Experiment 0 - Introduction to Verilog and Cocotb

Objectives

The purpose of the first laboratory work is to introduce you to Verilog, Quartus, and cocotb. Throughout this preliminary work, you will learn basic Verilog concepts and create a simple design. You will be introduced to cocotb and test your designs using the given cocotb testbenches. You will also be introduced to the Quartus software suite required to implement your design onto the DE1-SoC Board, equipped with a field programmable gate array (FPGA) and switch inputs, general purpose Input/Output (I/O) pins, and LEDs that we use in the laboratory.

Please take your time and familiarize yourself with everything present in this preliminary work as what is taught here will be used throughout the entire course.

Although the preliminary work grade is not included in the overall grade of the experiment,

1. **You need to get at least 50% from the report**
2. **Your codes should pass at least half of the testbenches (2/3 for this experiment)**

Both are required to be eligible to attend the experiment session. We will run the testbenches automatically on your submitted codes for the second requirement.

1 Preliminary Work

In this experiment, you will implement a few simple modules in Verilog. These modules will then be used in the laboratory to be embedded into the FPGA boards. This laboratory is designed as a tutorial, so take your time and learn what is taught at each step.

For the preliminary work, you will submit two separate assignments: A .pdf report, and Verilog codes inside a .zip file.

To fulfill the requirements of this laboratory work, the following tasks should be performed. **Note: If necessary you should only include important parts of the code in the pdf report. The full code should be submitted separately.**

Your report must include RTL schematic views of the specified components, along with screenshots showing the successful execution of the testbenches.

All designs must be written in Verilog HDL and included in your submission as a ".v" file wherever required.

1.1 Reading Assignment

The Laboratory Manual, where the regulations and other useful information exist, is available on the ODTUClass course page. **Read the Laboratory Manual thoroughly**. Refer to the Verilog Combinational Lecture Part1 PDF and the video published for this laboratory as they contain everything about Verilog you need **for this experiment**. We also assume you are familiar with Chapter 1 of the EE348 course as this experiment is complementary to that.

1.2 Verilog, Quartus and Cocotb

In the EE348 lectures, you will learn conventional digital design steps of pen&paper design where you will go through various methods to design a digital circuit.

In the laboratory, you will use a Hardware Description Language (HDL). An HDL is **NOT** an ordinary programming language; rather, it is a language designed to **describe hardware**—specifically, to define logic circuits in textual form. This allows you to convert the designs covered in lectures for use with FPGA (Field Programmable Gate Array) boards. There are three major HDLs: Verilog, SystemVerilog, and VHDL. We have chosen Verilog because its syntax is similar to C, and we do not require the additional testbench extensions found in SystemVerilog.

1.2.1 Quartus Installation

Quartus is a design software suite developed by Intel (formerly Altera) for FPGA, CPLD (Complex Programmable Logic Device), and SoC (System on Chip) development. It provides tools for design entry, synthesis, simulation, verification, and programming of FPGA-based designs. We will use Quartus Prime Lite, a free version for FPGAs like Cyclone IV/V. Follow the steps below to install Quartus on your computer.

1. Go to Quartus download page
2. Download Quartus Prime Lite 23.1, "qinst-lite-windows-23.1std.1-993.exe"
3. Accept the Legal Disclaimer.
4. Run the downloaded file.
5. When installing, check the boxes as seen in Figure 1.
6. Click "Download".
7. Follow the installation steps.
8. Quartus should be installed to your computer.

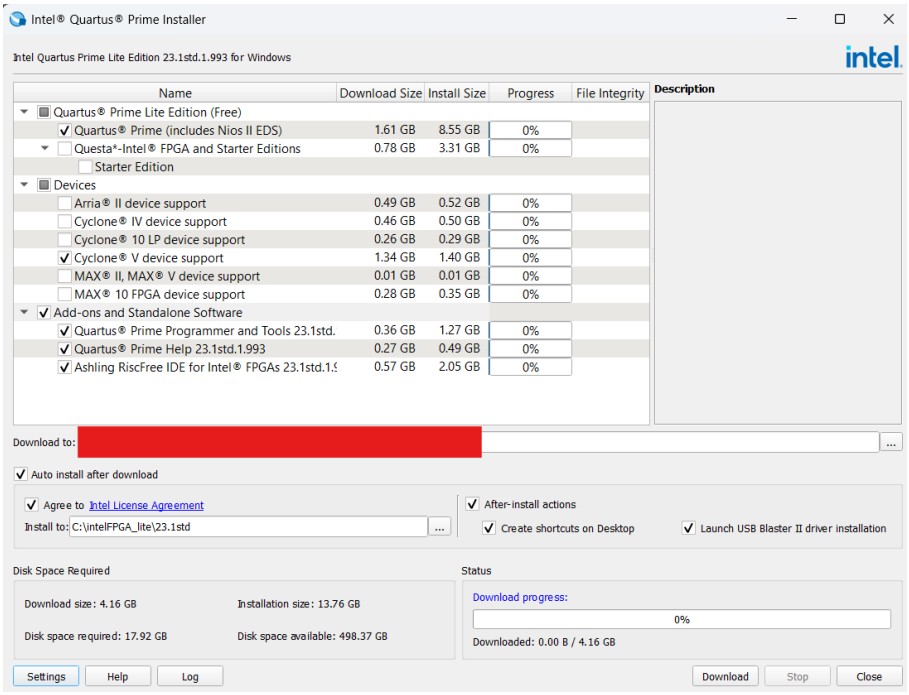


Figure 1: Quartus installation selection page.

1.2.2 Cocotb Installation

For verifying Verilog designs, we will use cocotb, a free, open-source, coroutine-based co-simulation testbench environment for VHDL and Verilog. Cocotb is hosted on GitHub and supports a wide range of HDL simulators on Linux, Windows, and macOS. Throughout most of the labs, you will receive ready-made testbenches for testing your designs. By the end of the semester, however, you will be required to write your own tests using these provided testbenches as a starting point.

Cocotb is regularly updated, meaning the installation steps given in this part might get outdated. The best way of installing cocotb is following the installation section of the cocotb documents.

The installation process for **Windows** as of writing this manual is as follows:

1. Install the latest miniconda version from conda.io documents, you do not need to add miniconda to PATH but select to register it as default. This will give you a new package management system and environment with Anaconda Prompt terminal. **(You can also install the full version of Anaconda instead of miniconda)**

2. Open the newly installed Anaconda Prompt and use the following line to install a compiler (GCC or Clang) and GNU Make:

   ```
   conda install -c msys2 m2-base m2-make
   ```

3. From the Anaconda Prompt install cocotb with the following line (You may need to install Visual Studio C++ 2014 redistributable if you don't have it):

   ```
   pip install cocotb
   ```

4. Also install "rich" with the following line as we use it to print debug outputs to the terminal:

   ```
   pip install rich
   ```

5. Now you need a Verilog simulator for cocotb to use. For this course we will use iVerilog (Icarus Verilog), you can download iverilog from Icarus Verilog for Windows site. Make sure to install the newest version (v12+). **Install iVerilog with add to PATH option selected.** If you forgot

to select the "add iVerilog to PATH" option you can manually add the bin folder of iVerilog to PATH.

6. (Optional) Using Anaconda Prompt install pytest with the following line:

   ```
   pip install pytest
   ```

   This will make the errors shown much more detailed, you can check exactly what it does at pytest site.

   Note: For pytest to actually do its job, your test-bench files must have the name format of test_*.py or *_test.py. For example, "adder_test" and "test_adder"

Execute the **cocotb-config** command in the Anaconda prompt to check if cocotb is installed correctly.

For more information please check the COCOTB for EE314 document on ODTUClass.

1.3 Creating and Managing Quartus Projects

To upload your designs to the FPGA, you will use SystemBuilder to create a project with proper pin assignments and module initialization. Section 4 of the DE1-SoC User manual explains how to use SystemBuilder to create projects. Note that you are encouraged **not** to use spaces or Turkish characters in your project name or directory.

To manage the Quartus projects, a little know-how in creating a project, adding files, compiling, and programming with the compiled board is required. To learn how to do this, you should read sections 5,6,8 of Quartus Prime Introduction Using Verilog Designs document on ODTUClass.

These are very light readings which should only take about 30 minutes of your time, you must read them.

1.4 Design Question: Simple Adder, Complementer, and Subtractor (100% Credits)

In this part, you will learn how to make a Verilog module that adds two 4-bit inputs and gives a 4-bit output **(We are not concerned with overflow for this experiment, but you should still know what overflow is.)**. As well as a module that outputs the 4-bit 2's complement of a 4-bit input. Then, you will combine these designs to create a 4-bit subtractor.

For all designs in this experiment you must only use assign statements for creating logic.

1.4.1 Creating the Project

1. Download SystemBuilder from ODTUClass course page if you haven't already.
2. Open SystemBuilder (Section 4 of the DE1-SoC User manual.).
3. Set your project name to EE314exp0.
4. You only need LEDx10 and Switchx10 for this experiment.
5. Generate the project.

Notice that the SystemBuilder also creates a Verilog file with the same name as the project, "EE314exp0" in this case. This file will be useful when uploading your designs to the FPGA boards as described in section 2.

1.4.2 Describing the Adder and Complementer with Verilog HDL

First, you will create two separate modules: an adder and a complementer. For this, you must use the supplied template files and only utilize **assign** statements for creating the logic.

The adder module should accept two 4-bit inputs "a" and "b", and output a 4-bit "sum" signal representing their sum.

The complementer module should take a 4-bit input "a", and output a 4-bit "compout" signal, which is the 2's complement of the input "a".

1. Add the shared template files "OurAdder.v" and "OurComplementer.v" to your project. You can do this by checking the Add file to current project option when opening a file in Quartus. Check **section 5.2** of the Quartus Prime Introduction Using Verilog Designs for more information.
2. Fill in the templates.
3. Generate RTL schematics for "OurAdder" and "OurComplementer" and add the screenshot of the RTL schematics to your report as described in subsubsection 1.4.4. **This part is graded.**
4. Test "OurAdder" and "OurComplementer" modules and add the screenshot of successfully passed tests to your report as described in subsubsection 1.4.5. **This part is graded.**

1.4.3 Combining the Modules on a Block Diagram

Now that you have developed both an adder and a complementer, we will integrate them to create a subtractor using Quartus's schematic editor. The subtractor will operate on two 4-bit inputs, `a` and `b`, and produce:

- A 4-bit output `subout` representing the subtraction `a - b`.
- A 4-bit output `compout` that displays the output from the complementer module.

Note: To implement this design, you must only use **one instance of the previously created adder and one instance of the complementer.**

1. For both of the modules, open their Verilog file and go to File → Create / Update and pick Create Symbol Files for Current File.
2. Add the shared template file "OurSubtractor.bdf" to your project then open it.
3. Open the Symbol Tool(Figure 2). And add the created symbols. They are under the Library named "Project".

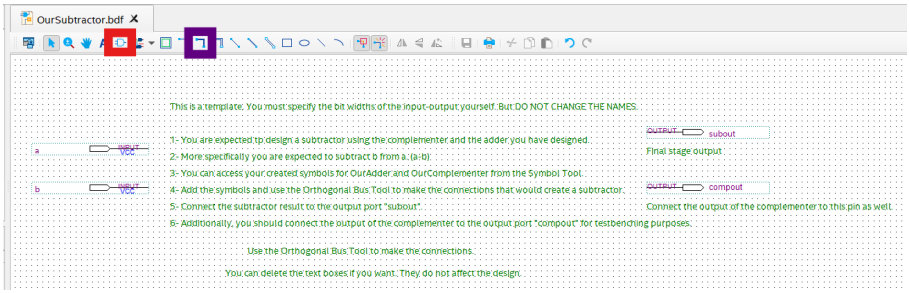


Figure 2: Quartus block diagram visual. The "Symbol Tool" is outlined in red. And the "Orthogonal Bus Tool" is outlined in purple.

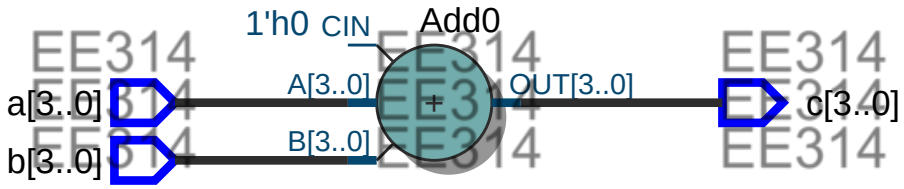
4. Fill in the template.
5. Go to File → Create / Update once again and pick Create HDL Design File for Current File.
6. Make sure that "Verilog HDL" is picked as the File type and click OK.
7. Change the name of the created file to "OurSubtractorVerilog.v".
8. Add the file to the project.
9. Open the file and change the Module name to "OurSubtractorVerilog".

10. Generate the RTL schematic for "OurSubtractorVerilog" as described in subsubsection 1.4.4. **This part is graded.**
11. Test "OurSubtractorVerilog" module as described in subsubsection 1.4.5. **This part is graded.**

1.4.4 Generating RTL Schematics (70% Credits)

Follow these steps for the Adder, Complementer, and Subtractor design:

1. Set the corresponding Verilog file as Top-Level Entity. You can do this by going to the Files tab of the Project Navigator, right-clicking on the desired file, and choosing Set as Top-Level Entity.
2. Start Analysis & Synthesis (Keyboard Shortcut Ctrl + K).
3. After compilation, go to the main menu and select Tools → RTL Viewer. Note: If the RTL Viewer option is disabled, check that the design is fully compiled and that you're in the correct Quartus project context.
4. You should observe a schematic of your synthesized circuits as connected logic elements. Below is for the adder:



5. From the RTL Viewer use File → Export to export your schematic as a .pdf file. Then, **You must add it to your report**. You can also convert it into .png or similar image formats to add.

Remember that one of the conditions of the eligibility for the experiment session is **getting at least 50% on the report.** The grade distribution for each RTL schematic is as follows:

- Correct Adder RTL view (20%)
- Correct Complementer RTL view (20%)
- Correct Subtractor RTL view (30%)

We will grade your schematics exported from the RTL viewer using the criteria below.

- Schematic should not have any unconnected inputs or outputs.
- Schematic should not be overly simplified by the synthesizer indicating a logical error (e.g., a huge design synthesizing into a simple logic element.).

1.4.5 Testing with Cocotb (30% Credits)

Remember that one of the conditions of the eligibility for the experiment session is **your Verilog codes must successfully pass at least half of the testbenches (2 out of 3).**

For this preliminary work, you will be given testbenches on ODTUClass for the 3 modules designed: OurAdder, OurComplemeneter, and OurSubtractorVerilog Verilog modules. Before running any testbenches, please make sure to properly install cocotb and all the required dependencies as explained in subsubsection 1.2.2.

Each given cocotb testbench folder will have the following structure:

```
Test_Project/
|-- HDL/        # Folder for your Verilog design files
|   |-- design.v  # Example Verilog design file
|-- Tests/      # Folder containing testbenches and Makefile
|   |-- test_design.py  # Cocotb testbench for your design
|   |-- Makefile   # Makefile to run simulations
```

HDL Folder: This folder is where you will place your Verilog design files (.v files). Make sure to use the template files that are given so that the testbenches can hook into your design.

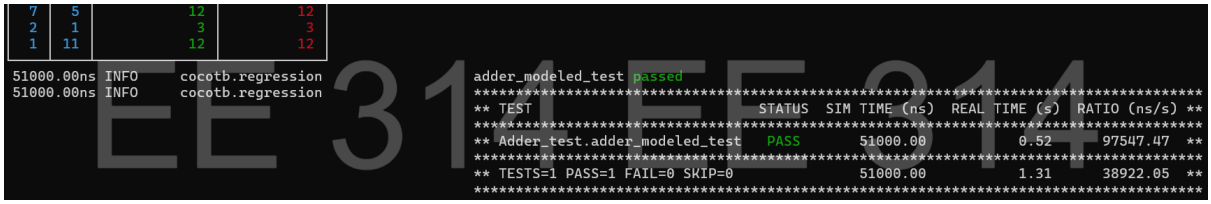
Tests Folder: This folder contains the testbenches written in Python using Cocotb, along with a Makefile to automate the simulation process.

To run the simulations, open the Anaconda prompt and navigate to the `Test_Project/Tests` folder using the `cd` command. Once inside the folder, type the following command:

make

Press Enter to start the simulation. For each test case, the testbench will display the provided test inputs, the expected output, and the actual output from your Verilog design, known as the Design Under Test (DUT). The DUT refers to the top-level Verilog module currently under evaluation. If a mismatch occurs, the test will terminate with a "Fail" status. In such cases, you can analyze the printed output to pinpoint errors in your design.

After running each testbench you must **include the screenshot of a successful pass of the testbench to your report.** Below is an example of a successful pass of the Adder testbench:



You will only get grades from a successful pass of a testbench, if the testbench fails you must fix the errors in your design. The grade distribution for each testbench for the report is as follows:

- Successful pass of the Adder testbench (10%)
- Successful pass of the Complementer testbench (10%)
- Successful pass of the Subtractor testbench (10%)

2 Experimental Work

For this experiment, there is no additional design in the experiment sessions. You will only show your "Subtractor Design" from the preliminary work to your laboratory instructor TA.

2.1 Demonstrate a Successful Testbench

Run the cocotb testbench for the "OurSubtractorVerilog" module once again and show it to your laboratory instructor.

2.2 Demonstrate the Design on the FPGA

1. Add the EE314exp0.v file which is in your project directory to your project. This file should already have been created by SystemBuilder as the top-level for uploading your designs.
2. Set this file as the Top-Level Entity.
3. Instantiate the "OurSubtractor" module inside this file. Connect the inputs "a" and "b" to SW[7:4] and SW[3:0] respectively. Connect the outputs "subout" and "compout" to LEDR[7:4] and LEDR[3:0] respectively.
4. Start Compilation (Keyboard Shortcut Ctrl + L).
5. Follow the steps on **section 8.2** of the Quartus Prime Introduction Using Verilog Designs document and upload your design into the FPGA.
6. Verify the operation of the design and demonstrate it to your lab instructor.

A Problems with running Cocotb

A.1 Path Issues

Cocotb does not like Turkish characters (as do most software libraries). If you have a Turkish character in your files or in your file path (either the test folder or the folder Icarus/Python is installed) you will get a "Unicode Error" when trying to run tests.

A similar error might be present if you have empty spaces " " in your folder paths.

If your Windows user name includes Turkish characters or spaces, it may cause this error. You can try to create a new Windows user and install Cocotb for that user.

A.2 "I give up" error

This is usually because one or more components of your environment are not set up correctly.

The best way to solve this is to go to ODTUClass or the cocotb website and reinstall everything by following the steps. We highly suggest using a version of Anaconda as it handles most of the environment for you.

Also, older Icarus versions can cause this error, so delete the old version and install the latest version (should be v12+).

A.3 Loss of the Colors

The console output of Cocotb may lose its color due to cached files within the test directories. To resolve this, either manually delete the __pycache__ and sim_build folders or run the command `make clean` to remove the cached files automatically.

B Parts List

`DE1-SoC Board`

C Changelog

v1.1: Changed the example adder RTL to make the output 4-bits