Lab 2: Introduction to Verilog HDL and Quartus

September 8, 2009

In the previous lab you designed simple circuits using discrete chips. In this lab you will do the same but by programming the CPLD. At the end of the lab you should be able to understand the process of programming a PLD, and the advantages of such an approach over using discrete components.

Contents

1 Prelab

1. Read the material provided in the supplementary section. 3
2. Using the schematics of the board, if the CPLD sends a logic 1 to one of the LEDs, would it turn on or off?

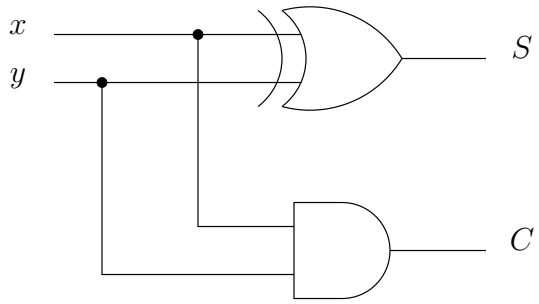

Figure 1: Half Adder

2 Lab

1. Write a gate-level Verilog program to implement the half adder circuit.
2. Assign your two inputs to two of the push buttons and your two outputs to two of the LEDs. **Before you continue, ask the TA to check your assignment**.
3. Simulate your designed circuit, perform both functional and timing simulations. Print your waveforms. Are the two waveforms the same? Discuss.
4. Write a Verilog program to implement the same half adder circuit using dataflow modeling and resimulate your cicuit. *Note: you don't have to respecify your waveforms as you have already saved them in a file.*
5. Program your CPLD to implement your circuit and verify that it works. **Ask the TA to initial your labbook once you have it working**.
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 cicuit when to press the button and zero otherwise.

3 Supplementary Material

3.1 Introduction to Verilog

1. In order to write an Verilog HDL description of any circuit you will need to write a *module* which is the fundamental descriptive unit in Verilog. A *module* is a set of text describing your circuit and is enclosed by the keywords **module** and **endmodule**.
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.
3. There are multiple ways to model a circuit
   - gate-level modeling,
   - dataflow modeling,
   - behavioral modeling,
   - or a combination of the above.
4. A simple program modeling a circuit (see Figure 2) at the gate-level, is provided below.

Program 1 Simple program in Verilog modeling a circuit at the gate-level

```
module simple_circuit(output D,E, input A,B,C);
wire    w1;

and     G1(w1,A,B);
not     G2(E,C);
or      G3(D,w1,E);
endmodule           // no semi-colon
```

5. As seen above the outputs come first in the port list followed by the inputs.
6. Single line comments begin with //

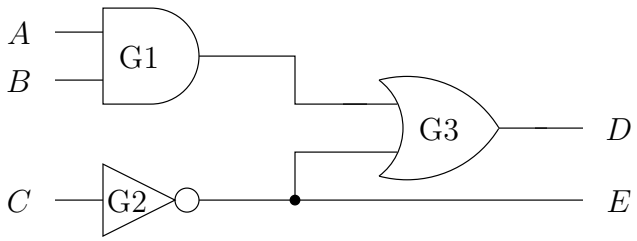

Figure 2: Simple Circuit

7. Multi-line comments are enclosed by /* */
8. Verilog is case sensitive.
9. A simple program modeling a circuit using dataflow, is provided below.

Program 2 Simple program in Verilog modeling a circuit using dataflow

```
module simple_circuit(output D,E, input A,B,C);

assign  D = (A & B) | ~C;
assign  E = ~C;

endmodule
```

.

10. You can identifiers describing multiple bits known as *vectors*. For example you may write Program 2 as

Program 3 Simple program in Verilog modeling a circuit using dataflow using vectors

```
module simple_circuit(output [1:0] Y, input [0:2] X);

assign  Y[1] = (X[0] & X[1]) | ~X[2];
assign  Y[0] = ~X[2];

endmodule
```

.

In this example, we have the input as three bits representing `A, B, C` and we have denoted them as `[0:2] X` which means we have three bits with the index 0 representing the MSB. We could have specified it as `[2:0] X` in which case the index 2 represents the MSB.

11. Given an identifier `[7:0] X` you can assign it values by

    `assign [7:0] X = 8'b00101011;`

    where the `8'b` specifies that we are defining an 8-bit binary number and `00101011` is that 8-bit binary number. You can also assign parts of the number as

    `assign [2:0] X = 3'b101;`

    which assigns only the last three bits of `X`.

3.2 Using Quartus

To implement the circuits that you will design on the CPLD there are few key steps.

1. Write your program using Verilog HDL.
2. Compile your code.
3. Correct any syntax errors.
4. Simulate your circuit to make sure that you are getting the behavior you expect.
5. Download your program onto the CPLD.
6. Test the operation of circuit.

Quartus helps you implement all of the above easily. The following sections describe how to do those basics steps.

3.2.1 Start New Project

1. Select `File > New Project Wizard`.
2. Set the directory name. You may want to have a directory with the name `ee231` where you save all your projects for this class. **Make sure that you create a new project for each project and do not just copy the director over.**
3. Set the name of the project. It will be simple if you name it by the lab name, e.g., `lab1`.
4. Click `Yes` to create the directory if it does not exist.
5. You can add existing files if you have already them, otherwise select `Next`.
6. Next you need to specify the device that you are using. Set the Device Family to `MAX II` and select `EPM2210F324C3`.
7. Press `Next`.
8. Press `Finish`.

3.2.2 Writing the Code

1. Select `File > New`.
2. Choose `Verilog HDL File`.
3. Click `Ok`.
4. Select `File > Save As`. *For one file project name, the name of the file should be the same as the project. In addition, the module name should be the same as the filename*
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 don't add the file now you cal latter added by selecting** `Project > Add/Remove Files in Project`.

7. Click `Save`.

8. Now you are ready to type in your program.

3.2.3 Compiling

1. Select `Processing > Start Compilation`, or click on the play icon on the toolbar.

2. You can click on `Processing > Compilation Report` to see the results of the compilation.

3. If there are no errors, then your program is correct from the syntax point of view. **This may not mean that your program will do what you want, because you may have some logic errors that the compiler will not be able to detect**.

3.2.4 Pin Assignment

You need to specify which pins of the CPLD are connected to which inputs and which outputs. Some pins have already been wired to the LEDs and the push buttons. A list of those pins are provided in Table 1.

1. Select `Assignments > Assignment Editor`.

2. Under `Category` select `Pin`.

3. Double click on `<<new>>`, a drop-down menu will appear, select the input or output you want.

4. In the column labeled Location select the pin you want. Table 1 shows the locations of hardwire for your evaluation board. For example, if you wanted to connect the output "E" to the first red LED, you would select Location Pin_U4 for Node E.

5. Repeat steps 3 and 4 to assign all the inputs and outputs of 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 which may be connected to, say, a clock input – you 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`. To do this,

   (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 No.	Description
LED[0]	PIN_U13	Blue LED
LED[1]	PIN_V13	Green LED
LED[2]	PIN_U12	Yellow LED
LED[3]	PIN_V12	Red LED
LED[4]	PIN_V5	Blue LED
LED[5]	PIN_U5	Green LED
LED[6]	PIN_V4	Yellow LED
LED[7]	PIN_U4	Red LED
KEY[0]	PIN_U15	Button1
KEY[1]	PIN_V15	Button2
KEY[2]	PIN_U14	Button3
KEY[3]	PIN_V14	Button4
CLOCK_50	PIN_J6	50 MHz clock input

Table 1: Pin Assignments for the LEDs, buttons, and clock input

3.2.5 Simulating the Designed Circuit

When simulating a circuit you need to figure out the waveforms for the inputs that will make you confident that your circuit works. If you have a simple circuit, you can easily test all the possibilities. As the circuit gets more and more complicated you will need to figure out a scheme to verify its operation. In simulating your circuit there are three 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 detailed explanation of the above steps are described below.

1. Select `File > New`.
2. Click on `Vector Waveform File`.
3. Click `Ok`.
4. Save the file using some meaningful name, *filename.vwf*.
5. Set the desired simulation time by selecting `Edit > End`.
6. Select `View > Fit in Window`.
7. Select the inputs and outputs you want to observe by clicking `Edit > Insert > Insert Node or Bus`.

8. Click on `Node Finder`.
9. Click on the input or output you want to observe and click on the `>` sign. Repeat this process for all your inputs and outputs.
10. The next step is to specify the logic value of each of the inputs you have selected and the duration of that value.
11. Save the file, e.g. *lab2.vwf*.
12. After the waveforms have been defined, we can simulate our circuit. There are two types of modes that we are concerned with. 1) functional: we are not worried about the delays and we are interested to make sure that logically the circuit is working; 2) timing: we simulate the circuit and include the delays in all the gates. First perform the functional simulation and then perform the timing. To select the mode of the simulation:
    (a) Select `Assignments > Settings`.
    (b) Click on `Simulator Settings`.
    (c) Choose `Functional` or `Timing`. *For now choose* `Functional` *unless otherwise instructed.*
13. Create the required netlist that the waveform file will be applied to by selecting `Processing > Generate Functional Simulation Netlist`.
14. Run the simulation by clicking `Processing > Start Simulation`, or by clicking on the play icon in the simulation waveform window.

3.2.6 Programming the CPLD

The final step is to program the CPLD with your designed circuit.

1. Select `Tools > Prgorammer`.
2. Select `JTAG` in the `Mode` box.
3. If USB-Blaster is not chosen in the box next to the `Hardware Setup`, selected by clicking on the `Hardware Setup`.
4. You should see a file listed with extension *.pof*, if not add it.
5. Finally, press `Start`. The program will download on your board and once it is finished you can test your circuit in hardware.