E85: Digital Design and Computer Engineering

Lab 2: FPGA Tools and Combinatorial Logic Design

Objective

The purpose of this lab is to learn to use Field Programmable Gate Array (FPGA) tools to simulate a SystemVerilog description of combinational logic, then synthesize it onto the FPGA and download it onto an FPGA board. The lab tutorial will walk you through a full adder and then you will design an instruction decoder circuit.

1. Tutorial: Altera FPGA Tools

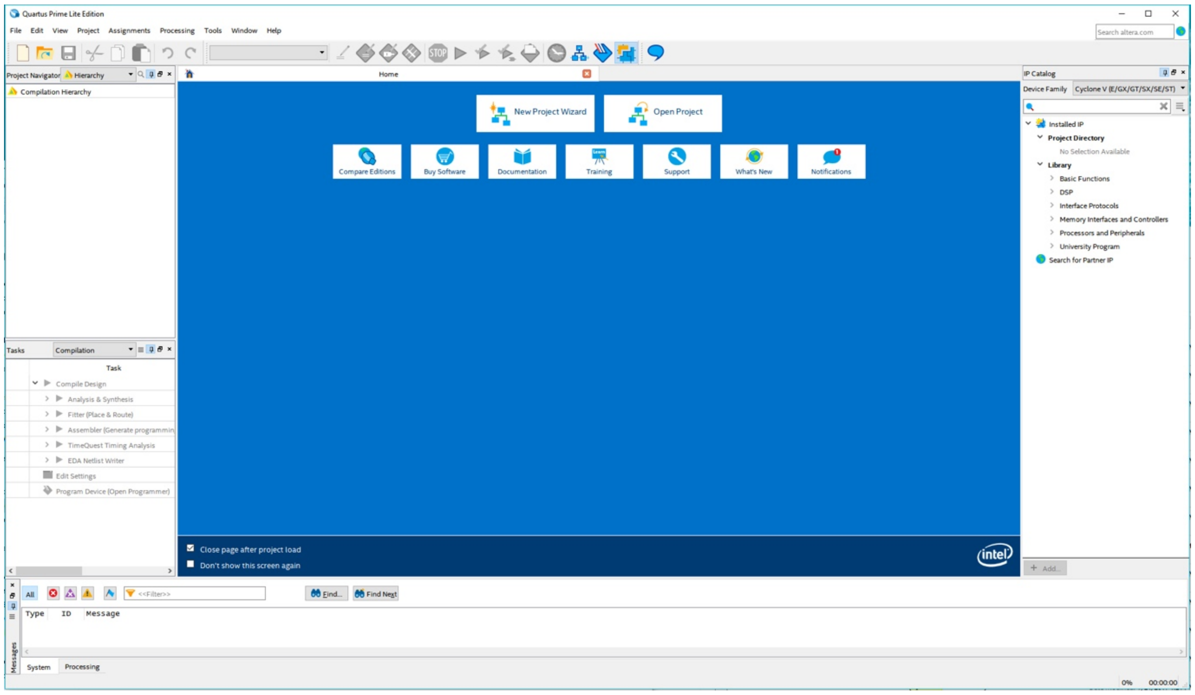
All of the FPGA labs in E85 will be using the Altera/Intel Quartus Prime FPGA software (Version 17 was current as of this writing) and the Altera DE0-CV evaluation board with the Cyclone V 5CEBA4F23C7N[1] chip. You can download and install the software on your own Windows PC to do parts of the labs from home, but will need to go to the E85 lab to use the DE0-CV boards.

In this tutorial, you will take the full adder that you designed in Lab 1, simulate it in ModelSim, and implement it on the DE0-CV board. You will hook up three switches for input and two LEDs for output and check that the circuit behaves correctly.

Make sure the DE0-CV board is powered on (with a wall adapter plugged into the DC 9V jack) and the USB Blaster (J9) port is plugged into the computer you are using. Press the red button to turn on power and confirm that the POWER LED is glowing blue.

Create a new folder in your Charlie directory for this tutorial, such as lab2_xx, where xx are your initials. Open Quartus Prime. It is found under the **Start** menu under **INTEL FPGA 17.xx… Lite Edition->Quartus**. You will be greeted with a getting started window. Click on the **New Project Wizard**.

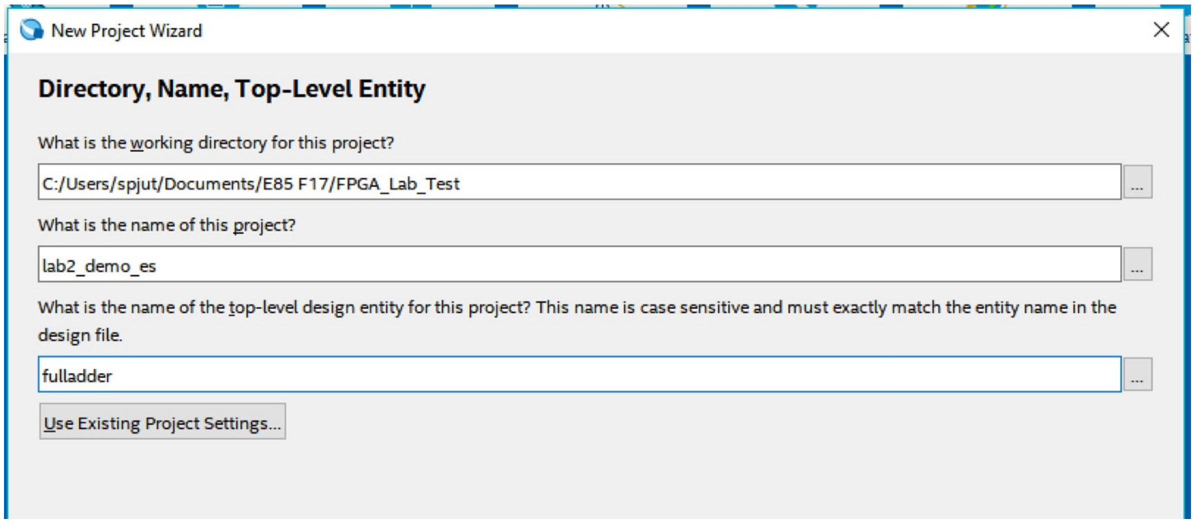
[1] 5C indicates the Cyclone V family of chips. The E indicates Enhanced logic/memory. The B indicates no hardware PCIe or memory controller. The A4 indicates the number of logic elements (49k, a medium-sized chip). F23 indicates that the chip is in a 484-pin ball grid array package. C indicates commercial temperature grade, and 7 is the medium speed grade for this chip. N indicates lead-free packaging (standard these days).



If the getting started screen is not present you can reach the same wizard by selecting **File-> New Project Wizard**.

If the Introduction screen appears, click the **Don't show me this introduction** again box and click on **Next**.

On the Directory, Name, Top-Level Entry screen, change the working location of the project to the folder you created, change the name of the project to something suitable such as lab2_XX. Set the top-level design entity to **fulladder**.


New Project Wizard

Directory, Name, Top-Level Entity

What is the working directory for this project?

C:/Users/spjut/Documents/E85 F17/FPGA_Lab_Test

What is the name of this project?

lab2_demo_es

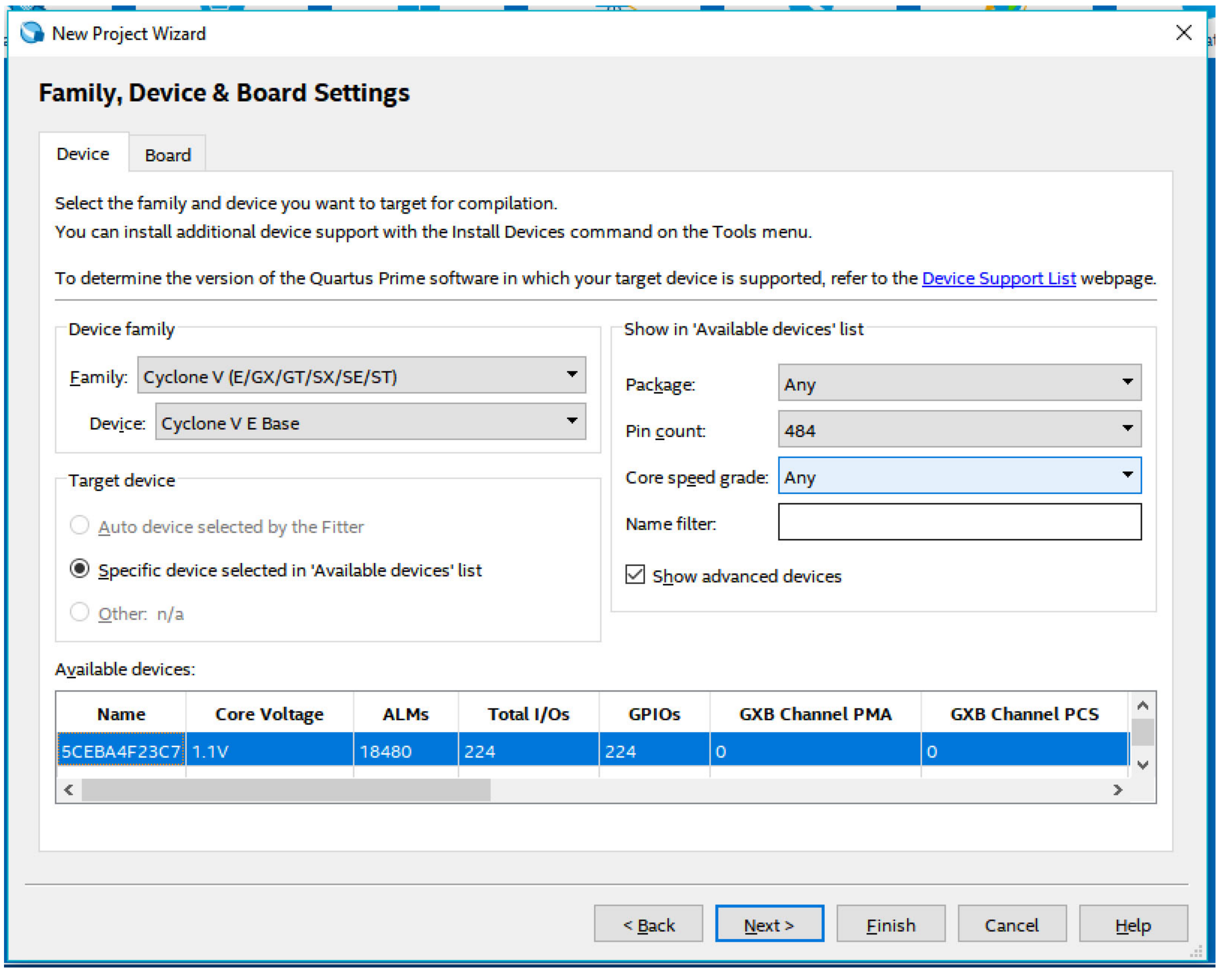
What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

fulladder

Use Existing Project Settings...

Click **Next**. On the Project Type screen, select **Empty project**, and click **Next**.

Click **Next** on the Add Files page as we have no files to add. The next page will set the specific FPGA we want the tool to target.


New Project Wizard

Family, Device & Board Settings

Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

To determine the version of the Quartus Prime software in which your target device is supported, refer to the Device Support List webpage.

Device family
Family: Cyclone V (E/GX/GT/SX/SE/ST)
Device: Cyclone V E Base

Target device
○ Auto device selected by the Fitter
● Specific device selected in 'Available devices' list
○ Other: n/a

Show in 'Available devices' list
Package: Any
Pin count: 484
Core speed grade: Any
Name filter:
☑ Show advanced devices

Available devices:

Name	Core Voltage	ALMs	Total I/Os	GPIOs	GXB Channel PMA	GXB Channel PCS
5CEBA4F23C7	1.1V	18480	224	224	0	0

Select **Family->Cyclone V**, then **Pin Count->484**, then **Device->Cyclone V E Base**; this will greatly reduce the choices. Click **5CEBA4F23C7** in available devices and click next. On the next page, change **Simulation** to **ModelSim-Altera** and the **Format** to **SystemVerilog HDL**, click next, then Finish.

For this tutorial we will create a full adder. Choose **File->New->SystemVerilog HDL**. Copy and paste the HDL for the full adder below into the file.

```
module fulladder(input  logic a, b, cin,
                 output logic sum, cout);

  logic ns, n1, n2, n3, n4;

  // sum logic
  xor x1(ns, a, b);
  xor x2(sum, ns, cin);

  // carry logic
  and a1(n1, a, b);
  and a2(n2, a, cin);
  and a3(n3, b, cin);
  or  o1(n4, n1, n2);
  or  o2(cout, n3, n4);
endmodule
```

Save your file as fulladder.sv in your lab2_xx directory.

1.1 Synthesis

Having completed the code we can now synthesize it into hardware. Quartus Prime calls this process compilation. Choose **Processing->Start Compilation** (or click the Start Compilation arrow next to the STOP icon). Watch for notes, warnings, and errors in the bottom panel. It is a good habit to learn which warnings are normal and to track down the root cause of abnormal warnings that can signal something awry that would otherwise take you hours to debug.

In addition to other warnings, you should get a critical warning that the pins have not been assigned. Now you will need to assign the proper pins so that the signals in your design connect to switches and LEDs on the board. Look at Chapter 3, starting on Page 24, of the DE0-CV User Manual for the FPGA pin numbers for each function on the board.

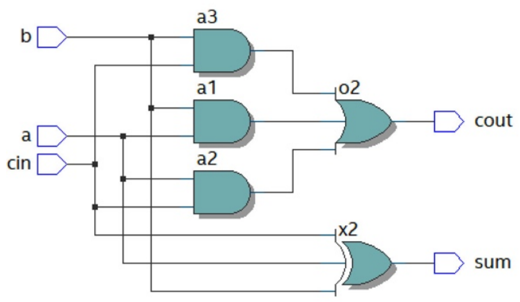
Now that synthesis has run, Quartus knows what signals are used by your top-level module, so you can assign them to pins. Let's assign inputs a, b, and c to SW0, SW1, and SW2, respectively. The manual shows that SW0 is PIN_U13 on your FPGA. Choose **Assignments –> Pin Planner** and set the *location* for input a to PIN_U13. Likewise, set b to PIN_V13 and c to PIN_T13. Hook sum to LEDR0 (PIN_AA2), and look up the pin assignment for LEDR1 for cout. Then close the Pin Planner and synthesize again. You should see two critical warnings that *Synopsys Design Constraints File file* not found because you have specified no timing requirements for your circuit, but the critical warnings about pin assignments should go away. The rest of the lab should operate properly even with the set of warnings that are left. The file QuartusFulladderCompOut.txt contains typical output at this stage. If you have *lots* of spare time, you can see how many of the remaining warnings you can make go away.

1.2 RTL Viewer

Now we will look at what the synthesizer created using the register transfer level (RTL) viewer.

Tools->Netlist viewer-> RTL Viewer

You should see the following circuit that matches your code.



1.3 Simulation

Next, we will simulate our circuit to make sure it performs the intended function. The best way to do a simulation is with a self-checking testbench written in System Verilog. The testbench applies inputs and checks that the outputs match expectation. If you find a mistake, you can correct the design and rerun the simulation to confirm. This reduces the tedium and risk of introducing errors running simulations and checking the results manually.

Create a new SystemVerilog file and paste the following code into it. Save it as testbench.sv in your lab2_xx directory. Observe that this code is a very different style of Verilog than you have previously seen; instead of implying physical hardware, it reads inputs called test vectors from a file, applies them, and checks the result. SystemVerilog is powerful in that it supports both hardware modeling and testbenches, but you will have to be careful not to use the kinds of programming language constructs of a testbench when you intend to imply hardware.

```
module testbench();
  logic        clk, reset;
  logic        a, b, cin, s, cout, sexpected, coutexpected;
  logic [31:0] vectornum, errors;
  logic [4:0]  testvectors[10000:0];

// instantiate device under test
// inputs are a, b, cin.
// outputs are s and cout
fulladder dut(a, b, cin, s, cout);

// generate clock
// Clock has period of 10 units.  High for 5, low for 5.
always
  begin
    clk=1; #5; clk=0; #5;
  end

// at start of test, load vectors
// and pulse reset for 2.2 clock cycles so it falls after a clock edge
// vectors are stored as 0s and 1s (binary) in fulladder.tv
// initialize count of number of vectors applied and errors detected
initial
  begin
    $readmemb("fulladder.tv", testvectors);
    vectornum = 0; errors = 0; reset = 1; #22; reset = 0;
  end

// apply test vectors on rising edge of clk
// take the current 5-bit test vector
// break it into 3 inputs and 2 expected outputs
// apply them 1 unit of time after rising edge of clock
// so they aren’t changing concurrently with the clock
always @(posedge clk)
  begin
    #1; {a, b, cin, coutexpected, sexpected} = testvectors[vectornum];
  end

// Check results on falling edge of clk
// Don’t do anything during reset
// Otherwise, check if outputs match expectations
// Use !== instead of != so that x or z are treated as mismatches
// If an error is detected, report inputs, outputs, expected outputs
// and increment the count of errors,
// In any event, increment the count of vectors,
// When the test vector becomes all x, we’ve know we’ve processed all
// the vectors that were initially loaded and the test is complete,
// Report the number of vectors applied and errors detected.
// Then stop the simulation.

always @(negedge clk)
  if (~reset) begin    // skip during reset
```

```
    if (s !== sexpected || cout !== coutexpected) begin // check result
      $display("Error: inputs = %b", {a, b, cin});
      $display(" outputs = %b %b (%b %b expected)",
        s, cout, sexpected, coutexpected);
      errors = errors + 1;
    end
    vectornum = vectornum + 1;
    if (testvectors[vectornum] === 5'bx) begin
      $display("%d tests completed with %d errors", vectornum,
errors);
      $stop;
    end
  end
endmodule
```

Create another file called fulladder.tv (easiest in a different text editor, not inside Quartus) and add the following lines. Each line has 5 bits corresponding to the three inputs and two expected outputs. Underscores in the test vector file are ignored, so the underscores are placed between the inputs and expected outputs to make them easier to read. For example, this test vector file indicates that a, b, and cin should be 0 on the first test, and that s (sum) and cout are expected to both be 0. On the second test, cin becomes 1 and the expected s becomes 1 as well.

```
// a b cin _ cout s
000_00
001_01
010_01
011_10
100_01
101_10
110_10
111_11
```

We will use ModelSim, a commercial hardware description language (HDL) simulator made by Mentor Graphics. You can download and install ModelSim either as part of the Quartus Prime installation or directly from Mentor Graphics on your computer if you wish. On the lab computers it is found under the **Start** menu under **INTEL FPGA 17.xx… Lite Edition->ModelSim**.

Choose **File -> New Project** and create a project named lab2_ms_xx in your Charlie directory. Click **Add Existing File** and browse to add your fulladder.sv and testbench.sv files. Choose **Compile-> Compile All**. You should see a message "2 compiles, 0 failed with no errors." If you do get errors, click on the red errors message to bring up the errors, and correct the bad file, then compile again.

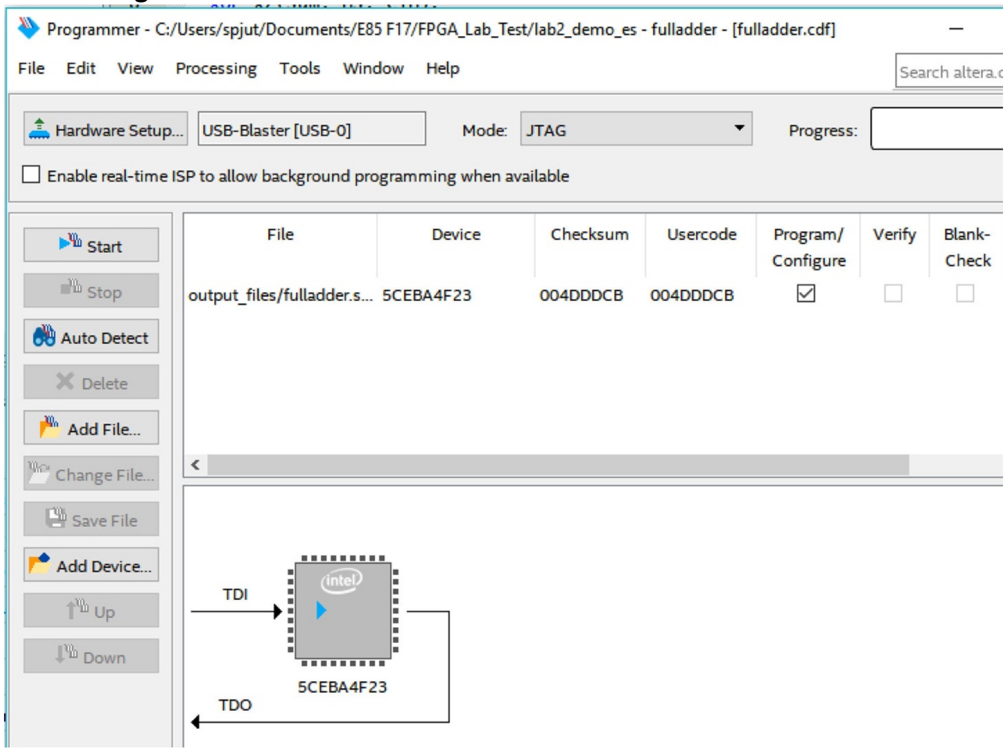
Choose **Simulate -> Start Simulation**… Expand the + symbol next to the work library, then click on your testbench module. Choose **OK** to simulate it. In the Objects pane, select all of the signals, then choose **Add -> To Wave -> Selected Signals** so that all of your inputs and outputs show up in a waveform viewer.

Type **run 200** in the Transcript pane to run the simulation for 200 time units. You should see a message "8 tests completed with 0 errors." If you see a warning that ModelSim can't find your fulladder.tv file, move it to the same directory that you chose for your ModelSim project (lab2_ms_xx). Then type **restart –f** in the Transcript pane to restart your simulation without losing all your waveforms and **run 200** to rerun. If you ever need to stop a runaway simulation, you can use the **Simulate -> Break** menu. If you make any changes to your code, be sure to choose Compile All again before rerunning, or you'll resimulate the old code.

1.4 Hardware Programming

Synthesis generates a bitfile indicating how each logic block and interconnection on the FPGA should be configured. We can now program the DE0-CV board with the bitfile to place your design on the chip.

Tools->Programmer


Programmer - C:/Users/spjut/Documents/E85 F17/FPGA_Lab_Test/lab2_demo_es - fulladder - [fulladder.cdf]

Hardware Setup... USB-Blaster [USB-0] Mode: JTAG Progress:

☐ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check
output_files/fulladder.s...	5CEBA4F23	004DDDCB	004DDDCB	☑	☐	☐

If programmer window does not say **USB-Blaster** next to **Hardware Setup**, then use the **Hardware Setup** button to set it to **USB-Blaster**. If, necessary (if the **5CEBA4F23** icon in the above image isn't visible), click **Add File…** and browse to the "output_files" folder of your project. Select the .SOF file. It may be pre-selected. Click the **Start** button. It should program the FPGA and run to 100% successful.

Now you can move the toggle switches **SW[2:0]** on the DE0-CV board and look at the red LEDs just above the switches. Check that your adder adds properly.

2. DE0-CV Board

If you like to know what is happing under the hood, you may wish to familiarize yourself with your Altera DE0-CV board, which has many nifty capabilities. Skim through the DE0-CV User Manual on the class website. The **DE0-CV_ControlPanel** on the desktop will let you test many of the features.

3. ALU Decoder

Now it is your turn to design a combinational logic circuit and build it on your FPGA board.

Table 7.3 from the textbook describes the function of a circuit with six inputs (ALUOp and Funct4:0) and four outputs (ALUControl1:0 and FlagW1:0). We will use this circuit in the second part of the semester to control an arithmetic/logic unit (ALU) in a microprocessor. For the purposes of this lab, you can assume that your circuit only has to correctly handle the inputs in the table, and that the output for all other cases are don't cares.

Write Boolean equations for the four outputs and sketch a schematic of a circuit that implements your equations. Write structural Verilog code implementing your schematic. Build a self-checking test-bench that applies all the interesting inputs and checks the output. Simulate your code in your testbench and check that it performs the function you intended; debug any discrepancies. Assign pins for your FPGA, using SW5 through SW0 to provide inputs and LEDR3 through LEDR0 to display the outputs. Synthesize your Verilog code and examine it in the RTL Viewer and check that it matches your expectations. Write a self-checking testbench with appropriate test vectors, simulate your design, and debug any errors. Download it onto the DE0-CV board and apply the inputs with the switches and check that the outputs match expectations.

Table 7.3 **ALU Decoder truth table**

ALUOp	*Funct*$_{4:1}$ (*cmd*)	*Funct*$_0$ (*S*)	Type	*ALUControl*$_{1:0}$	*FlagW*$_{1:0}$
0	X	X	Not DP	00 (Add)	00
1	0100	0	ADD	00 (Add)	00
		1			11
	0010	0	SUB	01 (Sub)	00
		1			11
	0000	0	AND	10 (And)	00
		1			10
	1100	0	ORR	11 (Or)	00
		1			10

What to Turn In

1. Please indicate how many hours you spent on this lab. This will be helpful for calibrating the workload for next time the course is taught.
2. Boolean equations for your ALU Decoder
3. Gate-level schematic of your ALU Decoder
4. Structural Verilog code for your ALU Decoder
5. RTL Viewer schematics of your synthesized ALU Decoder
6. Self-checking test bench for your ALU Decoder with a test vector file
7. Simulation waveforms showing the ALU Decoder simulation. Did it work correctly?
8. Did the ALU Decoder function correctly on the DE0-CV Board?

If you have suggestions for further improvements of this lab, you're welcome to include them at the end of your lab.