EE 231 Lab 1

Introduction to Verilog HDL and Quartus

In this lab you will design simple circuits by programming the field-programmable gate array (FPGA). At the end of the lab you should be able to understand the process of programming a programmable logic device (PLD), and the advantages of such an approach over using discrete components.

1. Lab

1.1. Write a gate level Verilog program to implement the half adder circuit shown in Figure 1.

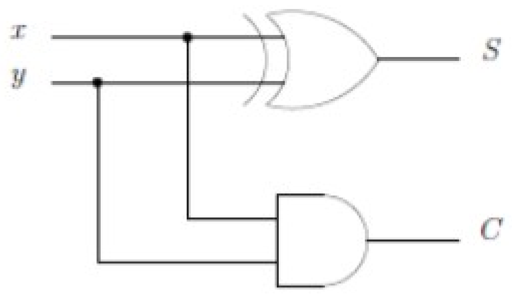


Figure 1: Half Adder Circuit

1.2. Assign your two inputs to the two push buttons and your two outputs to two of the LEDs. **Before you continue, ask the TA to check your assignment.**

1.3. Perform both functional and timing simulations. The simulations can also be done with the Logic Analyzer. Print your waveforms. **Are the two waveforms the same? Discuss.**

1.4. Write a Verilog program to implement the same half adder circuit using dataflow modeling and simulate your circuit again. **Note: you don't have to respecify your waveforms as you have already saved them in a file.**

1.5. Program your PLD (be sure to have the correct device selected) to implement your circuit and verify that it works. **Ask the TA to initial your lab book once you have it working.**

1.6. Use vectors to describe your inputs and outputs. Send signals to all of the LEDs to have them off when you send them a logic 0 and turn on when you send them a logic 1, and similarly, the input to be logic 1 into your circuit when you press the button and zero otherwise.

2. Supplementary Material

2.1. Introduction to Verilog

2.1.1. In order to write a Verilog HDL description of any circuit you will need to write a module. A **module** is the fundamental descriptive unit in Verilog and it is a set of text describing your circuit and is enclosed by the key words **module** and **endmodule**.

2.1.2. As the program describes a physical circuit you will need to specify the inputs, the outputs, the behavior of the circuit, and how the gates are wired. To accomplish this, you need the keywords input, output, and wire to define the inputs, outputs, and the wiring between the gates, respectively.

2.1.3. There are multiple ways to model a circuit:
- Gate level modeling
- Dataflow modeling
- Behavioral modeling
- Or a combination of the above

2.1.4. A simple program modeling the circuit in Figure 2 at the gate level is shown below.

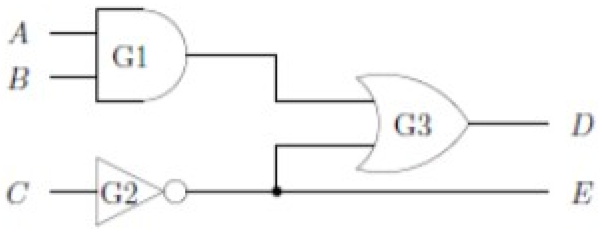


Figure 2: Simple Circuit

```
module simple_circuit(output D,E, input A,B,C);
    wire w1; //Creating a virtual wire
    AND G1(w1,A,B); //Creates AND gate with output w1 and inputs A and B
    NOT G2(E,C); //Creates NOT gate output E and input C
    OR  G3(D,w1,E); //Creates OR gate with output D and inputs w1 and E
endmodule
```

Program 1: Simple Program in Verilog Modeling a Circuit at the Gate Level

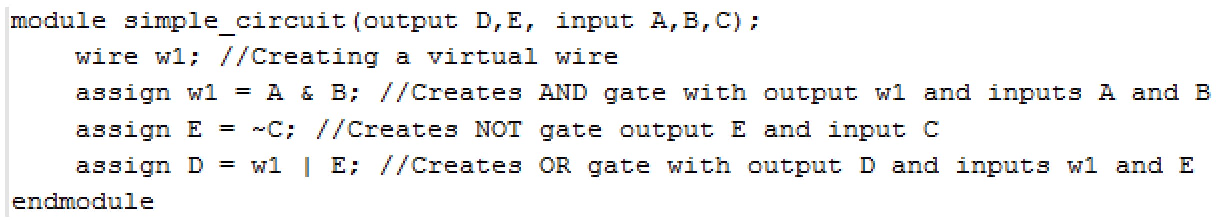
2.1.5. As seen above, the outputs come first in the port list followed by the inputs.

2.1.6. Single line comments begin with //

2.1.7. Multi-line comments are enclosed by /*...*/

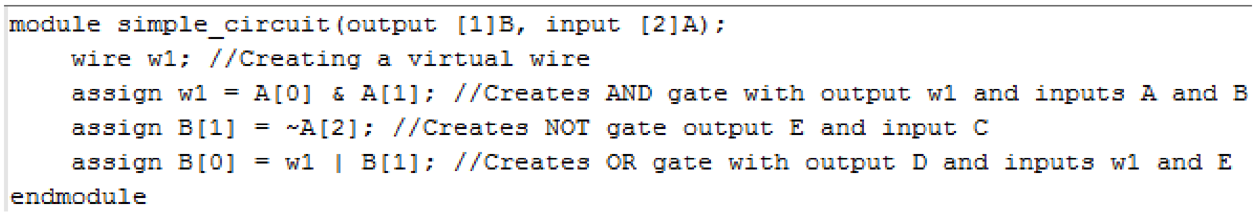
2.1.8. Verilog is case sensitive.

2.1.9. A simple program modeling a circuit using dataflow and vectors are provided below.



```
module simple_circuit(output D,E, input A,B,C);
    wire w1; //Creating a virtual wire
    assign w1 = A & B; //Creates AND gate with output w1 and inputs A and B
    assign E = ~C; //Creates NOT gate output E and input C
    assign D = w1 | E; //Creates OR gate with output D and inputs w1 and E
endmodule
```

Program 2: Simple Program in Verilog Modeling a Circuit using Data Flow



```
module simple_circuit(output [1]B, input [2]A);
    wire w1; //Creating a virtual wire
    assign w1 = A[0] & A[1]; //Creates AND gate with output w1 and inputs A and B
    assign B[1] = ~A[2]; //Creates NOT gate output E and input C
    assign B[0] = w1 | B[1]; //Creates OR gate with output D and inputs w1 and E
endmodule
```

Program 3: Simple Program in Verilog Modeling a Circuit using Data Flow with Vectors

2.1.10. You can use identities describing multiple bits known as vectors.

2.1.11. Given an identifier [7:0]X you can assign values by:

assign [7:0]X = 8’b00001111;

Where the 8’b specifies that we are defining an 8 bit binary number and 00001111 is that 8 bit binary number. You can also assign parts of the number as:

assign [2:0]X = 3’b101;

This assigns only the last three bits of X.

2.2. Using Quartus

In order to implement the circuits that you will design on the FPGA, there are a few key steps.

2.2.1. Starting a new Project

1. Click “New Project Wizard.”

2. Set the directory name. **Make sure that you create a new folder/directory for each project and do not copy the directory over.**

3. Set the name of the project. Naming each project after the number of the lab is a good idea, e.g., Lab 1.

4. Click “Yes” to create a directory if it does not exist.

5. You can add existing files if you already have them, otherwise select "Next."
6. Next you need to specify the device that you are using. The device family we are using is the Cyclone IV E and the device is the **EP4CE22F17C6**
7. Click Next.
8. Click Finish.

2.2.2. Writing the Code

1. Select File > New or Ctrl + N.
2. Choose Verilog HDL File.
3. Click Ok.
4. Select File > Save As. **NOTE: The name of the file should be the name of the Project. Also, the name of the main module needs to be the same name as the file.**
5. Choose Save as type, and select Verilog HDL File.
6. Put a check mark in the box "Add file" to current project. Unless the file is part of the project you won't be able to proceed. If you do not add the file now you can always add it later by selecting Project > Add/Remove Files to Project.
7. Click Save.

Now you are ready to type your program.

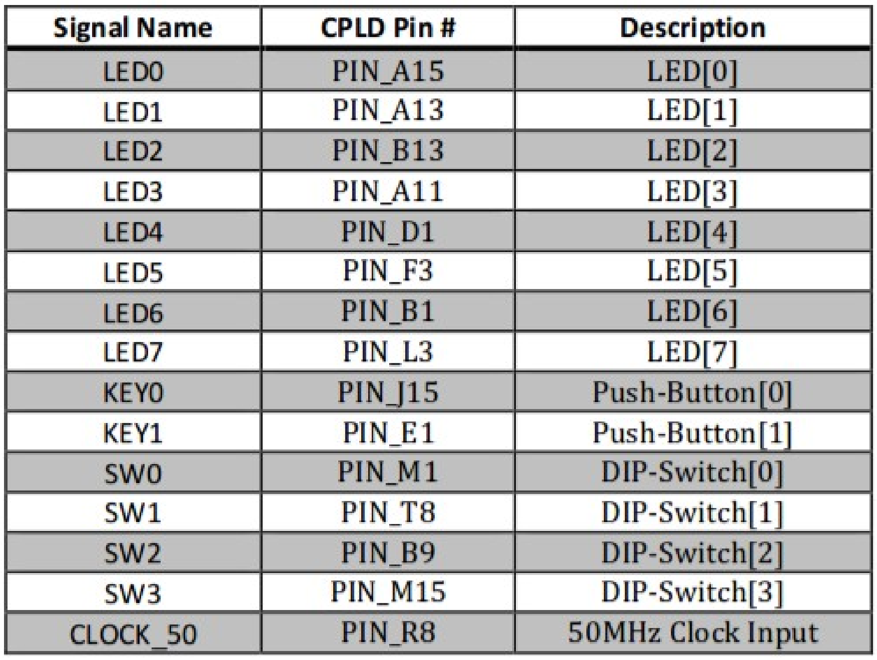
2.2.3. Compiling

1. To compile your code, you can either go to Processing > Start Compilation, click the play icon on the toolbar, or use Ctrl + L.
2. You can click on Processing > Compilation Report to see the results of the compilation.
3. If there are no errors, then your program's syntax is correct. This does not mean that your program will do what you want because you may have some logic errors. **Note: Warnings will not stop your code from compiling but may contain helpful information for diagnosing logical issues.**

2.2.4. Pin Assignment

Assigning the pins on the FPGA allows the user to interface with the hardware and confirm that their code works as intended. Some pins have already been internally wired to the LEDs and push buttons on the board. A list of those pins is provided in Table 1. **Be sure the selected device is the Cyclone IV E: EP4CE22F17C6.**

1. Go to Assignments > Pin Planner or Ctrl + Shift + N.
2. Under Filter select Pins: all.
3. After successful compilation your input/outputs should already be listed but not assigned. If you are missing a node, double click <<new node>>; a drop-down menu should appear to add an input/output. Select the input/output you want to assign.
4. In the column labeled Location, select the pin you want to assign to that input/output node. Table 1 shows the locations of the hardwired pins for your evaluation board. For example: To connect output C to the LED7, you would select Location Pin L3 for Node C.
5. Repeat Steps 3 and 4 until all inputs/outputs are assigned for your circuit.
6. The Altera default is that all unused pins should be assigned "As outputs driving ground." This is a good choice for pins not connected to anything (it reduces power and noise), but is not good for pins that may be connected to a clock input – you would then have both the clock and the Altera chip trying to drive this input. **A safer choice is to define all unused pins "As input tri-stated with weak pull up resistor."**
   a. Go to Assignments > Device.
   b. Click on Device and Pin Options.
   c. Select the Unused Pins tab.
   d. From the Reserve all unused pins drop-down menu, select "As input tri-stated with weak pull-up resistor."



Signal Name	CPLD Pin #	Description
LED0	PIN_A15	LED[0]
LED1	PIN_A13	LED[1]
LED2	PIN_B13	LED[2]
LED3	PIN_A11	LED[3]
LED4	PIN_D1	LED[4]
LED5	PIN_F3	LED[5]
LED6	PIN_B1	LED[6]
LED7	PIN_L3	LED[7]
KEY0	PIN_J15	Push-Button[0]
KEY1	PIN_E1	Push-Button[1]
SW0	PIN_M1	DIP-Switch[0]
SW1	PIN_T8	DIP-Switch[1]
SW2	PIN_B9	DIP-Switch[2]
SW3	PIN_M15	DIP-Switch[3]
CLOCK_50	PIN_R8	50MHz Clock Input

Table 1: Pin Assignments for the LEDs, Push Buttons, DIP-Switches, and Clock inputs

2.2.5. Simulating the Designed Circuit

Once you have assigned places for the inputs and the outputs as well as values for the inputs, you can simulate the circuit. The simulation of the circuit can be done in two modes: Functional and Timing. The Functional Simulation checks that the logic for your function is working correctly. The Timing Simulation runs the simulation with the delay in the gates included.

In simulating your circuit there are four main steps:

1. Create a waveform file.
2. Select your inputs and outputs.
3. Create a waveform for each input.
4. Run the simulation to generate the output for verification with your expected results.

A more detailed explanation of the above steps is described below.

1. Select File > New.
2. Click on University Program VWF.
3. Click Ok.

4. Save the file using some meaningful name; the name of the lab is a good idea, i.e. Lab0.vwf.

5. Right-click below Name and select Insert Node or Bus.

6. Click on Node Finder and click List. This should bring all the outputs and inputs onto the Nodes Found list.

7. Use >> to move all the nodes to the Selected Nodes Side.

8. Click Ok and Ok again.

9. Set the desired simulation time by selecting Edit > End Time.

10. Select View > Fit in Window.

11. Select the inputs and outputs you want to observe by clicking Edit > Insert > Insert Node or Bus.

12. Click on Node Finder.

13. Click on the input/output you want to observe by clicking on the > sign. Repeat this process for all your inputs/outputs.

14. The next step is to specify the logic value of each of the inputs you have selected and duration of that value by right clicking on the input and selecting Value > *your selected value*

15. Save the file, e.g. lab0.vwf.

16. After the waveforms have been defined, we can simulate our circuit. First perform the functional simulation, and then perform the timing.
    a. Click Start Functional Simulation or Start Timing Simulation at the top of the window.

2.2.6. Programing the FPGA

The final step is to program the FPGA with the circuit you designed. Be sure the correct device is selected before continuing.

1. Select Tools > Programmer.

2. Select JTAG in the Mode box.

3. If USB Blaster is not chosen in the box next to Hardware Setup, select by clicking on the Hardware Setup. Then click on USB-Blaster, the Close.

4. You should see a file listed with extension .sof, if not add it from Output.

5. Finally, press Start. The program will download on your board and once it is finished you can test your circuit in hardware.

Example of Bad Code

```
module Bad          Code(ABCD,EF);
    wire1; Define Wire
    assign w1=C //Assign W1 to input C;
    asign D = A & B; //Assign D as OR gate with C, A
    aign E = B | C; / XOR gate
    assign F = A ^ 1w; //AND gate
end modules
```

The first line of code can either be:

Module BadCode(input A, B, C, output D, E, F) or
Module BadCode(A, B, C, D, E, F)
    Input A, B, C;
    Output D, E, F;

The second line and third lines are necessary if you use define the variables inside the module instead of in the module declaration.

Wires should be defined as:

Wire w1;

Whenever a variable is being given a value, `assign` is in front of the variable being assigned followed by the logic done.

Assign w1 = C; //Assign w1 as C;
Assign D = A & B; //Assign D as AND gate with A, B
Assign E = C | B; //Assign E as OR gate with B, C
Assign F = A ^ w1; //Assign F as XOR gate with A, w1

The module should be closed with *endmodule*

The proper way to write the code above is this:

module BadCode(input A, B, C, output D, E, F);
    wire w1; //Define Wire
    assign w1 = C; //Assign W1 to input C
    assign D = A & B; //Assign D as AND gate with A, B

```
	assign E = B | C; //Assign E as OR gate with B, C
	assign F = A ^ w1; //Assign F as XOR gate with A, w1
endmodule
```