

# Introduction to Zynq Hardware

## Lab 6

### Improving Data flow between PL and PS utilizing PS DMA



June 2017  
Version 11

## Lab 6 Overview

In the last lab, we expanded our block design by extending our memory space with a PL-based Block RAM (BRAM). The BRAM can be used to buffer data going between the PS and PL. In this lab, we'll add a software application that enables the PS DMA engine to show how efficiency gains can be achieved when passing data between the PL-based BRAM and external DDR3 memory.

## Lab 6 Objectives

When you have completed Lab 6, you will know how to do the following:

- Create a new C Software Application and import C source code
- Use PS DMA and GIC controllers
- Perform DMA operations using the PS DMA

## Experiment 1: Test the PS DMA Controller

This experiment shows how the PS DMA Controller can improve data transactions between the PS and PL.

### Experiment 1 General Instruction:

Create a new C application and import C-code (dma\_test) to test DMA transaction.

### Experiment 1 Step-by-Step Instructions:

1. <Optional> If you did not complete Lab 5 or wish to start with a clean copy, delete the ZynqDesign and SDK\_Workspace folders in the ZynqHW/2017\_1 folder. Then unzip **Solutions\ZynqHW\_Lab5\_Solution.zip** to the 2017\_1 folder. If you have 7-zip installed, you can do this by right-clicking and dragging **ZynqHW\_Lab5\_Solution.zip** to the 2017\_1 folder. Select **7-Zip → Extract Here**. (If the BSP does not build, try cleaning the SDK workspace (Project → Clean.)
2. In Vivado, **Open Implemented Design** if not already open (under **Flow Navigator → Implementation**). Click **Reload** if necessary.
3. **Export Hardware** (File → Export → Export Hardware) and select location **ZynqDesign.lab6**. Make sure you export the bitstream as well as we have IP in the PL now. Click **OK**.

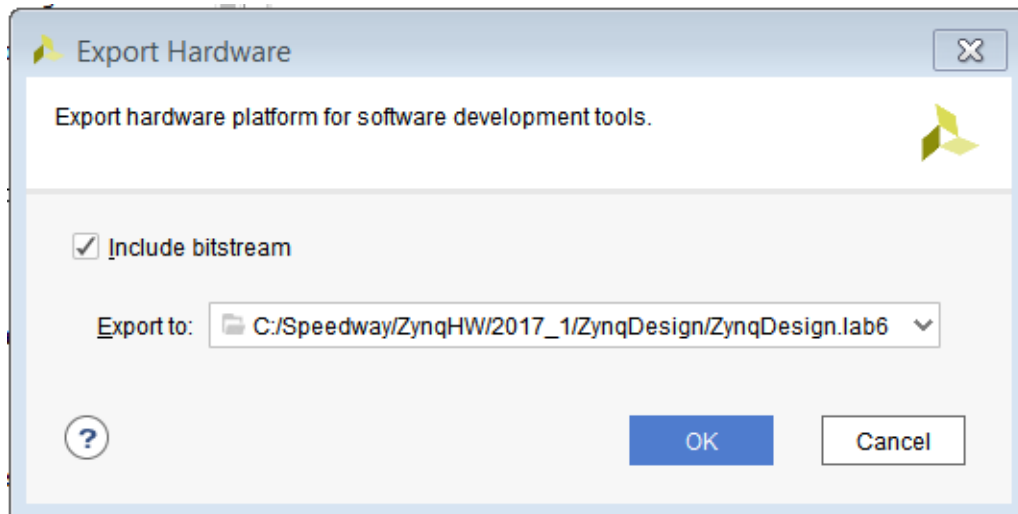
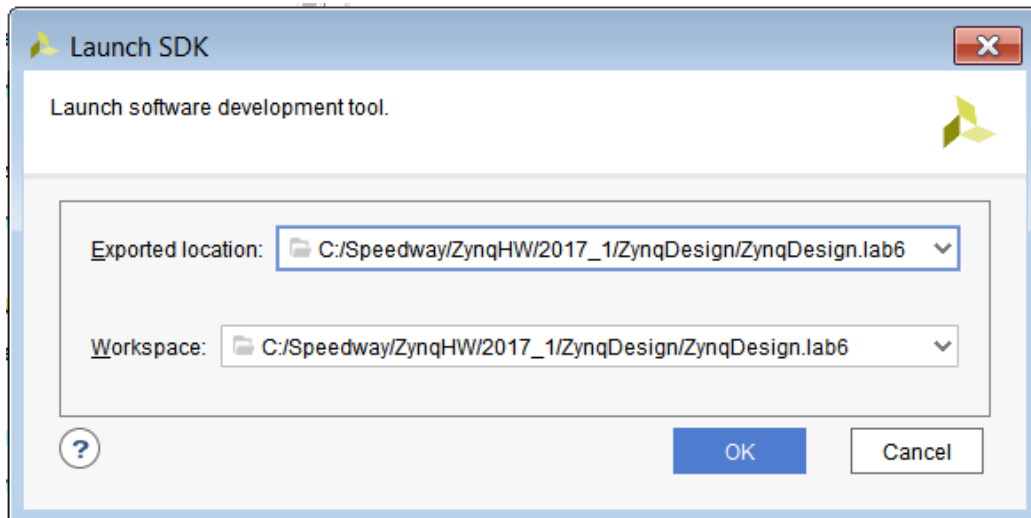


Figure 1 - New SDK Workspace

4. Select **File → Launch SDK**. Change the *Exported location* and *Workspace* to the new ZynqDesign.lab6 location. Click **OK**.



**Figure 2 – Launch SDK**

Creating a new workspace means our previously created BSP and software applications will not appear in this workspace. However, controlling your workspace and what gets imported and updated is worth it. We will now re-create the standalone BSP (very important with a new hardware platform!) and import the applications that we previously created.

5. Create the standalone BSP using **File → New → Board Support Package**.
6. Now import the previously created applications. Select **File → Import → General → Existing Projects into Workspace**.

7. Browse to the ZynqDesign.lab3 directory and select **OK**. Make sure the three applications are checked. Check the box to **Copy projects into workspace**. Then click **Finish**.

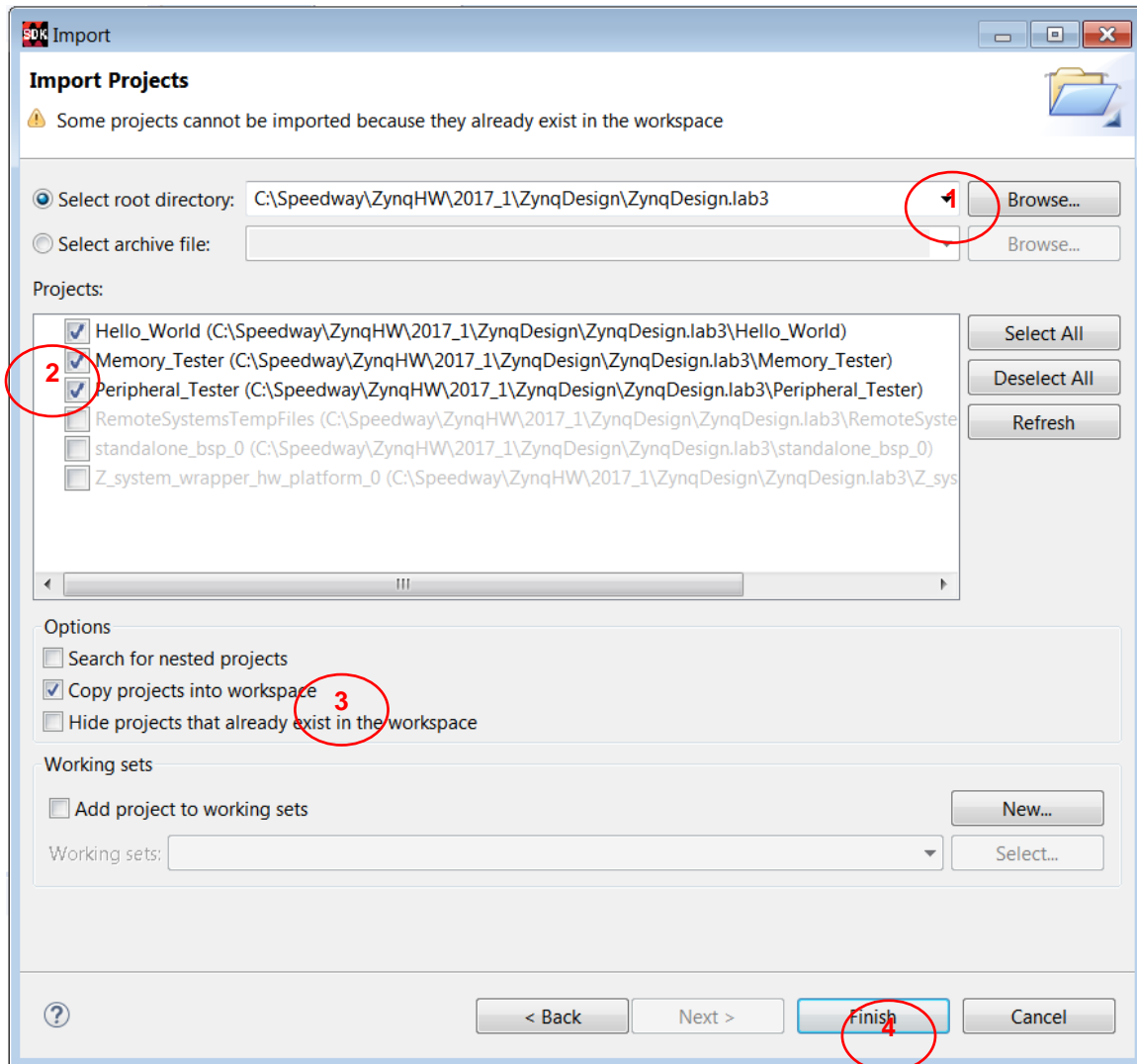


Figure 3 – Import Previous Applications

8. Verify your Hardware Platform changes in SDK. View the **system.hdf** file. If not open already, expand *HW\_platform\_0*, open **system.hdf** and scroll down to view IP listing. Verify the IP versions were loaded for the BRAM (IP versions may be in different order). If not, close SDK and export the hardware platform from Vivado again.

#### IP blocks present in the design

ps7_intc_dist_0	ps7_intc_dist	1.00.a
axi_bram_ctrl_0_bram	blk_mem_gen	8.3
ps7_gpio_0	ps7_gpio	1.00.a
ps7_scutimer_0	ps7_scutimer	1.00.a
ps7_slcr_0	ps7_slcr	1.00.a
ps7_scuwdt_0	ps7_scuwdt	1.00.a
ps7_l2cachec_0	ps7_l2cachec	1.00.a
ps7_scuc_0	ps7_scuc	1.00.a
ps7_qspi_linear_0	ps7_qspi_linear	1.00.a
ps7_m_axi_gp0	ps7_m_axi_gp	1.00.a
ps7_pmu_0	ps7_pmu	1.00.a
axi_bram_ctrl_0	axi_bram_ctrl	4.0
ps7_afi_1	ps7_afi	1.00.a
ps7_qspi_0	ps7_qspi	1.00.a
ps7_usb_0	ps7_usb	1.00.a
ps7_afi_0	ps7_afi	1.00.a
ps7_afi_3	ps7_afi	1.00.a
ps7_axi_interconnect_0	ps7_axi_interconnect	1.00.a
ps7_globaltimer_0	ps7_globaltimer	1.00.a
ps7_afi_2	ps7_afi	1.00.a
ps7_dma_s	ps7_dma	1.00.a
ps7_xadc_0	ps7_xadc	1.00.a
ps7_iop_bus_config_0	ps7_iop_bus_config	1.00.a
axi mem intercon	axi interconnect	2.1

IP Version

**Figure 4 - Hardware Platform – IP Version for BRAM**

9. To test our BRAM, software code is provided; however a C application project needs to be created first. Create a new application project (**File → New → Application Project**).
10. Enter **BRAM\_DMA\_Test** for the *Project name* and select **Use existing** for the BSP. Click **Next >**.

11. Select **Empty Application**, click **Finish**.
12. In the Project Explorer Window, expand **BRAM\_DMA\_TEST**, right-click on **src** and choose **Import**.

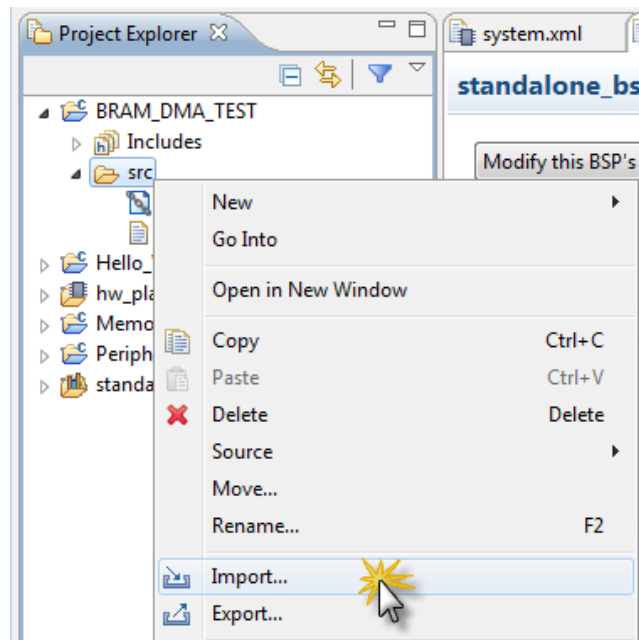


Figure 5 - Import code source

13. In Import window, expand **General**, select **File System**, then click **Next >**.

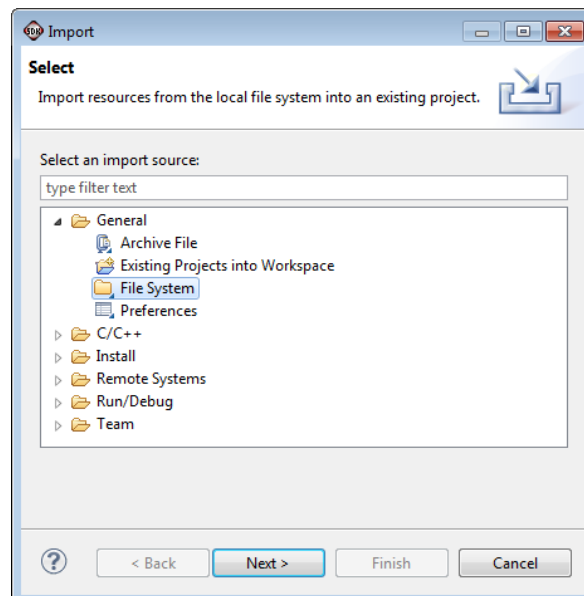
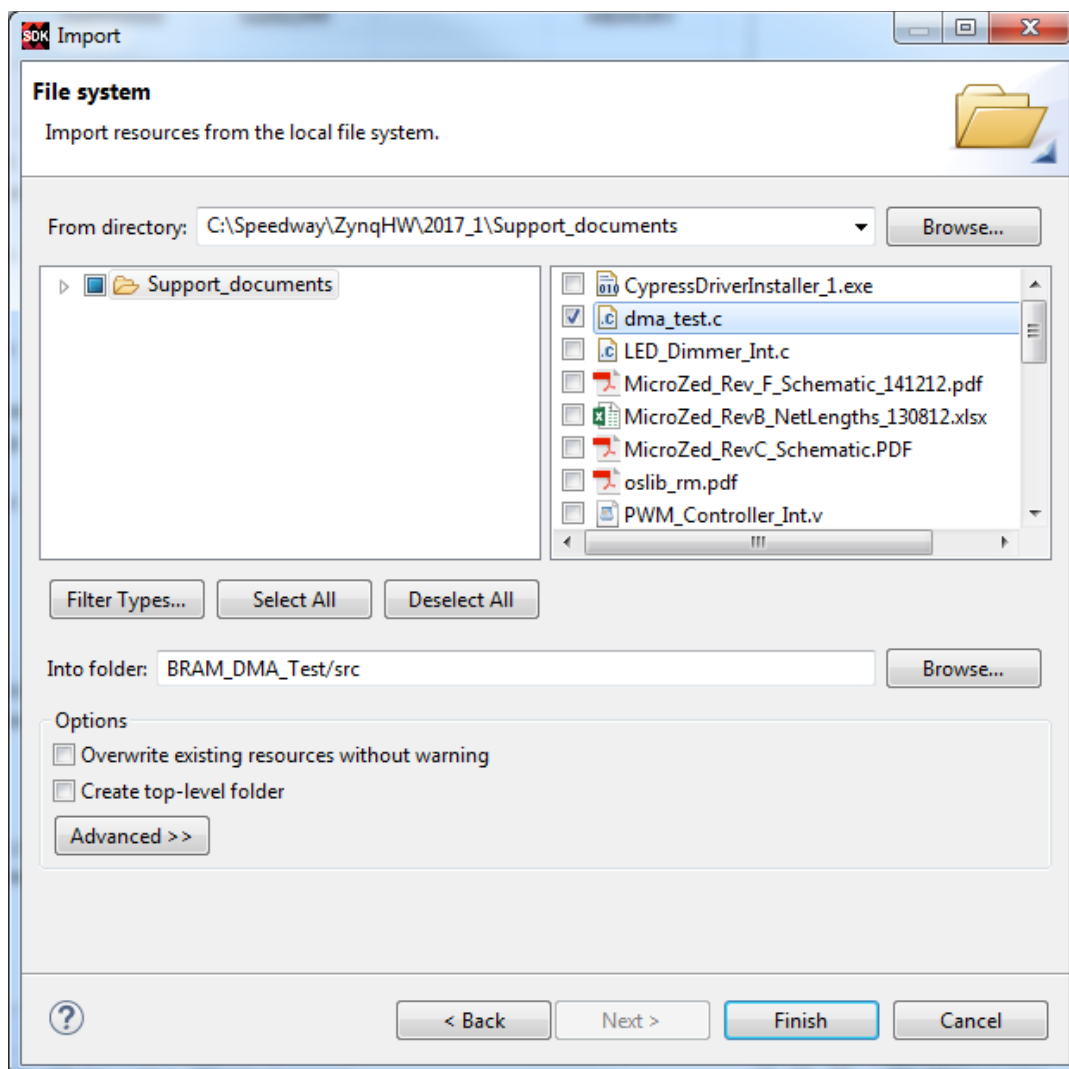


Figure 6 - Import File System

14. Click the Browse button and navigate to the support documents directory, **C:\Speedway\ZynqHW2017\_1\Support\_documents**. Select the checkbox next to **dma\_test.c**, then click **Finish**.



**Figure 7 - Import C Source**

Immediately the application will start building, when complete you should see the result in the Console.

```
CDT Build Console [BRAM_DMA_Test]

'Invoking: ARM v7 Print Size'
arm-none-eabi-size BRAM_DMA_Test.elf |tee "BRAM_DMA_Test.elf.size"
  text    data     bss     dec     hex filename
40016    2764    25160    67940   10964 BRAM DMA Test.elf
'Finished building: BRAM_DMA_Test.elf.size'
.

14:22:24 Build Finished (took 2s.581ms)
```

**Figure 8 - Application Build Results**



15. Now that we are utilizing the PL, it must be programmed. Click the **Program FPGA** button on the top shortcut bar:



Figure 9 - Program FPGA

16. The default program options are OK. Click **Program**.

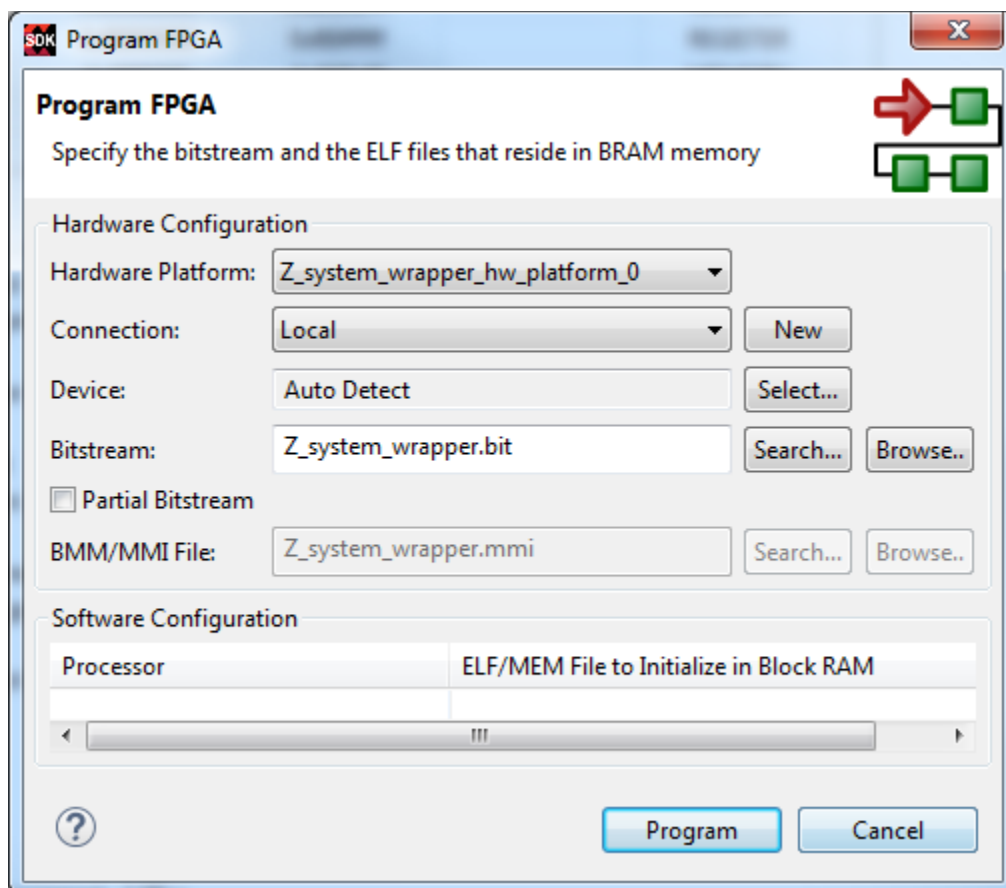
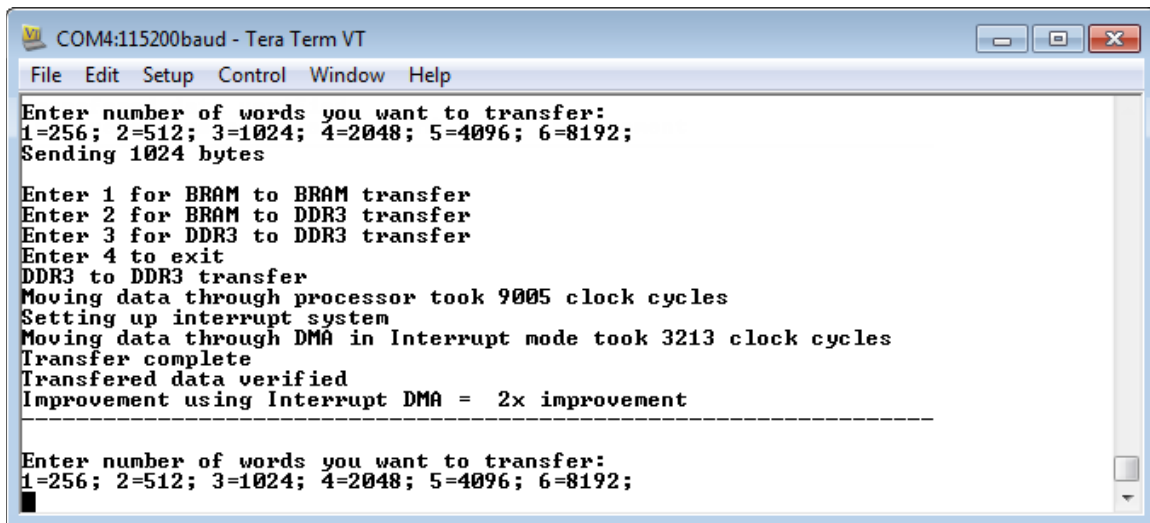


Figure 10 - Program FPGA Options

17. Open Terminal, such as **Tera Term**, and set the **COM port** to active COM setting for your board and set the **Baud Rate at 115,200**.

18. Right-click on **BRAM\_DMA\_Test** and select **Run As → Launch on Hardware (System Debugger)**.
19. Once the code is downloaded to the board, TeraTerm should show the burst size prompt. Explore different byte sizes and transfer modes to compare the improvement using the DMA.



```
COM4:115200baud - Tera Term VT
File Edit Setup Control Window Help
Enter number of words you want to transfer:
1=256; 2=512; 3=1024; 4=2048; 5=4096; 6=8192;
Sending 1024 bytes

Enter 1 for BRAM to BRAM transfer
Enter 2 for BRAM to DDR3 transfer
Enter 3 for DDR3 to DDR3 transfer
Enter 4 to exit
DDR3 to DDR3 transfer
Moving data through processor took 9005 clock cycles
Setting up interrupt system
Moving data through DMA in Interrupt mode took 3213 clock cycles
Transfer complete
Transferred data verified
Improvement using Interrupt DMA = 2x improvement
-----
Enter number of words you want to transfer:
1=256; 2=512; 3=1024; 4=2048; 5=4096; 6=8192;
█
```

Figure 11 - BRAM\_DMA\_TEST running on Terminal

This lab led you through exercising block RAM connected to a processor system so that you can see how to perform DMA transfers within a memory or between two different kinds of memory. You verified the functionality by using the provided application. You observed that DMA is considerably improved for block RAM compared to DDR because the transactions to block RAM are across the AXI interconnect and block RAM is operating at a slower speed.

20. Continue to experiment with DMA in the terminal window in answering the questions below.

### Questions:

**Answer the following questions:**

- Was a new BSP required for this application?  
\_\_\_\_\_  
\_\_\_\_\_
- What is the performance increase when transferring 8192 bytes from BRAM to DDR3? Try again from DDR3 to DDR3? BRAM to BRAM?  
\_\_\_\_\_
- What is the performance increase when transferring 256 bytes from BRAM to DDR3? Try again from DDR3 to DDR3? BRAM to BRAM?  
\_\_\_\_\_

21. When done with the questions, **Close SDK**.

## Exploring Further

If you have more time and would like to investigate more...

- Explore the provided C source code. Examine the PS DMA Configuration.

This concludes Lab 6.

## Revision History

Date	Version	Revision
6 Nov 13	02	Initial Draft
19 Nov 13	03	Pilot updates
6 Nov 14	04	Updated for Vivado 2014.3
5 Jan 15	05	Updated for Vivado 2014.4
06 Mar 15	06	Finalize for Vivado 2014.4. Update dma_test.c to use valid xil_types and also for BRAM xparameters.h bug in 2014.4
16 Mar 15	07	Minor edits for release
Oct 15	08	Updated to Vivado 2015.2
July 2016	09	Updated to Vivado 2016.2
May 2017	10	Updated to Vivado 2017.1
June 2017	11	Updated MiniZed to Vivado 2017.1 + Rebranding

## Resources

[www.minized.org](http://www.minized.org)

[www.microzed.org](http://www.microzed.org)

[www.picozed.org](http://www.picozed.org)

[www.zedboard.org](http://www.zedboard.org)

[www.xilinx.com/zyng](http://www.xilinx.com/zyng)

[www.xilinx.com/sdk](http://www.xilinx.com/sdk)

[www.xilinx.com/vivado](http://www.xilinx.com/vivado)

## Answers

### Experiment 1

- *Was a new BSP required for this application?*

**A new BSP must be generated every time there is a change to the hardware platform. We named it the same, but it was new.**

- *What is the performance increase when transferring 8192 bytes from BRAM to DDR3? Try again from DDR3 to DDR3? BRAM to BRAM?*

**11x, 3x, 20x (Results may vary)**

- *What is the performance increase when transferring 256 bytes from BRAM to DDR3? Try again from DDR3 to DDR3? BRAM to BRAM?*

**10x, 1x, 18x (Results may vary)**