Pragmas and Data Motion Networks

Introduction

This lab guides you through the process of handling data transfers between the software and hardware accelerators using various pragmas and the SDSoC API.

Objectives

After completing this lab, you will be able to:

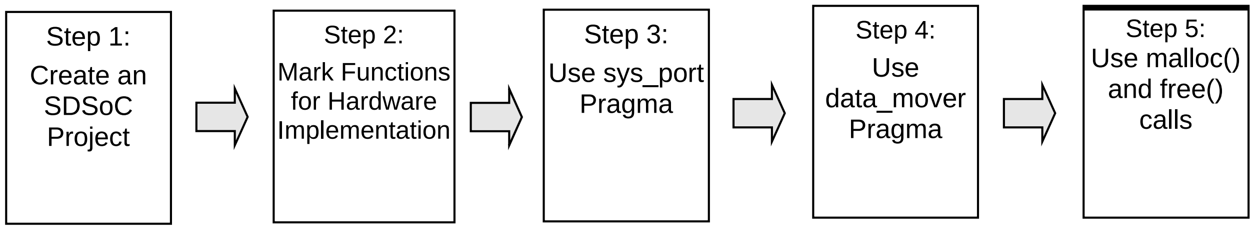
- Use pragmas to select ACP or AFI ports for data transfer
- Use pragmas to select different data movers for your hardware function arguments
- Understand the use of sds_alloc() and sds_free() calls
- Understand the use of malloc() and free() calls
- Analyze built hardware

Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

This lab comprises five primary steps: You will create an SDSoC project, mark two functions for hardware implementation, use sys_port and data_mover pragmas and analyze the built hardware, and, use malloc() and free() calls and see their impact on the hardware.

General Flow for this Lab



Create an SDSoC Project Step 1

1-1. Launch SDSoC and create a project, called *lab2*, using the *matrix-multiply and add* template, targeting the Zed or Zybo board.

1-1-1. Open SDSoC by selecting **Start > All Programs > Xilinx Design Tools > SDSoC 2015.4 > SDSoC 2015.4**

The Workspace Launcher window will appear.

1-1-2. Click on the Browse button and browse to **c:\xup\SDSoC\labs,** if necessary and click **OK**.

1-1-3. Click **OK**.

Click **X** on the *Welcome* tab, if displayed, to close it.

1-1-4. Select **File > New > SDSoc Project** to open the New Project GUI.

1-1-5. Enter **lab2** as the project name, select either *zybo* or *zed* (depending on the board you are using) via drop-down button, select *Linux* as the target OS, and click **Next**.

The Templates page appears, containing source code examples for the selected platform.

1-1-6. Select **Matrix Multiply-Add (area reduced)** in case of *zybo* or **Matrix Multiplication and Addition** in case of *zed* as the source.

1-1-7. Click **Finish**.

The *Project Explorer* tab will display the **lab2** project directory. The **lab2** folder also shows the **project.sdsoc** file. Double-clicking on it will display what you see in the right-side pane.

Mark Functions for Hardware Implementation Step 2

2-1. Mark *madd* and *mmult* functions for the hardware accelerations with default clock speed.

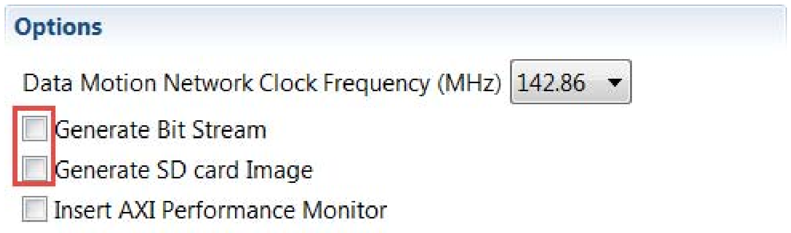
2-1-1. Expand **mmult.cpp** and **madd.cpp** under *lab2 > src* in the Project Explorer tab, right click on *mmult* and *madd* functions.

Alternatively, click on the “+” sign in the Hardware Functions area to open up the list of functions which are in the source file. Using Ctrl key and mouse clicks, select *mmult* and *madd* entries and click **OK**.

The two function names will be added into the Hardware Functions window. Notice that they will be using the default clocks.

2-1-2. Select **Build Configurations > Set Active > SDRelease**

2-1-3. In the SDSoC Project Overview pane on right, deselect the Bitstream and SD card image generation options since we want to explore the generated system hardware.


Options

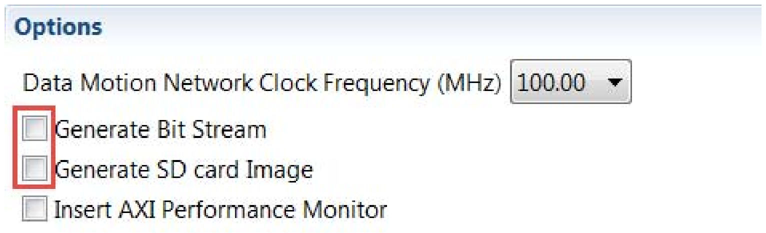
Data Motion Network Clock Frequency (MHz) 142.86

Generate Bit Stream [ ]

Generate SD card Image [ ]

Insert AXI Performance Monitor [ ]

(a) Zed


Options

Data Motion Network Clock Frequency (MHz) 100.00

Generate Bit Stream [ ]

Generate SD card Image [ ]

Insert AXI Performance Monitor [ ]

(b) Zybo

Figure 1. Deselecting Bitstream and SD card image generation options

2-2. Build the project. When done, analyze the data motion network through the report and built hardware using Vivado IPI.

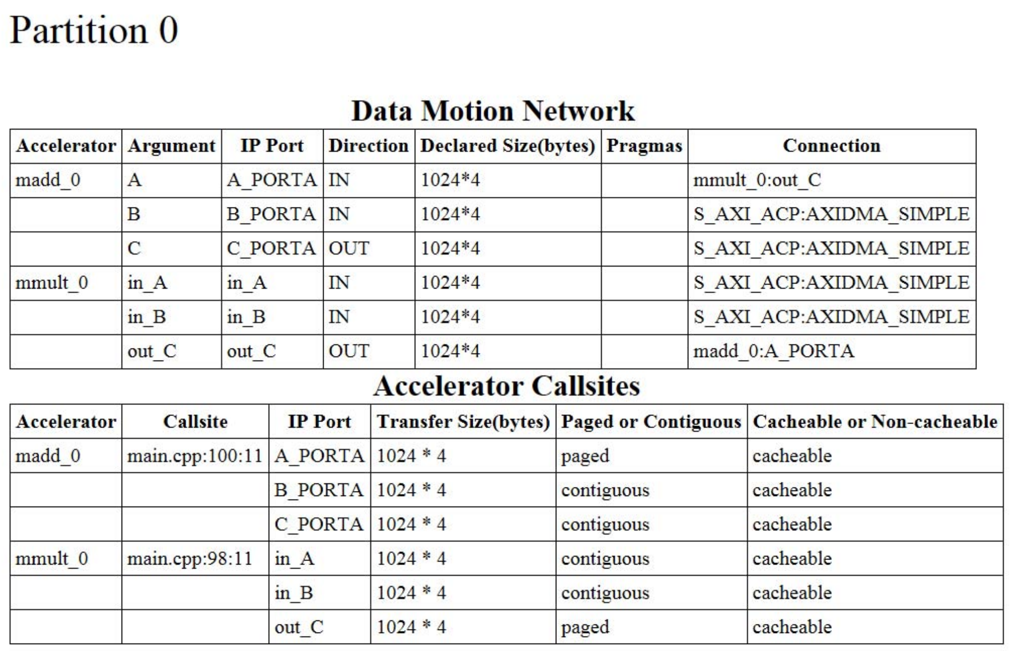
2-2-1. Right-click on **lab2** and select **Build Project**

This may take about 5 minutes.

2-2-2. Expand the **lab2** directory in Project Explorer and observe that *SDRelease* folder is created along with virtual folders of *Binaries* and *Archives*. Expanding the *SDRelease* folder shows _**sds** and **src** folders along with **lab2.elf** (executable)**, lab2.elf.bit** (hardware bit file) and several make files.

2-2-3. In the SDSoC Project Overview window, under the *Reports* pane, click on **Data motion** link to view the Data Motion Network report.

The report shows the connections made by the SDSoC environment and the types of data transfers for each function implemented in hardware. You can also open this report file by double-clicking **data_motion.html** entry in **SDRelease > _sds > reports** of Project Explorer. This will be used for reference later.



Partition 0

Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
madd_0	A	A_PORTA	IN	1024*4		mmult_0:out_C
	B	B_PORTA	IN	1024*4		S_AXI_ACP:AXIDMA_SIMPLE
	C	C_PORTA	OUT	1024*4		S_AXI_ACP:AXIDMA_SIMPLE
mmult_0	in_A	in_A	IN	1024*4		S_AXI_ACP:AXIDMA_SIMPLE
	in_B	in_B	IN	1024*4		S_AXI_ACP:AXIDMA_SIMPLE
	out_C	out_C	OUT	1024*4		madd_0:A_PORTA

Accelerator Callsites

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Cacheable or Non-cacheable
madd_0	main.cpp:100:11	A_PORTA	1024 * 4	paged	cacheable
		B_PORTA	1024 * 4	contiguous	cacheable
		C_PORTA	1024 * 4	contiguous	cacheable
mmult_0	main.cpp:98:11	in_A	1024 * 4	contiguous	cacheable
		in_B	1024 * 4	contiguous	cacheable
		out_C	1024 * 4	paged	cacheable

Figure 2. Data motion network and accelerator callsites

There are two accelerated functions- madd and mmult. They are given instance names as madd_0 and mmult_0. Each function has three arguments and hence three ports. Notice that the out_C port of mmult_0 is directly connected to A_PORTA port of madd_0 port, whereas the other two ports of each hardware are connected in the system via AXIDMA_SIMPLE channels on ACP.

The transfer size id 4096 bytes or 1024 words on each ports of the two accelerators.

2-2-4. As before, open Vivado by selecting **Start > All Programs > Xilinx Design Tools > SDSoC 2015.4 > Vivado Design Suite > Vivado 2015.4**

2-2-5. Open the design by browsing to *c:\xup\SDSoC\labs\lab2\SDRelease_sds\p0\ipi* and selecting either the **zybo.xpr** or **zed.xpr**.

2-2-6. Click on **Open Block Design** in the Flow Navigator pane. The block design will open. Note various system blocks which connect to the Cortex-A9 processor (identified by ZYNQ in the diagram).

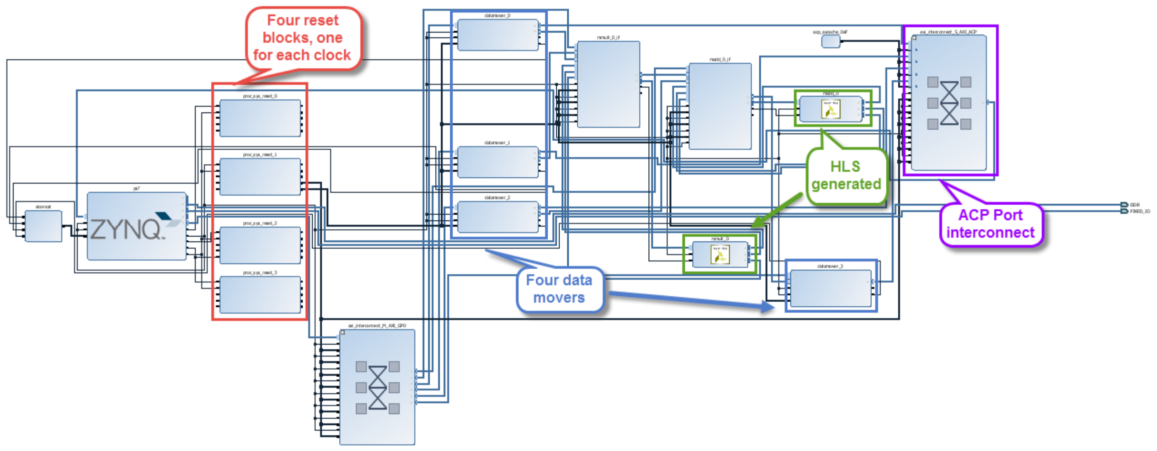


Figure 3. The generated block design

2-2-7. Click on the **show interface connections only** ( ) button followed by click on the **regenerate layout** ( ) button.

2-2-8. Follow the connections, and notice a data path from *in_A* and *in_B* of *mmult_0* and observe that there is a master connection to the **S_AXI_ACP** port of *PS7* through the *datamover*.

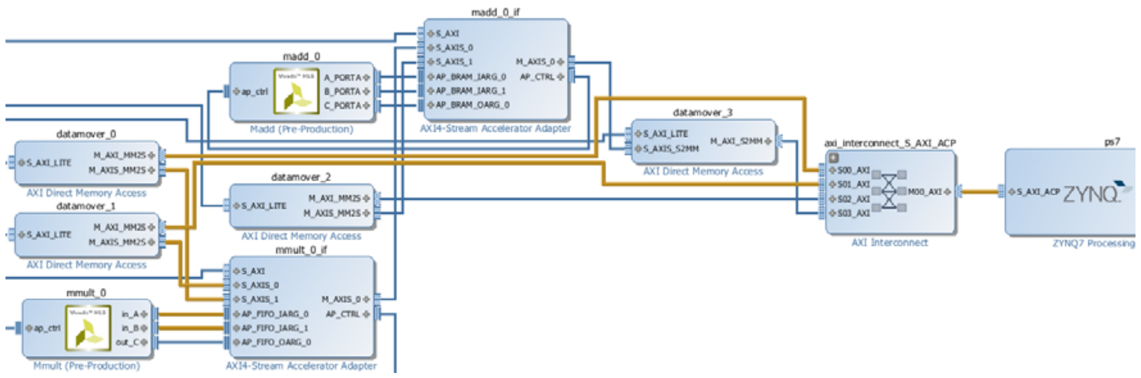


Figure 4. Tracing input datapath of mmult_0

Notice that input data *in_A* (Master) is connected to *AP_FIFO_IARG_0 (Slave)*, which in connected to *M_AXIS_MM2S* of *datamover_1 (Master)* through *S_AXIS_0 (Slave)* (corresponds to *AP_FIFO_IARG_0*). The *M_AXI_MM2S* (data fetch) is connected to *S01_AXI* of the *axi_interconnect_S_AXI_ACP* instance. Similarly, the datapath of *in_B* is *in_B > AP_FIFO_IARG_1 > S_AXIS_1 > M_AXIS_MM2S* of *datamover_0 > M_AXI_MM2S* of *datamover_0 > S00_AXI* of *axi_interconnect_S_AXI_ACP*. Both operands (input data to *mmult*) are provided by *axi_interconnect_S_AXI_ACP* (Master) which is connected to the *S_AXI_ACP (ACP Slave)* of PS7.

2-2-9. Close Vivado by selecting **File > Exit.** Do not save the block design.

Using sys_port Pragma Step 3

3-1. Add sys_port pragma in mmult.h file. Build the project and analyze the data motion network.

3-1-1. Expand **lab2 > src** and double-click on *main.cpp* to see its content.

If line numbers are not visible then you can right-click in the left border of the file and select Show Line Numbers.

3-1-2. Double-click the *mmult.h* file in the Project Explorer view, to open the file in the source editor.

3-1-3. Immediately preceding the declaration for the *mmult* function (line 9), insert the following to specify the system port for each of the input arrays

```
#pragma SDS data sys_port(in_A:ACP, in_B:AFI)
```

ACP is the default connection type, but it will be specified explicitly for in_A. *in_B* will have an AFI type which will connect it to one of the PS7 HP ports.

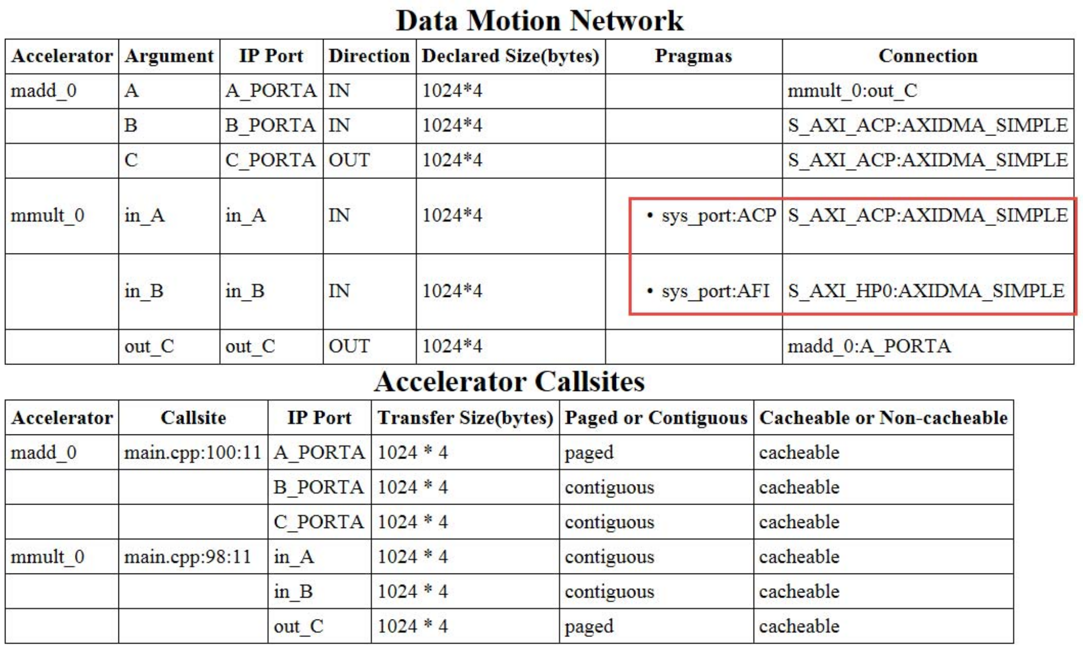
3-1-4. Save the file by selecting **File > Save**

3-1-5. Right-click the top-level folder for the project and click on **Clean Project** in the menu.

3-1-6. Right-click the top-level folder for the project and click on **Build Project** in the menu.

3-1-7. When build process is done, select the **lab2** tab so you can access Data Motion link.

3-1-8. Click on the **Data Motion report** link and analyze the result.



Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
madd_0	A	A_PORTA	IN	1024*4		mmult_0:out_C
	B	B_PORTA	IN	1024*4		S_AXI_ACP:AXIDMA_SIMPLE
	C	C_PORTA	OUT	1024*4		S_AXI_ACP:AXIDMA_SIMPLE
mmult_0	in_A	in_A	IN	1024*4	• sys_port:ACP	S_AXI_ACP:AXIDMA_SIMPLE
	in_B	in_B	IN	1024*4	• sys_port:AFI	S_AXI_HP0:AXIDMA_SIMPLE
	out_C	out_C	OUT	1024*4		madd_0:A_PORTA

Accelerator Callsites

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Cacheable or Non-cacheable
madd_0	main.cpp:100:11	A_PORTA	1024 * 4	paged	cacheable
		B_PORTA	1024 * 4	contiguous	cacheable
		C_PORTA	1024 * 4	contiguous	cacheable
mmult_0	main.cpp:98:11	in_A	1024 * 4	contiguous	cacheable
		in_B	1024 * 4	contiguous	cacheable
		out_C	1024 * 4	paged	cacheable

Figure 5. Data Motion network after applying sys_port pragma

Compared to Figure 2, observe that the *Pragmas* column has two *sys_port* entries for the *mmult_0* instance. The same column shows that *in_A* port is connected to *ACP* whereas *in_B* is connected to *AFI* (HPx). The connections are made to the *S_AXI_ACP* and *S_AXI_HP0* ports of the PS7. The AXIDMA_SIMPLE transfer is also selected.

3-2. Open Vivado IPI design.

3-2-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > SDSoC 2015.4 > Vivado Design Suite > Vivado 2015.4**

3-2-2. Open the design again by browsing to *c:\xup\SDSoC\labs\lab2\SDRelease_sds\p0\ipi* and selecting either the **zybo.xpr** or **zed.xpr**.

3-2-3. Click on **Open Block Design** in the *Flow Navigator* pane. The block design will open. Note various system blocks which connect to the Cortex-A9 processor (identified by ZYNQ in the diagram).

3-2-4. Click on the **show interface connections only** ( ) button followed by click on the **regenerate layout** ( ) button.

3-2-5. Follow through the data path of *in_B* of *mmult_0* and observe that there is a connection to the **S_AXI_HP0** port of *PS7*.

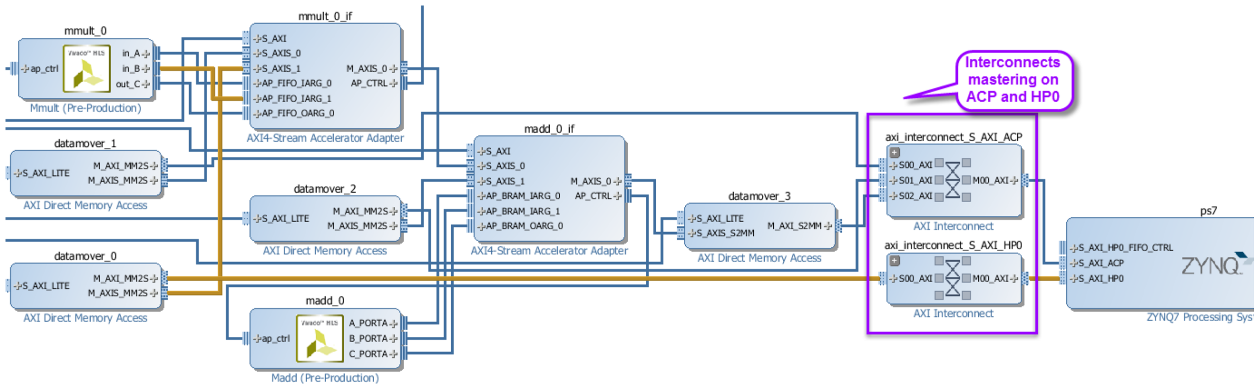


Figure 6. Tracing in_B input datapath of mmult_0

Notice that input_data *in_B* (Master) connects to *AP_FIFO_IARG_1*, which is connected to *M_AXIS_MM2S* of *datamover_0* through *S_AXIS_1*. The *M_AXI_MM2S* is connected to *S00_AXI* of the *axi_interconnect_S_AXI_HP0* instance. The *axi_interconnect_S_AXI_HP0* connects to *S_AXI_HP0* of PS7 to provide the requested data. Four datamover instances, all of AXI DMA type, are used.

3-2-6. Close Vivado by selecting **File > Exit.** Do not save the block design.

Using data_mover Pragma Step 4

4-1. Comment out the sys_port pragma and add data_mover pragma in mmult.h file. Build the project and analyze the data motion network.

4-1-1. Double-click the **mmult.h** under *lab2 > src*.

4-1-2. Comment out the pragma that you had inserted in the previous section.

4-1-3. Add the following pragma statement above the mmult function declaration.

```
#pragma SDS data data_mover(in_A:AXIDMA_SG, in_B:AXIDMA_SIMPLE,
out_C:AXIFIFO)
```

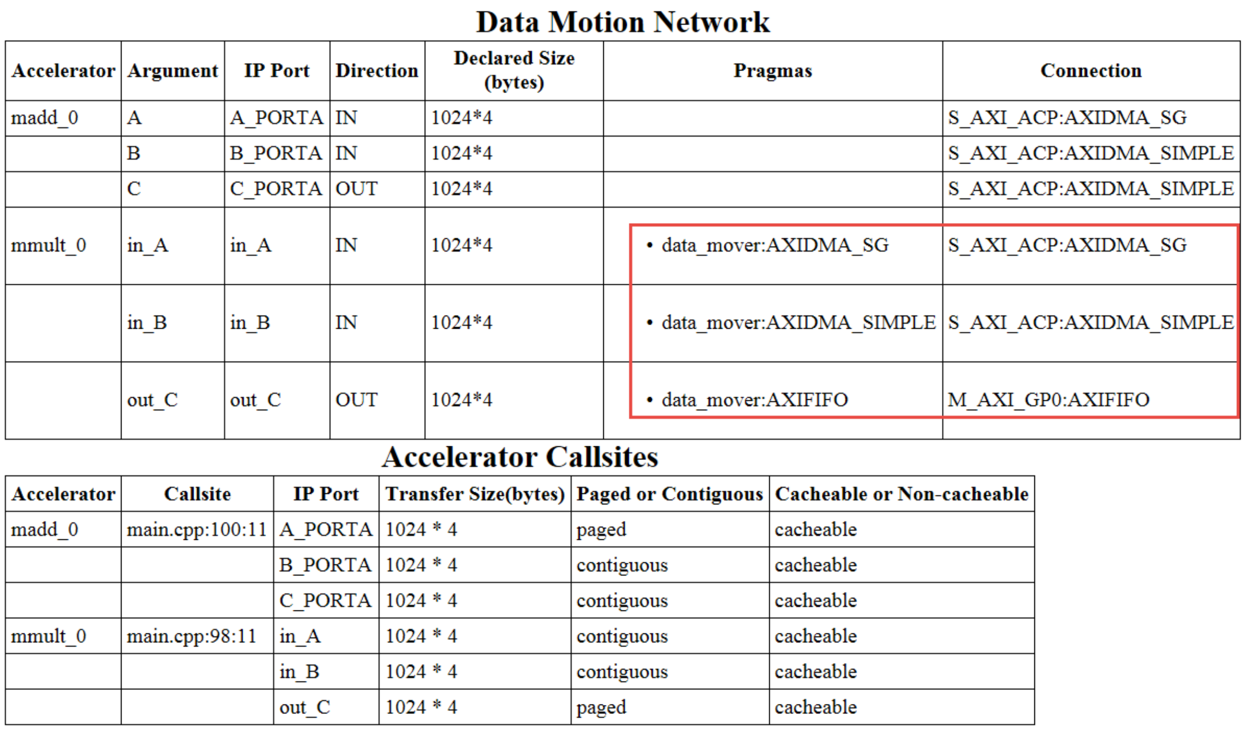
4-1-4. Save the file by selecting **File > Save**

4-1-5. Right-click the top-level folder for the project and click on **Clean Project** in the menu.

4-1-6. Right-click the top-level folder for the project and click on **Build Project** in the menu.

4-1-7. When build process is done, select the **lab2** tab so you can access Data Motion link.

4-1-8. Click on the **Data Motion report** link and analyze the result.



Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size (bytes)	Pragmas	Connection
madd_0	A	A_PORTA	IN	1024*4		S_AXI_ACP:AXIDMA_SG
	B	B_PORTA	IN	1024*4		S_AXI_ACP:AXIDMA_SIMPLE
	C	C_PORTA	OUT	1024*4		S_AXI_ACP:AXIDMA_SIMPLE
mmult_0	in_A	in_A	IN	1024*4	• data_mover:AXIDMA_SG	S_AXI_ACP:AXIDMA_SG
	in_B	in_B	IN	1024*4	• data_mover:AXIDMA_SIMPLE	S_AXI_ACP:AXIDMA_SIMPLE
	out_C	out_C	OUT	1024*4	• data_mover:AXIFIFO	M_AXI_GP0:AXIFIFO

Accelerator Callsites

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Cacheable or Non-cacheable
madd_0	main.cpp:100:11	A_PORTA	1024 * 4	paged	cacheable
		B_PORTA	1024 * 4	contiguous	cacheable
		C_PORTA	1024 * 4	contiguous	cacheable
mmult_0	main.cpp:98:11	in_A	1024 * 4	contiguous	cacheable
		in_B	1024 * 4	contiguous	cacheable
		out_C	1024 * 4	paged	cacheable

Figure 7. Data Motion network after applying data_mover pragma

Compared to Figure 2, observe that Pragmas columns has three *data mover* entries for the mmult_0 instance. The same column shows that in_A port is using AXIDMA_SG data mover, in_B is using AXIDMA_SIMPLE data mover, and out_C is using AXIFIFO data mover. The connection column indicates that in_A and in_B are connected to ACP whereas out_C is connected to GP0 of the PS7.

4-2. Open Vivado IPI design.

4-2-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > SDSoC 2015.4 > Vivado Design Suite > Vivado 2015.4**

4-2-2. Open the design by browsing to *c:\xup\SDSoC\labs\lab2\SDRelease_sds\p0\ipi* and selecting either the **zybo.xpr** or **zed.xpr**.

4-2-3. Click on **Open Block Design** in the *Flow Navigator* pane.

4-2-4. Click on the **show interface connections only** button followed by click on the **regenerate layout** button.

4-2-5. Follow through the data path of *in_A* of *mmult_0* and observe that there is a connection to the **S_AXI_ACP** port of *PS7* using AXI DMA SG data mover (*datamover_0*).

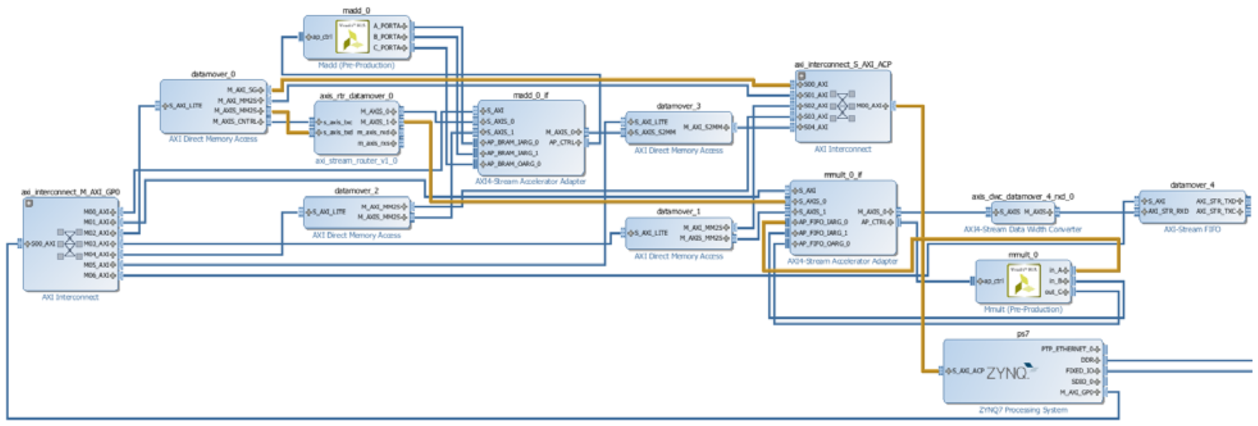


Figure 8. Tracing in_B input datapath of mmult_0 through SG datamover

Notice that in_A is connected to AP_FIFO_IARG_0, which is connected to M_AXIS_1 of axis_rtl_datamover_0. The data to axis_rtl_datamover_0 is connected through s_axis_txd which is connected to datamover_0. The M_AXIS_SG connects to S00_AXI of axi_interconnect_S_AXI_ACP which connects to S_AXI_ACP of PS7.

4-2-6. Close Vivado by selecting **File > Exit.** Do not save the block design.

Using malloc() Step 5

5-1. Comment out the `data_mover` pragma in mmult.h file. Replace `sds_alloc` and `sds_free` calls with `malloc` and `free` calls in the main.cpp file. Build the project and analyze the data motion network.

The `sds_alloc()` call uses a single physical memory space which may or may not be available in Linux OS. The `sds_alloc()` call uses simple DMA data mover. Linux OS can translate contiguous virtual address into multiple physical address ranges. In Linux OS, `malloc()` can be used to enable single virtual address space mapping to multiple physical address space segments however it must use Scatter Gather (SG) DMA. Memory allocated using `sds_alloc` call must be released using `sds_free` call whereas memory allocated using `malloc` must be freed using `free` calls.

5-1-1. Double-click the **mmult.h** under *lab2* > *src*.

5-1-2. Comment out the pragma for data_mover that you had inserted in the previous section and save the file.

5-1-3. Save the file by selecting **File > Save**

5-1-4. Double-click the **main.cpp** under *lab2* > *src*.

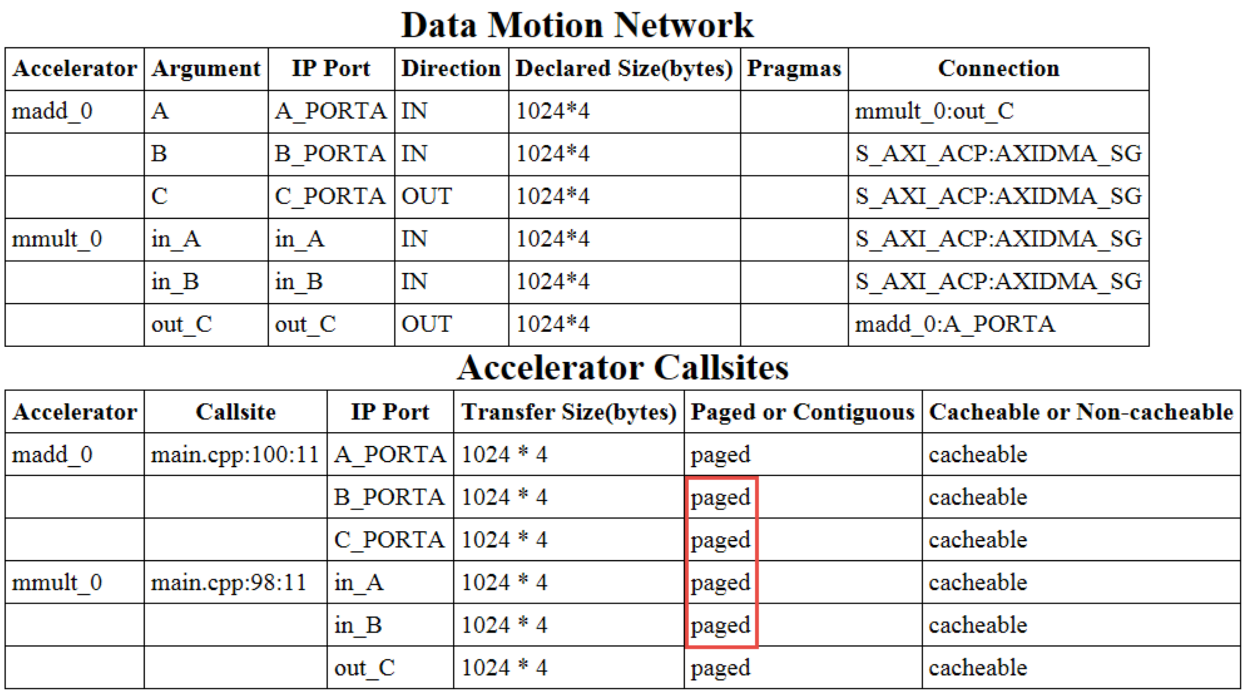
5-1-5. Replace 5 *sds_alloc()* calls with *malloc()* and 10 *sds_free* calls with *free()(CTRL+F to access Find and Replace)* and save the file.

5-1-6. Right-click the top-level folder for the project and click on **Clean Project** in the menu.

5-1-7. Right-click the top-level folder for the project and click on **Build Project** in the menu.

5-1-8. When the build process is complete, select the **lab2** tab so you can access Data Motion link.

5-1-9. Click on the **Data Motion report** link and analyze the result.



Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
madd_0	A	A_PORTA	IN	1024*4		mmult_0:out_C
	B	B_PORTA	IN	1024*4		S_AXI_ACP:AXIDMA_SG
	C	C_PORTA	OUT	1024*4		S_AXI_ACP:AXIDMA_SG
mmult_0	in_A	in_A	IN	1024*4		S_AXI_ACP:AXIDMA_SG
	in_B	in_B	IN	1024*4		S_AXI_ACP:AXIDMA_SG
	out_C	out_C	OUT	1024*4		madd_0:A_PORTA

Accelerator Callsites

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Cacheable or Non-cacheable
madd_0	main.cpp:100:11	A_PORTA	1024 * 4	paged	cacheable
		B_PORTA	1024 * 4	paged	cacheable
		C_PORTA	1024 * 4	paged	cacheable
mmult_0	main.cpp:98:11	in_A	1024 * 4	paged	cacheable
		in_B	1024 * 4	paged	cacheable
		out_C	1024 * 4	paged	cacheable

Figure 9. Data Motion network after applying data_mover pragma

Compared to Figure 2, observe that *Paged* or *Contiguous* column has *paged* type of data movement instead of contiguous. The Connection column shows AXIDMA_SG on S_AXI_ACP.

5-2. Open Vivado IPI design.

5-2-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > SDSoC 2015.4 > Vivado Design Suite > Vivado 2015.4**

5-2-2. Open the design by browsing to *c:\xup\SDSoC\labs\lab2\SDRelease_sds\p0\ipi* and selecting either the **zybo.xpr** or **zed.xpr**.

5-2-3. Click on **Open Block Design** in the *Flow Navigator* pane.

5-2-4. Click on the **show interface connections only** button followed by click on the **regenerate layout** button.

5-2-5. Notice that there are only two datamover instances and only the *S_AXI_ACP* port on the PS7 is used.

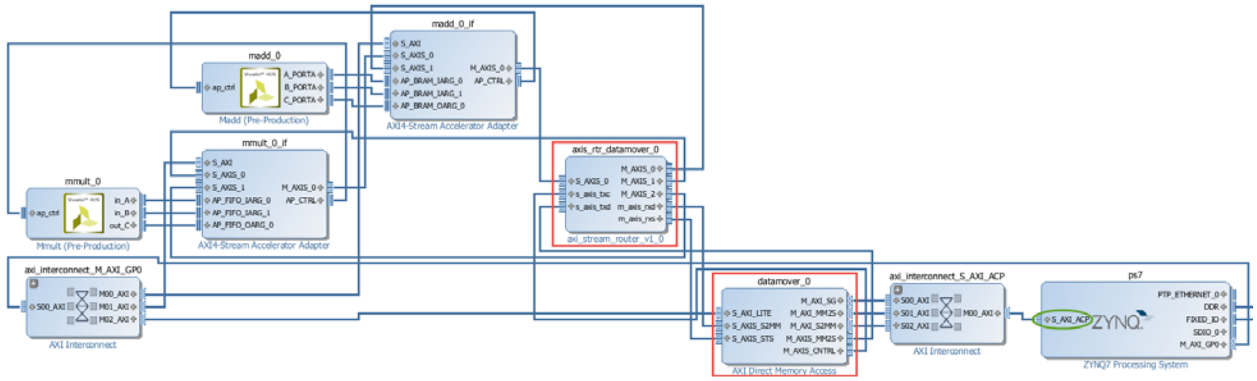


Figure 10. Tracing in_B input datapath of mmult_0 through SG datamover

5-2-6. Close Vivado by selecting **File > Exit.** Do not save the block design.

5-2-7. Close SDSoc by selecting **File > Exit**

Conclusion

In this lab, you used various pragmas to control the generated data motion network and number of data movers. You used sys_port and data_mover pragmas and observed the type of ports used. You also used malloc() and free() calls instead of sds_alloc() and sds_free() calls to handle the non-contiguous memory usage. The built hardware design was analyzed using Vivado IPI and you observed that the number of data movers IP and type of data movers are controlled by the type of pragma used and the type of memory allocation call used.