EE 271, Lab 1
An Introduction to Verilog and Digital Components

Lab Objectives

Read the whole lab first before starting on any work. The first lab for EE 271 will introduce you to the DE1-SoC Development board and Quartus Prime edition version 17. Both components are very important for all future labs so please pay attention and do not rush through this first lab.

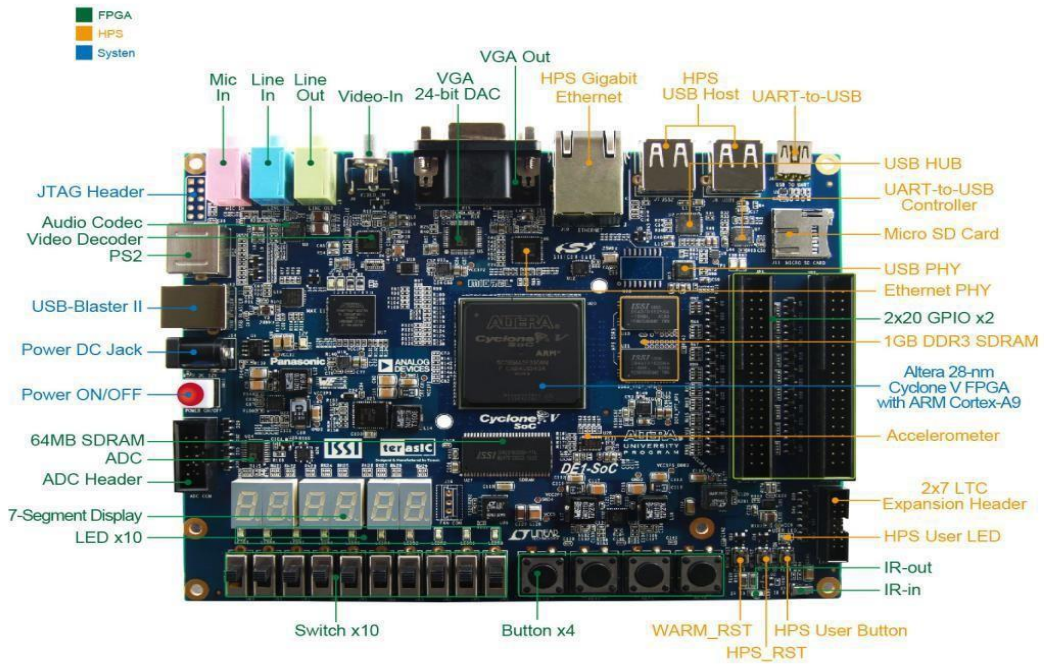
If you have any questions at any point please ask; you don't want to damage any of the expensive hardware that is provided to you.

Laboratory Design Kits

The design kits contain the resources you need to complete the labs. You are responsible for your lab kit and we expect that you will return the kit in good working order with all pieces intact. The kit includes:

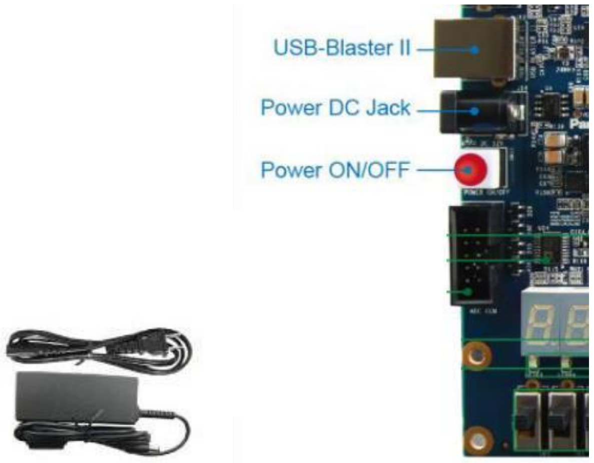
- The Terasic DE1-SoC Development Board (the "DE1") with UW Proto board attached on the top. The green Proto board has a white solder-less breadboard on it.
- A black power cord for powering the DE1.
- A grey USB cable for hooking the DE1 to a host computer.
- LED matrix board that may be used for the final project.
- TTL chips (in case needed for the final project)

Altera/Terasic DE1-SoC Development Board



The picture above is a diagram of the DE1 with most of the major components highlighted. Take note of the large FPGA (Field Programmable Gate Array) that is in the middle of the board. Think of it like a universal logic unit that all the devices can talk to.

The following picture shows how to connect the board to your computer.



You will need to use the black power cord to power the board. Plug the cord into an outlet and into the "Power DC Jack" socket just above the red on/off button. The grey USB cable is used to connect the "USB Blaster II" to the computer.

Using Quartus Software

The designs in this class will be done through the Quartus software. You can install the software for free by going to https://fpgasoftware.intel.com/?edition=lite , download the free web edition and install it. Note that you will have to register to be able to download the software.

First, make sure you Select Edition "Lite" and Select Release "17.0", then download the 5.8 GB tar file under the "Combined files" tab. Extract the tar file using a program like 7-zip and run the QuartusLiteSetup-17.0-windows.exe file. When it asks for the components to install, make sure you select each of these:

- Quartus Prime Lite Edition (Free)
- Devices: Cyclone V
- ModelSim: Intel FPGA Starter Edition (Free)

When the software is done, make sure to install the USB blaster driver. Run Quartus next, and if asked about licensing just run the software (we use the free version, so no license required).

Assigned Tasks

Task 1: Creating a schematic diagram and simulating a fulladder in Quartus

The objective of this task is to see how circuits are constructed at the hardware level using logic gates. However, instead of physically building a circuit on a breadboard and test its functionality, we will do this in Quartus. In this task you will follow two video tutorials on creating a schematic diagram for a full adder circuit and simulate it with timing diagrams. Please note that these tutorials were created using an older version of Quartus and so to adjust to version 17.0, you will need to follow the document called **"schematic-tutorial"** on canvas which includes the links to the video tutorials. Also note that this will be the only time we will use these tutorials and they are meant to replace what you would have done if you wired the full adder circuit and tested it in the lab. In this course we will be writing Verilog code to implement designs and will use ModelSim for simulation.

Task 2: Implementing and simulating the full adder using SystemVerilog and ModelSim

You should come to realize from task 1 that it is a tedious task to build circuits physically on breadboards especially as designs become more complicated. In this task we will do the same task we did in task1 but using Verilog and ModelSim.

For this task, you should follow along a series of video tutorials that will walk you through the steps of creating your first design using SystemVerilog and simulating the design in ModelSim. For your reference, the source code used in the tutorials is in the appendix of this document.

Please follow the tutorials in the following order:

1. Launch the Quartus Prime software.
2. Create a project from scratch. Please follow the steps in the following videos and use the same project name as in the video https://youtu.be/iLbmSTG7bpA
3. Implement the full adder using SystemVerilog and simulate it using ModelSim. Please follow the following video: https://youtu.be/BcvclrqZ2fc

   Please note that in the video when it refers to compiling the project for the first time, it may give you a compilation error. If you run the video for few more seconds it 'll tell you about setting the top level modules so that the program compiles.

   - After you successfully simulate the full adder, save a screenshot of the simulation. You can do that by going to File->Export->image and then save the image. You will need to include this image in your lab report as a proof that you went over the tutorials.
4. Mapping a SystemVerilog design to an FPGA. Please follow the steps in the following video and save an image of the simulation. https://youtu.be/mnZt2iNNfp4
5. Loading the design onto the FPGA. Please follow the steps in the following video and test your full adder on the DE1_SoC board and verify that it matches the truth table.

   https://youtu.be/uMxA3VcS3f8

After you are done with this task, close the current project before moving to the next task. You may quit Quartus at this point.

Task 3: Design a 4-bits adder

In the previous tasks, we created an adder that can add 3 bits. However, in reality, the numbers consist of multiple bits and therefore we want to extend our design accordingly. Your task is to use the "fullAdder" SystemVerilog module to extend the design to 4 bits numbers as shown in the following figure

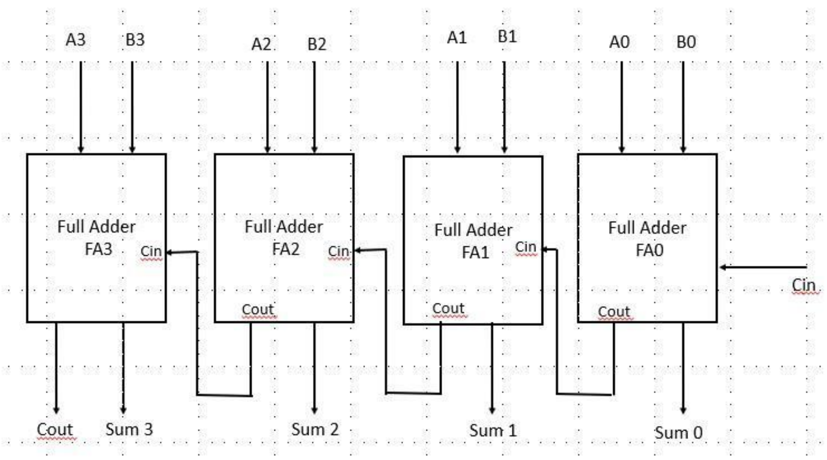
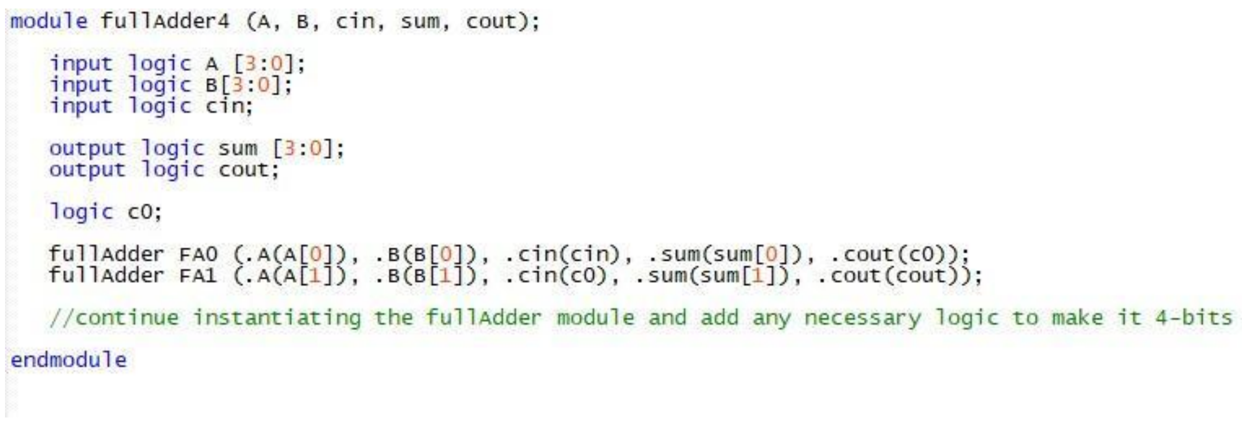


Figure 1. 4-bits Adder

To get started, you need to do the following:

1. Instead of creating a project from scratch, make a copy of the lab1 folder that was created in task 2 and call it "lab1a". Launch quartus by double clicking on DE1_Soc.qpf file under the lab1a folder. If you followed all the steps of task 2, you should have two .sv files: DE1_SoC.sv and fullAdder.sv



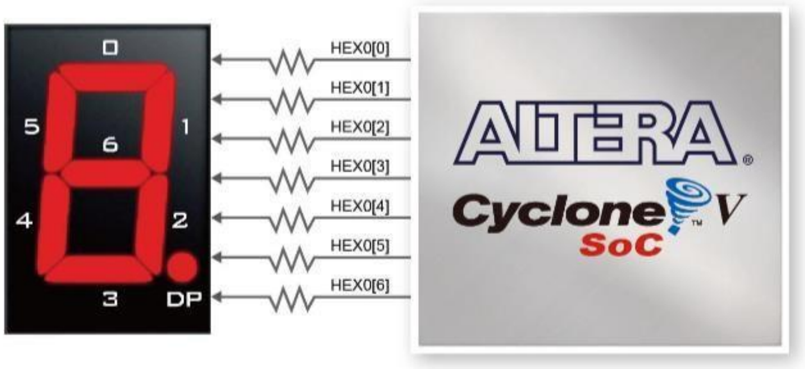
```
module fullAdder4 (A, B, cin, sum, cout);

   input logic A [3:0];
   input logic B[3:0];
   input logic cin;

   output logic sum [3:0];
   output logic cout;

   logic c0;

   fullAdder FA0 (.A(A[0]), .B(B[0]), .cin(cin), .sum(sum[0]), .cout(c0));
   fullAdder FA1 (.A(A[1]), .B(B[1]), .cin(c0), .sum(sum[1]), .cout(cout));

   //continue instantiating the fullAdder module and add any necessary logic to make it 4-bits

endmodule
```

2. Create a new SystemVerilog file and call it fullAdder4. This module will instantiate the fullAdder module created in task 2. To help you get started, here's a skeleton of fullAdder4 please modify it accordingly. Make the fullAdder4 module your top level entity and simulate it in ModelSim and save a picture of the simulation for your lab report.

3. Modify the DE1_SoC.sv such that you use 9 switches and 5 LEDs. SW0 is for Cin, SW4-SW1 for the 4-bits A and SW8-SW5 for B. The 5 LEDs should show Sum3-Sum0 and Cout. For better readability, show the output in the same order shown in Figure 1 from left to right (i.e Cout should be the left most LED and Sum0 is the right most LED).

4. Additionally, modify the assign statements for the HEX displays in the DE1_SoC.sv such that it displays the word "Adding". As an example, to make HEX5 shows the letter 'A', all the segments need to be turned on except segment 3 (HEX5[3]). Figure 2 is from the DE1_SoC datasheet and shows the 7-segments display connections. The segments are active low which means that a value of 0 is needed to turn on any segment.

Figure 2. HEX0 connections on the DE1_SoC board



Accordingly, to display the letter ‘A’ on HEX5, the following statement can be used

assign HEX5 = 7'b0001000; //displays 'A' where all segments except 3 are turned on.

5. Make DE1_SoC.sv your top level entity and simulate it in ModelSim and save a picture of the simulation for your lab report.
6. Load the program to the FPGA and test it.

Lab Demonstration and Submission Requirements

- Submit a video demonstrating the 4-bits adder. Your video is expected to be around 1 minute. In the video demonstrate the functionality on the board using the switches and LEDs and the 7-segment display.
- Submit a short lab report (about 3 pages and it’s ok if you go above 3 pages) that should include 3 main sections, detailed below.

  Procedure
  - Describe how you approached the problem and include any visuals (like block diagrams, truth tables, schematics, ..etc)

  Results
  - Include a screenshot of the waveforms you created for task1
  - Include screenshots of ModelSim results for task 3
  - Describe what you tested in the simulation, and what the results in the screenshot show
  - Give a brief overview of the finished project, compared to what was asked

  Appendix
  - Include screen shots for your code

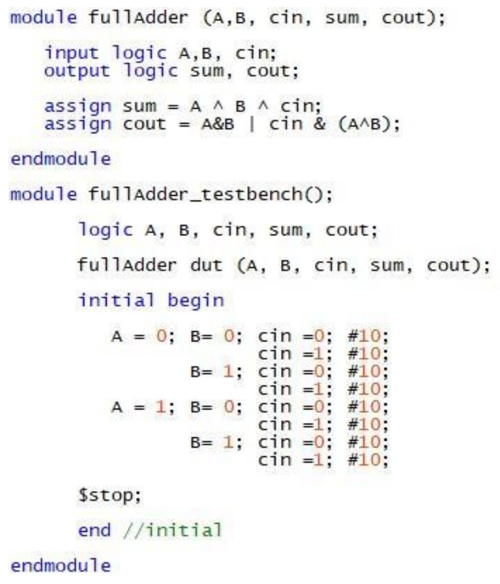
- **On Padlet**, write about a problem you had in the lab and the fix to it, share a tip or trick you learned while working on the lab. You can also share an aha moment that you discovered while working on the lab. Avoid duplicating comments made by your classmates. NO videos for this padlet task, please use textual comments. The link to the padlet is **here**
- Submit the SystemVerilog files (files with extension .sv) of task 3. Make sure to follow the commenting guide provided. **A significant amount of grade will depend on the commenting style.**
- Submit your report, video and programs to Canvas.

.

Appendix

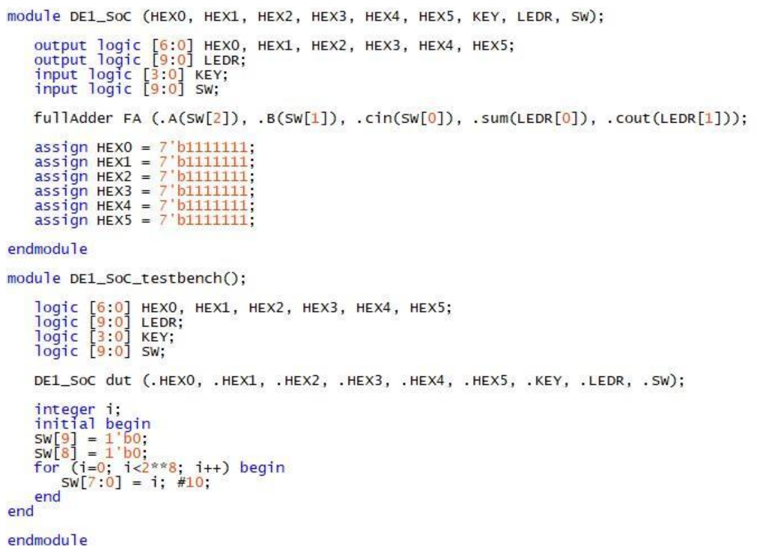
Source code used in the videos

1. FullAdder module



```
module fullAdder (A,B, cin, sum, cout);

   input logic A,B, cin;
   output logic sum, cout;

   assign sum = A ^ B ^ cin;
   assign cout = A&B | cin & (A^B);

endmodule

module fullAdder_testbench();

      logic A, B, cin, sum, cout;

      fullAdder dut (A, B, cin, sum, cout);

      initial begin

         A = 0; B= 0; cin =0; #10;
                      cin =1; #10;
                B= 1; cin =0; #10;
                      cin =1; #10;
         A = 1; B= 0; cin =0; #10;
                      cin =1; #10;
                B= 1; cin =0; #10;
                      cin =1; #10;

      $stop;

      end //initial

endmodule
```

2. DE1_Soc module



```
module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);

   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output logic [9:0] LEDR;
   input logic [3:0] KEY;
   input logic [9:0] SW;

   fullAdder FA (.A(SW[2]), .B(SW[1]), .cin(SW[0]), .sum(LEDR[0]), .cout(LEDR[1]));

   assign HEX0 = 7'b1111111;
   assign HEX1 = 7'b1111111;
   assign HEX2 = 7'b1111111;
   assign HEX3 = 7'b1111111;
   assign HEX4 = 7'b1111111;
   assign HEX5 = 7'b1111111;

endmodule

module DE1_SoC_testbench();

   logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   logic [9:0] LEDR;
   logic [3:0] KEY;
   logic [9:0] SW;

   DE1_SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .LEDR, .SW);

   integer i;
   initial begin
   SW[9] = 1'b0;
   SW[8] = 1'b0;
   for (i=0; i<2**8; i++) begin
      SW[7:0] = i; #10;
   end
end

endmodule
```