits until Ethernet connection is ready by monitoring Linkup status (EMAC_STS_REG[0]). This step is finished when EMAC_STS_REG[0]='1' (Linkup).
2) The parameters for setting UDP10GRx-IP are divided into two types, i.e., common parameters which are shared parameters for all sessions and session parameters which are independent for each session. CPU displays default value of common parameters on the consoles, i.e., IP Mode (Multicast or Unicast), FPGA MAC address, and FPGA IP address. The user selects to set the new value of common parameters or skip the setting step by using default value. The input value of each parameter from the user is verified by CPU. If the input value is invalid, the parameter does not change the value.
3) CPU displays default value of session parameters, i.e., session enable, Target IP address, Multicast IP address, Target port number, and FPGA port number. The session order for displaying parameters is session#0, #1, #2, and #3. The user selects to set the new value for session parameters or skip the setting step by using default value, similar to the common parameters. After that, the input is verified by CPU.
4) CPU sets all common parameters and session parameters to the hardware register, i.e.,
   UDP_SML/H_REG = FPGA MAC address
   UDP_SIP_REG = FPGA IP address
   UDP_MCEN_REG = IP Mode
   UDP_SPN0-3_REG = FPGA port number
   UDP_DPN0-3_REG = Target port number
   UDP_DIP0-3_REG = Target IP address
   UDP_MIP0-3_REG = Multicast IP address
   UDP_SSEN_REG = Session enable
5) CPU waits until the initialization process is finished by monitoring UDP_SSAC_REG. After all sessions finish the initialization, UDP_SSAC_REG must be equal to UDP_SSEN_REG. Finally, main menu is displayed on the console. There are three test operations in the main menu which are described as follows.

3.1 Display parameters

This menu is designed to display the current value of all parameters. All common parameters and the session parameters of the active session are displayed on the console. While the inactive session shows only DISABLE status.

The step to display parameters is as follows.

1) Read all network parameters from each variable in firmware.
2) Display all common parameters, i.e., IP Mode, FPGA MAC address, and FPGA IP address.
3) Check the session active flag of each session. If the session is active, the session parameters are displayed, i.e., Target IP address, Multicast IP address (when running in Multicast mode), Target port number, and FPGA port number.

3.2 Set parameters

This menu is designed to change some input parameters of UDP10GRx-IP. To set common parameters, all sessions must be disabled and then re-enabled. To set session parameters, only the modified session must be disabled and then re-enabled. After finishing setting parameters, UDP10GRx-IP begins the initialization process. UDP_SSAC_REG is monitored until the initialization is completed.

The step to set parameter to the UDP10GRx-IP is as follows.

1) Display all common parameters on the console.
2) Skip to the next step if the user confirms to use default value. Otherwise, the menu for setting common parameters is displayed. CPU receives the value of each common parameter from the user and validate it. If the input is invalid, the parameter does not change the value. When updating common parameter is selected, UDP_SSEN_REG is set to 0 to disable all sessions.
3) The menu for setting session parameters is displayed. There are four sessions in the demo which the user can set the parameter independently. Similar to common parameters, CPU receives the parameter from the console and validate it. If the input is invalid, the parameter does not change the value. Only the session which updates the parameters must be disabled by setting UDP_SSEN_REG to ‘0’.
4) CPU waits until the session is disabled by comparing UDP_SSAC_REG = UDP_SSEN_REG.
5) CPU sets all parameters to UDP10GRx-IP, similar to step 4) of the initialization process.
6) CPU asserts SSEnable of the session that needs to change parameter to ‘1’ to begin the initialization process.
7) CPU waits until the session finishes the initialization process, UDP_SSAC_REG = UDP_SSEN_REG.

3.3 Receive data test

This menu is designed to set the parameters for data verification module. Some sessions must be enabled before running this menu. User sets total number of received data for checking after finishing the operation. Also, the user sets data verification enable flag which are shared parameters for all sessions. When enabling data verification, the user can set the start value of 32-bit incremental pattern for each active session independently. The operation is cancelled when some inputs are invalid.

The step to run receive data test is as follows.

1) CPU confirms that at least one session is enabled. The operation is cancelled if no session is enabled.
2) CPU receives total received data size and data verification enable flag from the user. The operation is cancelled if some inputs are invalid.
3) Skip this step if data verification is disabled. Otherwise, the menu to set start pattern for all active sessions are displayed. The input value is set to USER_PAT0-3_REG. The operation is cancelled if the input is invalid.
4) Start UserDataVer operation by setting USER_CMD_REG=1 (disable data verification) or 3 (enable data verification).
5) Display recommended parameters for running test application on PC, processed from the current parameters.
6) Wait until IP receives the first data on some active sessions, current received size (USER_RXLENL0-3_REG) not equal to ‘0’. After that, CPU starts timer.
7) Wait until the receive operation of all operating session is completed. Each session can complete by two ways. First, the received data size (USER_RXLENL0-3_REG and USER_RXLENH0-3_REG) does not change more than 100 msec. Second, total data of the session is received. During receiving data, CPU displays current number of received data on the console every second by reading USER_RXLENL0-3_REG and USER_RXLENH0-3_REG.
8) Stop timer. Check error interrupt (USER_STS_REG[1], [5], [9], and [13]) and data verification flag (USER_STS_REG[0], [4], [8] and [12]) when data verification is enabled. If some errors are asserted to ‘1’, the error message is displayed.
9) Calculate performance and display the result with the latency time in UDP10GRx-IP which is read from USER_RXTIM_REG.

3.4 Function list in User application

This topic describes the function list to run UDP10GRx-IP operation.

void init_param(void)	
Parameters	None
Return value	None
Description	Run for setting parameter, as described in topic 3.2. During running, show_cm_param, input_cm_param, show_ss_param, input_ss_param, and show_param are called.

int input_cm_param(void)	
Parameters	None
Return value	0: Valid input, -1: Invalid input
Description	Receive common parameters from user, i.e., IP mode, FPGA MAC address, and FPGA IP address. When the input is valid, the parameters are updated. Otherwise, the value does not change.

int input_ss_param(int ss_number)	
Parameters	ss_number: session number to set parameter, valid from 0 to 3.
Return value	0: Valid input, -1: Invalid input
Description	Receive session parameters from user, i.e., SSEnable, Target IP, Multicast IP (for Multicast mode), Target port number, and FPGA port number. When the input is valid, the parameters are updated. Otherwise, the value does not change. The session for setting parameter is defined by ss_number.

void show_cursize(void)	
Parameters	None
Return value	None
Description	Read USER_RXLENL0-3_REG and USER_RXLENH0-3_REG and then display the current number of received data in Byte, KByte, MByte, or GByte unit of all sessions.

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

void show_cm_param(void)	
Parameters	None
Return value	None
Description	Display the current value of the common parameters that is set to UDP10GRx-IP, i.e., IP Mode, FPGA IP address, and FPGA MAC address.

void show_param(void)	
Parameters	None
Return value	None
Description	Display the current value of all parameters that is set to UDP10GRx-IP by calling show_cm_param to show common parameters and show_ss_param to show session parameters.

void show_result(void)	
Parameters	None
Return value	None
Description	Read USER_RXLENL0-3_REG and USER_RXLENH0-3_REG to display total number of received data. Read timer parameters (timer_val and timer_upper_val) and calculate total time usage to display in usec, msec, or sec unit. After that, transfer performance is calculated and displayed on MB/s unit. Finally, USER_RXTIM_REG is read to show latency time of UDP10GRx-IP.

void show_ss_param(int ss_number, int show_mode)	
Parameters	ss_number: The session number for displaying, valid from 0 to 3. show_mode: 0 – Display all parameters when ss_number is active status 1 – Display all parameters without checking active status
Return value	None
Description	Display the current value of the session parameters that is set to UDP10GRx-IP such as Target IP address and Target port number when the session, defined by ss_number, is active or show_mode is equal to 1 (SHOW_ALL).

void show_verfail(void)	
Parameters	None
Return value	None
Description	Read USER_STS_REG[0], [4], [8], and [12] to check verification fail flag. If verification is failed, the failure information is displayed by reading registers, i.e., USER_RDFAILL0-3_REG/USER_RDFAILH0-3_REG (failure position), USER_EXPPAT0-3_REG (expected data), and USER_RDPAT0-3_REG (read data).

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 3.3

void wait_ethlink(void)	
Parameters	None
Return value	None
Description	Read EMAC_STS_REG[0] and wait until linkup status is found

4 Test Software on PC

"trans_udp_multi" is an application on PC for sending or receiving UDP data. The application supports to transfer up to 4 sessions at the same time. The parameters are divided into two groups, i.e., common parameters which are shared for all sessions and session parameters which are independent for each session. Therefore, the application is designed to receive the inputs in two steps following the parameter type – common parameters and session parameters.

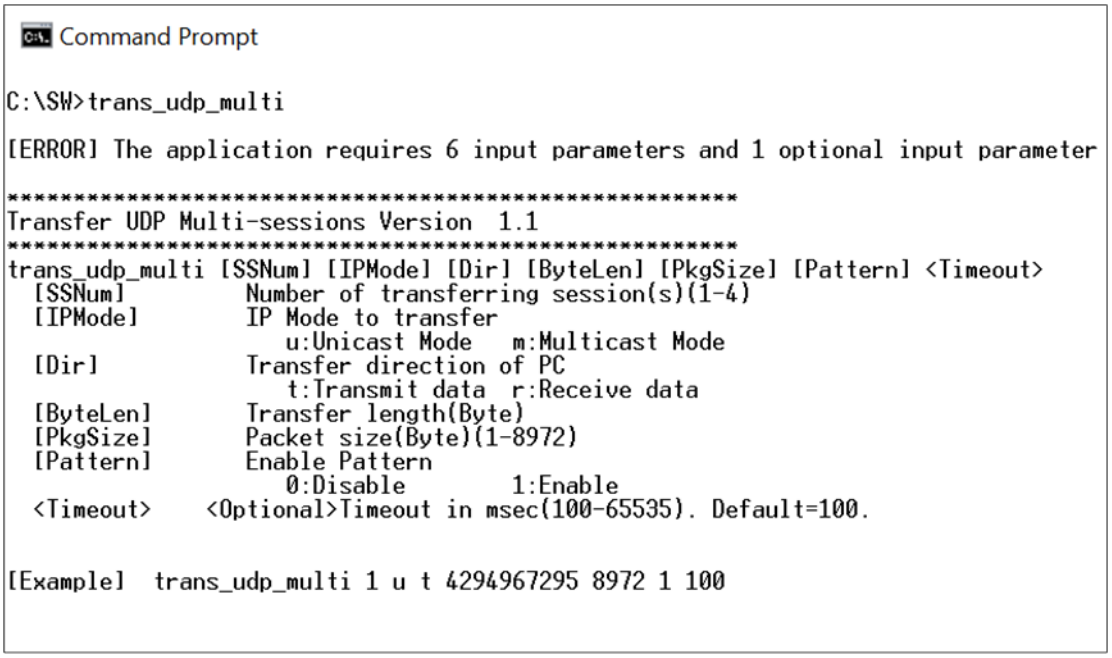

Figure 4-1 trans_udp_multi application to input common parameters

First, the user sets common parameters which consists of six parameters and one optional parameter, as shown in Figure 4-1. If some parameters are not valid, the application is cancelled with displaying error message. More details of the first parameter group are as follows.

1) SSNum : Number of transferring sessions, valid from 1 to 4
2) IPMode : u – Unicast Mode
   m – Multicast Mode
3) Dir : t – PC sends data to FPGA
4) ByteLen : Total transfer length in byte unit.
5) PkgSize : Packet size for sending, valid from 1-8972.
6) Pattern : 0 – Generate dummy data in transmit mode
   1 – Generate 32-bit incremental data in transmit mode
7) Timeout (optional): Timeout during transferring data in milli seconds unit.
   Default value when user does not input this parameter is 100.

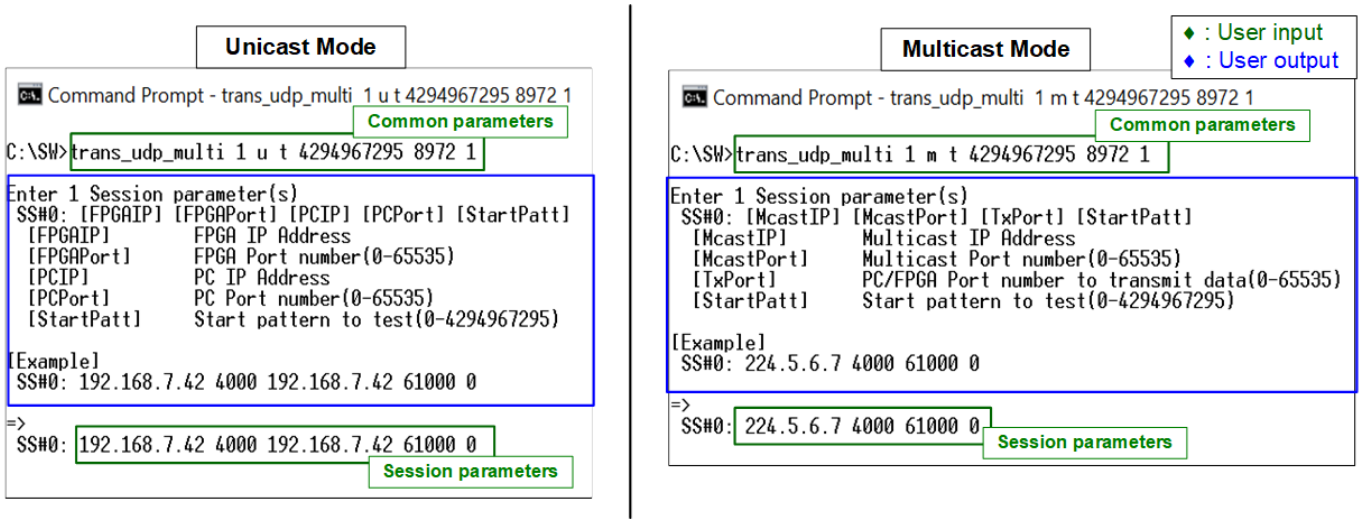

Figure 4-2 trans_udp_multi application to input session parameters

Next, the console for receiving the session parameters is displayed after all common parameters from user are valid. The message for receiving session parameters when running in Unicast mode is different from Multicast mode. Figure 4-2 shows the message when sending data for one session in Unicast mode and Multicast mode. The details of each parameter are described as follows.

In Unicast mode,

1) FPGAIP : IP address setting on FPGA
2) FPGAPort : Port number of FPGA
3) PCIP : IP address setting on PC
4) PCPort : PC port number for sending data
5) StartPatt : Start value of 32-bit incremental data for sending in the test

In Multicast mode,

1) McastIP : Multicast IP address
2) McastPort : Multicast port number
3) TxPort : PC port number for transmitting data
4) StartPatt : Start value of 32-bit incremental data for sending in the test

When many sessions are selected, the message for receiving the next session parameters is displayed after finishing the first session setting. Data begins transmission after finishing to set all session parameters.

The details when the application is run for transmitting data are described as follows.

1) Get common parameters from user and verify that the input is valid. The operation is cancelled when some parameters are invalid.
2) Get session parameters from user and verify that the input is valid. When many sessions are run, this step is repeated to get the next session parameters until completing all sessions. The first session is session#0. The operation is cancelled when some parameters are invalid.
3) Create the socket and then set properties of transmit buffer.
4) Set IP address and port number from the user parameters to the socket.
5) This step is run following IP mode.
   a) In unicast mode, the application sends ARP request packet.
   b) In multicast mode, the application sends IGMP Membership Report to Join Group.
6) When many sessions are run, repeat Step (3)-(5) to create socket for each session.
7) Allocate memory to be transmit buffer in the application.
8) Create child thread for sending data of each session. Therefore, the number of child threads is equal to the number of sessions. After that, child thread sends the data with or without filling test data to the buffer, depending on [Pattern] parameters from the user. If Pattern=1, the send buffer is filled by 32-bit incremental pattern, starting by [StartPatt] input. Otherwise, the send buffer is not filled.

   At the same time, the main thread checks transfer status of every child thread until all threads are completely operated. During checking status, the main thread displays current transfer data size of each session on the console every second.
9) After finishing sending the data in every session, the application calculates and displays performance with total transfer size as a test result.
10) Close the socket, kill the child thread, and free memory before finishing the application.

5 Revision History

Revision	Date	Description
1.0	22-Apr-21	Initial version release