LL10GEMAC in AAT Demo reference design

Rev1.1 14-Jun-23

1 Introduction

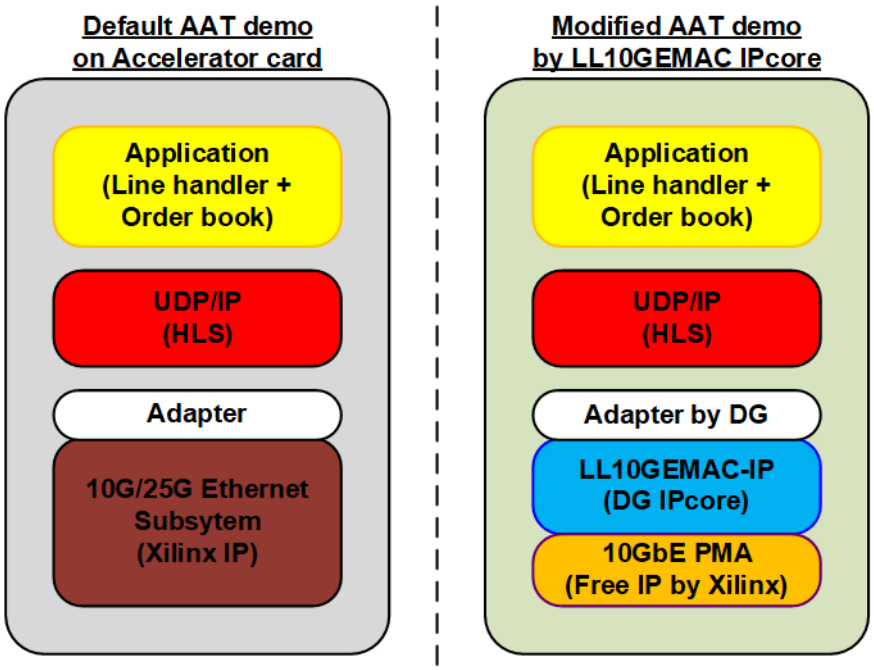

Figure 1-1 The comparison of modified AAT demo and default AAT demo

Xilinx provides and Accelerated Algorithmic Trading (AAT) that runs on an Accelerator card for trading applications which require low latency features. The AAT system is designed to reduce the CPU utilization and the latency of the market data processing system. More details about the AAT demo can be requested from the following link.
https://www.xilinx.com/applications/data-center/financial-technology/accelerated-algorithmic-trading.html

The default AAT demo uses a 10G/25G Ethernet Subsystem (Xilinx IP core) configured for low-latency applications. The user interface of the Ethernet Subsystem is 32-bit data interface at 312.5 MHz. The Ethernet kernel interface is a 64-bit interface at the application clock, which is configured by the application system. Therefore, the adapter logic must be designed to interface the 10G/25G Ethernet Subsystem with the application.

Similarly, LL10GEMAC-IP, which is Design Gateway IP core, uses a 32-bit interface at 322.266 MHz. The adapter logic is also designed by HDL to transfer packets between the application and LL10GEMAC-IP in both transfer directions with low latency time. By using LL10GEMAC-IP and the new adapter, the modified AAT demo achieves lower latency time than the default AAT demo. The PMA logic for 10GBASE-R to connect with LL10GEMAC-IP is free provided by Xilinx.

The applications of AAT demo such as UDP/IP, line handler, and order book are coded by HLS (High-Level Synthesis). Therefore, it is flexible for users to modify it to be compatible with each market data standard. The details about system overview, hardware components, and interface are described as follows.

2 Hardware overview

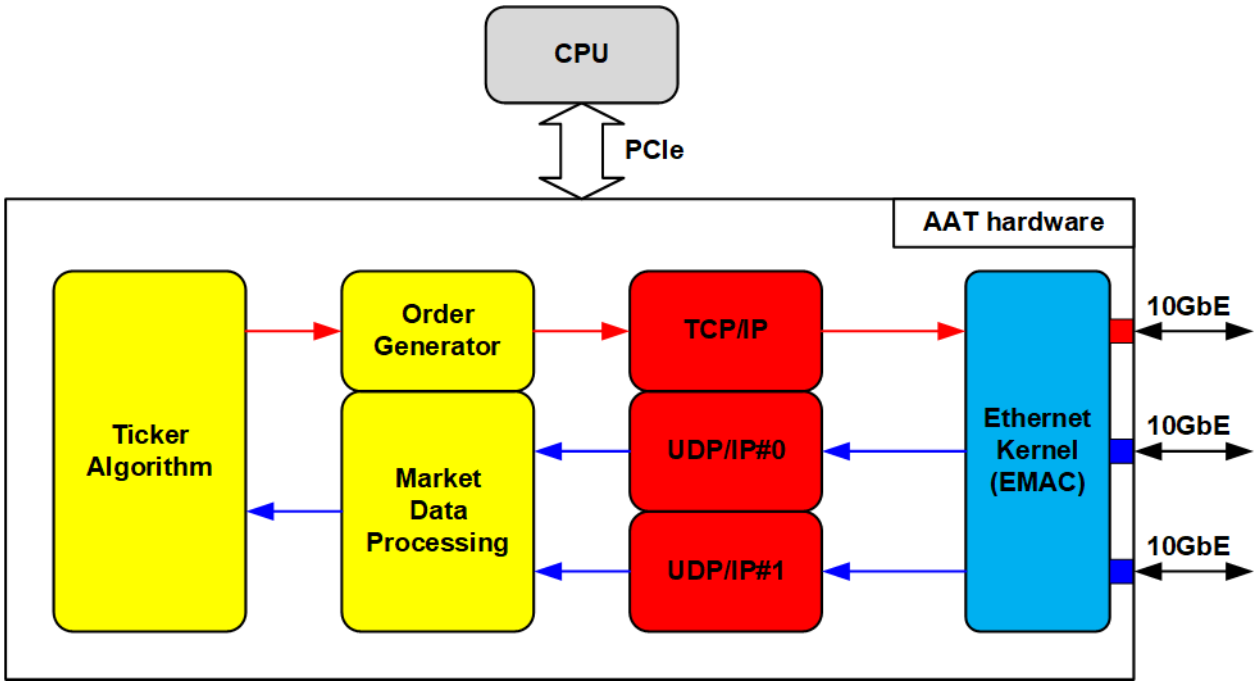

Figure 2-1 AAT hardware block diagram

As shown in Figure 1-1, the AAT hardware implemented on the Accelerator card consists of many blocks. Two 10G Ethernet channels are applied to receive market data using UDP/IP protocols. The output of two EMACs is connected to UDP/IP#0 and #1 for decoding UDP/IP packet. After processing the market data and the ticker condition is found, the trading order is generated by the Order Generator. The order packet uses TCP/IP protocol which is a reliable protocol, so the TCP/IP block is applied to create the order packet. The TCP/IP packet is transferred using the third 10G Ethernet channel.

This reference design modifies the Ethernet kernel by replacing LL10GEMAC-IP while the other modules are not modified. The software that runs on CPU is the default application. Thus, this document describes only the modification part of AAT demo which is Ethernet kernel hardware.

3 Ethernet Kernel

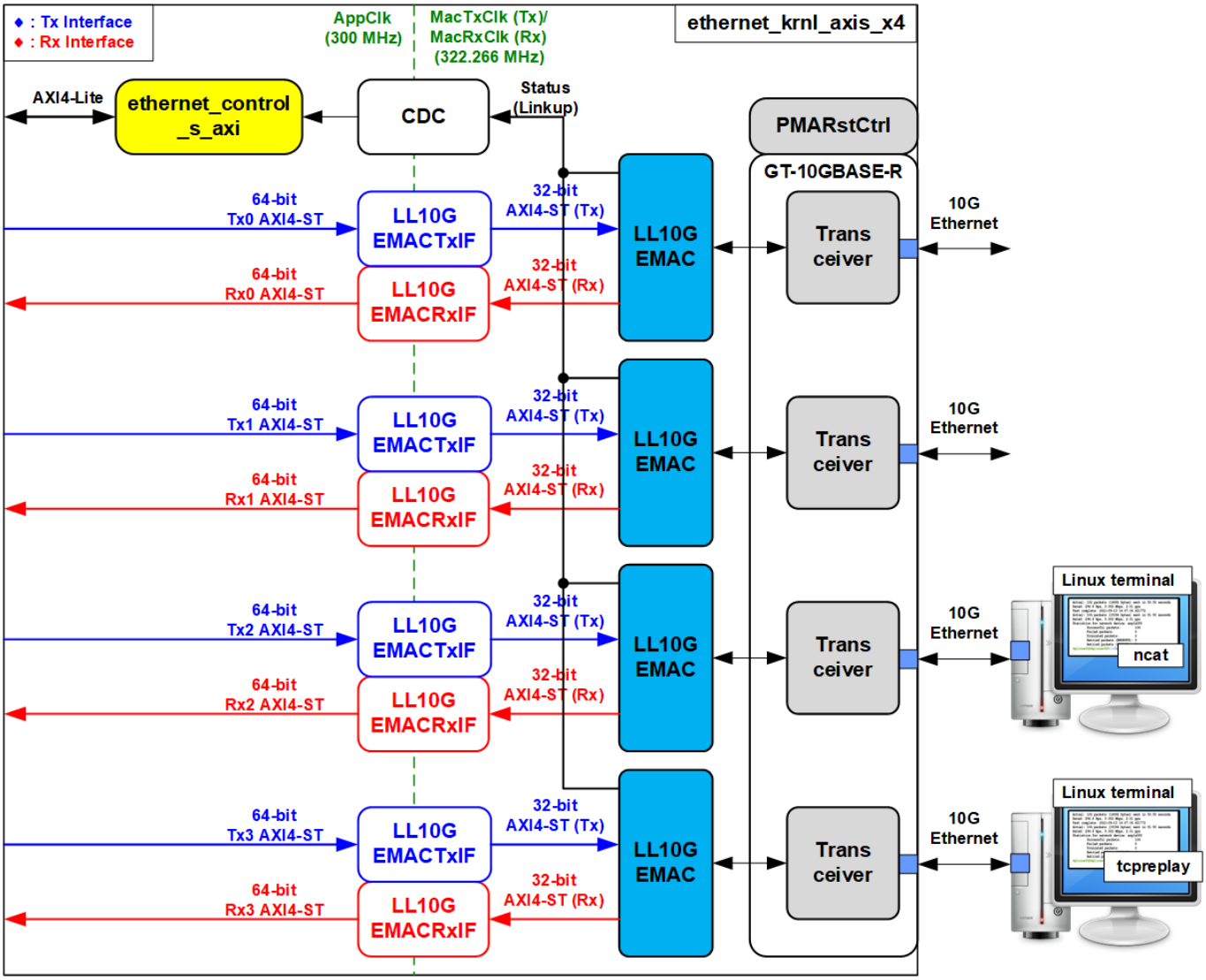

Figure 3-1 Ethernet kernel block diagram

The Ethernet kernel is modified from the Ethernet hardware kernel of the Accelerated Algorithmic Trading system by integrating LL10GEMAC-IP and modifying LL10GEMACTxIF/ LL10GEMACRxIF to reduce latency time. To achieve the Vitis kernel interface requirements, AXI4-Lite is applied to be the control and status interface while AXI4-ST is applied to be data interface. The Accelerator card uses a QSFP28 connector which can be split into four channels of 10G Ethernet for network connection. Thus, the Ethernet kernel integrates four sets of EMAC interfaces, LL10GEMAC-IP, and Transceiver (configured to 10GBASE-R PMA). The kernel interface must run in AppClk domain which can set the frequency by the tools. While the user interface of Transceiver for 10GBASE-R must be run in MacTxClk for Tx interface and MacRxClk for Rx interface. Therefore, CDC (Clock domain crossing) must be integrated to interface the signals that are run in different clock domains.

The Xilinx demo provides ethernet_control_s_axi module to convert AXI4-Lite interface to internal signals for write/read access. It is mapped to the control/status signals of the other modules. If the signals are run in different clock domains, CDC is applied. The status signal returned by LL10GEMAC is Linkup status of all Ethernet connections.

The data interface of Ethernet kernel has four interfaces but uses the same interface type, 64-bit AXI4-Stream standard interface. The transmitting and receiving AXI4-stream interfaces are connected to LL10GEMACTxIF and LL10GEMACRxIF, respectively. The LL10GEMACTxIF module is a data adapter from MacTxClk to AppClk and downsizes data width from 64 bits to 32 bits. The LL10GEMACRxIF module is a data adapter from AppClk to MacRxClk and upsizes data width from 32 bits to 64 bits. The LL10GEMACTxIF and LL10GEMACRxIF are connected with 10G Ethernet MAC controller (LL10GEMAC-IP) using a 32-bit AXI4 stream interface (AXI4-ST) in Tx interface and Rx interface, respectively.

LL10GEMAC-IP, provided by Design Gateway, implements the Ethernet MAC layer and PCS layer with low latency time. The Tx interface and Rx interface of AXI4-ST are run in different clock domains, MacTxClk and MacRxClk respectively. LL10GEMAC-IP needs to run with Xilinx Transceiver which is configured to be PMA module for 10GBASE-R interface. PMARstCtrl is designed to control reset sequence of Xilinx Transceiver.

To run AAT demo, at least two 10G Ethernet connections are applied. The first connection is connected with the PC that runs “tcpreplay” command to transfer market data via UDP/IP protocol on Linux OS. The second connection is connected with the PC that runs “ncat” command to listen TCP port for trading order via TCP/IP protocol on Linux OS.

The ethernet_control_s_axi module, the peripheral of this kernel, the overview of the AAT system, the host-software shell, and the test procedure of the AAT system can be requested from Xilinx AAT Site.
https://www.xilinx.com/applications/data-center/financial-technology/accelerated-algorithmic-trading.html

3.1 Xilinx Transceiver (PMA for 10GBASE-R)

The PMA IP core for 10G Ethernet (BASE-R) can be generated using Vivado IP catalog. In the 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

Four channels are enabled in the AAT demo for four Ethernet connections.
An example of the Transceiver wizard in Ultrascale model is described in the following link.
https://www.xilinx.com/products/intellectual-property/ultrascale_transceivers_wizard.html

3.2 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

3.3 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 1b for one clock cycle.
2) Wail until Tx reset done, output from the transceiver, is asserted to 1b.
3) Finish 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 1b.
6) Finish Rx reset sequence and de-assert Rx reset, user interface output, to allow the user logic beginning Rx operation.

3.4 LL10GEMACTxIF

This module is an AXI4-stream data adapter that interfaces with transmitting path of LL10EMAC. It performs AXI4-stream crossing clock domains and 64-to-32-bit data width conversion. This module is designed for low-latency performance. However, there are some limitations for using this module listed below.

- The MacTxClk frequency over UserClk frequency ratio must be less than two.
- If UserTxValid is dropped during transmission, the latency increases. The error is asserted to EMAC to cancel packet transmission, and then the logic starts packet retransmission.

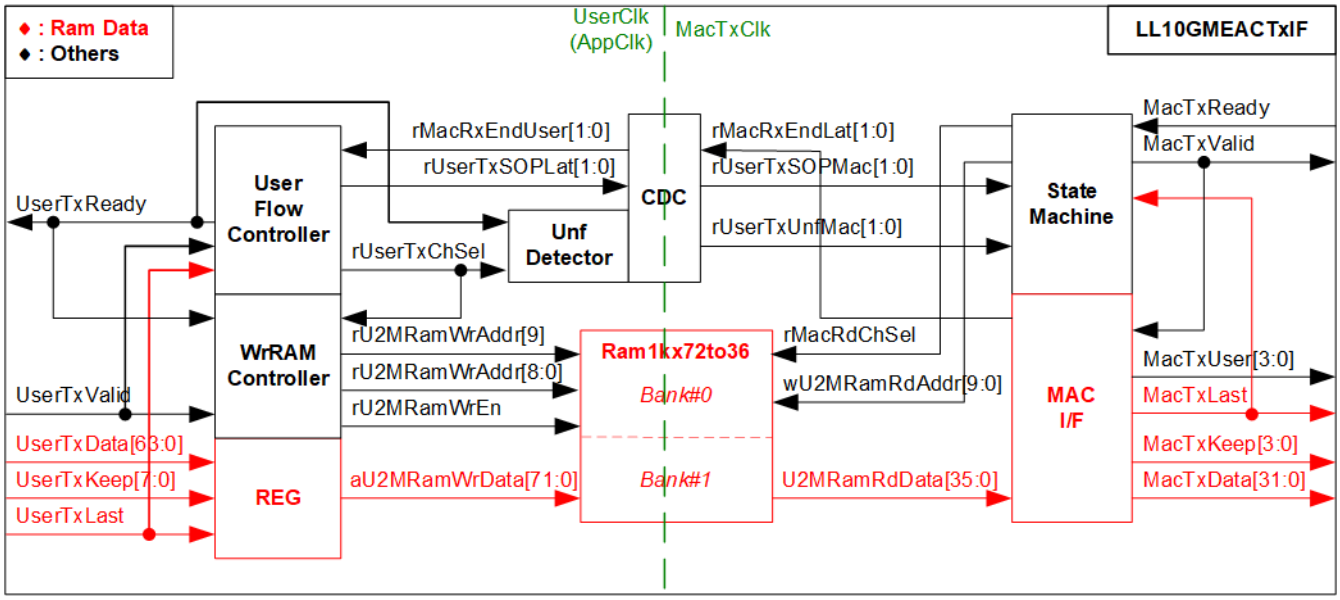

Figure 3-2 LL10GEMACTxIF Logic diagram

As shown in Figure 3-2, Ram1k72to36 stores data stream from User interface to EMAC. The Write controller (the logic to write RAM) runs on UserClk while the Read controller (the logic to read RAM) runs on MacTxClk. Also, CDC (clock domain crossing) is included to transfer flow control signals across clock domain, i.e., Start-of-packet on User I/F, Underflow flag on User I/F, and End-of-packet on Mac I/F.

Ram1kx72to36

Ram1kx72to36 is block memory that is configured for simple dual-port block RAM type with asymmetric data width feature. This memory stores 72-bit data width in UserClk domain and crosses to 36-bit data width in MacTxClk domain. 36-bit data consists of 32-bit data, 2-bit encoded keep, 1-bit last flag, and 1-bit reserved signal. Ram1kx72to36 module is divided into two banks to store AXI4-Stream data from user. The first bank (ch#0) has a memory address at 0 to 511 and the second bank (ch#1) has a memory address at 512 to 1023. One bank is designed to store one packet, so the maximum packet length is equal bank size, 4096 bytes (512 x 64-bit). The packet length can be extended by extending RAM depth.

Write controller

The User flow controller asserts UserTxReady to accept the new packet transmission when the next RAM bank is free. The WrRAM Controller generates Write address to store one packet data to the same bank during transferring packet to RAM. After transferring final data of a packet, MSB of the address is inverted to switch the active bank. Besides, User flow controller also asserts Start-of-frame flag (rUserTxSOPLat) when the first data of the new packet is received. This signal is forwarded to the Read controller via CDC to start packet transmission. After the Read controller transfers the final data of packet from each bank, End-of-frame flag (rMacRxEndLat) will be asserted. User flow control uses End-of-frame flag to free the RAM.

There is the limitation of EMAC that a packet cannot pause transmission before transferring the final data. Therefore, data in RAM must be always ready for reading and the Write controller needs to write data to RAM without pausing until end of packet. Unf detector is designed to assert Underflow flag (rUserTxUnfLat) when UserTxValid is de-asserted to 0b before end of packet. This signal is also fed to the Read controller via CDC to cancel the packet transmission.

Read controller

The State machine controls data flow to forward the read data from RAM to EMAC. MacTxValid is asserted to 1b when State machine detects Start-of-frame flag of the new packet. The read address is created by State machine to read RAM data which consists of 32-bit data, keep data, and last flag. MAC I/F includes the decoder to convert 2-bit encoded keep signal to be 4-bit signal. If Underflow flag is asserted before transmitting the final data, MacTxLast and MacTxUser will be asserted to cancel the current packet transmission. After that, State machine starts packet retransmission by reading the first data of the same bank. When the final data is transmitted successfully, State machine will switch the RAM bank for the next packet transmission.

Figure 3-3 and Figure 3-4 display the timing diagram of the Write Controller, illustrating both normal and underflow conditions. The following information describes the timing diagram.

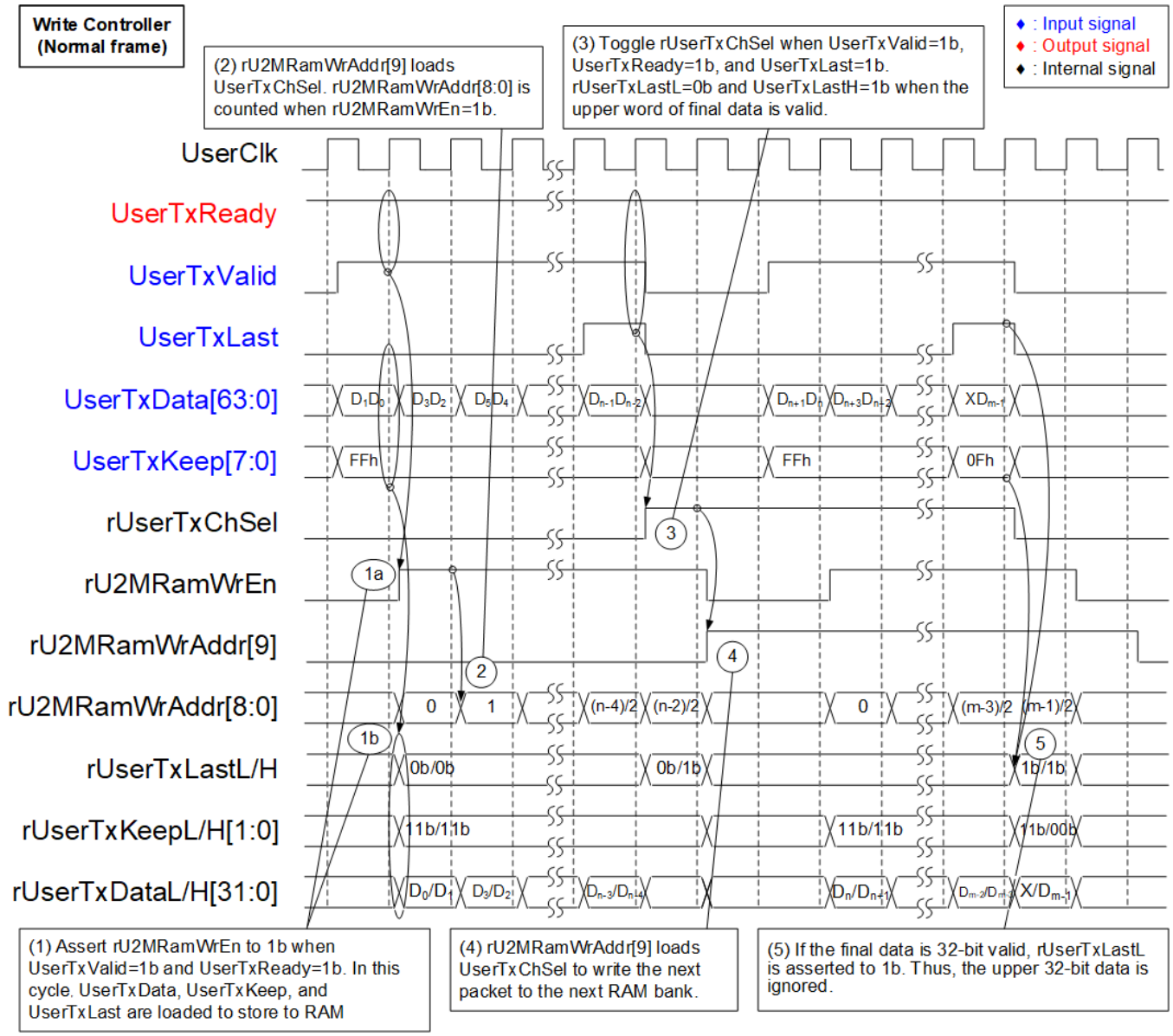

Figure 3-3 Write controller of LL10GEMACTxIF Timing diagram in normal condition

1) The Write Controller accepts a 64-bit AXI4-Stream data input by setting UserTxReady to 1b when UserTxValid=1b. After that, the write enable signal (rU2MRamWrEn) is asserted to 1b. At this point, the User data (UserTxData), User byte enable (UserTxKeep), and User last flag (UserTxLast) are loaded into rUserTxDataL/H, rUserTxKeepL/H, and rUserTxLastL/H, respectively. To reduce the data width of RAM, the 8-bit UserTxKeep is encoded as a 2-bit value (UserTxKeepL/H) using the following mapping.

   01h maps to 00b/00b, 03h maps to 00b/01b, 07h maps to 00b/10b, 0Fh maps to 00b/11b, 1Fh maps to 00b/11b, 3Fh maps to 01b/11b, 7Fh maps to 10b/11b, FFh maps to 11b/11b.

   The value 00b,01b,10b, or 11b represent UserTxKeepL/H depending on the validity of the lower/upper 32-bit data for 0-1 byte, 2 bytes, 3 bytes, or 4 bytes. The 72-bit write data to RAM is assigned as follows.

   Bit[31:0]: rUserTxDataL, Bit[33:32]: rUserTxKeepL, Bit[34]: rUserTxLastL, Bit[35]:RSV, Bit[67:36]: rUserTxDataH, Bit[69:68]: rUserTxKeepH, Bit[70]: rUserTxLastH, Bit[71]:RSV.

   MSB of the Write RAM address (rU2MRamWrAddr[9]) selects the active RAM bank. This value is loaded from rUserTxChSel, indicating the active bank. The remaining bits of the Write RAM address are reset to 0 to store the first data at the first address of the active bank.

2) rU2MRamWrAddr[8:0] increments when the write enable signal (rU2MRamWrEn) is asserted to 1b, allowing the next data to be stored at the next RAM address.
3) When the final data of a packet is received (UserTxValid=1b, UserTxLast=1b, and UserTxReady=1b), the bank indicator (rUserTxChSel) toggles to store the next packet in the next bank. To store the final data in RAM, rUserTxLastH is always set to 1b when UserTxLast=1b. The value of rUserTxLastL depends on the UserTxLast and UserTxKeep. If the upper word is valid (UserTxKeep[7:4] is not equal to 0), rUserTxLastL is set to 0b to indicate that the final data is placed in the upper 32-bit data.
4) The MSB of the Write RAM address (rU2MRamWrAddr[9]) updates its value to write data to the next bank. Subsequently, the next packet is stored to the next bank until the end of the packet is reached.
5) If the upper 32-bit data is not valid (UserTxKeep[7:4]=0b), rUserTxLastL is asserted to 1b, indicating that the final data is positioned on the lower 32-bit data.

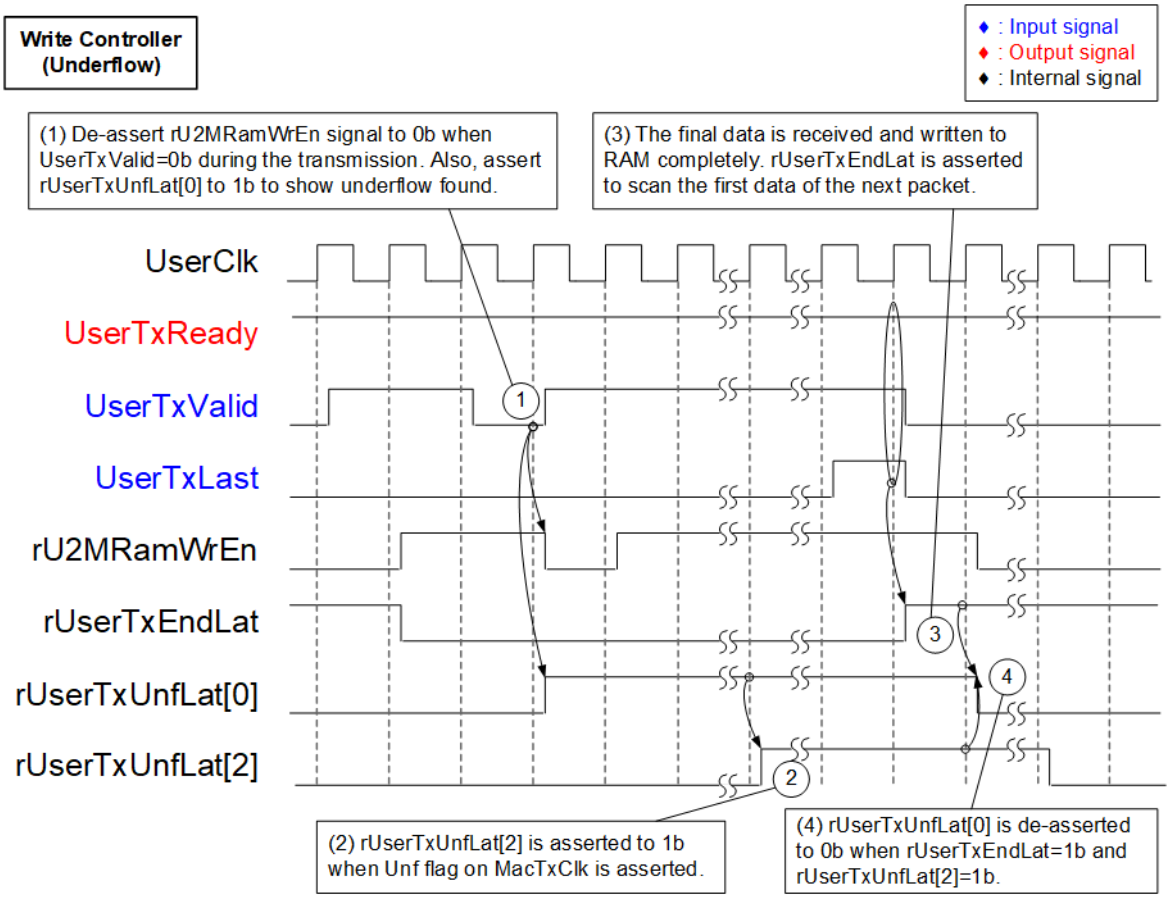

Figure 3-4 Write controller of LL10GEMACTxIF Timing diagram in underflow condition

LL10GEMACTxIF triggers the Underflow flag, an internal signal used to cancel ongoing packet transmission when UserTxValid is set to 0b before the completion of the packet. In such cases, the packet needs to be retransmitted. Figure 3-4 illustrates the process of activating the Underflow flag in the Write controller and forwarding it to the Read controller to halt the packet transmission. Further details about the retransmission process will be provided later.

1) If UserTxValid is set to 0b before the final data (UserTxEndLat=0b) is sent, rUserTxUnfLat[0] is asserted to 1b. Similar to the normal condition, the write enable signal (rU2MRamWrEn) is asserted to 1b to store the User data when it is accepted (UserTxValid=1b and UserTxReady=1b). Consequently, all data in the packet is completely stored in RAM, regardless of UserTxValid being de-asserted.
2) The Underflow flag (rUserTxUnfLat[0]) is forwarded through a Clock Domain Crossing (CDC) to assert the Underflow flag in MacTxClk domain. Subsequently, the Read controller cancels the ongoing packet transmission and initiates the retransmission. rUserTxUnfLat[2] is asserted to 1b when the Read controller cancels the packet transmission.
3) After receiving the final data of the packet (UserTxLast=1b, UserTxValid=1b, and UserTxReady=1b), rUserTxEndLat is asserted to 1b.
4) The Underflow flag (rUserTxUnfLat[0]) is de-asserted to 0b when the final data is received (rUserTxEndLat=1b), and the Read controller cancels the packet transmission (rUserTxUnfLat[2]=1b).

Figure 3-5 presents the timing diagram of the State Machine, which represents the Read Controller operating under normal conditions. Here are the details explaining the timing diagram.

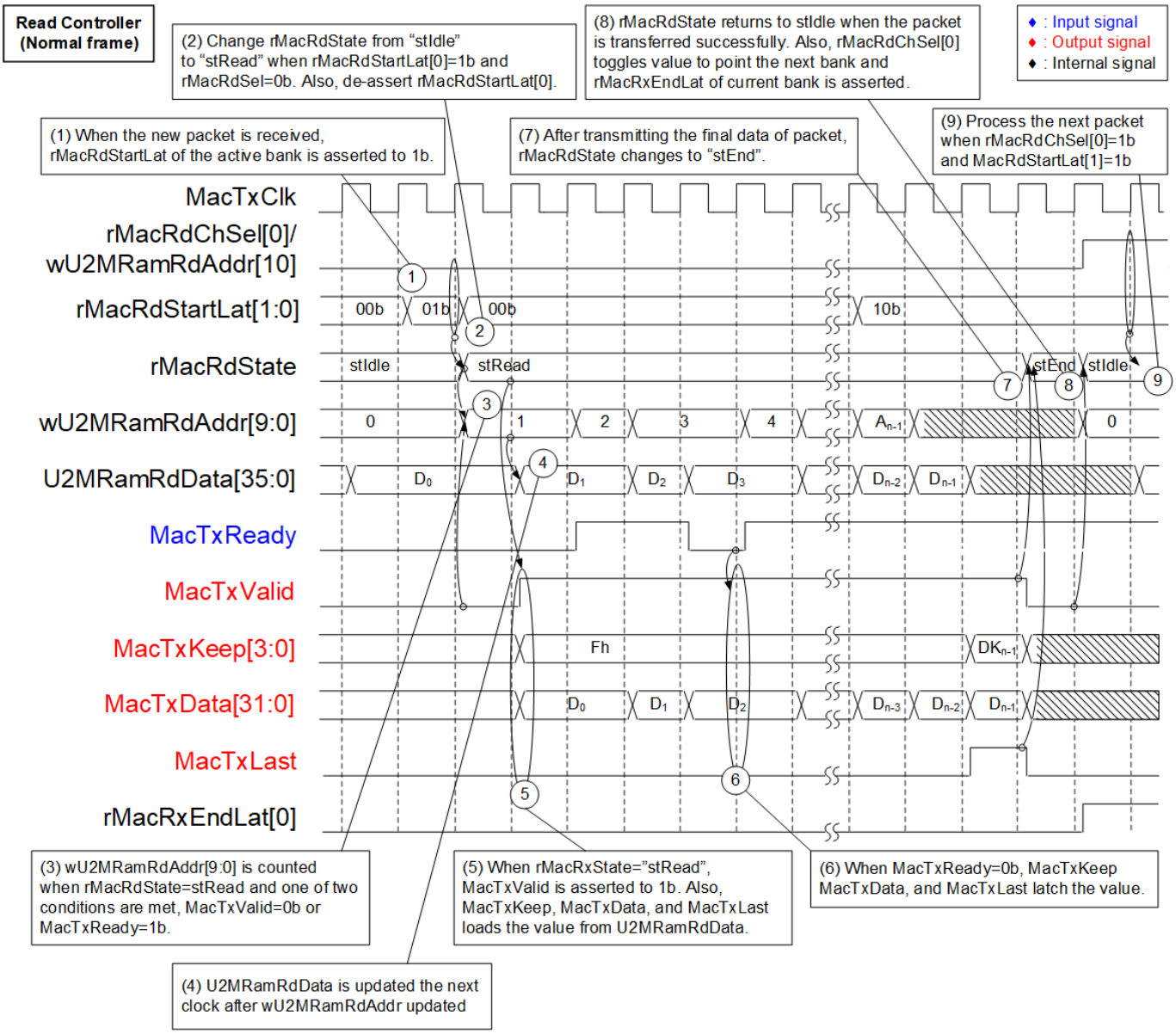

Figure 3-5 Read controller of LL10GEMACTxIF Timing diagram in normal condition

1) The Start flag (rMacRdStartLat) is asserted to 1b by the Write controller and then fed to the Read controller through a CDC. This flag signifies the arrival of a new packet, prompting the Read controller to initiate processing.
2) If the start flag of the currently active bank is asserted (rMacRdStartLat[0]=1b when rMacRdChSel[0]=0b), the state transitions from Idle state (“stIdle”) to Read state (“stRead”).
   Note: rMacRdChSel[0] indicates the currently active bank (0b-Bank#0 and 1b-Bank#1).
3) The read address, wU2MRamRdAddr[9:0], is always initialized to 0. It is only incremented when the main state is in Read state (rMacRdState=stRead), and either it is the first time loading data (MacTxValid=0b) or the EMAC is receiving AXI4-Stream output (MacTxReady= 1b). This signal is designed using combinational logic, so the value is updated at the same cycle when MacTxValid=0b or MacRxReady=1b. The MSB of the read address (wUMRamRdAddr[10]) matches the channel indicator (rMacRdChSel), which selects one of two memory areas.
4) The output of the read data from RAM, U2MRamRdData, is updated after the read address (wUMRamRdAddr[10:0]) is modified.
5) When the main state (rMacRdState) is in Read state (“stRead”), the MacTxValid signal is asserted to 1b. Simultaneously, the 36-bit read data from RAM (U2MRamRdData) is loaded as the output to the MAC interface.
   Bit[31:0]: MacTxData, Bit[33:32]: Decoded as MacTxKeep, Bit[34]: MacTxLast
   The decoding equation for MacTxKeep is as follows.
   00b maps to 0001b, 01b maps to 0011b, 10b maps to 0111b, and 11b maps to 1111b.
6) When the current data is not accepted by the EMAC (MacTxReady=0b and MacTxValid=1b), the values of MacTxKeep, MacTxData, and MacTxLast are latched.
7) After the final data is transmitted to the EMAC (MacTxValid=1b and MacTxLast=1b), the state transitions to “stEnd”.
8) In “stEnd”, the state waits until MacTxValid is de-asserted to 0b, confirming that the final data has been fully accepted. Afterward, the state changes back to “stIdle”, awaiting the next transfer to the next RAM bank. During this cycle, the active bank (rMacRdChSel[0]) toggles, and the End transfer flag (rMacRxEndLat) is asserted. The End flag is fed back to the Write controller to release the RAM for storing newly received packets from the user.
9) The process returns to step (1), waiting for rMacRdStartLat of the active bank to be asserted. Subsequently, the active bank is switched to the new bank (rMacRdChSel[0]=1b). This ensures that the Read controller is prepared to handle the next packet transmission in the updated bank.

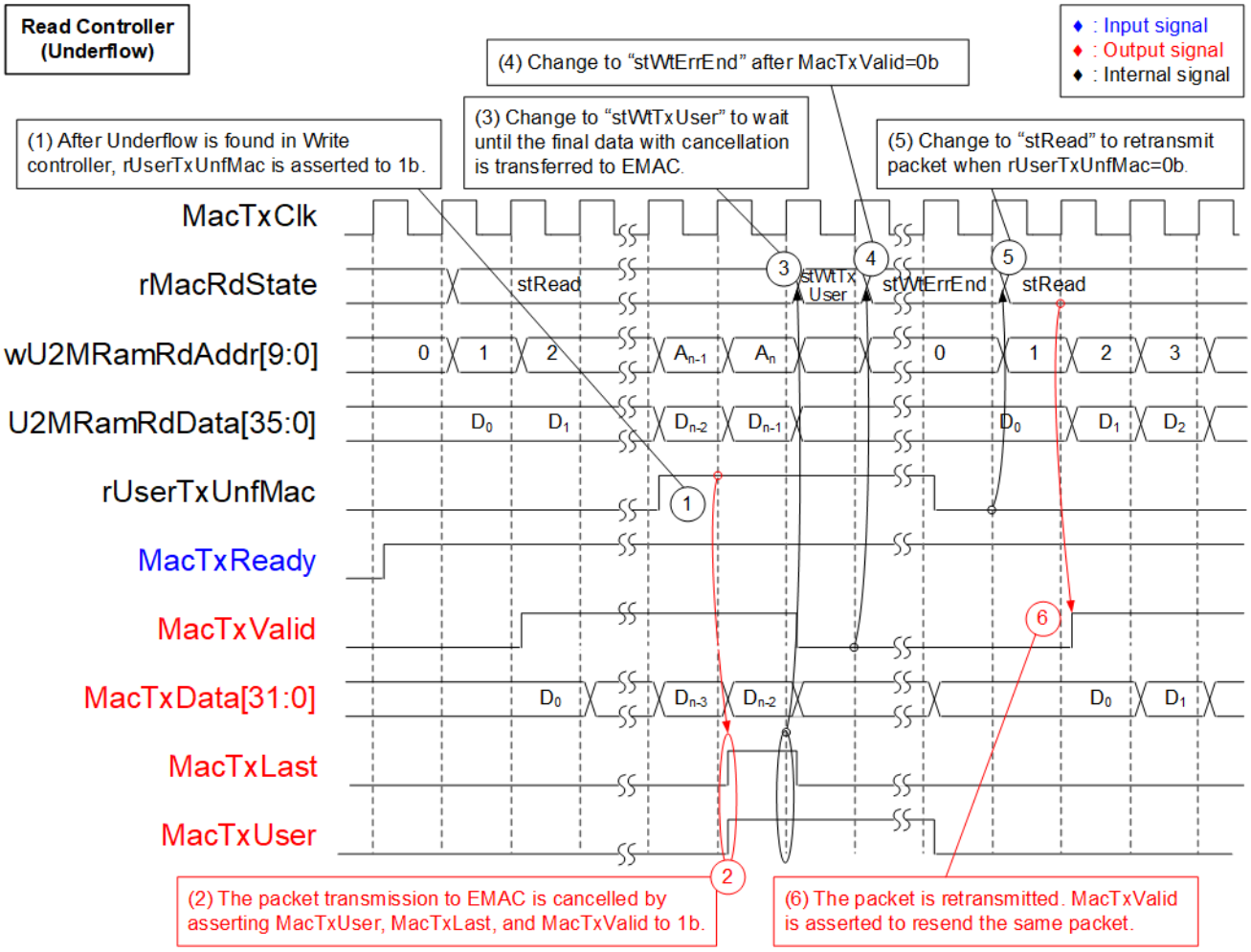

Figure 3-6 Read controller of LL10GEMACTxIF Timing diagram in underflow condition

1) When the Write controller detects that the user de-asserts UserTxValid to 0b before transmitting the final data of the packet, the Underflow flag in the MacTxClk domain (rUserTxUnfMac) is asserted to 1b.
2) If the rUserTxUnfMac flag is asserted to 1b while the main state is "stRead", the packet transmission of the packet to the EMAC is halted. This is achieved by asserting MacTxValid, MacTxLast, and MacTxUser to 1b. When MacTxUser is asserted at the end of the packet, the EMAC concludes the packet transmission with an error condition directed at the target. Consequently, the target will disregard this packet.
3) The state machine transitions to "stWtTxUser", where it awaits the successful transmission of the final data to the EMAC.
4) The state machine verifies the successful status by detecting the de-assertion of MacTxValid to 0b. Subsequently, the state changes to "stWtErrEnd".
5) The state machine remains in "stWtErrEnd" until the underflow flag (rUserTxUnfMac) is de-asserted to 0b. Following this, the state transitions back to "stRead" to initiate packet retransmission.
6) MacTxValid is asserted to 1b, and the same packet is retransmitted. The first data (D0) is once again transmitted on MacTxData.

3.5 LL10GEMACRxIF

The LL10GEMACRxIF module serves as an AXI4-Stream data adapter that interfaces with the receiving path of LL10GEMAC. It performs clock domain crossing for AXI4-stream and converts the data width from 32-bit to 64-bit. The module consists of several components, including the AXI4-ST 32bto64b Converter, Error Detection, AsyncFWFT32x69, and AXI4-ST Read Controller.

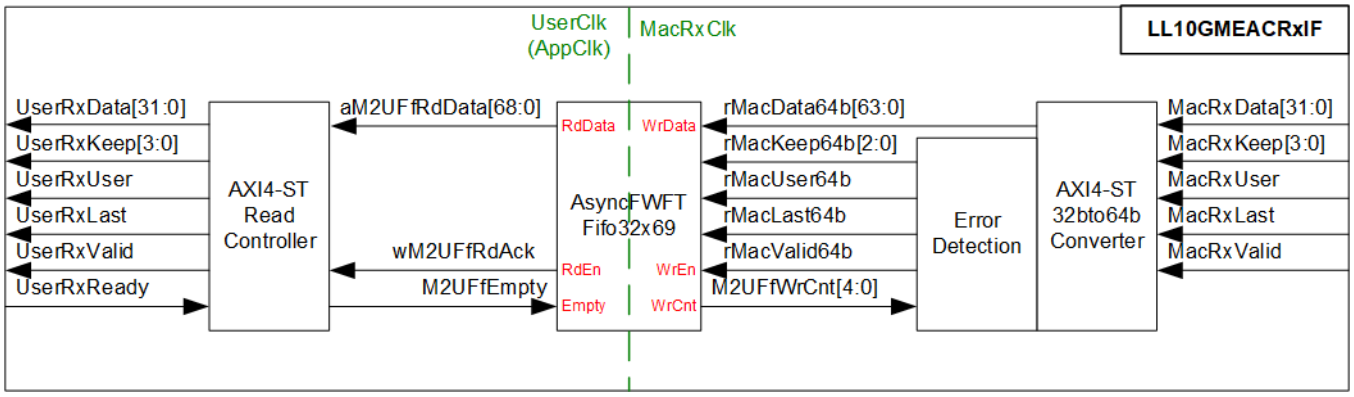

Figure 3-7 LL10GEMACRxIF Logic diagram

The AXI4-ST 32bto64b Converter is responsible for converting 32-bit data to 64-bit data and writing it into the AsyncFWFT32x69 module. Error Detection is incorporated to cancel the write operation under two conditions. Firstly, if the size of the received packet size exceeds the threshold value of 9014 bytes. Secondly, if the available space in the FIFO drops below the threshold value of 8 data. When an error is detected, Error Detection asserts rMacUser64b, rMacLast64b, and rMacValid64b to 1b, thereby marking the completion of the current packet transmission with an error status.

The AsyncFWFT32x69 module functions as an asynchronous FWFT FIFO, storing data in the MacRxClk domain and facilitating the transition to the UserClk domain. It can store 69-bit data, comprising of 64-bit data, a 3-bit keep signal, a 1-bit error flag, and a 1-bit last flag. The 3-bit keep signal encodes the validity of each byte of data, representing 1-8 bytes. The AXI4-Stream Read Controller reads data from the AsyncFWFT32x69 and forwards it as AXI4-Stream data to the user. Data is read from the FIFO only when it is not empty, and the user must be ready to receive the data or when the transmitted data is the first data of a packet.

Figure 3-8 illustrates the timing diagram for writing to the FIFO inside LL10GEMACRxIF. The following details provide a description of the timing diagram.

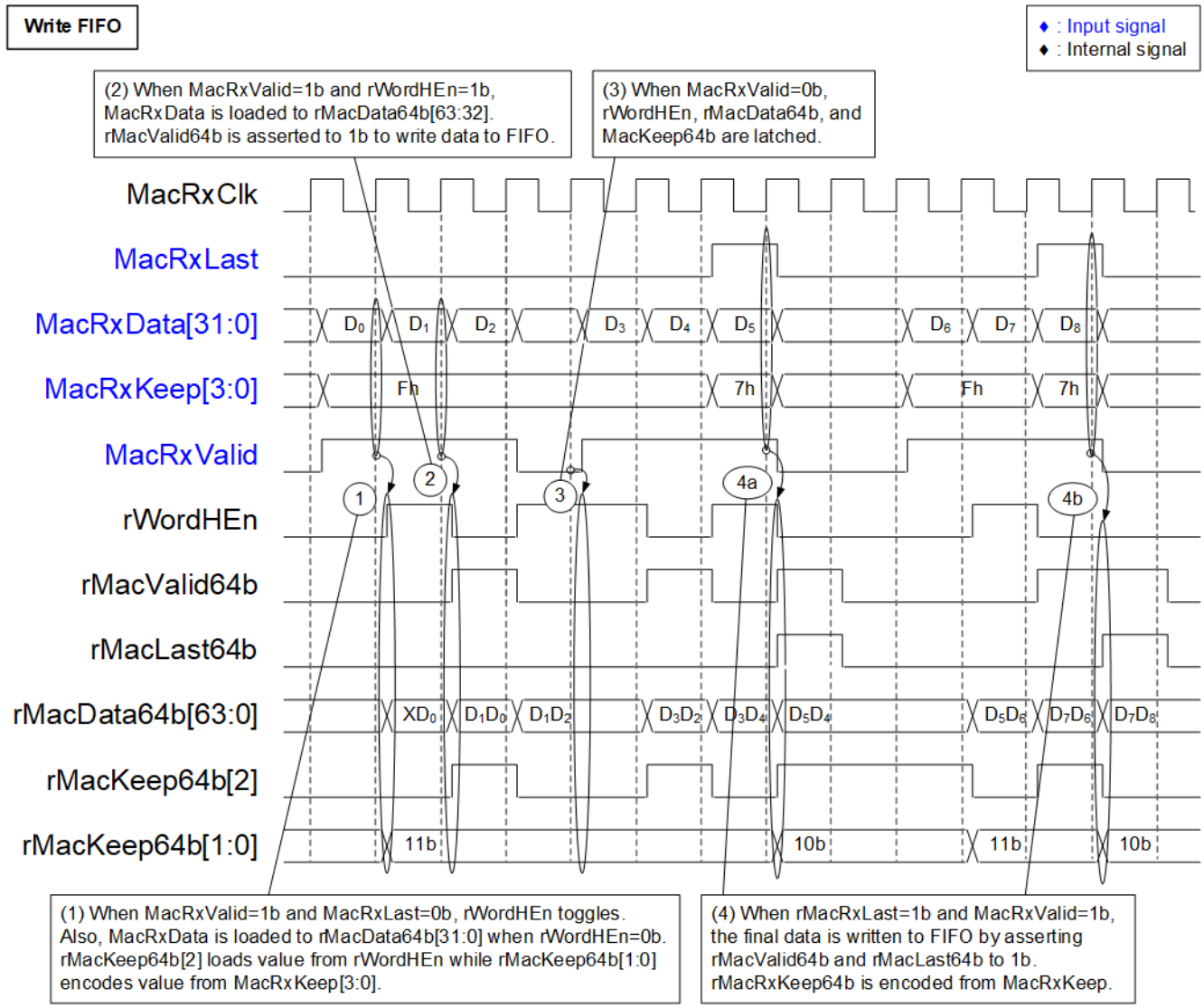

Figure 3-8 Write FIFO logic of LL10GEMACRxIF Timing diagram

1) The rWordHEn signal is initially set to 0b. It is asserted to 1b when receiving the 32-bit lower data (MacRxValid=1b and rWordHEn=0b). Conversely, it is de-asserted to 0b to receive the 32-bit upper data (MacRxValid=1b and rWordHEn=1b).

   The 8-bit keep signal, MacRxKeep[3:0], which indicates byte enable, is encoded into a 3-bit register, rMacKeep64b[2:0]. To accommodate the 32-bit data, MacRxKeep[3:0] is mapped to rMacKeep64b[1:0] using the following table: 0001b->00b, 0011b->01b, 0111b->10b, 1111b->11b. The value of rMacKeep64b[2] is determined by loading rWordHEn=1b, indicating the validity of the 32-bit upper data.

   The 32-bit upper data (rMacData64b[64:32]) and the 32-bit lower data (rMacData64b[31:0]) are loaded from MacRxData[31:0] based on the control of rWordHEn. The lower word is loaded when rWordHEn=0b, while the upper word is loaded when rWordHEn=1b.

2) Simultaneously with the loading of the 32-bit upper data (rMacData64b[64:32]), the 64-bit data (rMacData64b), 3-bit keep (rMacKeep64b), 1-bit error flag (rMacUser64b), and 1-bit last flag (rMacLast64b) are written to the FIFO by asserting rMacValid64b to 1b.
3) When the EMAC pauses data transmission by de-asserting MacRxValid to 0b, the values of rWordHEn, rMacData64b, and rMacKeep64b are latched.
4) When the end of the packet is transferred (MacRxValid=1b and MacRxLast=1b), MacRxKeep[3:0] may not be all ones. In Figure 3-8, MacRxKeep[3:0]=0111b, so rMacKeep64b[1:0] is encoded as 10b. Additionally, rMacLast64b is asserted to 1b. There are two different conditions for storing the final data.
   a) MacRxLast=1b, MacRxValid=1b, and rWordHEn=1b. In this condition, the upper data (rMacData64b[63:32]) is valid, and rMacKeep64b[2] is asserted to 1b.
   b) MacRxLast=1b, MacRxValid=1b, and rWordHEn=0b. In this condition, only the lower data (rMacData64b[31:0]) is valid, rMacKeep64b[2] is de-asserted to 0b, and rWordHEn remains 0b after receiving the final data.

Figure 3-9 presents the timing diagram of the AXI4-Stream Read Controller, and the following details provide a description of the timing diagram.

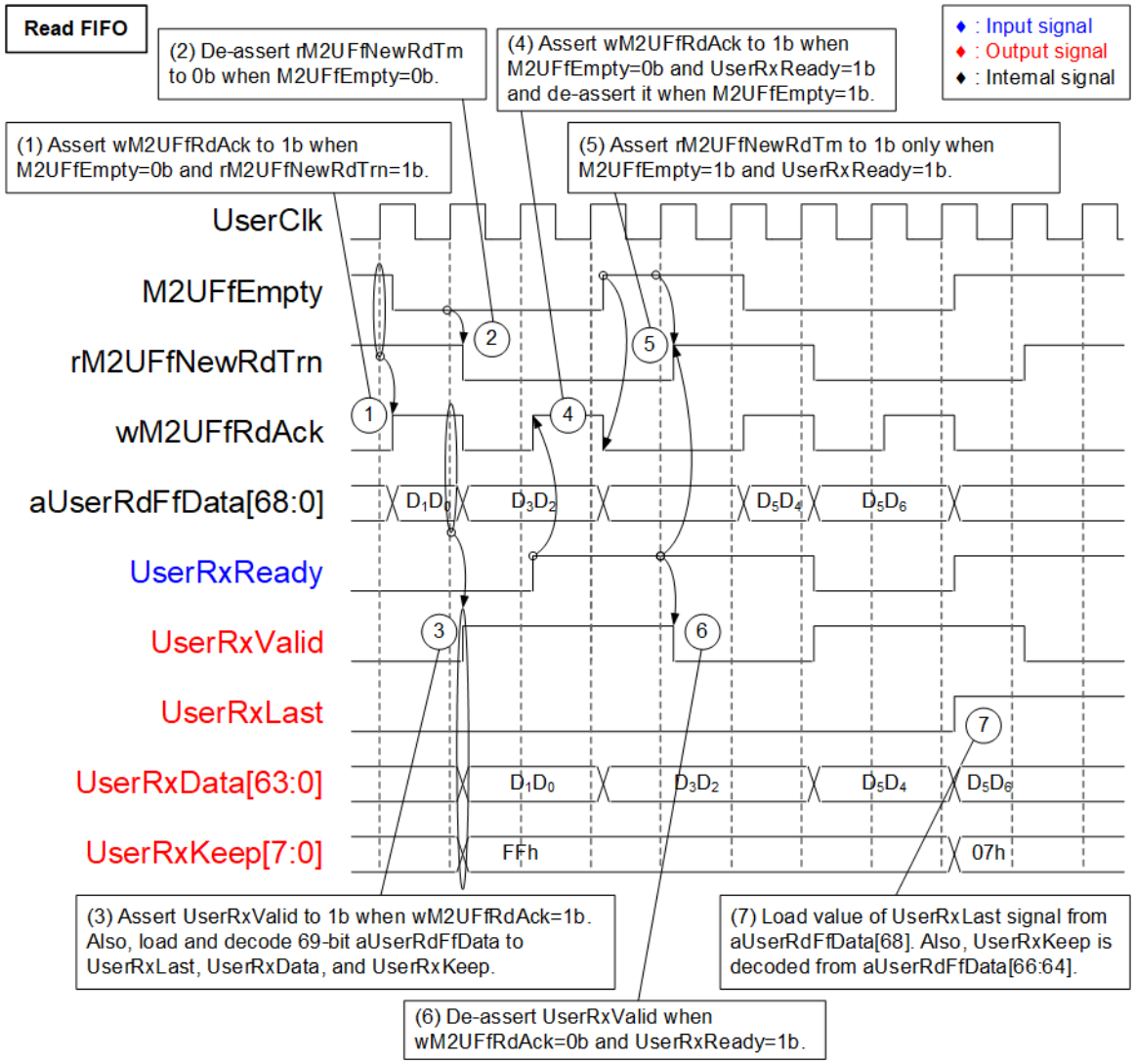

Figure 3-9 LL10GEMACRxIF Read Controller Timing diagram

1) Once the data is written to the FIFO, M2UFfEmpty is de-asserted to 0b. Since the FIFO is the FWFT type, the read FIFO data (aUserRdFfData) becomes valid when the FIFO read enable (wM2UFfRdAck) is asserted. wM2UFfRdAck is immediately set to 1b when the FIFO is not empty (M2UFfEmpty=0b) and there is no remaining latched data in the Read Controller (rM2UFfNewRdTrn=1b). Subsequently, the data is transferred to the user.
2) rM2UFfNewRdTrn is de-asserted to pause reading data from the FIFO when there is remaining latch data, which is checked by M2UFfEmpty=0b. After that, wM2UFfRdAck is controlled by UserRxReady.
3) Once the read FIFO enable signal (wM2UFfRdAck) is asserted to 1b, the AXI4-Stream valid output (UserRxValid) is also set to 1b. Furthermore, the AXI4-Stream data (UserRxData[63:0]), last flag (UserRxLast), and keep signal (UserRxKeep[7:0]) are loaded from aUserRdFfData[63:0], aUserRdFfData[68], and aUserRdFfData[66:64] respectively. The keep signal decoding table is as follows.
   000b -> 01h, 001b -> 03h, 010b -> 07h, 011b -> 0Fh,
   100b -> 1Fh, 101b -> 3Fh, 110b -> 7Fh, and 111b -> FFh.
4) When rM2UFfNewRdTrn=0b and M2UFfEmpty=0b, wM2UFfRdAck is controlled by UserRxReady. It is asserted to 1b to read the next data when UserRxReady is also asserted to 1b to accept the current data.
5) Once the latched data is completely transferred to the user (UserRxReady=1b) and there is no remaining data in the FIFO (M2UFfEmpty=1b), rM2UFfNewRdTrn is re-asserted to 1b. The status now resembles step (1), waiting for new received data in the FIFO.
6) At the same time, UserRxValid is de-asserted to 0b, pausing the data transmission.
7) The last flag (UserRxLast) is asserted to 1b when aUserRdFfData[68] is equal to 1b. When sending the last data, UserRxKeep may be equal to 01h, 03h, 07h, …, FFh, indicating the valid bytes of the final data.

4 Software

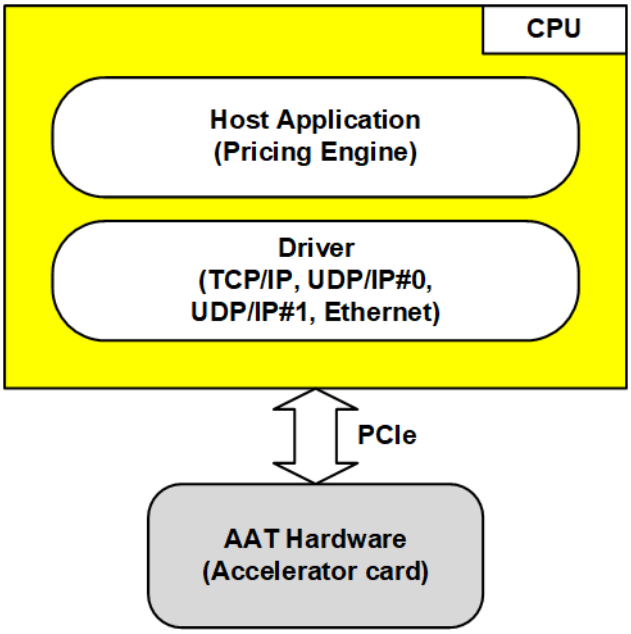

Figure 4-1 Software on AAT demo

The AAT demo provides software source code that can be compiled and run on a processor system integrated with an Accelerator card. In this reference design, no modification has been made to the AAT software. Therefore, all the source code and documentation can be requested from the Xilinx AAT Site.

The application aims to demonstrate the hardware status while processing market data. Each hardware kernel has its own status, which can be viewed on the console. For instance, the orderbook kernel produces decoded Bid/Ask price from the received market data, as illustrated in Figure 4-2.

```
>> orderbook readdata
+-----------------------------------------------------------------------------+
|        Symbol Index =  10          ||    Timestamp = 0x0067015FB0F5C700     |
+------------------------------------++-----------------------------------+
|                BID                 ||                ASK                |
+-----------+-----------+------------++-----------+-----------+-----------+
|   Count   |  Quantity |    Price   ||   Price   |  Quantity |   Count   |
+-----------+-----------+------------++-----------+-----------+-----------+
|         9 |       830 |   10000.00 ||  10200.00 |       140 |         5 |
|         8 |        50 |    9900.00 ||  10250.00 |       770 |         3 |
|         6 |       930 |    9800.00 ||  10300.00 |       790 |         7 |
|         4 |       860 |    9700.00 ||  10350.00 |       910 |         1 |
|         6 |       190 |    9600.00 ||  10400.00 |       120 |         6 |
+-----------+-----------+------------++-----------+-----------+-----------+

>>
```

Figure 4-2 The example of Orderbook status

5 Revision History

Revision	Date	Description
1.1	26-May-23	Add table of contents
1.0	7-Dec-21	Initial version release