UDP10GRx-IP 16-Session reference design

Rev1.0 23-Nov-21

1 Introduction

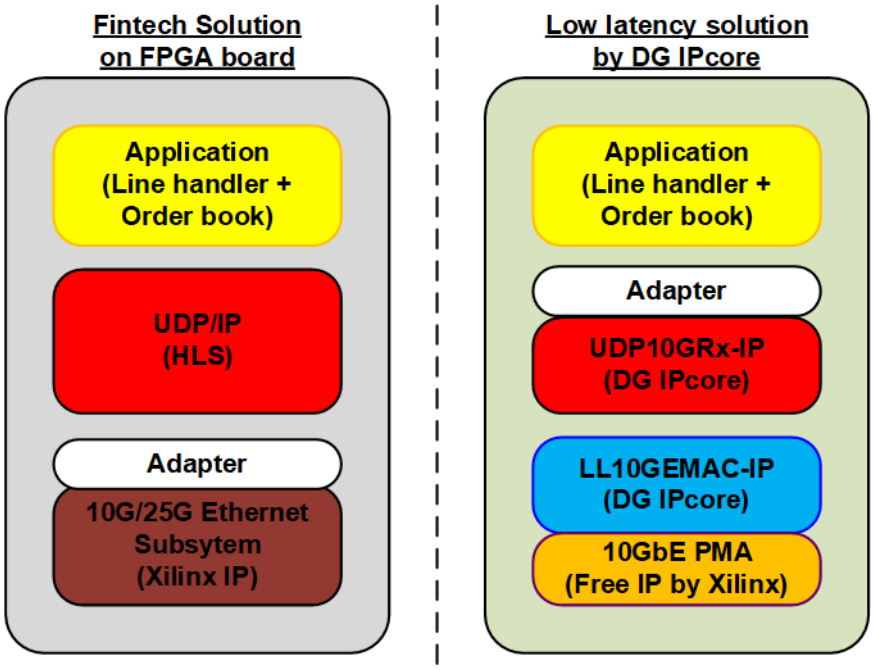

Figure 1-1 Low latency solution for Fintech application

The key point of Fintech application is to receive the market data at the lowest latency time. The left side of Figure 1-1 shows the example design of Fintech application on Xilinx board by using 10G/25G Ethernet Subsystem, Xilinx IP core, to implement EMAC, PCS, and PMA layer. To integrate the system coded by HLS, the adapter logic must be designed to convert Ethernet subsystem interface to AXI4 standard bus and run at application clock. The remaining logic of Fintech such as UDP/IP, Line handler, and Order book management are also coded by HLS. Using HLS reduces the system development time but the resource and latency time is generally larger than using HDL code.

Design Gateway purposes the very low latency solution for Fintech application by using UDP10GRx-IP and LL10GEMAC-IP, as shown in the right side of Figure 1-1. DG solution achieves the lowest latency time. The adapter logic is designed to integrate many UDP10GRx-IPs for supporting more than four session operation. Also, the adapter logic converts the interface to AXI4 standard bus at application clock for integrating with the remaining blocks that are coded by HLS.

For simple demo, the reference design is designed by HDL. The application block is replaced by UserLogic HDL module that is designed to verify the received UDP data stream. The adapter supports 16 sessions, so four UDP10GRx-IPs are integrated. Small adapter logic is also designed to connect four UDP10GRx-IPs to one LL10GEMAC-IP.

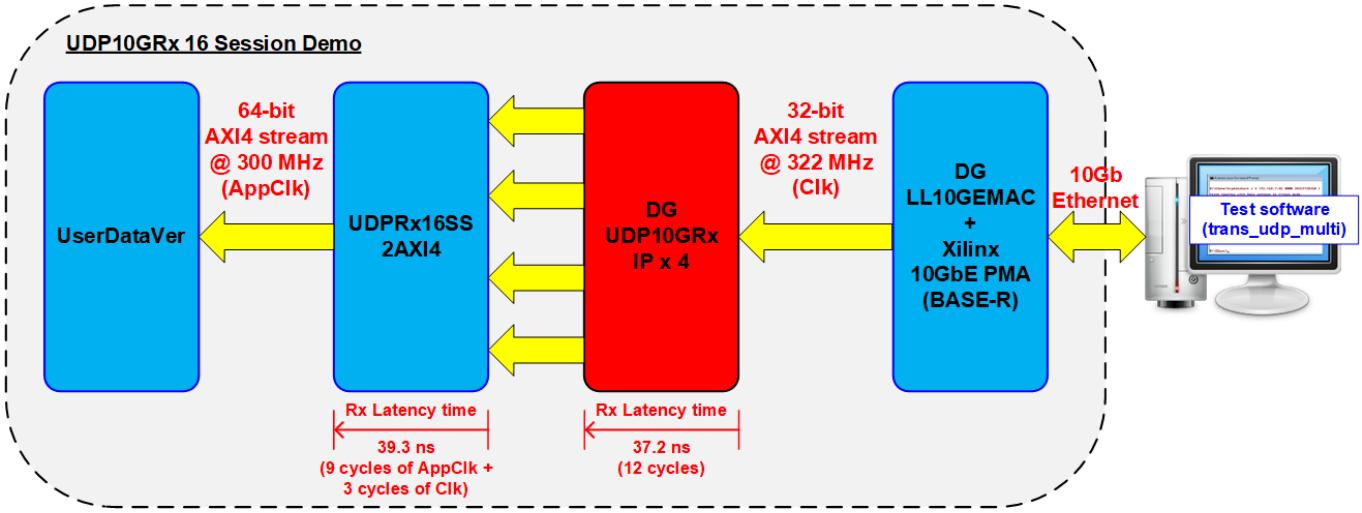

Figure 1-2 Test logic for running 16 sessions by 4 UDP10GRx-IPs

UserDataVer is the logic to receive 16-session data from the adapter (UDPRx16SS2AXI4) which is sent by four UDP10GRx-IPs. UserDataVer is coded by HDL to verify the data. Besides, two timers are integrated to measure latency time of received path inside UDP10GRx-IP and UDPRx16SS2AXI4. While Clk is fixed clock that is generated by PMA module at 322.265625 MHz, AppClk is the application clock to synchronous with the user application. In the reference design, AppClk is set to 300 MHz. Rx latency time of UDP10GRx-IP is about 37.2 ns (12 cycles of Clk) and Rx latency time of UDPRx16SS2AXI4 is about 39.3 ns (9 cycles of AppClk + 3 cycles of Clk).

Test application on Windows OS (trans_udp_multi) is designed to transmit UDP/IP packets up to 16 sessions, set by user. The test operation on FPGA board is controlled by user via Serial console. More details of the demo are described as follows.

2 Hardware overview

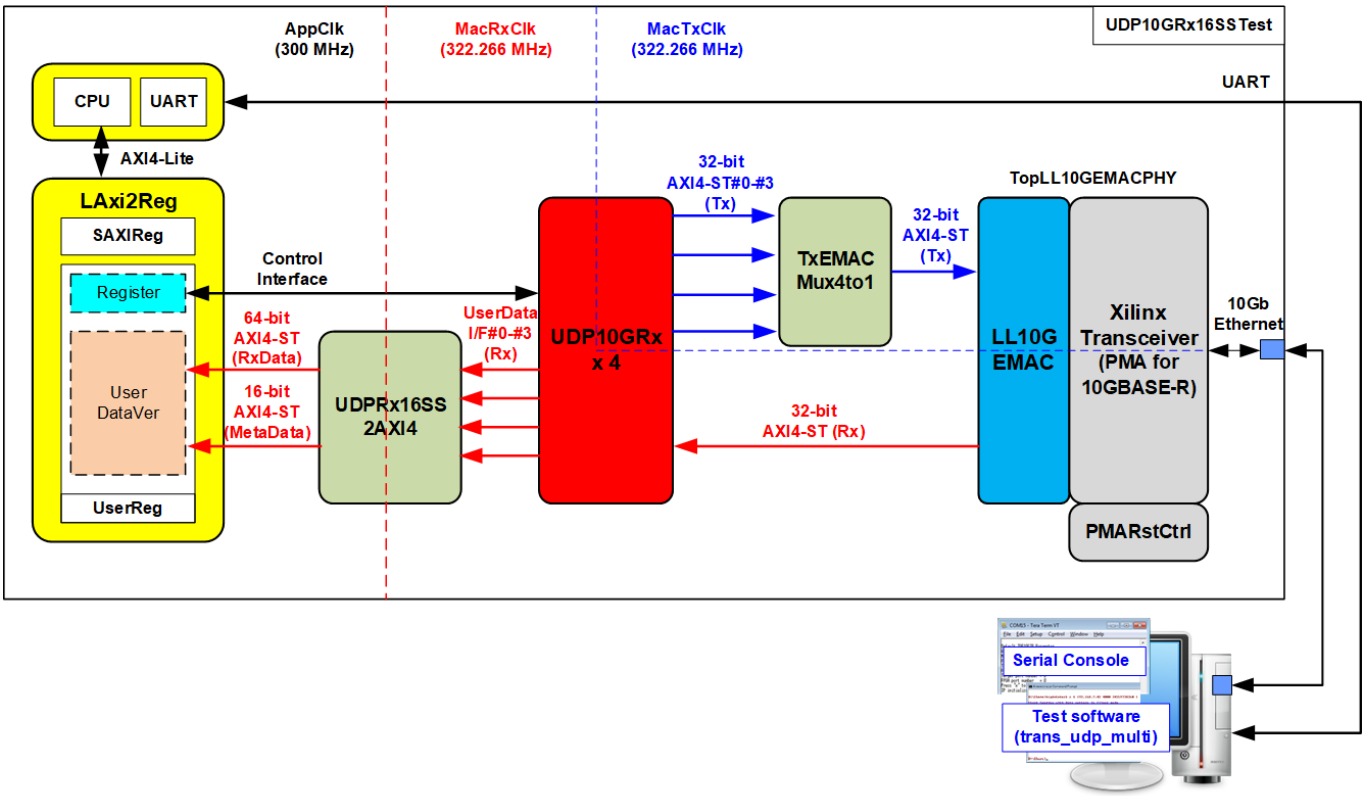

Figure 2-1 UDP10GRx16SSTest Block Diagram

The test system includes CPU for simple user interface and flexible test. Test parameters such as network parameters and transfer size are set by the user via Serial console. During running the test, the progress status of the test such as current transfer size is displayed on the console. The standard bus, AXI4-Lite bus, is applied to connect the hardware to CPU system. LAxi2Reg is the interface module to convert AXI4-Lite interface to be Control interface of UDP10GRx module. Also, it includes UserDataVer to verify the received data of 16 sessions via 64-bit AXI4-ST Rx data bus and 16-bit AXI4-ST Metadata bus.

UDPRx16SS2AXI4 is the module for converting the received data of four UDP10GRx-IPs which is 32-bit bus size to be 64-bit AXI4-ST bus with Metadata bus. While 64-bit AXI4-ST transfers the received data, Metadata transfers the active session number, session no#0-#15. Also, UDPRx16SS2AXI4 includes asynchronous logic to convert clock domain from MacRxClk, output from PMA, that is fixed to 322.266 MHz to be application clock domain (AppClk) that is user clock.

Besides, TxEMACMux4to1 is the module to be data multiplexer that receives TxEMAC I/F of four UDP10GRx-IP to send to TxEMAC I/F of LL10GEMAC via 32-bit AXI4-ST bus. TxEMAC is run on MacTxClk domain which is equal to 322.266 MHz.

LL10GEMAC-IP is IP core provided by Design Gateway while PMA module (BASE-R) is provided by Xilinx without charge. PMARstCtrl is the logic to control reset sequence of Xilinx Transceiver.

“trans_udp_multi” is the application, run on Windows OS, to transfer UDP data up to 16 sessions by using Multicast mode or Unicast mode.

2.1 Xilinx Transceiver (PMA for 10GBASE-R)

PMA IP core for 10Gb Ethernet (BASE-R) is 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

More details of Transceiver wizard in UltraScale/UltraScale+ FPGA are 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 includes state machine to run following steps.

(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) Complete Tx reset sequence and de-assert Tx reset, user interface output, to allow the user logic beginning Tx operation.
(4) Assert Rx reset to the transceiver.
(5) Wait until Rx reset done, output from the transceiver, is asserted to ‘1’.
(6) Complete Rx reset sequence and de-assert Rx reset, user interface output, to allow the user logic beginning Rx operation.

2.4 TxEMACMux4to1

The system consists of four UDP10GRx-IPs which shares the same EMAC. Therefore, TxEMACMux4to1 is designed to transfer the transmitted packet from four UDP10GRx-IPs to one EMAC. The core signal inside this module is rChSel that is applied to select one active UDP10GRx-IP module from four modules. Timing diagram of this module is displayed in Figure 2-2.

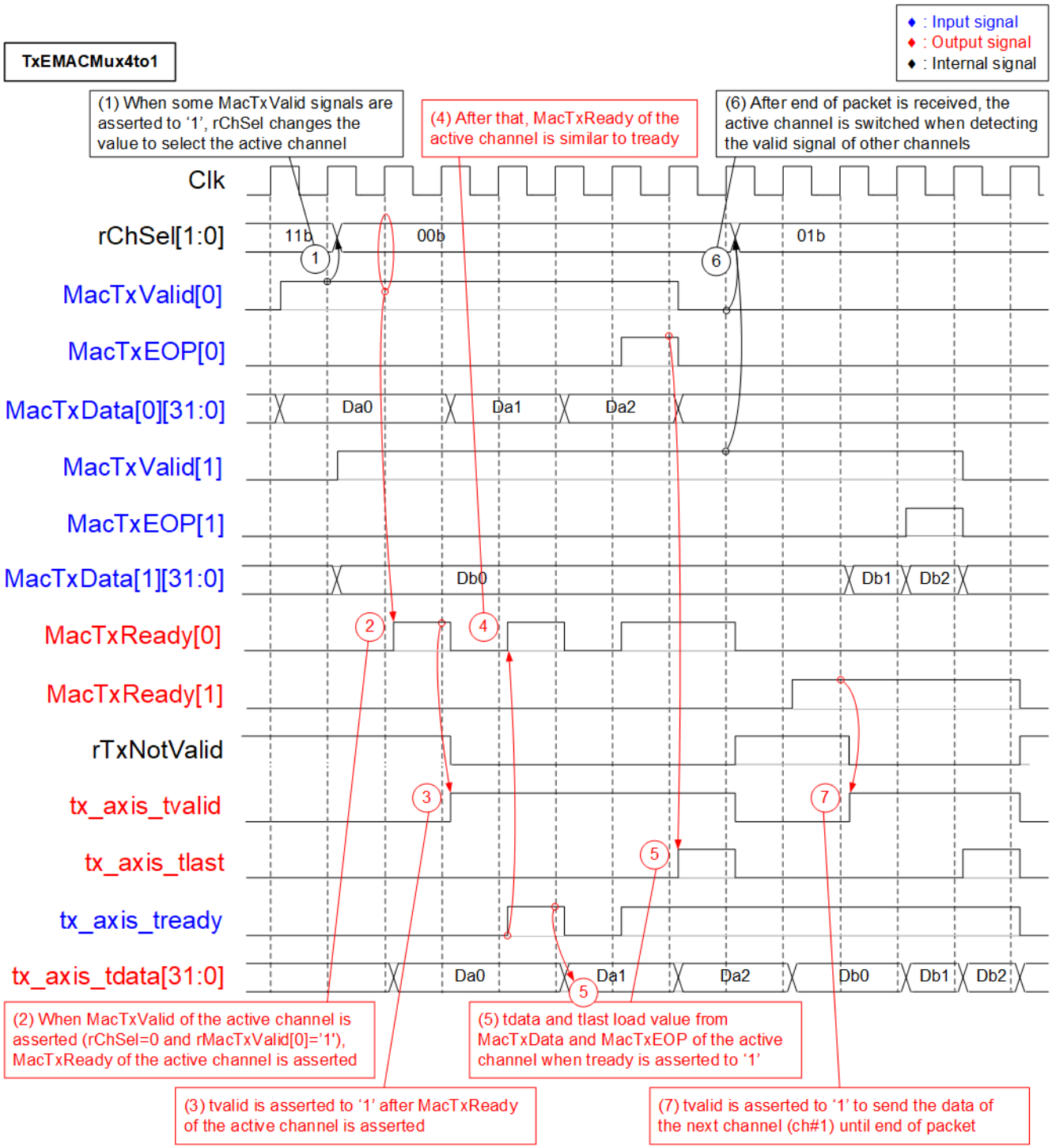

Figure 2-2 TxEMACMux4to1 timing diagram

(1) If there is no data transferred in the current channel and the data valid of the inactive channel is asserted to ‘1’, rChSel (the signal to indicate the active channel) will be switched to the new channel. As shown in Figure 2-2, rChSel changes from 11b to 00b to forwarding the data from Ch#0 to EMAC.
Note: Tx Latency time of TxEMACMux4to1 depends on rChSel characteristic.
a) If the active channel does not change (Receive two packets from the same channel), Tx latency time to assert tx_axis_tvalid after MacTxValid asserted is about 2 cycles.
b) When the active channel is changed (Two packets are received from different channel), Tx latency time from MacTxValid to tx_axis_tvalid is equal to 3 cycles.

(2) MacTxReady of the active channel is asserted to ‘1’ to accept the data from the active channel that asserts MacTxValid to ‘1’.

(3) After MacTxValid and MacTxReady of the active channel (Ch#0) are asserted to ‘1’, tx_axis_tvalid is asserted to ‘1’ to send the data of the active channel (MacTxData[0]) to EMAC.

(4) During transferring a packet, MacTxReady can be de-asserted to ‘0’ when EMAC pauses data reception by de-asserting tx_axis_tready to ’0’. Therefore, MacTxReady is similar to tx_axis_tready until transferring end of packet.

(5) tx_axis_tlast is asserted to ‘1’ to send the final data of packet when MacTxEOP of the active channel is asserted to ‘1’. The packet transmission is finished when EMAC accepts the final data by asserting tx_axis_tready to ‘1’.

(6) After transferring the final data, MacTxValid of the current channel must be de-asserted to ‘0’. The logic starts scanning MacTxValid of other channels. If there is data requested from the inactive channel, rChSel changes the value to change the active channel. As shown in Figure 2-2, rChSel changes from 00b to 01b to start receiving the data from Ch#1.

(7) After that, the data of Ch#1 is transferred to EMAC until end of a packet. Step (2) – (5) are repeated.

2.5 UDP10GRxIP

The IP core by Design Gateway is designed to receive and decode UDP/IP packet from EMAC. UDP payload 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-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

2.6 UDPRx16SS2AXI4

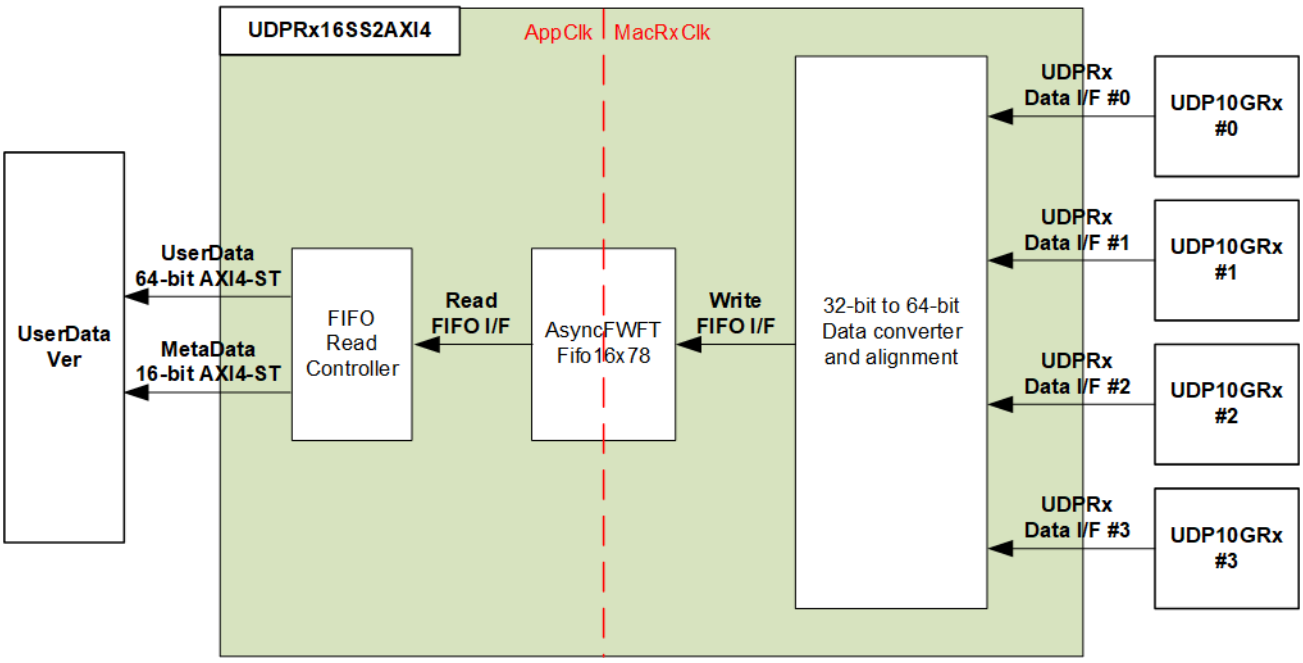

Figure 2-3 UDPRx16SS2AXI4 Block diagram

One UDP10GRx-IP supports to receive four session data from the same EMAC. When 16 session data is required, four UDP10GRx-IPs must be applied. User interface of UserDataVer has two AXI4-ST bus, 64-bit UserData and 16-bit MetaData. 64-bit UserData is applied to receive UDP payload data from 16 session. MetaData is the bus to send the session number in binary unit, so 4-bit value (0000b – 1111b) is assigned for 16 sessions. One MetaData is transferred to show the session number of one packet data on UserData bus.

UDPRx16SS2AXI4 forwards 16 session data from four UDP10GRx-IPs to UserDataVer by using AXI4-ST interface. Four UDP10GRx-IPs shares the same EMAC, so there is one session data forwarded from UDP10GRx-IPs. The data stream from UDP10GRx-IPs that is synchronous on MacRxClk must be converted to AppClk domain. Also, the session number must be encoded with the data stream. Small asynchronous FIFO with FWFT type (AsyncFWFTFifo16x78) is applied to convert 32-bit data bus in MacRxClk domain (322.265 MHz) to 64-bit data bus in AppClk domain (independent clock). Data size in FIFO is 78 bits (64-bit data, 8-bit byte enable, end-of-packet flag, error flag, and 4-bit session ID). The logics inside UDPRx16SS2AXI4 are divided to two groups, 32-bit to 64-bit data converter and alignment for writing FIFO and FIFO read controller for reading FIFO. More details are shown in Figure 2-4 - Figure 2-6.

Figure 2-4 shows timing diagram of 32-bit to 64-bit data converter and alignment to write FIFO when receiving the data from session#2 of UDP10GRx-IP#0. Total number of received data is aligned to 64-bit, so the dummy data is not filled by the logic.

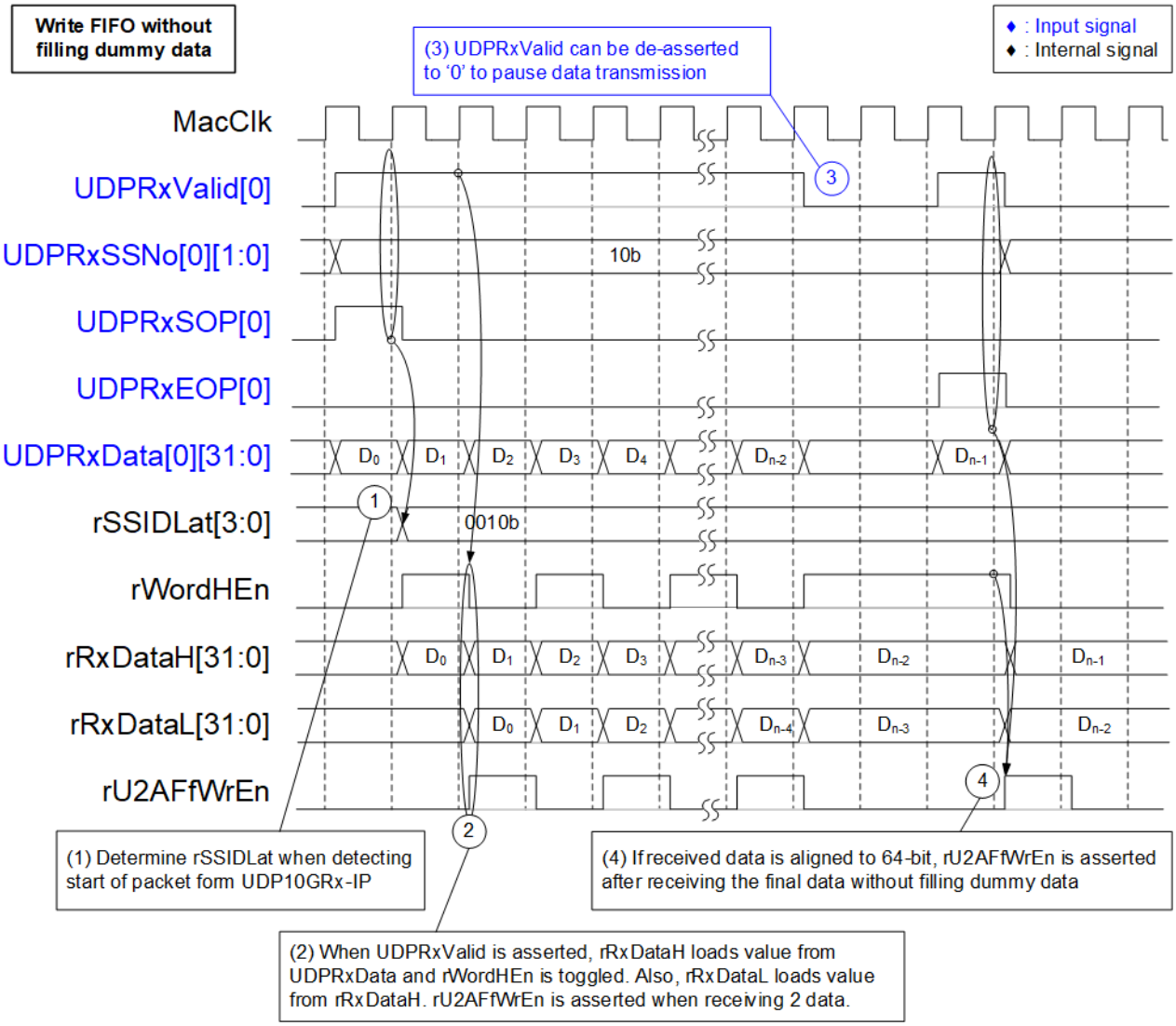

Figure 2-4 Timing diagram to write FIFO without filling dummy data

(1) When the new packet is received (UDPRxValid='1' and UDPRxSOP='1'), the session ID is decoded. For instance, rSSIDLat=2 when session#2 of UDP10GRx-IP#0 is transferred. rSSIDLat does not change until receiving the first data of the new packet.
Note: UDPRxValid can be asserted only 1 session at a time.

(2) When the data is received (UDPRxValid='1'), the received data (UDPRxData) of the active session is loaded to rRxDataH. At the same time, rWordHEn toggles the value and rRxDataL loads the value from rRxDataH. When two data is received (UDPRxValid='1' and rWordHEn='1'), 64-bit data (rRxDataH and rRxDataL) is stored to FIFO by asserting FIFO write enable (rU2AFfWrEn) to '1'.

(3) When the data is not ready, UDP10GRx-IP de-asserts UDPRxValid to '0' to pause data transmission. The logic to write the FIFO also pauses the operation (rU2AFfWrEn='0').

(4) When the final data of a packet is received with 64-bit alignment (UDPRxValid='1', UDPRxEOP='1', and rWordHEn='1'), the final 64-bit data is stored to FIFO without filling dummy data to align 64-bit.

The example when the received data is not aligned to 64-bit and one dummy data must be filled is shown in Figure 2-5. The data is received from session#1 of UDP10GRx-IP#1.

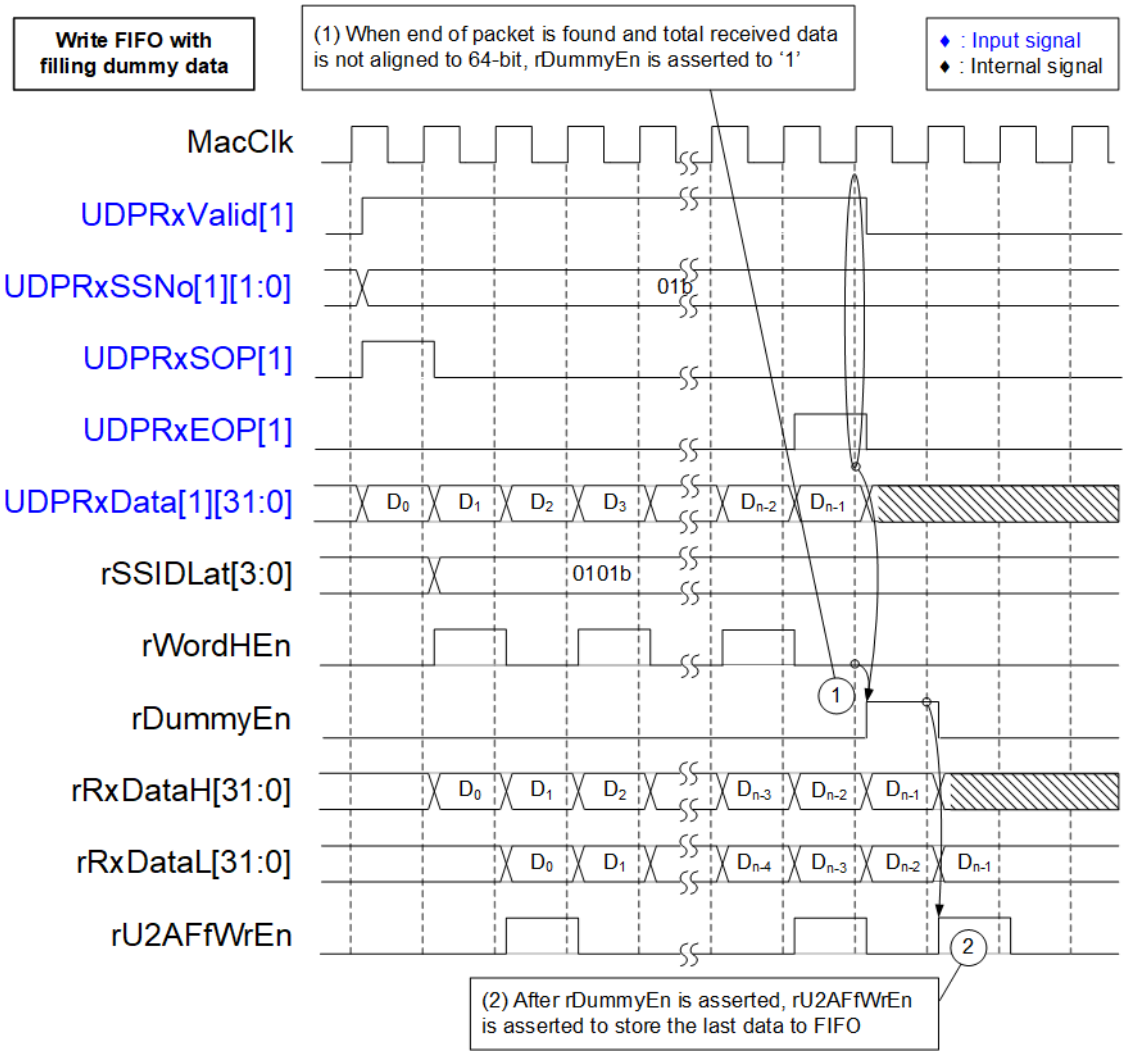

Figure 2-5 Timing diagram to write FIFO with filling dummy data

(1) When the final data of a packet is received and total number of received data is not aligned to 64-bit (UDPRxValid='1', UDPRxEOP='1', and rWordHEn='0'), rDummyEn is asserted to '1' to fill one dummy 32-bit data.

(2) After rDummyEn is asserted, rRxDataL loads the last data from rRxDataH while the dummy data is loaded to rRxDataH. At the same time, rU2AFfWrEn is asserted to '1' to store the final data which consists of 32-bit final data and 32-bit dummy data to FIFO.

Figure 2-6 shows timing diagram of FIFO Read controller to read FIFO and transfer read data to UserDataVer via 64-bit UserData AXI4-ST bus and session number via 16-bit MetaData AXI4-ST bus. One MetaData is written for one packet of UserData. MetaData is sent at the same time as sending the first data on UserData.

FIFO in the design is FWFT type, so Read data of FIFO (U2AFfRdData) is valid at the same time as Read enable of FIFO (wU2AFfRdAck) is asserted to '1'. 78-bit read data from FIFO (U2AFfRdData) description is assigned by following signals.

Bit[63:0] : 64-bit data
Bit[71:64] : 8-bit byte enable
Bit[72] : Last flag to show end of packet
Bit[73] : Error flag of the packet
Bit[77:74] : 4-bit Session ID

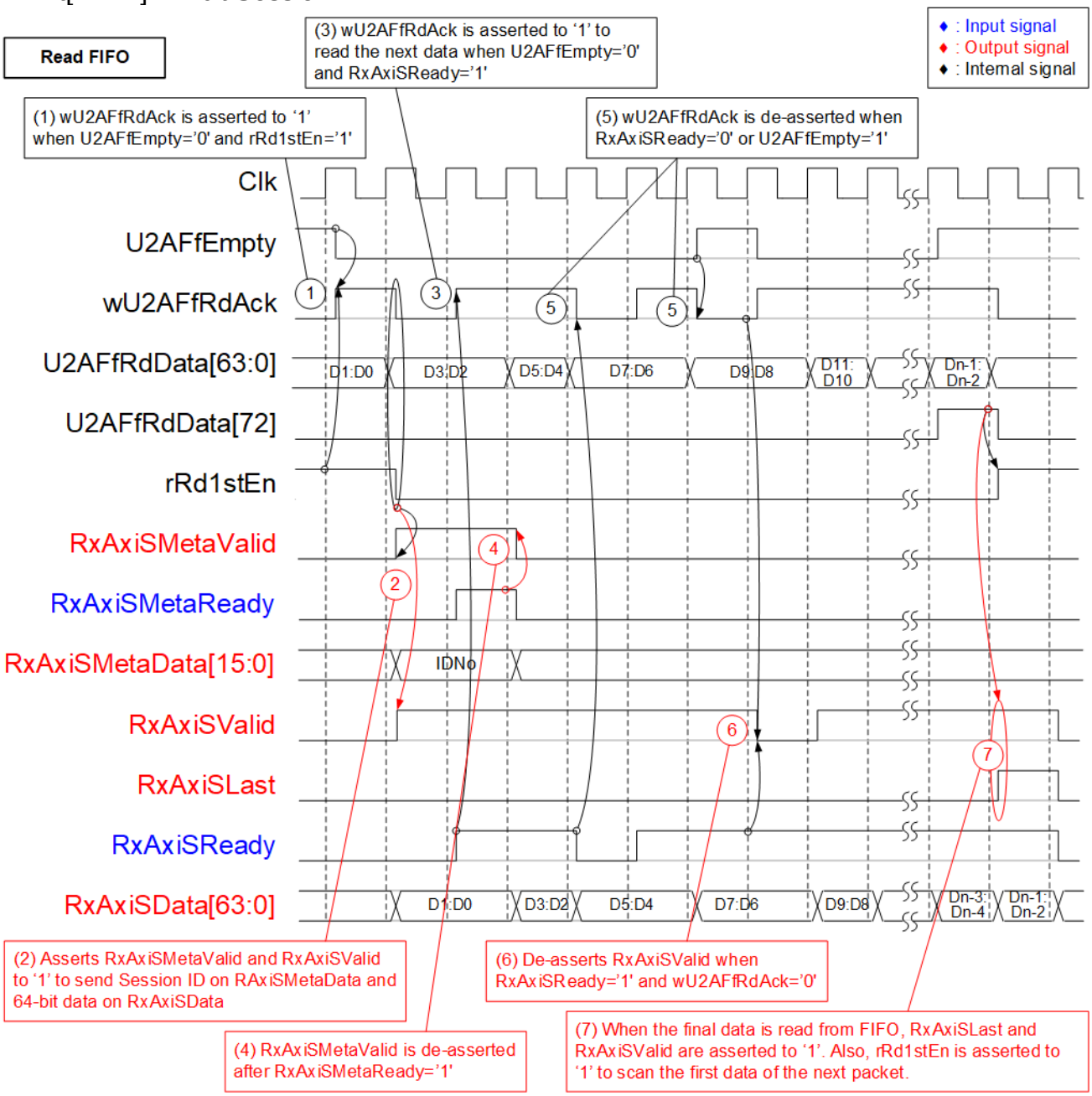

Figure 2-6 Timing diagram of FIFO Read controller

(1) rRd1stEn is asserted to ‘1’ when the read data from FIFO is the first data of a packet. FIFO read enable (wU2AFfRdAck) is asserted to ‘1’ by two methods.
   (i) If the data is the first data of a packet (rRd1stEn=’1’), wU2AFfRdAck is asserted to ‘1’ when FIFO is not empty (U2AFfEmpty=’0’).
   (ii) If the data is not the first data of a packet (rRd1stEn=’0’), wU2AFfRdAck is asserted to ‘1’ when FIFO is not empty (U2AFfEmpty=’0’) and RxAxiSReady=’1’.

   FIFO is FWFT type, so the read data of FIFO (U2AFfRdData) is valid at the same clock as wU2AFfRdAck asserted.

(2) Bit[77:74] of U2AFfRdData is loaded to Meta data output (RxAxiSMetaData) to show the session ID with asserting valid signal (RxAxiSMetaValid=’1’) when the first data is read (wU2AFfRdAck=’1’ and rRd1stEn=’1’). Bit[63:0] of U2AFfRdData is loaded to User data output (RxAxiSData) with asserting valid signal (RxAxiSValid=’1’) when FIFO read enable is asserted (wU2AFfRdAck=’1’). Besides, rRd1stEn is de-asserted to ‘0’ after the first data is read from FIFO.

(3) After reading the first data, the read enable of FIFO (wU2AFfRdAck) is controlled by FIFO empty (U2AFfEmpty) and User ready (RxAxiSReady). The read enable is asserted to ‘1’ when U2AFfEmpty=’0’ and RxAxiSReady=’1’. U2AFfRdData[63:0] is loaded to be RxAxiSData.

(4) RxAxisMetaValid is de-asserted to ‘0’ after the user sends acknowledge by asserting RxAxiSMetaReady to ‘1’.

(5) wU2AFfRdAck is de-asserted to ‘0’ to pause reading data when FIFO is empty (U2AFfEmpty=’1’) or the user is not ready (RxAxiSReady=’0’).

(6) If the data is sent to user (RxAxiSValid=’1’ and RxAxiSReady=’1’) but wU2AFfRdAck is de-asserted to ‘0’, RxAxiSValid will be de-asserted to ‘0’ in the next cycle. It will be re-asserted to ‘1’ when wU2AFfRdAck is re-asserted to ‘1’.

(7) When the final data of a packet is read from FIFO (U2AFfRdData[72]=’1’), the last flag (RxAxiSLast) is asserted to user. Also, rRd1stEn is re-asserted to ‘1’ to scan the first data of the next packet.

2.7 CPU and Peripherals

32-bit AXI4-Lite bus is applied to be the bus interface for CPU accessing the peripherals such as Timer and UART. The test logic is connected with CPU as a peripheral on 32-bit AXI4-Lite bus for CPU controlling and monitoring. CPU assigns the base address and the range for accessing the test logic. Therefore, the hardware logic must support AXI4-Lite bus standard for CPU writing and reading. LAxi2Reg module is designed to connect the CPU system via AXI4-Lite bus as shown in Figure 2-7.

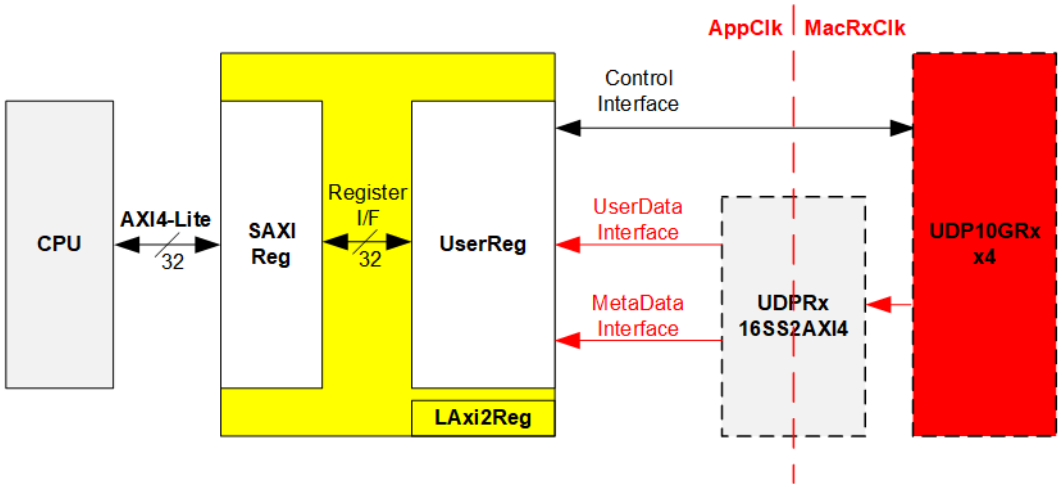

Figure 2-7 LAxi2Reg block diagram

LAxi2Reg consists of SAXIReg and UserReg. SAXIReg converts the AXI4-Lite signals to be the simple register interface which has 32-bit data bus size (similar to AXI4-Lite data bus size). UserReg includes the register file to set test parameters and store the status signals of the test logic. Data interface of UDPRx16SS2AXI4 is connected to UserReg to verify the received data. Also, the control interface of four UDP10GRx-IPs is connected to UserReg. More details of SAXIReg and UserReg are described as follows.

2.7.1 SAXIReg

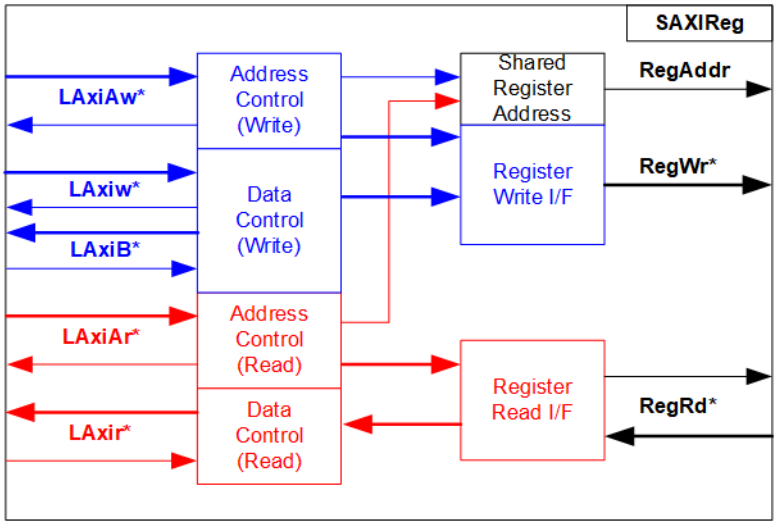

Figure 2-8 SAXIReg Interface

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 SAXIReg 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 2-8. Write control I/F and Write data I/F of AXI4-Lite bus are latched and transferred to be Write register interface. 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. 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 2-9.

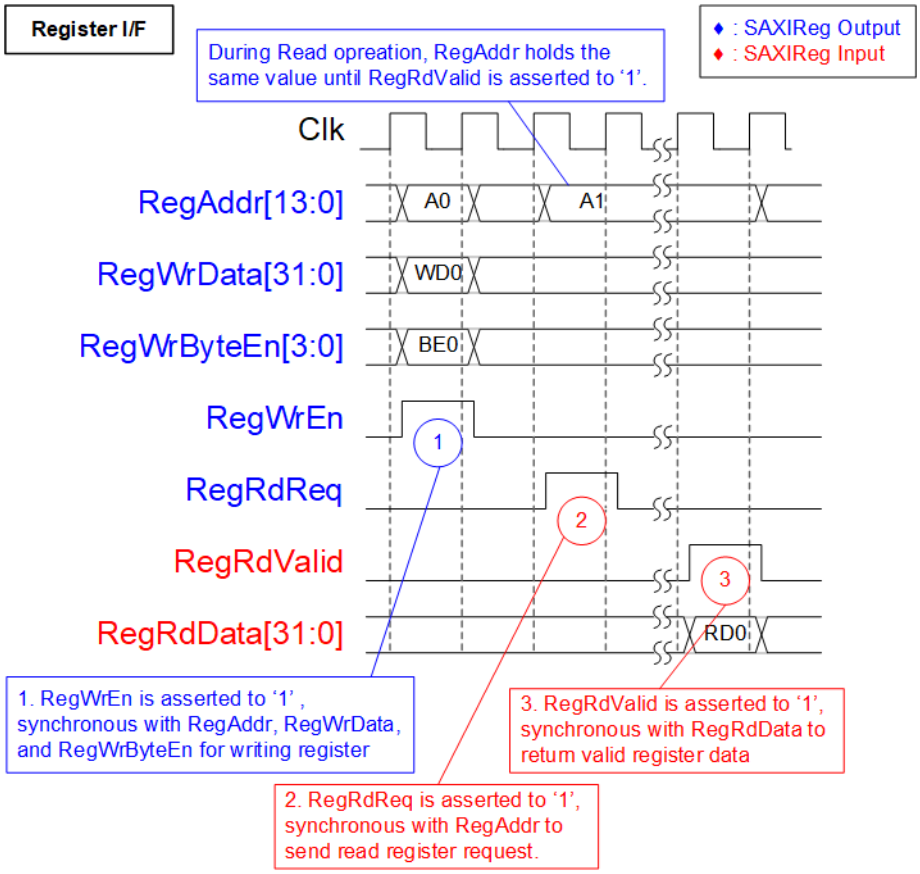

Figure 2-9 Register interface timing diagram

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, SAXIReg 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 multiplexers.
3) The read data is returned on RegRdData bus by the slave with asserting RegRdValid to '1'. After that, SAXIReg forwards the read value to LAxir* interface.

2.7.2 UserReg

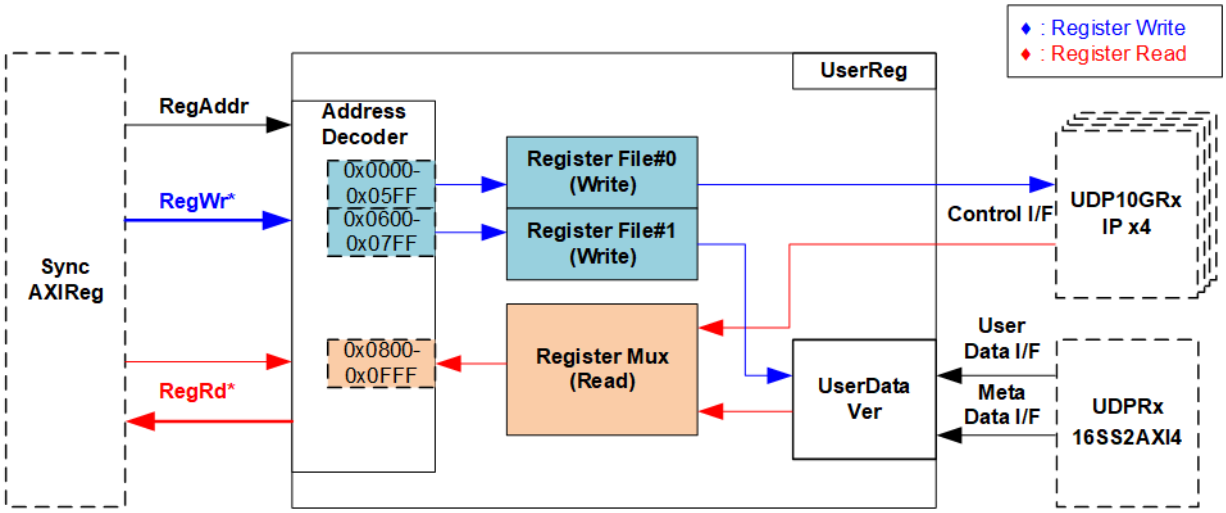

Figure 2-10 UserReg block diagram

The logic inside UserReg is divided to two parts, Register interface and UserDataVer. UseDataVer is the module to verify 16 session data, output of UDPRx16SSAXI4 which describes in more details in the next topic. The logic for register interface is split to three parts. First is the address decoder to decode the address to select the active register for write and read access. Second is the register file to store the test parameters that are set by user such as the network parameters of UDP10GRx-IP and the initial value to verify data of UserDataVer. Last is the data multiplexer to select the read data to return to CPU. Many signals are mapped from UDP10GRx-IP and UserDataVer for CPU reading.

Address Decoder

The address range inside UserReg is split to three areas, two areas for write access and one area for read access, as shown in Figure 2-10.

1) 0x0000 – 0x05FF: Write only area for setting the parameters of four UDP10GRx-IPs such as network parameters.
2) 0x0600 – 0x07FF: Write only area for setting the parameters of UserDataVer modules such as initial value of expected pattern.
3) 0x0800 – 0x0FFF: Read only area for reading the status signals of four UDP10GRx-IPs such as Error flags and active channel. Also, the status signals of UserDataVer such as current transfer size are mapped.

Address decoder decodes the upper bit of RegAddr to select active region while the lower bit is fed to Register File and Register Mux for selecting the active register.

Register

The register file inside UserReg is 32-bit bus size. Therefore, write byte enable (RegWrByteEn) is not applied in the test system. CPU uses 32-bit pointer to set the hardware registers. To read register, multiplexer is designed to select the active status register to return read data to CPU. There are many status registers for reading, so many multiplexers are applied to balance the number of inputs for each multiplexer. Totally, the latency of read data is equal to three clock cycles. Therefore, RegRdValid is created by RegRdReq with asserting three D Flip-flops. The address map in UserReg module is shown in Table 2-1.

Table 2-1 Register map Definition

Address Wr/Rd	Register Name (Label in the udp10grx16sstest.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: Shared signals		
BA+0x0000	Session Enable Reg (UDP_SSEN_REG)	[15:0]: Input to be Session enable of UDP10GRx-IP#0-#3 ([3:0]: SSEnable[3:0] of UDP10GRx-IP#0, [7:4]: SSEnable[3:0] of UDP10GRx-IP#1, [11:8]: SSEnable[3:0] of UDP10GRx-IP#2, [15:12]: SSEnable[3:0] of UDP10GRx-IP#3)
BA+0x0004	Multicast Enable Reg (UDP_MCEN_REG)	[0]: Input to be Multicast enable of UDP10GRx-IP (McastEn of UDP10GRx-IP#0-3)
BA+0x0040	Source MAC Address (Low) Reg (UDP_SML_REG)	[31:0]: Input to be source MAC address (SrcMacAddr[31:0] of UDP10GRx-IP#0-3)
BA+0x0044	Source MAC Address (High) Reg (UDP_SMH_REG)	[15:0]: Input to be source MAC address (SrcMacAddr[47:32] of UDP10GRx-IP#0-3)
BA+0x0048	Source IP Address Reg (UDP_SIP_REG)	[31:0]: Input to be source IP address (SrcIPAddr of UDP10GRx-IP#0-3)
BA+0x0200 – BA+0x027F: UDP10GRx-IP#0, BA+0x0280 – BA+0x02FF: UDP10GRx-IP#1 BA+0x0300 – BA+0x037F: UDP10GRx-IP#2, BA+0x0380 – BA+0x03FF: UDP10GRx-IP#3		
BA+0x0200	Source Port Num Reg (UDP_SPN0_REG)	[15:0]: Input to be source port number#0 (SrcPort0 of UDP10GRx-IP#0)
BA+0x0204	Target Port Num Reg (UDP_DPN0_REG)	[15:0]: Input to be target port number#0 (DstPort0 of UDP10GRx-IP#0)
BA+0x0208	Target IP Address Reg (UDP_DIP0_REG)	[31:0]: Input to be target IP address#0 (DstIPAddr0 of UDP10GRx-IP#0)
BA+0x020C	Multicast IP Address Reg (UDP_MIP0_REG)	[31:0]: Input to be multicast IP address#0 (McastIPAddr0 of UDP10GRx-IP#0)
BA+0x0220- BA+0x022C	Session#1 parameters (UDP_SPN1_REG - UDP_MIP1_REG)	0x0220: SrcPort1 of UDP10GRx-IP#0 0x0224: DstPort1 of UDP10GRx-IP#0 0x0228: DstIPAddr1 of UDP10GRx-IP#0 0x022C: McastIPAddr1 of UDP10GRx-IP#0
BA+0x0240- BA+0x024C	Session#2 parameters (UDP_SPN2_REG – UDP_MIP2_REG)	0x0240: SrcPort2 of UDP10GRx-IP#0 0x0244: DstPort2 of UDP10GRx-IP#0 0x0248: DstIPAddr2 of UDP10GRx-IP#0 0x024C: McastIPAddr2 of UDP10GRx-IP#0
BA+0x0260- BA+0x026C	Session#3 parameters (UDP_SPN3_REG – UDP_MIP3_REG)	0x0260: SrcPort3 of UDP10GRx-IP#0 0x0264: DstPort3 of UDP10GRx-IP#0 0x0268: DstIPAddr3 of UDP10GRx-IP#0 0x026C: McastIPAddr3 of UDP10GRx-IP#0
BA+0x0280- BA+0x02EC	Session#4 - #7 parameters	Similar to 0x0200 – 0x27F (UDP10GRx-IP#0), define parameters of Session#4-#7 to SS#0 – SS#3 of UDP10GRx-IP#1
BA+0x0300- BA+0x036C	Session#8 - #11 parameters	Similar to 0x0200 – 0x27F (UDP10GRx-IP#0), define parameters of Session#8-#11 to SS#0 – SS#3 of UDP10GRx-IP#2
BA+0x0380- BA+0x03EC	Session#12 - #15 parameters	Similar to 0x0200 – 0x27F (UDP10GRx-IP#0), define parameters of Session#12-#15 to SS#0 – SS#3 of UDP10GRx-IP#3

Address Wr/Rd	Register Name (Label in the udp10grx16sstest.c”)	Description
BA+0x0600 – BA+0x07FF: 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 flags (UserError) of UserDataVer which are mapped to USER_ERR0_7/8_15_REG. Set to ‘1’ to clear error signals. This flag is auto-cleared.
BA+0x0680- BA+0x06BC	User Start Pattern0-15 Reg (USER_PAT0-15_REG)	[31:0]: Start value of 32-bit incremental pattern for verifying data in Session#0-#15 of UserDataVer. 0x680: SS#0, 0x684: SS#1, 0x688: SS#2, 0x68C: SS#3, 0x690: SS#4, 0x694: SS#5, 0x698: SS#6, 0x69C: SS#7, 0x6A0: SS#8, 0x6A4: SS#9, 0x6A8: SS#10, 0x6AC: SS#11, 0x6B0: SS#12, 0x6B4: SS#13, 0x6B8: SS#14, 0x6BC: SS#15
BA+0x0800 – BA+0x0FFF: 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]: IP version of LL10GEMAC-IP
BA+0x0808	UDP10GRx-IP Version (UDP_VER_REG)	[31:0]: IP Version of UDP10GRx-IP#0
BA+0x080C	Session Active of UDP10GRx-IP (UDP_SSAC_REG)	[15:0] Session active status. (‘0’: Not active, ‘1’: Active). Bit[0]: SS#0, [1]: SS#1, …, and [15]: SS#15. ([3:0]: SSActive[3:0] of UDP10GRx-IP#0, [7:4]: SSActive[3:0] of UDP10GRx-IP#1, [11:8]: SSActive[3:0] of UDP10GRx-IP#2, [15:12]: SSActive[3:0] of UDP10GRx-IP#3)
BA+0x0820	Test pin of UDP10GRx-IP#0 (UDP_TESTPIN0_REG)	[31:0]: Mapped to TestPin of UDP10GRx-IP#0
BA+0x0824	Test pin of UDP10GRx-IP#1 (UDP_TESTPIN1_REG)	[31:0]: Mapped to TestPin of UDP10GRx-IP#1
BA+0x0828	Test pin of UDP10GRx-IP#2 (UDP_TESTPIN2_REG)	[31:0]: Mapped to TestPin of UDP10GRx-IP#2
BA+0x082C	Test pin of UDP10GRx-IP#3 (UDP_TESTPIN3_REG)	[31:0]: Mapped to TestPin of UDP10GRx-IP#3
BA+0x0840	User Error SS#0-#7 Reg (USER_ERR0_7_REG)	[31:0] Error status of Session#0 - #7. [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]: SS#1 error, similar to bit[1:0]. [9:8], [13:12], [17:16], [21:20], [25:24], [29:28]: SS#2, #3, .., #7 error.
BA+0x0844	User Error SS#8-#15 Reg (USER_ERR8_15_REG)	[31:0] Error status of Session#8 - #15, similar to 0x0840
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. Time unit is 3.1 ns.
BA+0x0864	UDP10G to AXI4 Latency time Reg (UDP2AXI_TIME_REG)	[31:0]: Latency time of received data of UDPRx16SS2AXI4, measured from start-of-packet to start-of-packet. Time unit is 3.1 ns.

Address Wr/Rd	Register Name (Label in the udp10grx16sstest.c”)	Description
BA+0x0800 – BA+0x0FFF: Output signals of UDP10GRx-IP and UserReg (Read access only)		
BA+0x0C00	Received byte size#0 (Low) Reg (USER_RXLENL0_REG)	[31:0]: The lower 32-bit of current received data size in byte unit when receiving data from Session#0
BA+0x0C04	Received byte size#0 (High) Reg (USER_RXLENH0_REG)	[15:0]: The upper 16-bit of current received data size in byte unit when receiving data from Session#0
BA+0x0C08	Failure byte position#0 (Low) Reg (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+0x0C0C	Failure byte position#0 (High) Reg (USER_RDFAILL0_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+0x0C10	Expected data#0 (Low) Reg (USER_EXPPATL0_REG)	[31:0]: The lower 32-bit of expected data at the 1st failure data when receiving data from Session#0
BA+0x0C14	Expected data#0 (High) Reg (USER_EXPPATH0_REG)	[31:0]: The upper 32-bit of expected data at the 1st failure data when receiving data from Session#0
BA+0x0C18	Received data#0 (Low) Reg (USER_RDPATL0_REG)	[31:0]: The lower 32-bit of received data at the 1st failure data when receiving data from Session#0
BA+0x0C1C	Received data#0 (High) Reg (USER_RDPATH0_REG)	[31:0]: The upper 32-bit of received data at the 1st failure data when receiving data from Session#0
BA+0x0C20- BA+0x0C3C	USER_RXLENL1_REG – USER_RDPATH1_REG	Similar to 0x0C00-0x0C1C, the received data status when receiving data from Session#1
BA+0x0C40- BA+0x0C5C	USER_RXLENL2_REG – USER_RDPATH2_REG	Similar to 0x0C00-0x0C1C, the received data status when receiving data from Session#2
BA+0x0C60- BA+0x0C7C	USER_RXLENL3_REG – USER_RDPATH3_REG	Similar to 0x0C00-0x0C1C, the received data status when receiving data from Session#3
BA+0x0C80- BA+0x0DFC	USER_RXLENL4-15_REG – USER_RXPATH4-15_REG	Similar to 0x0C00-0x0C1C, the received data status when receiving data from Session#4 - #15

2.7.3 UserDataVer

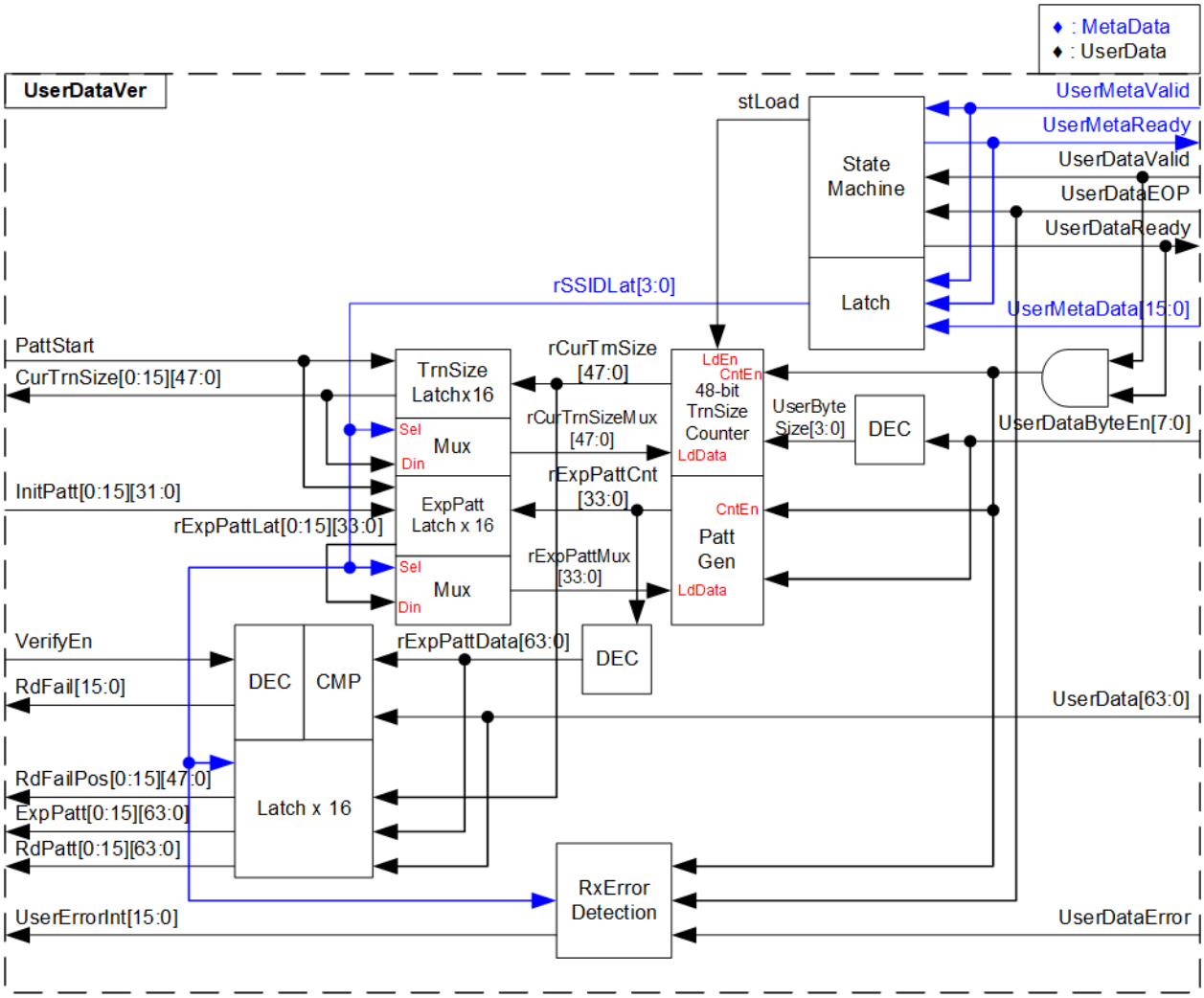

Figure 2-11 UserDataVer

UserDataVer is the example logic to show how to interface with UDPRx16SS2AXI4 for receiving data from 16 sessions. As shown in the right side of Figure 2-11, there are two user interfaces – MetaData and UserData. UserDataVer verifies the received data and monitors the error status.

The control signal and initial value for verification from Register are set by Register inside UserReg, as shown in the left side of Figure 2-11. PattStart is asserted to '1' to load the initial value of the pattern data in each session, assigned by InitPatt. Also, PattStart is applied to reset the value of the internal logic at the beginning of the new test loop.

MetaData

MetaData and UserData interface are controlled by State machine. Firstly, State machine reads the active session number from MetaData interface. UserMetaReady is asserted to '1' to receive one MetaData. After that, the session number is loaded to rSSIDLat. This value is valid until receiving the final data of a packet from UserData.

UserData

After loading MetaData, State machine asserts UserDataReady to ‘1’ to receive one packet data via UserData until receiving the final of a packet. When the received data is valid (UserDataValid=’1’ and UserDataReady=’1’), TrnSize counter to count the number of received data (rCurTrnSize) is increased. At the same time, PattGen is counted to create the next expected data (rExpPattCnt) to compare with the next received data (UserData). Typically, one cycle receives 8-byte user data except the final data of a packet that may be valid for 1-8 bytes, checked by UserDataByteEn. Therefore, UserDataByteEn is decoded to be the offset for counting rCurTrnSize and rExpPattCnt. The expected data is compared to the received data. If the value does not match, fail flag (RdFail) is asserted when user enables data verification (VerifyEn=’1’).

After receiving the final data of a packet, the current value of rCurTrnSize and rExpPattCnt are loaded the latch registers of the session, controlled by rSSIDLat. There are 16 latch registers to store the information of 16 sessions. Similarly, the failure information of the first failure data is latched to one of 16 session registers when fail flag is asserted. The failure information consists of failure position (RdRailPos), expected value (ExpPatt), and received value (RdPatt).

UserDataError

Besides, the error monitoring logic reads UserDataError when receiving the final data of packet. If UserDataError is asserted, UserErrorInt of the active session will be asserted. The error is cleared when the new test loop is run (PattStart=’1’).

After finishing receiving final data of packet, State machine returns to the first state to read the next session number from MetaData and read the received data of the next session from UserData.

3 CPU Firmware (FPGA)

```
+++ UDP10GRxIP 16 Session Demo [IPVer = 2.0]

 > Test with LL10GEMACIP Ver 2.3

+++ Set Common UDP10GRxIP parameter +++
 IP Mode                    = UNICAST
 FPGA MAC address           = 0x000102030405
 FPGA IP                    = 192.168.7.42
Press 'x' to skip common IP parameter setting: x

+++ Set Session UDP10GRxIP parameter +++

| Session | SSEnable | Target IP | Target Port Number | FPGA Port Number |
|---|---|---|---|---|
| 0 | ENABLE | 192.168.7.25 | 61000 | 4000 |

Press 'x' to skip SS# 0 IP parameter setting: x

+++ Current IP Parameter +++
 IP Mode                    = UNICAST
 FPGA MAC address           = 0x000102030405
 FPGA IP                    = 192.168.7.42

| Session | SSEnable | Target IP | Target Port Number | FPGA Port Number |
|---|---|---|---|---|
| 0 | ENABLE | 192.168.7.25 | 61000 | 4000 |
| 1 | DISABLE | - | - | - |
| 2 | DISABLE | - | - | - |
| 3 | DISABLE | - | - | - |
| 4 | DISABLE | - | - | - |
| 5 | DISABLE | - | - | - |
| 6 | DISABLE | - | - | - |
| 7 | DISABLE | - | - | - |
| 8 | DISABLE | - | - | - |
| 9 | DISABLE | - | - | - |
| 10 | DISABLE | - | - | - |
| 11 | DISABLE | - | - | - |
| 12 | DISABLE | - | - | - |
| 13 | DISABLE | - | - | - |
| 14 | DISABLE | - | - | - |
| 15 | DISABLE | - | - | - |

+++ IP initialization complete +++

--- UDP10GRxIP menu ---
[0] : Display UDP10GRxIP parameters
[1] : Set UDP10GRxIP parameters
[2] : Run Receive Data Test
```

1 Welcome message to show IP information

2 Display default value of common parameters and allow the user to change the value

3 Display default value of session parameters and allow the user to change the value for each session (SS#0-SS#15)

4 Display all parameters before running initialization process

5 Display main menu after finishing initialization process

Figure 3-1 Message on the console in initialization process

CPU firmware in the reference design is implemented on bare-metal OS to simply handle the hardware. After booting the test system, the system starts initialization process, as shown in Figure 3-1. User needs to configure network parameters and test parameters before starting initialization. More details of the initialization process are described as follows.

1) CPU reads UDP10GRx-IP and LL10GEMAC-IP information and displays on the console. 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 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 (for Multicast mode), Target port number, and FPGA port number. The session order for displaying parameters is Session#0, #1, …, and #15. 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 as follows.
   UDP_SML/H_REG = FPGA MAC address
   UDP_SIP_REG = FPGA IP address
   UDP_MCEN_REG = IP Mode
   UDP_SPN0-15_REG = FPGA port number
   UDP_DPN0-15_REG = Target port number
   UDP_DIP0-15_REG = Target IP address
   UDP_MIP0-15_REG = Multicast IP address
   UDP_SSEN_REG = Session enable
5) CPU waits until the initialization process is finished by monitoring UDP_SSAC_REG. UDP_SSAC_REG is equal to UDP_SSEN_REG after all sessions finish the initialization. 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 displays the current value of all parameters. The common parameters are displayed on the console before displaying the session parameters. After that, the session parameters of the active session are displayed. The sessions that is inactive shows “DISABLE” message.

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) Read the session active flag. 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. Otherwise, DISABLE is displayed.

3.2 Set parameters

This menu is applied 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 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 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 sixteen sessions in the demo which can be assigned by the user 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. At least one session must be enabled before running this menu. User sets total number of received data 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 sets 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. Otherwise, the operation is cancelled.
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 is displayed. The input value is set to USER_PAT0-15_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 parameter 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 of some active sessions (USER_RXLENL0-15_REG) is not equal to 0). After that, CPU starts timer to measure processing time.
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_RXLENL/H0-15_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_RXLENL/H0-15_REG.
8) Stop timer. Check error status by reading USER_ERR0_7_REG/USER_ERR8_15_REG. If some errors are asserted to ‘1’, the error message is displayed.
9) Calculate performance and display the result. Also, Rx latency time of UDP10GRx-IP and UDPRx16SS2AXI4 are read from UDPRX_TIME_REG and UDP2AXI_TIME_REG and then displayed on the console.

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
Return value	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 31.
Return value	None
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_RXLENL0-15_REG and USER_RXLENH0-15_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 error interrupt status from USER_ERR0_7_REG and USER_ERR8_15_REG. Display the session number that has the error asserted from the IP 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_RXLENL0-15_REG and USER_RXLENH0-15_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, UDPRX_TIME_REG and UDP2AXI_TIME_REG are read to show latency time of UDP10GRx-IP and UDPRx16SS2AXI4 module.

void show_ss_param(int ss_number, int show_mode)	
Parameters	ss_number: The session number for displaying, valid from 0 to 15. 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 failed flag from USER_ERR0_7_REG and USER_ERR8_15_REG. If the failed flag of which session is asserted, display the failure information by reading the register of error session, i.e., USER_RDFAILL/H0-15_REG, USER_EXPPATL/H0-15_REG, and USER_RDPATL/H0-15_REG to show failure position, expected data, and read data respectively.

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 detected.

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 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.

```
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 with displaying error message. More details of the first parameter group are described 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.

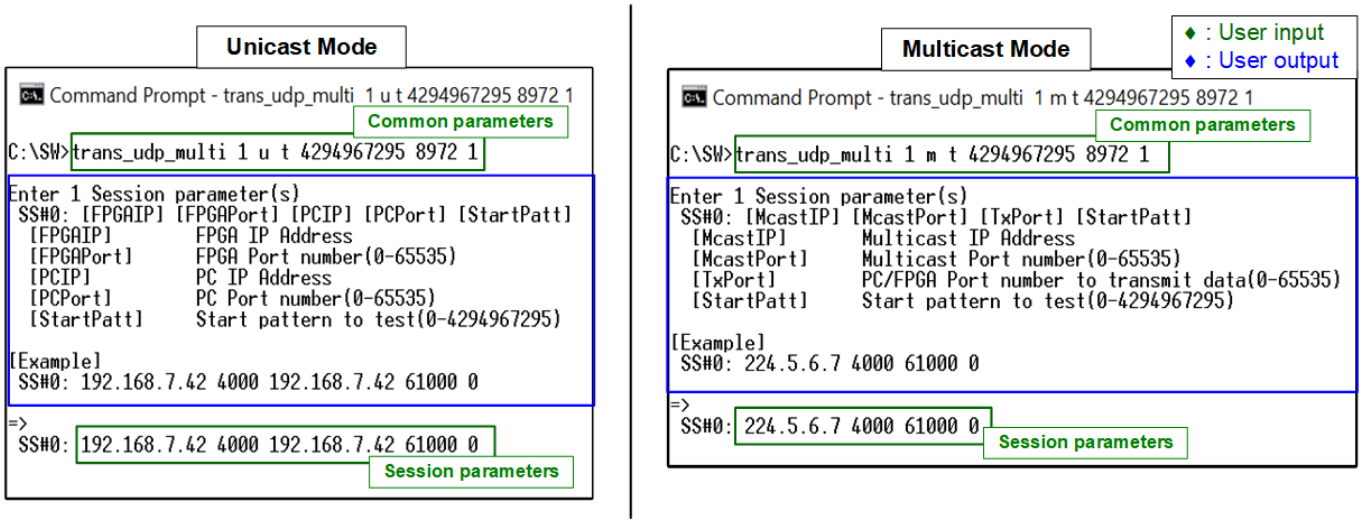

Figure 4-2 trans_udp_multi application to input session parameters

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 transmission begins after finishing setting 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, 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. During checking status, the main thread displays current transfer 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	23-Nov-21	Initial version release