Creating a System with SDSoC

Introduction

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

Note that now Xilinx includes SDSoC as part of SDx. From our point of view SDx and SDSoC are equivalent. Xilinx has two tools under SDx which are SDSoC (C/C+) and SDAccel (OpenCL). We will only work with SDSoC in these labs.

Objectives

After completing this lab, you will be able to:

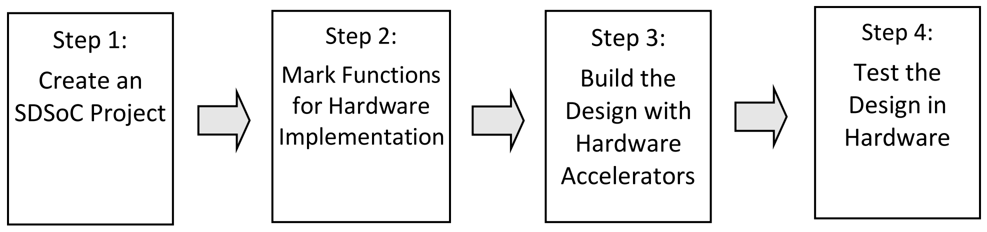
- Create a new SDSoC project for your application from a number of available platforms and project templates
- Mark functions for hardware implementation
- Build your project to generate a bitstream containing the hardware implemented function and an software executable file that invokes this hardware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

Procedure

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

This lab comprises four primary steps: You will create an SDSoC project, mark two functions for hardware implementation, build the design with the hardware accelerators, and, test the design in hardware.

General Flow for this Lab


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

Create an SDSoC Project Step 1

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

1-1-1. Open SDSoC by selecting **Start > All Programs > Xilinx Design Tools > SDx IDE 2018.2**

The Workspace Launcher window will appear.

1-1-2. Now create a directory that you want to use as your workspace directory. Give it a name such as "sdsoc_workspace". All your projects will be stored in there so make sure that you have read/write access and that your data is not lost.

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

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

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

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

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

1-1-6. You are creating an Application project. Enter **lab1** as the project name, select the *zed* board via drop-down button, select *Standalone* as the target OS, and click **Next**.

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

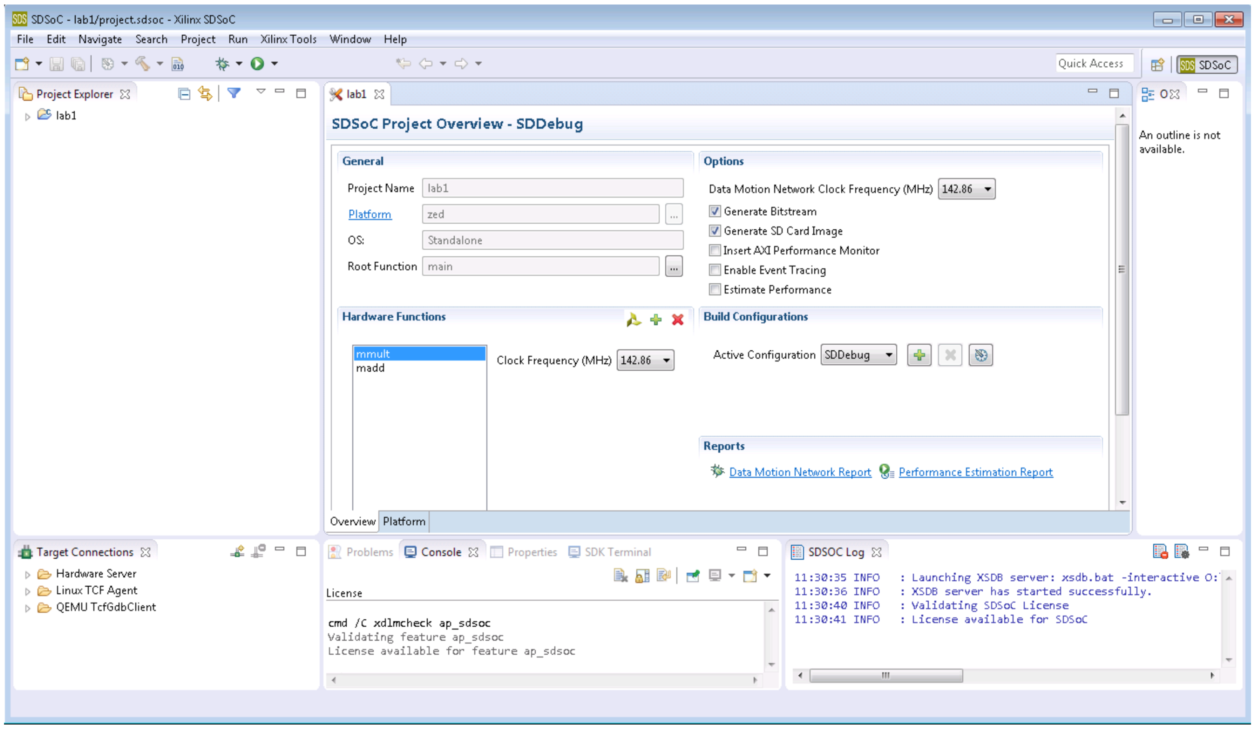
1-1-7. Select **empty application** for the *zed* as the source.

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

The created project will be displayed. In the left, you will see *Project Explorer* under which the **lab1** project directory will be displayed (you may have to expand the folder). It shows the *Includes* and *src* folders. We need to add the source code the the *src* folder. Right click on the *src* folder and select import then general and file system. Download the sources from blackboard for the lab, located in the FPGA lab directory => student_sources.zip and unzip the directory. Browse to that directory and select all the sources to be imported into your project. There is a single set of sources and you will use these for **lab1**, **lab2** and as the starting point for your project work.

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



SDSoC Project Overview - SDDebug

General

Project Name	lab1
Platform	zed
OS:	Standalone
Root Function	main

Options

Data Motion Network Clock Frequency (MHz) 142.86

- [✓] Generate Bitstream
- [✓] Generate SD Card Image
- [ ] Insert AXI Performance Monitor
- [ ] Enable Event Tracing
- [ ] Estimate Performance

Hardware Functions

mmult
madd

Clock Frequency (MHz) 142.86

Build Configurations

Active Configuration SDDebug

Reports

Data Motion Network Report Performance Estimation Report

License

```
cmd /C xdlmcheck ap_sdsoc
Validating feature ap_sdsoc
License available for feature ap_sdsoc
```

```
11:30:35 INFO   : Launching XSDB server: xsdb.bat -interactive O:
11:30:36 INFO   : XSDB server has started successfully.
11:30:40 INFO   : Validating SDSoC License
11:30:41 INFO   : License available for SDSoC
```

Figure 5. Created SDSoC project

Mark Functions for Hardware Implementation Step 2

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

2-1-1. Click on the "+" sign in the Hardware Functions area to open up the list of software functions in the design.

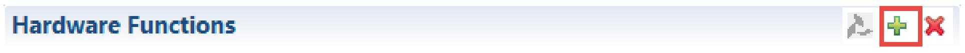
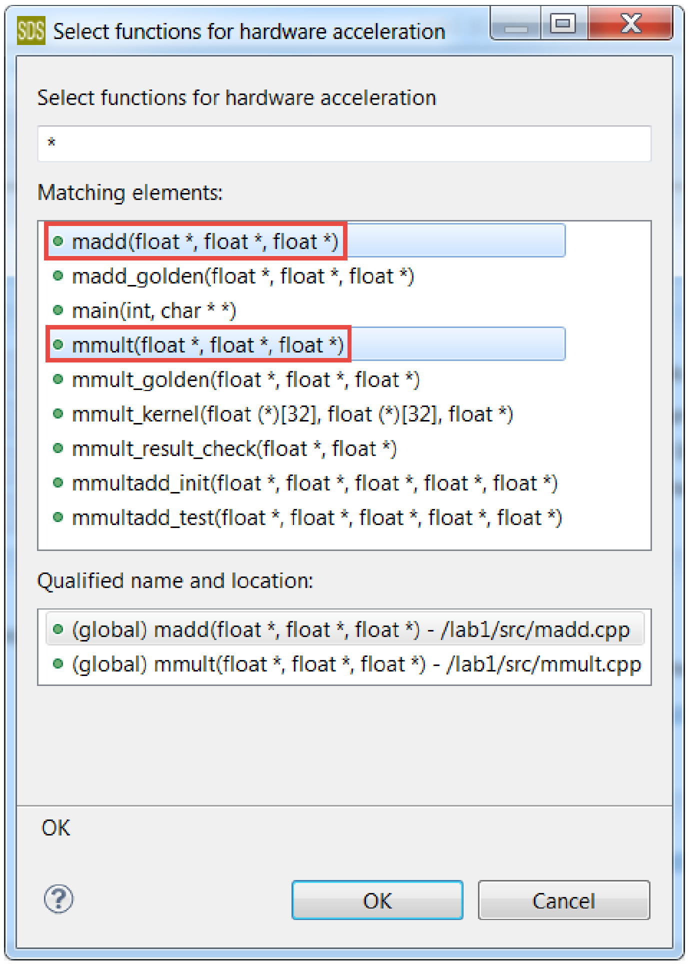


Figure 6. Invoking functions list to select functions which can be hardware target

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

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

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


Select functions for hardware acceleration
Select functions for hardware acceleration
*
Matching elements:

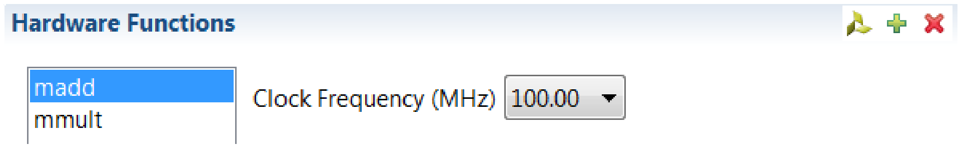
madd(float *, float *, float *)
madd_golden(float *, float *, float *)
main(int, char * *)
mmult(float *, float *, float *)
mmult_golden(float *, float *, float *)
mmult_kernel(float (*)[32], float (*)[32], float *)
mmult_result_check(float *, float *)
mmultadd_init(float *, float *, float *, float *, float *)
mmultadd_test(float *, float *, float *, float *, float *)

Qualified name and location:

(global) madd(float *, float *, float *) - /lab1/src/madd.cpp
(global) mmult(float *, float *, float *) - /lab1/src/mmult.cpp

Figure 7. Selecting functions for hardware target

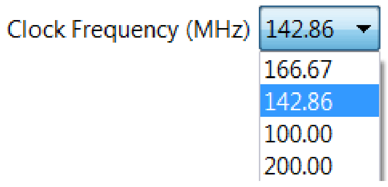
2-1-3. The two function names will be added into the Hardware Functions window (note that they were already added by default in the template).



Clock Frequency (MHz) 100.00

Figure 8. Target hardware accelerators entries updated

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



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

2-1-4. Leave the clocks set to the default frequency of 142.86 MHz for Zed.

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

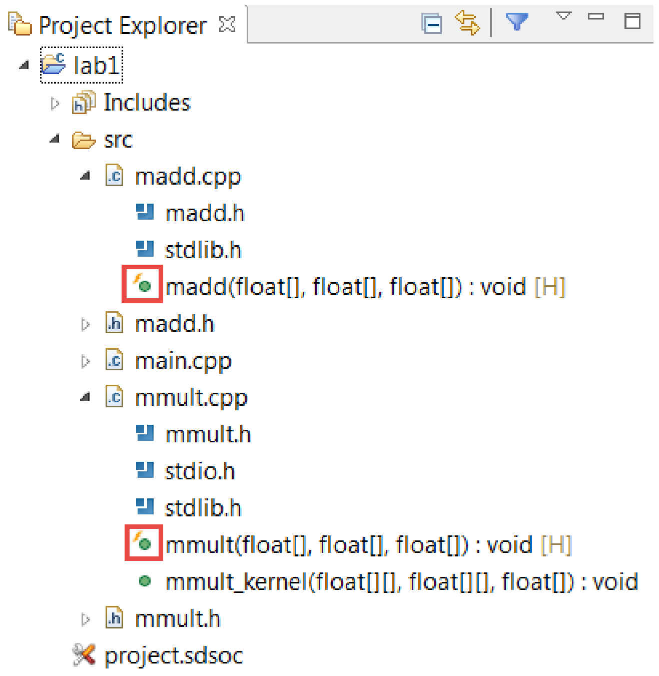

```
Project Explorer
lab1
  Includes
  src
    madd.cpp
      madd.h
      stdlib.h
      madd(float[], float[], float[]) : void [H]
    madd.h
    main.cpp
    mmult.cpp
      mmult.h
      stdio.h
      stdlib.h
      mmult(float[], float[], float[]) : void [H]
      mmult_kernel(float[][], float[][],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

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

Build the Design with Hardware Accelerators Step 3

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

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

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

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

Note1 :You will need to re-select the hardware functions when you change between SDDebug and SDRelease.

Note2 : Building the project and generating the results in the next step may take 20 to 30 minutes.

3-1-3. Uncheck "generate SD card image" which we are not going to use.

3-1-4. Right-click on **lab1** and select **Build Project**

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

This may take about 25 to 30 minutes. When it completes you can move to section 3-1-5. In the meantime do the task below.

While the tool does its magic we will use the time to learn a bit more about the current technology and business state of the FPGA market with two big companies: Xilinx and Intel/Altera battling for supremacy. These two companies control 85% of a $10B programmable integrated circuit market and as you might know Intel bought Altera in 2016. Read the documents "FPGA programming for the masses" and "Intel VS Xilinx" in the reference material section of Blackboard and answer the following questions:

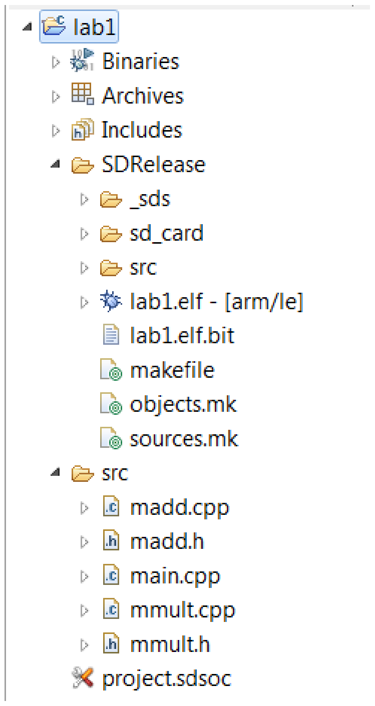
0.1 Xilinx has based their HLS tools around the C/C++ language. What is the language that Altera/Intel have chosen for their HLS efforts?

0.2 The current state-of-the-art devices from Xilinx are the Ultrascale family. What is the name of the state-of-the-art Altera FPGA devices?

0.3 Vivado is the name that Xilinx gives to their main set of implementation tools. What is the name that Altera/Intel gives to their main set of tools?

0.4 What is the name of the transistors that both Intel/Altera and Xilinx are using to create their latest very high-density FPGA chips?

3-1-5. Expand the **lab1** directory in Project Explorer and observe that *SDRelease* folder is created along with virtual folders of *Binaries* and *Archives*. Expanding the *SDRelease* folder shows **_sds, sd_card, src** folders along with **lab1.elf** [executable]**, lab1.elf.bit** [hardware bit file] and several make files.



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

(a) Generated

Figure 12. Project Explorer folders

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

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

3-1-6. Expand the _sds folder.

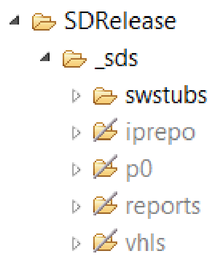


Figure 13. The generated _sds folder content

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

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

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

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

The *reports* folder consists of log files and the **data_motion.html** containing the data motion network report. It is always useful to understand how many resources your design is using in your Zynq device. Open sds.rpt and answer the following questions:

1. What is the percentage of LUTs and Registers used by the design (round the number to the closest integer) ?

3-1-7. In the SDSoC Project Overview window, under the Assistant or Reports pane, click on **Data Motion Network** report.

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

There are two accelerated functions- madd and mmult. They are given instance names of madd_0 and mmult_0. Each function has three arguments and hence three ports. Notice that the out_C port of mmult_0 is directly connected to A_PORTA port of madd_0 port. Now consider these questions:

2. What type of DMA IP is being used to move data in and out of the accelerators ?

3. What Zynq port this IP is connected to ?
4. How much data in bytes is being moved in and out of the accelerators ? Why ?

3-1-8. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2018.2**

3-1-9. Open the design by browsing to ….*\lab1\SDRelease_sds\p0\vivado\prj\* and selecting the prj.**xpr**.

3-1-10. 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).

This platform supports four clocks. There are four reset blocks, one for each clock. As only a single clock in Zed was used for the accelerators, it connects only one reset block is connected (second from top). The other blocks will be optimized away during synthesis.

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

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

Observing the block design Answer the following questions:

5. How many input and output ports for control, data, reset and clock has the mmult accelerated block? 1 Input Control, 1 Input Clock, 1 Input Reset, 2 Input Data (A and B) and 1 Output Data (C)

6. What is the polarity of the reset signal for the mmult block? Negative

Observe the address editor tap.

7. At what address position the mmult and mmad accelerators are mapped? Madd – 0x43C0,0000. Mmult – 0x43C1,0000

8. What is the size of the address range ? Size = 64K

Now go back to the main diagram and double click on the z7 (processor system PS) component and observe the clock configuration.

9. What are the clock frequencies requested for the CPU and DDR components? CPU – 666MHz. DDR (Memory) – 533MHz.

As you have now verified the processor and DDR work at much higher frequency than the FPGA which is typical since the FPGA obtains its performance with parallelism and custom logic rather than a high frequency.

3-1-11. Close Vivado by selecting **File > Exit**

Test the Design in Hardware Step 4

4-1. Make sure that board is set for JTAG programming. All the jumpers near the DIGILENT name should be set to GND.

4-2. Connect the Zed board cables. You need to connect the 2 USB cables. One is used for programming the FPGA device via JTAG and the other for UART communications with the computer. Then connect the power to the board and switch it on.

4-3. Create a terminal that will be used for the UART comms from the board. Go to Window -> show view -> other -> terminal. In the Terminal1 created go to settings and set the baud rate to 115200 and the port to the correct communications port. For example com5.

4-4. Then right click on the application elf under SDrelease. Then choose "run as" and then "**launch on hardware (SDx application debugger)"** (do not use system debugger) as the launch target. You will see the bit file programming the FPGA and then the application running with messages being printed in the terminal. Write down the answer to these questions:

10. How many clock cycles the application takes on software ?
11. How many clock cycles the application takes on hardware ?
12. Which is the acceleration factor (for example if the hardware is half the speed of the software then the acceleration factor is 0.5x if it is twice the speed is 2x ?

As you have now verified the FPGA can be quite slow if just C or C++ code with no further hardware optimizations is used for programming. In lab2 we will work out how to make our design faster.

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

4-4-2. Turn OFF the power to the board.

Conclusion

In this lab, you created a project in the SDx/SDSoC Development Environment using one of the available project templates. You then identified the functions which you wanted to put in the PL section of the Zynq chip to improve the performance. Once the system was built, you analyzed the Data Motion network and the created Vivado IPI project. Finally, you verified the design in hardware. You should have observed that the design works in hardware but it is slower than software. We will need to optimize the hardware design to accelerate it in the next lab.

Learning outcomes

1. Learn how to use modern FPGA implementation tools based on high-level descriptions such as C/C++.
2. Learn how to analyze the performance results to understand the effects of design decisions in the final solution.
3. Learn how to use real FPGA boards and the details of software/hardware interfacing and configuration.

4. Learn about the main FPGA manufactures and state-of-art FPGA chips and tools.

Marking scheme

Marking for this lab is based on the Quiz that will ask questions as shown in the lab and assign marks depending on the level of difficulty. Additional quiz questions will assess that you have understood the theory. You can use your notes when doing the quiz but it needs to be done individually. You will only do the quiz once you have finished the second part lab2.