Creating a System with SDSoC

Introduction

This lab guides you through the process of using SDSoC to create a new project using available templates, marking functions for hardware implementation, building a hardware implemented design, and running the design on either the Zed or ZYBO board.

Objectives

After completing this lab, you will be able to:

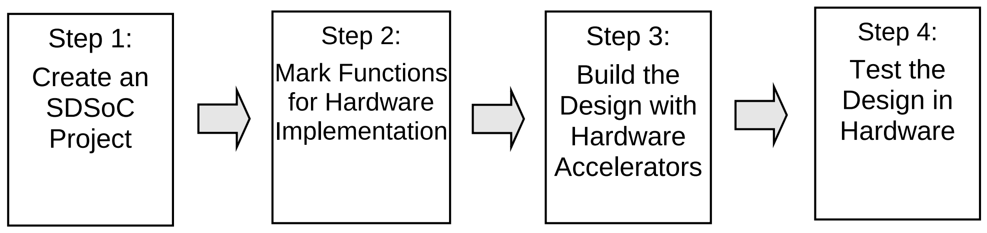
- Create a new SDSoC project for your application from a number of available platforms and project templates
- Mark functions for hardware implementation
- Build your project to generate a bitstream containing the hardware implemented function and an software executable file that invokes this hardware implemented function
- Test the design in 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 four primary steps: You will create an SDSoC project, mark two functions for hardware implementation, build the design with the hardware accelerators, and, test the design in hardware.

General Flow for this Lab


Step 1:
Create an SDSoC Project
Step 2:
Mark Functions for Hardware Implementation
Step 3:
Build the Design with Hardware Accelerators
Step 4:
Test the Design in Hardware

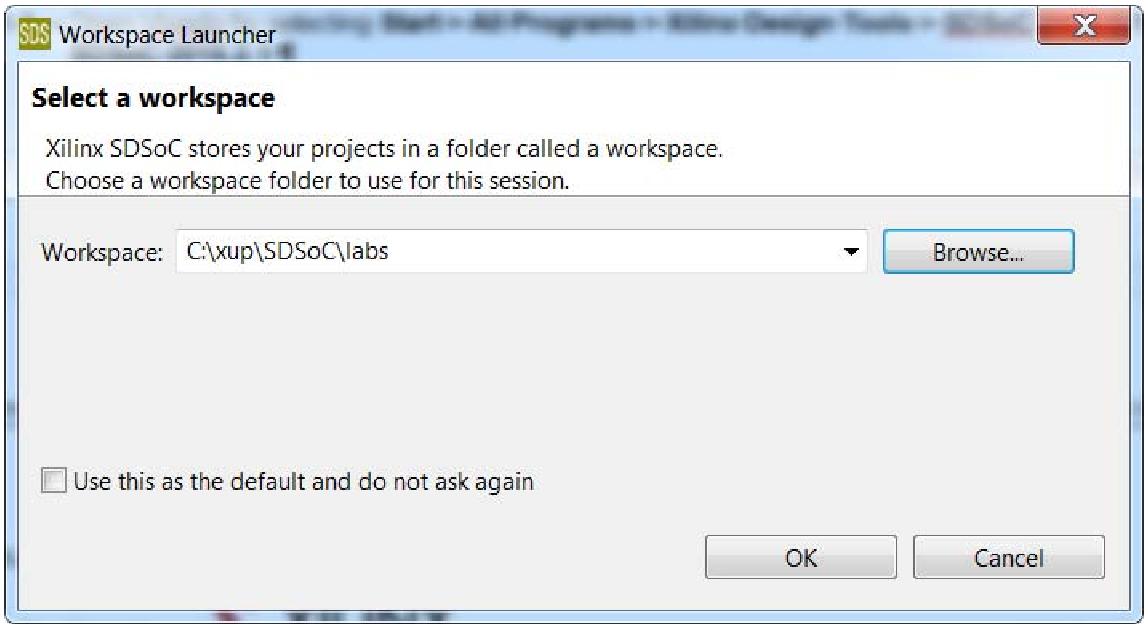
Create an SDSoC Project Step 1

1-1. Launch SDSoC and create a project, called *lab1*, using one of the available templates, 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**, and click **OK**.


Workspace Launcher

Select a workspace

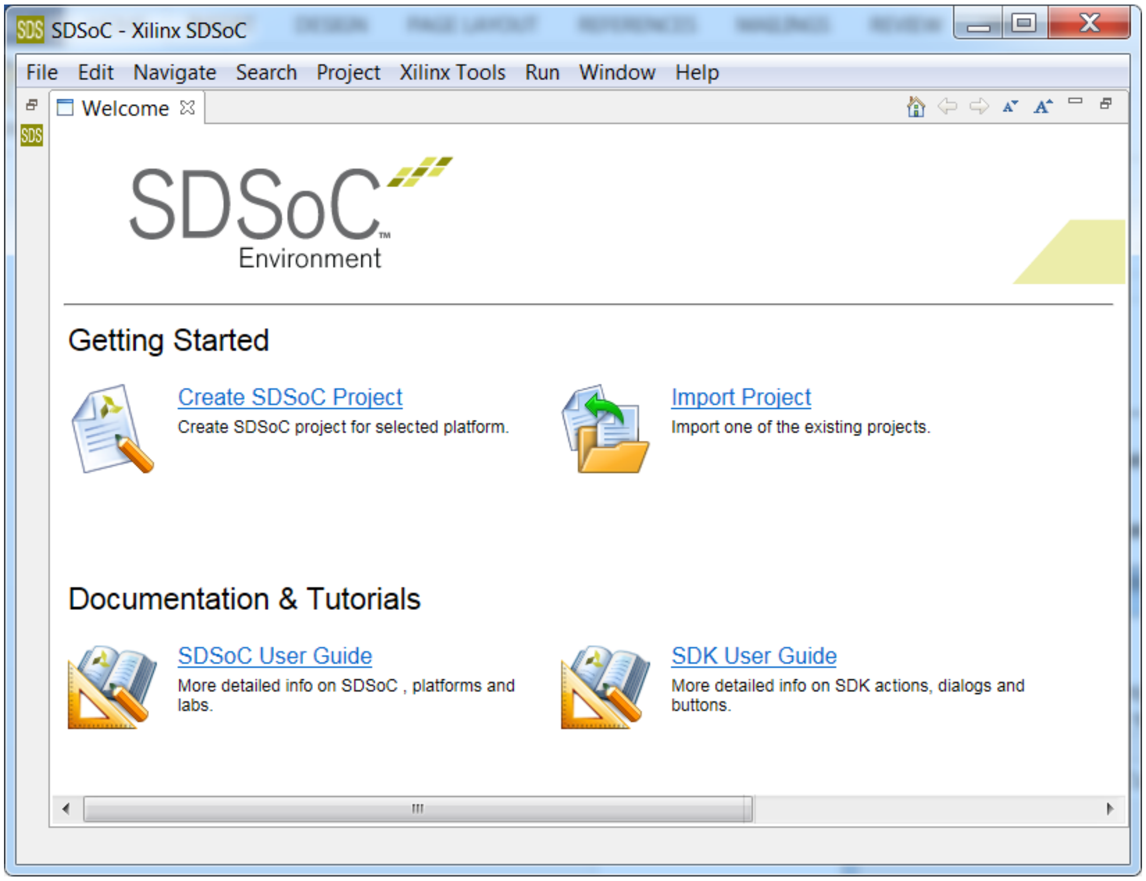
Xilinx SDSoC stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\xup\SDSoC\labs

Figure 1. Selecting a workspace

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

The SDSoC development environment window will appear showing the Welcome tab.


SDSoC - Xilinx SDSoC
File Edit Navigate Search Project Xilinx Tools Run Window Help
Welcome
SDSoC Environment

Getting Started

Create SDSoC Project
Create SDSoC project for selected platform.

Import Project
Import one of the existing projects.

Documentation & Tutorials

SDSoC User Guide
More detailed info on SDSoC , platforms and labs.

SDK User Guide
More detailed info on SDK actions, dialogs and buttons.

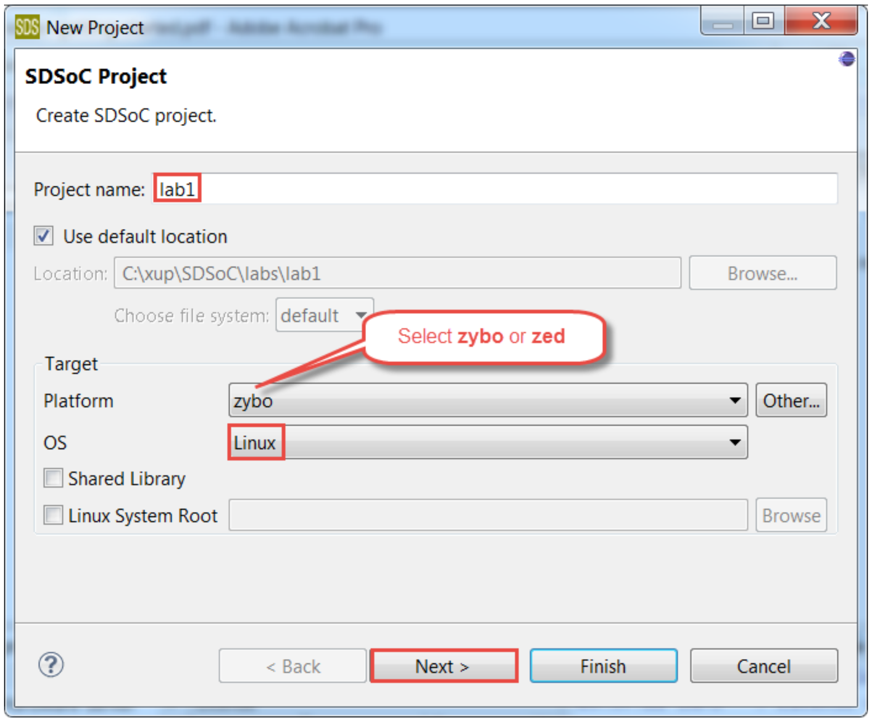
Figure 2. The SDSoC development environment with Welcome tab

From here you can create a new project, import an existing project, access the SDSoC user guide, and access the SDK user guide by clicking on the desired link.

1-1-4. Click **X** on the *Welcome* tab to close it, and you will see the empty workspace in the background.

1-1-5. Select **File > New > SDSoc Project** to open the *New Project* GUI.

1-1-6. Enter **lab1** 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.**



New Project

SDSoC Project

Create SDSoC project.

Project name: lab1

☑ Use default location

Location: C:\xup\SDSoC\labs\lab1 Browse...

Choose file system: default

Target

Platform: zybo Other...

OS: Linux

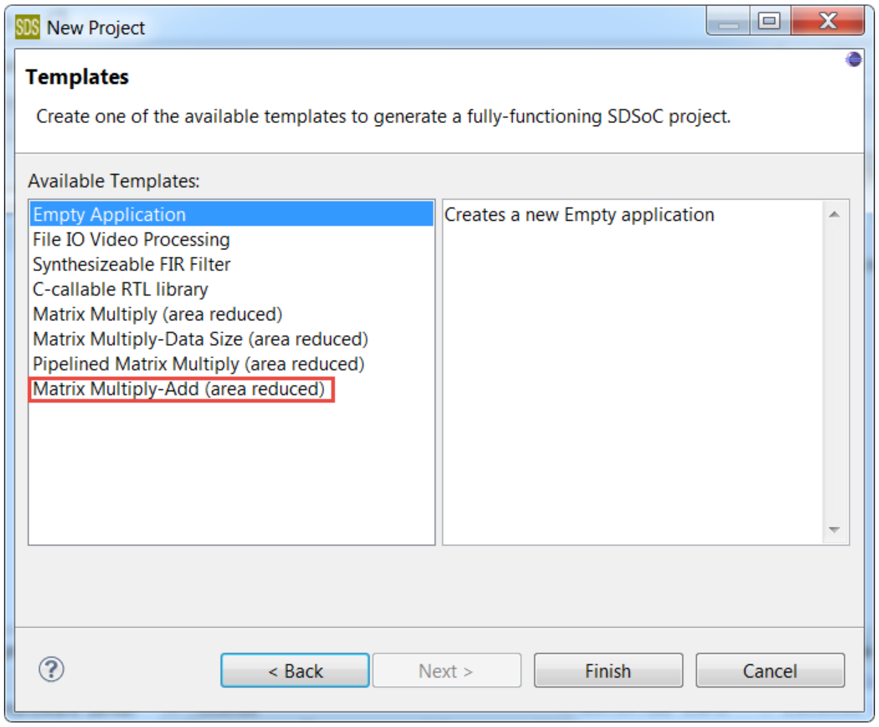
☐ Shared Library

☐ Linux System Root Browse

Figure 3. Naming the project and selecting target platform and OS

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

1-1-7. Select **Matrix Multiply-Add (area reduced)** in case of *zybo* or **Matrix Multiplication and Addition** in case of *zed* as the source.



New Project

Templates

Create one of the available templates to generate a fully-functioning SDSoC project.

Available Templates:

- Empty Application
- File IO Video Processing
- Synthesizeable FIR Filter
- C-callable RTL library
- Matrix Multiply (area reduced)
- Matrix Multiply-Data Size (area reduced)
- Pipelined Matrix Multiply (area reduced)
- Matrix Multiply-Add (area reduced)

Creates a new Empty application

Figure 4. Selecting from available templates

1-1-8. Click **Finish**.

The created project will be displayed. In the left, you will see *Project Explorer* under which the **lab1** project directory will be displayed (you may have to expand the folder). It shows the *Includes* and *src* folders. The *src* folder contains the source files which were copied from the template source directory located at <SDSoC_install_directory>\samples\<template_name>. The **lab1** folder also shows the **project.sdsoc** project file. Double-clicking on it will display what you see in the right-side pane.

In the SDSoC Project Overview pane, you see the *Overview* tab being displayed. Note that another tab, called *Platform*, is also available. In the *Overview* tab, you see *General*, *Hardware Functions, Options,* and *Actions* sub-areas. From here you will be able to change options, identify the function(s) that will be implemented in hardware, setup for debugging and estimation, and access various reports.

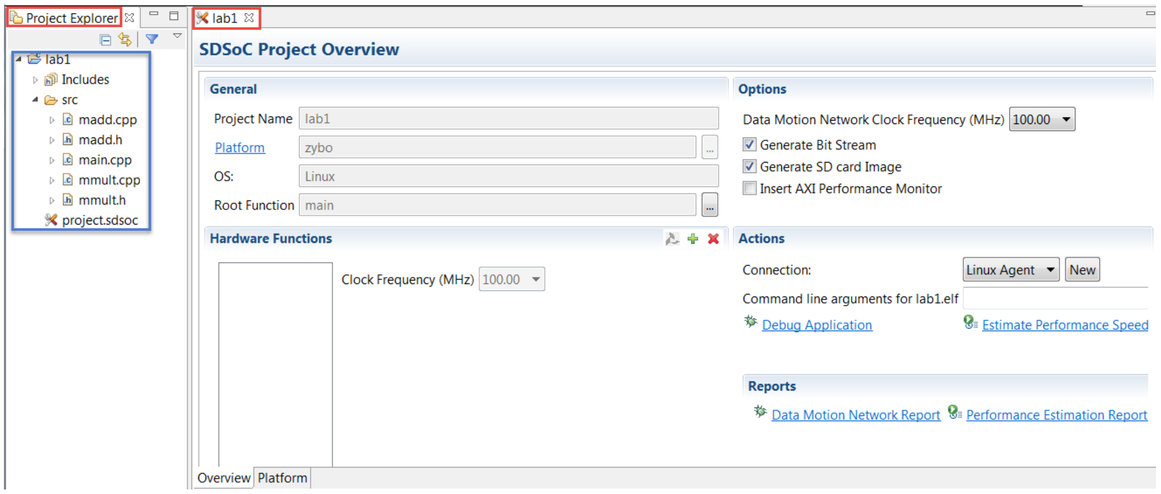


Figure 5. Created SDSoC project

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. Click on the "+" sign in the Hardware Functions area to open up the list of software functions in the design.

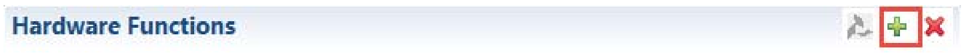


Figure 6. Invoking functions list to select functions which can be hardware target

The *Select functions for hardware acceleration* window will open. The source files will be scanned and available functions will be displayed.

2-1-2. While holding down the Ctrl key, select *mmult* and *madd* and click **OK**.

Alternatively, you can expand mmult.cpp and madd.cpp in the Project Explorer tab, right click on mmult and madd functions, and select **Toggle HW/SW** (when the function is already marked for hardware, you will see **Toggle HW/SW [H]**). When you have a source file open in the editor, you can also select hardware functions in the Outline window.

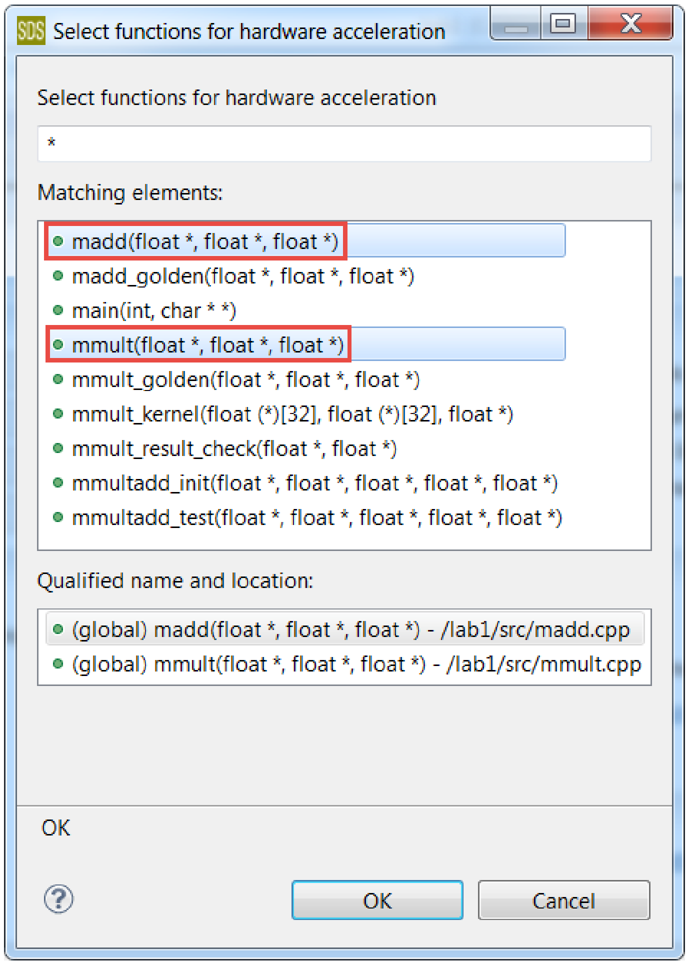

Select functions for hardware acceleration
Select functions for hardware acceleration
*
Matching elements:
madd(float *, float *, float *)
madd_golden(float *, float *, float *)
main(int, char * *)
mmult(float *, float *, float *)
mmult_golden(float *, float *, float *)
mmult_kernel(float (*)[32], float (*)[32], float *)
mmult_result_check(float *, float *)
mmultadd_init(float *, float *, float *, float *, float *)
mmultadd_test(float *, float *, float *, float *, float *)
Qualified name and location:
(global) madd(float *, float *, float *) - /lab1/src/madd.cpp
(global) mmult(float *, float *, float *) - /lab1/src/mmult.cpp
OK
OK
Cancel

Figure 7. Selecting functions for hardware target

2-1-3. The two function names will be added into the Hardware Functions window.

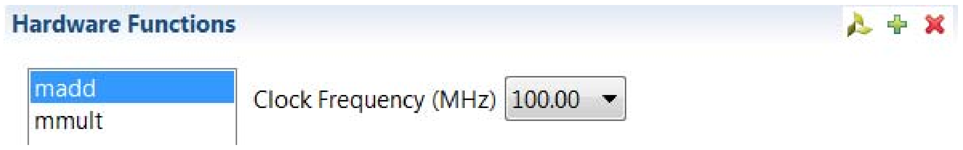
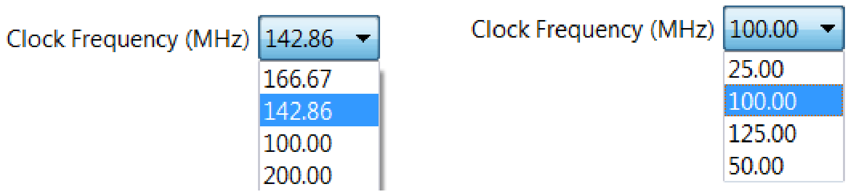


Figure 8. Target hardware accelerators entries updated

You can select each function, click on the drop-down button of the *Clock Frequency,* and select a clock speed that will be applied to that function. You can also select multiple functions (CTRL + Click) and apply the clock frequency to all of them. Note the default clock frequency of 142.86 MHz for Zed and 100.00 MHz for Zybo. These speeds are defined in the platform that was selected for the design.



(a) Zed **(b) Zybo**

Figure 9. Available clocks for the accelerators in a given platform

2-1-4. Leave the clocks set to the default frequency of 142.86 MHz for Zed and 100.00 MHz for Zybo for both functions.

2-1-5. Expand the **lab1** folder in the Project Explorer pane and notice that the functions are marked for hardware implementation in *madd.cpp* and *mmult.cpp* files.

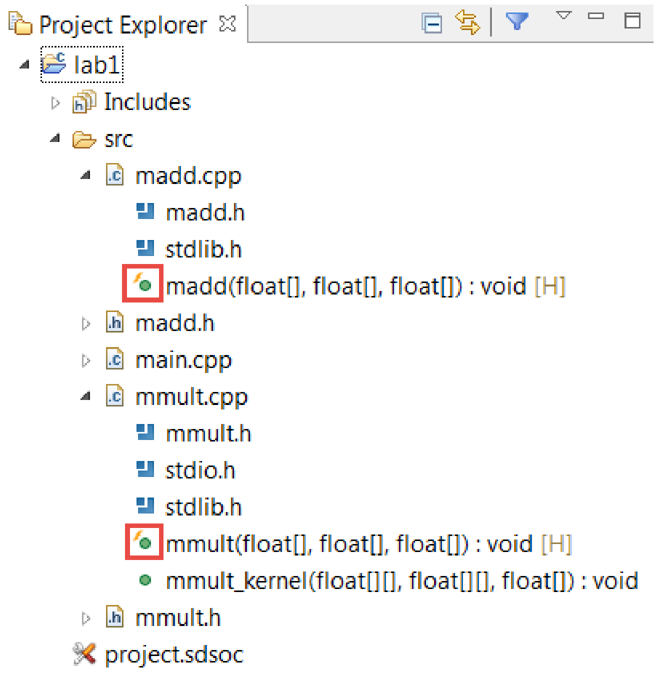

Project Explorer
lab1
Includes
src
madd.cpp
madd.h
stdlib.h
madd(float[], float[], float[]) : void [H]
madd.h
main.cpp
mmult.cpp
mmult.h
stdio.h
stdlib.h
mmult(float[], float[], float[]) : void [H]
mmult_kernel(float[][], float[][], float[]) : void
mmult.h
project.sdsoc

Figure 10. Expanded project view indicating the selected functions for hardware implementation

Build the Design with Hardware Accelerators Step 3

3-1. Select SDRelease configuration and build the project. When done, analyze the data motion network through the report and built hardware through Vivado IPI.

3-1-1. Right-click on **lab1** in the **Project Explorer** and select **Build Configurations > Set Active** to see possible configurations and what is currently selected.

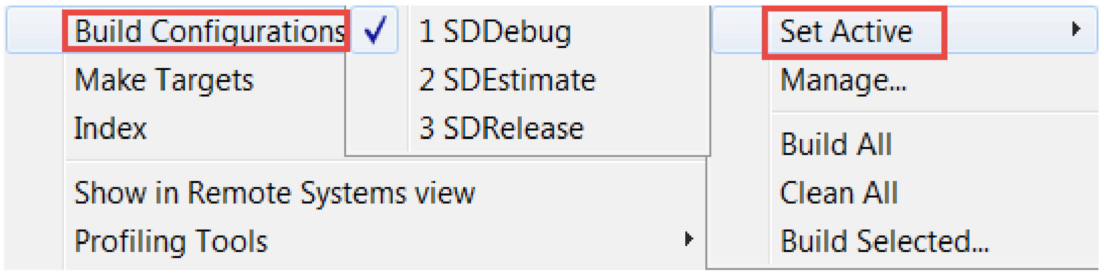


Figure 11. Selecting build configuration

3-1-2. Select **Build Configurations > Set Active > SDRelease**

The SDRelease build configuration uses a higher compiler optimization setting than the SDDebug build.

Building the project and generating the results in the next step may take 20 to 30 minutes. Alternatively, you can import provided pre-built files into your workspace and load the results.

3-1-3. Right-click on **lab1** and select **Build Project**

The output from the SDSoC compiler can be viewed in the Console tab. The functions selected for hardware are compiled into IP blocks using Vivado HLS. The IP blocks are then integrated into a Vivado design based on the selected base platform. Vivado will carry out synthesis, and place and route tools to build a bitstream. The software functions that have been moved to hardware will be replaced by function calls to the hardware blocks, and the software will be compiled to generate an ELF executable file.

This may take about 25 to 30 minutes.

You can also load the results by importing the pre-built files into your workspace with these steps:

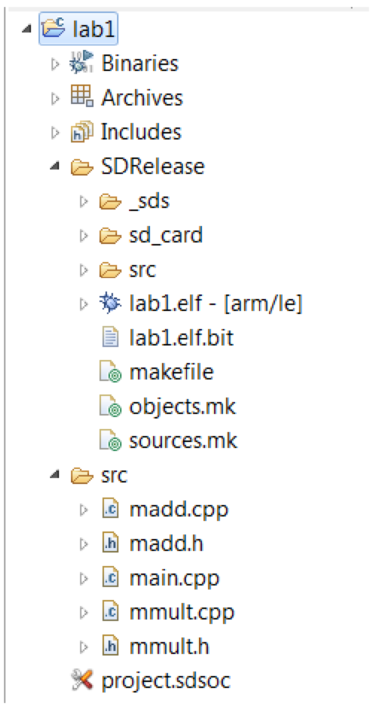
Select **File > Import** and then select **General > Existing Projects into Workspace** and click **Next**.

Select **Select archive file** and click **Browse** to navigate to *c:\xup\SDSoC\sources\lab1*

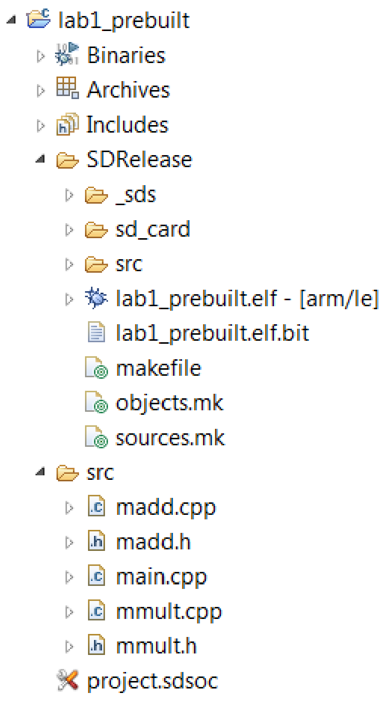
Select **lab1_prebuilt.zip**, and click **Open**

Click **Finish**.

3-1-4. Expand the **lab1** (or **lab1_prebuilt** if you have imported the project) 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, sd_card, src** folders along with **lab1.elf ((**or **lab1_prebuilt.elf** if you have imported the project) [executable]**, lab1.elf.bit** (or **lab1_prebuilt.elf.bit** if you have imported the project) [hardware bit file] and several make files.



```
lab1
  Binaries
  Archives
  Includes
  SDRelease
    _sds
    sd_card
    src
    lab1.elf - [arm/le]
    lab1.elf.bit
    makefile
    objects.mk
    sources.mk
  src
    madd.cpp
    madd.h
    main.cpp
    mmult.cpp
    mmult.h
  project.sdsoc
```



```
lab1_prebuilt
  Binaries
  Archives
  Includes
  SDRelease
    _sds
    sd_card
    src
    lab1_prebuilt.elf - [arm/le]
    lab1_prebuilt.elf.bit
    makefile
    objects.mk
    sources.mk
  src
    madd.cpp
    madd.h
    main.cpp
    mmult.cpp
    mmult.h
  project.sdsoc
```

(a) Generated **(b) Imported**

Figure 12. Project Explorer folders

The **sd_card** folder contains files and sub-directory (depending on the target OS) which can be copied to a SD card and then used to boot the system to test the design in hardware.

The **src** folder contains object and debug information carrying files of the main function as well as the hardware target files.

The **_sds** folder contains various sub-folders and files generated by SDSoC as well as other underlying used tools.

3-1-5. Expand the _sds folder.

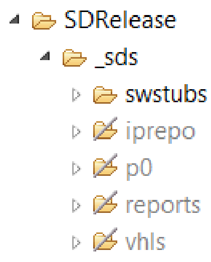


Figure 13. The generated _sds folder content

Notice that there are five sub-folders out of which four are greyed-out and one is shown as normal. The directory which is not greyed-out (*swstubs*), indicates the folder and its contents were generated by SDSoC. The greyed folders (*iprepo, p0, reports, vhls*) were generated by the underlying tools.

The *swstubs* contains various source files to handle data motion as well as communication with the hardware accelerators. It also contains various stub files and generated libraries.

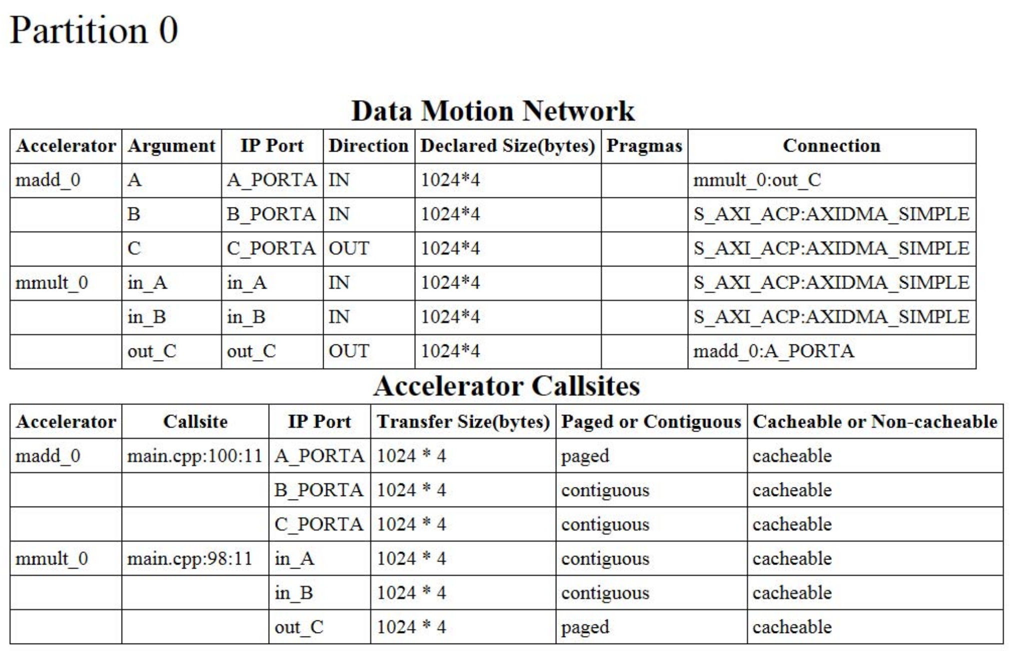
The *iprepo* and *vhls* folders are generated by Vivado HLS. The *iprepo* has a sub-folders for each hardware function. The *vhls* folder contains the complete HLS solution for each function. It also contains tcl files used to generate the solution.

The *p0* folder consists of *ipi* and *sd_card* sub-folders which includes the Vivado IPI project (synthesis and implementation tcl files and results) and the generated SD card contents. The project file, located in *ipi* sub-folder, has an extension of **xpr**. The project can be opened with Vivado to display the block diagram of the generated system-level hardware.

The *reports* folder consists of log files and the **data_motion.html** containing the data motion network report.

3-1-6. In the SDSoC Project Overview window, under the *Reports* pane, click on **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.



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 14. Data motion network and accelerator callsites

There are two accelerated functions- madd and mmult. They are given instance names of 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 is 4096 bytes or 1024 words on each ports of the two accelerators.

3-1-7. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > SDSoC 2015.4 > Vivado Design Suite > Vivado 2015.4**

3-1-8. Open the design by browsing to *c:\xup\SDSoC\labs\lab1\SDRelease_sds\p0\ipi* and selecting either the **zybo.xpr** or **zed.xpr**.

3-1-9. 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).

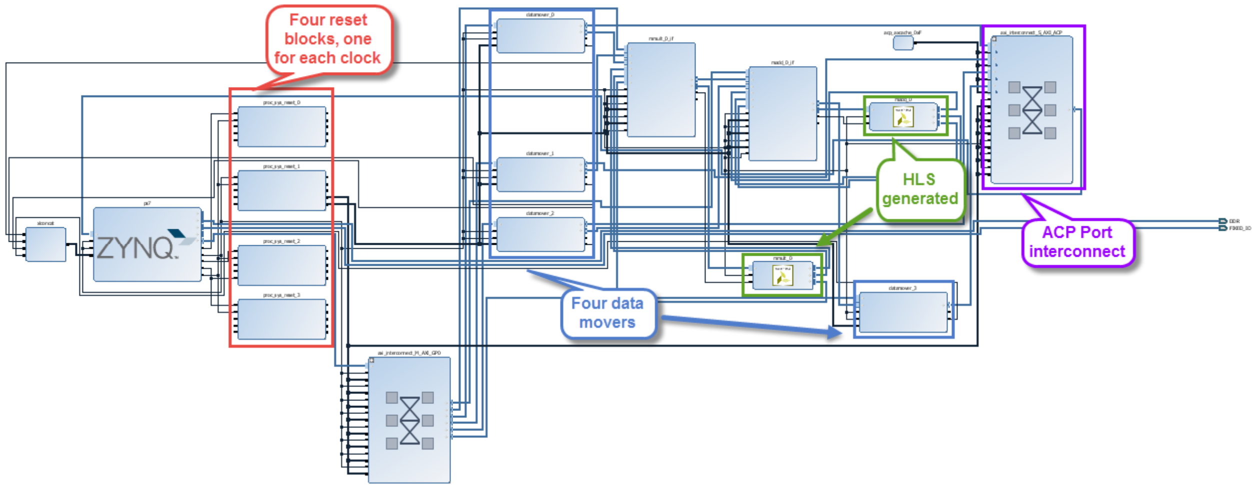


Figure 15. The generated block design

As seen before, this platform supports four clocks. There are four reset blocks, one for each clock. As only the 142.86 MHz clock in Zed or the 100 MHz clock in Zybo was used for the accelerators, it connects only one reset block is connected (second from top). The other blocks will be optimized away during synthesis.

We have targeted two functions for hardware acceleration, and hence two HLS created blocks are generated and included in the design.

Since there are four data mover connections using ACP, there are four data movers (indicated by blue color). The four data movers connect to the processor's ACP interface using the ACP interconnect instance.

The mmult_o output (Out C) has a direction path to madd_0 input (A) is a direct connection, i.e. it does not go through any data movers. You can see the path by zooming in and tracing the connections (highlighted below).

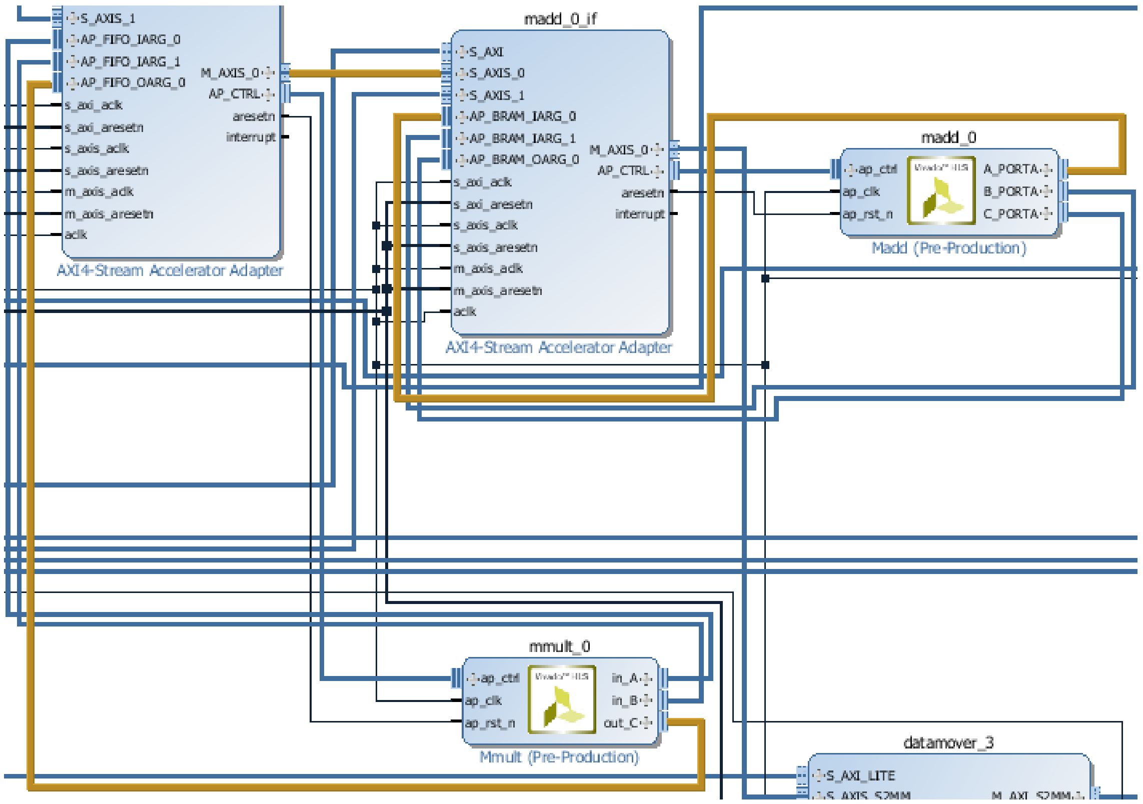


Figure 16. Non data mover connection between mmult_0 and madd_0

3-1-10. Close Vivado by selecting **File > Exit**

3-2. Open the main.cpp, mmult.cpp, and madd.cpp files under SDRelease > _sds> swstubs and lab1 > src folders and understand the added code segments.

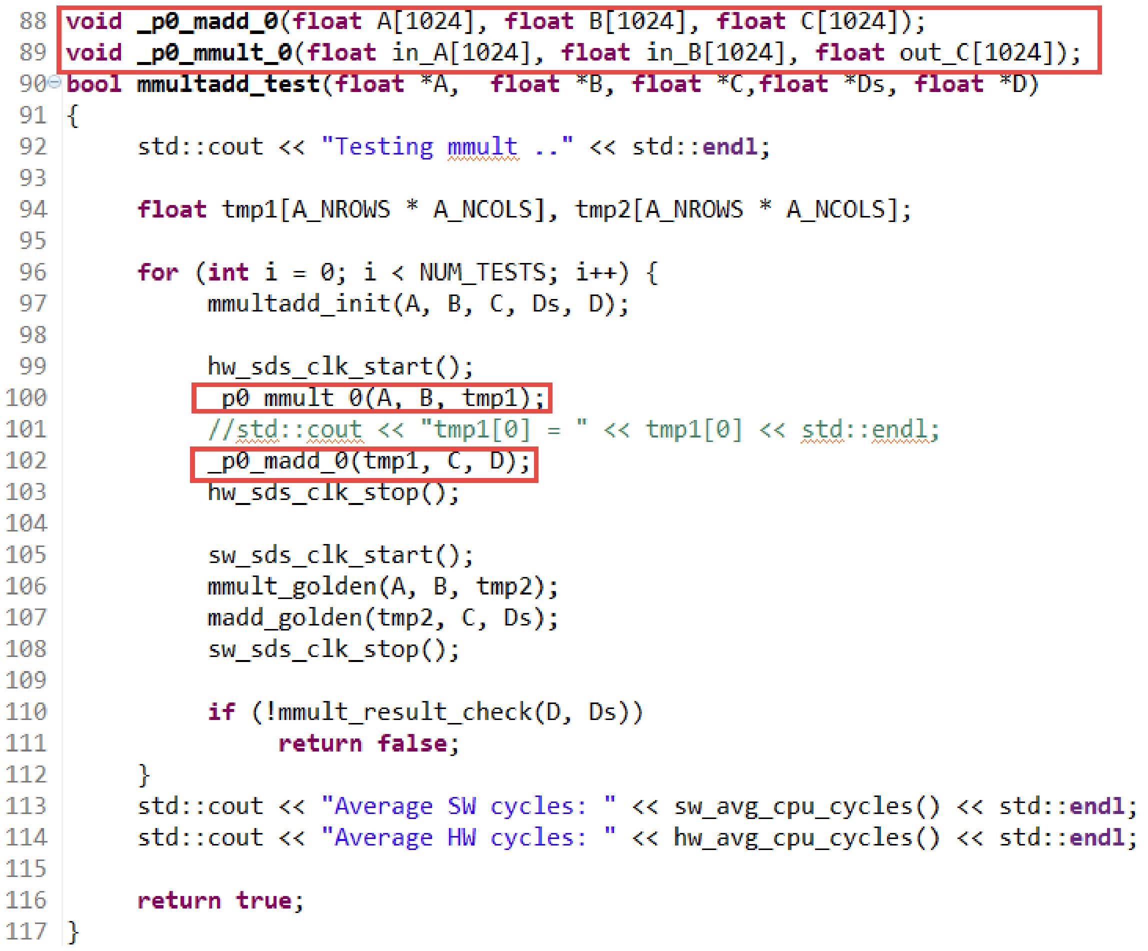
3-2-1. Expand **lab1 > src** and double-click on *main.cpp* to see its content.

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

Note that line 140 calls the *multadd_test* function call. The *multadd_test* function is defined between lines 88 and 115, which in turn calls *mmult* at line 98 and *madd* at line 100. The *mmult* function is defined in *mmult.cpp* (lines 31-57) and the *madd* function is defined in *madd.cpp* (lines 4-13). Both these files are under the same *src* folder.

3-2-2. Expand **SDRelease > _sds> swstubs** and double-click on *main.cpp* to see its content.

Note that line 142 calls multadd_test function call as seen in the original source file. The multadd_test function is now preceded by two function prototypes (lines 88 and 89) and the function is defined between lines 90 and 117. On lines 100 and 102 it makes calls to _p0_mmult_0 and _p0_madd_0 replacing the original calls.



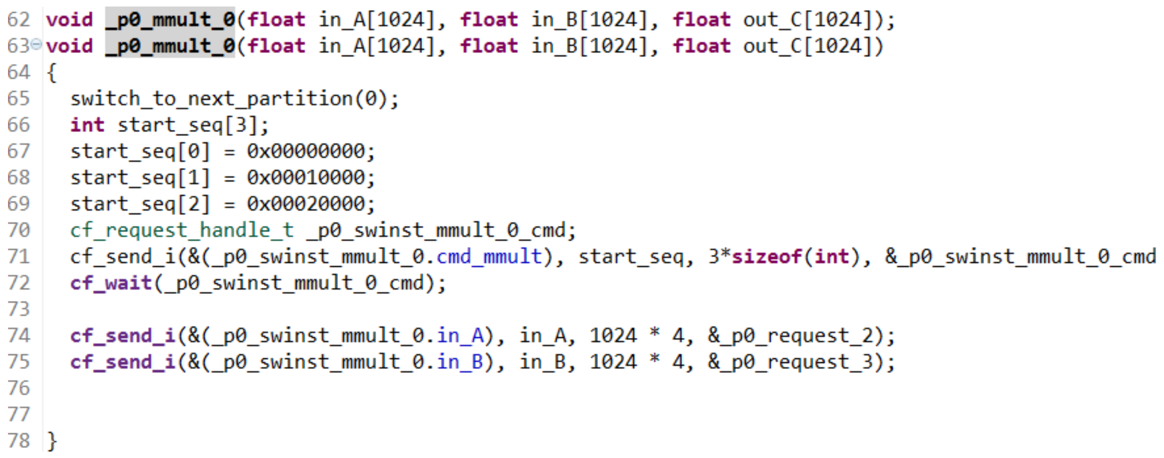
```
void _p0_madd_0(float A[1024], float B[1024], float C[1024]);
void _p0_mmult_0(float in_A[1024], float in_B[1024], float out_C[1024]);
bool mmultadd_test(float *A,  float *B, float *C,float *Ds, float *D)
{
     std::cout << "Testing mmult .." << std::endl;

     float tmp1[A_NROWS * A_NCOLS], tmp2[A_NROWS * A_NCOLS];

     for (int i = 0; i < NUM_TESTS; i++) {
          mmultadd_init(A, B, C, Ds, D);

          hw_sds_clk_start();
          _p0_mmult_0(A, B, tmp1);
          //std::cout << "tmp1[0] = " << tmp1[0] << std::endl;
          _p0_madd_0(tmp1, C, D);
          hw_sds_clk_stop();

          sw_sds_clk_start();
          mmult_golden(A, B, tmp2);
          madd_golden(tmp2, C, Ds);
          sw_sds_clk_stop();

          if (!mmult_result_check(D, Ds))
               return false;
     }
     std::cout << "Average SW cycles: " << sw_avg_cpu_cycles() << std::endl;
     std::cout << "Average HW cycles: " << hw_avg_cpu_cycles() << std::endl;

     return true;
}
```

Figure 17. Updated main.cpp file content

3-2-3. Double-click on the **mmult.cpp** under **SDRelease > _sds> swstubs** to see its content.

The _p0_mmult_0 function is defined in lines 63 through 78, replacing the original functionality with the data transfer using the cf_send command. Similarly, the madd function is updated in madd.cpp (lines 19-38). It additionally uses cf_receive call to get the results.

The subsequent labs will discuss these function calls.



```
62 void _p0_mmult_0(float in_A[1024], float in_B[1024], float out_C[1024]);
63 void _p0_mmult_0(float in_A[1024], float in_B[1024], float out_C[1024])
64 {
65   switch_to_next_partition(0);
66   int start_seq[3];
67   start_seq[0] = 0x00000000;
68   start_seq[1] = 0x00010000;
69   start_seq[2] = 0x00020000;
70   cf_request_handle_t _p0_swinst_mmult_0_cmd;
71   cf_send_i(&(_p0_swinst_mmult_0.cmd_mmult), start_seq, 3*sizeof(int), &_p0_swinst_mmult_0_cmd
72   cf_wait(_p0_swinst_mmult_0_cmd);
73 
74   cf_send_i(&(_p0_swinst_mmult_0.in_A), in_A, 1024 * 4, &_p0_request_2);
75   cf_send_i(&(_p0_swinst_mmult_0.in_B), in_B, 1024 * 4, &_p0_request_3);
76 
77 
78 }
```

Figure 18. The updated mmult.cpp file

Test the Design in Hardware Step 4

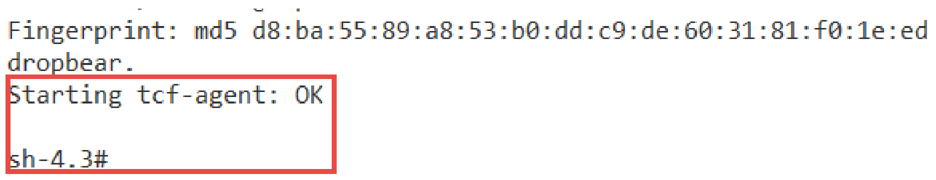
4-1. Copy the files located under SDRelease\SD_Card folder into a SD card. Place the card into the board. Configure the board to boot from SD. Connect and power up the board. Establish serial communication. Execute the application.

4-1-1. Using the Windows Explorer, copy all the files located under either *lab1\SDRelease\SD_Card* folder or *lab1_prebuilt\SDRelease\SD_Card* to the SD (Zed) or Micro-SD (Zybo) card.

4-1-2. Use **SDK Terminal** window on *Zed* or **Terminal** window on *Zybo* or third party terminal emulator program like PuTTy, HyperTerminal, TeraTerm programs.

4-1-3. Select appropriate C OM port (depending on your computer), and configure the terminal with the 115200 baud rate.

4-1-4. You should see the output in the console.



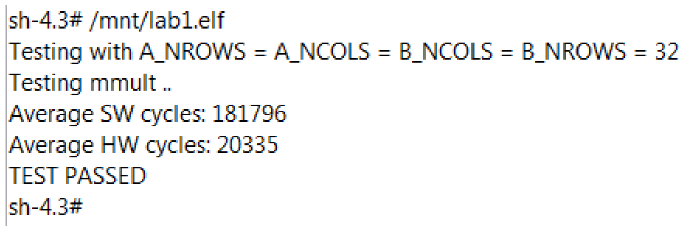
```
Fingerprint: md5 d8:ba:55:89:a8:53:b0:dd:c9:de:60:31:81:f0:1e:ed
dropbear.
Starting tcf-agent: OK

sh-4.3#
```

Figure 19. System bootup output

If you don't see the output then you can press PS-SRST push-button (Red) on the board.

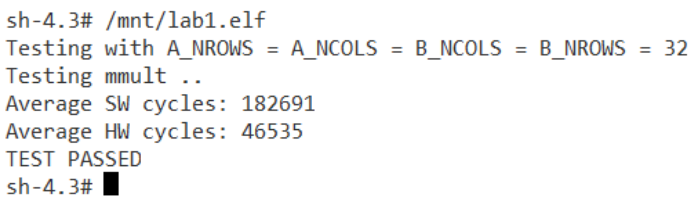
4-1-5. In the Terminal window, either enter **/mnt/lab1.elf** or **/mnt/lab1_prebuilt.elf** at the command prompt and hit the Enter key.

The program will be executed and the result will be displayed showing the number of cycles software execution takes vs the number of cycles taken using the hardware accelerators. It also shows the number rows and columns of the matrices.



```
sh-4.3# /mnt/lab1.elf
Testing with A_NROWS = A_NCOLS = B_NCOLS = B_NROWS = 32
Testing mmult ..
Average SW cycles: 181796
Average HW cycles: 20335
TEST PASSED
sh-4.3#
```

(a) Zed



```
sh-4.3# /mnt/lab1.elf
Testing with A_NROWS = A_NCOLS = B_NCOLS = B_NROWS = 32
Testing mmult ..
Average SW cycles: 182691
Average HW cycles: 46535
TEST PASSED
sh-4.3#
```

(a) Zybo

Figure 20. Program output

4-1-6. Close SDSoc by selecting **File > Exit**

4-1-7. Turn OFF the power to the board.

Conclusion

In this lab, you created a project in the SDSoC Development Environment using one of the available project templates. You then identified the functions which you wanted to put in the PL section of the Zynq chip to improve the performance. Once the system was built, you analyzed the Data Motion network and the created Vivado IPI project. Finally, you copied the relevant files on a SD card and verified the design in hardware.