

LAB 5: Direct Memory Access

Introduction

This lab introduces the Xilinx AXI Central Direct Memory Access (CDMA) Intellectual Property core. This soft core allows for high speed data transfers from a memory-mapped source to a memory-mapped destination using the AXI4 protocol [1]. The CDMA core will be instantiated within the programmable logic. The AXI CDMA will act as a master device controlling two of the four available high-performance slave ports available in the processor. These high-performance ports provide a high throughput of data between devices implemented in the programmable logic acting as AXI masters and the processor's DDR memory system [2]. The CDMA will be used to copy an array stored in one memory location to a destination array stored in another location.

Figure 1 shows a block diagram of the system which will be designed in XPS.

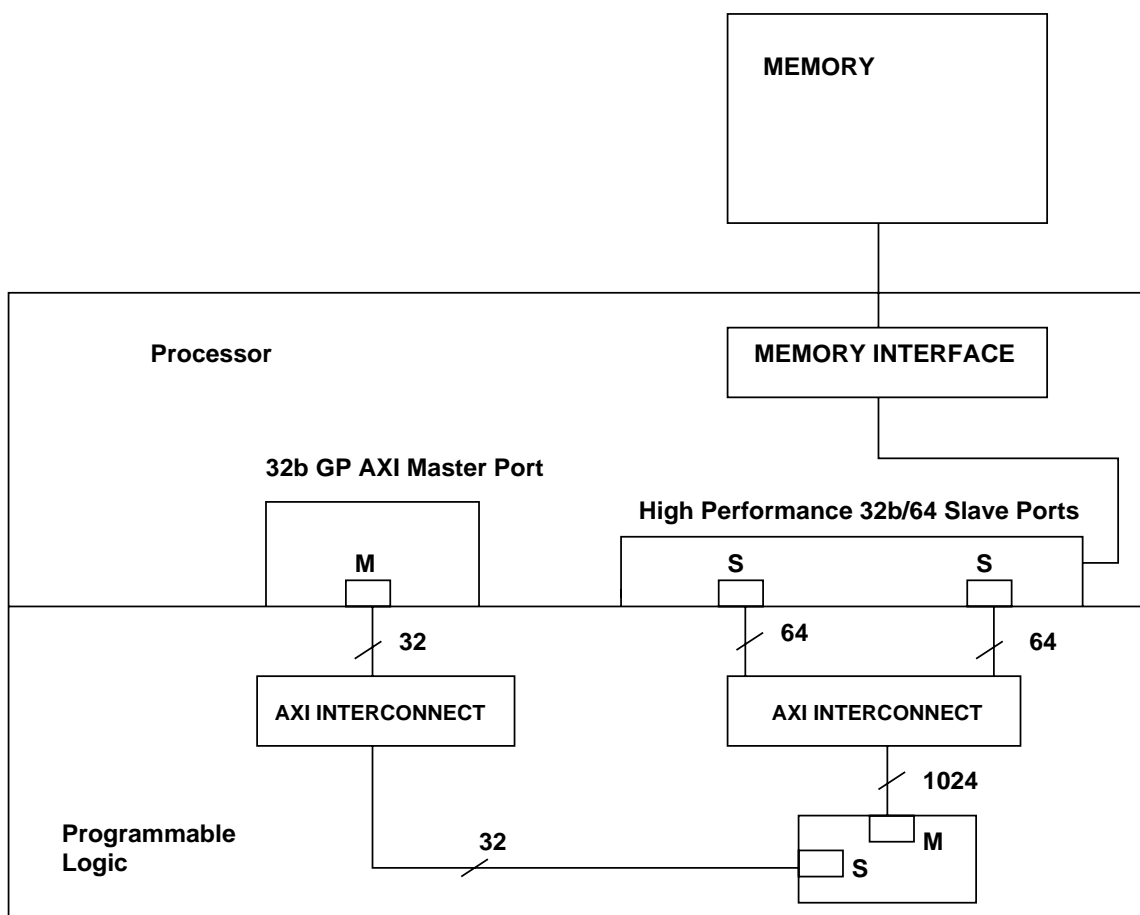


Figure 1: Block diagram of the system.

The block diagram shows the following connections:

- The processor's General Purpose Master Port is connected to the Slave port of the AXI CDMA. The processor will use this Master Port to initially configure the AXI CDMA (by writing to its control register, source address register, destination address register, and Bytes to Transfer registers) and it will also read the status of the device (by polling its status register to determine when the transfer has completed).
- The Master port of the AXI CDMA device is connected to one of the processor's high performance slave port (S_AXI_HP0). This port will be used by the CDMA to read the contents of the source array from the processor's memory.
- The Master port of the AXI CDMA device is also connected to another of the processor's high performance slave port (S_AXI_HP2) and will be used by the CDMA to write to the specified destination array in DDR memory.

This lab is based upon the example system in Chapter 6 of the Xilinx Concept, Tools, and Techniques (CTT) User Guide UG873. The interested reader is directed to this document for more detailed information of the capability of the AXI CDMA core.

Setting up the user Linux environment:

1:. Type the following on a terminal:

```
source /CMC/tools/xilinx_14.7/14.7/ISE_DS/settings64_CMC_central_license.csh
```

This sets up the Xilinx environment

Launching PlanAhead:

2. Launch the planAhead tool from your COEN317_labs/Lab5 directory:

```
planAhead &
```

Create a new project from PlanAhead:

3.1. File -> New Project

3.2. A window pops up, click Next

Give your project a name and choose the desired project path.

Project name: lab5

Project location: ../COEN317_LABS/Lab5

Leave the "Create project subdirectory" checked

Click Next

3.3. RTL Project should be selected. Keep the selection.

Click Next

3.4. No sources need to be specified. However, choose the target language as VHDL in the drop-down box.

Target language: VHDL

Click Next

3.5. We do not need to specify any IPs either

Click Next

3.6. We do not need to specify any UCF file.

Select Next.

3.7. Select the ZYNQ-7 ZC702 Evaluation Board for our project

Click Boards

Scroll down the list and choose ZYNQ-7 ZC702 Evaluation Board

Click Next

3.8. Review settings and click Finish

Create an embedded processor project with the Add source wizard:

4.1. On the left panel under Project Manager, click Add Sources

4.2. Choose Add or Create Embedded Sources and click Next

4.3. Click on Create Sub-design

A window should have popped up asking for a module name. Name it “system”

Module name: system

Click OK

Click Finish

Designing the system in XPS:

XPS will be used to design the system consisting of the AXI CDMA IP core, and 2 AXI interconnects.

5.1. A window pops up and asks:

This project appears to be a blank zynq project.

Do you want to create a Base System using the BSB Wizard?

Click Yes

5.2. The BSB Wizard asks for additional settings

The AXI System should be selected by default with no other parameters.

Click OK

Verify the Zynq Processing System 7 is selected in “Select a System”

Click Next

Remove the GPIO_SW and LEDs_4Bits Peripherals because they are not needed

Click on “Select All”, then “< Remove”

Click Finish

Click OK

5.3 Add the AXI CDMA. Expand the DMA and Timer tab from the IP catalog and select by double clicking the AXI Central DMA. Select Yes when prompted if you want to add the axi_cdma 3.04a IP instance.

5.4. In the CDMA configuration window, select the User tab and specify the following settings:

Data width of data transfer channel : 1024

Maximum Burst Length: 256

Select OK to close the configuration window.

5.5. Unlike the other labs, in this lab we (the user) will manually specify the connections between the AXI CDMA core and the AXI interconnects and processor. Select: User will make necessary connections and settings instead of selecting processing_system7_0 as was done in the previous labs.

Select OK.

5.6 An AXI interconnect will be added to the system and will be used to connect the processor's Master General Purpose port to the slave port of the AXI CDMA. From the IP catalog, expand the Bus and Bridge and select the AXI Interconnect IP by double clicking on it.

In the dialog window, select Yes when asked whether you want to add the axi_interconnect instance to the design.

5.7 In the AXI Interconnect configuration window, name the instance **axi_interconnect_gp1**.

5.8 Repeat the above three steps to add another instance of an AXI Interconnect and name it **axi_interconnect_hp**.

5.9 Select the ZYNQ tab from XPS System Assembly View to open the block diagram view of the system.

5.10. Select the green **32b GP AXI Master Port** (refer to Figure 2) to open the processing_system7_0 Configuration Window.



Figure 2: The 32b GP AXI Master Ports block in the System Assembly view Zynq tab.

5.11. In the processing_system7_0 Configuration Window, select User Tab and expand **General Purpose Master AXI Interfaces** and select the **Enable M_AXI_GP1 interface** as shown in Figure 3.

Select OK to enable the interface and to close the window.

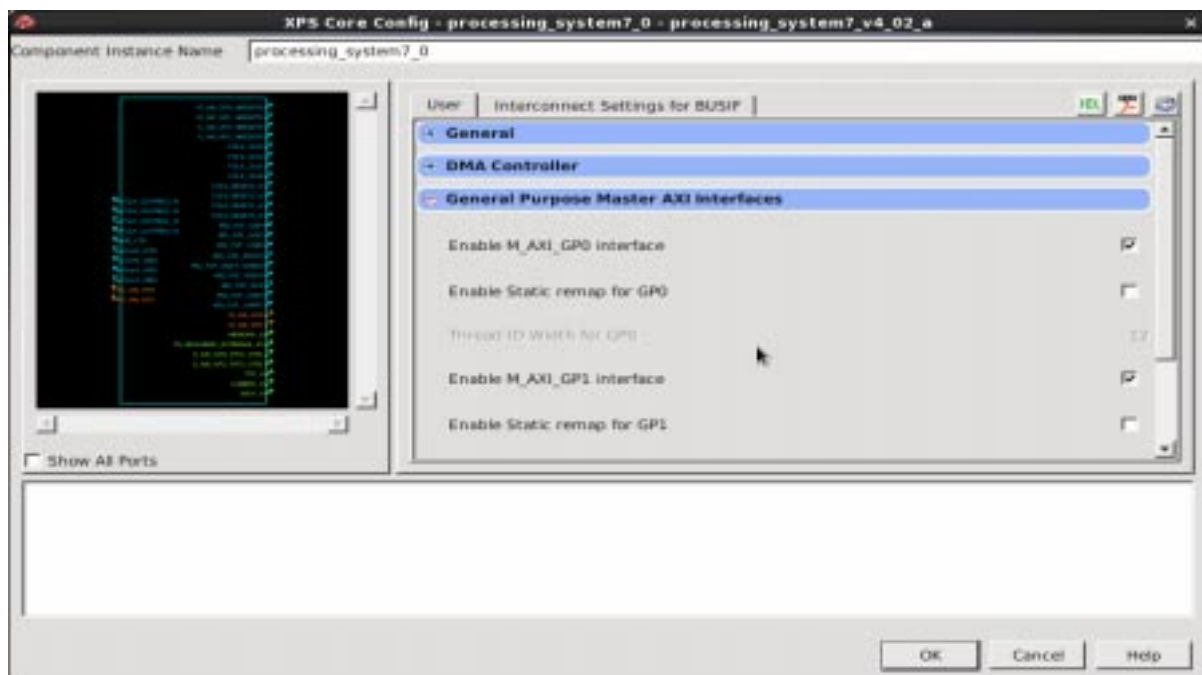


Figure 3: processing_system7_0 Configuration Window.

5.12. From the XPS System Assembly view, double click on the green **High Performance AXI 32b/64b Slave Ports** block (Figure 4). This will again open the Configuration wizard.

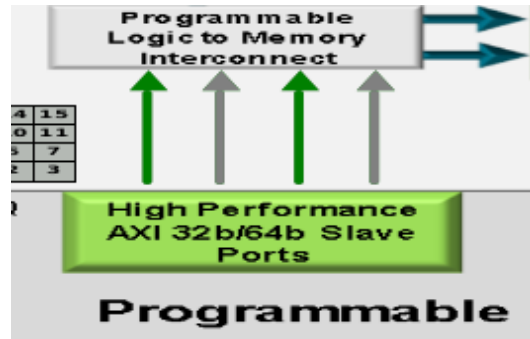


Figure 4: High Performance AXI 32b/64b Slave port block.

5.13. In the Configuration Wizard which opens, select the User tab and expand the **High Performance Slave AXI Interface** as shown in Figure 5.

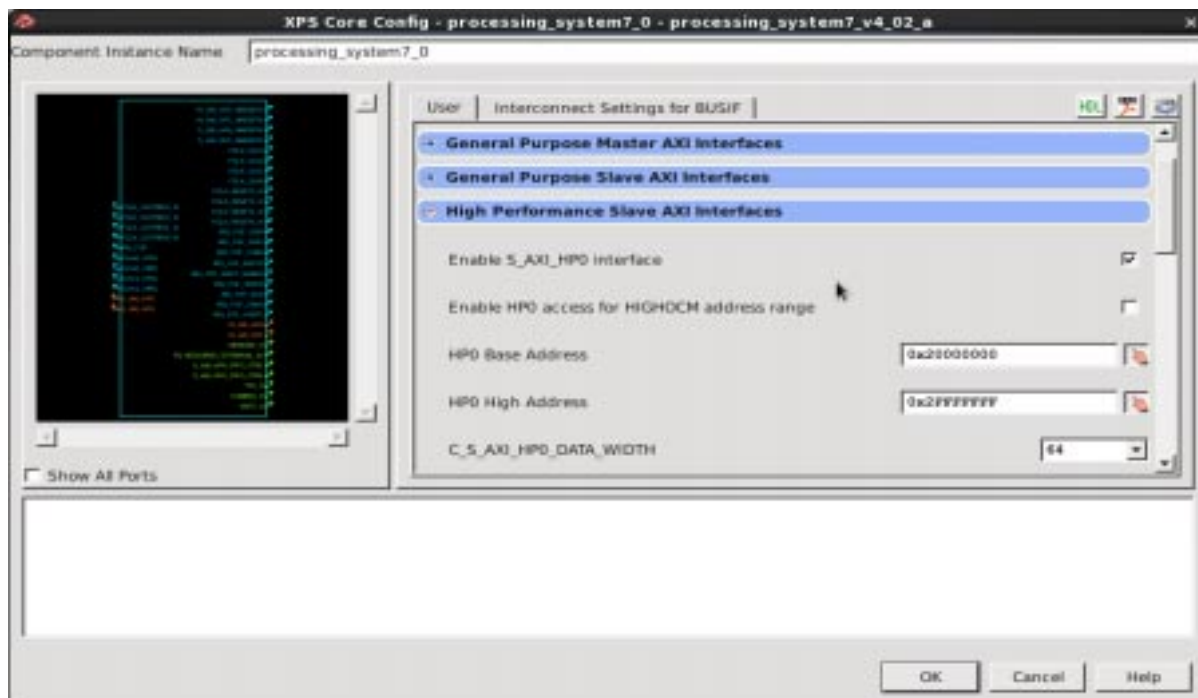


Figure 5: Configuration wizard with Slave AXI interface expanded.

In the Configuration wizard, set the following settings:

- select the check box for the **Enable S_AXI_HP0** interface
- Specify the **HP0 Base Address** to 0x20000000 and the **HP0 High Address** to 0x2fffffff. These settings allow the HP0 slave port to access the processor's DDR ram from these memory locations. The source array will be stored within this address space.
- select the check box for the **Enable S_AXI_HP2** interface.
- Specify the **HP2 Base Address** to 0x30000000 and the **HP2 High Address** to 0x3fffffff. These settings allow the HP2 slave port to access the processor's DDR ram from these memory locations. The destination array will be stored within this address space.

Note that the data widths of the high performance slave ports are set to 64 bits by default. Leave the data widths as their default value.

Select OK in the Configuration window to enable the settings and to close the window.

5.14. We will now manually specify the various connections between the Master and Slave ports of the processor and the AXI CDMA using the two AXI Interconnects which were added when the system was designed (axi_interconnect_gp1 and axi_interconnect_hp). XPS allows the user to specify the connections either graphically or by selecting value from drop down menus. Figure 6 shows the graphical method after the connections have been made. To specify the connections, select the Bus Interfaces tab from the System Assembly View and expand the processing_system_7_0 and the axi_cdma_0 instances to show their various ports. Note that in Figure 6, listed under the processing_system7_0 instance are four ports:

- M_AXI_GP0
- M_AXI_GP1
- S_AXI_HP0
- S_AXI_HP2

Your design will initially not list the M_AXI_GP0 port as it will be added to the system when an instance of an AXI timer (together with its associated axi interconnect) will be added to measure the time required to complete a memory transfer.

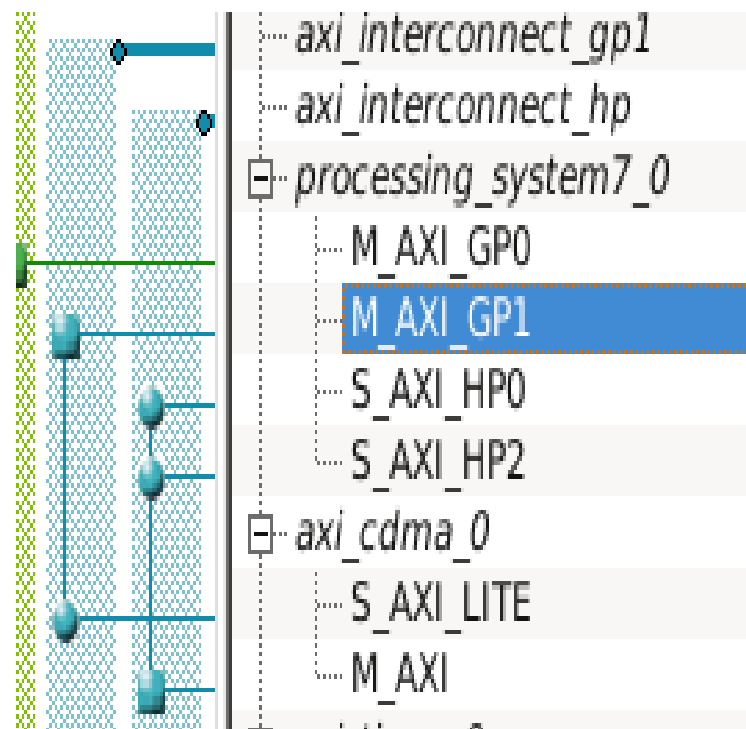


Fig 6. Specifying the Master-Slave connections between the two AXI Interconnects.

In Figure 6, a square box indicates that the listed port is acting as the Master, and a round circle indicates that the listed port is a Slave. The two vertical blue lines represent the connections between the AXI Interconnects in the system and the various Master/Slave ports. To specify that the M_AXI_GP1 port listed under processing_system7_0 is to be the Master connected to the axi_interconnect_gp1, we move the cursor to the square blue box and select it. Next, select the round blue circle listed in the “axi_interconnect_gp1” blue bus and the S_AXI_LITE port listed under the axi_cdma_0 instance. This will establish a connection between the M_AXI_GP1 port of the processor and the S_AXI_LITE port of the AXI CDMA core via the axi_interconnect_gp1.

In a similar fashion, we can specify that the M_AXI port of the axi_cdma_0 instance is the Master of S_AXI_HP0 and S_AXI_HP2 ports of the processing_system7_0 using the axi_interconnect_hp instance by selecting the blue square which appears next to M_AXI of the axi_cdma_0 instance in the rightmost blue vertical “bus” which represent the axi_interconnect_hp and the two blue circles next to the S_AXI_HP0 and S_AXI_HP1 ports listed under processing_system7_0.

Alternatively, one may specify the connections using dialog windows in the following manner:

1. Select the Bus Interfaces tab in the System Assembly view, expand the processing_system7_0 to list its port. Select M_AXI_GP1 and select No Connection listed in the Bus Name column (the right most column). Refer to Figure 7.

Zynq		
Bus Interfaces	Ports	Addresses
Name	IP Version	Bus Name
axi_interconnect_1	1.06.a	
axi_interconnect_gp1	1.06.a	
axi_interconnect_hp	1.06.a	
processing_system7_0	4.02.a	
M_AXI_GP0		axi_interconnect_1
M_AXI_GP1		No Connection

Figure 7: Selecting the No Connection entry listed in the Bus Name column.

2. A drop down list will appear listing the available AXI Interconnects in the system. Select the axi_interconnect_gp1.
3. Expand the axi_cdma_0 instance and select its S_AXI_LITE port and select the No Connection entry listed in the Bus Name column to open a Connection dialog box.
4. In the Connection dialog box, select the axi_interconnect_gp1 from the listed interconnects in the left hand portion of the window. You will note that in the right hand portion of the window, that the processing_system7_0.M_AXI_GP1 appears in the Select Master(s) list. Select the check box next to it and select OK. Refer to Figure 8.

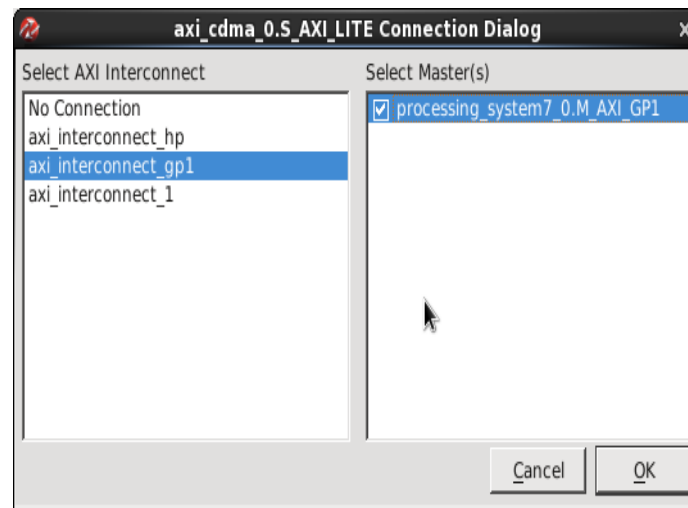


Figure 8: Connection Dialog window.

XPS will connect the S_AXI_LITE slave port of the axi_cdma_0 to the M_AXI_GP1 port of the processor. You can visually view these connections in the left most portion of the Bus Interfaces view.

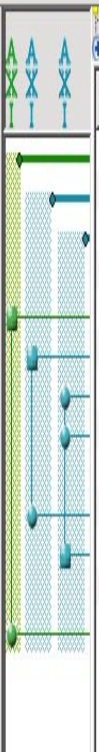
(5) In a similar manner, select the M_AXI port of the axi_cdma_0 and select No Connection in its Bus Name column and in the drop down list which will appear select axi_interconnect_hp.

(6) Select the S_AXI_HP0 listed under the processing_system7_0 instance and select the No Connection entry in the Bus Name column to open the Connection Dialog Box.

(7) In the Connection Dialog box, select the axi_interconnect_hp from the list of interconnects on the left hand side. The axi_cdma_0.M_AXI port will be listed in the Select Master(s) list on the right hand side. Select the check box next to it. Select OK to make the connection.

(8) Repeat the above two steps to connect the axi_cdma_0.M_AXI master port with the processing_system7_0.S_AXI_HP2 port.

5.15. Select the Bus Interfaces Tab to review the connections. They should appear as listed in Figure 9.



Name	IP Version	Bus Name
axi_interconnect_1	1.06.a	
axi_interconnect_gp1	1.06.a	
axi_interconnect_hp	1.06.a	
processing_system7_0	4.02.a	
M_AXI_GP0		axi_interconnect_1
M_AXI_GP1		axi_interconnect_gp1
S_AXI_HP0		axi_interconnect hp:axi cdma 0.M AXI
S_AXI_HP2		axi_interconnect hp:axi cdma 0.M AXI
axi_cdma_0	3.04.a	
S_AXI_LITE		axi_interconnect gp1:processing_system7_0.M AXI GP1
M_AXI		axi_interconnect_hp
axi_timer_0	1.03.a	
S_AXI		axi_interconnect_1

Figure 9: Final Bus Interface connections.

5.16. The next step is to connect the clock and the reset inputs of the axi_interconnect_gp1, axi_interconnect_hp, and axi_cdma_0 instances to the FCLK_CLK0 and FCLK_RESET0 of processing_system7_0. Select the Ports tab and expand these three instances and make the clock and reset connections as indicated in Figure 10. To make the connection, select a signal such as INTERCONNECT_ACLK listed under the axi_interconnect_gp1 instance (it will be highlighted in blue when selected), select the Connected port column and then right click the blue portion under the Connected Port column and select processing_system7_0:FCLK_CLK0 from the drop down list. Repeat this procedure for all the clocks and resets listed in Figure 10.

Zynq Bus Interfaces Ports Addresses		
Name	Connected Port	Direction
External Ports		
axi_interconnect_1		
INTERCONNECT_ACLK	processing_system7_0::FCLK_CLK0	I
INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N	I
axi_interconnect_gp1		
INTERCONNECT_ACLK	processing_system7_0::FCLK_CLK0	I
INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N	I
axi_interconnect_hp		
INTERCONNECT_ACLK	processing_system7_0::FCLK_CLK0	I
INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N	I
processing_system7_0		
axi_cdma_0		
cdma_introut		O
cdma_tvect_out		O
(BUS_IF) S_AXI_LITE	Connected to BUS axi_interconnect_gp1	
s_axi_lite_aclk	processing_system7_0::FCLK_CLK0	I
(BUS_IF) M_AXI	Connected to BUS axi_interconnect_hp	
m_axi_aclk	processing_system7_0::FCLK_CLK0	I

Figure 10: Clock and Reset connections for the axi_interconnect_gp1, axi_interconnect_hp, and axi_cdma_0 instances in the system.

5.17. Collapse the three IP instances and expand processing_system7_0 in the Ports tab. Scroll through the list to find the (BUS_IF) M_AXI_GP1, (BUS_IF) S_AXI_HP0, and (BUS_IF) S_AXI_HP2 ports and expand them to list their associated clock signals and specify that they are connected to the processing_system7_0:FCLK_CLK0 signal as shown in Figure 11.

(BUS_IF) M_AXI_GP1	Connected to BUS axi_interconnect_gp1
M_AXI_GP1_ACLK	processing_system7_0::FCLK_CLK0
(BUS_IF) S_AXI_HP0	Connected to BUS axi_interconnect_hp
S_AXI_HP0_ACLK	processing_system7_0::FCLK_CLK0
(BUS_IF) S_AXI_HP2	Connected to BUS axi_interconnect_hp
S_AXI_HP2_ACLK	processing_system7_0::FCLK_CLK0

Figure 11: Specifying the clock for the three (BUS_IF) ports.

5.18. The last step is to generate addresses for any unmapped instances, in this case there is only one: the axi_cdma_0. Select the Addresses tab and then select the Generate Addresses icon (it is at the top right most portion of the window it is the icon with the red arrow pointing to a yellow folder, refer to Figure 12). to generate the addresses.

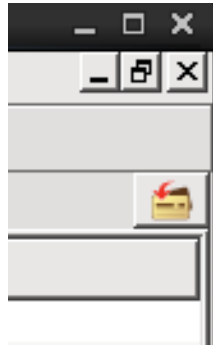


Figure 12: Generate Addresses button.

Exporting the Hardware to SDK:

Follow the steps to create the top-level HDL stub, synthesize, implement, generate the bitstream, and program the FPGA with the .bit file using Impact and export the hardware to SDK. Note that for this design, the synthesize and implement steps may take on the order of 10-15 minutes.

Using SDK to create a new application project:

Follow the steps as in the previous labs, to create a new C++ application project. Refer to the LogiCORE IP AXI Central Direct Memory Access Product Guide found in :

/home/t/ted/PUBLIC/COEN317/Lab5/axi_cdma_product_guide.pdf

to become familiar with the operation of the CDMA core. The salient sections to review are:

- Register Space in Chapter 2 which describes the control and status registers used to configure the CDMA and to read its status. Table 2 on page 21 gives the register address map as offsets from the device base address (recall that this base address was generated by XPS during the Generate Address step) and is also given as value in the xparameters file through a #define XPAR_AXI_CDMA_0_BASEADDR directive. The five important registers for this lab are: CDMACR (control), CDMA SR (status), SA (Source Address), DA (Destination Address), and BTT (Byte to Transfer). The registers may be written to and read from using a pointer dereference operation (as was done in the GPIO lab and the timer lab)
- refer to the Register Detail section (pp. 22 - 33) for the meaning of the individual bits within the register which are used to program the device (via the control register) and to obtain information concerning the status of the device (accomplished by reading the status register).

- the Sequence of Operation given on page 44 summarizes the required steps to use the CDMA in Simple DMA mode. We will not be using the other available mode (Scatter Gather Mode) in this lab.

- recall that the source address was specified in XPS as 0x20000000 and the destination address was specified as 0x30000000. These were the base addresses associated with the slave HP0 and HP2 ports. These addresses are also defined in the xparameters.h file with the two #define directives defining XPAR_PS7_DDR_0_S_AXI_HP0_BASEADDR and XPAR_PS7_DDR_0_S_AXI_HP2_BASEADDR.

Include the following libraries in your source code:

```
#include "xil_exception.h"
#include "xil_cache.h"
#include "xparameters.h"
#include <iostream>
using namespace std;
```

The following is pseudocode for the main function. You are required to convert it in a full C++ program based upon the information obtained by reading the product data sheet.

```
int main()
{
    -declare and initialize three pointers to u32 for the
    cdma_base_address, the source_address and the destination_address

    -initialize the contents of the source array with some known ini-
    tial value. The array may be of any size up to a certain maximum
    size ( a careful reading of the AXI CDMA product guide will reveal
    what this size is... HINT: study the description of the BTT reg-
    ister on page 33). It is suggested you do something along the
    lines of:

    source_address[0] = 0;
    source_address[1] = 1;
    source_address[2] = 2;
    source_address[3] = 3;
    etc.

    -initialize the contents of the entire destination array all with
    some value which is not found in the source_address (for example,
    some negative value). This will make it easier for you
```

determine whether the transfer correctly copied the source array to the destination array.

- reset the CDMA by writing the appropriate bit pattern to the control register
- poll the CDMA status register to determine if the device is idle before configuring it.
- configure the CDMA to function in simple mode with no interrupts enabled (as in this lab we will not be making use of the interrupt generation capability of the device, but rather will rely upon polling the idle bit contained in the status register to determine whether the transfer has completed or not.
- load the value of the source array address in the SA register.
- load the value of the destination array address in the DA register.
- flush the cache by calling the function `Xil_DCacheFlush()`.
- write the value of the number of bytes to transfer in the BTT register to initiate the DMA transfer.

```
while (the idle bit in the status register == 0 )
```

```
// use a mask initialized to a certain hex value
// and do a bitwise logical AND of the mask with
// the status register to isolate the idle bit
{
    //wait for transfer to complete (i.e. wait until the idle
    // bit becomes 1 )
    // p.29 of Product spec states
    //    0 = not Idle - simple DMA operations are in progress
    //    1 = Idle - Simple DMA operations completed
    //        or not started.
}
```

- compare the contents of the destination array after the transfer with the source array. If the two arrays are different, display a message stating this.

- display a "End of Application" message

```
return 0; // end of main
}
```

Program Debugging Hints:

If your program does not initially work, don't panic. Print out the value of the CDMA status register after each step to determine the cause of the problem. It will be easier if you display the values in hexadecimal format instead of as unsigned integer. A simple way in C++ to display values in hex is to use the `std::hex` manipulator as input to the `<<` operator in a `cout` statement as in:

```
// Author: Ted Obuchowicz
// file: cout_hex.C

#include <iostream>
using namespace std;

int main()
{
    cout << std::hex << 255 << endl; // will display 255 in hex as ff
    return 0;
}
```

The above program gives the following output when run from the Linux command line:

```
ted@deadflowers Programs 11:05pm >cout_hex
ff
```

Questions:

1. Add an instance of an AXI Timer to your design (using XPS). Use the timer to measure how long (in number of clock cycles) it takes to transfer the array without using the CDMA and how long it takes to perform transfer using the CDMA. Obtain values for different sizes of the array: 1024 integers, 4096 integers, 8192, 16384, 32767, 1048576, up to a maximum number of integers allowed in one transfer. Compute the speedup given by the use of the CDMA to perform the memory transfer.
2. What will happen if we program the BTT register with a number which is greater than the maximum allowed ?

References:

- [1]. LogiCORE IP AXI Central direct Memory Access v3.03a Product Guide, Xilinx Inc., PG034 October 16, 2012, p.5
- [2]. Zynq-7000 All Programmable SoC: Concepts, Tools, and Techniques (CTT), Xilinx Inc., UG873 (v14.4) December 18 (Keith's birthday), 2012, p.52.

T. Obuchowicz
November 13, 2014

Revision History:

- Sept. 8, 2016: Revised for Lab 5 Fall 2016. Modified to source `/CMC/tools/xilinx_14.7/14.7/ISE_DS/settings64_CMC_central_license.csh`.