Setting Global Bus Configuration Registers

Application Note

DWC USB 3.0 Device
DWC USB 3.0 Host
DWC USB 3.0 Dual Role Device

Version 1.10a
November 2020

Contents

Revision History

Date	Version	Description
November 2020	1.10a	No Technical Change, only formatting and editing corrections.
April 2020	1.00a	Initial Version

1 Setting Global Bus Configuration Registers

Note

- The *DesignWare Cores SuperSpeed USB 3.0 Controller Databook* is used as a reference document in this Application Note.
- This application note pertains to 2.90a version of the DWC_usb3 controller. There may be variations in signal, parameter, and register names depending on the DWC_usb3 controller version. For more information, see the DWC_usb3 documentations for up-to-date information pertaining to your version of the product.

This application note provides information on how setting the Global Bus Configuration registers (GSBUSCFG0 and GSBUSCFG1) results in different AXI master requests to read/write data from/to the memory. You can use this application note to determine the appropriate setting that gives the best performance for your system.

For description of the Global Bus Configuration registers, see section “GSBUSCFG0” and “GSBUSCFG1” in the “Registers” chapter of the *DWC SuperSpeed USB 3.0 Controller Databook*.

1.1 Data and Descriptor Read (Tx Transfer)

For each data or descriptor read (Tx transfer), the controller Buffer Management Unit (BMU) issues two outstanding DMA requests. The AXI gasket converts each DMA request into multiple AXI transfers based on the value of the burst length (*arlen*) and the number of outstanding pipelined transfer burst limit (*PipeTransLimit*) set in the GSBUSCFG0 and GSBUSCFG1 registers. For more information, see the “System Bus Interface” section in the “Architecture Details” chapter of the *DWC SuperSpeed USB 3.0 Controller Databook*.

This section discusses about the two examples to understand how the controller issues read requests based on the different values of the GSBUSCFG0 and GSBUSCFG1 registers.

1.1.1 Data and Descriptor Read – Example 1

In this example, consider that you configured the AXI bus width as 64 bits and set the Global Bus Configuration registers as follows:

- GSBUSCFG0 register = 0x0000_0004 (that is, *INCR8 Burst Type Enable* field = 1). This indicates the AXI master uses INCR to do 8-beat burst; the *arlen* can be 8 or lower.
- GSBUSCFG1 register = 0x0000_0700 (that is, *AXI Pipelined Transfers Burst Request Limit (PipeTransLimit)* field = 7 (8 requests))

This means that when the number of requests reaches 8, the AXI master does not make any more requests on the AXI ARADDR bus until the associated data phases are complete.

Assume that each DMA request is for maximum data or TRB payload (for example, 1 KB). The AXI gasket, based on the GSBUSCFG0 and GSBUSCFG1 register settings, could do multiple outstanding AXI requests for each DMA request.

In this example, it takes 16 AXI burst requests to complete a 1 KB read (8 (64-bit data) x 8 (*arlen*=7) x 16 (AXI burst requests) = 1 KB).

The sequence of operation is as follows:

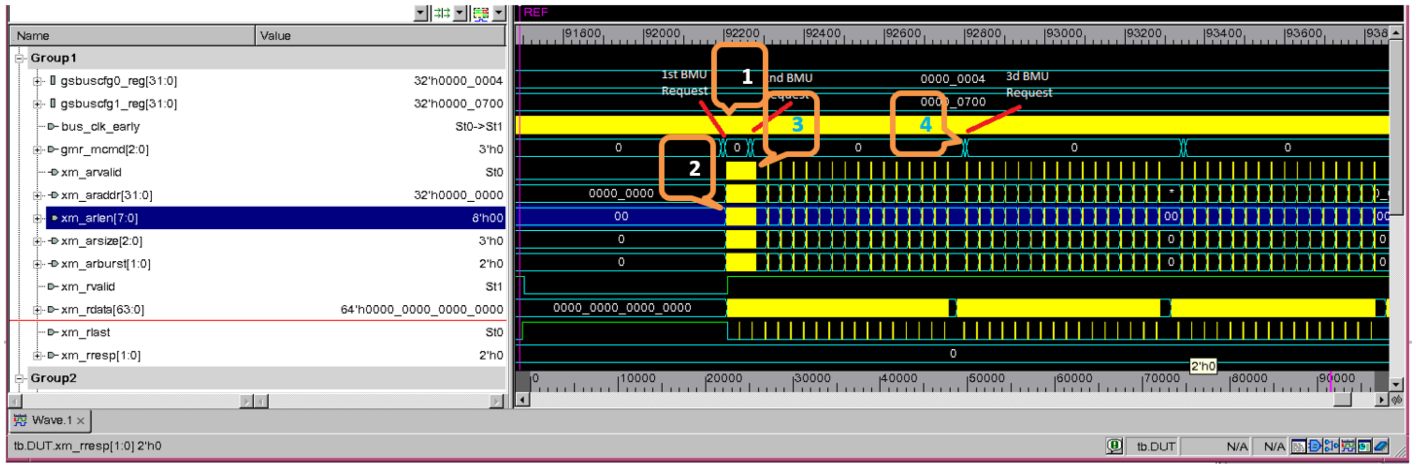
1. The controller starts with two outstanding DMA requests.
2. Because *PipeTransLimit* is set to 7, the AXI gasket issues only 8 AXI outstanding pipelined transfer burst requests instead of 16. This reads only 512 bytes.
3. After all the associated data phases of a burst are completed, the AXI gasket issues the next single AXI burst request. It continues this until the remaining 512 bytes is read.
4. After Step 3 is completed, the controller starts the third DMA request, and so on.

In such a scenario, if your system has large latency on each AXI request, the latency accumulated becomes remarkably large because many single AXI burst requests are needed to complete a transfer.

Note

- The initial system latency for AXI request is not mentioned in this section because it depends on your system.
- In Figure 1-1 and Figure 1-2, the *gmr_mcmd* signal = 1 indicates a DMA request.

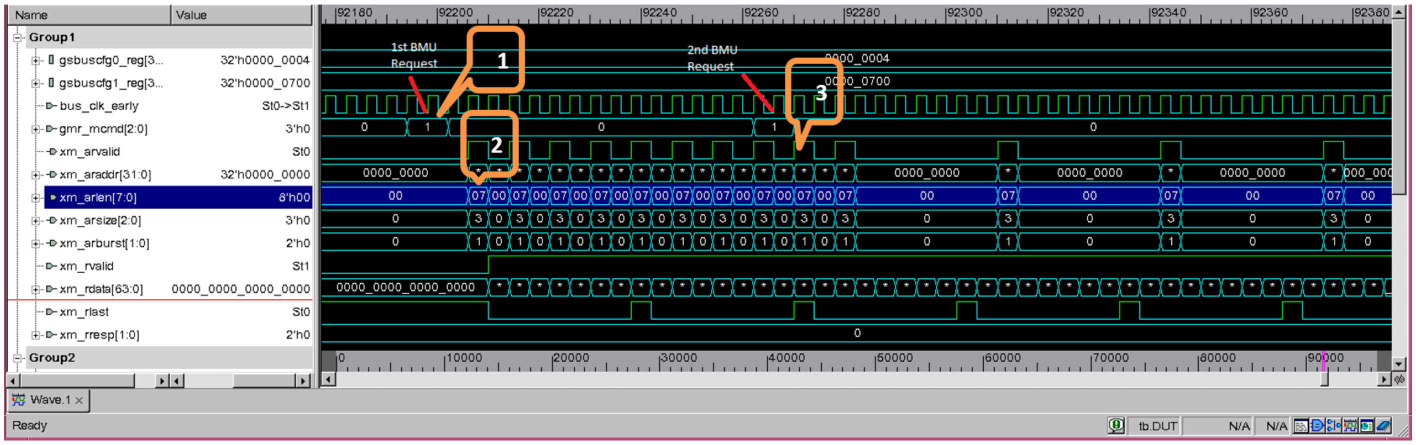
Figure 1-1 Data and Descriptor Read – Timing Diagram for Example 1



Name	Value
Group1	
gsbuscfg0_reg[31:0]	32'h0000_0004
gsbuscfg1_reg[31:0]	32'h0000_0700
bus_clk_early	St0->St1
gmr_mcmd[2:0]	3'h0
xm_arvalid	St0
xm_araddr[31:0]	32'h0000_0000
xm_arlen[7:0]	8'h00
xm_arsize[2:0]	3'h0
xm_arburst[1:0]	2'h0
xm_rvalid	St1
xm_rdata[63:0]	64'h0000_0000_0000_0000
xm_rlast	St0
xm_rresp[1:0]	2'h0
Group2	

tb.DUT.xm_rresp[1:0] 2'h0

Figure 1-2 Data and Descriptor Read – Timing Diagram for Example 1 (Zoomed in)



Name	Value
Group1	
gsbuscfg0_reg[3...	32'h0000_0004
gsbuscfg1_reg[3...	32'h0000_0700
bus_clk_early	St0->St1
gmr_mcmd[2:0]	3'h0
xm_arvalid	St0
xm_araddr[31:0]	32'h0000_0000
xm_arlen[7:0]	8'h00
xm_arsize[2:0]	3'h0
xm_arburst[1:0]	2'h0
xm_rvalid	St1
xm_rdata[63:0]	0000_0000_0000_0000
xm_rlast	St0
xm_rresp[1:0]	2'h0
Group2	

Ready

1.1.2 Data and Descriptor Read – Example 2

In this example, consider that you configured the AXI bus width as 64 bits and set the Global Bus Configuration registers as follows:

- GSBUSCFG0 register = 0x0000_000f (that is, INCR, INCR4, INCR8, and INCR16 *Burst Type Enable* fields are set to 1). This indicates the AXI master uses any length less than or equal to the largest-enabled burst length of INCR4/8/16; the *arlen* can be 15 or lower.
- GSBUSCFG1 register = 0x0000_0f00 (that is, *AXI Pipelined Transfer Burst Request Limit (PipeTransLimit)* field = 'hf (16 requests)). This means that when the number of requests reaches 16, the AXI master does not make any more requests on the AXI ARADDR bus until the associated data phases are completed.

Assume that each DMA request is for maximum data or TRB payload (for example, 1 KB). The AXI gasket, based on the GSBUSCFG0 and GSBUSCFG1 register settings, could do multiple outstanding AXI requests for each DMA request.

In this example, it takes 8 AXI burst requests to complete a 1 KB read (8 (64-bit data) x 16 (*arlen*=16) x 8 (AXI burst requests) = 1 KB)

The sequence of operation is as follows:

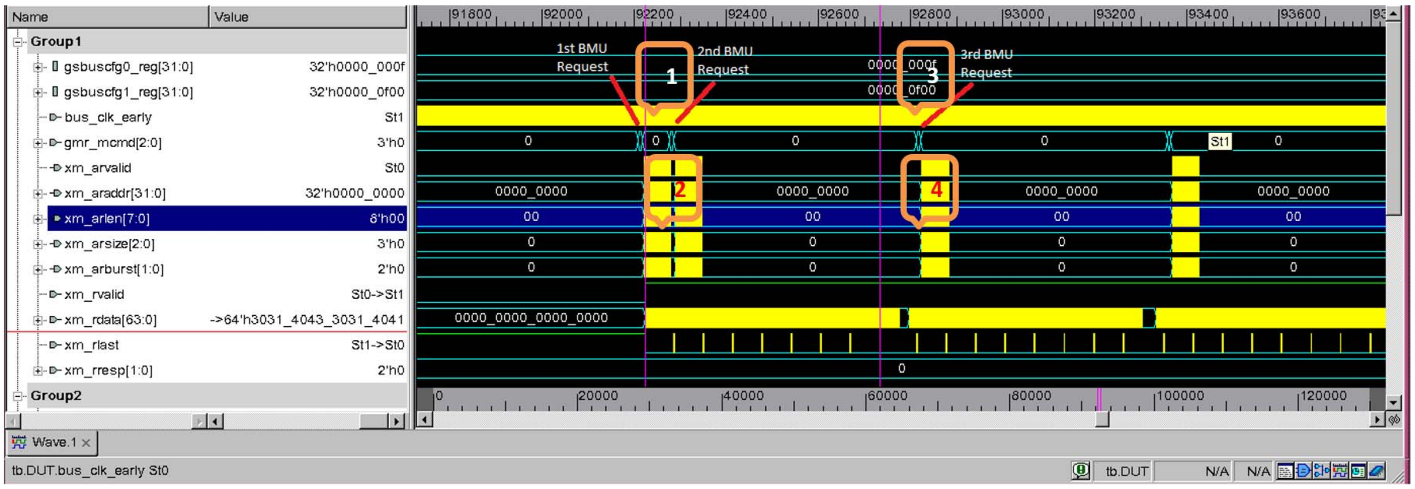
1. The controller starts with 2 outstanding DMA requests.
2. Because *PipeTransLimit* is set to 15, the AXI gasket issues 16 AXI outstanding pipelined transfer burst requests (without waiting for any data transfer to be completed) to read 2KB data.
3. As soon as 1 KB data is read, the controller starts the third DMA request.
4. The AXI gasket issues 8 AXI outstanding pipelined transfer burst requests to read 1 KB, and so on.

In such a scenario, because there are 16 AXI outstanding pipelines and maximum data length is 16 (*arlen*=16), the controller can do a burst of AXI outstanding pipelines to complete 1 KB or 2 KB transfer. Therefore, if your system has large initial delay, this setting is more beneficial as the controller performs only single burst of AXI outstanding pipelines to complete the 1 KB or 2 KB transfer.

Note

- The initial system latency for AXI request is not mentioned in this section because it depends on your system.
- In Figure 1-3 and Figure 1-4, the *gmr_mcmd* signal = 1 indicates a DMA request.

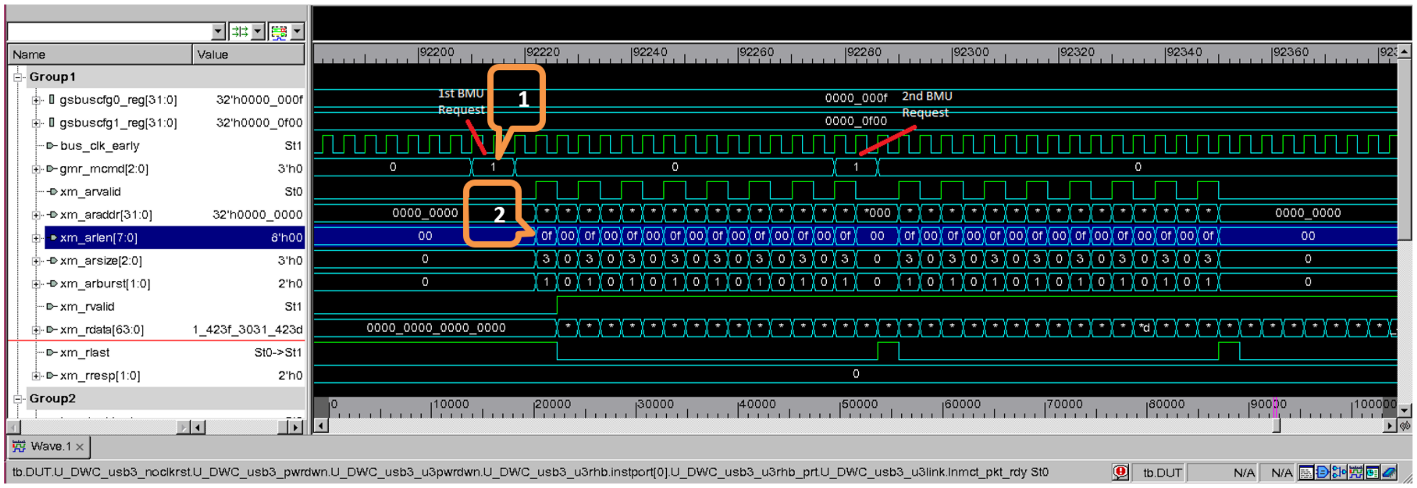
Figure 1-3 Data and Descriptor Read – Timing Diagram for Example 2



Name	Value
Group1	
gsbuscfg0_reg[31:0]	32'h0000_000f
gsbuscfg1_reg[31:0]	32'h0000_0f00
bus_clk_early	St1
gmr_mcmd[2:0]	3'h0
xm_arvalid	St0
xm_araddr[31:0]	32'h0000_0000
xm_arlen[7:0]	8'h00
xm_arsize[2:0]	3'h0
xm_arburst[1:0]	2'h0
xm_rvalid	St0->St1
xm_rdata[63:0]	->64'h3031_4043_3031_4041
xm_rlast	St1->St0
xm_rresp[1:0]	2'h0
Group2	

tb.DUT.bus_clk_early St0

Figure 1-4 Data and Descriptor Read – Timing Diagram for Example 2 (Zoomed in)



Name	Value
Group1	
gsbuscfg0_reg[31:0]	32'h0000_000f
gsbuscfg1_reg[31:0]	32'h0000_0f00
bus_clk_early	St1
gmr_mcmd[2:0]	3'h0
xm_arvalid	St0
xm_araddr[31:0]	32'h0000_0000
xm_arlen[7:0]	8'h00
xm_arsize[2:0]	3'h0
xm_arburst[1:0]	2'h0
xm_rvalid	St1
xm_rdata[63:0]	1_423f_3031_423d
xm_rlast	St0->St1
xm_rresp[1:0]	2'h0
Group2	

tb.DUT.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrdwn.U_DWC_usb3_u3pwrdwn.U_DWC_usb3_u3rhb.instport[0].U_DWC_usb3_u3rhb_prt.U_DWC_usb3_u3link.lnmct_pkt_rdy St0

1.2 Data and Descriptor Write (Rx Transfer)

For data/descriptor writes (Rx transfer), the controller BMU issues only one outstanding DMA request. The AXI gasket converts this request into multiple AXI transfers based on the value of the burst length (*awlen*) and the number of outstanding pipelined transfer burst limit (*PipeTransLimit*) set in the GSBUSCFG0 and GSBUSCFG1 registers. For more information, see the section "System Bus Interface" in the "Architecture Details" chapter of the *USB 3.0 Controller Databook*.

This section discusses about the two examples to understand how the controller issues write requests based on different values of the GSBUSCFG0 and GSBUSCFG1 registers.

1.2.1 Data and Descriptor Write – Example 1

In this example, consider that you configured the AXI bus width as 64 bits and set the Global Bus Configuration registers as follows:

- GSBUSCFG0 register = 0x0000_0004 (that is, *INCR8 Burst Type Enable* field = 1)

This indicates the AXI master uses INCR to do 8-beat burst; the *awlen* can be 8 or lower.

- GSBUSCFG1 register = 0x0000_0700 (that is, *AXI Pipelined Transfer Burst Request Limit (PipeTransLimit)* field = 7 (8 requests))

  This means that when the number of requests made reaches 8, the AXI master does not make any more requests on the AXI AWADDR bus until the associated data phases are completed.

Assume that each DMA request is for maximum data or TRB payload (for example, 1 KB). The AXI gasket, based on the GSBUSCFG0 and GSBUSCFG1 register settings, could do multiple outstanding AXI requests for each DMA request.

In this example, it takes 16 AXI burst requests to complete a 1 KB read (8 (64-bit data) x 8 (*awlen*=7) x 16 (AXI burst requests) = 1 KB).

The sequence of operation is as follows:

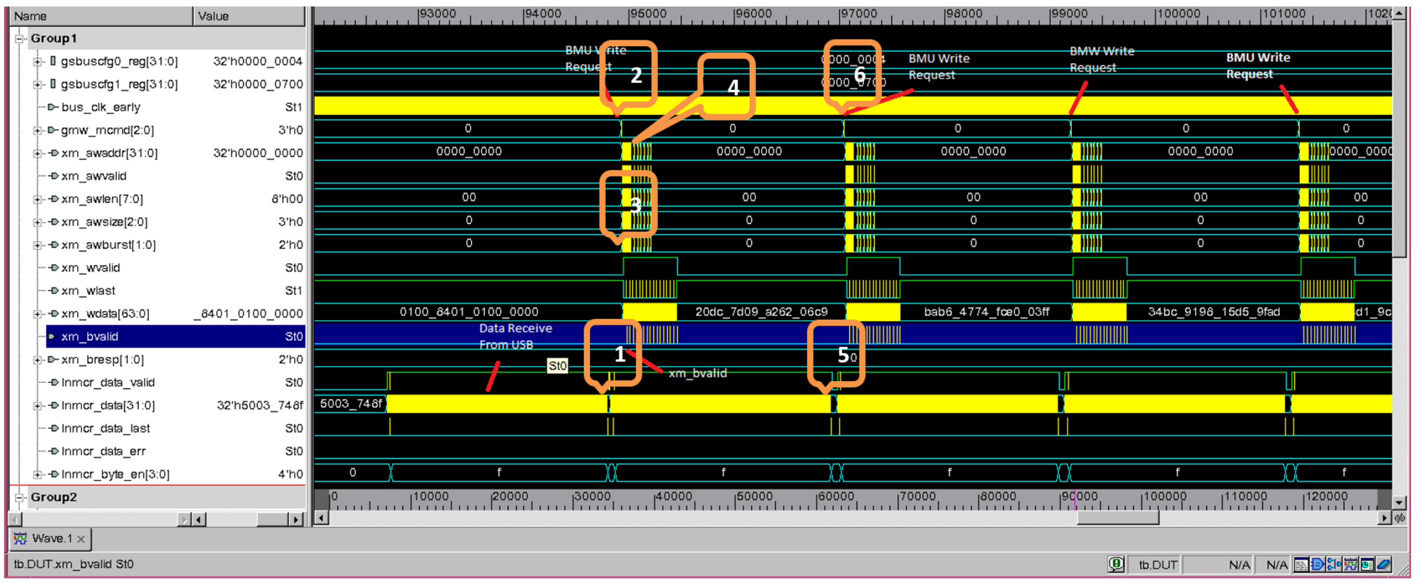
1. The controller receives the first 1 KB packet on the USB bus.
2. The controller starts DMA write request for 1 KB.
3. Because *PipeTransLimit* is set to 7, the AXI gasket issues only 8 AXI outstanding pipelined transfer burst requests instead of 16. This writes only 512 bytes.
4. After all the associated data phases of a burst are completed, the AXI gasket issues the next single AXI burst request. It continues this until the remaining 512 bytes is written.

   Note that because the AXI bus is much faster than the USB, the 1 KB AXI transfer is completed way before the next packet arrives on the USB bus.
5. The controller receives the second 1 KB packet on the USB bus.
6. The controller starts the next DMA request, and so on.

Note

- The initial system latency for AXI request is not mentioned in this section because it depends on your system.
- In Figure 1-5 and Figure 1-6, the *gmw_mcmd* signal = 1 indicates a DMA request.

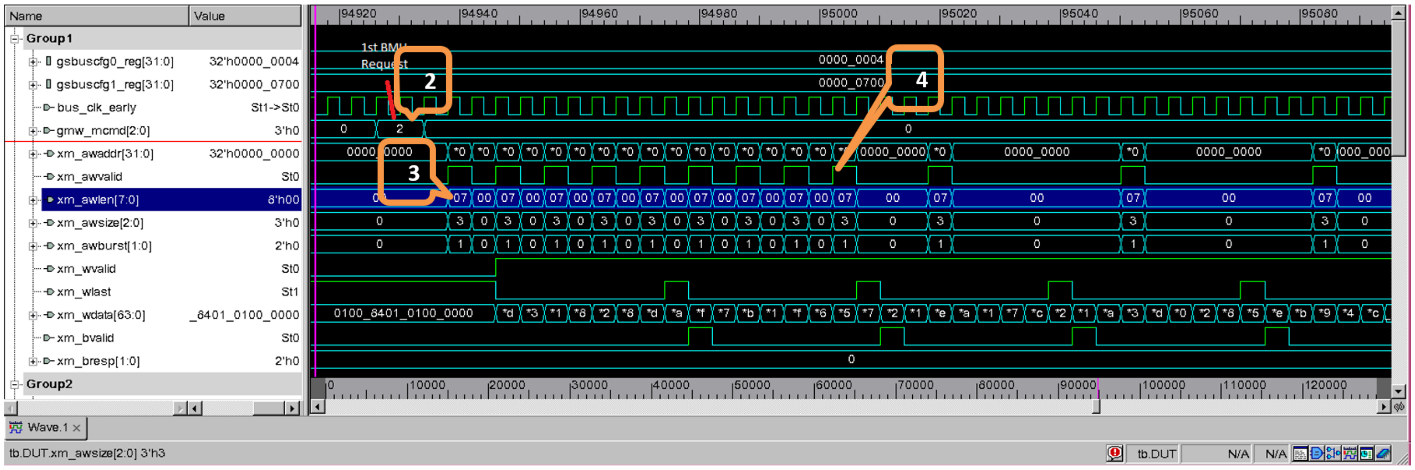
Figure 1-5 Data and Descriptor Write – Timing Diagram for Example 1



Name	Value
Group1	
gsbuscfg0_reg[31:0]	32'h0000_0004
gsbuscfg1_reg[31:0]	32'h0000_0700
bus_clk_early	St1
gmw_mcmd[2:0]	3'h0
xm_awaddr[31:0]	32'h0000_0000
xm_awvalid	St0
xm_awlen[7:0]	8'h00
xm_awsize[2:0]	3'h0
xm_awburst[1:0]	2'h0
xm_wvalid	St0
xm_wlast	St1
xm_wdata[63:0]	_8401_0100_0000
xm_bvalid	St0
xm_bresp[1:0]	2'h0
lnmcr_data_valid	St0
lnmcr_data[31:0]	32'h5003_748f
lnmcr_data_last	St0
lnmcr_data_err	St0
lnmcr_byte_en[3:0]	4'h0
Group2	

tb.DUT.xm_bvalid St0

Figure 1-6 Data and Descriptor Write – Timing Diagram for Example 1 (Zoomed in)



Name	Value
Group1	
gsbuscfg0_reg[31:0]	32'h0000_0004
gsbuscfg1_reg[31:0]	32'h0000_0700
bus_clk_early	St1->St0
gmw_mcmd[2:0]	3'h0
xm_awaddr[31:0]	32'h0000_0000
xm_awvalid	St0
xm_awlen[7:0]	8'h00
xm_awsize[2:0]	3'h0
xm_awburst[1:0]	2'h0
xm_wvalid	St0
xm_wlast	St1
xm_wdata[63:0]	_8401_0100_0000
xm_bvalid	St0
xm_bresp[1:0]	2'h0
Group2	

tb.DUT.xm_awsize[2:0] 3'h3

1.2.2 Data and Descriptor Write – Example 2 (Non-Posted/Non-Bufferable Writes versus Posted/Bufferable Writes)

This example demonstrates the difference between non-posted/non-bufferable write and posted/bufferable write. The posted/bufferable write feature is useful for systems that has large initial latency for bus request.

1.2.2.1 Non-Posted/Non-Bufferable Write

In the non-posted/non-bufferable write, the controller starts the next AXI/DMA request only after receiving a high on the *xm_bvalid* signal for the previous transfer.

In this example, consider that you configured the AXI bus width as 64 bits and set the Global Bus Configuration registers as follows:

- GSBUSCFG0 register = 0x0000_0001 (that is, *Undefined Length INCR Burst Type Enable* field = 1, and *DATWRREQINFO* field = 0)

  This indicates that the *awlen* can be 16 or lower, and it is a Non-Posted/Non-Bufferable Write (for more information, see Table "Cache Type Bit Assignment" in the *USB 3.0 Controller Databook*).

- GSBUSCFG1 register = 0x0000_0f00 (that is, *AXI Pipelined Transfer Burst Request Limit (PipeTransLimit)* field = 'hf (16 requests))

The sequence of operation is as follows:

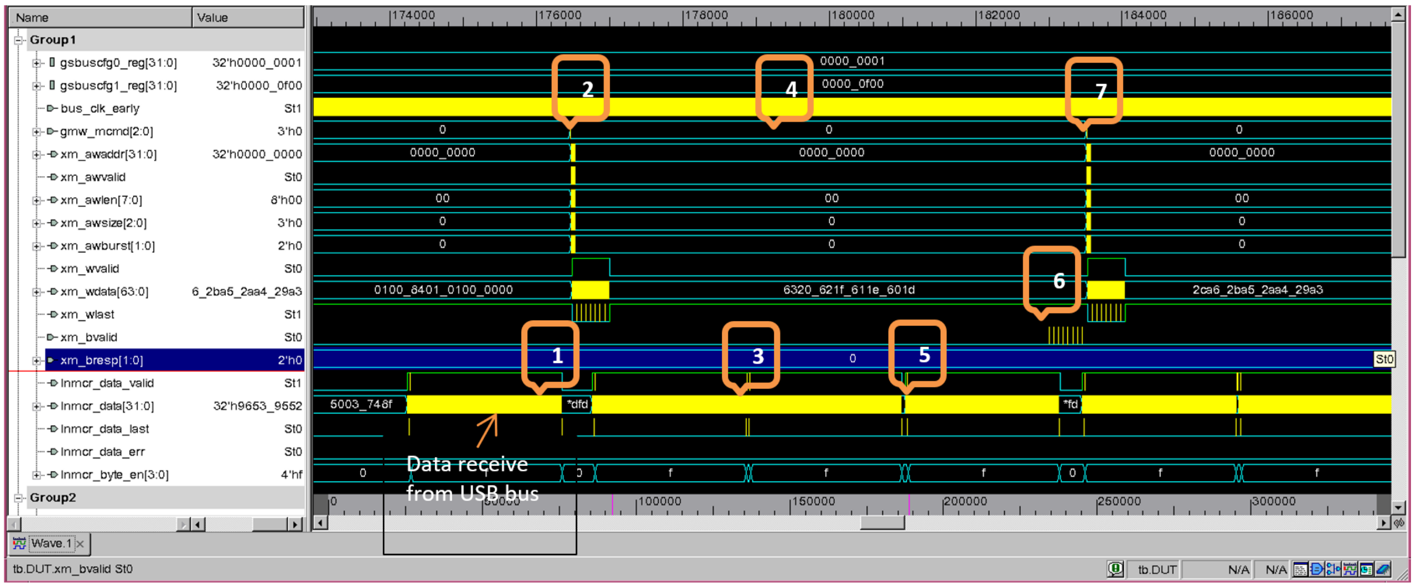
1. The controller receives the first 1 KB packet on USB bus.
2. The controller starts a DMA write request for 1 KB (8 (64-bit data) x16 (*awlen*=15) x 8 (AXI burst requests)).

   Note that unlike the read request, not all 16 AXI burst requests are enabled in the beginning because this is an IN transaction.
3. The controller receives second 1 KB packet on USB bus.
4. The controller does not start the next DMA write request because it is waiting for the *xm_bvalid* signal to become high.
5. The controller receives third 1 KB packet on USB bus.
6. The *xm_bvalid* signal goes high.
7. The controller starts a DMA write request for the second 1 KB packet, and so on.

Note

- The initial system latency for AXI request is not mentioned in this section because it depends on your system.
- In Figure 1-7, the *gmw_mcmd* signal = 1 indicates a DMA request.

Figure 1-7 Data and Descriptor Write – Timing Diagram for Example 2 (Non-Posted/Non-Bufferable Write)



1.2.2.2 Posted/ Bufferable Write

In the posted/bufferable write, the controller starts a new DMA transfer without waiting for the previous DMA transfer to be completed. The controller starts the next AXI transfer based on the number of AXI outstanding pipelines available before waiting for the *xm_bvalid* signal to become high for the previous transfer.

In this example, consider that you configured the AXI bus width as 64 bits and set the Global Bus Configuration registers as follows:

- GSBUSCFG0 register = 0x0010_0001 (that is, *Undefined Length INCR Burst Type Enable* field = 1, and *DATWRREQINFO* field = 1)

  This indicates that the *awlen* can be 16 or lower, and it is a Posted/Bufferable Write (see Table "Cache Type Bit Assignment" in the *Databook*.)

- GSBUSCFG1 register = 0x0000_0f00 (that is, *AXI Pipelined Transfer Burst Request Limit (PipeTransLimit)* field = 'hf (16 requests))

The sequence of operation is as follows:

1. The controller receives the first 1 KB packet on the USB bus.
2. The controller starts a DMA write request for 1 KB (8 (64-bit data) x16 (*awlen*=15) x 8 (AXI burst requests))
3. The controller receives the second 1 KB packet on the USB bus.
4. The controller starts a DMA write request for the second 1 KB (8 AXI burst requests) without waiting for the *xm_bvalid* signal to become high because the posted/bufferable write option is enabled and 8 AXI pipeline requests are still available.
5. The controller receives the third 1 KB packet on the USB bus.

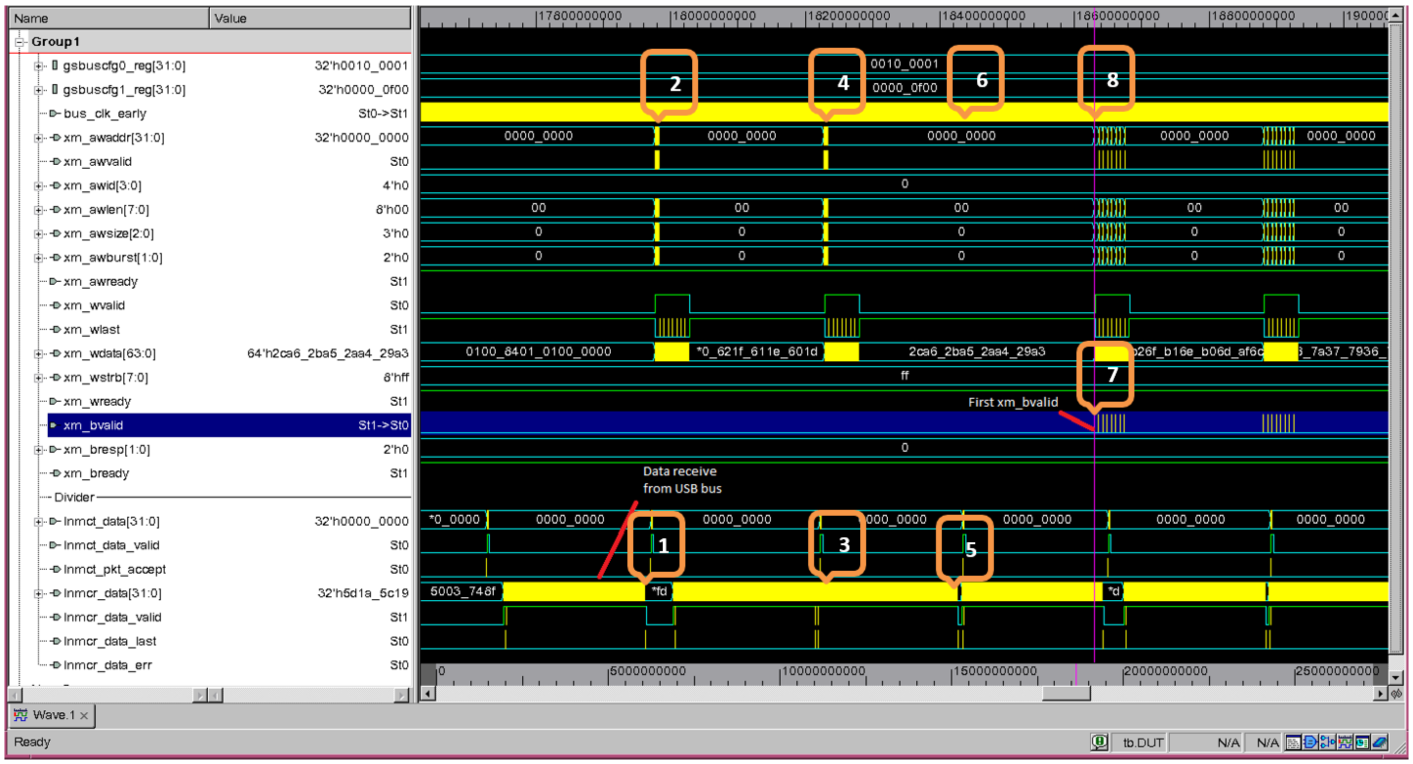
6. The controller does not start another DMA requests because all 16 AXI outstanding pipelines have been used.
7. The controller receives a high on the *xm_bvalid* signal.
8. The controller issues DMA write requests.

For posted/bufferable write with initial system delay, the controller can transfer three packets after receiving four packets from the USB bus. Compared to the non-posted/non-bufferable write with same initial system delay, the controller can only transfer two packets after receiving four packets from the USB bus.

Note

- The initial system latency for AXI request is not mentioned in this section because it depends on your system.
- In Figure 1-8, the *gmw_mcmd* signal = 1 indicates a DMA request.

Figure 1-8 Data and Descriptor Write – Timing Diagram for Example 2 (Posted/Bufferable Write)



Name	Value
Group1	
gsbuscfg0_reg[31:0]	32'h0010_0001
gsbuscfg1_reg[31:0]	32'h0000_0f00
bus_clk_early	St0->St1
xm_awaddr[31:0]	32'h0000_0000
xm_awvalid	St0
xm_awid[3:0]	4'h0
xm_awlen[7:0]	8'h00
xm_awsize[2:0]	3'h0
xm_awburst[1:0]	2'h0
xm_awready	St1
xm_wvalid	St0
xm_wlast	St1
xm_wdata[63:0]	64'h2ca6_2ba5_2aa4_29a3
xm_wstrb[7:0]	8'hff
xm_wready	St1
xm_bvalid	St1->St0
xm_bresp[1:0]	2'h0
xm_bready	St1
Divider	
lnmct_data[31:0]	32'h0000_0000
lnmct_data_valid	St0
lnmct_pkt_accept	St0
lnmcr_data[31:0]	32'h5d1a_5c19
lnmcr_data_valid	St1
lnmcr_data_last	St0
lnmcr_data_err	St0