Creating a System with SDx

Introduction

This lab guides you through the process of using SDx to create a new SDSoC 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:

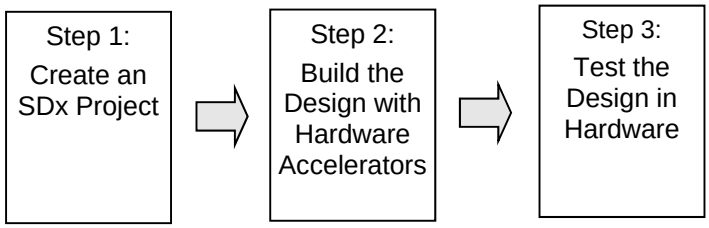
- Create a new SDx project for your application from a number of available platforms and project templates
- Mark functions for hardware implementation
- Build your project to generate a bitstream containing the hardware implemented function and an software executable file that invokes this hardware 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 SDx 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



Create an SDx Project Step 1

1-1. Launch SDx and create a project, called *lab1*, using one of the available templates, targeting the Zed or Zybo board.

1-1-1. Open SDx by selecting **Start > All Programs > Xilinx Design Tools > SDx 2017.2 > SDx IDE 2017.2**

The Workspace Launcher window will appear.

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

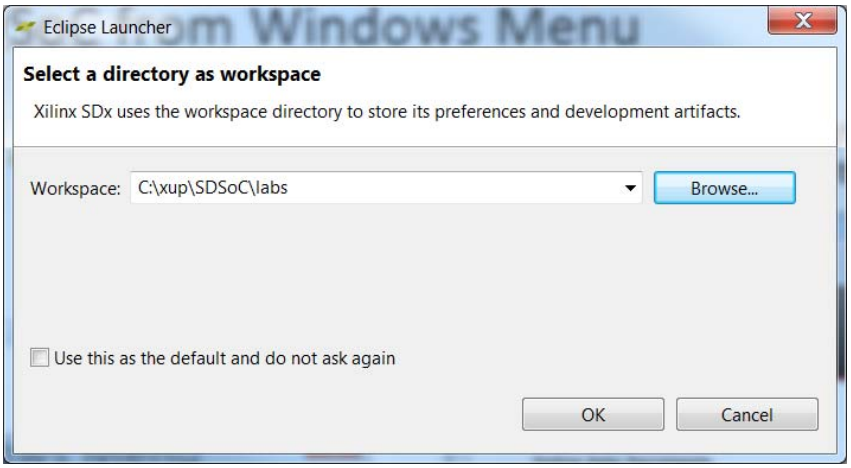


Figure 1. Selecting a workspace

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

The SDx development environment window will appear showing the *Welcome* tab.

Figure 2. The SDx development environment with Welcome tab

From here you can create a new project, create an application project, import an existing project, and access the tutorials and web resources by clicking on the desired link.

1-1-4. Click on the **Create SDx Projec**t link and *Create a New SDx Project* form will appear. Enter **lab1** in the *Project name* field**.**

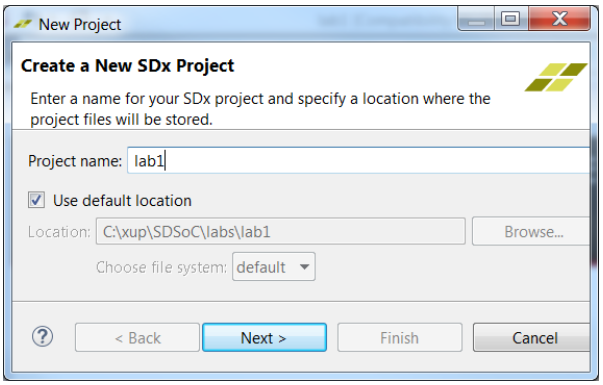


Figure 3. Entering project name

You could have clicked Click **X** on the *Welcome* tab to close it, and you would have seen the empty workspace in the background. From there, you would have selected **File > New > Xilinx SDx Project** to open the *New Project* GUI. You then would have entered **lab1** as the project name.

1-1-5. Click **Next** to see *Choose Hardware Platform* window showing various available platforms.

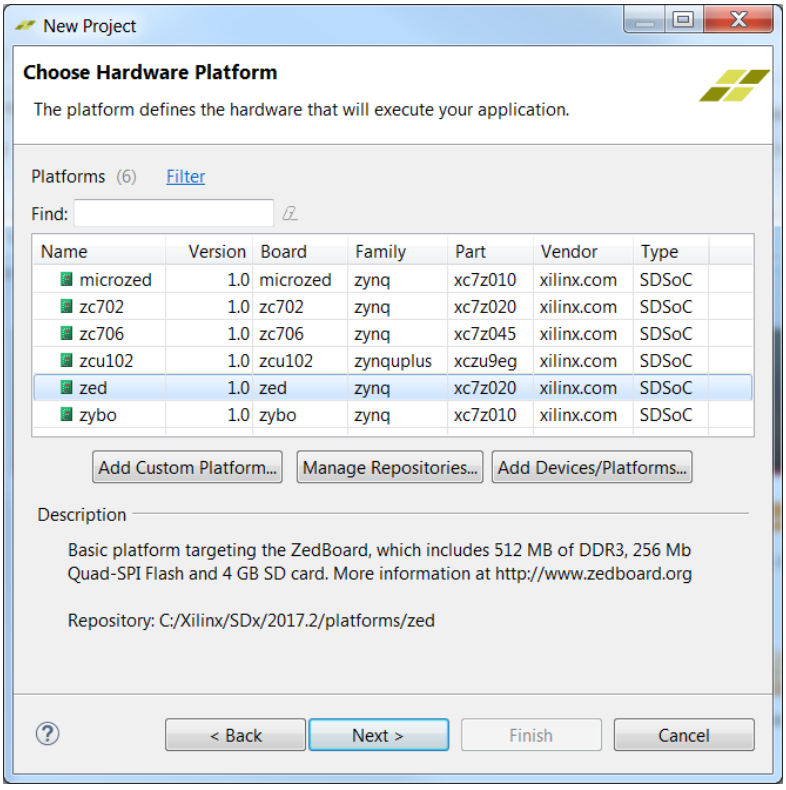


Figure 4. Available hardware platforms

1-1-6. Select either *zybo* or *zed* (depending on the board you are using) and click **Next**.

1-1-7. In the *Choose Software Platform and Target CPU* window select *Linux* as the target OS, and click **Next**.

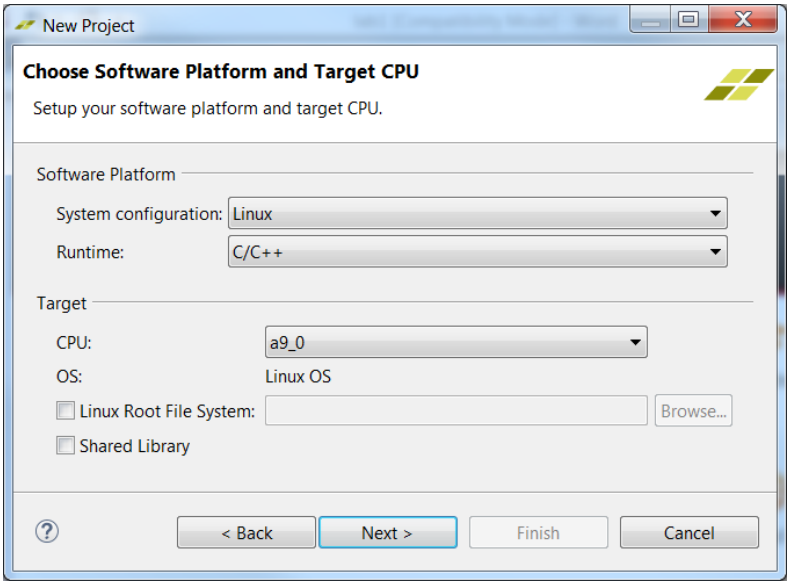


Figure 5. Naming the project and selecting target platform and OS

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

1-1-8. Select **Matrix Multiplication and Addition (area reduced)** in case of *zybo* or **Matrix Multiplication and Addition** in case of *zed* as the source.

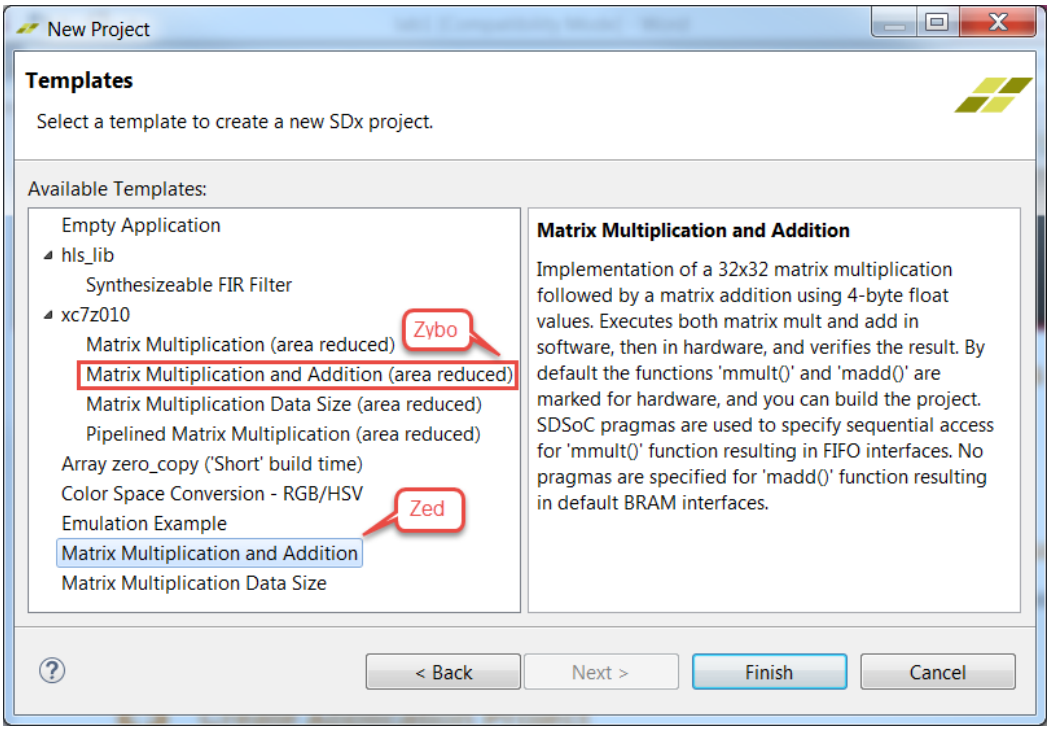


Figure 6. Selecting from available templates

1-1-9. 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 <SDx_install_directory>\samples\<template_name>. The **lab1** folder also shows the **project.sdx** project file. Double-clicking on it will display what you see in the right-side pane.

In the SDx Project Settings pane, you see *General*, *Hardware Functions,* and *Options* areas. From here you will be able to change options, identify/modify the function(s) that will be implemented in hardware, setup for debugging and estimation, and access various reports.

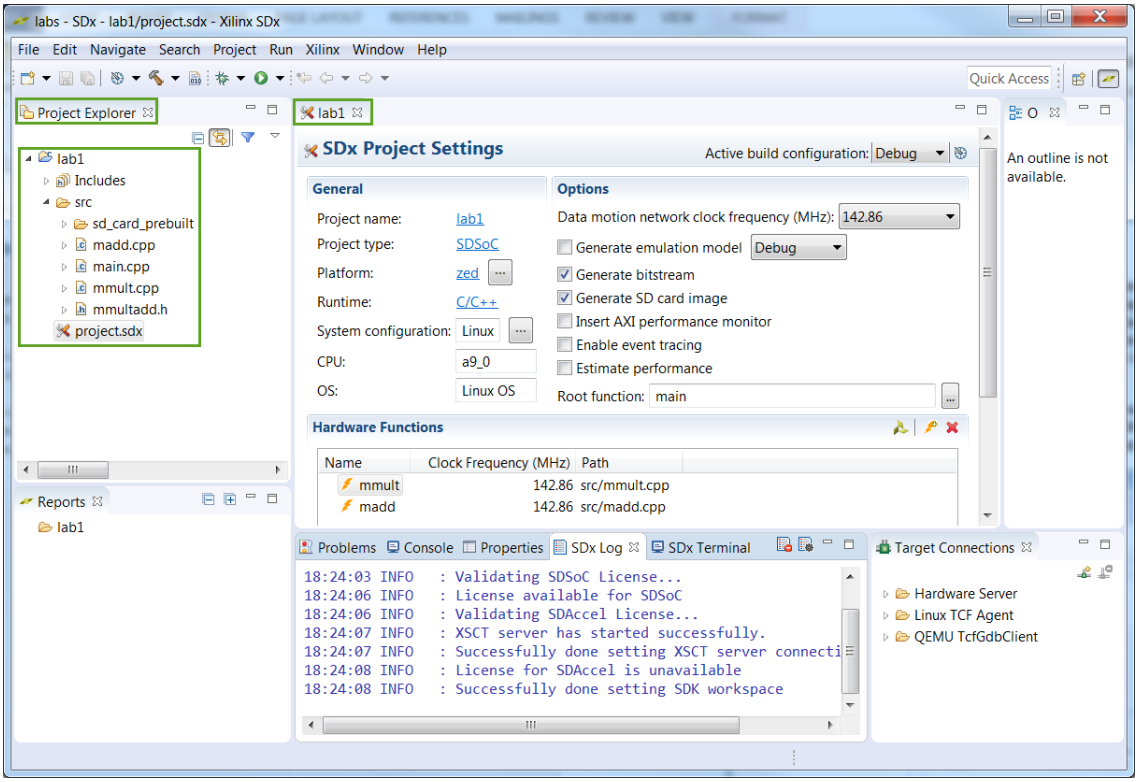
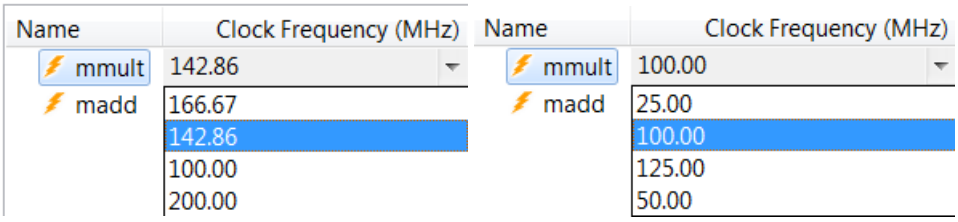


Figure 7. Created SDx project

1-2. Managing functions to accelerate using various buttons.

1-2-1. Notice the two functions, *mmult* and *madd*, are already targeted for hardware acceleration. Also, the data movement frequency selected is 142.86 MHz for Zed and 100.00 MHz for Zybo. You can change the frequency by clicking in the corresponding field and selecting appropriate frequency.



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

Figure 8. Available frequencies

1-2-2. You can add additional functions by click on the *Add HW Functions…* () button. When clicked, the source files will be scanned and the available functions in the project will be displayed along with possible candidates.

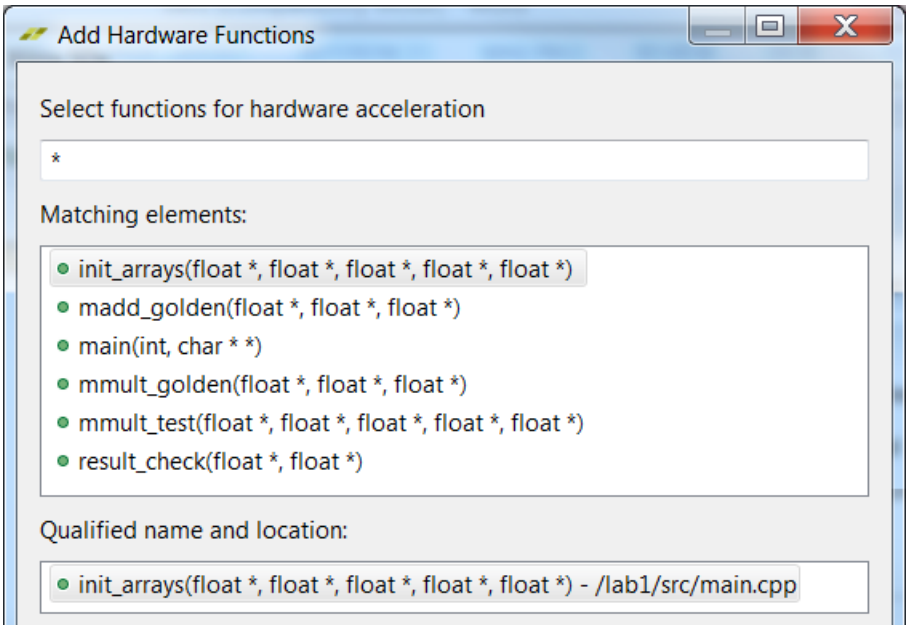


Figure 9. Functions in the project

1-2-3. Click **Cancel** since we do not want to add any other functions.

1-2-4. You can remove an already added functions by selecting their corresponding entries and click on the *Remove HW Functions* () button.

1-2-5. 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.

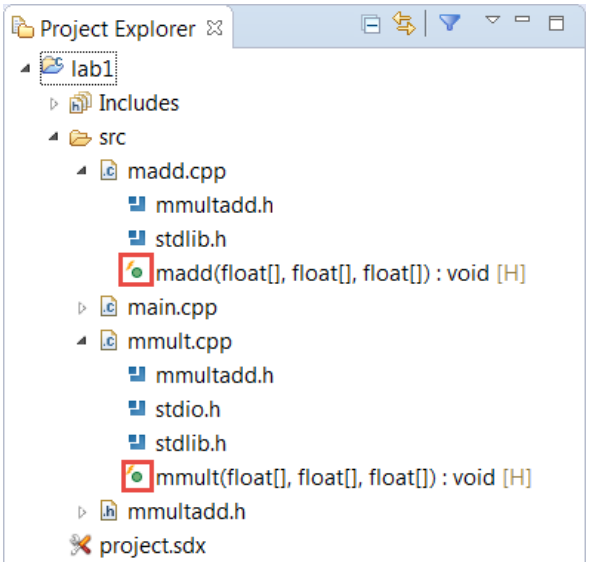


Figure 10. Expanded project view indicating the selected functions for hardware implementation

Build the Design with Hardware Accelerators Step 2

2-1. Select Release configuration and build the project. When done, analyze the data motion network through the report and built hardware through Vivado IPI.

2-1-1. Right-click on **lab1** in the **Project Explorer** and select *C/C++ Build Settings*. Select **Miscellaneous** under *SDS++ Linker*, click "+" button of the *Other Options* and enter **–maxjobs <host core count>/2** (substitute <host core count> with the actual number of cores of your machine) and click **OK**.

> The above step is optional but recommended otherwise Vivado invocation under the hood during the hardware generation will use all the cores bringing your machine unresponsive to do anything else.

2-1-2. Right-click on **lab1** in the **Project Explorer** and select **Build Configurations > Set Active** to see possible configurations and what is currently selected.

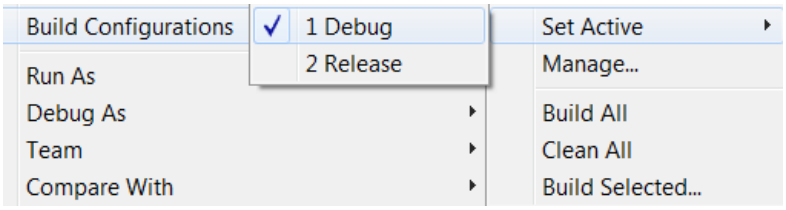


Figure 11. Selecting build configuration

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

You can also select the configuration by clicking on the drop-down button of the Active build configuration field of the SDx Project Settings pane.

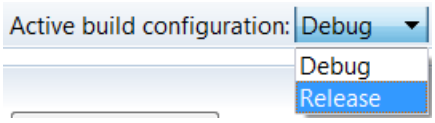


Figure 12. Available configurations and selecting them

The Release build configuration uses a higher compiler optimization setting than the Debug 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.

2-1-4. Right-click on **lab1** and select **Build Project**

The output from the SDx 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 50 to 60 minutes.

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

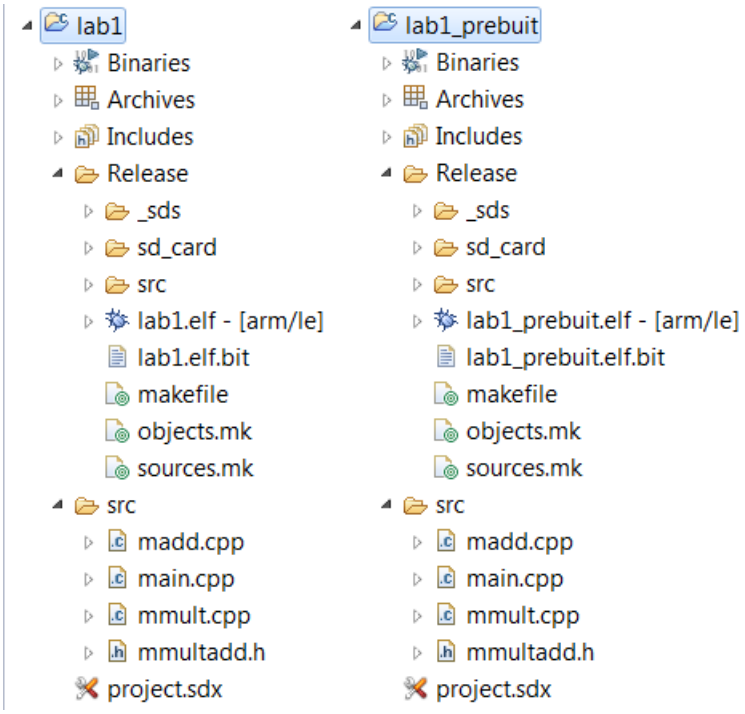
Select **File > Import** and then select **General > Existing Projects into Workspace** and click **Next**.

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

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

Click **Finish**.

2-1-5. Expand the **lab1** (or **lab1_prebuilt** if you have imported the project) directory in the *Project Explorer* tab and observe that *Release* folder is created along with virtual folders of *Binaries* and *Archives*. Expanding the *Release* 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.



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

Figure 13. 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 SDx as well as other underlying used tools.

2-1-6. Expand the _sds folder.

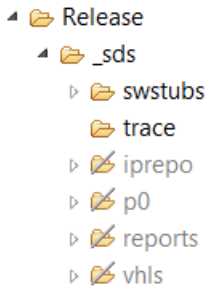


Figure 14. The generated _sds folder content

Notice that there are six sub-folders out of which four are greyed-out and two are shown as normal. The directories which are not greyed-out (*swstubs* and *trace*), indicate the folders and their contents were generated by SDx. 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.

2-1-7. Another pane, **Reports**, is created below the *Project Explorer* pane. Select the **Lab1** entry in the *Project Explorer* pane and see the available html and report files.

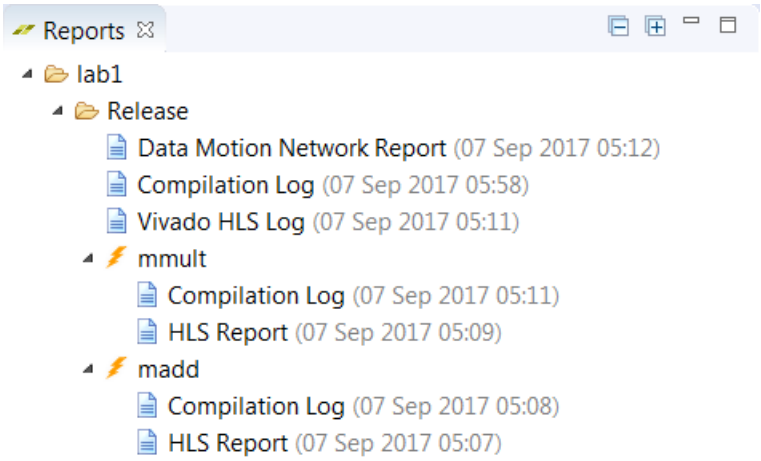


Figure 15. The Reports pane

2-1-8. Under the *Reports* pane, double-click on the **Data Motion Network Report** entry.

The report shows the connections made by the SDx 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 **Release > _sds > reports** of Project Explorer.

Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
madd_1	A	A	IN	1024*4		mmult_1:C
	B	B	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	C	C	OUT	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
mmult_1	A	A	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	B	B	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	C	C	OUT	1024*4		madd_1:A

Accelerator Callsites

Accelerator	Callsite	IP Port	Transfer Size (bytes)	Paged or Contiguous	Datamover Setup Time (CPU cycles)	Transfer Time(CPU cycles)
madd_1	main.cpp:128:11	A	4096	paged		
		B	4096	contiguous	1112	5583
		C	4096	contiguous	1112	5583
mmult_1	main.cpp:127:11	A	4096	contiguous	1112	5583
		B	4096	contiguous	1112	5583
		C	4096	paged		

Figure 16. Data motion network and accelerator callsites

There are two accelerated functions- madd and mmult. They are given instance names of madd_1 and mmult_1. Each function has three arguments and hence three ports. Notice that the C port of mmult_1 is directly connected to A port of madd_1 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.

2-1-9. Start Vivado by selecting **Start > All Programs > Xilinx Design Tools > SDx 2017.2 > Vivado Design Suite > Vivado 2017.2**

2-1-10. Click the **Open Project** link, open the design by browsing to *C:/xup/SDSoC/labs/lab1/Release/_sds/p0/_vpl/ipi/ipiprj* and selecting **ipiprj.xpr**.

2-1-11. 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).

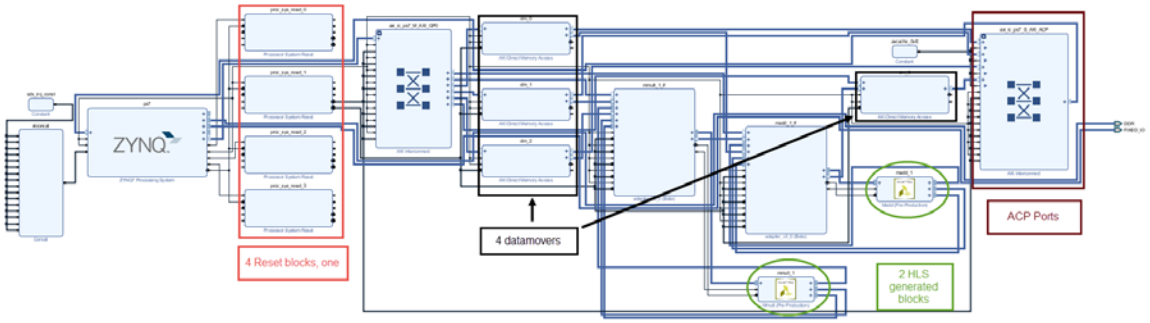


Figure 17. The generated block design

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 (second from top). The other blocks will be optimized away during the 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_1 output (Out C) has a direct connection path to madd_1 input (A), i.e. it does not go through any data movers. You can see the path by zooming in and tracing the connections (highlighted below).

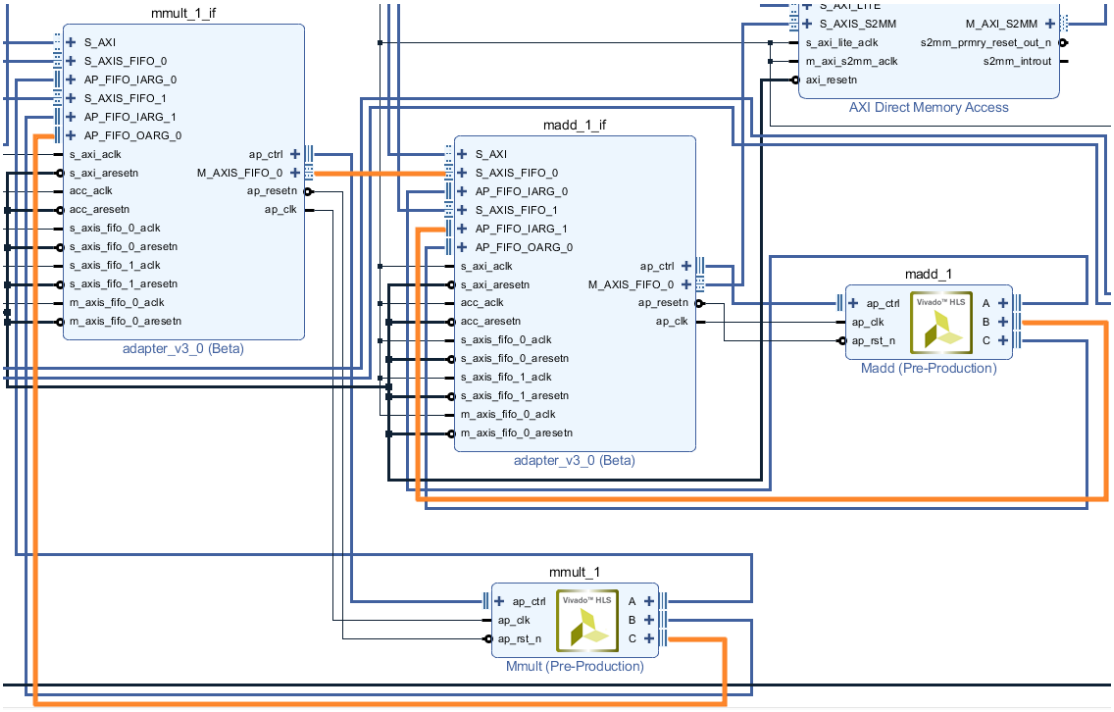


Figure 18. Non data mover connection between mmult_1 and madd_1

2-1-12. Close Vivado by selecting **File > Exit**

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

2-2-1. 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 167 calls the mmult_test function call. The mmult_test function is defined between lines 109 and 146, which in turn calls *mmult* at line 127 and *madd* at line 128. The *mmult* function is defined in *mmult.cpp* (lines 53-78) and the *madd* function is defined in *madd.cpp* (lines 43-52). Both these files are under the same *src* folder.

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

Note that line 169 calls mmult_test function call as seen in the original source file. The mmult_test function is now preceded by two function prototypes (lines 109 and 110) and the function is

defined between lines 111 and 148. On lines 129 and 130 it makes calls to _p0_mmult_1 and _p0_madd_1 replacing the original calls.

```
109 void _p0_madd_1_noasync(float A[1024], float B[1024], float C[1024]);
110 void _p0_mmult_1_noasync(float A[1024], float B[1024], float C[1024]);
111 int mmult_test(float *A,  float *B, float *C, float *D, float *D_sw)
112 {
113      std::cout << "Testing " << NUM_TESTS << " iterations of " << N << "x" << N
114                << " floating point mmultadd..." << std::endl;
115
116      perf_counter hw_ctr, sw_ctr;
117
118      for (int i = 0; i < NUM_TESTS; i++)
119      {
120           init_arrays(A, B, C, D, D_sw);
121
122           float tmp[N*N], tmp1[N*N];
123           sw_ctr.start();
124           mmult_golden(A, B, tmp);
125           madd_golden(tmp, C, D_sw);
126           sw_ctr.stop();
127
128           hw_ctr.start();
129           _p0_mmult_1_noasync(A, B, tmp1);
130           _p0_madd_1_noasync(tmp1, C, D);
131           hw_ctr.stop();
132
133           if (result_check(D, D_sw))
134                return 1;
135      }
```

Figure 19. Updated main.cpp file content

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

The _p0_mmult_1 function is defined in lines 82 through 95, replacing the original functionality with the data transfer using the cf_send command. Similarly, the madd function is updated in madd.cpp (lines 54-73). It additionally uses cf_receive call to get the results.

The subsequent labs will discuss these function calls.

```
80 #include "cf_stub.h"
81 void _p0_mmult_1_noasync(float A[1024], float B[1024], float C[1024]);
82 void _p0_mmult_1_noasync(float A[1024], float B[1024], float C[1024])
83 {
84   switch_to_next_partition(0);
85   int start_seq[1];
86   start_seq[0] = 0;
87   cf_request_handle_t _p0_swinst_mmult_1_cmd;
88   cf_send_i(&(_p0_swinst_mmult_1.cmd_mmult), start_seq, 1 * sizeof(int), &_p0_swinst_mmult_1_cmd);
89   cf_wait(_p0_swinst_mmult_1_cmd);
90
91   cf_send_i(&(_p0_swinst_mmult_1.A), A, 4096, &_p0_request_2);
92   cf_send_i(&(_p0_swinst_mmult_1.B), B, 4096, &_p0_request_3);
93
94
95 }
```

Figure 20. The updated mmult.cpp file

Test the Design in Hardware Step 3

3-1. Copy the files located under Release\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.

3-1-1. Using the Windows Explorer, copy all the files located under either *lab1\Release\SD_Card* folder or *lab1_prebuilt\Release\SD_Card* to the SD (Zed) or Micro-SD (Zybo) card.

3-1-2. Use **SDK Terminal** window on *Zed* or **Terminal** window on *Zybo* or third party terminal emulator program like PuTTy, HyperTerminal, TeraTerm programs.

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

3-1-4. You should see the output in the console.

```
Starting tcf-agent: OK

root@zybo:~#
```

Figure 21. System bootup output

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

3-1-5. 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.

```
root@plnx_arm:~# /mnt/lab1.elf
Testing 1024 iterations of 32x32 floating point mmultadd...
Average number of CPU cycles running mmultadd in software: 184001
Average number of CPU cycles running mmultadd in hardware: 19241
Speed up: 9.56296
TEST PASSED
root@plnx_arm:~#
```

(a) Zed

```
root@zybo:~# /mnt/lab1.elf
Testing 1024 iterations of 32x32 inting point mmultadd...
Average number of CPU cycles running mmultadd in software: 304231
Average number of CPU cycles running mmultadd in hardware: 19825
Speed up: 15.3458
TEST PASSED
root@zybo:~#
```

(a) Zybo

Figure 22. Program output

3-1-6. Close SDx by selecting **File > Exit**

3-1-7. Turn OFF the power to the board.

Conclusion

In this lab, you created a project in the SDx 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.