Xilinx Answer 50166
LogiCORE IP Serial RapidIO Gen2 – Debugging and Packet Analysis Guide

Important Note: This downloadable PDF of an answer record is provided to enhance its usability and readability. It is important to note that answer records are Web-based content that are frequently updated as new information becomes available. You are reminded to visit the Xilinx Technical Support Website and review (Xilinx Answer 50166) for the latest version of this solution.

Introduction

This document describes techniques for debugging issues related to the LogiCORE Serial RapidIO Gen2 Core. When debugging SRIO issues, you should have a clear understanding on the flow of packets through different interfaces of your design. You should be able to identify what types of packets are seen on those interfaces and correctly decode them by checking the contents of the packets. This document provides a detailed description of tracking SRIO packets (Control Symbols and Data Characters) on different interfaces in the core.

The main objective of this document is to help you debug your design by going into the low level details of the core. The packet and signal analysis have been described based on the simulation of the example design provided along with the generation of the core in the CORE Generator tool. However, the same concept also applies when debugging issues in hardware using the ChipScope tool.

Example Design Architecture

Figure 1 shows the entire hierarchy of the example design simulation setup. It comprises both testbench and user core with the backend user application. The srio_sim.v top level testbench connects srio_example_top_primary which represents the user core, and srio_example_top_mirror which acts as the link partner.

The srio_example_top.v instantiates all components of the core and the example design that are needed to implement the design in hardware. This includes the following modules:

- Clock and Reset (srio_clk.v / srio_rst.v)
- Configuration Fabric (cfg_fabric.v)
- Stimulus Generators to create transactions (srio_request_gen.v)

The srio_quick_start module which is instantiated in srio_example_top, connects to the Maintenance Port and generates maintenance transactions. This module sends a fixed set of instructions to the configuration register space, both local and remote. The user can add, modify, or remove transactions by editing the maintenance_list.v file. The srio_request_gen module, which is instantiated within the srio_example_top, is where I/O (and message) transactions are generated.

The transaction types to be generated are passed down through parameters so that only transactions supported on the connected port are produced. This file also stores the expected response for any response-generating-requests that it creates, and compares received responses to the expected value. The srio_response_gen module, also instantiated in srio_example_top, generates responses to requests received from the link partner.

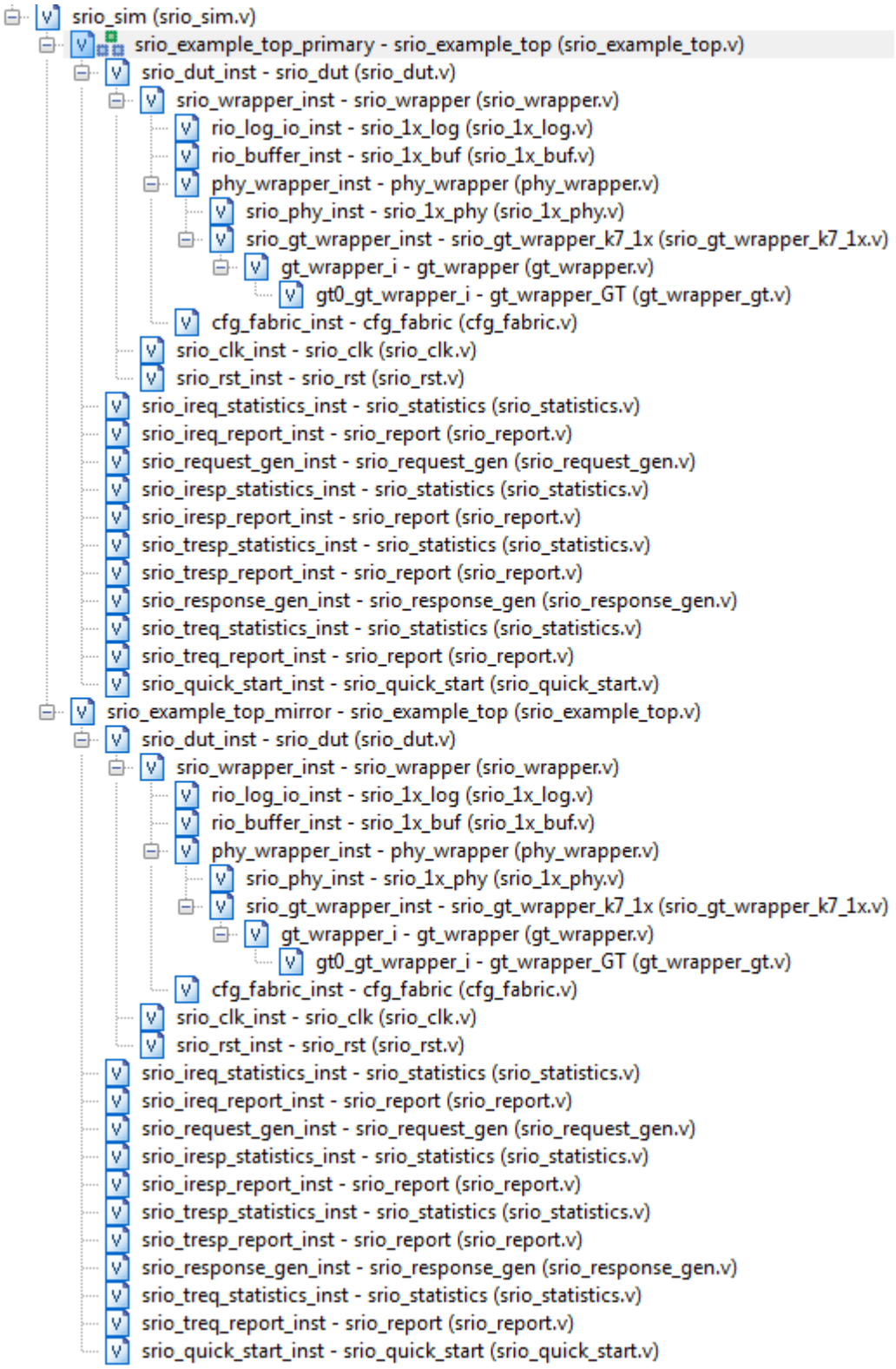

Figure 1 - Example Design Simulation Modules Hierarchy

Figure 2 shows the block diagram of the module hierarchy in Figure 1.

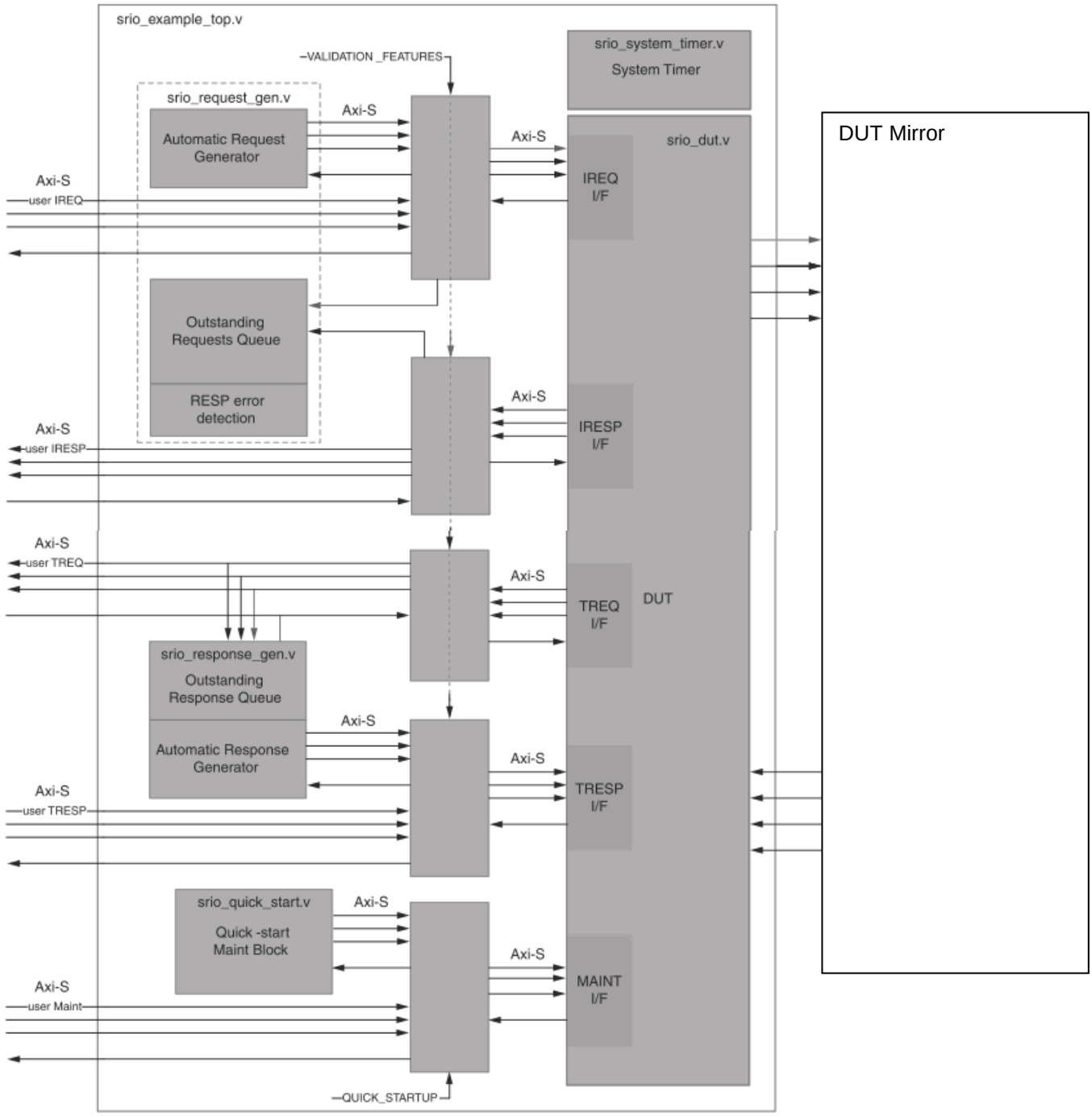

Figure 2 - Example Design Simulation Top Level Architecture

The Xilinx RapidIO Endpoint solution (srio_dut) is comprised of the following components:

- Serial RapidIO Physical Layer core (PHY)
- RapidIO Logical (I/O) and Transport Layer core (LOG)
- Serial RapidIO Buffer core (BUF)
- Reference designs to handle clocking, resets, and configuration accesses

Figure 3 shows a basic block diagram of different components and interfaces between those components in srio_dut.v.

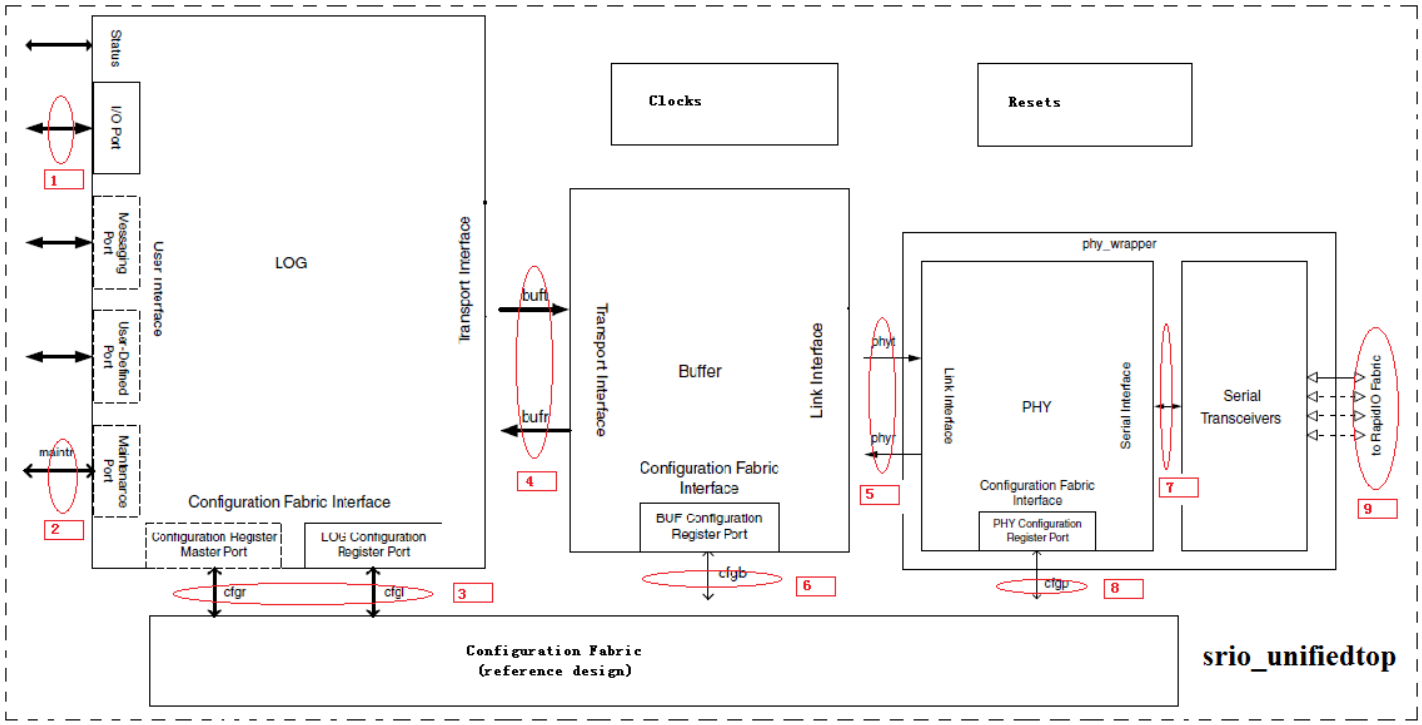

Figure 3 – srio_dut.v Components and its Connecting Interfaces

Example Design Interfaces

This section lists different interfaces you could check to track an incoming or an outgoing packet for debugging your SRIO design. As shown in Figure 3, each interface has been numbered to make it easier to reference. A list of signals in each interface is also given below.

Initiator/Target IO Port (Interface-1)

The following signals in this interface are generated when I/O Style is selected as Initiator/Target during the core configuration. The other interface is Condensed I/O style. In this document, the packet analysis has been described for Initiator/Target Style only.

s_axis_ireq_tvalid
s_axis_ireq_tready
s_axis_ireq_tdata[63:0]
s_axis_ireq_tkeep[7:0]
s_axis_ireq_tlast
s_axis_ireq_tuser[31:0]
m_axis_iresp_tvalid
m_axis_iresp_tready
m_axis_iresp_tdata[63:0]
m_axis_iresp_tkeep[7:0]
m_axis_iresp_tlast
m_axis_iresp_tuser[31:0]
m_axis_treq_tvalid
m_axis_treq_tready

m_axis_treq_tdata[63:0]
m_axis_treq_tkeep[7:0]
m_axis_treq_tlast
m_axis_treq_tuser[31:0]
s_axis_tresp_tvalid
s_axis_tresp_tready
s_axis_tresp_tdata[63:0]
s_axis_tresp_tkeep[7:0]
s_axis_tresp_tlast
s_axis_tresp_tuser[31:0]

Maintenance Port (Interface-2)

s_axi_maintr_rst
s_axi_maintr_awvalid
s_axi_maintr_awready
s_axi_maintr_awaddr[31:0]
s_axi_maintr_wvalid
s_axi_maintr_wready
s_axi_maintr_wdata[31:0]
s_axi_maintr_bvalid
s_axi_maintr_bready
s_axi_maintr_bresp[1:0]
s_axi_maintr_arvalid
s_axi_maintr_arready
s_axi_maintr_araddr[31:0]
s_axi_maintr_rvalid
s_axi_maintr_rready
s_axi_maintr_rresp[1:0]
s_axi_maintr_rdat[31:0]

LOG Configuration Fabric (Interface-3)

s_axi_cfgl_awvalid
s_axi_cfgl_awready
s_axi_cfgl_awaddr[23:0]
s_axi_cfgl_wvalid
s_axi_cfgl_wready
s_axi_cfgl_wdata[31:0]
s_axi_cfgl_wstrb[3:0]
s_axi_cfgl_bvalid
s_axi_cfgl_bready
s_axi_cfgl_arvalid
s_axi_cfgl_arready
s_axi_cfgl_araddr[23:0]
s_axi_cfgl_rvalid
s_axi_cfgl_rready
s_axi_cfgl_rdata[31:0]
m_axi_cfgr_awvalid
m_axi_cfgr_awready
m_axi_cfgr_awaddr[23:0]
m_axi_cfgr_wvalid
m_axi_cfgr_wready
m_axi_cfgr_wdata[31:0]
m_axi_cfgr_wstrb[3:0]
m_axi_cfgr_bvalid
m_axi_cfgr_bready

m_axi_cfgr_bresp[1:0]
m_axi_cfgr_arvalid
m_axi_cfgr_arready
m_axi_cfgr_araddr[23:0]
m_axi_cfgr_arprot[2:0]
m_axi_cfgr_rvalid
m_axi_cfgr_rready
m_axi_cfgr_rresp[1:0]
m_axi_cfgr_rdata[31:0]

LOG Transport Interface (Interface-4)

m_axis_buft_tvalid (s_axis_buft_tvalid – BUF Transport)
m_axis_buft_tready (s_axis_buft_tready)
m_axis_buft_tdata[63:0] (s_axis_buft_tdata[63:0])
m_axis_buft_tkeep[7:0] (s_axis_buft_tkeep[7:0])
m_axis_buft_tlast (s_axis_buft_tlast)
m_axis_buft_tuser[7:0] (s_axis_buft_tuser[7:0])
s_axis_bufr_tvalid (m_axis_bufr_tvalid)
s_axis_bufr_tready (m_axis_bufr_tready)
s_axis_bufr_tdata[63:0] (m_axis_bufr_tdata[63:0])
s_axis_bufr_tkeep[7:0] (m_axis_bufr_tkeep[7:0])
s_axis_bufr_tlast (m_axis_bufr_tlast)
s_axis_bufr_tuser[7:0] (m_axis_bufr_tuser[7:0])
response_only
maint_only

BUF Link Interface (Interface-5)

m_axis_phyt_tvalid (s_axis_phyt_tvalid – PHY Link)
m_axis_phyt_tready (s_axis_phyt_tready)
m_axis_phyt_tdata[63:0] (s_axis_phyt_tdata[63:0])
m_axis_phyt_tkeep[7:0] (s_axis_phyt_tkeep[7:0])
m_axis_phyt_tlast (s_axis_phyt_tlast)
m_axis_phyt_tuser[7:0] (s_axis_phyt_tuser[7:0])
s_axis_phyr_tvalid (m_axis_phyr_tvalid)
s_axis_phyr_tready (m_axis_phyr_tready)
s_axis_phyr_tdata[63:0] (m_axis_phyr_tdata[63:0])
s_axis_phyr_tkeep[7:0] (m_axis_phyr_tkeep[7:0])
s_axis_phyr_tlast (m_axis_phyr_tlast)
s_axis_phyr_tuser[7:0] (m_axis_phyr_tuser[7:0])
srio_host
master_enable
idle2_selected
phy_rewind
phy_next_fm
phy_last_ack
phy_rcvd_buf_stat
phy_buf_stat
tx_flow_control

BUF Configuration Fabric (Interface-6)

s_axi_bcfg_awvalid
s_axi_bcfg_awready
s_axi_bcfg_awaddr[23:0]
s_axi_bcfg_wvalid

s_axi_bcfg_wready
s_axi_bcfg_wdata[31:0]
s_axi_bcfg_wstrb[3:0]
s_axi_bcfg_bvalid
s_axi_bcfg_bready
s_axi_bcfg_arvalid
s_axi_bcfg_arready
s_axi_bcfg_araddr[23:0]
s_axi_bcfg_rvalid
s_axi_bcfg_rready
s_axi_bcfg_rdata[31:0]

PHY Serial Interface (Interface-7)

gttx_data[32*LW-1:0]
gttx_charisk[4*LW-1:0]
gttx_inhibit[LW-1:0]
gtrx_data[32*LW-1:0]
gtrx_charisk[4*LW-1:0]
gtrx_chariscomma[4*LW-1:0]
gtrx_disperr[4*LW-1:0]
gtrx_notintable[4*LW-1:0]
gtrx_chanbondseq[LW-1:0]
gtrx_chanisaligned[LW-1:0]
gtrx_chanbonden
gtrx_reset_req
gtrx_reset
gtrx_reset_done[LW-1:0]

PHY Configuration Fabric (Interface-8)

s_axi_cfgp_awvalid
s_axi_cfgp_awready
s_axi_cfgp_awaddr[23:0]
s_axi_cfgp_wvalid
s_axi_cfgp_wready
s_axi_cfgp_wdata[31:0]
s_axi_cfgp_wstrb[3:0]
s_axi_cfgp_bvalid
s_axi_cfgp_bready
s_axi_cfgp_avalid
s_axi_cfgp_arready
s_axi_cfgp_araddr[23:0]
s_axi_cfgp_arvalid
s_axi_cfgp_rready
s_axi_cfgp_rdata[31:0]

Transceiver Interface (Interface-9)

srio_rxnN
srio_rxpN
srio_txnN
srio_txpN

Interface Packet and Signal Analysis

Link Initialization and Control Symbols

When you are debugging link issues, the first thing you should do is check whether the SRIO top level wrapper signals are toggling correctly or not. Those signals include: reset signals, clk_lock, port_initialized, link_initialized, port_error, mode_1x, and port_decore_error. Figure 4 shows the list of signals you should look at in the SRIO top level wrapper.

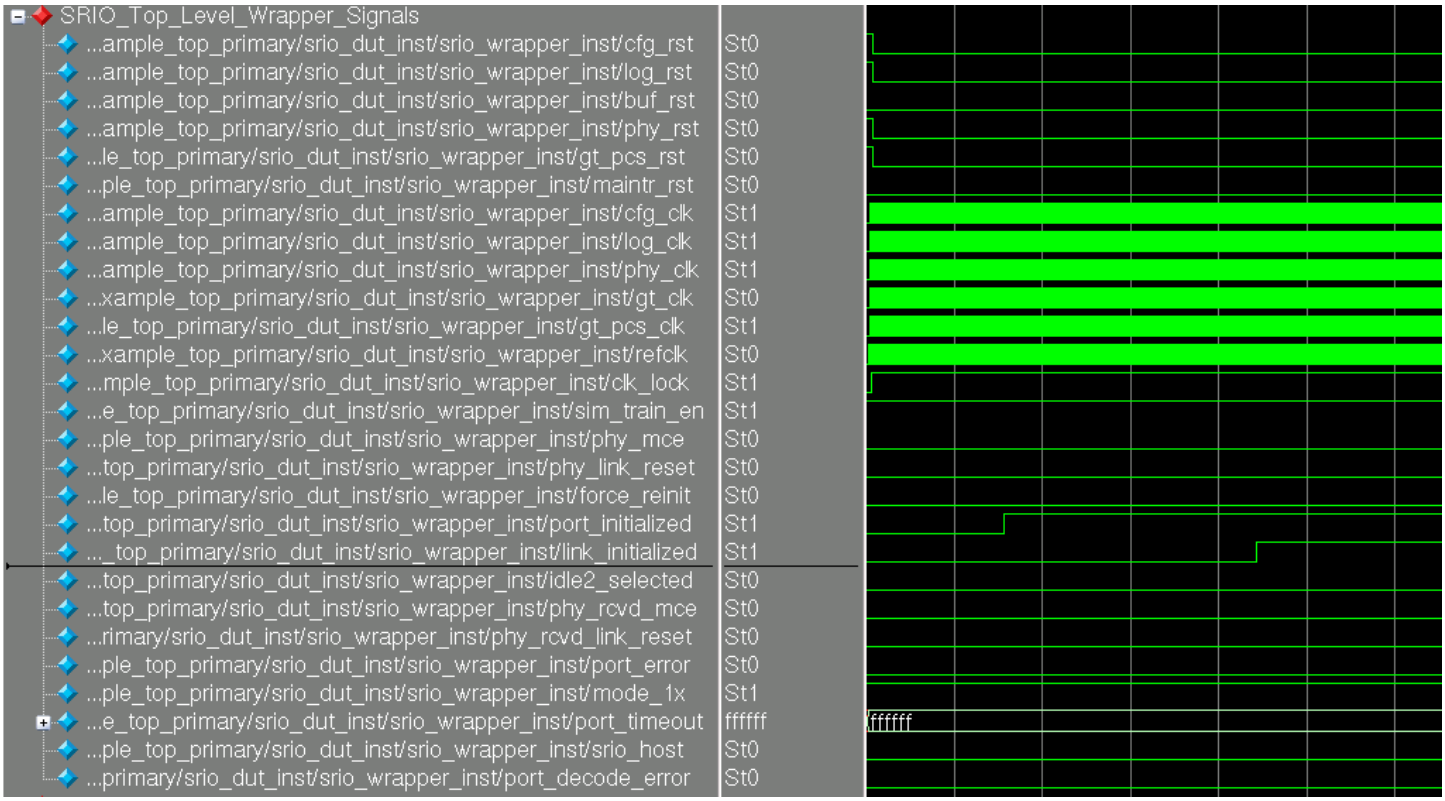

Figure 4 - SRIO Top Level Wrapper Signals

Figure 5 shows a close-up view of Figure 4 to illustrate the correct toggling of the reset signals in the core.

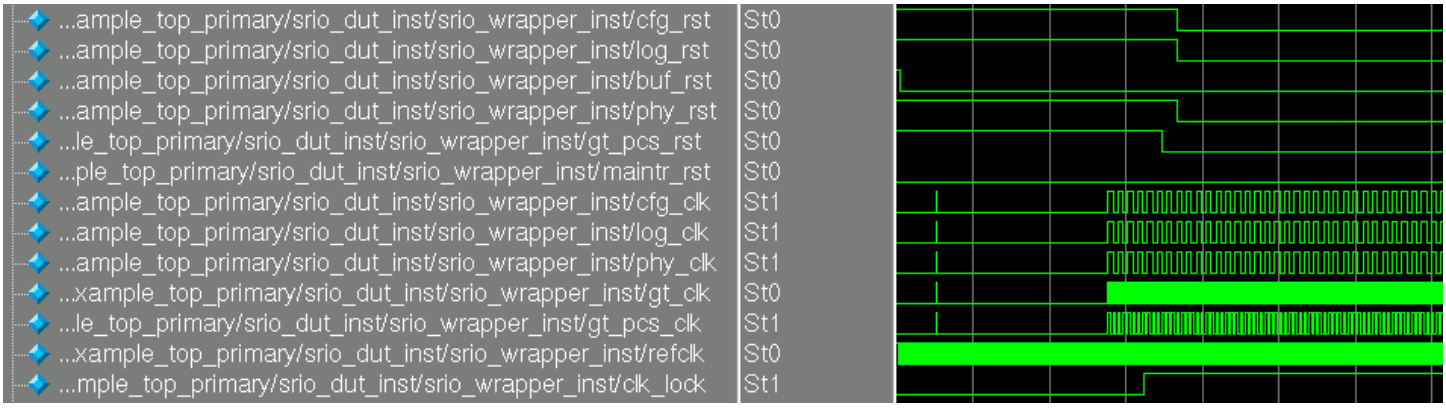

Figure 5 - Core Reset Signals

Figure 6 shows a capture of port_initalized and link_initalized signals. When port_initialized and link_initialized are both valid, the user can send/receive packets. link_initialized indicates that 7 consecutive error free control symbols have been received and 15 consecutive symbols have been sent. The red circles in Figure 6 are control symbols.

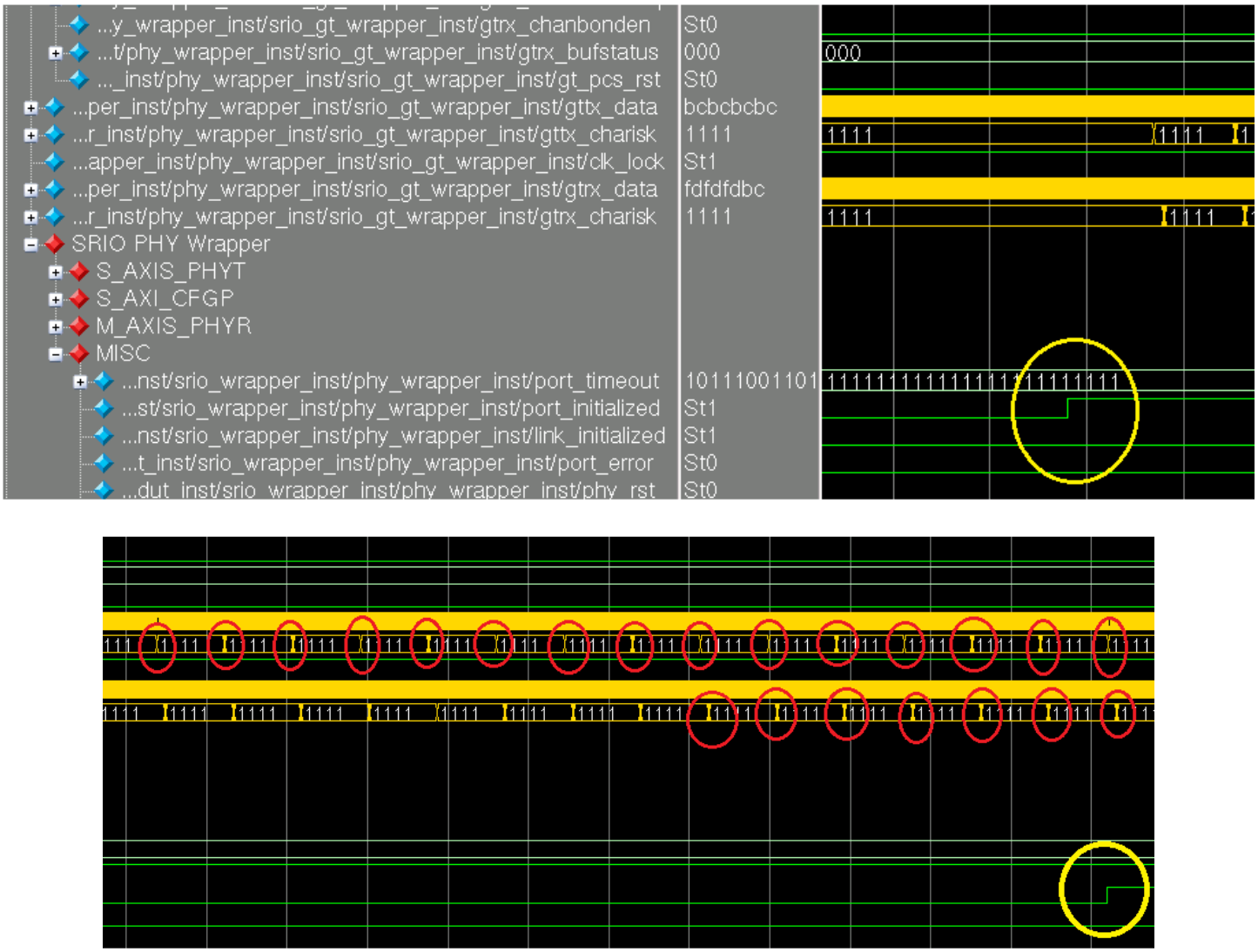

Figure 6 - Port_Initialized and Link_Initialized Signals (*broken into two parts for clarity*)

Figure 7 shows a close-up view of one of the red circles in Figure 6 depicting the transmission of a control symbol. In the red box, “1c” is K28.0 which indicates the start of the control symbol. “bc” is K28.5, /K/, IDLE for sync. The whole packet is decoded as follows:

80ff0f = 1000_0000_1111_1111_0000_1111

stype0=100 (Status)
ackID_status=00000
buf_status=11111
stype1=111 (NOP)
cmd=000
CRC=01111

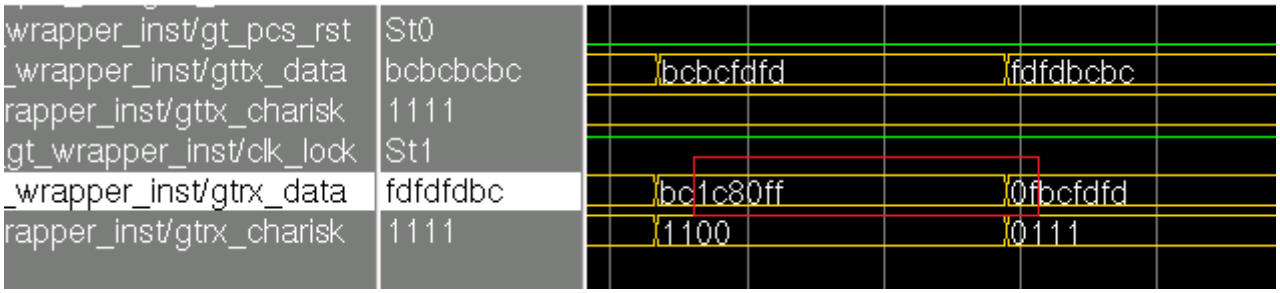

Figure 7 - Control Symbol Before Assertion of link_initialized Signal

If the initialization does not complete (i.e., port_initialized and link_initalized signals are not asserted), check GT_RXDATA to see whether IDLE (bcfd) ordered sets are being received or not, and also check that there are no coding errors. This can be done by looking at RXNOTINTABLE which should be '0'.

stype0=000 pkt acptd	packet_ackID 5	buf_status 5
stype0 =001 pkt retry	packet_ackID 5	buf_status 5
stype0 =010 pkt not acptd	packet_ackID 5	cause 5
stype0=100 status	ackID_status 5	buf_status 5
stype0=110 link resp.	ackID_status 5	port_status 5

stype1=000 start of pkt	cmd 000	CRC 5
stype1=001 stomp	cmd 000	CRC 5
stype1=010 end of pkt	cmd 000	CRC 5
stype1=011 restart retry	cmd 000	CRC 5
stype1=100 link request	cmd 3	CRC 5
stype1=101 multicast	cmd 000	CRC 5
stype1=111 NOP	cmd 000	CRC 5

Stype0

000	Packet Accepted
001	Packet Retry
010	Packet Not Accepted
100	Status
110	Link Response

cause

00001	Rcvd unexpeted ackID on pkt
00010	Rcvd a ctrl symbol w/ bad CRC
00011	Non-maintenance pkt reception is stopped
00100	Rcvd pkt w/ bad CRC
00101	Rcvd invld char. or vld but illegal char.
11111	General Error

port_status

00010	Error
00100	Retry-stopped
00101	Error-stopped
10000	OK

Stype1

000	Start-of-packet
001	Stomp
010	End-of-packet
011	Restart-from -retry
100	Link-request
101	Multicast-event
111	NOP

cmd

011	Reset-device
100	Input-status

Code Group Column Designation	Code Group Column Use	Number of Code-groups	Encoding
/PD/	Packet_Delimiator Conrol Symbol	1	/K28.3/
/SC/	Start_of_Control_Symbol	1	/K28.0/
/I/	Idle		
/K/	1x Sync	1	/K28.5/
/R/	1x Skip	1	/K29.7/
/A/	1x Align	1	/K27.7/
||I||	Idle Column		
||K||	4x Sync Column	4	/K28.5/K28.5/K28.5/K28.5/
||R||	4x Skip Column	4	/K29.7/K29.7/K29.7/K29.7/
||A||	4x Align Column	4	/K27.7/K27.7/K27.7/K27.7/

Figure 8 - Packet Format for Control Symbols

In general, you have to make sure all the signals at the GT interface are toggling correctly. Figure 9 shows signals that you should look into and capture in the ChipScope tool if you are debugging in hardware. If the signals are not toggling as shown, you should focus on the reason why that is happening and resolve that first before proceeding any further.

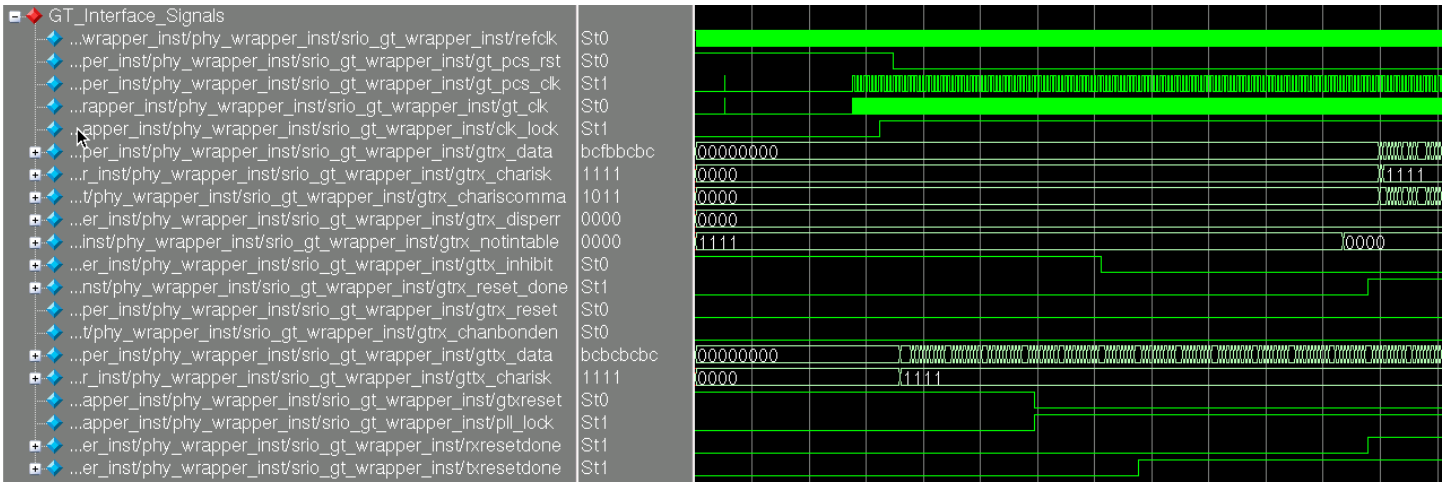

Figure 9 - GT Interface Signals

SWRITE Packet Analysis

Figure 10 shows an example of a SWRITE Packet sent to the IREQ Interface (Figure 3, Interface-1), the data on s_axis_ireq_tdata is “0060200000000777_0000000000000000”.

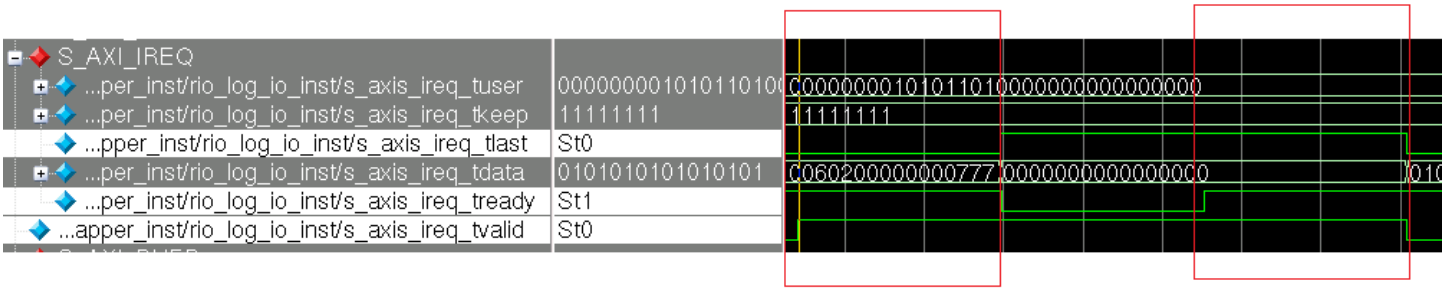

Figure 10 - SWRITE Packet Transmission on IREQ Interface

Figure 12 shows the HELLO FTYPE Packet Format. This format allows standardization of header field placement across packet types. It also segments the header and data into separate transfers on the interface. This leads to simpler control logic and allows data to be aligned to transfer boundaries for easier data management. According to the HELLO FTYPE Packet Format, the corresponding field interpretation for the above data in Figure 10 will be as follows:

0060200000000777 =

0000_0000_0110_0000_0010_0000_0000_0000_0000_0000_0000_0000_0111_0111_0111

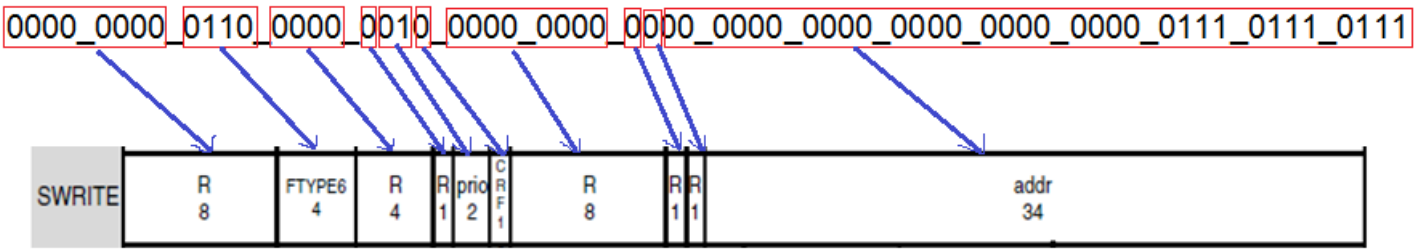

Figure 11 - SWRITE HELLO FTYPE Packet Format

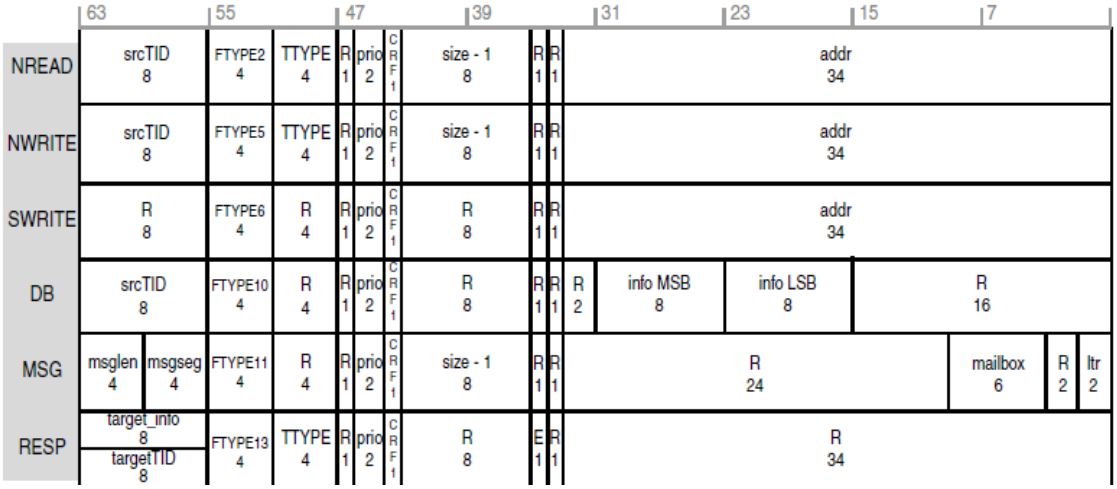

Figure 12 - HELLO FTYPE Packet Format

The data received on the IREQ interface of the SRIO core is captured by the SRIO LOG layer (Figure 3, Interface-1). The LOG layer adds transport layer information into the packet, sends it to BUF (Figure 3, Interface-4), and then BUF sends the same data to PHY later (Figure 3, Interface 5). The naming of the signals is based on the block diagram in Figure 14.

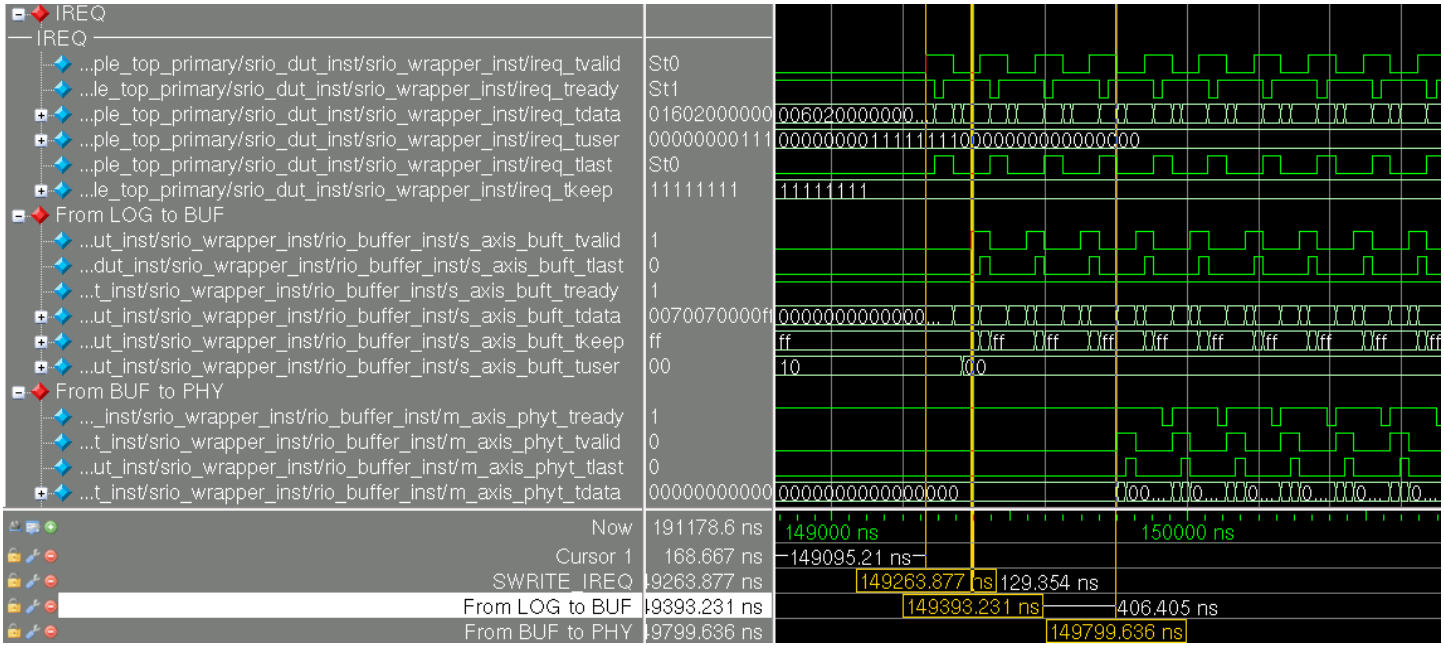

Figure 13 - SWRITE Packet Transmission at IREQ, LOG to BUF and BUF to PHY

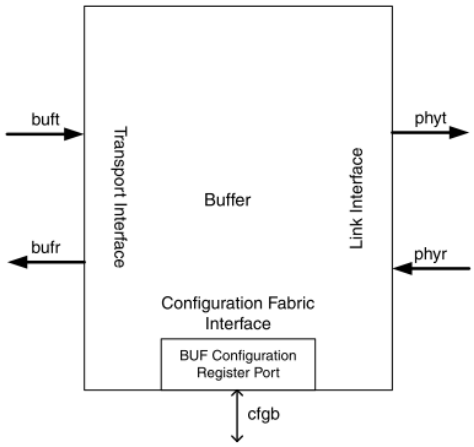

Figure 14 – Buffer Layer Interfaces

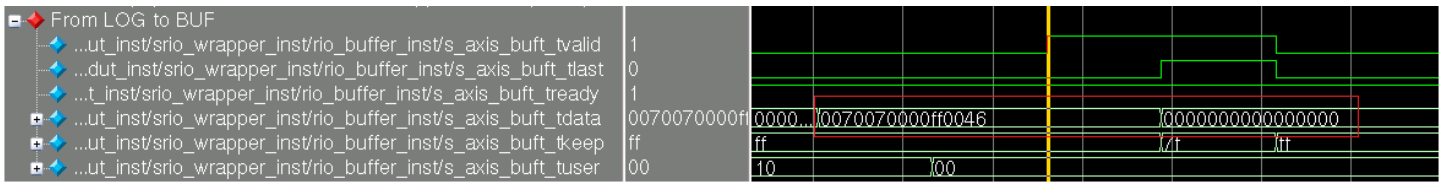

Figure 15 - SWRITE Packet from LOG to BUF

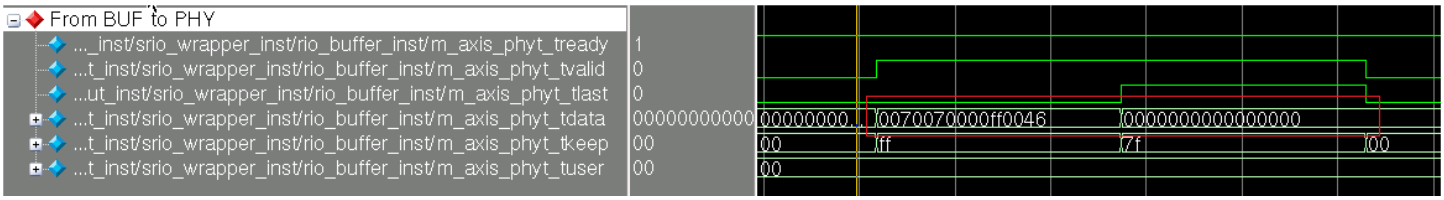

Figure 16 - SWRITE Packet from BUF to PHY

Figure 13 shows simulation capture of all three interfaces discussed. Figure 15 and Figure 16 show the same data "0070070000ff0046_0000000000000000" being passed from the Transport interface to Link interface of BUF.

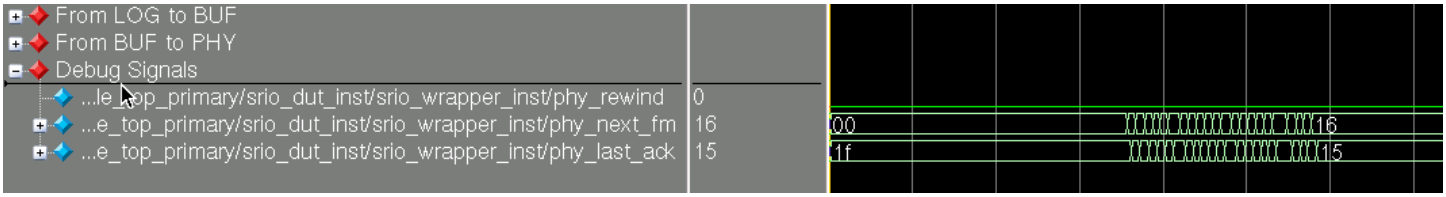

Figure 17 – Debug Signals

The signal shown in Figure 17 could be used for debugging SRIO designs.

- "phy_rewind" indicates that the PHY is requesting a rewind (e.g., the packet is not accepted by the link partner)
- "phy_next_fm" is used to communicate the next packet to send, indexed by ackID. This signal increments for each packet, except in a rewind scenario.
- "phy_last_ack" is used to communicate the ackID for the last received acknowledge control symbol. This frees up slots in the buffer by allowing the buffer to clear the corresponding packet. If this signal increments by more than one, multiple packet can be cleared. In Figure 17, "phy_last_ack" has not been increased as this is the first packet and SRIO has not received the ACK yet.

The BUF layer transmits the packet coming from the LOG layer to the PHY layer. The PHY layer adds PHY layer information and sends the packet to the GT. Figure 18 shows SWRITE packet traveling from IREQ to GT interface. The four cursors in the simulation waveform show the start of the packet in each interface.

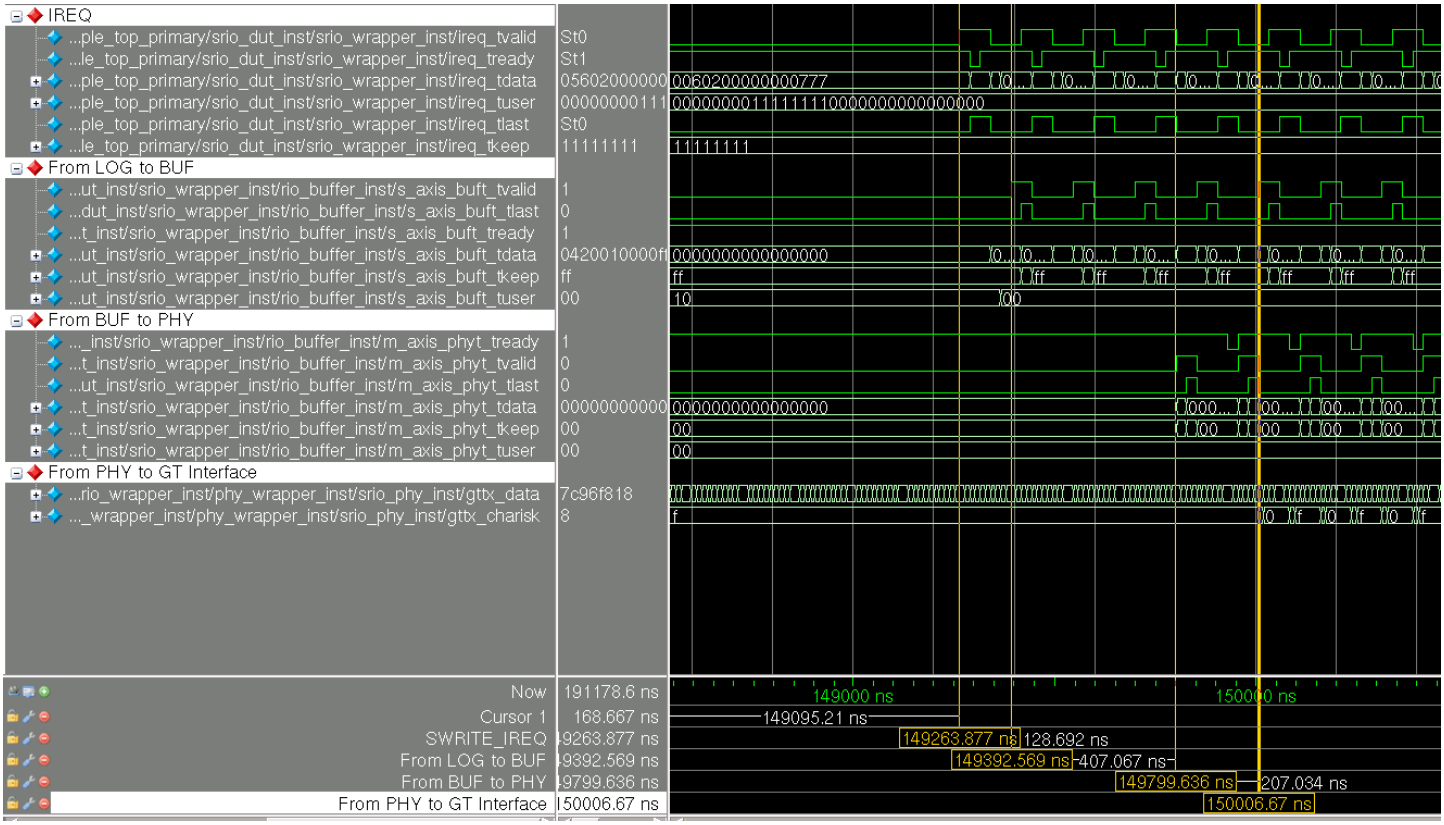

Figure 18 – SWRITE Packet from IREQ to GT Interface

To view this packet, look at the GT interface on gttx_data (Figure 3, Interface-7). As shown in Figure 19, the whole packet transmitted from the PHY layer to the GT consists of 7c96f818_ b04600ff_00000770_00000000_00000000_34b70000_7c97fa1b.

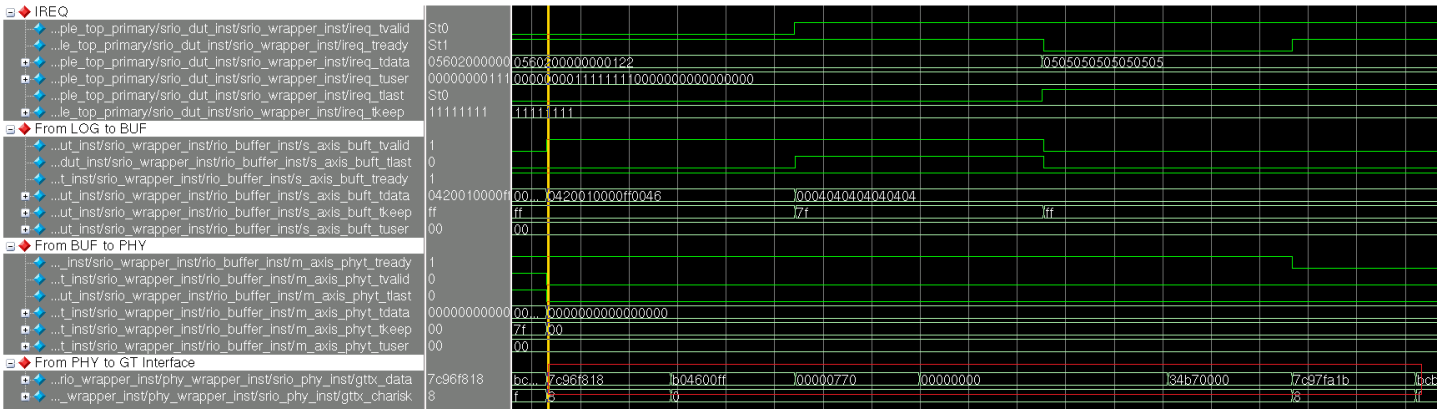

Figure 19 – SWRITE Packet Transmission on GT Interface

This packet is interpreted as follows:

Start of Packet (SOP) -> “7c96f818”, 7c indicates packet deliminator control symbol

“96f818” = 1001_0110_1111_1000_0001_1000

- 100 (Status)
- 10110 (ackID 16)
- 11111 (buf_status)
- 000 (SOP)
- 000 (cmd)
- 11000 (CRC)

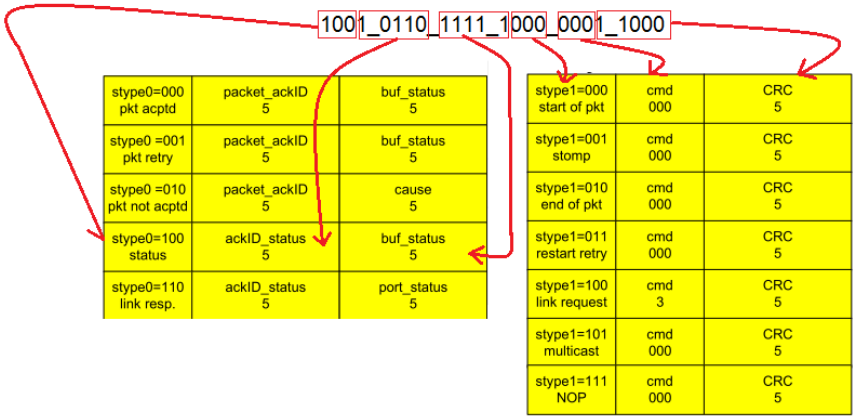

Figure 20 – Packet Deliminator Control Symbol (SOP) Analysis

SWRITE packet -> “b04600ff 00000770”, write data is all “0” (64bits),

“b04600ff 00000770” =
“1011_0000_0100_0110_0000_0000_1111_1111_0000_0000_0000_0000_0000_0111_0111 0000”.

- ackID “10110” (16)
- rsvd “00”
- crf “0”
- prio “01”
- tt “00”,
- FTYPE “0110” (6, SWRITE)
- Destination ID “0000 0000”
- Source ID “1111 1111”
- Addr (29bits) “0000 0000 0000 0000 0000 0111 0111 0”
- rsrv “0”
- xamsbs “00”.

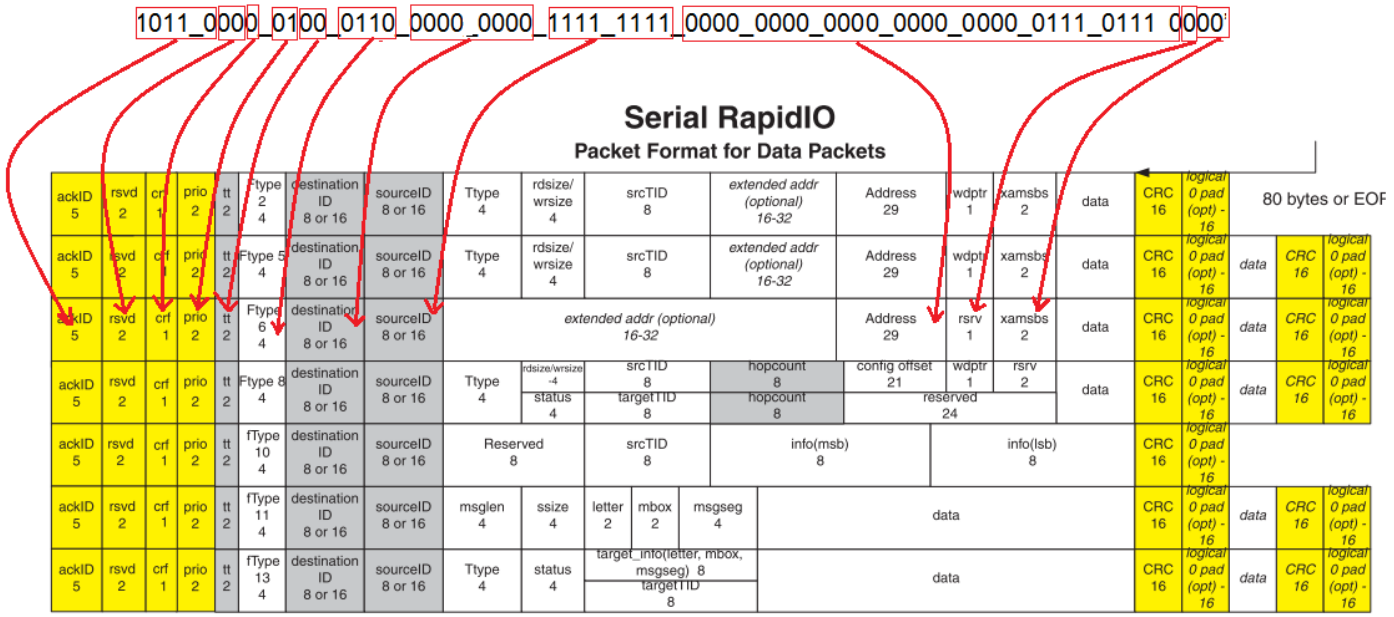

Figure 21 – SWRITE Packet Header Field Analysis

End of Packet (EOP) -> “7c97fa1b”, 7c indicates packet deliminator control symbol

“97f81c” = 1001_0111_1111_1010_0001_1011

- 100 (Status)
- 10111 (ackID 17)
- 11111 (buf_status)
- 010 (EOP)
- 000 (cmd)
- 11011 (CRC)

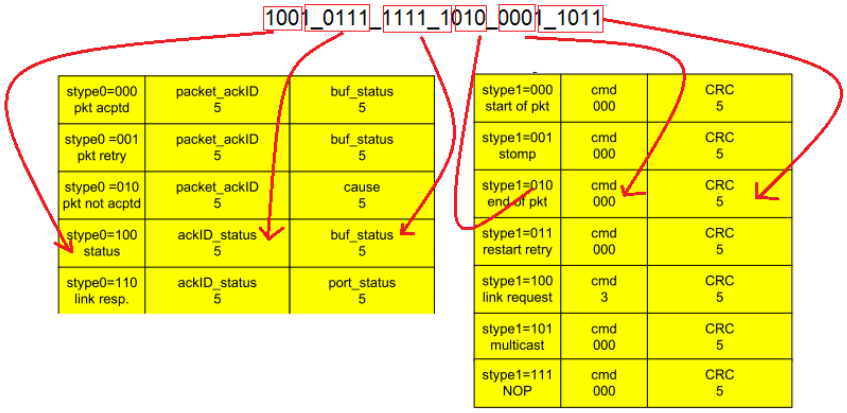

Figure 22 – Packet Deliminator Control Symbol (EOP) Analysis

NWRITE_R Packet Analysis

Figure 23 shows an example of a NWRITE_R Packet sent to the IREQ interface (Figure 3, Interface 1). The data on ireq_tdata is “3655204004550000”.

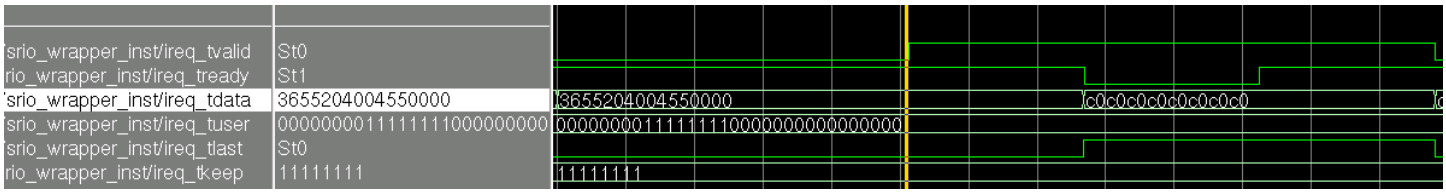

Figure 23 - NWRITE_R Packet Transmission on IREQ Interface

According to Figure 12 HELLO FTYPE Packet Format, the corresponding field interpretation for the above data in Figure 23 is as follows:

NWRITE_R packet -> “3655204004550000”, write data is “c0c0c0c0 c0c0c0c0” (64bits),

bit[63:56]=36 (srcTID)
bit[55:52]=0101 (FTYPE 5)
bit[51:48]=0101 (TTYPE 5)

Figure 25 shows NWRITE_R packet transmitted on the GT interface (Figure 3, Interface 7) which consists of 604500ff_57360455_0004c0c0_c0c0c0c0_c0c01631.

According to SRIO Packet Format, this packet is interpreted as follows:

NWRITE_R packet -> “604500ff 57360455”

- ackID, rsvd, crf, prio, tt (604)
- FTYPE “0101” (5, NWRITE)
- Destination ID “0000 0000”
- Source ID “1111 1111”
- TTYPE “0101” (5, NWRITE_R)
- wrsize (7)
- srcTID (36)

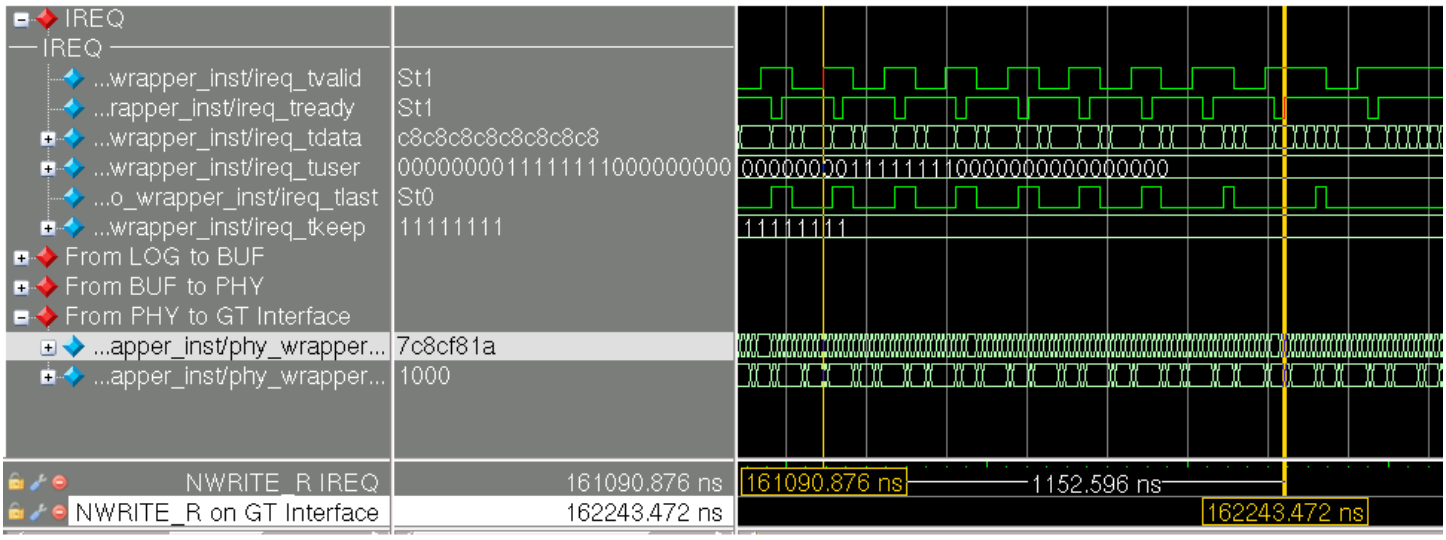

Figure 24 - NWRITE_R on IREQ and GT Interface

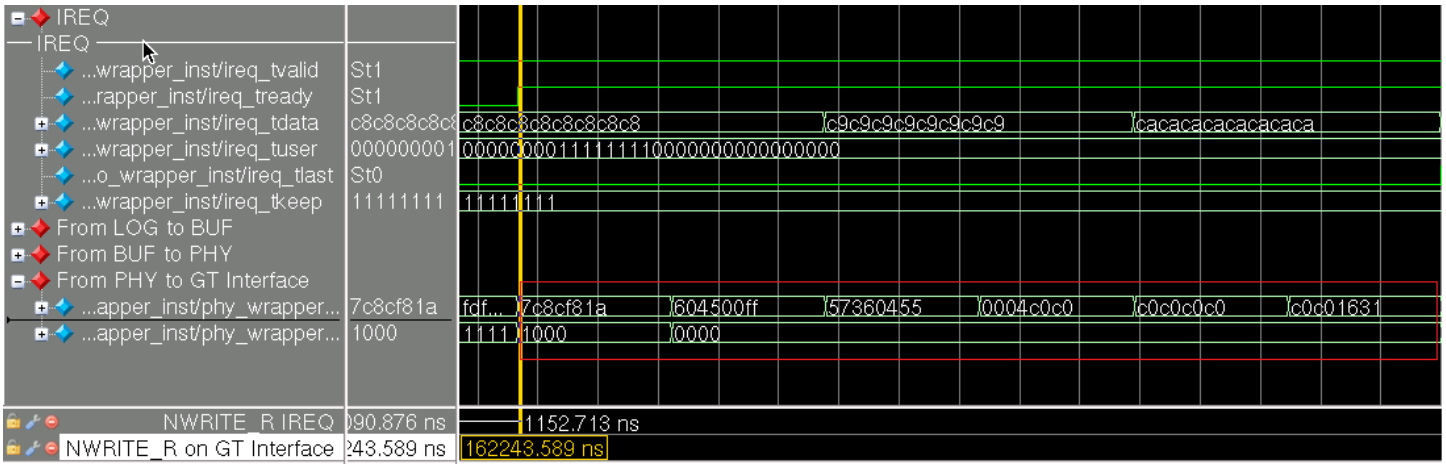

Figure 25 - NWRITE_R Packet Transmission on GT Interface

In the example design, the primary module is connected to the mirror module. Therefore, GTRX (Figure 3, Interface 7) of the mirror module receives the exact same data as GTTX of the primary module, as shown in Figure 26.

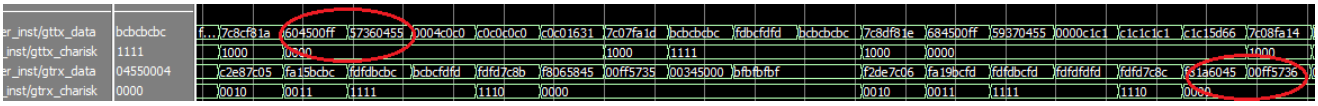

Figure 26 - NWRITE_R Packet Received on GT Interface of the Mirror Module

The receiving SRIO decodes the packet in the same way that the transmitting SRIO does. The same NWRITE_R packet is received on the TREQ interface (Interface 1 in Figure 3). Figure 27 shows the received NWRITE_R packet on TREQ interface of the mirror module. The data on m_axis_treq_tdata interface is “3655204004550000” which is the same as the data on s_axis_ireq_tdata.

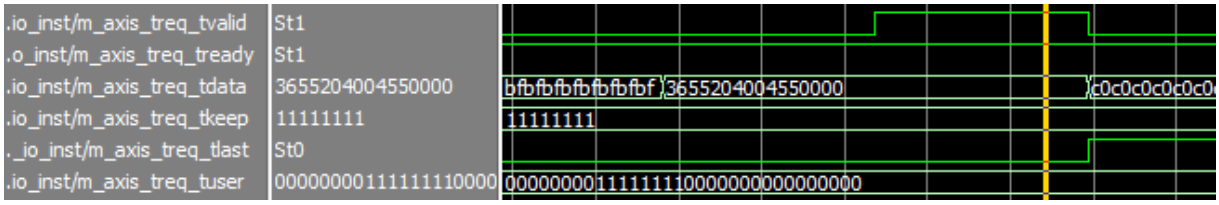

Figure 27 - NWRITE_R Packet Received on TREQ Interface (Figure 3, Interface 1)

NWRITE_R REPSONSE packet Analysis

For a NWRITE_R packet, the target should send a response to the Initiator after it receives a packet. Figure 31 shows a response packet sent to the TRESP interface (Figure 3, Interface 1). The data on s_axis_tresp_tdata is “36d0000000000000”.

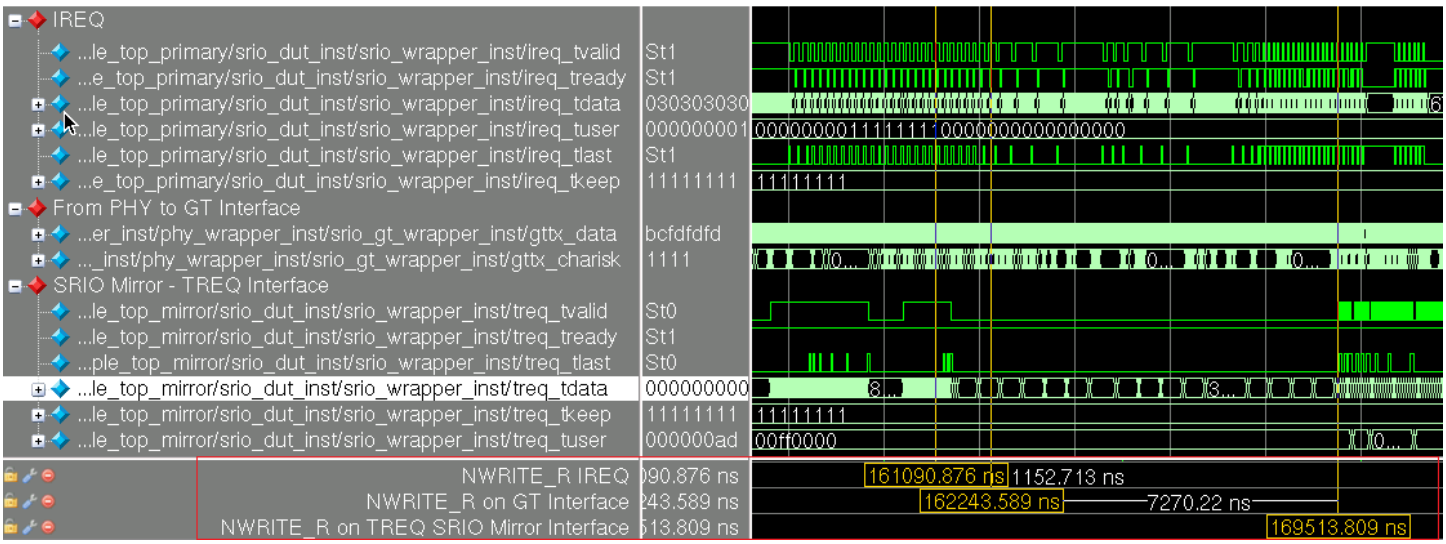

Figure 28 – NWRITE_R on IREQ, GT Interface of SRIO Primary and TREQ of SRIO Mirror

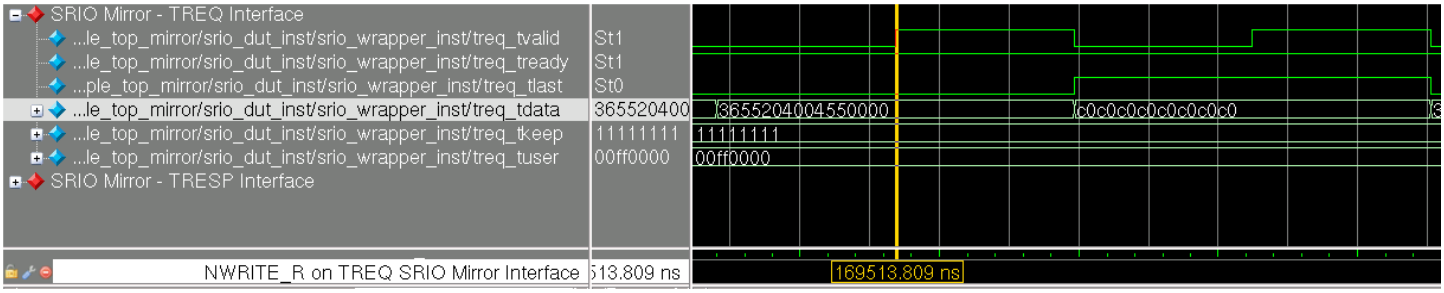

Figure 29 – NWRITE_R on SRIO Mirror TREQ Interface

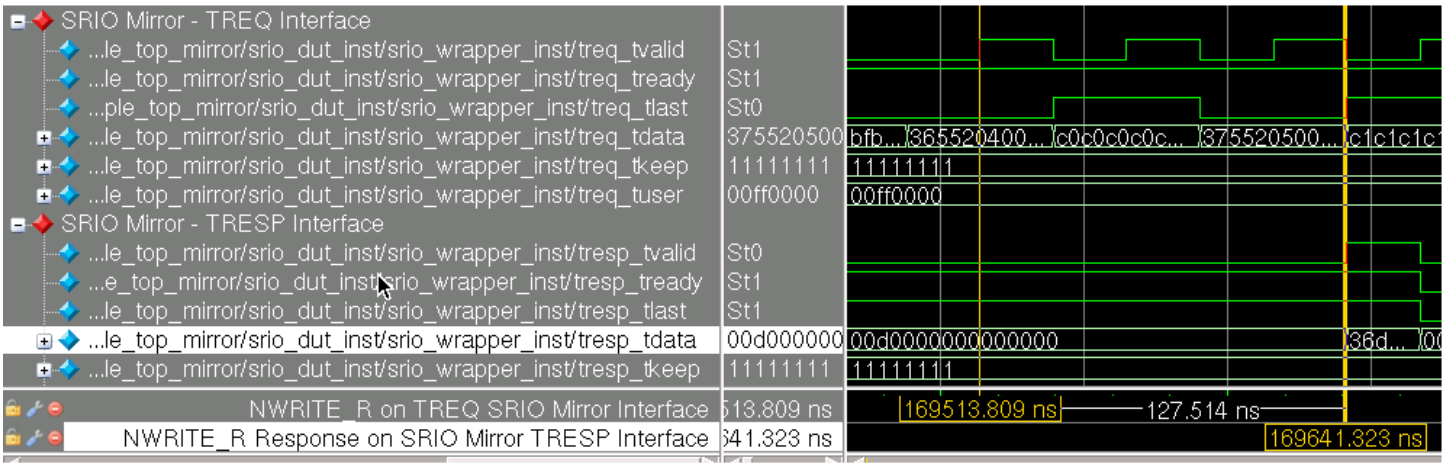

Figure 30 – NWRITE_R Response on SRIO Mirror TRESP Interface

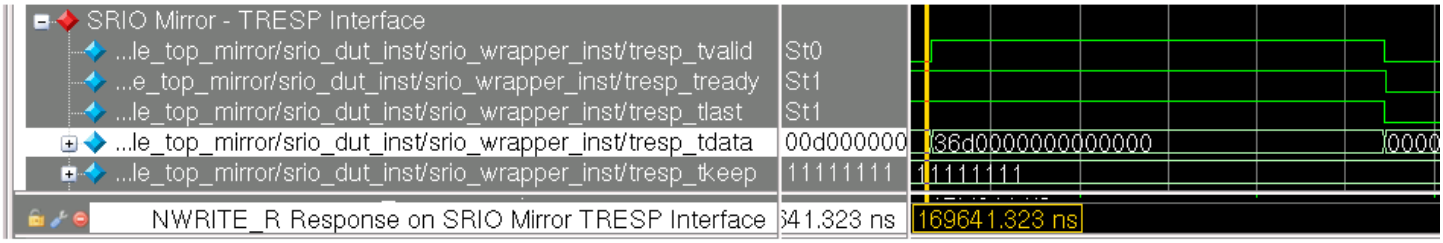

Figure 31 - Response Packet Transmission on TRESP Interface (Figure 3, Interface 1)

According to Figure 12 HELLO FTYPE Packet Format, the corresponding field interpretation for the above data in Figure 31 is as follows:

36d0000000000000 =
0011_0110_1101_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000

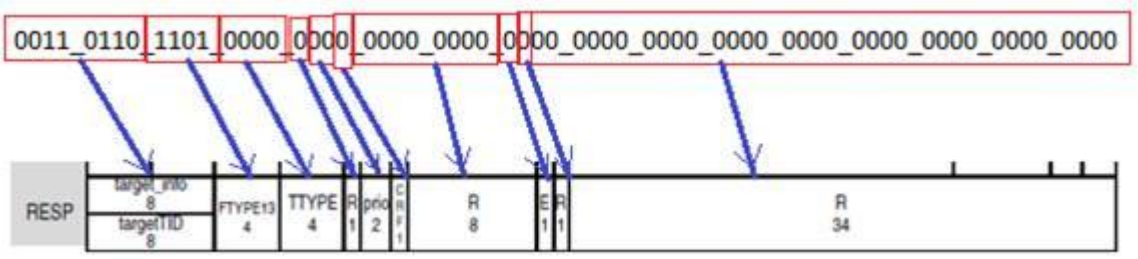

Figure 32 – RESPONSE HELLO Packet Format

bit[63:56]=36 (targetTID)
bit[55:52]=1101 (FTYPE d)
bit[51:48]=0000 (TTYPE 0, resp no data)

Figure 35 shows the RESPONSE packet transmitted on the GT interface (Interface 7 in Figure 3). It consists of 900d00ff_0036b69c.This packet is interpreted as follows:

RESPONSE packet -> “900d00ff_0036b69c” =

“1001_0000_0000_1101_0000_0000_1111_1111_0000_0000_0011_0110_1011_0110_1001_1100”

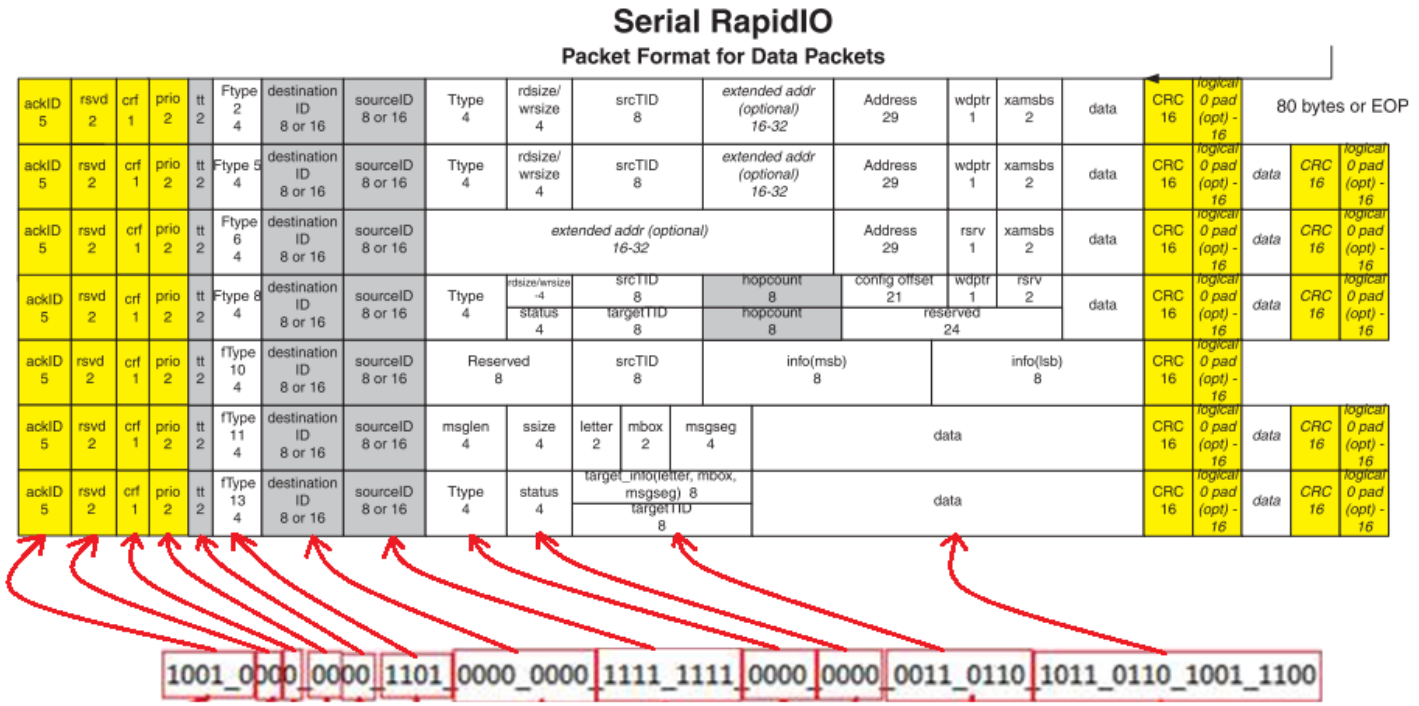

Figure 33 - RESPONSE Packet Header Field Analysis

- ackID, rsvd, crf, prio, tt (900)
- FTYPE “1101” (d, RESPONSE)
- Destination ID “0000 0000”
- Source ID “1111 1111”
- TTYPE “0000” (0, resp no data)
- STATUS “0000” (0, done)
- Target TID (36)

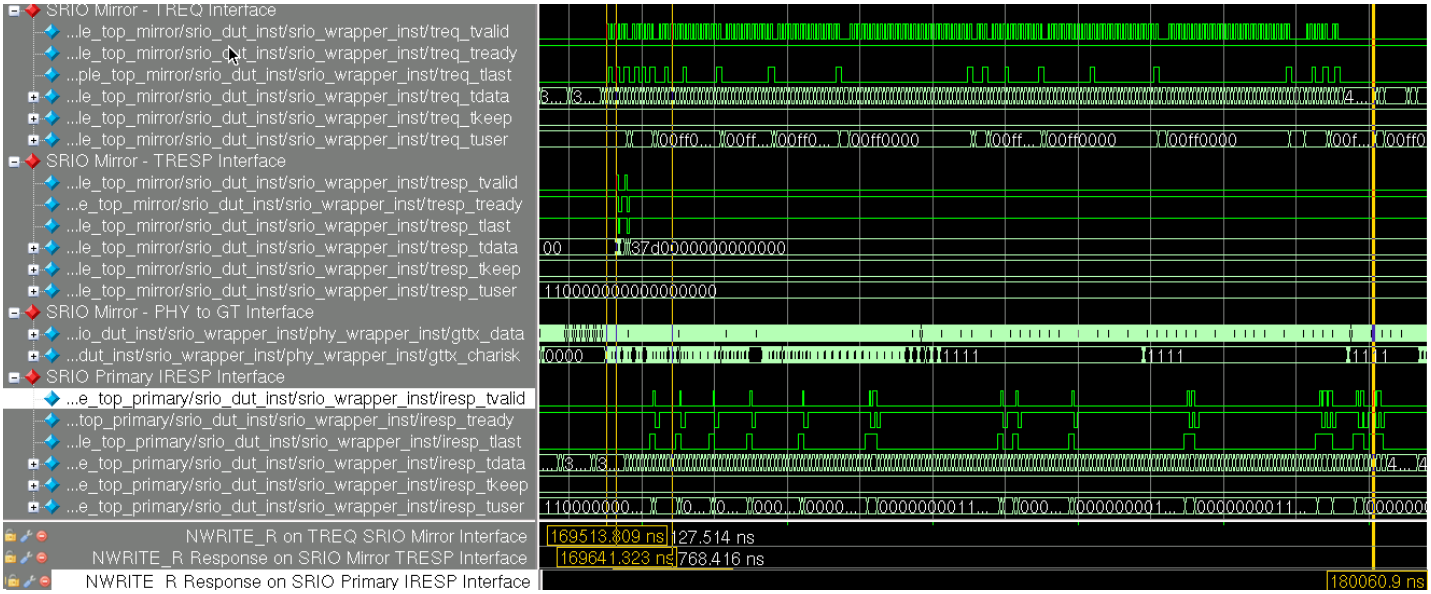

Figure 34 - NWRITE_R on TREQ SRIO Mirror, NWRITE_R Response on SRIO Mirror TRESP and NWRITE_R Response on SRIO Primary IRESP Interface

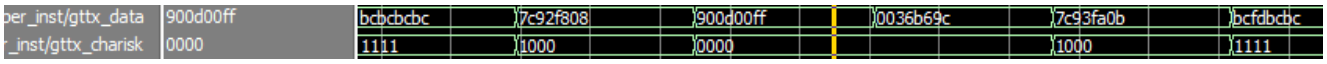

Figure 35 - RESPONSE Packet Transmission on GT Interface

Finally, the initiator gets its NWRITE_R RESPONSE from the target on the IRESP interface. Figure 36 shows an example of the RESPONSE packet received on the IRESP interface. The data on m_axis_iresp_tdata is “36d0000000000000”.

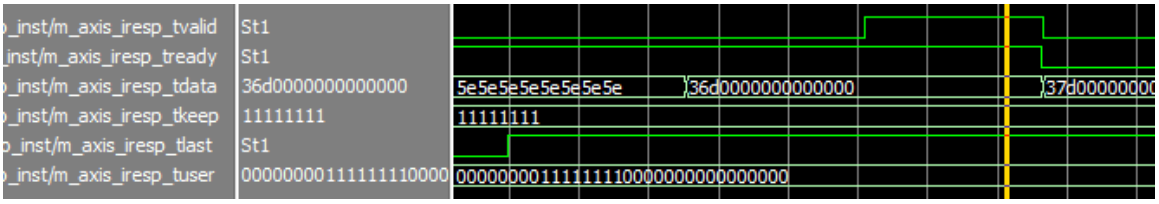

Figure 36 - RESPONSE Packet Received on IRESP Interface

According to Figure 12 HELLO FTYPE Packet Format, the corresponding field interpretation for the above data in Figure 36 is as follows:

bit[63:56]=36 (targetTID)
bit[55:52]=1101 (FTYPE d)
bit[51:48]=0000 (TTYPE 0)

NWRITE Packet Analysis

Figure 37 shows an example of a NWRITE packet sent to the IREQ interface (Figure 3, Interface 1). The data on s_axis_ireq_tdata is “3854205198877600”.

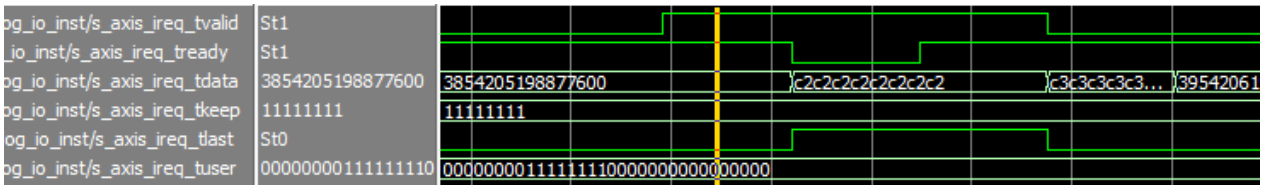

Figure 37 - NWRITE Packet Transmission on IREQ Interface

According to Figure 12 HELLO FTYPE Packet Format, the corresponding field interpretation for the above data in Figure 37 is as follows:

3854205198877600 =
0011_1000_0101_0100_0010_0000_0101_0001_1001_1000_1000_0111_0111_0100_0000_0000

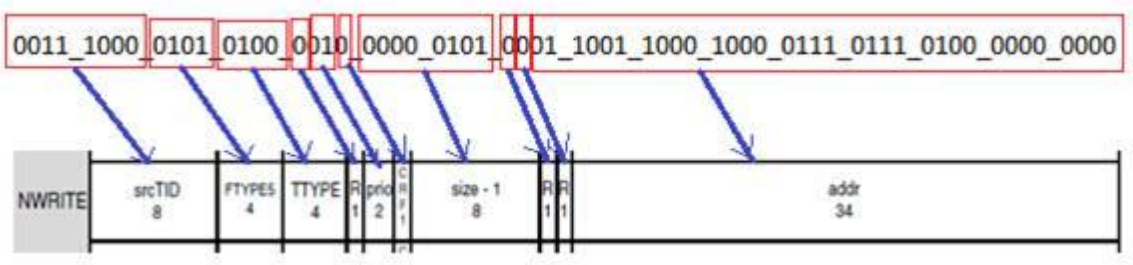

Figure 38 - NWRITE HELLO Packet Format

bit[63:56]=38 (srcTID)

bit[55:52]=0101 (FTYPE 5)
bit[51:48]=0100 (TTYPE 4)

Figure 40 shows the NWRITE packet transmitted on the GT interface (Interface 7 in Figure 2) which consists of 704500ff_49389887_7605c2c2_.....

According to Figure 12 SRIO Packet Format for Data Packets, this packet is interpreted as follows:

NWRITE packet -> “704500ff_49389887_7605c2c2” =

> “0111_0000_0100_0101_0000_0000_1111_1111_0100_1001_0011_1000_1001_1000_1000_0111_0111_0110_0000_0101_1100_0010_1100_0010”

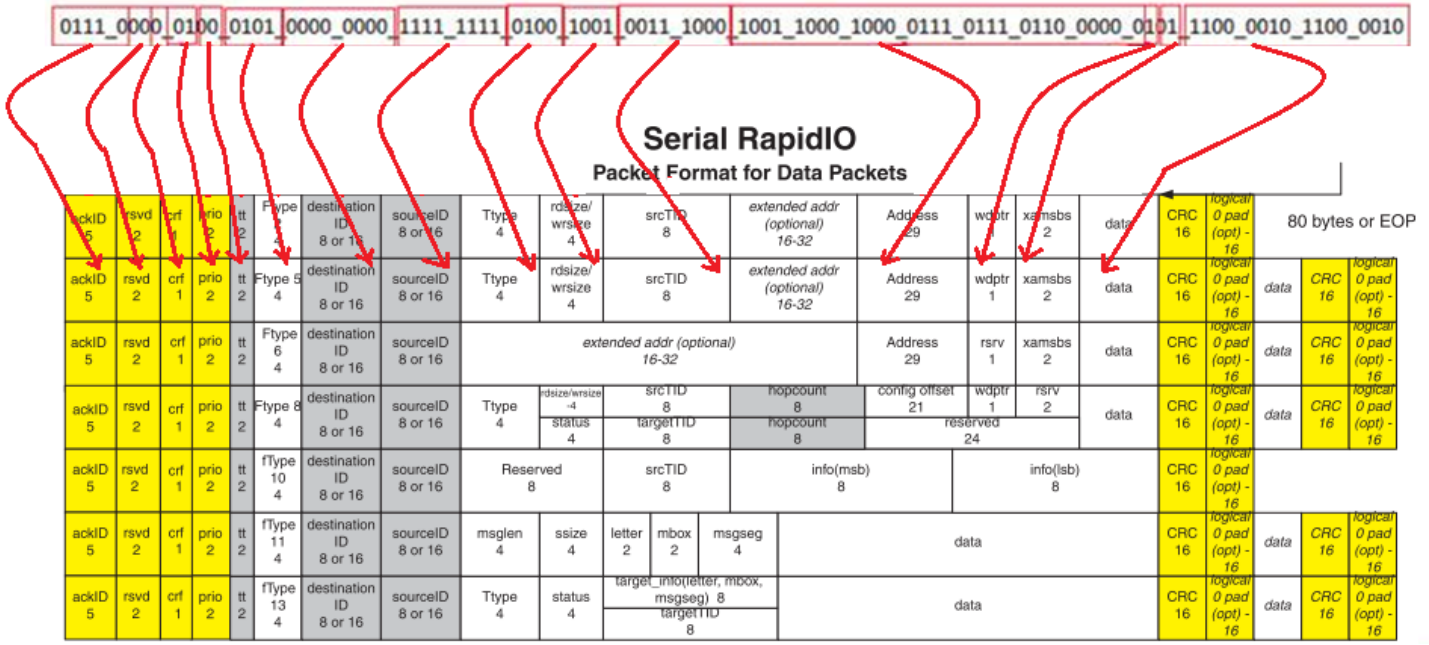

Figure 39 - NWRITE Packet Header Field Analysis

- ackID, rsvd, crf, prio, tt (d84)
- FTYPE “0101” (5, NWRITE)
- Destination ID “0000 0000”
- Source ID “1111 1111”
- TTYPE “0101” (4, NWRITE)
- wrsize (9)
- srcTID (38)

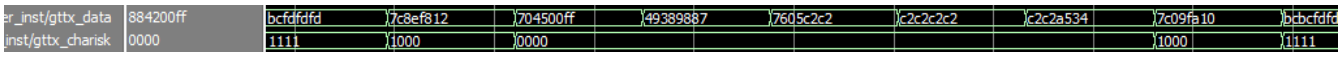

Figure 40 - NWRITE Packet Transmission on GT Interface

NREAD Packet Analysis

Figure 41 shows an example of a NREAD packet sent to the IREQ interface (Interface 1 in Figure 2). The data on s_axis_ireq_tdata is “4c24200000002406”.

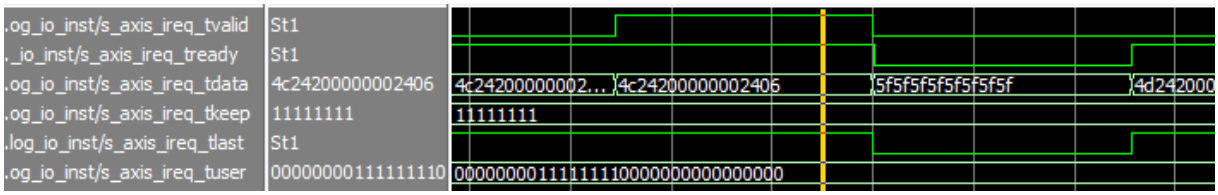

Figure 41 - NREAD Packet Transmission on IREQ Interface

According to the HELLO FTYPE Packet Format, the corresponding field interpretation for the above data in Figure 16 will be as follows:

bit[63:56]=4C (srcTID)
bit[55:52]=0010 (FTYPE 2)
bit[51:48]=0100 (TTYPE 4)

4c24200000002406 =
0100_1100_0010_0100_0010_0000_0000_0000_0000_0000_0000_0000_0010_0100_0000_0110

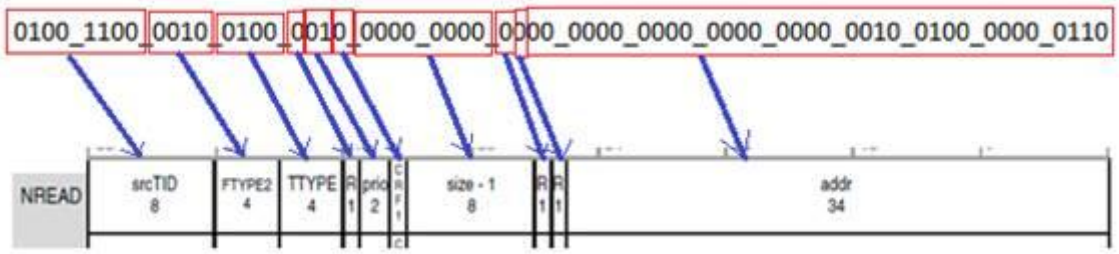

Figure 42 - NREAD HELLO Packet Format

Figure 44 shows the NREAD packet transmitted on the GT interface which consists of 884200ff_414c0000_2400a27. This packet is interpreted as follows:

NREAD packet ->

“884200ff 414c0000 2400a227” =
“1000_1000_0100_0010_0000_0000_1111_1111_0100_0001_0100_1100_0000_0000_0000_0000_0010_0100_0000_0000_0000_1010_0010_0010_0111”

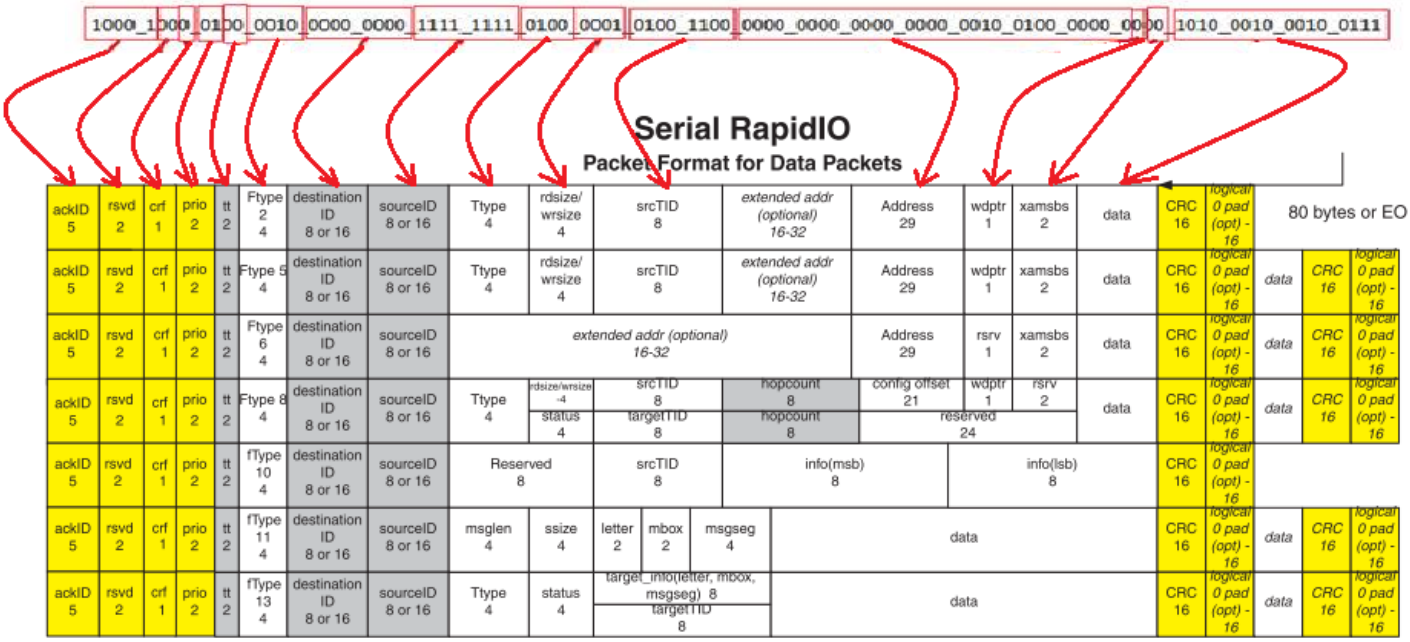

Figure 43 - NREAD Packet Header Field Analysis

- ackID, rsvd, crf, prio, tt (884)
- FTYPE “0010” (2, NREAD)
- Destination ID “0000 0000”
- Source ID “1111 1111”
- TTYPE “0101” (4, NREAD)
- rdsize (1)
- srcTID (4C)

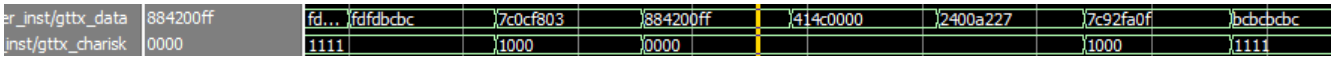

Figure 44 - NREAD Packet Transmission on GT Interface

NREAD RESPONSE Packet Analysis

Figure 46 shows the RESPONSE with data on the GT interface which consists of xxxxxx88_8dff0080_4c000000. This is received in response to the NREAD packet in Figure 44 and is interpreted as follows:

“888dff0080 4c000000” =

“1000_1000_1000_1101_1111_1111_0000_0000_1000_0000_0100_1100_0000_0000_0000_0000_0000_0000”

- ackID “10001”
- rsvd “00”
- crf “0”
- prio “10”
- tt “00”,
- FTYPE “1101” (13, RESPONSE)
- Destination ID “1111 1111”
- Source ID “0000 0000”
- Ttype “1000” (8, Response with Data)
- Status “0000” (0,done)
- TargetID “0100 1100” (4C)

- data “0000 0000 0000 0000 0000 0000”

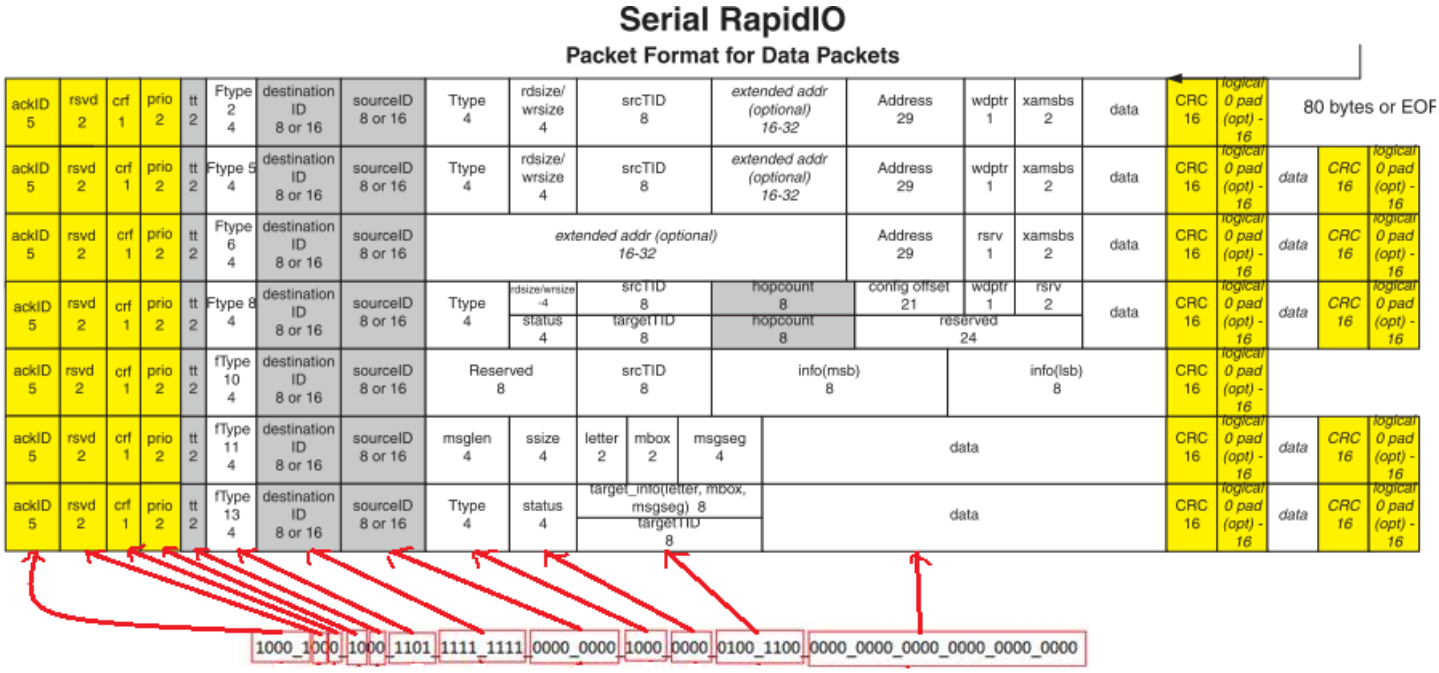

Figure 45 - NREAD Response Header Field Analysis

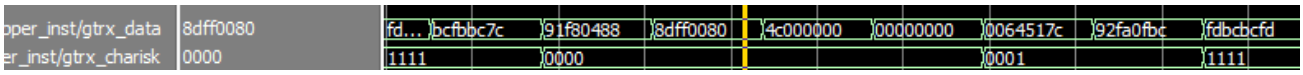

Figure 46 - RESPONSE with Data for NREAD Packet Received on GT Interface

Figure 47 shows the RESPONSE with data packet received on the IRESP Interface. The data on m_axis_iresp_tdata is “4cd8400000000000”.

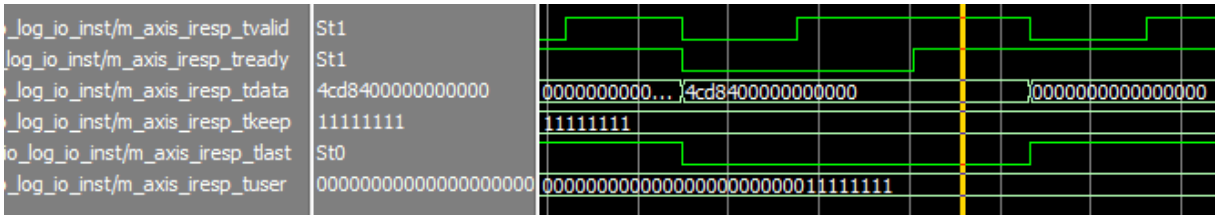

Figure 47 - RESPONSE with Data Packet Received on IRESP Interface

According to Figure 12 HELLO FTYPE Packet Format, the corresponding field interpretation for the above data in Figure 47 will be as follows:

4cd8400000000000 =
0100_1100_1101_1000_0100_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000

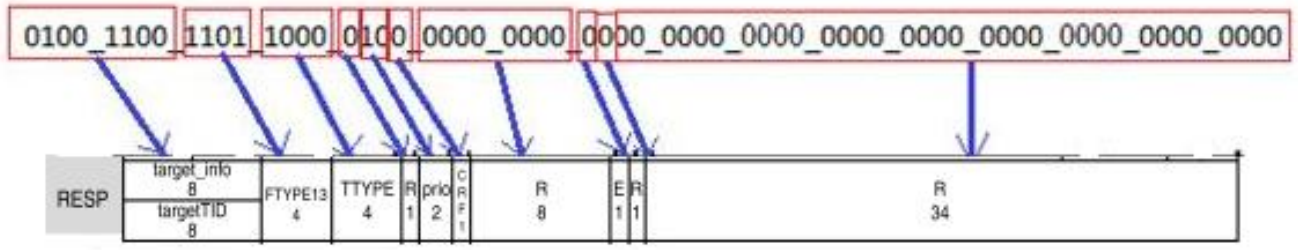

Figure 48 – REPONSE HELLO Packet Format

Example Design Maintenance Transactions

The srio_quick_start module, which is instantiated in srio_example_top, connects to the Maintenance port and generates maintenance transactions. The user can add, modify, or remove transactions by editing the maintenance_list file. Figure 49 shows the list of maintenance transactions included in the example design.

```
                                    // RSVD, LOCAL/REMTE, ADDRESS,    RD/WR, DATA,         DATA MASK
localparam [63:0] MAINTENANCE0   = {2'b0, REMTE,        24'h000000, READ,  32'h0360000E, 4'hF};
localparam [63:0] MAINTENANCE1   = {2'b0, LOCAL,        24'h000000, READ,  32'h0360000E, 4'hF};

localparam [63:0] MAINTENANCE2   = {2'b0, LOCAL,        24'h000060, WRITE, 32'hFF020304, 4'h0}; //
localparam [63:0] MAINTENANCE3   = {2'b0, REMTE,        24'h000060, READ,  32'h00020304, 4'h0};
localparam [63:0] MAINTENANCE4   = {2'b0, REMTE,        24'h000060, WRITE, 32'hDEADBEEF, 4'h0};
localparam [63:0] MAINTENANCE5   = {2'b0, LOCAL,        24'h000060, READ,  32'h00ADBEEF, 4'h0};

localparam [63:0] MAINTENANCE6   = {2'b0, REMTE,        24'h00006C, WRITE, 32'h55555555, 4'h0};
localparam [63:0] MAINTENANCE7   = {2'b0, LOCAL,        24'h00006C, READ,  32'h55555555, 4'h0};
localparam [63:0] MAINTENANCE8   = {2'b0, LOCAL,        24'h00006C, WRITE, 32'hAAAAAAAA, 4'h0};
localparam [63:0] MAINTENANCE9   = {2'b0, REMTE,        24'h00006C, READ,  32'hAAAAAAAA, 4'h0};

localparam [63:0] MAINTENANCE10  = {2'b0, REMTE,        24'h010000, WRITE, 32'hFF08090A, 4'h0};
localparam [63:0] MAINTENANCE11  = {2'b0, LOCAL,        24'h010000, READ,  32'h0008090A, 4'h0};
localparam [63:0] MAINTENANCE12  = {2'b0, LOCAL,        24'h010000, WRITE, 32'h00030405, 4'h0};
localparam [63:0] MAINTENANCE13  = {2'b0, REMTE,        24'h010000, READ,  32'h00030405, 4'h0};

localparam [63:0] MAINTENANCE14  = {2'b0, REMTE,        24'h000120, WRITE, 32'hFFFFFFFF, 4'h0};
localparam [63:0] MAINTENANCE15  = {2'b0, REMTE,        24'h000120, READ,  32'hFFFFFF00, 4'h0};
localparam [63:0] MAINTENANCE16  = {2'b0, LOCAL,        24'h000124, WRITE, 32'h01020304, 4'h0};
localparam [63:0] MAINTENANCE17  = {2'b0, LOCAL,        24'h000124, READ,  32'h01020300, 4'h0};

localparam [63:0] MAINTENANCE18  = {2'b0, LOCAL,        24'h000410, READ,  32'h00007FC8, 4'h0};
localparam [63:0] MAINTENANCE19  = {2'b0, LOCAL,        24'h000430, READ,  32'h00107FC8, 4'h0};
localparam [63:0] MAINTENANCE20  = {2'b0, REMTE,        24'h000450, READ,  32'h00207FC8, 4'h0};
localparam [63:0] MAINTENANCE21  = {2'b0, REMTE,        24'h000470, READ,  32'h00307FC8, 4'h0};
```

Figure 49 - Maintenance Transactions in Generated by the Example Design

Figure 50 shows one of the maintenance transactions in the example design **(**Figure 3**,** Interface-2).

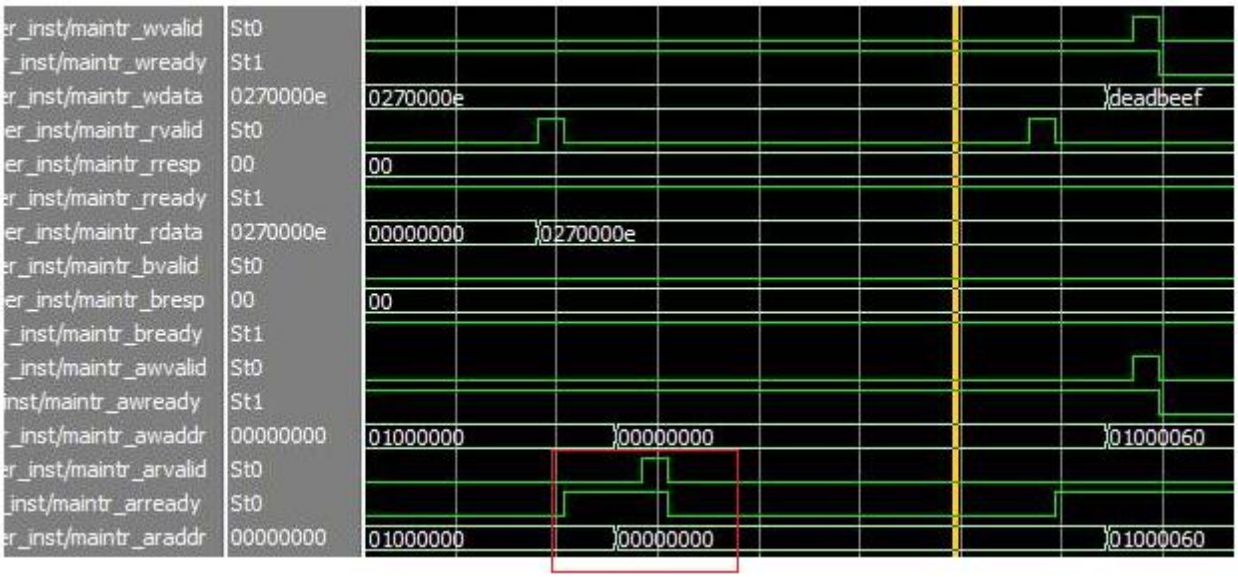

Figure 50 - Maintenance Transactions

There are two types of Maintenance transactions: Local and Remote. These transactions are identified in the waveform by checking maintr_araddr. As shown in Figure 51, bits 31:24 indicate whether the maintenance read transaction is local or remote; bits 23:0 gives configuration offset for read.

If maintr_araddr=01xxxxxx, it indicates Remote Maintenance Read Transaction
If maintr_araddr=00xxxxxx, it indicates Local Maintenance Read Transaction

s_axi_maintr_araddr[31:0]	Input	Read address. • [31:24] - Hop count + 1 (8'h00 for local read) • [23:0] - Configuration offset for read

Figure 51 - Local and Remote Maintenance Read Transaction Identification

In Figure 50, the transaction in the red box is Local Maintenance Read Transaction. The offset value is “000000” which means it is issuing Local Maintenance read transaction to read Capability Register Space in the LOG core.

Table 1 - Register Space

Configuration Space Byte Offset	RapidIO Specification Register Space	Xilinx SRIO Core
0x00-0x3C	Capability Register Space	LOG(1)
0x040-0xFC	Command and Status Register Space	LOG
0x0100-0xFFFC	Extended Features Space	PHY
0x010000-0xFFFFFC	Implementation-defined Space	BUF, LOG(2)

Figure 52 shows maintenance ports (Figure 3, Interface-2) and LOG Configuration Fabric (Figure 3, Interface-3).

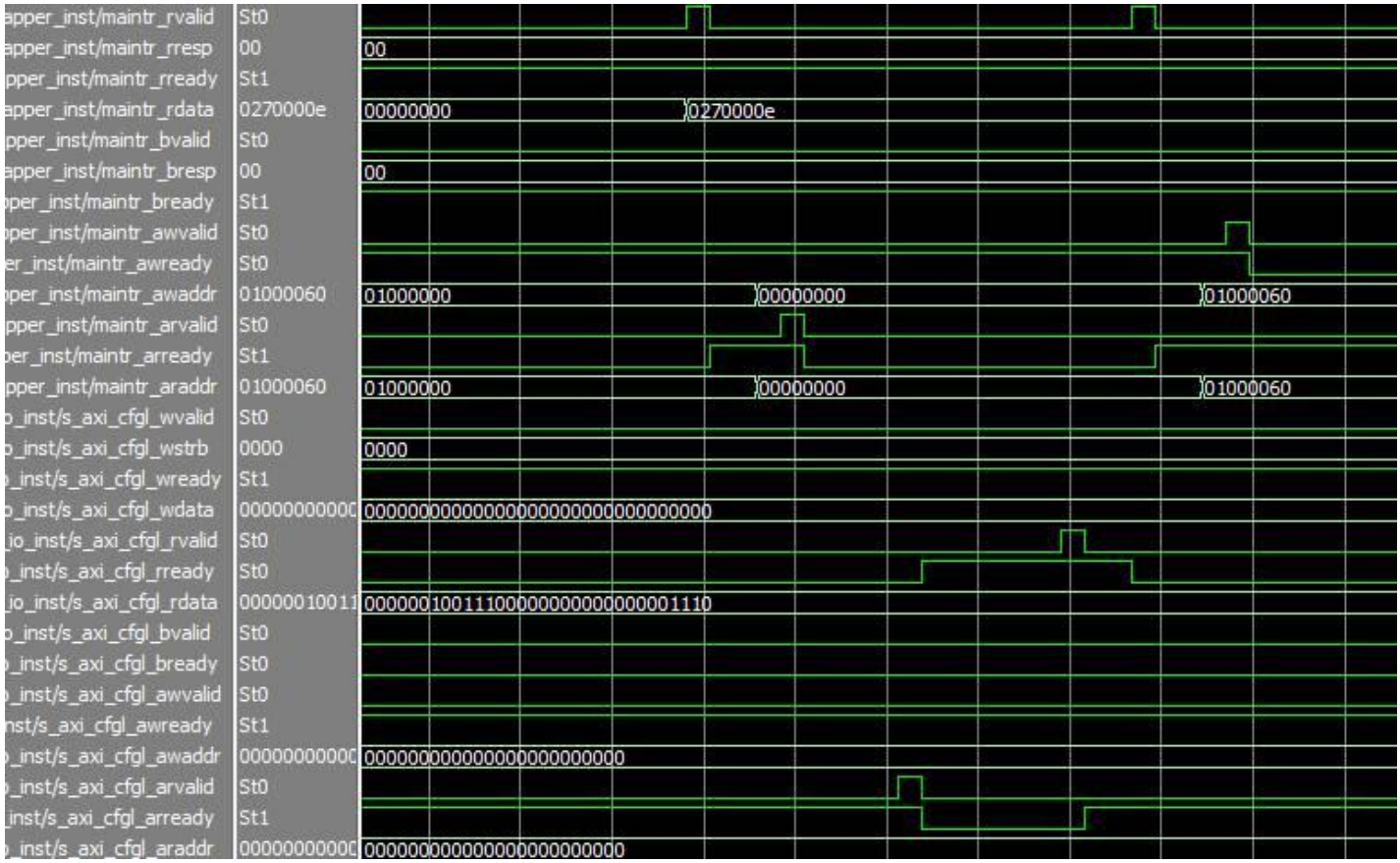

Figure 52 - Maintenance Transaction and LOG Configuration Fabric

When maintr_araddr = 01000000, it is a remote READ Maintenance transaction from the LOG core. The GTX sends a Maintenance packet to read from the remote register space (Figure 3, Interface-7). Figure 53 shows Maintenance read packet on the GT interface which is “104800ff 18010000”. This packet is interpreted as follows:

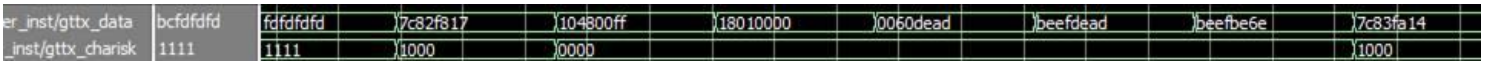

Figure 53 - Maintenance Transaction on GT Interface

Maintenance Read Packet ->

ackID, rsvd, crf, prio, tt (104)
FTYPE “1000” (8, MAINTENANCE)
Destination ID “0000 0000”
Source ID “1111 1111”
TTYPE “0000” (0, config read req)
rdsize “0001” (1)
Source TID (8)

Example Design Features and Limitations

The Serial RapidIO Gen2 Endpoint example design connects two instances of the core together. Each core instance can be configured to send supported packet types, check for receive packet mismatches, and report in the simulation transcript with details about the link traffic. Both the simulation host and the primary core being tested use the RapidIO Endpoint reference design, consisting of a Serial RapidIO Physical Layer core, Buffer core, RapidIO Logical Layer core, and Configuration Fabric reference design.

By default, the example design generates stimulus automatically for both the I/O and maintenance transactions. The maintenance transactions are generated in the srio_quick_start module. This module sends a fixed set of instructions to the configuration register space, both local and remote. Figure 49 shows the list of maintenance transactions generated in the example design.

The I/O transactions are generated in the srio_request_gen and srio_response_gen modules. The srio_request_gen generates the initiator request I/O transactions as specified in instruction_list.v file. Transactions can be added, modified, or removed in the instruction_list.v. The list contains random transactions but is run in the same order each time. Only transactions supported by the connected port are produced. The srio_request_gen also keeps track of any expected responses for response-generating-requests and compares the received response to the expected value. This comparison function is only available when using the automatic stimulus generation. The example design generates SWRITE, NWRITE_R, NWRITE, NREAD, DOORBELL, and MESSAGE packet types. Figure 54 shows a list of SWRITE instructions in the instruction_list.v.

```
                              // RSVD,    FTYPE,  TTYPE,  ADDRESS,       SIZE
// SWRITEs
swrite_instruction[0]    = {12'h000, SWRITE, 4'h0,   36'h000000777, 8'd0};
swrite_instruction[1]    = {12'h000, SWRITE, 4'h0,   36'h000008806, 8'd0};
swrite_instruction[2]    = {12'h000, SWRITE, 4'h0,   36'h000000125, 8'd0};
swrite_instruction[3]    = {12'h000, SWRITE, 4'h0,   36'h000000124, 8'd0};
swrite_instruction[4]    = {12'h000, SWRITE, 4'h0,   36'h000000123, 8'd0};
swrite_instruction[5]    = {12'h000, SWRITE, 4'h0,   36'h000000122, 8'd0};
swrite_instruction[6]    = {12'h000, SWRITE, 4'h0,   36'h000000121, 8'd0};
swrite_instruction[7]    = {12'h000, SWRITE, 4'h0,   36'h000000120, 8'd0};
swrite_instruction[8]    = {12'h000, SWRITE, 4'h0,   36'h000000126, 8'd1};
swrite_instruction[9]    = {12'h000, SWRITE, 4'h0,   36'h000000124, 8'd1};
swrite_instruction[10]   = {12'h000, SWRITE, 4'h0,   36'h000000122, 8'd1};
```

Figure 54 - SWRITE Instructions in instruction_list.v

The srio_response_gen generates the target response I/O transactions in response to the requests received from the link partner. Requests that do not require a response are dropped and the srio_response_gen does not respond to SWRITE transaction types. This module automatically fills the I/O transaction data with an incrementing pattern and bumps the priority by 1. The responses are generated in the order that the requests are received; so no out-of-order transactions are generated.

To check the sequence of packet transactions in simulation, you can take a look into the waveform and decode the packets. An easier way is to check the simulation transcript where it reports the transmission of I/O packets and the corresponding response packets if any. Simulation timestamp is also provided in the transcript. For further analysis of a particular packet, you can track the packet in the given timestamp in the waveform. Figure 55 shows a snapshot of the simulation transcript at the start of the I/O transactions in the example design. As seen in the figure, the same instruction number is listed twice. This is because srio_request_gen is instantiated in both SRIO Primary and Mirror instance of the simulation. Each module generates its own transactions.

```
149289600] INFO: Transmitting IO packet
149289600]       Instruction #: 00 FTYPE: 6 TTYPE: 0 size-1: 00

149341800] INFO: Transmitting IO packet
149341800]       Instruction #: 00 FTYPE: 0 TTYPE: 0 size-1: 00

149443200] INFO: Transmitting IO packet
149443200]       Instruction #: 01 FTYPE: 6 TTYPE: 0 size-1: 00

149495400] INFO: Transmitting IO packet
149495400]       Instruction #: 01 FTYPE: 0 TTYPE: 1 size-1: 10

149596800] INFO: Transmitting IO packet
149596800]       Instruction #: 02 FTYPE: 6 TTYPE: 0 size-1: 00
```

Figure 55 - Simulation Transcript

The first SWRITE packet in Figure 55 has a simulation timestamp of 149289.600 ns. Figure 56 shows the portion of the waveform in this timestamp. This is a SWRITE packet as illustrated in Figure 10.

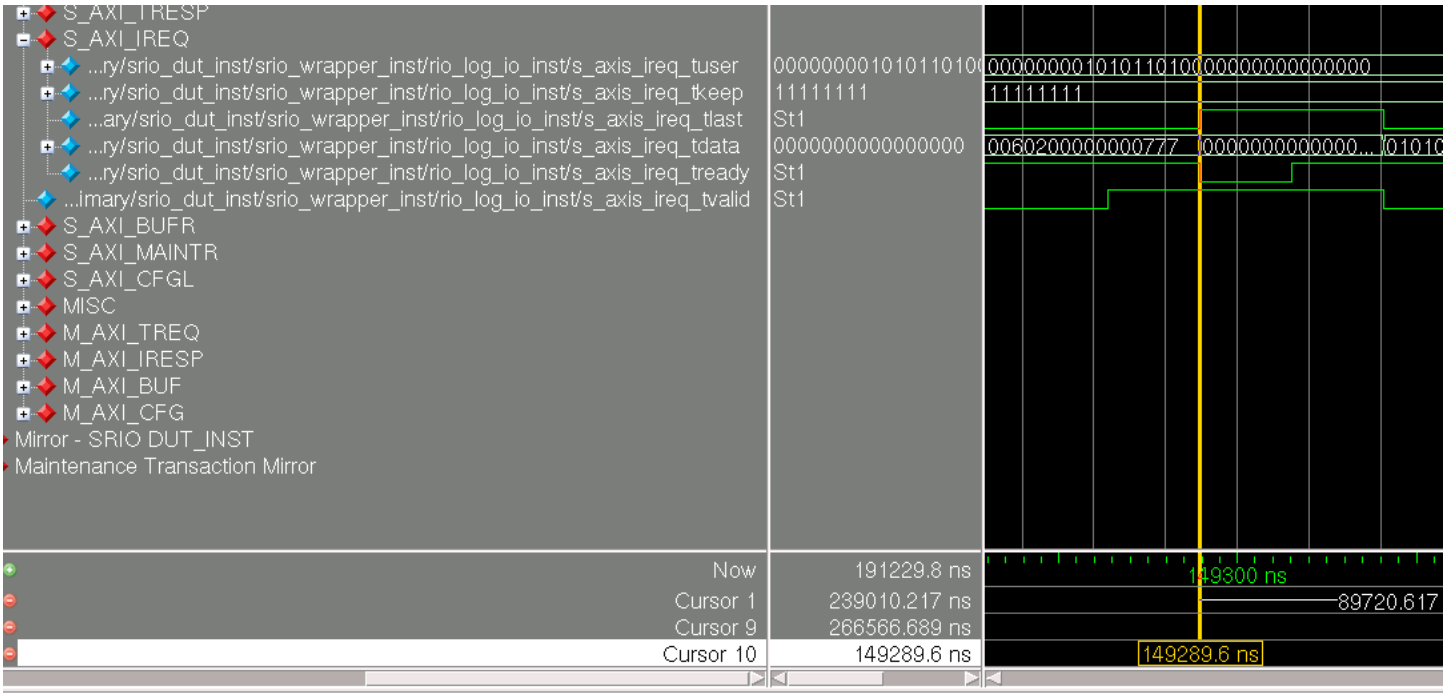

Figure 56 - SWRITE at Timestamp 149289.600 ns

New Features in SRIO Gen2 v1.4

In SRIO Gen2 v1.4 core, a list of debug signals is provided that the user could probe in the ChipScope tool. These signals are listed in (Xilinx Answer 47191).

Other new features introduced in SRIO Gen2 v1.4 are Statistics Capture, Addressable Memory Space, and ChipScope Core insertion in the generated example design. For more information on all these features, see (Xilinx Answer 47387).

Appendix-A

This section provides helpful details on packet transmission on the SRIO serial link. The provided screenshots are from RapidIO Part 6: LP-Serial Physical Layer Specification Rev. 2.2. When capturing signals on the GT interface, you should see the packets being transferred as shown in Figure 57 (on 1x link) and Figure 58 (on 4x link).

Packet Transmission on a 1x LP-Serial Link

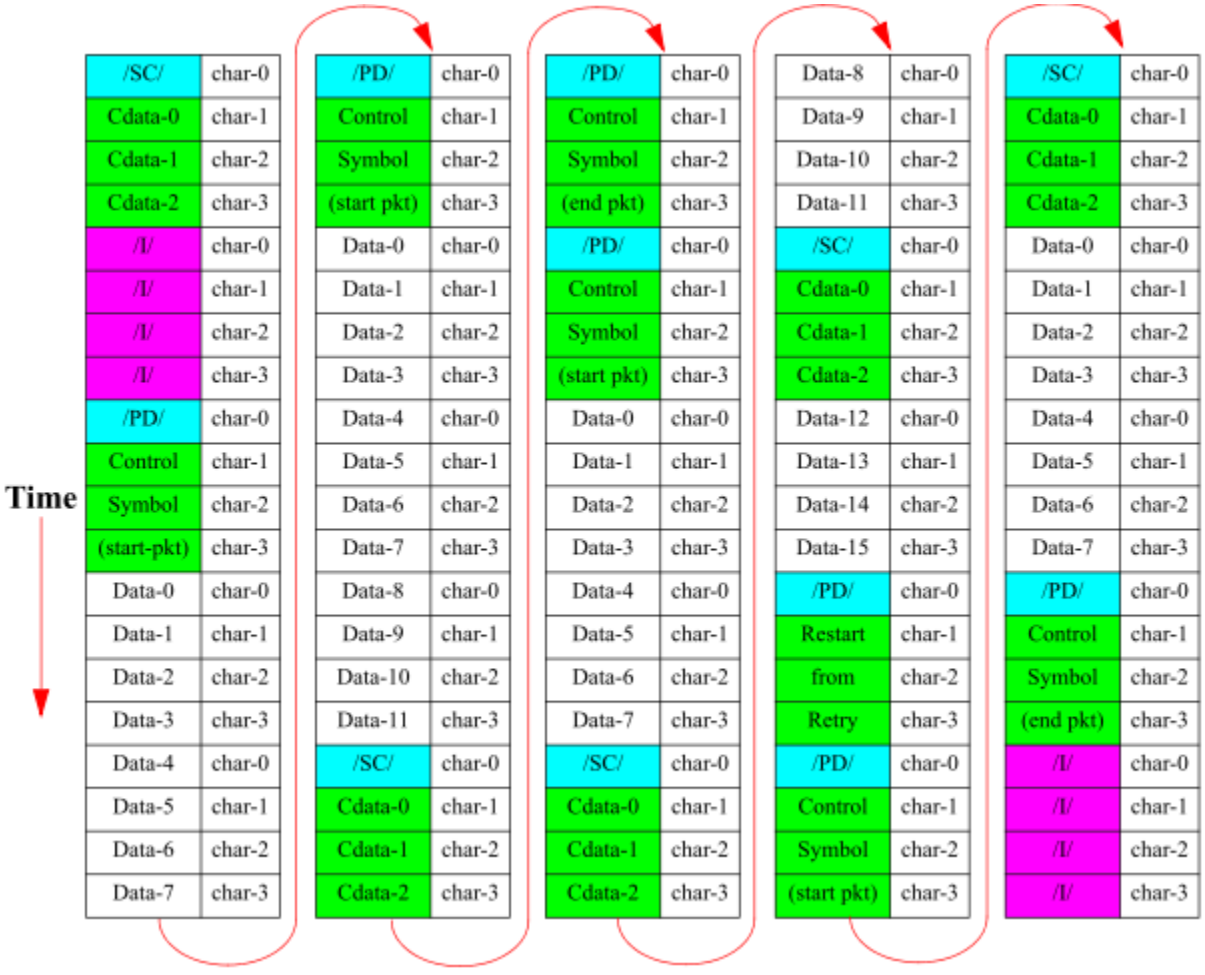

Figure 57 - 1x Serial Link Packet Transmission

Lane-0	Lane-1	**Lane-2**	**Lane-3**
/SC/	Cdata-0	Cdata-1	Cdata-2
/I/	/I/	/I/	/I/
/PD/	Control Symbol (Start-of-packet)		
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
/PD/	Control Symbol (Start-of-packet)		
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
Data-8	Data-9	Data-10	Data-11
/SC/	Cdata-0	Cdata-1	Cdata-2
/PD/	Control Symbol (End-of-packet)		
/PD/	Control Symbol (Start-of-packet)		
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
/SC/	Cdata-0	Cdata-1	Cdata-2
Data-8	Data-9	Data-10	Data-11
/SC/	Cdata-0	Cdata-1	Cdata-2
Data-12	Data-13	Data-14	Data-15
/PD/	Control Symbol (Restart-from-retry)		
/PD/	Control Symbol (Start-of-packet)		
/SC/	Cdata-0	Cdata-1	Cdata-2
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
/PD/	Control Symbol (End-of-packet)		
/I/	/I/	/I/	/I/

Figure 58 - 4x Serial Link Packet Transmission

The following table shows a list of code groups that you need to be familiar with when decoding packets in the waveform. For more details, see “4.5.7 Special Characters and Columns” in RapidIO Part 6: LP-Serial Physical Layer Specification Rev. 2.2.

Code-Group/Column Designation	Code-Group/Column Use	Number of Code-groups	Encoding
/PD/	Packet_Delimiter Control Symbol	1	/K28.3/
/SC/	Start_of_Control_Symbol	1	/K28.0/
/K/	Sync	1	/K28.5/
/R/	Skip	1	/K29.7/
/A/	Align	1	/K27.7/
/M/	Mark	1	/K28.1/
/I/	Idle	1	
||K||	Sync column	N	a column of /K28.5/
||R||	Skip column	N	a coulmn of /K29.7/
||A||	Align column	N	a column of /K27.7/
||M||	Mark column	N	a column of /K28.1/
||I||	Idle column	N	a column of Idle

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +	Notes
			abcdei fghj	abcdei fghj	
K28.0	1C	000 11100	001111 0100	110000 1011	
K28.1	3C	001 11100	001111 1001	110000 0110	2,3
K28.2	5C	010 11100	001111 0101	110000 1010	1
K28.3	7C	011 11100	001111 0011	110000 1100	
K28.4	9C	100 11100	001111 0010	110000 1101	1
K28.5	BC	101 11100	001111 1010	110000 0101	2
K28.6	DC	110 11100	001111 0110	110000 1001	1
K28.7	FC	111 11100	001111 1000	110000 0111	1,2
K23.7	F7	111 10111	111010 1000	000101 0111	1
K27.7	FB	111 11011	110110 1000	001001 0111	
K29.7	FD	111 11101	101110 1000	010001 0111	
K30.7	FE	111 11110	011110 1000	100001 0111	1

Notes
1. Reserved code-group.
2. The code-group contain a comma.
3. A Reserved code-group for Idle Sequence 1

Appendix-B

The following table shows a list of Transaction Types in Serial Rapid I/O. The information in the table is taken from the core product guide for a quick reference.

Transaction Types

Packet Type	FTYPE	TTYPE	Transmit Port	Receive Port	Description
NREAD	0010	0100	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Basic read request transaction. Request does not have a data payload. Results in a response with data.
NWRITE	0101	0100	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Basic write operation. Request has a data payload. Does not result in a response.
NWRITE_R	0101	0101	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Basic write operation. Request has a data payload. Results in a response with no data.
SWRITE	0110	N/A	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Streaming write operation (uses less header fields than NWRITE). Request has a data payload. Does not result in a response.
DOORBELL	1010	N/A	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Very short message between processing elements. Request has no data payload. Results in a response with no data.
MESSAGE	1011	N/A	• If Separate Messaging port is used: msgireq • If Messages are combined with I/O, ◦ Condensed I/O: iotx ◦ Initiator/Target: ireq	• If Separate Messaging port is used: msgtreq • If Messages are combined with I/O, ◦ Condensed I/O: iorx ◦ Initiator/Target: treq	Messaging operation - typically used by processors in distributed memory system machines. Request has no data payload. Results in a message response (with no data).
ATOMIC with no payload	0010	1100 1101 1110 1111	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Read-modify-write operation. Request does not have a data payload. Results in a response with data.
ATOMIC with payload	0101	1100 1101 1110	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Read-modify-write operation. Request has a data payload. Results in a response with data.

Packet Type	FTYPE	TTYPE	Transmit Port	Receive Port	Description
RESPONSE without data	1101	0000	• Condensed I/O: iotx • Initiator/Target: tresp	• Condensed I/O: iorx • Initiator/Target: iresp	Response transaction with no data payload.
RESPONSE with data	1101	1000	• Condensed I/O: iotx • Initiator/Target: tresp	• Condensed I/O: iorx • Initiator/Target: iresp	Response transaction with a data payload.
MESSAGE RESPONSE	1101	0001	• If Separate Messaging port is used: msgtresp • If Messages are combined with I/O, ◦ Condensed I/O: iotx ◦ Initiator/Target: tresp	• If Separate Messaging port is used: msgiresp • If Messages are combined with I/O, ◦ Condensed I/O: iorx ◦ Initiator/Target: iresp	Response to a MESSAGE transaction. Does not contain a data payload.
MAITNENANCE READ REQUEST	1000	0000	• If AXI4-Lite Maintenance port is used: maintr_ar • If Direct Maintenance Access is selected: maintreq	• If AXI4-Lite Maintenance port is used, received transactions are handled by the core. • If Direct Maintenance Access is selected: maintrx	Configuration space read transaction. Does not contain a data payload. Results in MAINTENANCE READ RESPONSE.
MAITNENANCE WRITE REQUEST	1000	0001	• If AXI4-Lite Maintenance port is used: maintr_w and maintr_aw • If Direct Maintenance Access is selected: maintreq	• If AXI4-Lite Maintenance port is used, received transactions are handled by the core. • If Direct Maintenance Access is selected: maintrx	Configuration space write transaction. Contains a data payload. Results in MAINTENANCE WRITE RESPONSE.

Packet Type	FTYPE	TTYPE	Transmit Port	Receive Port	Description
MAITNENANCE READ RESPONSE	1000	0010	• If AXI4-Lite Maintenance port is used, responses are generated by the core. • If Direct Maintenance Access is selected: maintresp	• If AXI4-Lite Maintenance port is used: maintr_r • If Direct Maintenance Access is selected: maintrx	Configuration space read transaction. Does not contain a data payload. Results in MAINTENANCE READ RESPONSE.
MAITNENANCE WRITE RESPONSE	1000	0011	• If AXI4-Lite Maintenance port is used, responses are generated by the core. • If Direct Maintenance Access is selected: maintresp	• If AXI4-Lite Maintenance port is used: maintr_b • If Direct Maintenance Access is selected: maintrx	Configuration space write transaction. Contains a data payload. Results in MAINTENANCE WRITE RESPONSE.

References

1. RapidIO TM Interconnect Specification v2.2
2. LogiCORE IP Serial RapidIO Gen2 Product Guide
   http://www.xilinx.com/support/documentation/ipbusinterfacei-o_rapidio_do-di-rio-log.htm

Revision History

07/15/2012 - Initial release