

SDSoC User Guide

Getting Started

UG1028 (v2014.4) February 28, 2015



Revision History

The following table shows the revision history for this document.

Date	Version	Changes
02/28/2015	2014.4	Initial Release.

Table of Contents

Revision History	2
Chapter 1 Introduction.....	6
Overview.....	6
Requirements	6
Chapter 2 Obtaining and Managing a License.....	7
Introduction.....	7
Chapter 3 Download and Installation.....	8
Introduction.....	8
Downloading the Software.....	8
Installation	8
Linux Environment Setup Script	12
Windows Environment Setup Script.....	12
Chapter 4 Getting Started Using an Example.....	14
Overview.....	14
Run a Pre-Built Application.....	14
Build the Matrix Multiplication Application	16
Linux	16
Windows.....	17
Running the Example on the ZC702 Board.....	17
Chapter 5 Self-Directed SDSoC Labs.....	19
Introduction.....	19
Lab 1 Creating, Building, and Running an SDSoC System	19
Introduction.....	19
Objectives	19
General Flow for this Lab.....	20
Step 1: Create an SDSoC Matrix Multiply Project.....	20
Step 2: Specify Hardware Functions.....	22
Step 3: Generate executable, bitstream and SD Card boot image.....	25

Step 4: Run Application on a zc702	29
Step 5: Additional Exercises (Optional)	32
Conclusion	32
Lab 1 Answers.....	33
Lab 2 Introduction to System Optimizations	34
Introduction.....	34
Background on System Ports and DMA.....	34
Objectives	35
Procedure	35
General Flow for this Lab.....	36
Step 1: Specifying System Ports.....	36
Step 2: Error Reporting	39
Step 3: Additional Exercises.....	40
Conclusion	44
Lab2 Answers.....	44
Lab 3 Introduction to System Debugging	45
Introduction.....	45
Objectives	45
Procedure	45
Step 1: Setup the board and cables and power on	45
Step 3: Setup the Debug Configuration with Prebuilt Lab	49
Step 4: Run the Application	51
Step 5: Additional Exercises (Optional)	52
Conclusion	54
Lab 3 Answers.....	55
Lab 4 Introduction to Performance Estimation	56
Introduction.....	56
Objectives	56
Procedure	56
Step 1: Setup the board and cables and power on	56
Step 2: Set up the project and use SDEstimate configuration	56
Step 3: Collect Software Run Data and Generate Performance Estimation Report	59
Step 4: Change the scope of overall speedup comparison	60

Step 5: Using the Performance Estimation Flow With Linux.....	61
Appendix A Supplementary Installation Notes.....	64
GNU Toolchain Requirements	64
Tips and Suggestions	64
Appendix B Additional Resources and Legal Notices	66
Xilinx Resources	66
Solution Centers	66
References	66
Please Read: Important Legal Notices	66

Overview

The SDSoC Environment is a C/C++ based design environment for implementing heterogeneous systems on the Zynq®-7000 All Programmable (AP) SoC family of devices.

SDSoC abstracts hardware through increasing layers of software abstraction that includes cross-compilation and linking of C/C++ functions into programmable logic fabric as well as the ARM CPUs within a Zynq device. Based on a user specification of program modules to run in programmable hardware, SDSoC performs program analysis, task scheduling and binding onto programmable logic and embedded CPUs, as well as hardware and software code generation that automatically orchestrates communication and cooperation among hardware and software components.

The SDSoC 2014.4 release includes support for the ZC702, ZC706, MicroZed, ZedBoard and Zybo development boards featuring Zynq-7000 All Programmable SoC.

Requirements

- Host running one of the following operating systems:
 - Linux – Red Hat Enterprise Workstation 5 or 6 (64-bit)
 - Windows – Windows 7 Professional (64-bit)
- Installation of the Xilinx SDSoC Environment, which includes:
 - SDSoC 2014.4, including an Eclipse/CDT-based GUI, high-level system compiler, and ARM GNU toolchain,
 - Vivado® Design Suite System Edition 2014.4, with Vivado and Vivado High Level Synthesis (HLS)
- The included ARM GNU toolchain requires host 32-bit compatibility libraries to run
- Linux Only - Availability of the following host operating system command: xsltproc
- A mini-USB cable to observe the UART output from the board

This document covers the following topics:

- Licensing
- Download and installation, including user environment setup
- Getting started using an example

For additional information on using the SDSoC design environment, see the tutorial labs at the end of this document, and the *Introduction to SDSoC User Guide* (UG1027).

Chapter 2 Obtaining and Managing a License

Introduction

The SDSoC 2014.4 release uses the Xilinx FLEXnet license configuration manager. Ask your Xilinx technical contact for information on how to obtain a license key for the SDSoC design environment.

Install your license key using the appropriate method for node-locked or floating license servers. Node-locked licenses are typically copied to <home>/.Xilinx (Linux) or C:\.Xilinx (Windows). For existing floating license installations, you typically add the new license file contents to the existing license file and restart the server. For new floating license installations, run the FLEXnet utility lmgrd, for example:

```
lmgrd -c <path_to_license>/Xilinx.lic -l <path_to_license>/log1.log
```

Client machines for node-locked licenses look for the license in one or more fixed locations. For the floating license, add the path to the license file or license server in the port@server format in the XILINXD_LICENSE_FILE environment variable.

Note: If you use Windows Explorer to create the folder C:\.Xilinx, navigate to C:\ and when you click on New Folder, enter the folder name with a trailing dot .Xilinx. (dot Xilinx dot). Press return and Windows creates the folder name .Xilinx (dot Xilinx); the trailing dot tells Windows to allow dot as the first character of the folder name.



TIP: The SDSoC 2014.4 release licenses are administered in the same manner as other Xilinx products. Your local Xilinx license administrator can help install the SDSoC license key file. Contact the Early Access program administrator or technical support if you need assistance.

Chapter 3 Download and Installation

Introduction

This chapter describes download and installation instructions for the SDSoC Environment.

Downloading the Software

Your Xilinx® technical contact provides access to the download lounge and information for downloading the SDSoC Environment.

Installation

After downloading the installer files, run `xsetup.exe` (on Windows) or `xsetup` (on Linux) and follow the on-screen instructions.

The screen shots below illustrate a typical installation session. After each page, click **Next** to advance to the next page, or **Finish** to complete the installation.

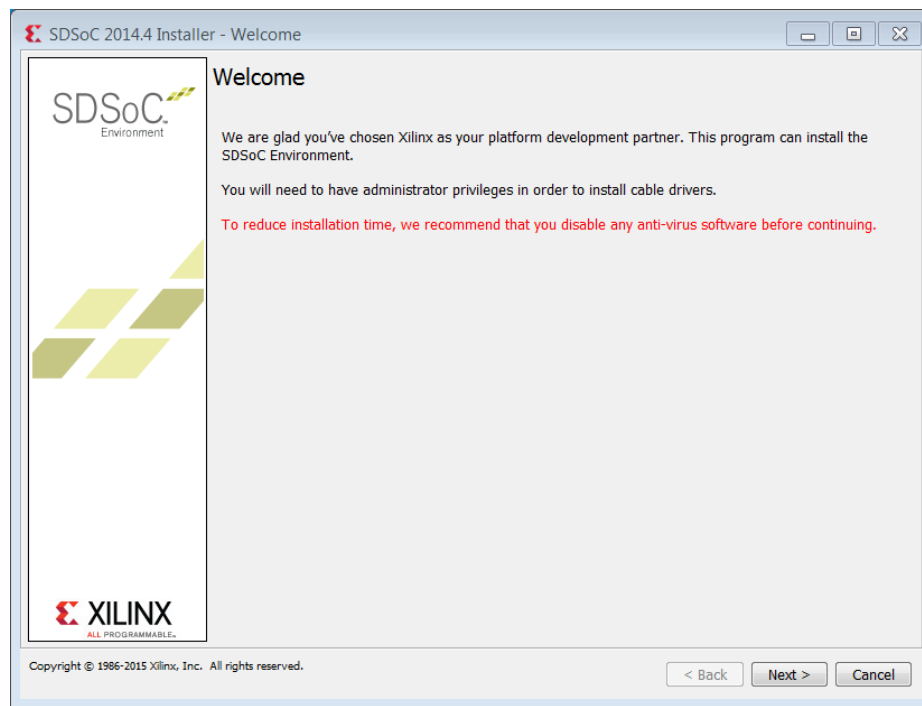


Figure 1: SDIntegrator Welcome Page

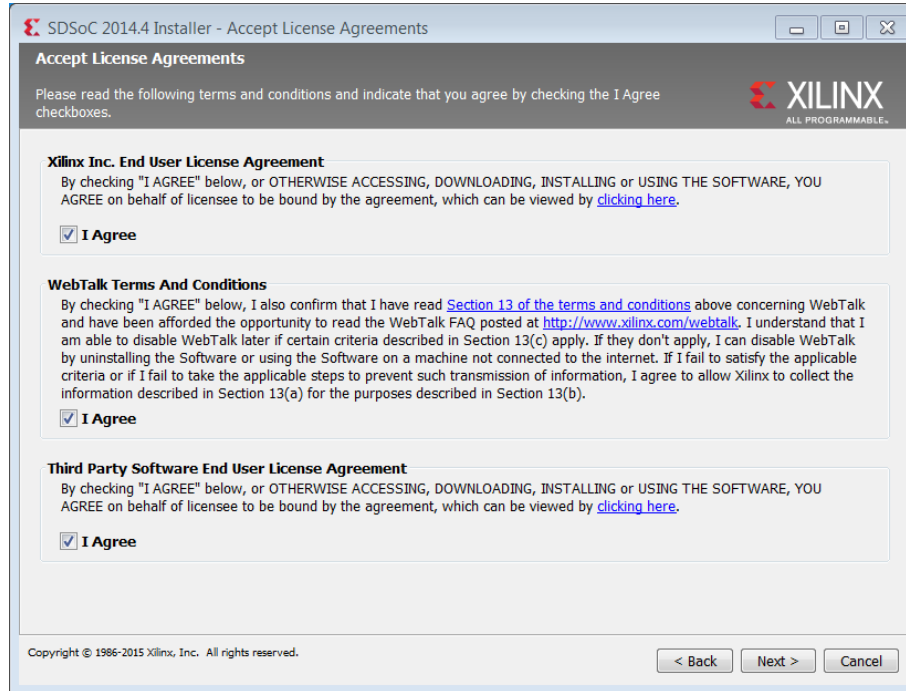


Figure 2: License Agreements

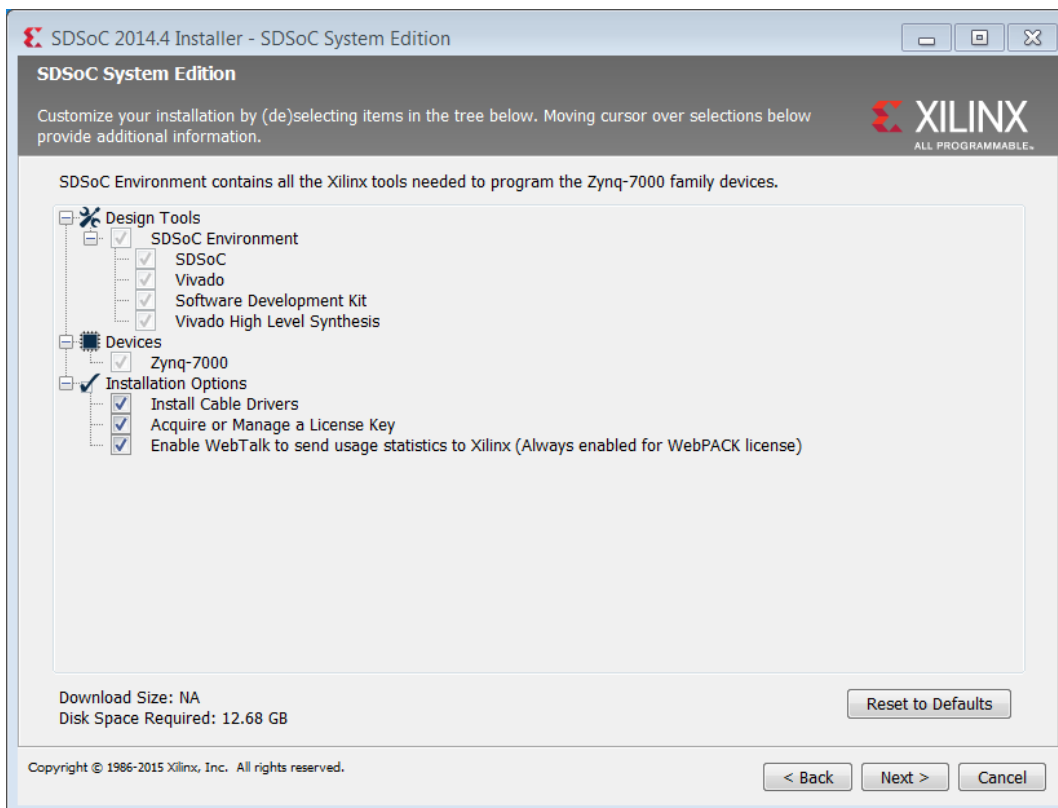


Figure 3: SDIntegrator Installer Options

If you have installed Vivado 2014.4 previously, you still need to install the SDSoC version of the Vivado Design Suite, but you do not need to reinstall the Cable Drivers.

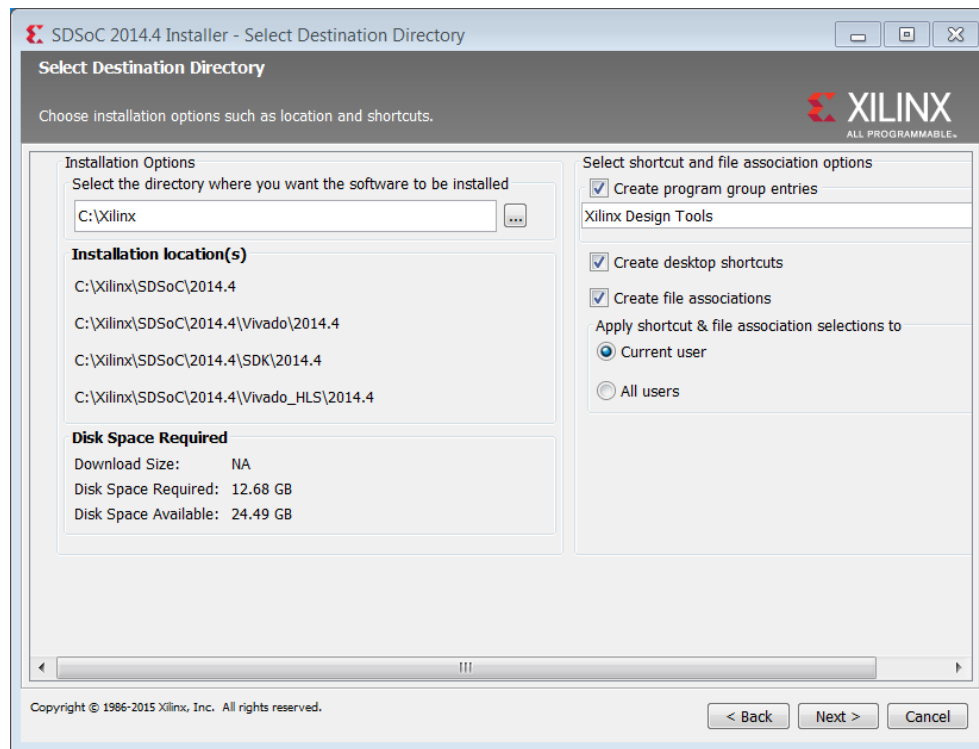


Figure 4: Select Destination Directory

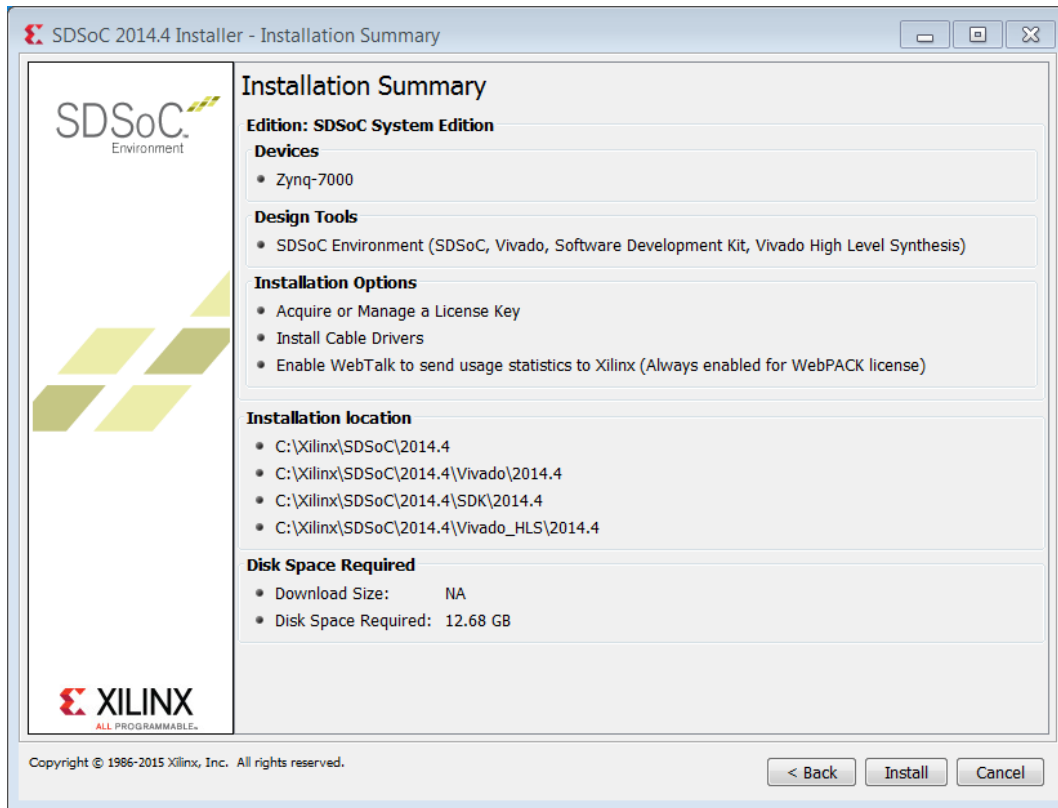


Figure 5: Installation Summary

After the installation completes, you have a directory with the following structure:

```
<path_to_install>/SDSoC/2014.4
  arm-xilinx-eabi
  arm-xilinx-linux-gnueabi
  bin
  data
  docs
  lib
  llvm-clang
  platforms
  samples
  scripts
  SDK
  tps
  Vivado
  Vivado_HLS
  settings64.[csh|sh|bat]
```

The installed software includes a copy of SDSoC 2014.4, Vivado 2014.4, Vivado HLS 2014.4, Xilinx SDK 2014.4 and scripts to set the environment to run in a Linux shell.

Linux Environment Setup Script

To run SDSoC, use the environment setup script (settings64.csh or settings64.sh) created by the installer. This script in turn runs setup scripts in the installation directory of each of the underlying tools to update the PATH environment.

To confirm that the environment is setup up properly, type the commands below and confirm that the commands find the installation locations consistent with the tool setup script:

```
% source settings64.csh
% which sdsccl          # SDSoC C/C++ build tool
% which vivado          # Vivado design tool
% which vivado_hls      # Vivado High-Level Synthesis (HLS) tools
% which bootgen         # Boot image creation tool (2014.4 version)
```

If the paths returned by the Linux “which” command are not consistent with the path to the installation directory, or the command was not found, confirm that the correct setup script was run.

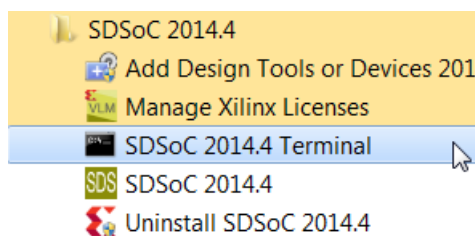


CAUTION! In each shell used to run SDSoC, use only the environment setup scripts corresponding to the Xilinx tool releases or PATH environment setting listed above. Running Xilinx design tool environment setup scripts from other or additional releases in the same shell may result in incorrect SDSoC tool behavior or results.

Windows Environment Setup Script

To run SDSoC, you will normally invoke the tool from the Windows Start Menu by clicking on **SDSoC 2014.4 -> SDSoC 2014.4** or by double-clicking a desktop shortcut created by the installer.

To confirm that the environment is setup up properly, invoke the **SDSoC 2014.4 Terminal** shortcut created by the installer. This starts a script (settings64.bat) to set environment variables for each of the underlying tools.



SDIntegrator 2014.2 Terminal

Type the following commands, and confirm that the installation location is consistent with the tool setup script. (Do not enter the comments beginning with REM):

```
> REM SDSoC C/C++ build tool
> where sdsccl
> REM Vivado design tool
> where vivado
> REM Vivado High-Level Synthesis (HLS) tools
> where vivado_hls
```

```
> REM Boot image creation tool (2014.4 version)
> where bootgen
```

If the paths returned by the “where” command are not consistent with the path to the installation directory, or the command was not found, confirm that settings64.bat file created by the installer contains the correct paths or the correct setup script was run.



IMPORTANT: When using Vivado tools on a Windows platform, path names longer than 260 characters may result in the error message: `ERROR: [Common 17-143] Path length exceeds 260-Byte maximum allowed by Windows: <LongPathToFileNam>`. Possible solutions to shorten path lengths or to avoid this are described in Answer Record [52787](#), for example use shorter path names, map a new drive letter to a lower directory in the path, and other methods.



CAUTION! If Cygwin is included in a global PATH environment variable and issues are encountered, it may need to be temporarily removed when running SDSoC tool flows. For example, in a command shell, type: `set PATH=%PATH:c:\cygwin\bin;=%`.



CAUTION! In each shell used to run SDSoC, use only the environment setup scripts corresponding to the Xilinx tool releases or PATH environment setting listed above. Running Xilinx design tool environment setup scripts from other or additional releases in the same shell will result in incorrect SDSoC tool behavior or results.



CAUTION! If you encounter the following make error on Windows “make interrupt/exception caught”, this can be caused by Git or other MinGW-based tools in your PATH. To eliminate this error, you may need to remove the tool from PATH or in some situations, temporarily uninstall the tool.

Chapter 4 Getting Started Using an Example

Overview

This chapter uses a simple example to validate the board setup, proper installation and setting of user environment for running SDSoC and dependent tools, while demonstrating a basic use flow for the SDSoC design environment.

First, test the board setup, connections and terminal setup using a prebuilt design that can be copied to an SD card. Insert the SD card into the ZC702 board, power on the board, run the ELF and observe the output of the matrix multiplication on a terminal connected to the board via a USB UART connection.

Next, you will use SDSoC to create a simple matrix multiplication application example, targeting the matrix multiplication function for conversion to a hardware accelerator block that resides on the programmable logic. This validates the tool installation and environment setup.

Finally, you will run the example on the board. SDSoC produces an SD card image containing a Linux bootloader, Linux kernel with the required drivers to communicate with the hardware accelerator, filesystem, and the application ELF. Use the SD card to run the ELF and observe the output.

Run a Pre-Built Application

The SDSoC installation directory contains several examples in the folder

`<path_to_install>/SDSoC/2014.4/samples.`

The `mmult_pipelined` folder contains C++ source files containing the main application that calls a matrix multiplication function and displays the output on stdout using `printf()` statements, a Makefile to build the application, and an `sd_card_prebuilt` folder containing prebuilt files. The files in the `sd_card_prebuilt` folder will be used to validate you board, board connections and terminal setup before using SDSoC.

```
<path_to_install>/SDSoC/2014.4/samples/mmult_pipelined
Makefile
mmult.cpp
mmult_accel.cpp
mmult_accel.h
sd_card_prebuilt
    BOOT.BIN
    README.txt
    boot.bif
    devicetree.dtb
    mmult.elf
    uImage
    uramdisk.image.gz
```

To run the pre-built application on the ZC702 board, follow these steps:

1. **Copy the contents of the `sd_card_prebuilt` folder to the root folder of an SD card. The SD card must be formatted using FAT32 (not NTFS).**
2. Insert the card into the SD card slot of the ZC702 board, and confirm jumpers or switches are set to boot from the SD card.
 - For Revision C and earlier boards:
 - J22 1-2 (connects pins 1 and 2)
 - J25 1-2 (connects pins 1 and 2)
 - For Revision D and newer boards:
 - DIP switch SW16 (light blue/grey color) positions 3 and 4 should be set to 1.
3. Connect an Ethernet cable from the board to your network. This is optional, it allows you to connect to the network.
4. Set up a serial terminal:
 - a. Connect a mini-USB cable from the board to the computer where you will run a serial terminal
 - b. Start a serial terminal (e.g. puTTY, minicom, et al) on your computer and set it to run at 115200 baud. If the baud rate is set incorrectly, terminal output may be garbled or not observed.
 - c. In Windows, the serial port will be set to COMn, where n is a number (typically 3, 4, 5, etc). See the Windows device manager to confirm the COM port number by following these steps:
 - i. Select Start > Computer, right click on Properties.
 - ii. Select Device Manager and open Ports (COM & LPT).
 - iii. Use the COM port labeled Silicon Labs CP210x USB to UART Bridge.
 - d. The board needs to be powered on at least once with the mini-USB cable connected for Windows to recognize the UART and install the driver. You may need to power cycle the board.
 - e. If the driver does not load, you can download and install it manually from the following location: <http://www.silabs.com/products/mcu/pages/usbtouartbridgevcpcdrivers.aspx>
5. With the SD card inserted and cables connected power up the board and start the serial terminal session. You should see the Done LED turn green and Linux boot.
6. At the prompt, type the following command to go to the SD card folder containing the application ELF file:

```
cd /mnt
```
7. To run the application ELF, type:

```
./mmult.elf
```
8. The application displays information about the run and the results of the matrix multiplication. You see output similar to that shown below:

```
Testing mmult ...  
Average number of processor cycles for golden version: 182299
```

```
Average number of processor cycles for hardware version: 18685
TEST PASSED
```

If you are able to run the prebuild design, continue on to the next section.

IMPORTANT: Do not proceed to the next section if you are not able to run the prebuilt example.

If you power up the board and the Done LED light does not turn green, this indicates the bootloader is not configuring the programmable logic. Confirm that the prebuilt SD card files were copied to the root (top) location of the SD card and not into a folder, and that the file sizes match. Confirm jumper settings. Use the SD card to boot another board to determine if the first one is not working properly. Confirm the SD card was formatted using FAT32 (not NTFS).

If you do not see Linux booting on the terminal, check the baud rate and COM port settings. Confirm the USB UART drivers are installed (uninstall and reinstall if unsure)



Build the Matrix Multiplication Application

You will now build the sample design `<path_to_installation>/SDSoC/2014.4/samples/mmult_pipelined`, and in doing so validate your tool installation and environment setup.

The `mmult_pipelined` folder contains C++ source files containing the main application that calls a matrix multiplication function and displays the output on stdout using `printf()` statements.

A user makefile invokes SDSoC to produce the hardware system including hardware accelerators, along with software libraries and API to utilize the hardware. The top-level makefile contains targets to build object files (.o) from application source files. Link them to create the application ELF and produce an SD card image.

Linux

To build the application on a Linux host, set up your environment to run the SDSoC design environment by running the setup script created by the installer for a C-Shell

```
% source <path_to_install>/SDSoC/2014.4/settings64.csh
```

or for a Bourne Shell / Bash

```
% . <path_to_install>/SDSoC/2014.4/settings64.sh
```

Copy the folder named `mmult_pipelined` to a working directory where you have write permission

```
% cp -r <path_to_install>/SDSoC/2014.4/samples/mmult_pipelined .
% cd mmult_pipelined
```

Build the application and the SD card image

```
% which sdscc # displays path to the sdscc tool
% make all
```

The build will take a while. After completion, folder named `sd_card` contains the ELF file and boot image required to start Linux on the ZC702 board and run the ELF application.

Windows

To run the same example on a Windows host, run the Launch SDSoc Terminal shortcut. This sets up your environment using commands described in the previous chapter. Type the commands below at the Windows command shell prompt '>' (comments beginning with REM do not need to be entered):

```
> cp -r <path_to_sdsoc_install>\SDSoC\2014.4\samples\mmult_pipeline .
> cd mmult_pipeline
> REM displays path to the sdscc tool
> where sdscc
> make
```

Note that the "cp" command is a Linux command, which is a part of your environment inherited from the Xilinx settings64.bat scripts used to set up your environment. A subset of Linux shell commands are available if you are comfortable using them, otherwise you may use Windows commands, for example:

```
> xcopy <path_to_install>\SDSoC\2014.4\samples\mmult_pipeline
<path_to_new_folder> /s /e /h
> cd <path_to_new_folder>
> make
```

If the make command completes successfully, continue to the next section and run the application on the board.

IMPORTANT: If you are unable to use the makefile to build the application, the likely cause is an incorrect path specified in the settings64.bat (Windows) or settings64.sh/settings64.csh (Linux) environment setup scripts.

If tools are not able to checkout a license, verify that the XILINXD_LICENSE_FILE in the setup script points to a valid license for running SDSoc as part of the Early Access program. Confirm that this variable has not been overridden in your environment by displaying the environment variable and examining the license.

On Windows, confirm that Cygwin is not included in the PATH environment variable. To temporarily remove Cygwin from PATH in a command shell, type: set
PATH=%PATH;c:\cygwin\bin;=

Running the Example on the ZC702 Board

After generating the application, run it on the ZC702 board, use the same steps described at the beginning of the chapter when validating board setup with a prebuilt SD card image. In the summary below, steps that were performed earlier have been omitted, for example terminal setup.

1. Copy the contents of the sd_card folder to an SD card.
2. Insert the card into the SD card slot of the ZC702 board, and confirm jumpers or switches are set to boot from the SD card. Connect the mini-USB cable from the board to the computer.

3. With the SD card inserted and cables connected, power up the board and start the serial terminal session set to 115200 baud. You should see Linux booting.
4. At the prompt, type the command below to go to the SD card folder containing the application ELF file.

```
cd /mnt
```

5. To run the application ELF, type

```
./mmult.elf
```

6. The application displays information about the run and the results of the matrix multiplication. You will see output similar to that shown below:
 - a. Testing mmult ...
 - b. Average number of processor cycles for golden version: 182299
 - c. Average number of processor cycles for hardware version: 18685
 - d. TEST PASSED

If the application runs properly on the board, try running and modify additional designs in the samples folder.

Introduction to SDSoc, UG1027 demonstrates techniques for increasing performance across a range of implementation examples.



IMPORTANT: If you are unable to run the application and have confirmed the board has been set up properly, contact Xilinx support.

Chapter 5 Self-Directed SDSoC Labs

Introduction

SDSoC is an Eclipse-based integrated development environment for developing Zynq®-7000 AP SoC processor applications.

To run SDSoC on Windows, invoke the tool from the Windows Start Menu by clicking **SDSoC 2014.4 -> SDSoC 2014.4** or by double-clicking the desktop shortcut created by the installer.

To run SDSoC on Linux, invoke the tool from the desktop shortcut created by the installer. You can also start the tool by running the **sdsoc** command.

The best way to become familiar with SDSoC is to use it. To get you started, the tools come with self-directed labs that walk you through common development tasks, with lab material available in the folder: `<path_to_installation>/SDSoC/2014.4/docs/labs`.

- Lab1 shows how to create a new SDSoC project using available templates, compile it and run the application on a board.
- Lab2 demonstrates how to modify code to optimize the hardware-software system generated by SDSoC.
- Lab3 shows how to retarget your design to a different platform/OS, and how to use the SDSoC GUI to download and run your hardware accelerated application on a board.

Lab 1 Creating, Building, and Running an SDSoC System

Introduction

This lab shows you how to use SDSoC to create a new SDSoC project using the available templates. You will then compile and run this SDSoC application on your zc702 board.

Objectives

After completing this lab, you will be able to:

- Use SDSoC to create a new SDSoC project for your application
- Choose from a number of available platforms and project templates for your application
- Mark a function for hardware implementation
- Build your project to generate a bitstream containing the hardware implemented function and an executable file that invokes this hardware implemented function

General Flow for this Lab

- **Step 1:** Create an SDSoC project for Matrix Multiplication and Addition
- **Step 2:** Select hardware functions to compile with Vivado HLS
- **Step 3:** Generate executable, bitstream and SD card boot image
- **Step 4:** Run application on a zc702

Step 1: Create an SDSoC Matrix Multiply Project

1. Select **File > New > SDSoC Project**. This brings up the new project creation wizard.

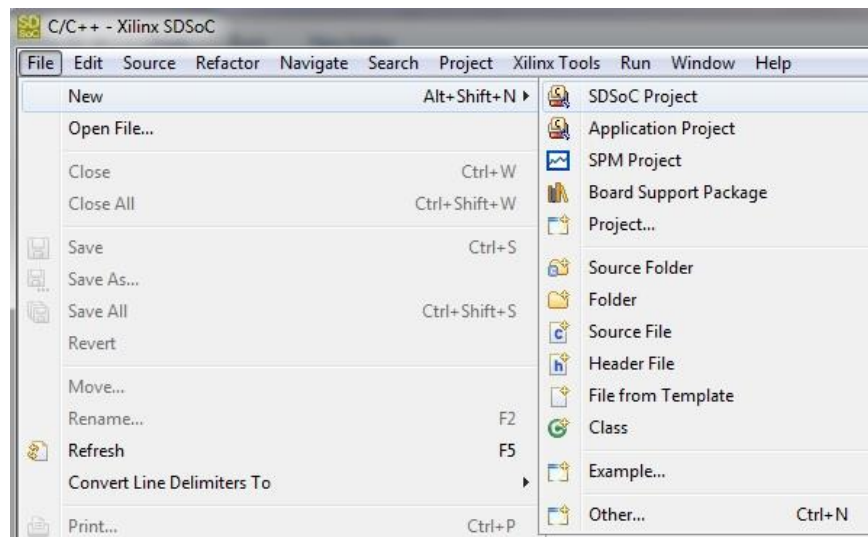


Figure 6. Creating a new SDSoC Project

2. Enter **lab1** as the **Project name** in the **New Project** dialog., and set **Platform** to **zc702** and **OS** to **Linux**. Click **Next** (not **Finish**).

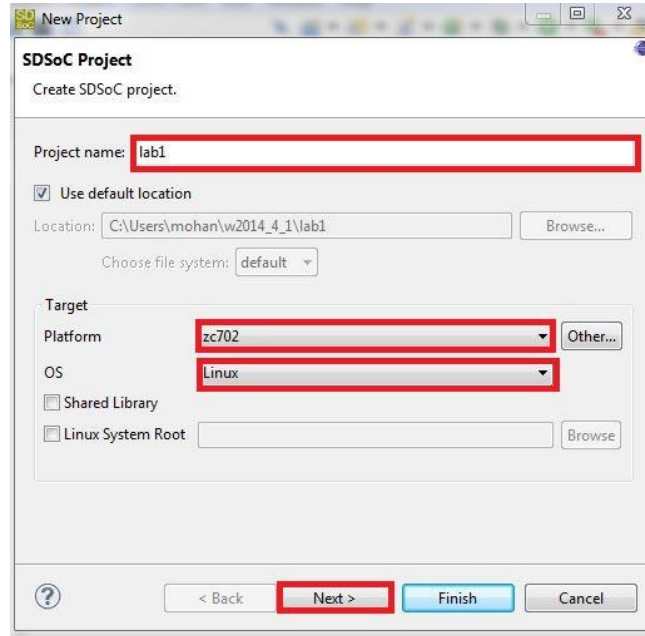


Figure 7. SDSoC New Project Wizard First Dialog Box

3. Select **Matrix Multiplication and Addition** from the list of application templates and click **Finish**.

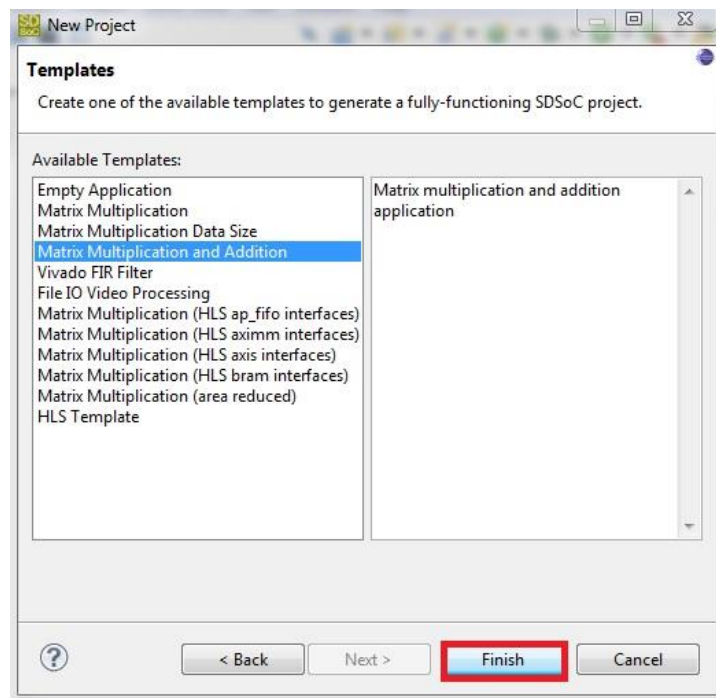


Figure 8. Selecting an Application Template

- To get the best performance, switch to use the **SDRelease** configuration by right clicking on **lab1** and selecting **Build Configurations > Set Active > SDRelease**.

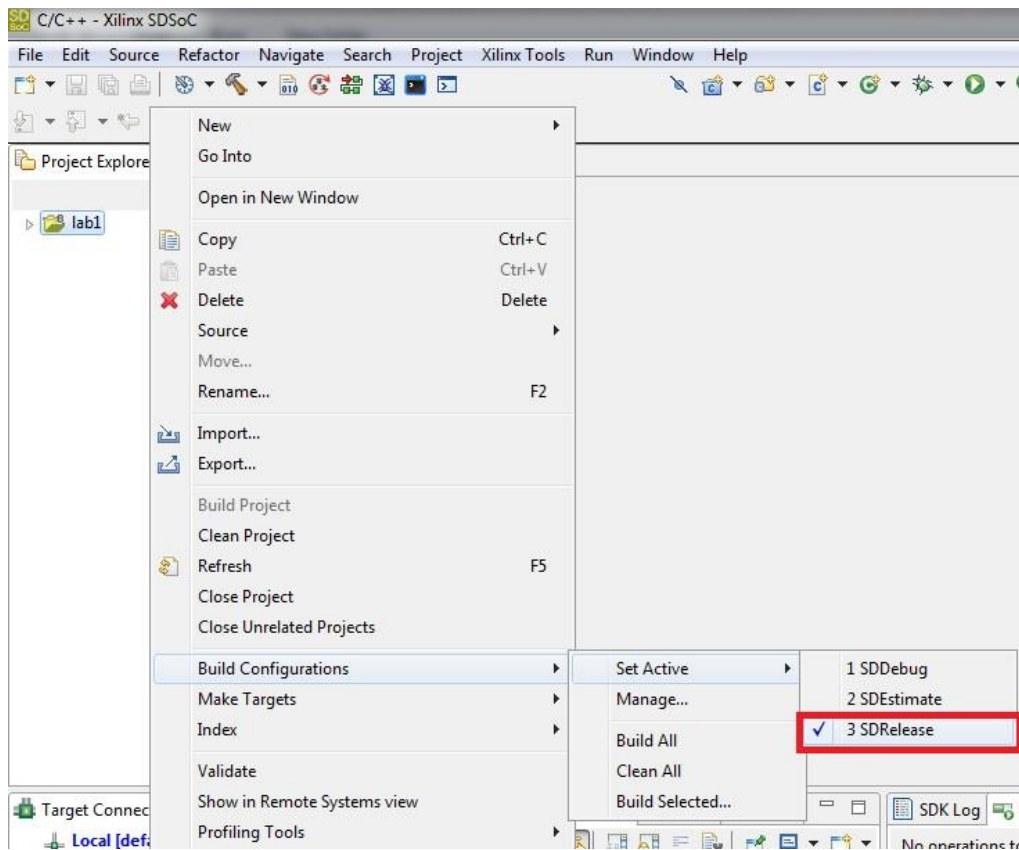




Figure 9. Set Active Build Configuration

Questions

- What top level folders do you see in the lab1 project? 
- Why do we switch to the SDRelease Configuration? 

Step 2: Specify Hardware Functions

- This application has two kernels – one kernel, `mmult`, multiplies two input matrices and the second kernel, `madd`, adds a third matrix to the resulting product. You will be specifying both kernels `mmult` and `madd` to be implemented in hardware using two different methods.

In the Project Explorer tab, double-click on `mmult.cpp` under the `src` folder to open the file in the editor. A nice feature of Eclipse is the Outline tab, which contains the function hierarchy within the file.

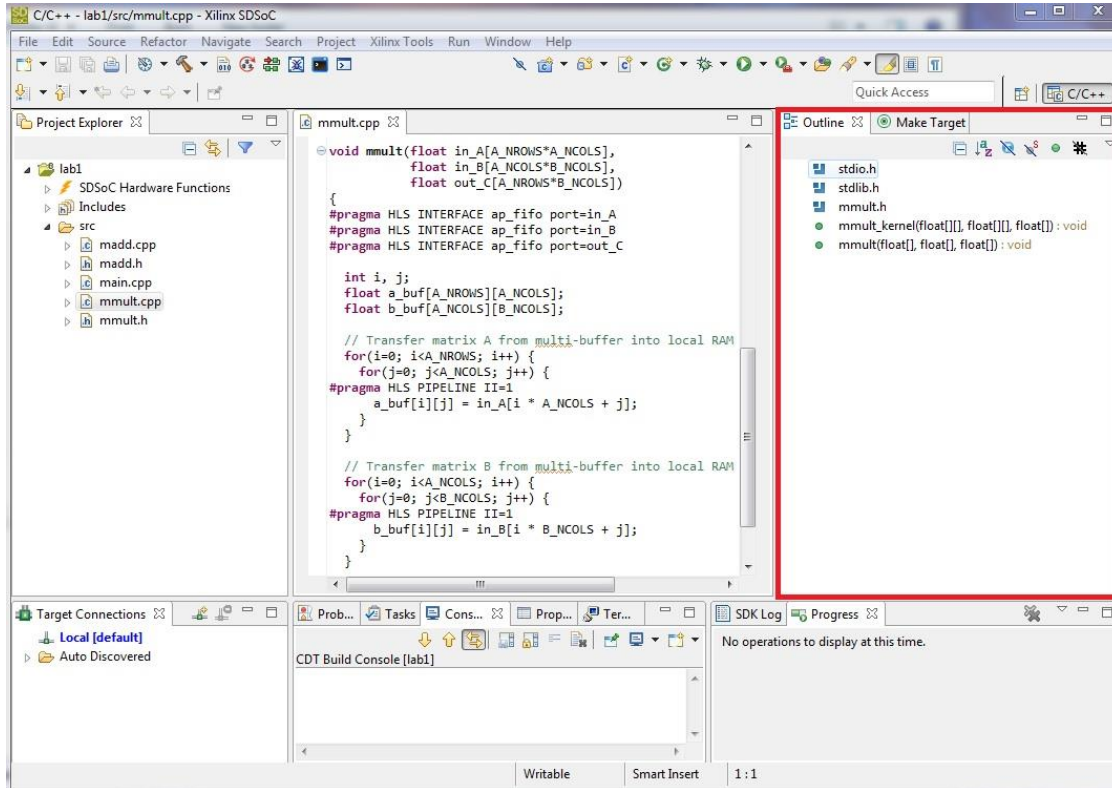


Figure 10. Outline panel

2. The first method to mark a function for hardware implementation is to right click on the `mmult` function signature in the Outline panel and select `Toggle HW/SW` in the drop-down menu. This is how you direct SDSoC to map a function into hardware. If you open `mmult.cpp`, you will see that this function contains pragmas to direct Vivado HLS to generate an efficient implementation of the matrix multiply algorithm.

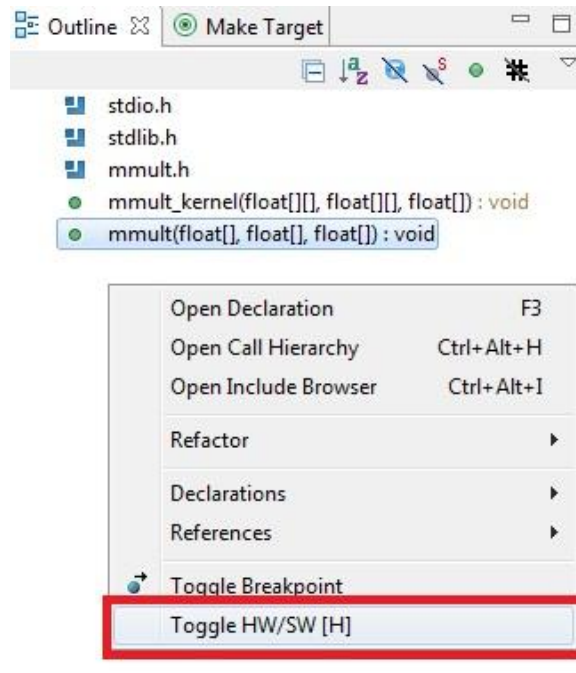


Figure 11. Marking a function for hardware implementation

- As visual cues that `mmult` will be mapped to hardware, SDSoC provides the [H] tag, an icon on `mmult` in the Outline panel, and highlights `mmult` in the editor.

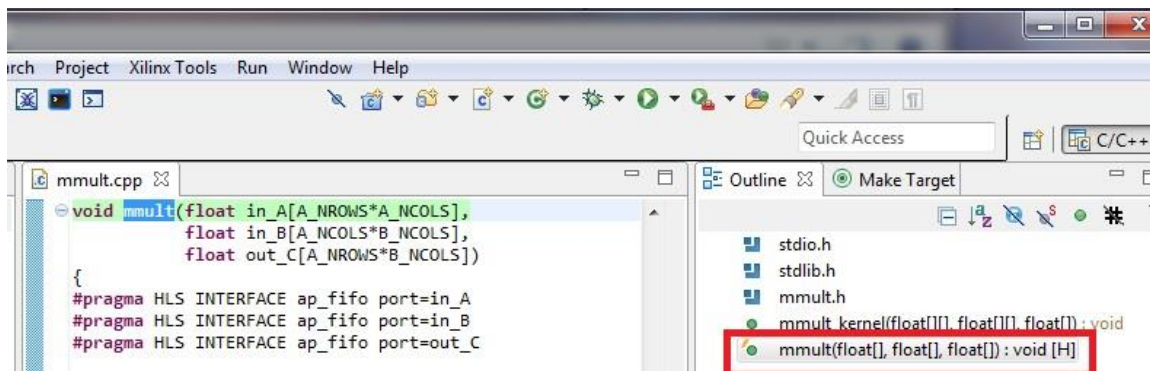


Figure 12. Source file and Outline showing a function marked for hardware implementation

- You can also mark a function for hardware implementation right from the Project Explorer without opening the `src` file. Right-click the `madd` function under the `madd.cpp` file in the Project Explorer and selecting Toggle HW/SW.

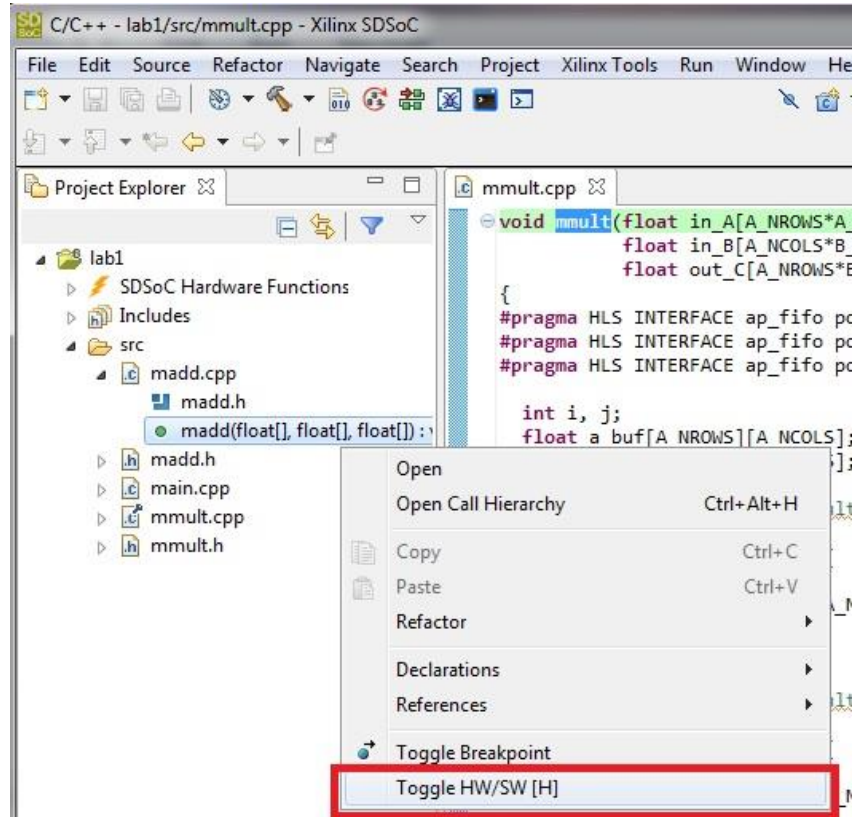


Figure 13. Alternate way to mark a function for hardware implementation



CAUTION! Although SDSoC will allow you to mark any function for implementation, not all functions can be implemented in hardware. See the Coding Guidelines section of the SDSoC user guide for more information. Also, the number of functions that can be implemented in hardware is device specific.

Questions

- Why is the number of functions that can be implemented in hardware device specific?

Step 3: Generate executable, bitstream and SD Card boot image

1. Right-click lab1 and click Build Project on the menu. SDK builds the project and shows you a popup with a progress bar. After the build starts, SDSoC stdout is directed to the Console tab. Vivado synthesis and implementation takes some time, so in the interest of time, Cancel the build.

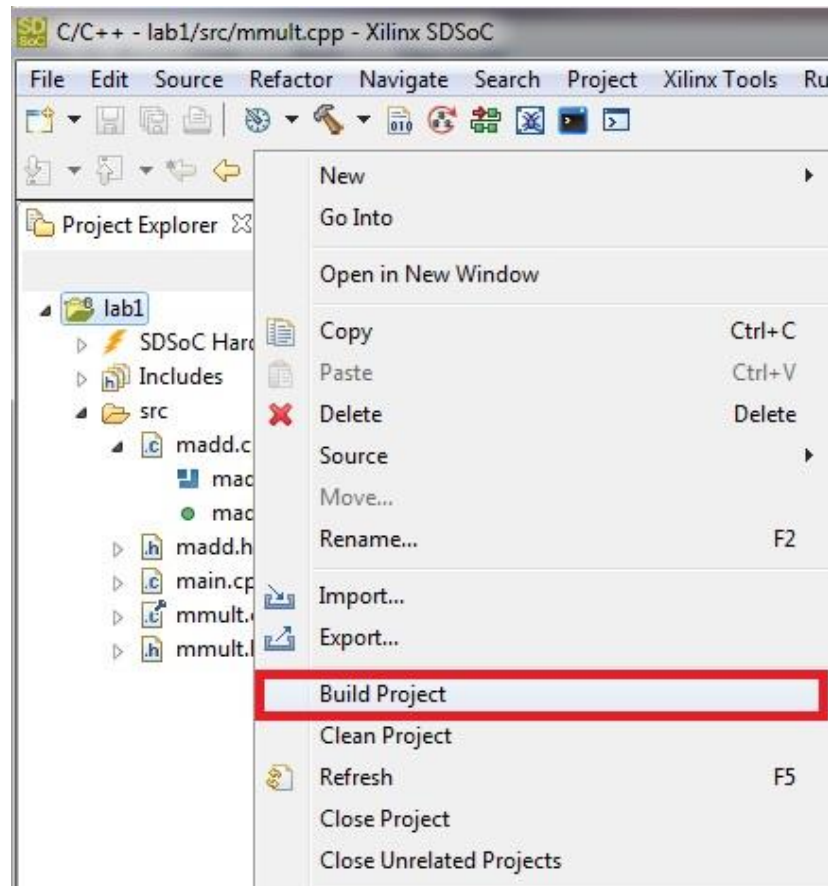


Figure 14. Building a project

2. Instead of waiting to build the entire design in hardware, you can import the prebuilt files into your workspace by selecting **File > Import...** and then select **General > Existing Projects into Workspace** and click **Next**.

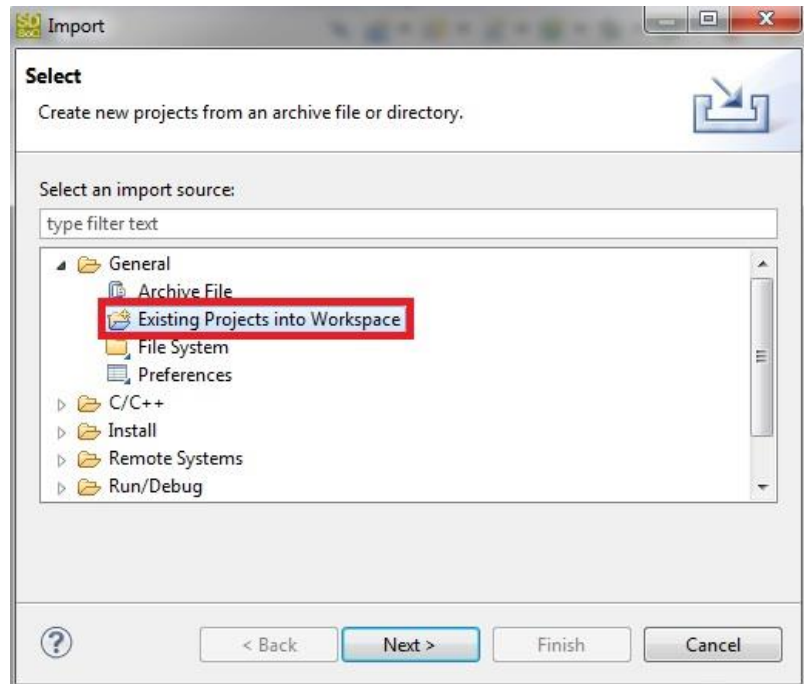


Figure 15. Import Dialog

3. Click on **Select archive file** and **Browse** to the `lab1_prebuilt.zip` file included in the lab files (same location as `lab1.pdf`; for example `<path_to_install>/SDSoC/2014.4/docs/labs`).
4. Make sure to put a checkmark on **lab1_prebuilt** and click **Finish**.

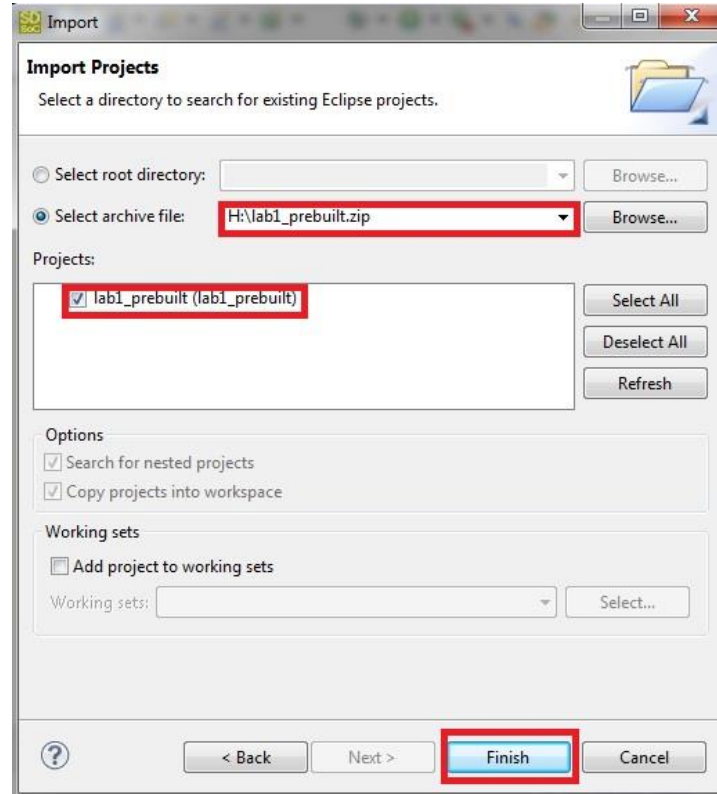


Figure 16. Import Projects Dialog

5. Look at the Data Motion Network report by double clicking on the **Project Explorer** window `lab1_prebuilt/SDRelease/_sds/reports/data_motion.html`. This report shows the connections done by SDSoC and the types of data transfers for each function implemented in hardware. We will go over this in more detail in Lab2.

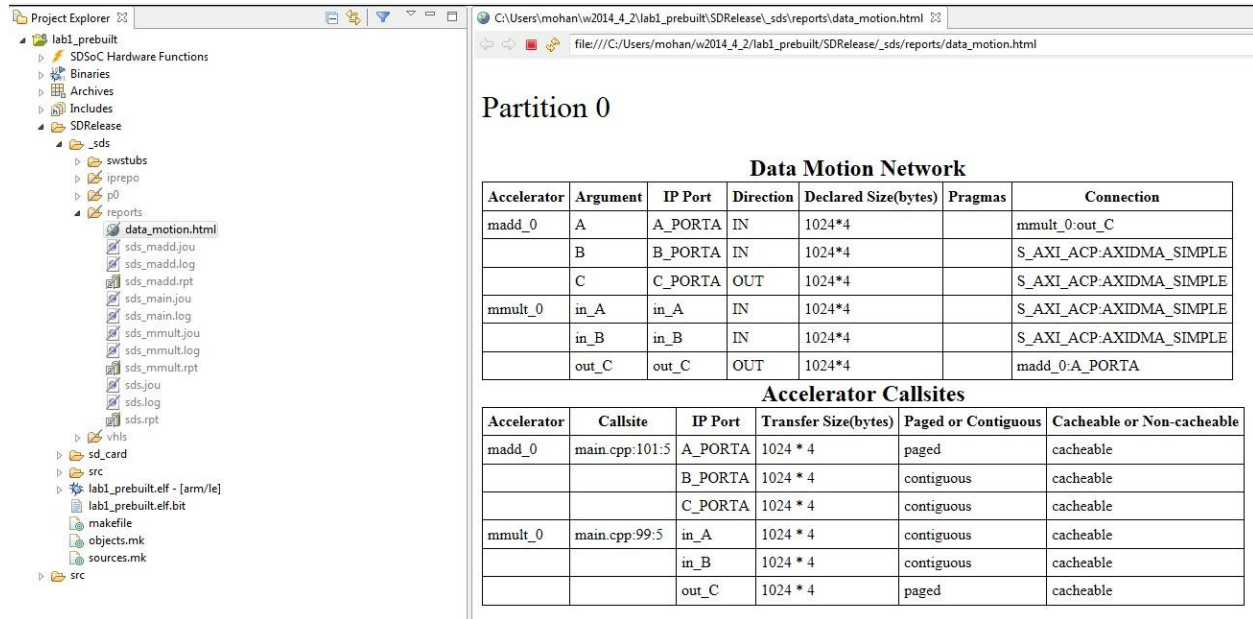


Figure 17. Data Motion Network Report

- Open the file `lab1_prebuilt/SDRelease/_sds/swstubs/mmult.cpp` to see how SDSoC replaced the original `mmult` function with one that performs transfers to and from the FPGA.

Step 4: Run Application on a zc702

- From your Windows Explorer, change to the `lab1_prebuilt/SDRelease` directory and copy all files inside the `sd_card` directory to the root of an SD card.
- Insert the SD card into the zc702 and power on the board.
- Connect to the board from a serial terminal in the SDSoC Terminal tab. Click the yellow-pad icon to open the settings. You can also maximize the Terminal window to make it easier to work with.

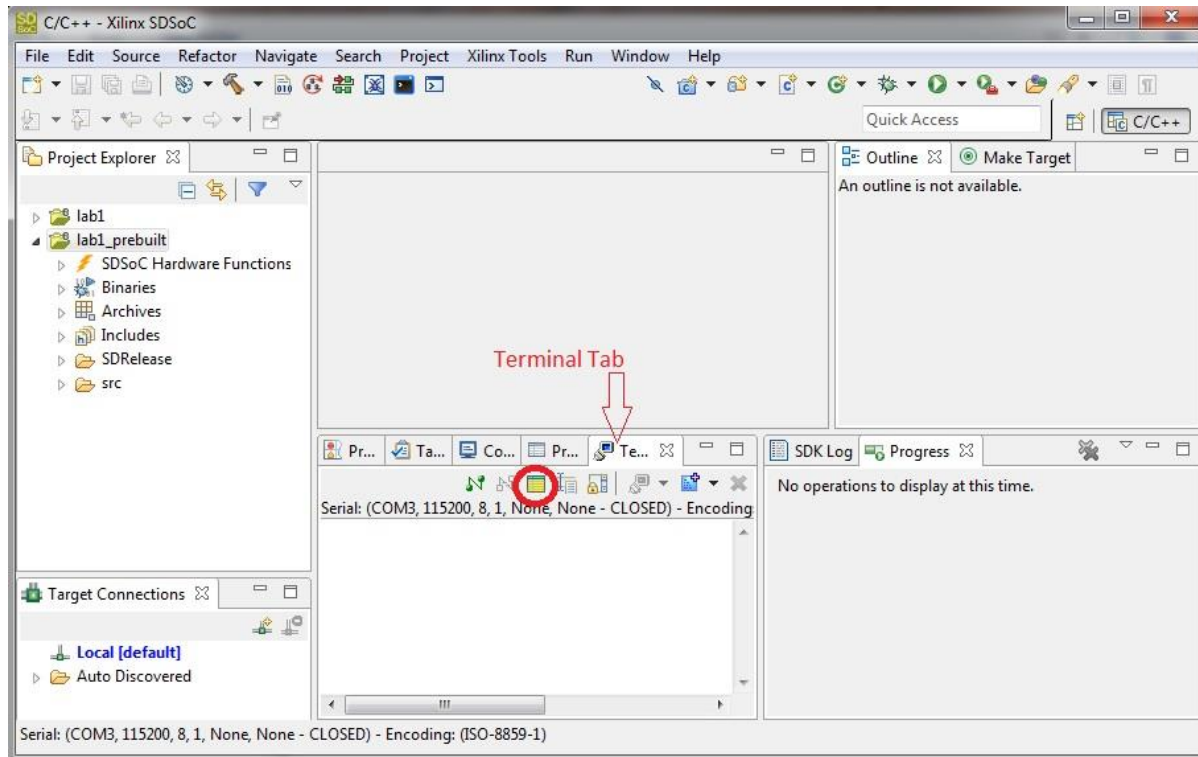


Figure 18. SDSoC Terminal

4. Set up the terminal as shown (connection type: serial, speed: 115200 baud). You may need to choose a different COM port. To verify which port you need, right click on Start → Computer then select Properties and open the Windows Device Manager. Now look under Ports to determine which COM port to use for the Silicon Labs USB to UART bridge.

Note: If the right COM port does not appear on the Terminal Settings window, make sure the board is connected to the USB port and turned on. Restart SDK by selecting File → Restart. After SDSoC restarts, the COM port should appear on the list.

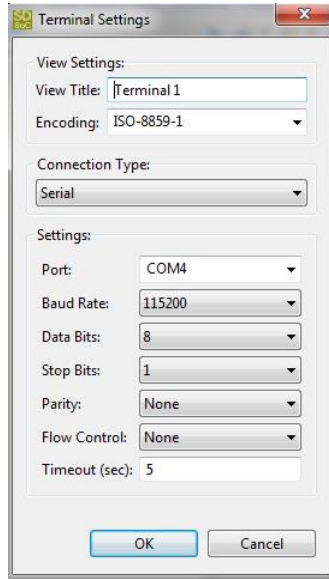


Figure 19. Terminal Settings

After the board boots up, you see a linux prompt where you can execute the application.

5. Type `/mnt/lab1_prebuilt.elf`.

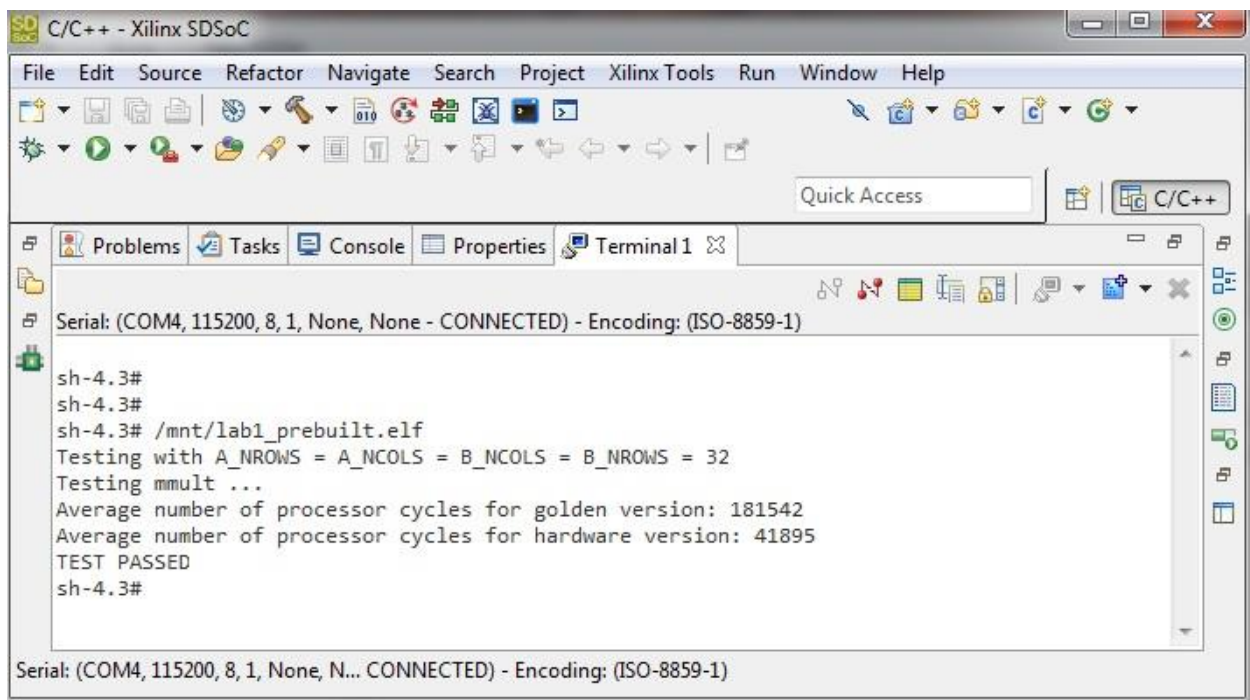


Figure 20. Maximized Terminal Window

Questions


- What is the speedup obtained by implementing functions in hardware? 

Step 5: Additional Exercises (Optional)

1. Examine the reports.

Examine the contents of the `SDRelease/_sds` folder. Notice the **reports** folder. This folder contains multiple log files and report (.rpt) files with detailed logs and reports from all the tools invoked by the build.

Question

- What tools does build invoke? 
2. For people familiar with Vivado IPI

Start Vivado and open the project located in the `SDRelease/_sds/p0/ipi` folder. This is the hardware design that SDSoC generated for your software. Open up the block diagram and look for the hardware matrix multiplier, the adder, the data mover (DMA) and the interconnect blocks in the block diagram.

Conclusion

You have learned how to use SDSoC to create a new SDSoC project, how to mark a function for hardware implementation, and how to build a hardware implemented design using SDSoC.

Lab 1 Answers

- What top level folders do you see in the lab1 project?

Includes and src

- Why do we switch to the SDRelease Configuration?

The SDRelease Configuration contains compiler options used for the best performance, such as `-O3`

- Why is the number of functions that can be implemented in hardware device specific?

This is because the amount of Programmable Logic varies from one device to another. Larger devices will fit multiple functions to be implemented in hardware while smaller devices do not.

- What is the speedup obtained by implementing functions in hardware?

The speedup is about 4.3 times faster. The application running on the processor takes 180k cycles while the application running on both the processor and FPGA takes 41k cycles.

- What tools does build invoke?

`sdscc`, `sds++`, `arm-xilinx-linux-gnueabi-gcc`, `arm-xilinx-linux-gnueabi-g++`, `vivado_hls`, `vivado`, `bootgen`

- `sdscc` is used to compile C language sources
- `sds++` is used to compile C++ language sources and also to link the object files created by `sdscc` and `sds++`
- `arm-xilinx-linux-gnueabi-gcc` is called by `sdscc` to generate object code for C language sources that are targeted to the processor
- `arm-xilinx-linux-gnueabi-g++` is called by `sds++` to generate object code for C++ language sources that are targeted to the processor, and also to link all the object files to create an executable that runs on the processor
- `vivado_hls` is called by `sdscc/sds++` to generate RTL code for C/C++ functions that are marked for hardware implementation
- `vivado` is called by `sds++` to generate the bitstream
- `bootgen` is called by `sds++` to create a bootable image containing the executable that runs on the processor along with the bitstream for the PL or FPGA logic portion of the chip.

Lab 2 Introduction to System Optimizations

Introduction

This lab shows you how to modify your code to optimize the hardware-software system generated by SDSoC. You will also see some of the error reporting capabilities of SDSoC.

Background on System Ports and DMA

In Zynq[®]-7000 All Programmable SoC device systems, the memory seen by the ARM A9 processors has two levels of on-chip cache followed by a large off-chip DDR memory. From the Programmable Logic side, SDSoC creates a hardware design that may contain a DMA (Direct Memory Access) block to allow a hardware function to directly read and/or write to the processor system memory via the system interface ports.

As shown in the simplified diagram in Figure 21, the Processing System block (PS) in Zynq devices has three kinds of system ports that are used to transfer data from processor memory to the Zynq device Programmable Logic (PL) and back. They are ACP (Accelerator Coherence Port) which allows the hardware to directly access the L2 Cache of the processor in a coherent fashion, HP0-3 (High Performance ports 0-3), which provide direct buffered access to the DDR memory or the on-chip memory from the hardware by-passing the processor cache via AFI (Asynchronous FIFO Interface), and GP0/GP1 (General-Purpose IO ports) which allow the processor to read/write hardware registers.

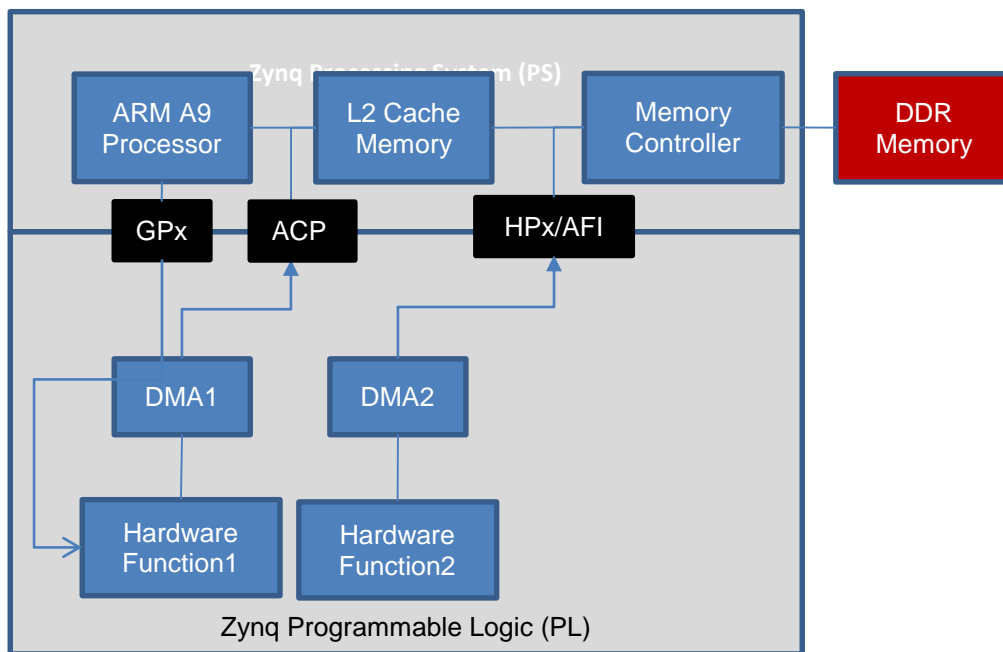


Figure 21. Simplified Zynq + DDR diagram showing memory access ports and memories

When the software running on the ARM A9 processor “calls” a hardware function, it actually calls an SDSoC generated stub function that calls underlying drivers to send data from the processor memories to the hardware function and to get data back from the hardware function to the processor memories over the three types of system ports shown – GPx, ACP, and AFI. Table 1 shows the different system ports and their properties. SDSoC automatically chooses the best possible system port to use for any data transfer, but allows users to override this selection by using pragmas.

System Port	Properties
ACP	Processor and Hardware function access the same fast cache memory as shared memory
AFI (HPx)	Driver must flush cache to DDR before Hardware function can read the data from DDR. Driver provides hand-shaking between processor and hardware function so that only one of them accesses the shared memory at any point.
GPx	Processor directly writes/reads data to/from hardware function. Inefficient for large data transfers

Table 1. System Port Properties

Objectives

After completing this lab, you will be able to:

- Use pragmas to select ACP or HP ports for data transfer
- Observe the error detection and reporting capabilities of SDSoC

If you go through the additional exercises you will also be able to

- Use pragmas to select different datamovers for your hardware function arguments
- Understand the use of `sds_alloc()`
- Use pragmas to control the number of data elements that are transferred to/from the hardware function.

Procedure

This lab is separated into steps, followed by general instructions and supplementary detailed steps allowing you to make choices based on your skill level as you progress through the lab.

If you need help completing a general instruction, go to the detailed steps below it, or if you are ready, simply skip the step-by-step directions and move on to the next general instruction.

This lab comprises two primary steps: You will start with the matrix multiply-add design from Lab1 and use SDSoC pragmas to control which PS7 system port to use for data transfer – ACP or AFI. Finally you will introduce some errors in your code and see how these errors are reported to you to allow you to fix them.

General Flow for this Lab

Step 1: Start SDSoC and create a new project based on the **Matrix Multiplication and Addition** template. Modify the source code to use a specific PS7 port (ACP or AFI)

Step 2: Introduce errors in the source code and see how they are reported to you.

Step 1: Specifying System Ports

The **sys_port** pragma in SDSoC allows you to specify whether the hardware uses the ACP or the AFI (HP0-3) ports to access the processor memory. In this step, we will look at some uses of this pragma.

1. Start SDSoC and create a new project for the zc702, Linux platform using the **Matrix Multiplication and Addition** design template
 - a. Start SDSoC and click **File->New->SDSoC Project**
 - In the new project dialog box type in a name (**lab2**) for the project, and select **zc702** and **Linux**, and click **Next**.
 - Select **Matrix Multiplication and Addition** as the application and click finish
 - Mark the **mmult** function for hardware acceleration as you did in Lab1
2. Set project linker options to prevent generating the bit stream and boot image and build.
 - a. Right-click lab2 folder, and select **C/C++ Build Settings**.
 - b. Select SDSoC under SDS++ Linux Linker in the settings panel and then check the **Do not generate the bitstream** and **Do not generate the SD card image** boxes, and click OK.

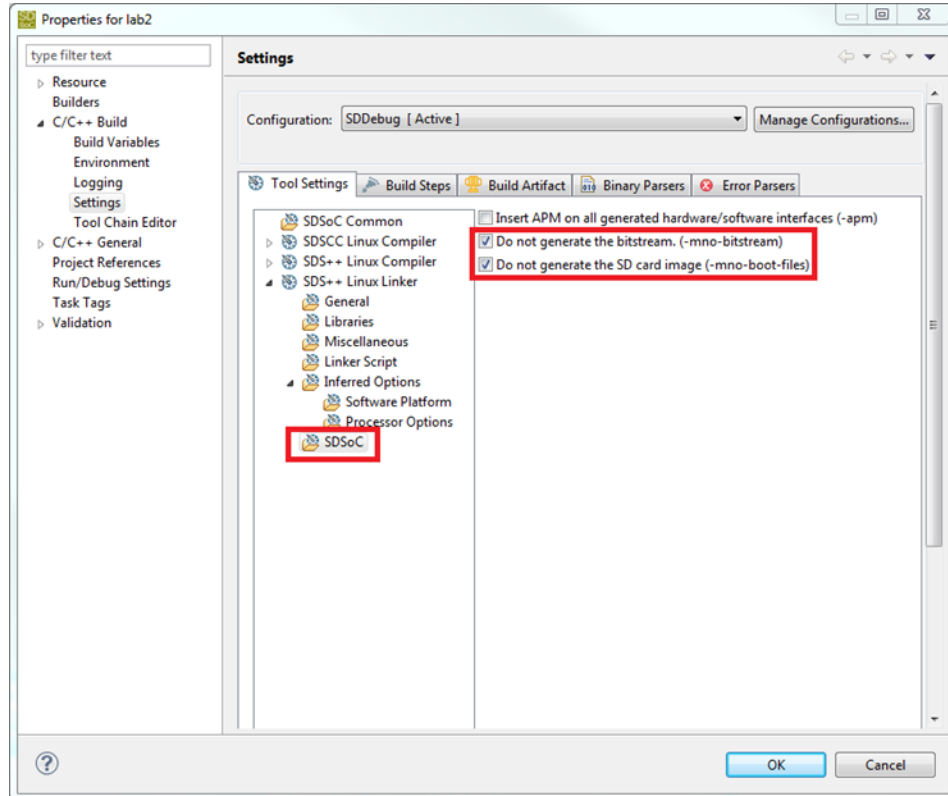


Figure 22: Properties for Lab2

- c. Right-click the top level folder for the project and click **Build Project** in the menu.



IMPORTANT: The build process can take 5-10 minutes to complete.

- d. When the build completes, in the Project Explorer panel, expand the `SDDebug` folder, then the `_sds` folder, then the **"reports"** folder, and double click on **data_motion.html** as shown in [Figure 22](#).

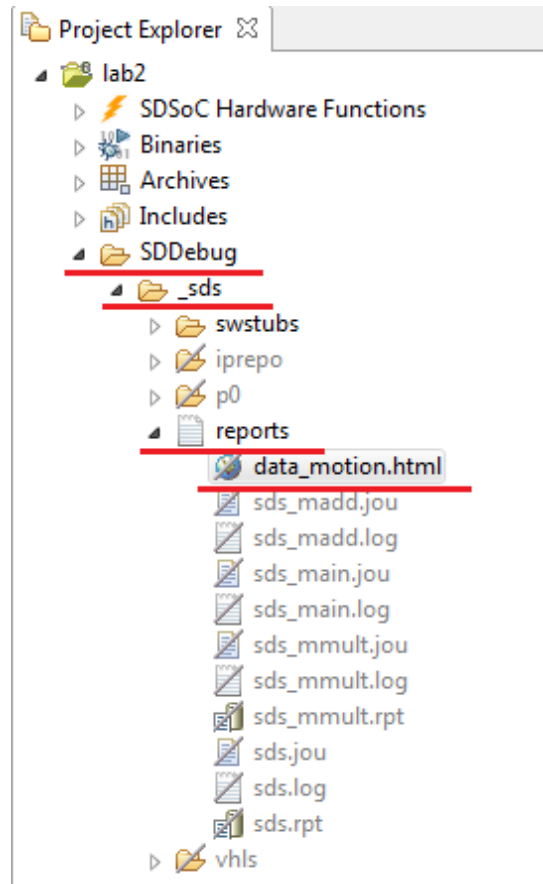


Figure 22. Location of data_motion.html

Look at the right-most column titled **connection** to see what kind of DMA is assigned to each input array of the matrix multiplier (`AXIDMA_SIMPLE` = simple DMA), and which PS7 port is used (`S_AXI_ACP`).

DATA MOTION NETWORK

Accelerator	Argument	IP Port	Direction	Declared Size (bytes)	Pragmas	Connection
Mmult_bd_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		S_AXI_ACP:AXIDMA_SG

Table 2 Partial view of data_motion.html before adding sys_port pragma

3. Add `sys_port` pragma.

- a. Double click `mmult.h` in the folder view to bring up the source editor panel
- b. Just above the `mmult` function declaration, insert the following line as shown to specify a different system port for each of the input arrays, and save the file

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

- c. Right-click the top-level folder for the project and click on **Clean Project** in the menu
- d. Right-click the top-level folder for the project and click on **Build Project** in the menu



IMPORTANT: *The build process can take 5 to 10 minutes to complete.*

- e. When the build completes, in the Project Explorer panel, expand `SDDebug` folder, then the `_sds` folder, then the `reports` folder, and double-click `data_motion.html`.
- f. Look at the right-most column titled **connection** to see what kind of system port is assigned to each input/output array of the matrix multiplier.

Questions

- What does the system port assignment imply in terms of whether the processor is pushing the data to the hardware or the hardware (DMA) is pulling the data from the processor's memory or pushing data into the processor's memory?

Step 2: Error Reporting

Once again you will continue from the previous step. You will introduce errors as described in each of the following steps and note the response from SDSoC.

1. Open the source file `main.cpp` and remove the semicolon at the end of the `printf()` statement near the bottom of the file. Notice how a yellow box shows up on the left edge of the line.
2. Move your cursor over the yellow box and notice that it tells you that you have a missing semicolon.
3. Insert the semicolon at the right place and notice how the yellow box disappears.
4. Now change `printf` to `print`, and see a pink box show up at the left edge.
5. Move the cursor over the pink box to see a popup displaying the "corrected" version of the line with `printf` instead of `print`.
6. Correct the previous error by changing `print` to `printf`, and introduce a new error by commenting out the line that declares all the variables used in `main()`.
7. Save and build the project (but do not wait for the build to complete, and go to the next step).
8. See the error messages scrolling by on the console. Open `Debug/_sds/reports/sds.log` and `Debug/_sds/reports/mmult.log` to see the detailed error reports.

Step 3: Additional Exercises

Background on Data Mover and DMA Types

When Linux is used as the target OS for your application, memory allocation for your application is handled by Linux and the supporting libraries. If you declare an array on stack within a scope (`int a[10000];`) or allocate it dynamically using the standard `malloc()` function, what you get is a section of memory that is contiguous in the Virtual Address Space provided by the processor and Linux. This buffer is typically split over multiple non-contiguous pages in the Physical Address Space, and Linux automatically does the Virtual-Physical address translation whenever the software accesses the array. However, the hardware functions and DMAs can only access the physical address space, and so the software drivers have to explicitly translate from the Virtual Address to the Physical Address for each array, and provide this physical address to the DMA or hardware function. As each array may be spread across multiple non-contiguous pages in Physical Address Space, the driver has to provide a list of physical page addresses to the DMA. DMA that can handle a list of pages for a single array is known as Scatter-Gather DMA. A DMA that can handle only single physical addresses is called Simple DMA. Simple DMA is cheaper than Scatter-Gather DMA in terms of the area and performance overheads, but it requires the use of a special allocator called `sds_alloc()` to obtain physically contiguous memory for each array.

Lab1 was written with `sds_alloc()` to allow the use of Simple DMA. In the following exercises you will first force the use of other Datamovers such as Scatter-Gather DMA or AXIFIFO using pragmas, and then modify the source code to use `malloc()` instead of `sds_alloc()` and notice how Scatter-Gather DMA is automatically selected.

1. Add data mover pragmas

For this exercise you will start with the source code from Lab1, and add datamover pragmas to specify what type of datamover will be used for each array. Then you will build the project and view the generated report (`data_motion.html`) to see the effect of these pragmas. Remember to prevent generation of bit stream and boot files as shown in section 1-2, so that your build does not synthesize the hardware.

- a. Double click **mmult.h** in the folder view under `lab1/src` to bring up the source editor panel
- b. Just above the `mmult` function declaration, insert the following line to specify a different data mover for each of the arrays and save the file

```
#pragma SDS data data_mover(in_A:AXIDMA_SG, in_B:AXIDMA_SIMPLE, out_C:AXIFIFO)
```
- c. Right-click the top-level folder for the project and click **Clean Project** in the menu.
- d. Right-click the top-level folder for the project and click **Build Project** in the menu.

IMPORTANT: The build process can take 5 to 10 minutes to complete.

- e. When the build completes, in the Project Explorer panel, expand `SDDebug` folder, then the `_sds` folder, then the `reports` folder, and double click `data_motion.html`.



- f. Look at the right-most column titled connection to see what kind of data mover is assigned to each input/output array of the matrix multiplier.

Note: The *Pragmas* column states the pragmas that were used. Also, the *AXIFIFO* data mover has been assigned the *M_AXI_GPO* port, while the other two data movers are associated with *S_AXI_ACP*.

2. Using `malloc()` instead of `sds_alloc()`

For this exercise you will start with the code from Lab1, and modify the source to use `malloc()` instead of `sds_alloc()`, and observe how the datamover changes from Simple DMA to Scatter-Gather DMA. Remember to prevent generation of bit stream and boot files as shown in section 1-2, so that your build does not synthesize the hardware. If you are continuing from the previous section then remember to remove the datamover pragma from the `mmult` declaration in file `lab1/src/mmult.h` to avoid overriding the compiler's choices.

- a. Double-click the `main.cpp` in the folder view to bring up the source editor panel
- b. Find all the lines to where buffers are allocated with `sds_alloc()`, and replace `sds_alloc` with `malloc` everywhere. Also remember to replace all calls to `sds_free()` with `free()`.
- c. Save your file.

```
int main(int argc, char* argv[]){
    int test_passed = 0;
    float *tin1Buf, *tin2Buf, *tin3Buf, *toutBufSw, *toutBufHw;

    printf("Testing with A_NROWS = A_NCOLS = B_NCOLS = B_NROWS = %d\n", A_NROWS);

    tin1Buf = (float *)malloc(A_NROWS * A_NCOLS * sizeof(float));
    tin2Buf = (float *)malloc(A_NCOLS * B_NCOLS * sizeof(float));
    tin3Buf = (float *)malloc(A_NCOLS * B_NCOLS * sizeof(float));
    toutBufHw = (float *)malloc(A_NROWS * B_NCOLS * sizeof(float));
    toutBufSw = (float *)malloc(A_NROWS * B_NCOLS * sizeof(float));
}
```

Figure 23. Allocation using `malloc/sds_alloc`

- d. Right-click the top-level folder for the project and click **Clean Project** in the menu.
- e. Right-click the top-level folder for the project and click **Build Project** in the menu.

IMPORTANT: The build process can take 5 to 10 minutes to complete.

- f. When the build completes, in the Project Explorer panel, expand `SDDebug` folder, then the `_sds` folder, then the `reports` folder, and double-click `data_motion.html`.



- g. Look at the right-most column titled connection to see what kind of DMA is assigned to each input/output array of the matrix multiplier (`AXIDMA_SG` = scatter gather DMA), and which PS7 port is used (`S_AXI_ACP`). You can also see on the Accelerator Call sites table whether the allocation of the memory that is used on each transfer is contiguous or paged.

3. Add pragmas to control amount of data transferred

For this step, you will use a different design template to show the use of the data copy pragma. In this template you will see an extra parameter called `dim1` being passed to the matrix multiply function. This parameter allows the matrix multiplier function to multiply two square matrices of any size `dim1*dim1` up to a maximum of 32*32. The top level allocation for the matrices creates matrices of the maximum size 32x32. The `dim1` parameter tells the matrix multiplier function the size of the matrices to multiply. And the data copy pragma tells SDSoC that it is sufficient to transfer a smaller amount of data corresponding to the actual matrix size instead of the maximum matrix size

- a. Start SDSoC and create a new project for the `zc702`, Linux platform using the matrix multiplication with variable data size design template.
 - i. Start SDSoC and click **File->New->SDSoC Project**.
 - ii. In the new project dialog box type in a name for the project (e.g. **lab2a**), and select **zc702** and **Linux**, and click **next**
 - iii. Select **Matrix Multiplication Data Size** as the application and click **Finish**.
 - iv. Mark the `src/mmult_accel/mmult_accel` function for hardware acceleration.
4. Setup the project to prevent building the bit-stream and boot files as in Section 1-2.
 - a. Add data copy pragma.
 - i. Double click on **mmult_accel.h** in the folder view to bring up the source editor panel
 - ii. Note the pragmas that specify a different data copy size for each of the arrays. In the pragmas, you can use any of the scalar arguments of the function to specify the data copy size. In this case, `dim1` is used to specify the size.

```
#pragma SDS data copy(in_A[0:dim1*dim1])
#pragma SDS data copy(in_B[0:dim1*dim1])
#pragma SDS data copy(out_C[0:dim1*dim1])
void mmult_accel (float in_A[A_NROWS*A_NCOLS],
                  float in_B[A_NCOLS*B_NCOLS],
                  float out_C[A_NROWS*B_NCOLS],
                  int dim1);
```

- b. Right-click the top-level folder for the project and click **Clean Project** in the menu.
- c. Right-click the top-level folder for the project and click **Build Project** in the menu.

IMPORTANT: The build process can take 5 to 10 minutes to complete.



- d. When the build completes, in the Project Explorer panel, expand `SDDebug` folder, then the `_sds` folder, then the `reports` folder, and double click `data_motion.html`.
- e. Look at the second column from the right, titled pragmas to see the length of the data transfer for each array. Also look at the second table showing the transfer size for each call site.

DATA MOTION NETWORK

Accelerator	Argument	IP Port	Direction	Declared Size (bytes)	Pragmas	Connection
mmult_accel_bd_0	in_A	in_A	IN	1024*4	Length: (dim1*dim1)	S_AXI_ACP:AXIDMA_SIMPL E
	in_B	in_B	IN	1024*4	Length: (dim1*dim1)	S_AXI_ACP:AXIDMA_SIMPL E
	out_C	out_C	OUT	1024*4	Length: (dim1*dim1)	S_AXI_ACP:AXIDMA_SG
	dim1	dim1	IN	4		M_AXI_GP0:AXILITE:0x80

Table 3 Data Motion Network Report

ACCELERATOR CALLSITES

Accelerator	Callsite	IP Port	Transfer Size (bytes)	Paged or Contiguous	Cacheable or Non-Cacheable
mmult_accel_bd_0	mmult.cpp:78:5	in_A	(dim1*dim1)*4	contiguous	cacheable
	in_B	in_B	(dim1*dim1)*4	contiguous	cacheable
	out_C	out_C	(dim1*dim1)*4	contiguous	cacheable
	dim1	dim1	4	paged	cacheable

Table 4 Accelerator Callsites

Questions

Why is the data copy size specified in terms of a variable?



IMPORTANT: *If the data transfer size is not specified correctly, the generated hardware-software application can “hang” or terminate with errors.*

Conclusion

This lab showed you how to optimize your hardware-software design implementation by using various SDSoC pragmas and special functions. You also learned how to find more info about build errors so that you can correct your code.

Lab2 Answers

- What does the system port assignment imply in terms of whether the processor is pushing the data to the hardware or the hardware (DMA) is pulling the data from the processor’s memory or pushing data into the processor’s memory?

If the hardware (DMA) is reading or writing to DDR, the system port is S_AXI_HP x , where x can be 0,1,2, or 3. If the hardware (DMA) is reading or writing to the L2 cache, the system port is S_AXI_ACP. All of these system ports are “Slave” ports meaning that the hardware controls them. If the processor is writing or reading from the hardware (from/to cache/DDR), the system port assigned is one of M_AXI_GP0 or M_AXI_GP1.

- Why is the data copy size specified in terms of a variable?

This allows each call to the hardware accelerated function to specify a different transfer size.

Lab 3 Introduction to System Debugging

Introduction

This lab shows you how to retarget your design to a different platform/OS, and how use SDSoC to download and run your hardware accelerated application on the board. You will start with the project Matrix Multiplication and Addition and target it to the standalone OS, and then build and run it on the zc702 board. As the build can take more than 30 minutes to complete, you will use a pre-built version of the project to learn how to download and run a standalone application.

Objectives

After completing this lab, you will be able to:

- Use SDSoC to download and run your standalone application
- Optionally step through your source code in SDSoC (debug mode) and observe various registers and memories. Note that this is limited to code running on the ARM A9, and does not apply to code that has been converted into hardware functions.

Procedure

This lab is separated into 4 steps, followed by general instructions and supplementary detailed steps allowing you to make choices based on your skill level as you progress through the lab.

General Flow for this Lab

- **Step 1:** Setup the zc702 board and cables, and power on
- **Step 2:** Create a project to target Standalone
- **Step 2:** Import the project for Lab3 (prebuilt Lab3) and setup the debug configuration
- **Step 4:** Run the application and observe the output on the terminal

Step 1: Setup the board and cables and power on

The first step for this lab is to set up the zc702 board for the lab. You will need a miniUSB cable to connect to the UART port on the board which will talk to a serial terminal on the SDK. You will also need a micro USB cable to connect to the Digilent port on the board to allow downloading the bitstream and binaries. Finally, you need to ensure that the jumpers to the side of the SD card slot are set correctly to allow booting from an SD card.

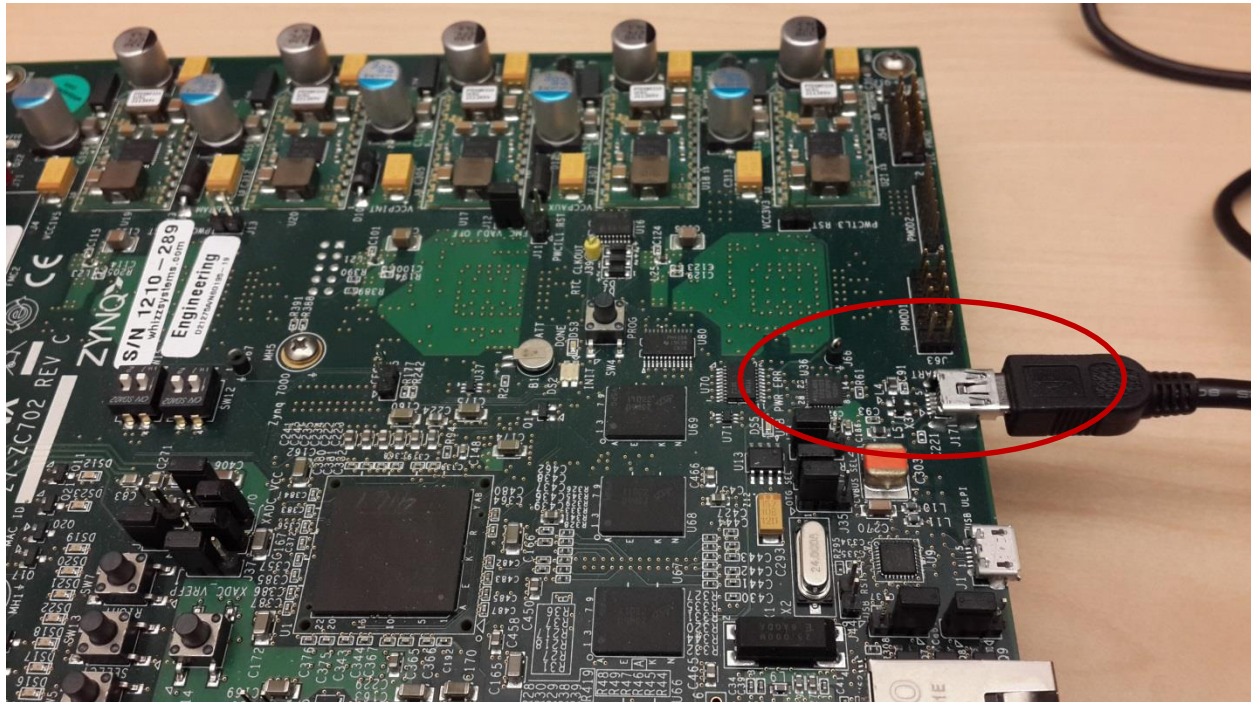


Figure 24. Location of the UART port

1. Make sure the mini USB cable is connected to the UART port as shown in [Figure 24](#).

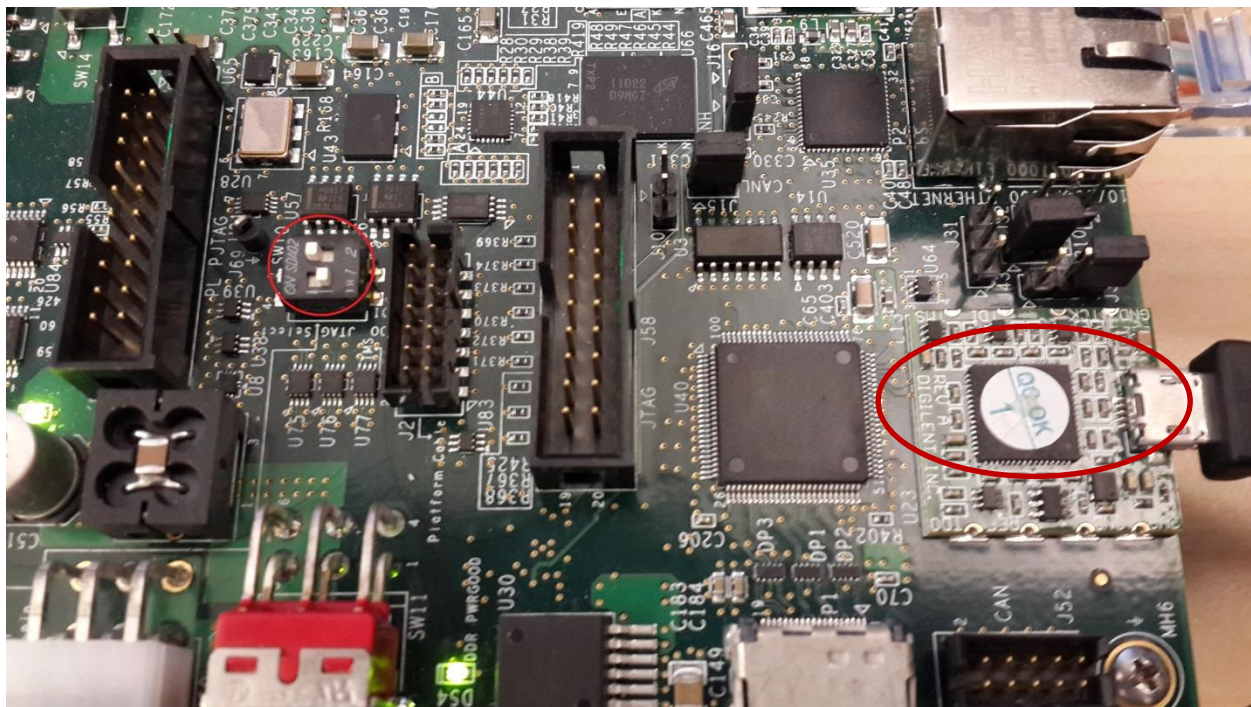


Figure 25. JTAG mode settings and MicroUSB connection

- Make sure the JTAG mode is set to use the Digilent cable, and the microUSB cable is connected as shown in [Figure 25](#).

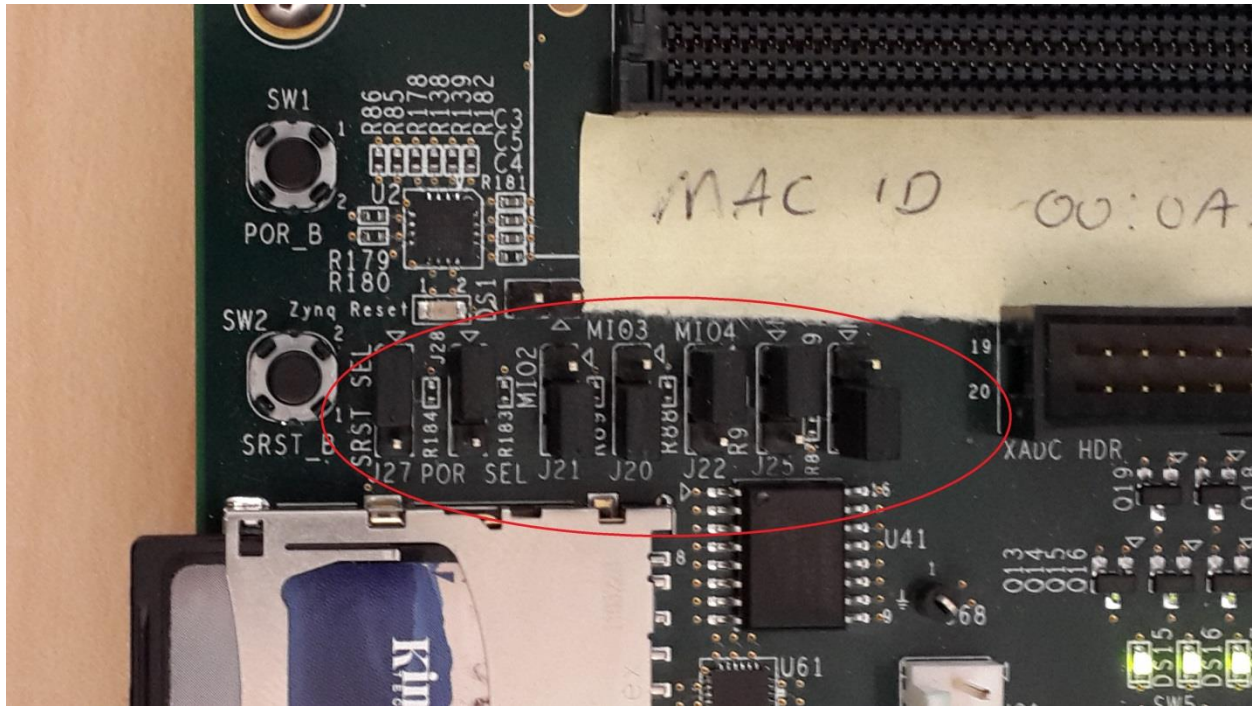


Figure 26. Jumper settings for SD boot

- Set the jumpers to SD-boot mode but do not plug in an SD card as shown in [Figure 26](#). Depending on the version of the zc702 board, you might see switches instead of jumpers and the SD boot settings are documented in ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide ([UG 850](#)), [Figure 1-2](#) and [Table 1-2](#).
- Power on the board before going to the next step.

Make sure you allow windows to install the USB-UART driver and the Digilent driver to enable SDK to communicate with the board.

Question

- What functionality does the miniUSB cable provide in this lab?

IMPORTANT: Make sure the jumper settings on the board correspond to SD-boot or JTAG-boot. Otherwise the board may power up in some other mode such as QSPI boot, and attempt to load something from the QSPI device or other boot device, which is not related to this lab



Step 2: Create a project targeted to Standalone

1. Create a new SDSoC project (lab3) for the ZC702 platform and Standalone using the design template for Matrix Multiplication and Addition.

You can also retarget this project to a different SDSoC platform if you need to, but do not change the platform for this exercise.

2. Right click on the **lab3** project folder and select the **C/C++ Build** Settings. Then select **C/C++ Build > Settings** to bring up the **Settings** panel.
3. You can now click **SDSoC Common**, then type in the name of the SDSoC-platform on the line labeled **SDSoC Platform**. Click **OK**.
4. Remember that this change applies only to the current build configuration **SDDebug** as shown near the top of the panel, unless you click the **Manage Configurations** button to select some other configuration or all configurations.

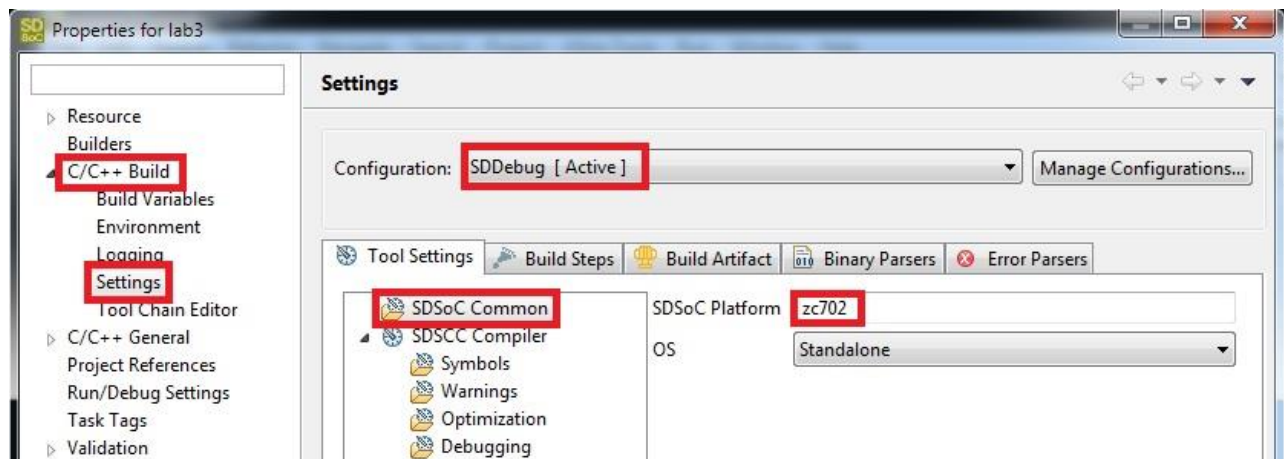


Figure 27: Retargeting to a different SDSoC-platform

Question

- Is it always possible to retarget designs from one platform or OS to another? When would this not work?



IMPORTANT: When you retarget a design, remember to clean the project and also delete the generated folders such as the Debug folder before building the project again.

5. Now you can build the project by right click on lab3 project and select Build Project with bitstream and boot files being generated for standalone mode. Because that **will take about 30-45 minutes**, you can **cancel** the build and use the prebuilt project as described in the next step.

Step 3: Setup the Debug Configuration with Prebuilt Lab

1. Import the lab3-prebuilt project.
 - a. Click **File > Import...** and select **General > Existing Projects into Workspace**. Click **Next**.
 - b. Click **Select archive file** and browse to find the **lab3_prebuilt.zip** file provided in the lab files folder. Click **Open**.
 - c. Click **Finish**.
 - d. Clean and rebuild the binary files without regenerating bitstream.

Since this project is imported, its binary elf file does not have the correct paths for source debugging. You would need to rebuild the binary but you do not want to rebuild the programmable logic bitstream and boot files.

- e. Right-click lab3_prebuilt folder, and select C/C++ Build Settings. Select SDSoC under SDS++ linker in the settings panel and then check the "Do not generate the bitstream" and "Do not generate the SD card image" boxes and click OK
 - f. Now, clean and rebuild project (**lab3_prebuilt > Clean Project, lab3-prebuilt > Build Project**). This may take a few minutes.
2. Setup the Debug Configuration.
 - a. Right-click the **lab3_prebuilt** folder and select **Debug As > Launch on Hardware (SDSoC Debugger)**

IMPORTANT: Make sure the board is turned on before doing this.

- b. Click the **Yes** button on the popup that asks you to confirm the Perspective Switch to the Debug perspective.

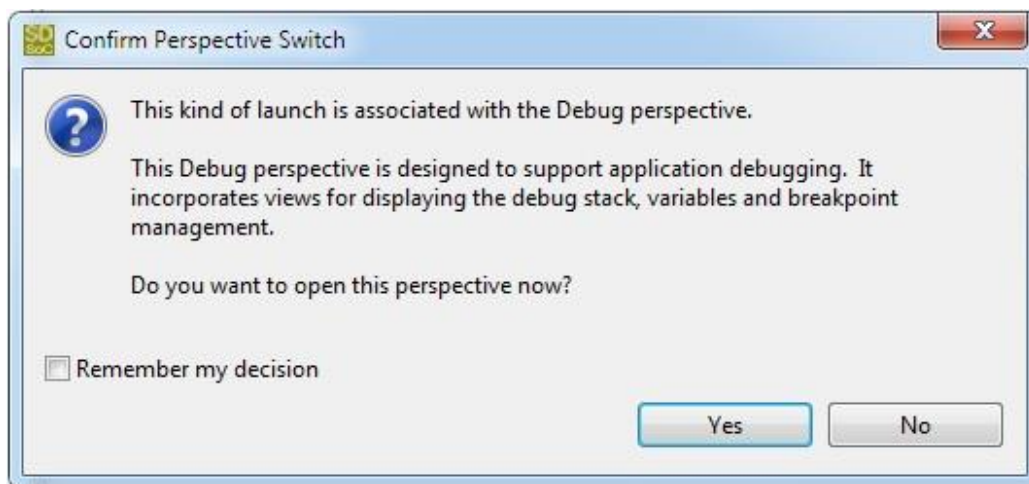


Figure 28. Switch to Debug Perspective

- c. You are now in the Debug Perspective of SDSoC. You would note that the debugger resets the system, programs and initializes the FPGA, then breaks at the main function. The source code is shown in the center panel, local variables in the top right corner panel and the SDK log is at the bottom right panel which shows a log of what the Debug configuration did.

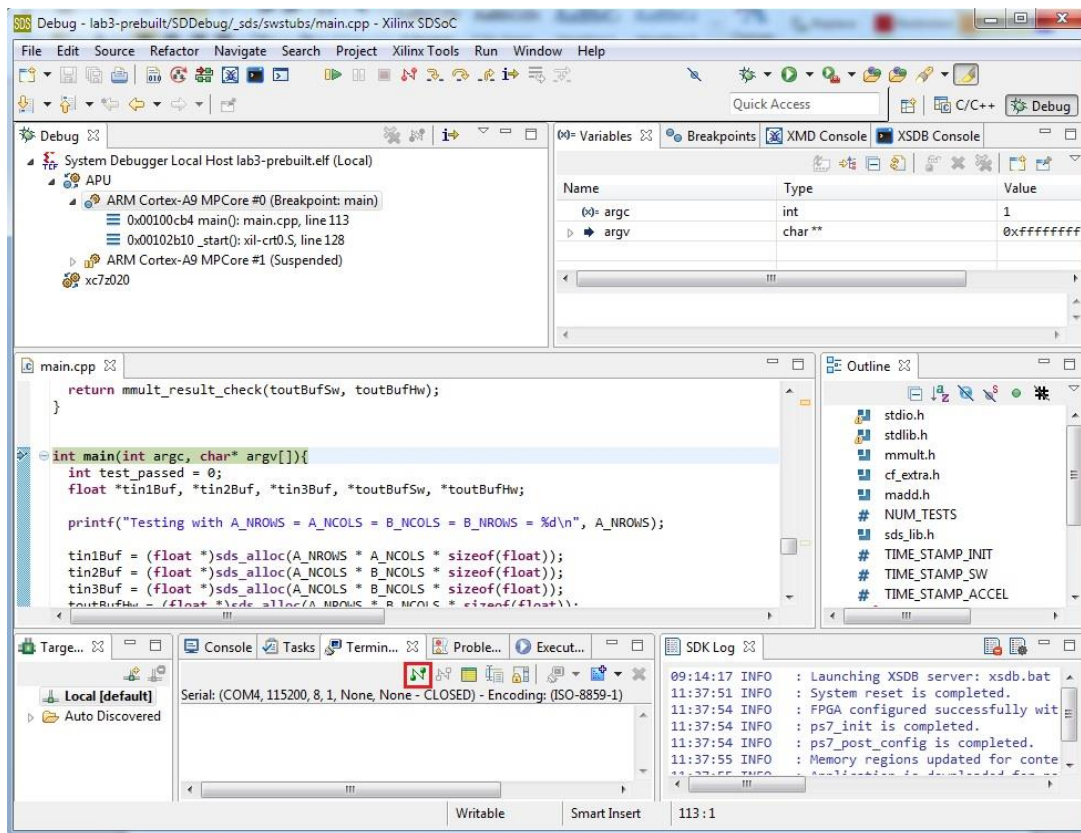


Figure 29. Debug Perspective

- d. Before you start running your application you need to connect the terminal window to the board so you can see the output from your program. Click the Terminal tab near the bottom of the debug window (configured with connection type: Serial, port: COM<n>, speed: 115200 baud), and then click the **green connection icon** to connect the terminal to the board (which should be powered up already).

Step 4: Run the Application

1. Click the Resume icon (green right-facing triangle with yellow base) to run your application, and observe the output in the terminal window.

Note: The source code window shows the `_exit` function, and the terminal tab shows the output from the matrix multiplication application.

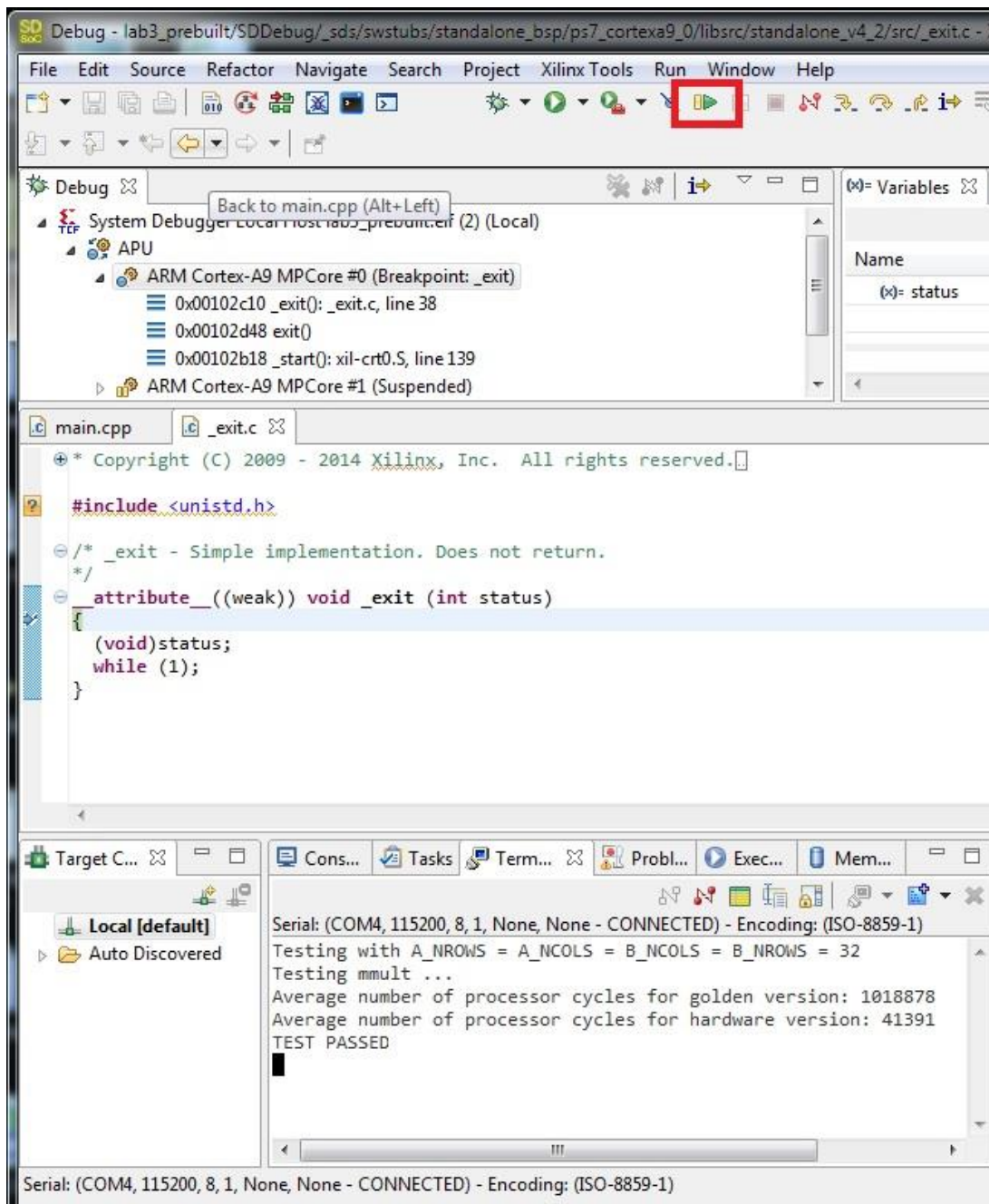


Figure 30. Debug perspective at the end of the application run

Step 5: Additional Exercises (Optional)

You might have noticed that the Debug perspective has many other capabilities that we have not explored in this lab. The most important is the ability to step through the code to debug it.

1. Step through the code using the debugger.
 - a. Right-click the folder at the top of the debug hierarchy in the Debug panel in top left corner, and click **Disconnect** in the menu.
 - b. Right-click the top-level debug folder again, and click **Remove all Terminated** in the menu.
 - c. Click on the BUG icon to launch the debugger. Then step through the code using the step-into, step-over, and step-return buttons

As you step through the code, examine the values of different variables.

IMPORTANT: *If you try to use the terminate and relaunch buttons SDSoC may display an error message saying it failed to launch. If that happens, restart SDSoC, and cycle power on the board. If you missed Step 3-2-1, SDK will attempt to clean and rebuild the application when you try to launch the debugger, and this could take up to 30 mins for this example.*

2. Boot from SD.
 - a. Copy the BOOT.BIN file from the sd_card folder inside the Debug folder to an SD card using Windows file copy/paste.
 - b. Plug in the SD card on the zc702 board, set the jumpers to sd-boot mode and power on the board.
3. Make sure the terminal tab in SDK is still connected, and you will see the output of your application on that terminal.
4. Debugging SDSoC Linux applications

In Lab1, you created an SDSoC Linux application and ran it from an SD card. You can also build a linux application with the Debug build-configuration and then debug it as you did in [Step 2](#) for standalone applications. The debug configuration setup for linux is slightly different, and is described below.

- a. Create a new project targeted to zc702 and Linux as in Lab1, but skip Step 1-4 of Lab1, where you set the active build configuration to Release. This gives you the default debug-build configuration, and when you build your SD card image the executable contains all the debug information. Select functions to move to hardware, and build the project as described in Steps 2 and 3 of Lab1 to completion without cancelling it midway.

IMPORTANT: *Building the executable can take 30-60 minutes depending on your machine.*

- b. Open the Terminal tab and connect the terminal as described in Step d.

- c. Copy the contents of the generated sd_card directory to an SD card, plug the SD card into the zc702 board.
- d. Make sure the board is connected to an Ethernet router using an Ethernet cable. Power on the board, and notice the Linux boot log displayed on the terminal. Look for a line that says "Sending select for 172.19.73.248...Lease of 172.19.73.248 obtained" or something similar, where the IP address assigned to your board is reported. Note this address for use in the next step.
- e. Back in SDSoc, right click on your project and select **Debug As > Debug Configurations...**, and then double-click **Xilinx SDSoc Application** on the left panel.
- f. Click **New** besides the Connection box and setup the IP address and port (1534) as shown in [Figure 31](#). Click **OK**.

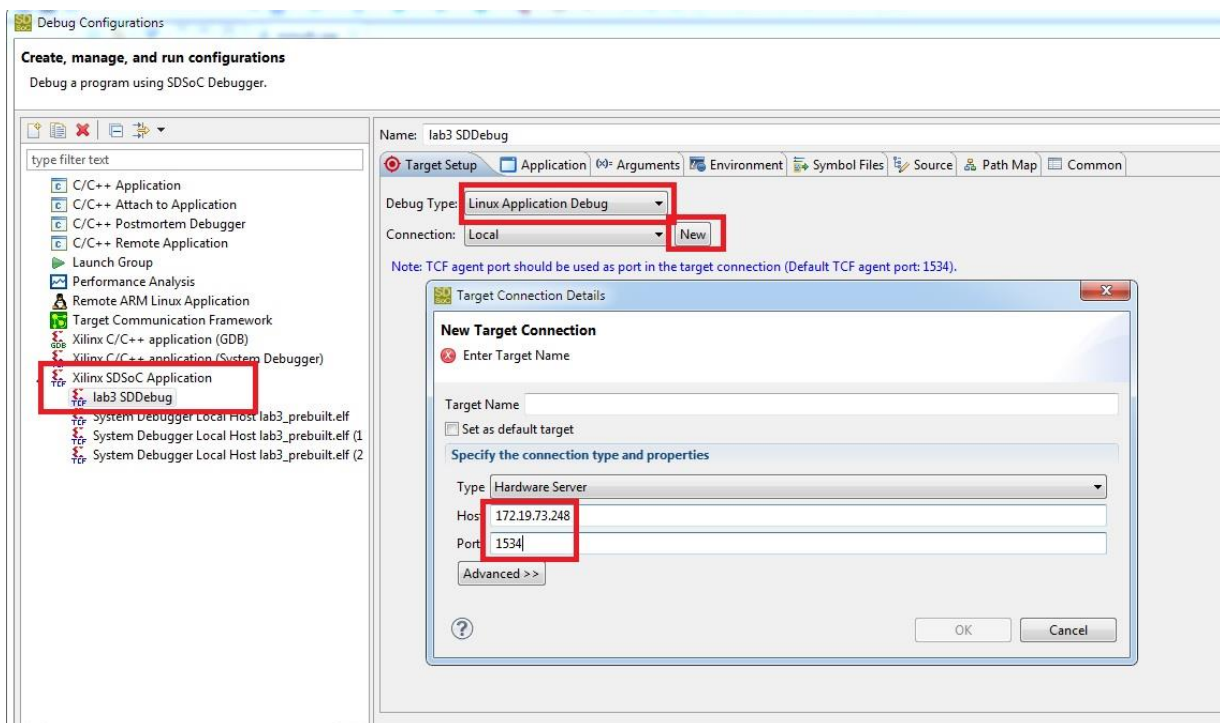


Figure 31. Setting up the Linux Debug Configuration

- g. Click the Application tab in the Debug Configuration panel and fill in the project name and the application's local and remote paths to the binary as shown in [Figure 32](#). The local path is the path to the ELF binary relative to your project (for example: Debug/linux_debug_example.elf) and the remote path is the path to the elf binary on the SD card starting at root (for example ./mnt/linux_debug_example.elf).

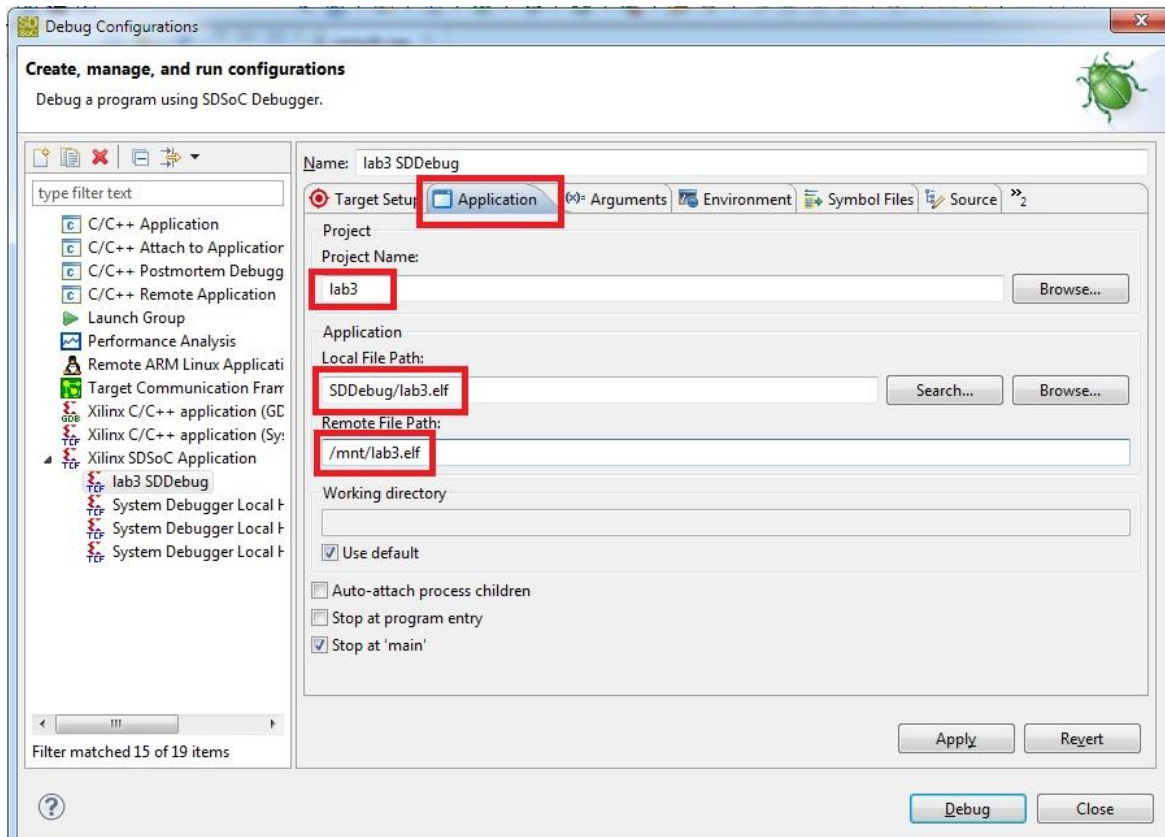


Figure 32. Setting up the Application path

- h. Click on the Debug button of [Figure 32](#) to go to the Debug Perspective, and run or step through your code as in Step 0, or Step c. Note that your application output is displayed in the Console tab not the Terminal tab.

Conclusion

In this lab you learned how to retarget your design to a different OS or platform, and how to run your standalone application through SDK. The optional exercises showed how to debug/step through the application, and how to run in sd-boot mode, and how to debug a Linux application.

Lab 3 Answers

- What functionality does the miniUSB cable provide in this lab?

It provides a UART functionality for the application. For this to work, the USB-UART driver must be installed on your laptop.

- Is it always possible to retarget designs from one platform or OS to another? When would this not work?

If a design depends on platform specific libraries, it cannot be retargeted to another platform. If you select the `zc702_hdmi` platform when you create a new project you will find some design examples that make use of platform specific libraries and cannot be retargeted to any other platform.

Some platforms may contain a smaller Zynq®-7000 All Programmable SoC device than others. So a design that fits on a `zc706` platform might not fit on the smaller, `zc702` platform, although you will know this only when you attempt to build the project.

Linux applications cannot be retargeted to standalone if they use threads or standard file system calls because standalone mode does not support these.

Lab 4 Introduction to Performance Estimation

Introduction

This lab shows you how to obtain an estimate of the performance that you can expect from your application, without going through the entire build cycle.

Objectives

After completing this lab, you will be able to use SDSoC to obtain an estimate of the speedup that you can expect from your selection of functions to accelerate.

Procedure

This lab is separated into 3 steps, followed by general instructions and supplementary detailed steps allowing you to make choices based on your skill level as you progress through the lab.

General Flow for this Lab

- **Step 1:** Setup the zc702 board and cables, and power on
- **Step 2:** Set up the project and use SDEstimate configuration
- **Step 3:** Build the application and observe the estimated performance

Step 1: Setup the board and cables and power on

The first step for this lab is to set up the zc702 board for the lab. You will need a miniUSB cable to connect to the UART port on the board which will talk to a serial terminal on the SDK. You will also need a micro USB cable to connect to the Digilent port on the board to allow downloading the bitstream and binaries. Finally, you need to ensure that the jumpers to the side of the SD card slot are set correctly to allow booting from an SD card.

Step 2: Set up the project and use SDEstimate configuration

5. Create a new SDSoC project (lab4) for the **ZC702 platform** and **Standalone** using the design template for Matrix Multiplication and Addition.
6. In the Project Explorer view, expand the src directory to show the source files. Expand the `madd.cpp` and `mmult.cpp` files to reveal the functions within those files. In `madd.cpp`, right-click the function `madd` and choose **Toggle HW/SW** and do the same for the function `mmult` in the file `mmult.cpp`.
7. Right-click the `lab4` project folder and select the **Build Configurations** menu item. Then select **Set Active > SDEstimate** menu item.

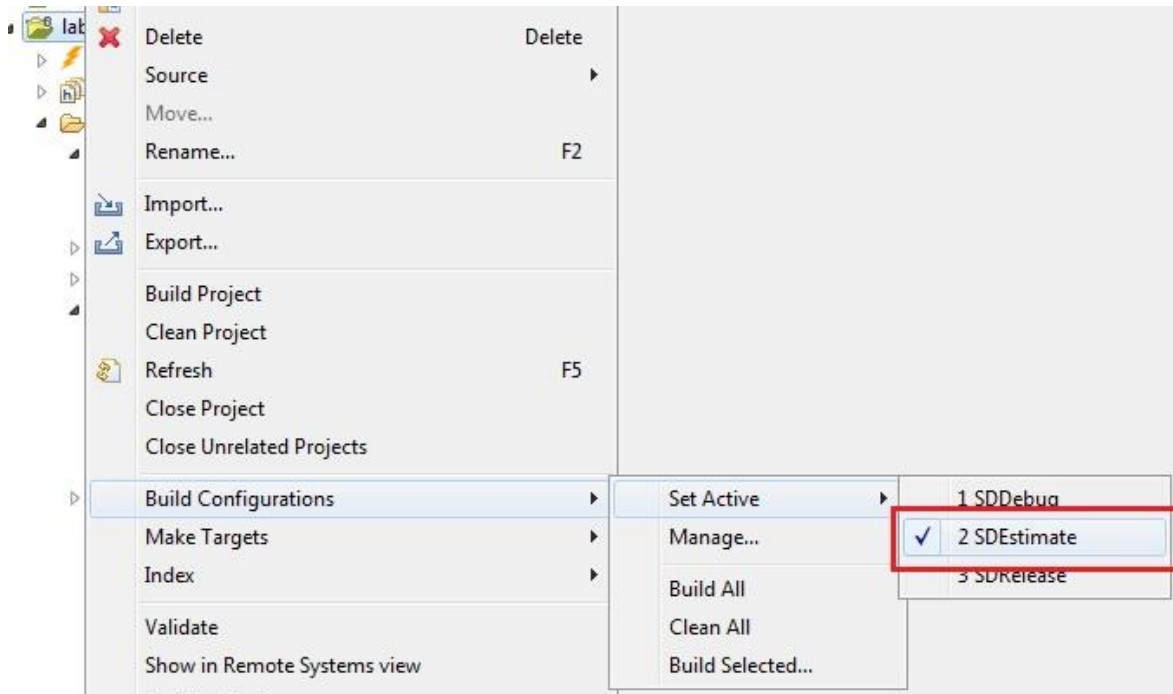


Figure 33, Change Build Configuration

8. Now you can build the project by right clicking on lab4 project and selecting **Build Project**. After the build is over, you can see an initial report as shown below. This report contains a hardware-only estimate summary and has a link which can be clicked to obtain the software run data, which updates the report with comparison of hardware implementation v/s the software-only information.

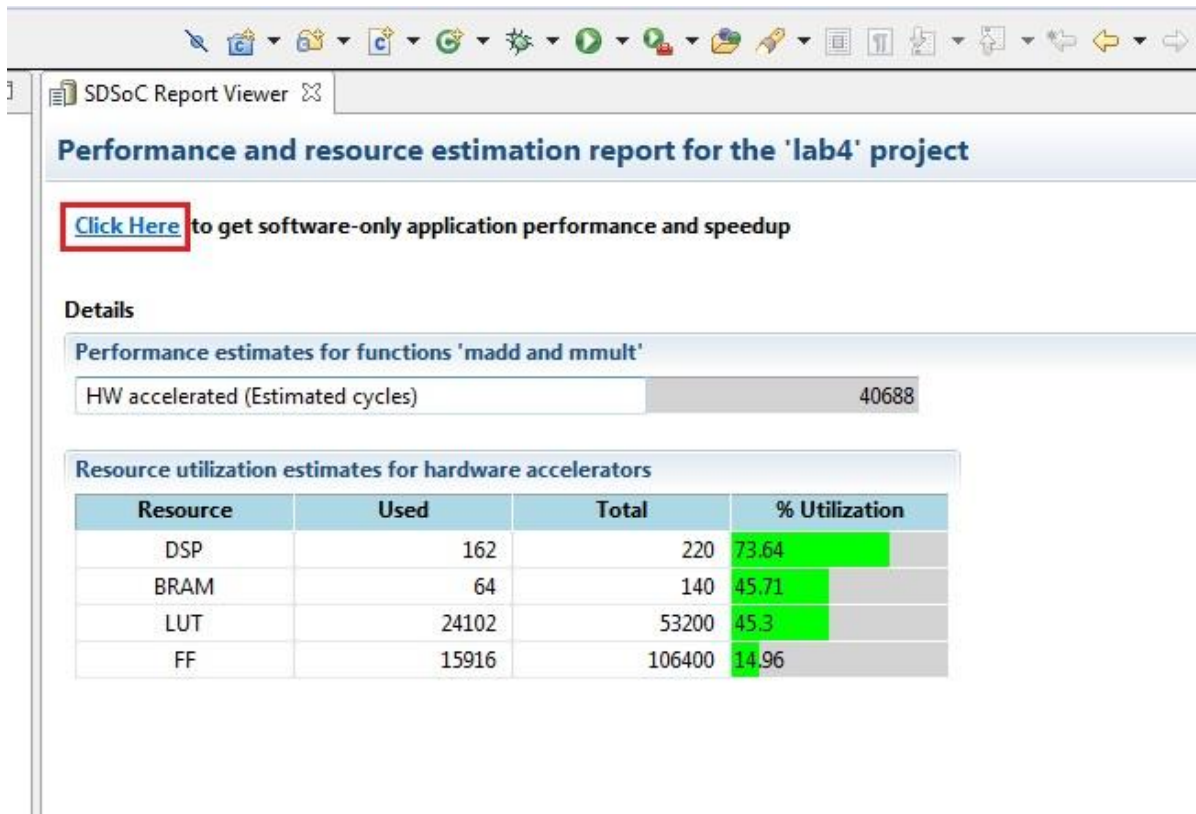


Figure 34, Initial report with hardware-only estimates



IMPORTANT: Make sure the board is turned on before going to the next step.

Step 3: Collect Software Run Data and Generate Performance Estimation Report

1. Click the **Click Here** link to launch the application on the board. The **Run application to ...** dialog box opens where you can select a pre-existing connection, or create a new connection to connect to the target board.

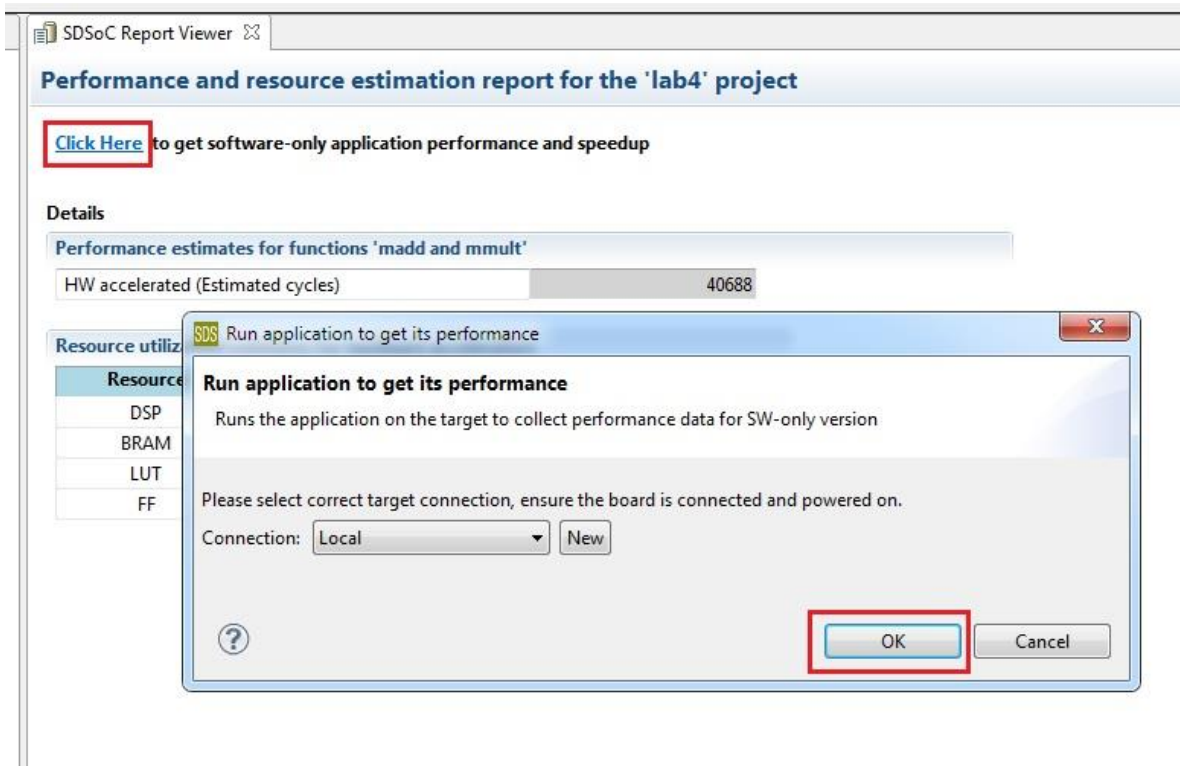


Figure 35. Run application on target board

You would note that the debugger resets the system, programs and initializes the FPGA, runs a software-only version of the application and collects performance data and uses it to display the performance estimation report.

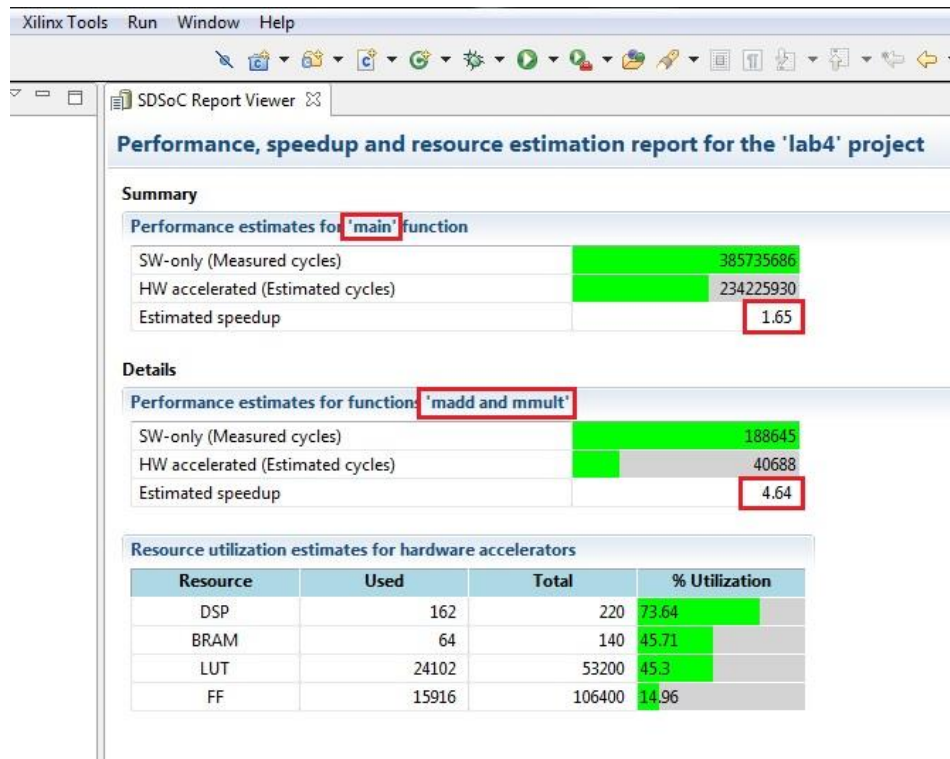


Figure 36. Performance Estimation Report

Step 4: Change the scope of overall speedup comparison

1. In the estimation report shown on the previous page, the first line line of the report shows the estimated speedup for the top level function. This function is set to **main** by default. However, there might be code that you would like to exclude form this comparison, e.g. allocating buffers, initialization and setup, etc. If you wish to see the overall speedup when considering some other function, you can do this by specifying a different function as the root for performance estimation flow. The flow works with the assumption that all functions selected for hardware acceleration are children of the root.
2. To change the perf root, in the **Project Explorer**, right-click the function that you are interested in selecting as root and click the menu item to **Mark as root for estimate flow**. You should be able to see a small R icon on the top left of that function listed as shown below. As mentioned earlier, it is expected that the selected function will be a parent of the functions that are selected for hardware acceleration.

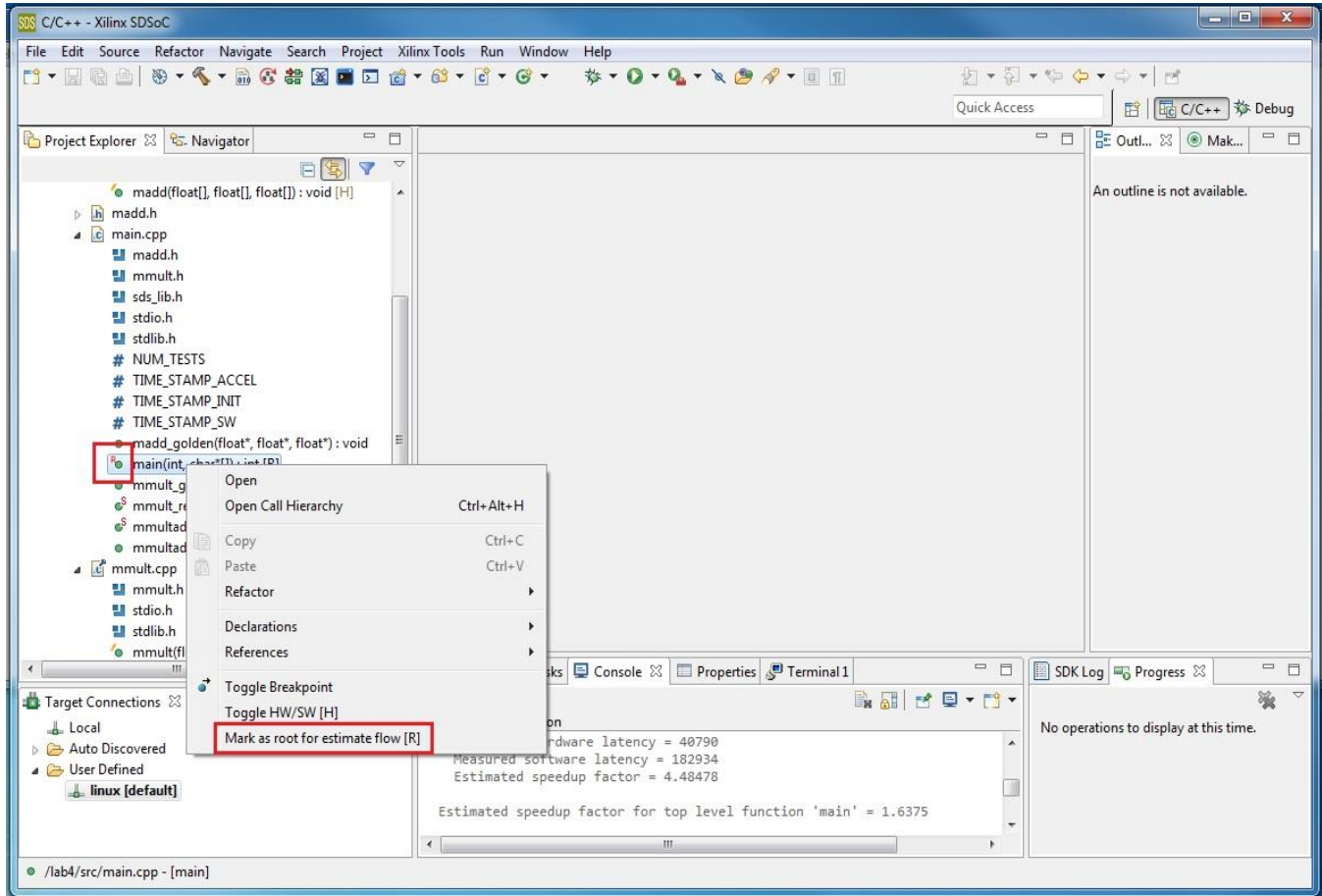


Figure 37, Change Perf root

- Now you can clean and build the project and generate the estimation report again and you will get the overall speedup estimate based on the function that you selected.

Step 5: Using the Performance Estimation Flow With Linux

- Create a new SDSoc project (lab4-Linux) for the **ZC702 platform** and **Linux** using the design template for Matrix Multiplication and Addition.
- In the Project Explorer view, expand the src directory to show the source files. Expand the madd.cpp and mmult.cpp files to reveal the functions within those files. In madd.cpp, right click on the function madd and choose **Toggle HW/SW** and do the same for the function mmult in the file mmult.cpp.
- Right-click the **lab4** project folder and select the **Build Configurations** menu item. Then select **Set Active > SDEstimate** menu item.
- Now you can build the project by right clicking on lab4 project and selecting **Build Project**.

5. At this point, the procedure deviates from the steps followed to obtain the performance estimation under Standalone. Copy the contents of the folder `sd_card` under `SDEstimate` to an sd card and boot up the board.
6. Make sure the board is connected to an Ethernet router using an Ethernet cable. Power on the board, and notice the Linux boot log displayed on the terminal. Look for a line that says "Sending select for 172.19.73.248...Lease of 172.19.73.248 obtained" or something similar, where the IP address assigned to your board is reported. Note this address for use in the next step. If you miss this statement in the log as it scrolls by, you can obtain the IP address of the board by running the command `ifconfig`.
7. Back in SDSoC, right-click your project and select **Debug As > Debug Configurations...**, and then double click on **Xilinx SDSoC Application** on the left panel.
8. Click **New** besides the Connection box and setup the IP address and port (1534) as shown in the figure below and click **OK**.

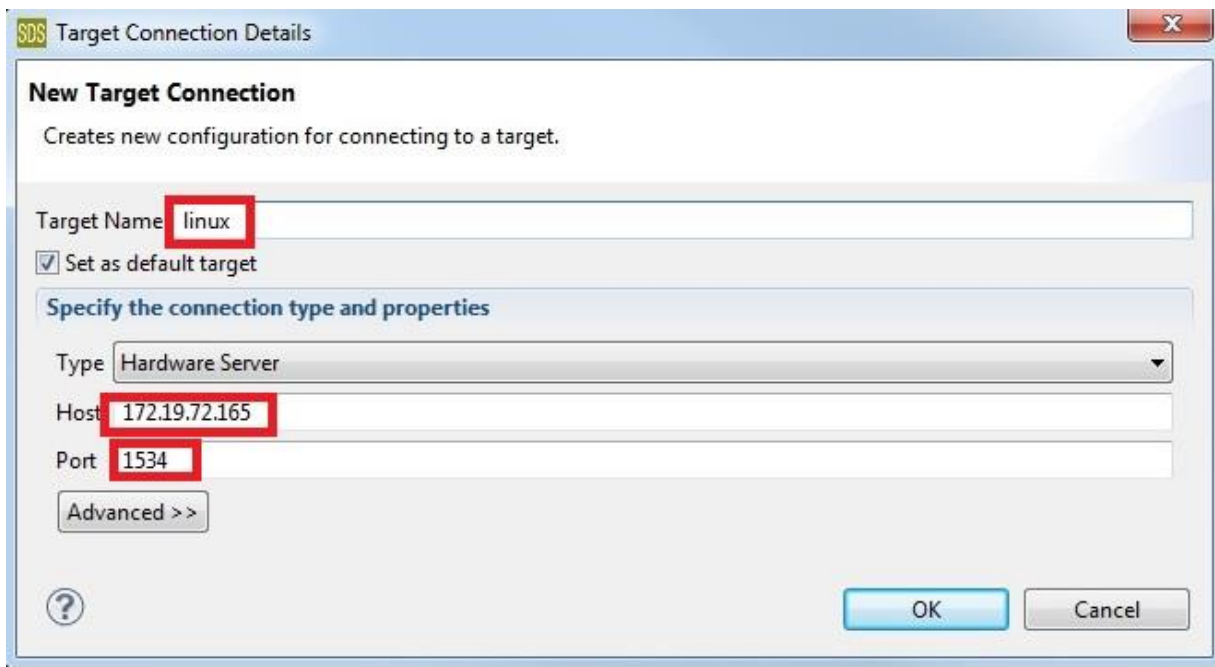


Figure 38. Setting up the Linux Debug Configuration

9. Click the Application tab in the Debug Configuration panel and fill in the project name and the application's local and remote paths to the binary as shown in the figure below. The local path is the path to the elf binary relative to your project (for example: `SDEstimate/lab4-Linux.elf`) and the remote path is the path to the ELF binary on the SD card starting at root (for example: `/mnt/lab4-Linux.elf`).

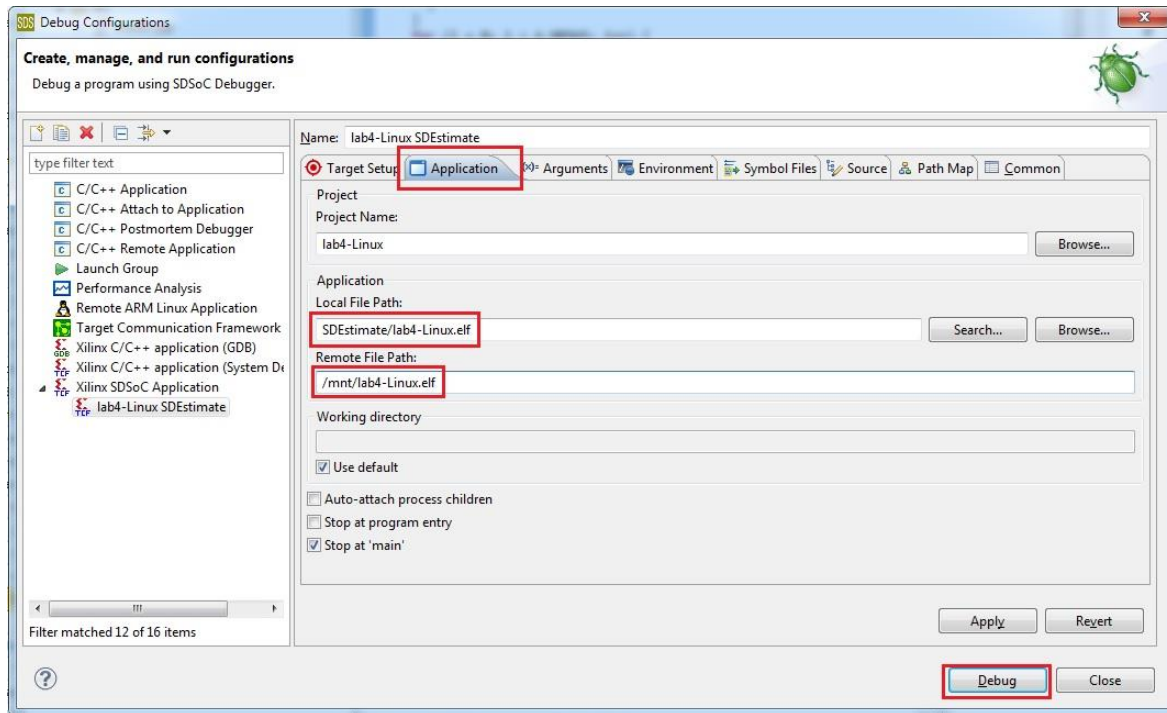


Figure 39. Setting up the Application path

- Click the **Debug** button as shown in the figure above to go to obtain the software profile data and complete the performance estimation flow under Linux.

Appendix A Supplementary Installation Notes

The SDSoC Environment includes the installation of the following supporting tools:

- SDSoC tools, including an Eclipse/CDT-based GUI and ARM GNU toolchain
- Vivado® Design Suite System Edition 2014.4, which includes Vivado and Vivado HLS

Additional requirements and recommendations are described below.

GNU Toolchain Requirements

SDSoC includes the same GNU ARM toolchain included with the Xilinx Software Development Kit (SDK) 2014.4, which also provides additional tools used by SDSoC. The SDSoC Environment setup script sets PATH variables to use this toolchain.

The Sourcery CodeBench toolchain contains 32-bit executables, requiring your Linux host OS installation to include 32-bit compatibility libraries. The Xilinx version of the Sourcery CodeBench Getting Started guide can be found as part of the SDSoC installation at SDK/SDK/gnu/arm/lin/share/doc/xilinx-arm-xilinx-linux-gnueabi/pdf/getting-started.pdf. Section 2.2.1 "Host Operating System Requirements" describes host requirements for Sourcery CodeBench software including the 32-bit libraries and where and how to access them. For more information, a similar getting started guide is available from Mentor Graphics at their website.

RHEL 5 64-bit x86 Linux installations typically include the 32-bit compatibility libraries, but RHEL 6 might not and may need to be added separately; see <https://access.redhat.com/site/solutions/36238>.

On RHEL, 32-bit compatibility libraries can be installed by becoming a superuser (or root) with root access privileges and running the command:

```
yum install glibc.i686
```

The version of the toolchain can be displayed by running the following command:

```
arm-xilinx-linux-gnueabi-c++ -v
```

The last line of the output printed in the shell window should be:

```
gcc version 4.8.3 (Sourcery CodeBench Lite 2014.05-23)
```

Tips and Suggestions

If you encounter issues using SDSoC after installation, consult this section for potential issues and their resolution.



IMPORTANT: When using Vivado tools on a Windows platform, path names longer than 260 characters may result in the error message: ERROR: [Common 17-143] Path length exceeds 260-Byte maximum allowed by Windows: <LongPathFileName>. Possible solutions to shorten path lengths or to avoid this are described in Answer Record AR# 52787

<http://www.xilinx.com/support/answers/52787.htm>, for example use shorter path names, map a new drive letter to a lower directory in the path, and other methods.



CAUTION! If Cygwin is included in a global PATH environment variable and issues are encountered, it may need to be temporarily removed when running SDSoC tool flows. For example, in a command shell, type: `set PATH=%PATH;c:\cygwin\bin;=%`.



CAUTION! In each shell used to run SDSoC, use only the environment setup scripts corresponding to the Xilinx tool releases or PATH environment setting listed above. Running Xilinx design tool environment setup scripts from other or additional releases in the same shell will result in incorrect SDSoC tool behaviors or results.

Appendix B Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

Solution Centers

See the [Xilinx Solution Centers](#) for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

These documents provide supplemental material useful with this guide:

[Vivado® Design Suite Documentation](#)

1. *Vivado Design Suite User Guide UG896: Designing with IP, Chapter 8, Creating and Packaging IP*
http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/ug896-vivado-ip.pdf
2. *User Guide 1146 - SDSoc Platforms and Libraries*, <sdsoc_install_root>/docs/ug1146-sdsoc_platforms_and_libraries.pdf.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>.

© Copyright 2014-2015 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.