E85: Digital Design and Computer Architecture

Lab 2: FPGA Tools and Combinational 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 II FPGA software (specifically, version 13.0 SP1 Web Edition, the last version to support our Cyclone II chip), and the Altera DE2 evaluation board with the Cyclone II EP2C35F672C6N[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 DE2 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 DE2 board. You will hook up three switches for input and two LEDs for output and check that the circuit behaves correctly.

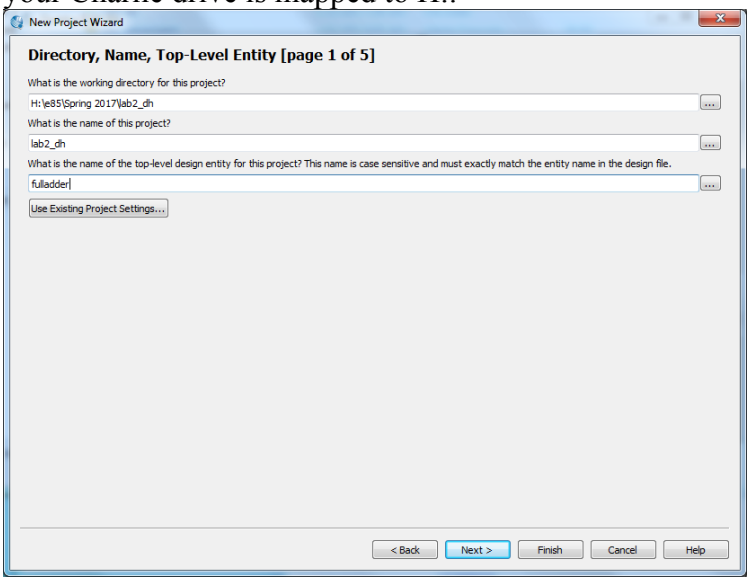
Make sure the DE2 board is powered on (with a wall adapter plugged into the DC 9V jack) and the USB Blaster (J9) port is plugged into the computer you are using. Press the red button to turn on power and confirm that the POWER LED is glowing blue.

Create a new folder in your Charlie directory for this tutorial, such as lab2_xx, where xx are your initials. Open Quartus II 13.0.1.232 Web Edition 64-bit. You will be greeted with a getting started window. Click Create a New Project.

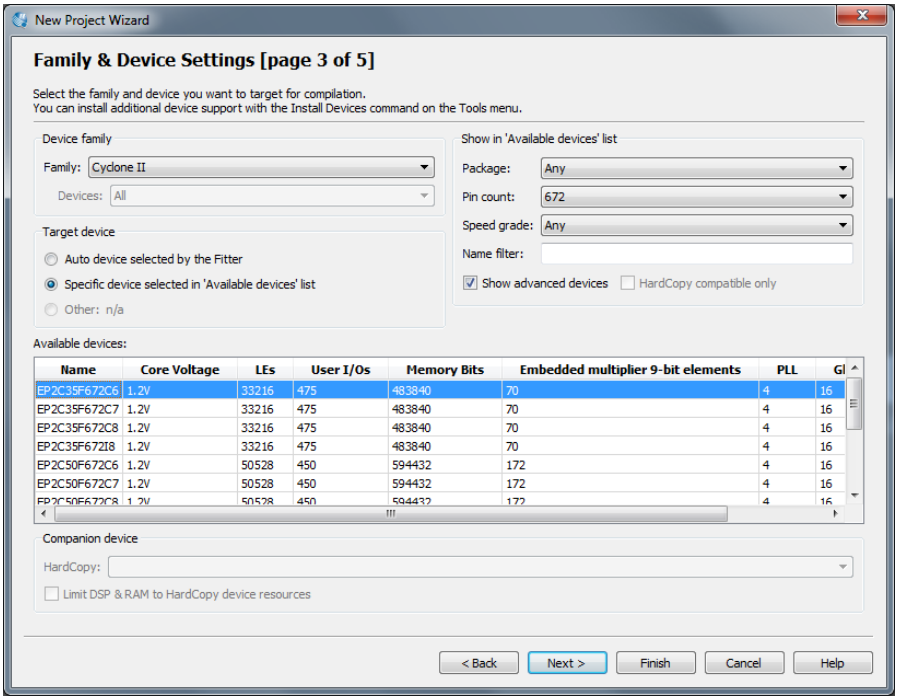
[1] EP2C indicates the Cyclone II family of chips. The 35 indicates the number of logic array blocks (2076, a medium-sized chip). F672 indicates that the chip is in a 672-pin ball grid array package. C indicates commercial temperature grade, and 6 is the slowest speed grade for this chip.

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

Change the working location of the project to the folder you created, change name of the project to something suitable as show below. Set the top-level design entity to **fulladder**. This assumes your Charlie drive is mapped to H:.



Click next, click next through the Add Files page [2 of 5] as we have no files to add
The next page will set the specific FPGA we want the tool to target.



Select Family->Cyclone II, then Pin Count 672; this will greatly reduce the choices. Click EP2C35F672C6 in available devices and click next. On page 4 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 II calls this compilation. Choose Processing->Start Compilation. This is also available in the panel in the bottom left of the main window. 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.

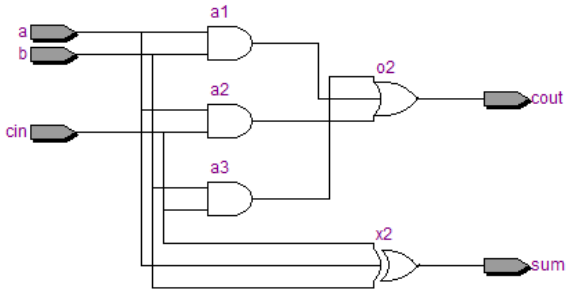
You should get five critical warnings because pins have not been assigned. Now you will need to assign the proper pins so that the signals in your design connect to switches and LEDs on the board. Look at the following file that defines the FPGA pin numbers for each function on the board.

\\charlie.hmc.edu\Courses\Engineering\E85\Labs\DE2_pin_assignments.csv

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 SW[0], [1], and [2], respectively. The pin assignment file shows that SW[0] is PIN_N25 on your FPGA. Choose Assignments –> Pin Planner and set the location for input a to PIN_N25. Likewise, set b to PIN_N26 and c to PIN_P25. Hook s to LEDG[0] (PIN_AE22), and look up the pin assignment for LEDG[1] for cout. Then close the Pin Planner and synthesize again. You should see two critical warnings that Synopsys Design Constraints are not assigned because you have specified no timing requirements for your circuit, but the other critical warnings should 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. 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
fulladder dut(a, b, cin, s, cout);

// generate clock
always
  begin
    clk=1; #5; clk=0; #5;
  end

// at start of test, load vectors
// and pulse reset
initial
  begin
    $readmemb("fulladder.tv", testvectors);
    vectornum = 0; errors = 0; reset = 1; #27; reset = 0;
  end

// apply test vectors on rising edge of clk
always @(posedge clk)
  begin
    #1; {a, b, cin, coutexpected, sexpected} = testvectors[vectornum];
  end

// check results on falling edge of clk
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 and add the following lines:

```
// a b c _ 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 PE Student Edition (for from the web to your own computer if you wish, or use the version already installed in the E85 lab.

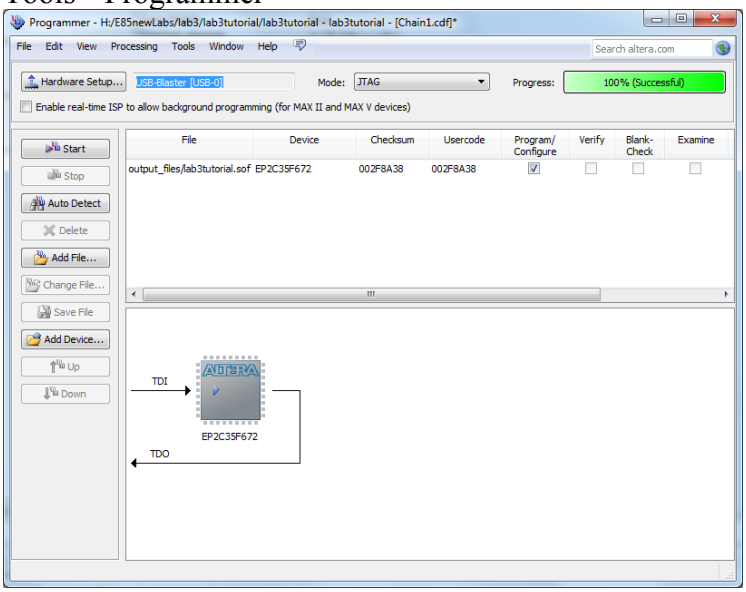
Choose File -> New Project and create a project named lab2_xx in your Charlie directory. Click Add Existing File and browse to add your lab2_xx.sv and testbench.sv files. Choose Compile-> Compile All. You should see a message "2 compiles, 0 failed with no errors." If you do get errors, click on the red 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. Uncheck Enable optimization so that ModelSim does not optimize away important signals. 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. 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.

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 DE2 board with the bitfile to place your design on the chip.
Tools->Programmer



If it does not say USB-Blaster next to Hardware Setup, then use the button to set it to USB-Blaster. Click Add File… and browse to the "output_files" folder of your project. Select the .SOF file. 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 DE2 board and look at the green LEDs. Check that your adder adds properly.

2. DE2 Board

If you like to know what is happing under the hood, you may wish to familiarize yourself with your Altera DE2 board, which has many nifty capabilities. Skim through the DE2 User Manual on the class website.

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 LEDG3 through LEDG0 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 DE2 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 DE2 Board?

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