UDP10GRx IP reference design

Rev1.2 26-Apr-22

1 Introduction

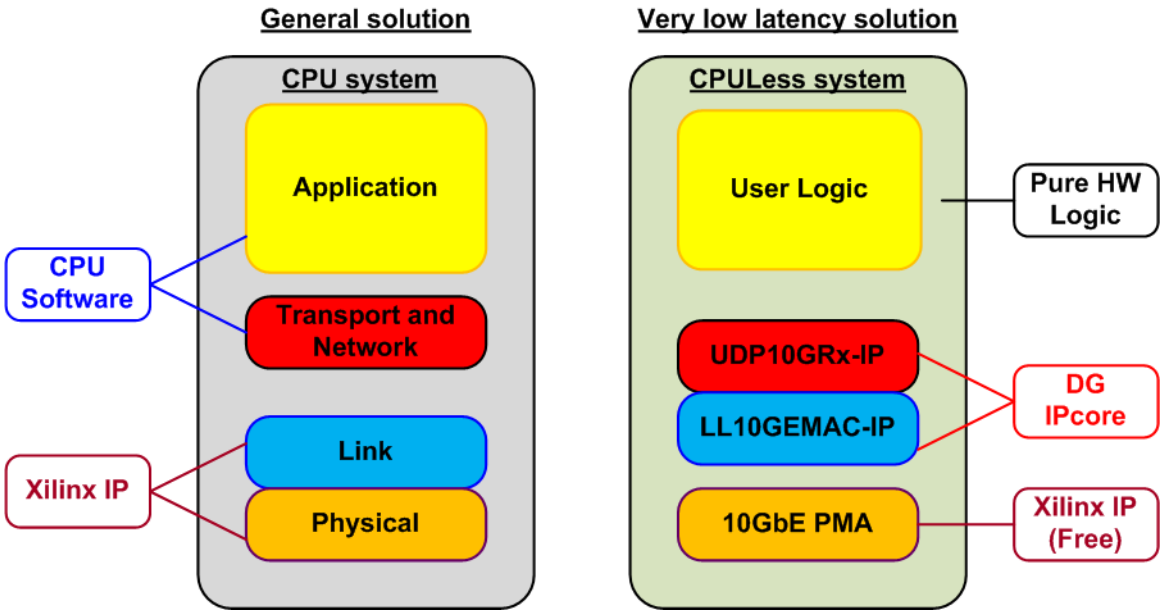

Figure 1-1 Low latency solution

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 using 10/25G Ethernet Subsystem which is Xilinx IP core. 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 purposed, 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 low-layer of hardware (PMA layer) is provided by Xilinx as a free IP core.

The UDP payload 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.

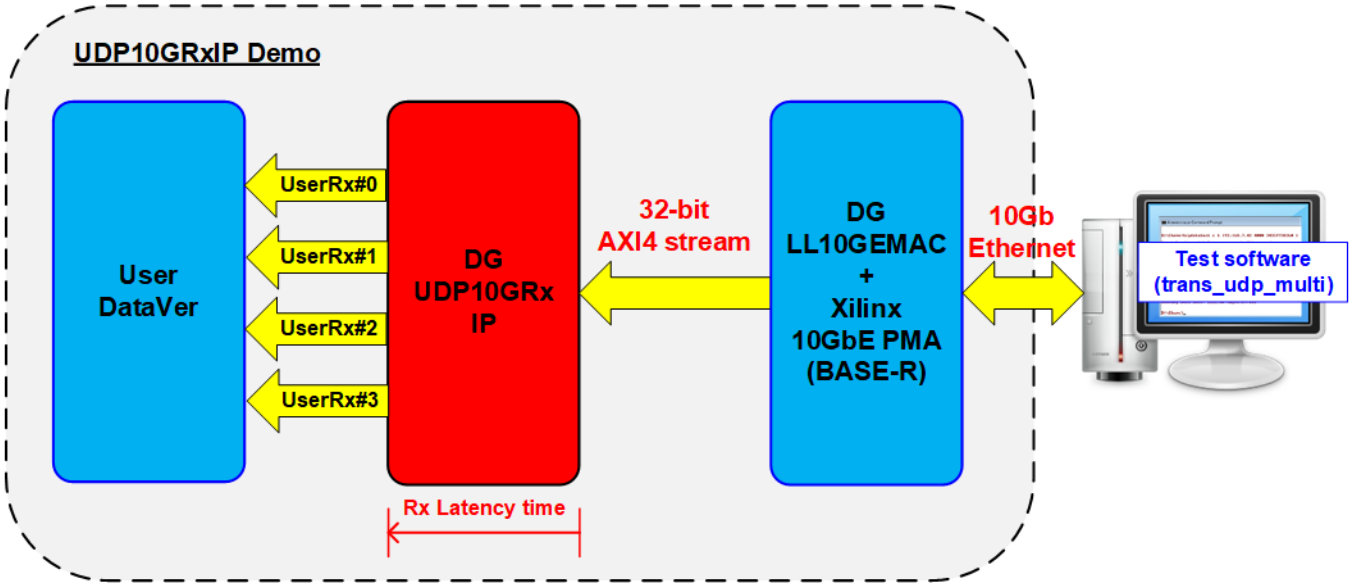

Figure 1-2 Test logic for UDP10GRx-IP

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. 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) which runs on the PC generates UDP/IP packet of four sessions via 10Gb Ethernet. DG LL10GEMAC-IP and Xilinx 10GbE PMA (BASE-R) implement the low-level interface module for transferring Ethernet packet with UDP10GRx-IP. The latency time of received data interface from UDP10GRx-IP is measured by using the timer.

CPU system is included for user interface via Serial console to setting the test parameters. Also, the test result and the progress of test operation are returned to CPU for displaying on the console. More details of the demo are described as follows.

2 Hardware overview

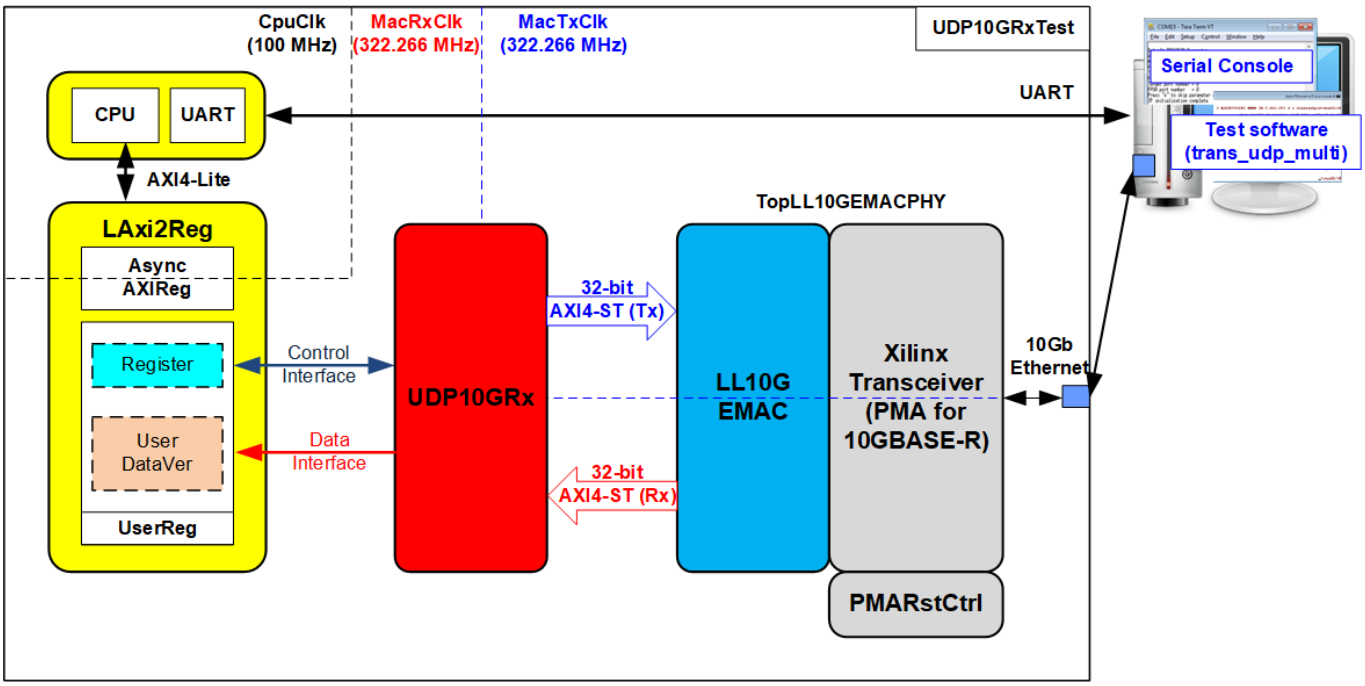

Figure 2-1 UDP10GRxTest Block Diagram

The test system includes CPU for easy user interface and flexible test. Test parameters such as network parameters and transfer size are set by user. Also, the current status of the test system such as current transfer size can be monitored on Serial console. To connect the hardware with CPU system, AXI4-Lite bus standard must be implemented. LAxi2Reg is the interface module to convert AXI4-Lite interface to be the user interface of UDP10GRx module. Also, AsyncAXIReg is included to be asynchronous module between CPU system clock (CpuClk) and UDP10GRx interface clock (MacRxClk).

The user interface of UDP10GRx is divided to two interfaces - control interface and data interface. In this reference design, the control interface is controlled by CPU system via Register Files while the data interface is connected to UserDataVer module for verifying the received data from UDP10GRx module. Another side of UDP10GRx-IP is connected to 10G Ethernet MAC controller (LL10GEMAC-IP) by using 32-bit AXI4 stream interface (AXI4-ST) to achieve the lowest latency time. Tx interface and Rx interface of LL10GEMAC-IP are run in different clock domain - MacTxClk and MacRxClk. The lowest layer module is PMA module which implements by Xilinx transceiver to support 10GBASE-R interface. PMARstCtrl is the module to control reset sequence of Xilinx Transceiver.

The target to communicate with the test system is TestPC which runs Test application (trans_udp_multi.exe, provided by Design Gateway) to send UDP payload data up to many sessions at the same time. Test application can be set to transfer by Multicast mode or Unicast mode.

2.1 Xilinx Transceiver (PMA for 10GBASE-R)

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

- Transceiver configuration preset : GT-10GBASE-R
- Encoding/Decoding : Raw
- Transmitter Buffer : Bypass
- Receiver Buffer : Bypass
- User/Internal data width : 32

The example of Transceiver wizard in Ultrascale model is described in the following link.
https://www.xilinx.com/products/intellectual-property/ultrascale_transceivers_wizard.html

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 AXI4-stream bus. Please see more details from LL10GEMAC-IP datasheet on our website.
https://dgway.com/products/IP/Lowlatency-IP/dg_ll10gemacip_data_sheet_xilinx_en.pdf

2.3 PMARstCtrl

When the buffer inside Xilinx Transceiver is bypassed, the user logic must control reset signal of Tx buffer and Rx buffer. The module is designed by state machine to run following step.

(1) Assert Tx reset of the transceiver to ‘1’ for one clock cycle.
(2) Wail until Tx reset done, output from the transceiver, is asserted to ‘1’.
(3) Finish Tx reset sequence and de-assert Tx reset to allow the user logic beginning Tx operation.
(4) Assert Rx reset to the transceiver.
(5) Wait until Rx reset done is asserted to ‘1’.
(6) Finish Rx reset sequence and de-assert Rx reset to allow the user logic beginning Rx operation.

2.4 UDP10GRx

UDP10GRx-IP is the IP core provided by Design Gateway to implement the UDP/IP stack and offload engine for the low latency solution. It extracts UDP payload data from Ethernet packet that is transmitted by EMAC and then forwards to user. Up to four sessions are supported to receive several data types at the same time. More details of UDP10GRx-IP are described in UDP10GRx-IP datasheet, provided on our website.
https://dgway.com/products/IP/Lowlatency-IP/dg_udp10grxip_data_sheet_xilinx_en.pdf

2.5 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 CPU system is built with one additional peripheral to access the test logic. Therefore, the hardware logic must be designed to support AXI4-Lite bus standard for CPU write access and read access. LAxi2Reg module is designed to connect the CPU system as shown in Figure 2-2.

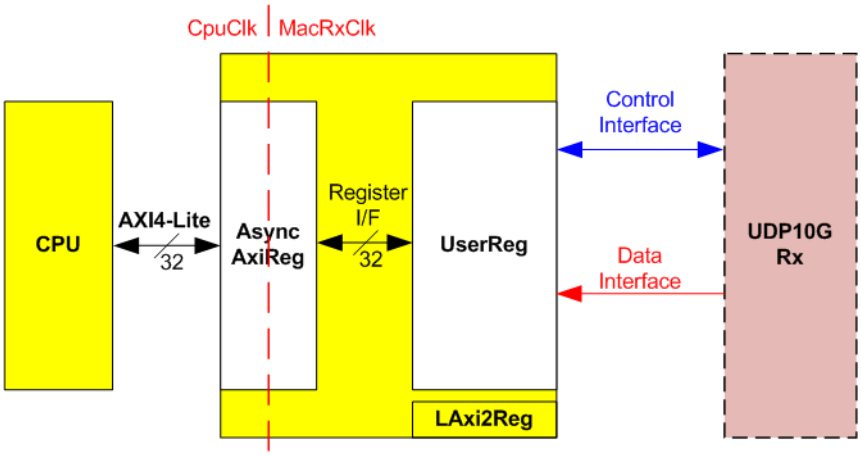

Figure 2-2 LAxi2Reg block diagram

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 MacRxClk domain.

UserReg includes the register file of the parameters and the status signals of test logics. Both data interface and control interface of UDP10GRx-IP are also connected to UserReg. More details of AsyncAxiReg and UserReg are described as follows.

2.5.1 AsyncAxiReg

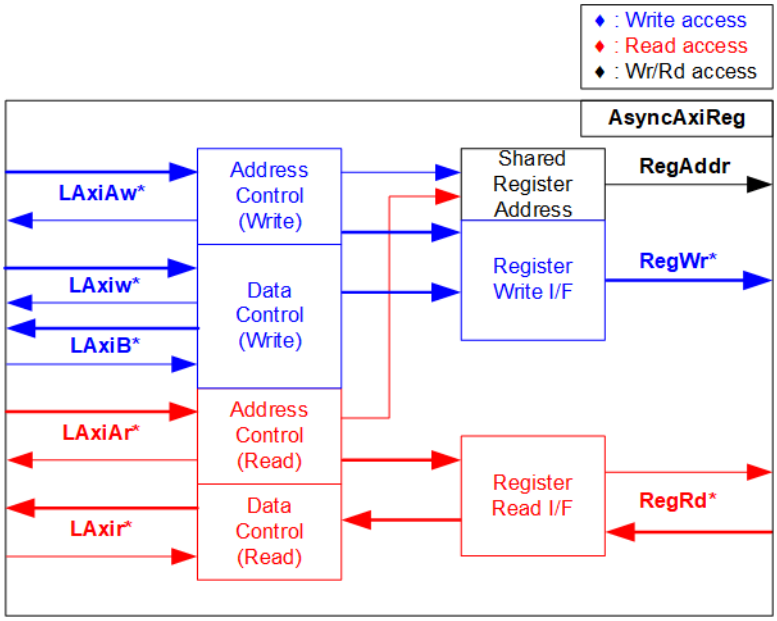

Figure 2-3 AsyncAxiReg Interface

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. 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.

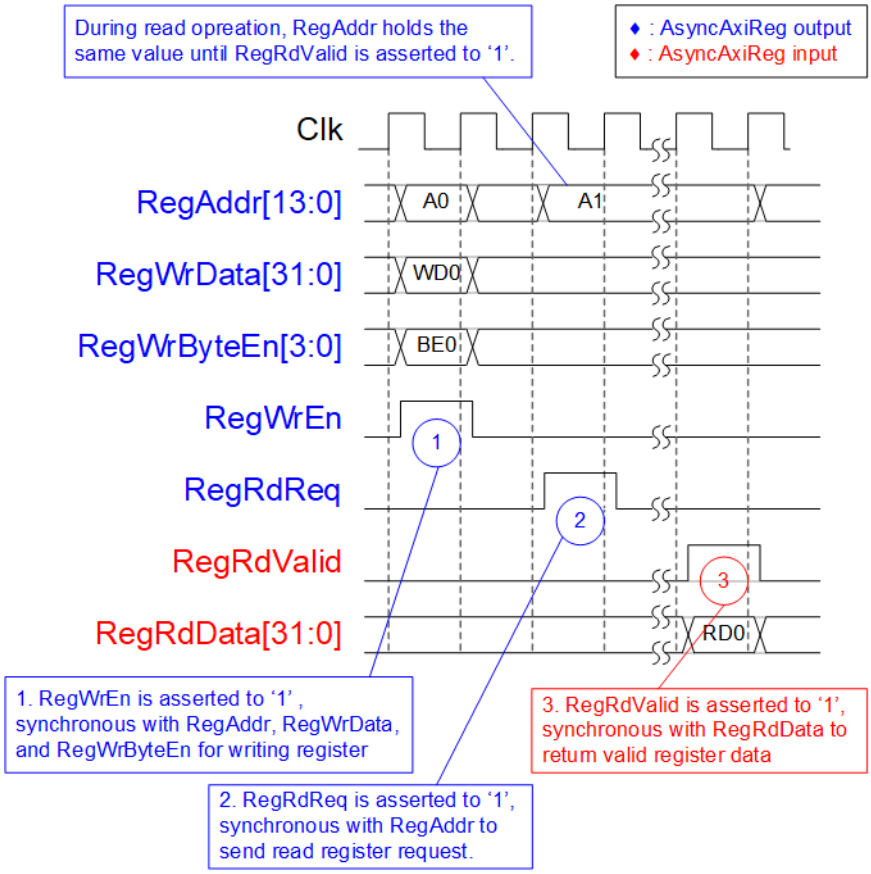

Figure 2-4 Register interface timing diagram

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 enable 4-byte data. 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 is returned after receiving the read request. The slave detects RegRdReq asserted to start the read transaction. During read operation, the address value (RegAddr) does not change until RegRdValid is asserted to '1'. Therefore, the address can be used for selecting the returned data by using multiple levels 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.

2.5.2 UserReg

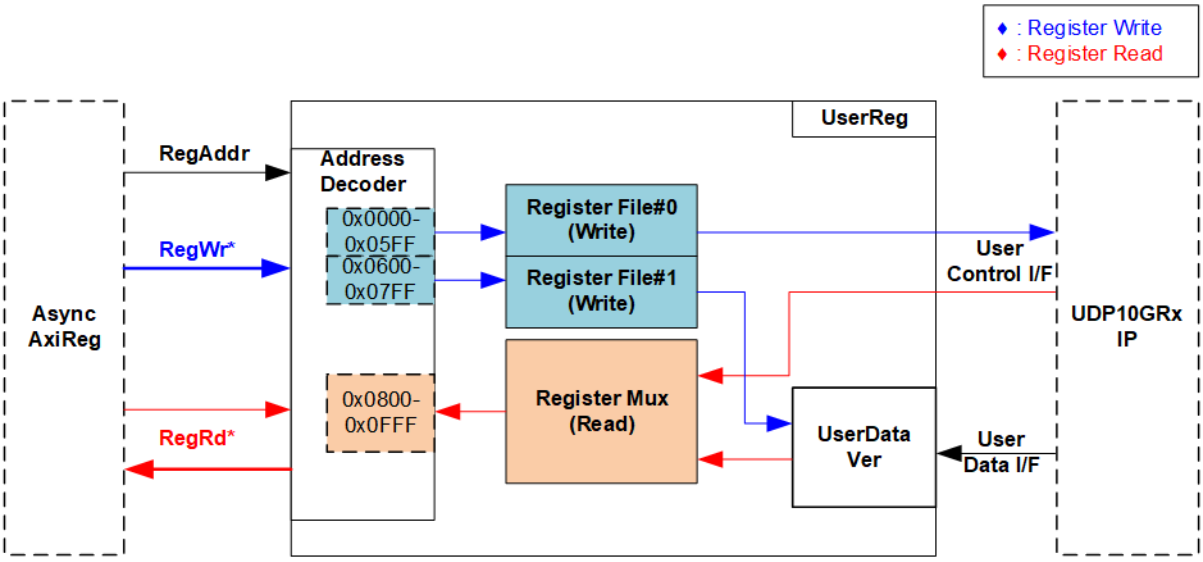

Figure 2-5 UserReg block diagram

The logic inside UserReg consists of two operations, i.e., Register interface (Address decoder, RegFile, and RegMux) and Data pattern verification (UserDataVer).

Register block decodes the address which is requested from AsyncAxiReg and then selects the active register for write or read transaction. UserDataVer block reads and verifies 32-bit data from UDP10GRx-IP following AXI4-Stream interface standard. More details of Register block and UserDataVer are described as follows.

Register Block

The address range assigned in UserReg is split into three areas, described as follows.

- 0x0000 – 0x05FF: UDP10GRx-IP (Write access)
- 0x0600 – 0x07FF: UserDataVer (Write access)
- 0x0800 – 0x0FFF: UDP10GRx-IP and UserDataVer (Read access)

Address decoder decodes the upper bit of RegAddr for selecting the active hardware that is UDP10GRx-IP or UserDataVer. 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 register.

To read register, multiplexer is designed to select the read data to return to CPU by using the address. The lower bit of RegAddr is fed to the submodule to select the active data. While the upper bit is applied to select the returned data from the submodule. Totally, the latency of read data is equal to two clock cycles. Therefore, RegRdValid is created by RegRdReq with asserting two 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+0x05FF: UDP10GRx-IP (Write access only) More details of UDP10GRx-IP signals are described in UDP10GRx-IP datasheet		
BA+0x0000 – BA+0x01FF: Common parameters and control of UDP10GRx-IP		
BA+0x0000	Session Enable Reg UDP_SSEN_REG	[3:0]: Input to be Session enable (SSEnable[3:0] or UDP10GRx-IP)
BA+0x0004	Multicast Enable Reg UDP_MCEN_REG	[0]: Input to be Multicast enable (McastEn of UDP10GRx-IP)
BA+0x0040	Source MAC Address (Low) Reg UDP_SML_REG	[31:0]: Input to be Source MAC Address (SrcMacAddr[31:0] of UDP10GRx-IP)
BA+0x0044	Source MAC Address (High) Reg UDP_SMH_REG	[15:0]: Input to be Source MAC Address (SrcMacAddr[47:32] of UDP10GRx-IP)
BA+0x0048	Source IP Address Reg UDP_SIP_REG	[31:0]: Input to be Source IP Address (SrcIPAddr[31:0] of UDP10GRx-IP)
BA+0x0200 – BA+0x02FF: Session parameters of UDP10GRx-IP		
BA+0x0200	Source Port Number Reg UDP_SPN0_REG	[15:0]: Input to be source port number#0 (SrcPort0 of UDP10GRx-IP)
BA+0x0204	Target Port Num Reg UDP_DPN0_REG	[15:0]: Input to be target port number#0 (DstPort0 of UDP10GRx-IP)
BA+0x0208	Target IP Address Reg UDP_DIP0_REG	[31:0]: Input to be target IP address#0 (DstIPAddr0 of UDP10GRx-IP)
BA+0x020C	Multicast IP Address Reg UDP_MIP0_REG	[31:0]: Input to be multicast IP address#0 (McastIPAddr0 of UDP10GRx-IP)
BA+0x0220- BA+0x022F	Session#1 parameters (UDP_SPN1_REG – UDP_MIP1_REG)	0x0220: SrcPort1 of UDP10GRx-IP 0x0224: DstPort1 of UDP10GRx-IP 0x0228: DstIPAddr1 of UDP10GRx-IP 0x022C: McastIPAddr1 of UDP10GRx-IP
BA+0x0240- BA+0x024F	Session#2 parameters (UDP_SPN2_REG – UDP_MIP2_REG)	0x0240: SrcPort2 of UDP10GRx-IP 0x0244: DstPort2 of UDP10GRx-IP 0x0248: DstIPAddr2 of UDP10GRx-IP 0x024C: McastIPAddr2 of UDP10GRx-IP
BA+0x0260- BA+0x026F	Session#3 parameters (UDP_SPN3_REG – UDP_MIP3_REG)	0x0260: SrcPort3 of UDP10GRx-IP 0x0264: DstPort3 of UDP10GRx-IP 0x0268: DstIPAddr3 of UDP10GRx-IP 0x026C: McastIPAddr3 of UDP10GRx-IP
BA+0x0600 – BA+0x07FF: Input signals for UserReg (Write access only)		
BA+0x0600	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+0x0604	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+0x0680- BA+0x068F	User Start Pattern0-3 Reg (USER_PAT0-3_REG)	[31:0]: Start value of 32-bit incremental pattern for verifying data in Session#0-#3 of UserDataVer. 0x680: SS#0, 0x684: SS#1, 0x688: SS#2, and 0x68C: SS#3.

Address Wr/Rd	Register Name (Label in the udp10grxtest.c")	Description
BA+0x0800 – BA+0x0FFF Output signals of UDP10GRx-IP outputs, UserReg, and UserDataVer (Read access only)		
BA+0x0800 – BA+0x09FF: Output signals of UDP10GRx-IP and UserReg (Read access only)		
BA+0x0800	EMAC Status Reg EMAC_STS_REG	[0]: Ethernet MAC Link up status. '0'-Link down, '1'-Link up. (Linkup of LL10GEMAC-IP)
BA+0x0804	EMAC IP Version EMAC_VER_REG	[31:0]: IPVersion of LL10GEMAC-IP
BA+0x0808	UDP10GRx-IP Version UDP_VER_REG	[31:0]: IPVersion of UDP10GRx-IP
BA+0x080C	Session Active of UDP10GRx-IP (UDP_SSAC_REG)	[3:0] Session active status. ('0': Not active, '1': Active). (SSActive[3:0] of UDP10GRx-IP)
BA+0x0820	Test pin of UDP10GRx-IP (UDP_TESTPIN0_REG)	[3:0]: Mapped to TestPin of UDP10GRx-IP
BA+0x0840	User Error SS#0-#3 Reg (USER_ERR0_3_REG)	[31:0] Error status of Session#0 - #3. [0]: Data verification of SS#0 is failed. ('0'-No error, '1'-Fail) [1]: UDP10GRx-IP detects error in SS#0 ('0'-No error, '1'-Error found) [5:4]: Similar to SS#0 error (assigned to bit[1:0]), SS#1 error [9:8]: SS#2 error, [13:12]: SS#3 error
BA+0x0860	UDP10GRx Latency time Reg (UDPRX_TIME_REG)	[31:0]: Latency time of received data of UDP10GRx-IP, measured from start-of-packet to start-of-packet. The unit count is 3.1 ns (1/MacRxClk).
BA+0x0880	UDP Error Reg (UDP_ERR_REG)	[7:0]: Latch value of UDPRxError when Session#0 has error. [15:8]: Latch value of UDPRxError when Session#1 has error. [23:16]: Latch value of UDPRxError when Session#2 has error. [31:24]: Latch value of UDPRxError when Session#3 has error.
BA+0x0A00 – BA+0x0BFF: Output signals of UserDataVer (Read access only)		
BA+0x0C00	Current Received Size0 (Low) (USER_RXLENL0_REG)	[31:0]: The lower 32-bit of current amount of received data from Session#0 in byte unit.
BA+0x0C04	Current Received Size0 (High) (USER_RXLENH0_REG)	[15:0]: The upper 16-bit of current amount of received data from Session#0 in byte unit.
BA+0x0C08	Data Failure Position0 (Low) (USER_RDFAILL0_REG)	[31:0]: The lower 32-bit of the position of the 1st failure data from Session#0 in byte unit.
BA+0x0C0C	Data Failure Position0 (High) (USER_RDFAILH0_REG)	[15:0]: The upper 16-bit of the position of the 1st failure data from Session#0 in byte unit.
BA+0x0C10	Expect Data0 (USER_EXPPAT0_REG)	[31:0]: Expect value of the 1st failure data when Session#0 data is failed.
BA+0x0C14	Read Data0 (USER_RDPAT0_REG)	[31:0]: Read value of the 1st failure data when Session#0 data is failed.
BA+0x0C20- BA+0x0C37	USER_RXLENL1_REG – USER_RDPAT1_REG	Similar to BA+0x0C00 – BA+0C14, the registers are applied for Session#1 data.
BA+0x0C40- BA+0x0C57	USER_RXLENL2_REG – USER_RDPAT2_REG	Similar to BA+0x0C00 – BA+0C14, the registers are applied for Session#2 data.
BA+0x0C60- BA+0x0C77	USER_RXLENL3_REG – USER_RDPAT3_REG	Similar to BA+0x0C00 – BA+0C14, the registers are applied for Session#3 data.

User Data Verification

UserDataVer receives the data from UDP10GRx-IP which has four sessions. The data is verified when verification enable is asserted by user. Four-session data shares the same data bus and valid signal, separated by 2-bit session number. Each session has its own initial value, assigned by InitPatt, which is loaded when Start signal (PattStart) is asserted. The test operation begins when Start signal is asserted.

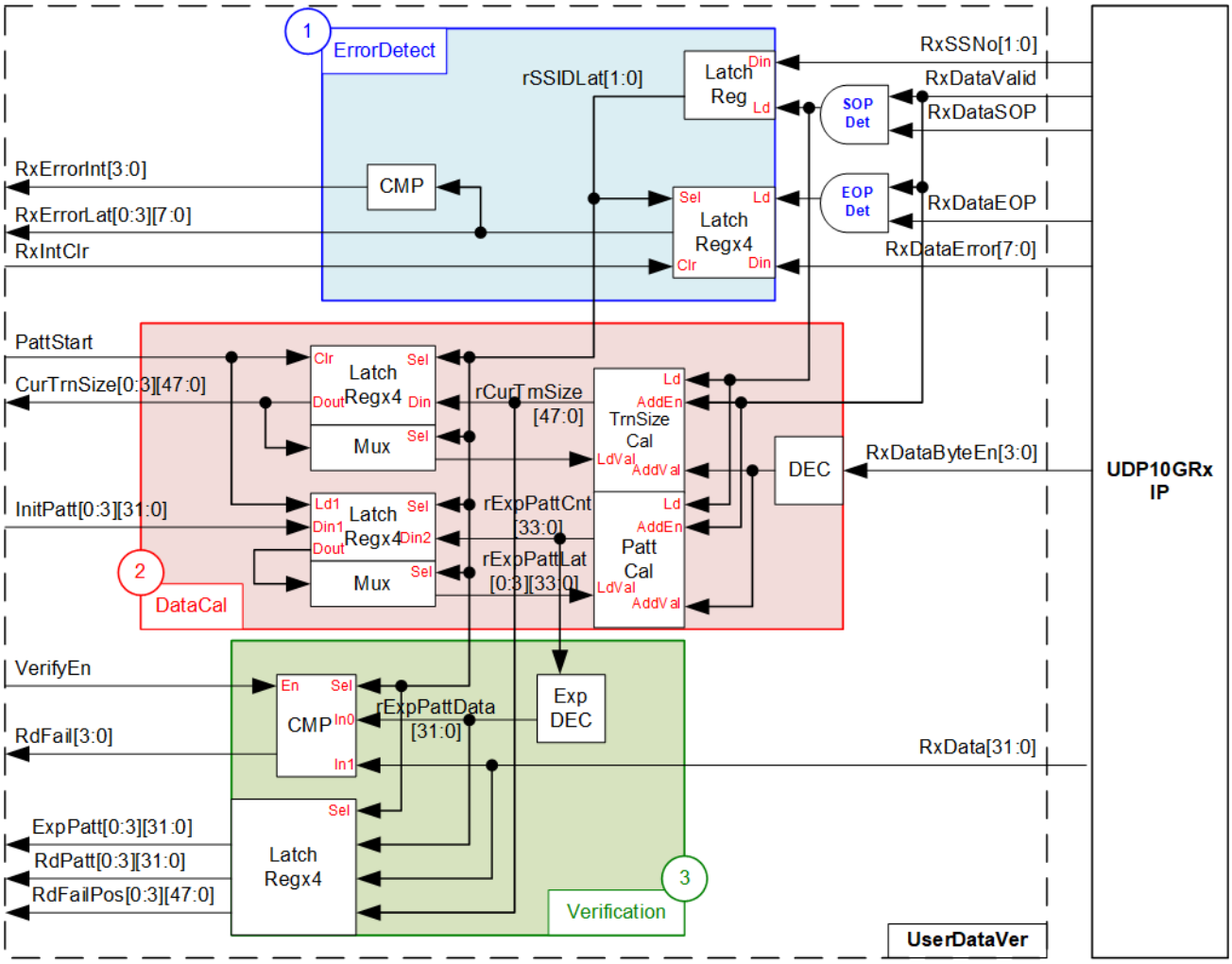

Figure 2-6 UserDataVer

As shown in Figure 2-6, UserDataVer has three operations, i.e., Error detection (1) to detect the error from UDP10GRx-IP, Data calculating (2) to calculate total amount of valid data, and Verification (3) to verify the received data.

ErrorDetect (1) reads the error output from UDP10GRx-IP (RxDataError) when the end of packet is received (RxDataEOP='1' and RxDataValid='1'). If the error is detected, the interrupt flag (RxErrorInt) is asserted to '1' and RxDataError is latched (RxErrorLat). The error interrupt and status have four sets for storing the status of each session. After the user asserts the signal to clear interrupt (RxIntClr), the error interrupt and status will be cleared. The session number from the IP (RxSSNo) is loaded the latch register for decoding the active session for each received packet. For error detection, the latched session number (rSSIDLat) is applied to load the error status to one of four latch register sets.

DataCal (2) includes the logic to calculate total amount of received data from UDP10GRx-IP (TrnSizeCal). Valid signal of received data (RxDataValid) and byte enable (RxDataByteEn) are monitored to check the number of valid bytes in each cycle that can be equal to 1 – 4 bytes. After that, rCurTrnSize which shows total amount of received data is fed to Latch register which has four sets for four sessions. Only one of four sets is loaded following the active session number. Besides, the output of latch register is returned to TrnSizeCal to be the start value when the new packet is received. TrnSizeCal loads the latest value from the active session for updating the amount of received data. Similarly, there is the logic to calculate expected data which is designed to be incremental pattern (PattCal). The operation of PattCal is the same as TrnSizeCal, but the start value of each session can be set by user via InitPatt signal.

Verification (3) has the decoder (ExpDEC) to calculate the real expected data following the byte offset. The byte offset is the offset value when the previous data is not aligned to 32-bit unit. Therefore, the next expected data will mix value between two 32-bit values. After that, the result is fed to compare with the received data (RxData). Fail flag (RdFail) is asserted if the received data is not equal to the expected data and user enables verification flag (VerifyEn). Also, four sets of latch registers are included to store the error information of four sessions, i.e., the expected value (ExpPatt), the received value (RdPatt), and the error position in byte unit (RdFailPos).

3 CPU Firmware (FPGA)

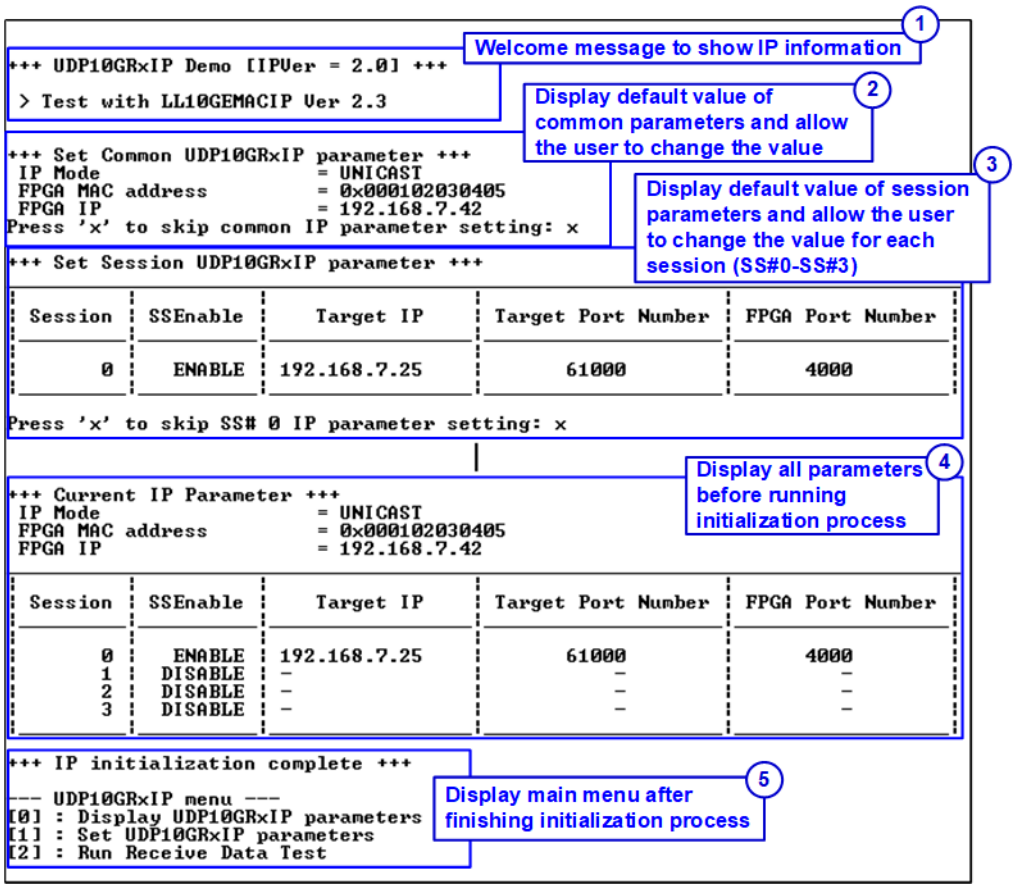

Figure 3-1 Message on the console in initialization process

In reference design, CPU firmware is implemented as bare-metal OS for simply 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 LL10GEMAC-IP information and displays on the console. After that, CPU waits until Ethernet link is established (EMAC_STS_REG[0]='1'). After that, continue to the next step.
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 set to each session individually. 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. After that, 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. Similar to common parameters, the user selects to set the new value for session parameters or skip the setting step by using default value. After that, the input is verified by CPU.
4) CPU sets all common parameters and session parameters to the hardware register as follows.
   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 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 common parameter value from the user and then validate it. If the input is invalid, the parameter does not change the value. When common parameter is updated, 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 can be set individually. 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 by comparing UDP_SSAC_REG = UDP_SSEN_REG.

3.3 Receive data test

This menu is designed to set the parameters for data verification module. At least one session must be active before running this menu. User sets total number of received data to be the expected value for comparing when the operation is finished. 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 amount of received data and data verification enable flag from the user. The operation is cancelled if some inputs are invalid.
3) Skip to step 4) if data verification is disabled. Otherwise, the menu to set start value of test data 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 the recommended parameters which read from the variable of running test application on PC.
6) Wait until at least one data of any active session is received by IP. The current received size of some sessions (USER_RXLENL0-3_REG) is not equal to 0. After that, CPU starts timer to measure operating time.
7) Wait until the receive operation of all operating session is completed. Each session is finished by two ways. First, the received data size (USER_RXLENL/H0-3_REG) does not change more than 100 msec. Second, total data of all active session are received. During receiving data, CPU displays current amount of received data on the console every second by reading USER_RXLENL/H0-3_REG.
8) Stop timer. Check error interrupt (USER_ERR0_3_REG[1], [5], [9], and [13]) and data verification flag (USER_ERR0_3_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 UDPRX_TIME_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.

void input_cm_param(void)	
Parameters	None
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.

void input_ss_param(int ss_number)	
Parameters	ss_number: session number to set parameter, valid from 0 to 3.
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_ERR0_3_REG and then decode the interrupt type to display 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 MAC address, and FPGA IP 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 amount 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 and calculated 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_ERR0_3_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 Ethernet link is established.

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 16 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 set to each session individually. Therefore, the application has two steps to receive two parameter groups.

```
Command Prompt

C:\SW>trans_udp_multi

[ERROR] The application requires 6 input parameters and 1 optional input parameter

*****************************************************
Transfer UDP Multi-sessions Version  1.2
*****************************************************
trans_udp_multi [SSNum] [IPMode] [Dir] [ByteLen] [PkSize] [Pattern] <Timeout>
  [SSNum]         Number of transferring session(s)(1-16)
  [IPMode]        IP Mode to transfer
                     u:Unicast Mode   m:Multicast Mode
  [Dir]           Transfer direction of PC
                     t:Transmit data  r:Receive data
  [ByteLen]       Transfer length(Byte)
  [PkSize]        Packet size(Byte)(1-8972)
  [Pattern]       Enable Pattern
                     0:Disable        1:Enable
  <Timeout>    <Optional>Timeout in msec(100-65535). Default=100.


[Example]  trans_udp_multi 1 u t 4294967295 8972 1 100


C:\SW>
```

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 and error message is displayed. More details of the first parameter group are as follows.

1) SSNum : Number of transferring sessions, valid from 1 to 16
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) PkSize : 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.

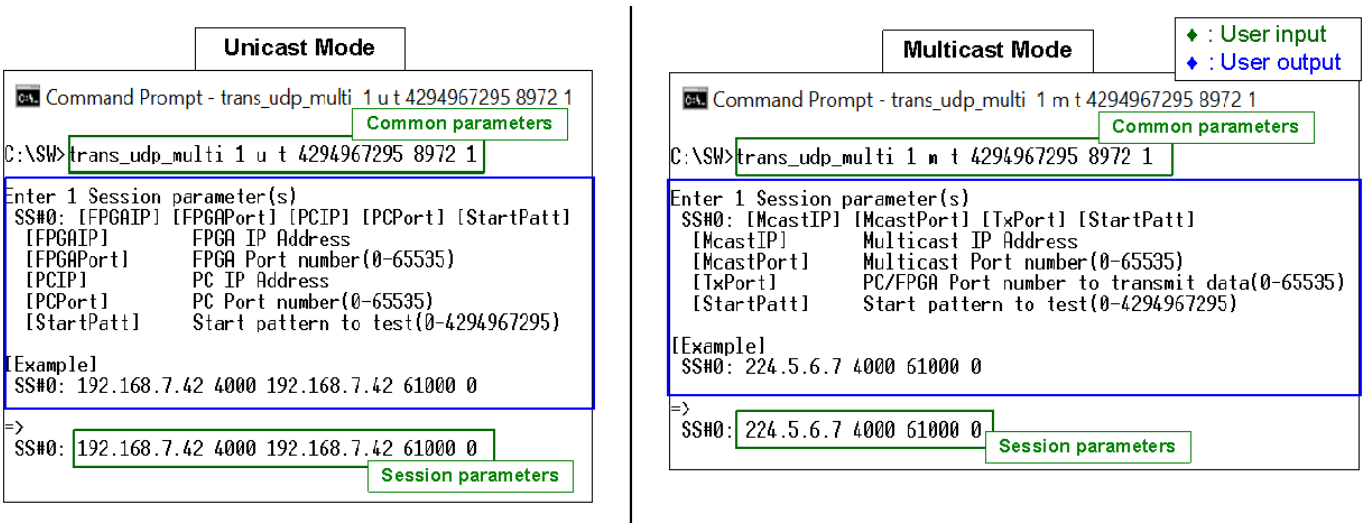

Figure 4-2 trans_udp_multi application to input session parameters

After all common parameters from user are valid, the console for receiving the session parameters is displayed. 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 on the left window and Multicast mode on the right window. 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 transmitting data
5) StartPatt : Start value of 32-bit incremental test data

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 test data

When many sessions are selected, the user needs to type the inputs for each session individually. Data begins transmission after finishing to set the parameters of all sessions.

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, step (3)-(5) are repeated to create socket for every 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. When the test is not completed, the main thread displays current amount of data in 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 to be a test result.
10) Close the socket, kill the child thread, and free memory before finishing the application.

5 Revision History

Revision	Date	Description
1.2	26-Apr-22	Update UserDataVer
1.1	30-Apr-21	- Update PMA to be raw data mode - Update trans_multi_udp parameters
1.0	23-Jun-20	Initial version release