MARIO PALACIOS

LAB COURSE (CpE 64) SECTION #20

WEDNESDAY

INSTRUCTOR: EMMANUEL DUPART

Part 1A. IDE Tutorial

Description:

The objective of this part is to complete the Altera-Quartus tutorial and make a led blink on the FPGA.

Problem Definition:

The Altera-Quartus is the software used to write, compile, and test Verilog code, and it is important to know how use the IDE provided on the software. This tutorial will provide as a great example on how to write Verilog, test using a testbench, and use the code on the FPGA to make an LED blink. The introduction will be the foundation to further projects.

Engineering Data:

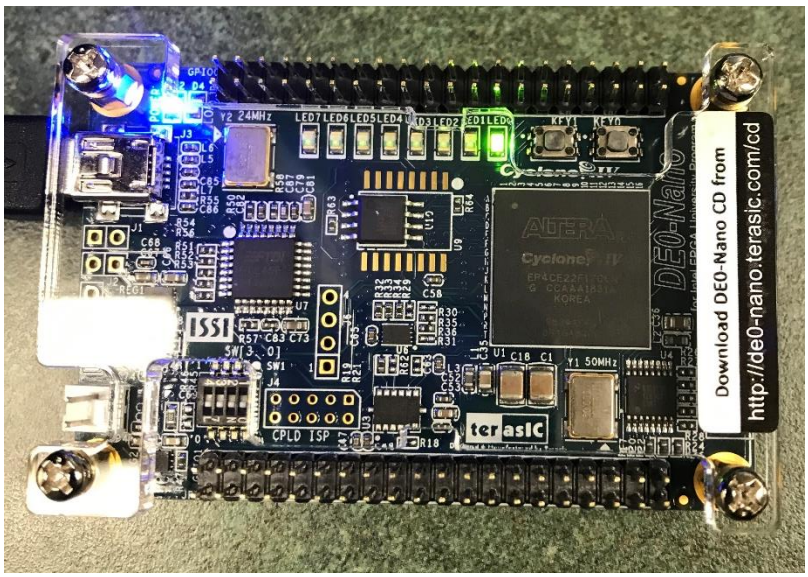

Figure 1. Blinking FPGA LED Light, LED [0]

There is a list of 8 pins [0-7] of LED lights to choose from, and for this part LED light zero was selected (PIN_A15). A list of inputs was also needed and for this part PIN_R8 was chosen.

Conclusion:

Once completing this part of the lab the foundation for future Verilog projects, knowing how a testbench is written is important for testing the functionality of the code. Creating waveforms to visually see how these inputs and outputs work together is an interesting part of this part. Overall this part took me 15 minutes.

Part 1B. Gates in Verilog

Description:

Using structural Verilog programming create a module for the 7400 Series TTL gates. Then create a single module combing all four logic gates, only having 2 inputs and 4 outputs.

Problem Definition:

Begin with creating a Verilog description of the 7408 AND Gate, then assign pins to the two inputs and one output. The inputs should be assigned a switch while the output an LED light. Afterwards download the design onto the FPGA and test the circuit against the truth table. Repeat these steps for the 7400(NAND), 7432(OR), and 7404(NOT) gates. Then finally create one single module that includes all four modules. There should only be 2 inputs and 4 outputs.

Engineering Data:

```
module AllGates(in1,in2,out1,out2,out3,out4);
        input in1,in2;
        output out1,out2,out3,out4;

        wire in1,in2,out1;
        and g1(out1,in1,in2);

        wire out2;
        or g2(out2,in1,in2);

        wire out3;
        nand g3(out3,in1,in2);

        wire out4;
        not g4(out4,in1);
endmodule
```

Figure 2. Structural Verilog Module of Four 7400 TTL Series Gates

Structural modeling in Verilog includes which gate is being used, while also declaring what inputs and outputs being used. The gates are structured in a way that the output is always first followed by the first input then the second input. The variable type *wire* is used to denote what you will be using in the logic gate. Waveforms are displayed using a testbench, where the testbench is designed using a combination of ones and zeroes, allowing for various outputs. This can be seen on Figure 3.

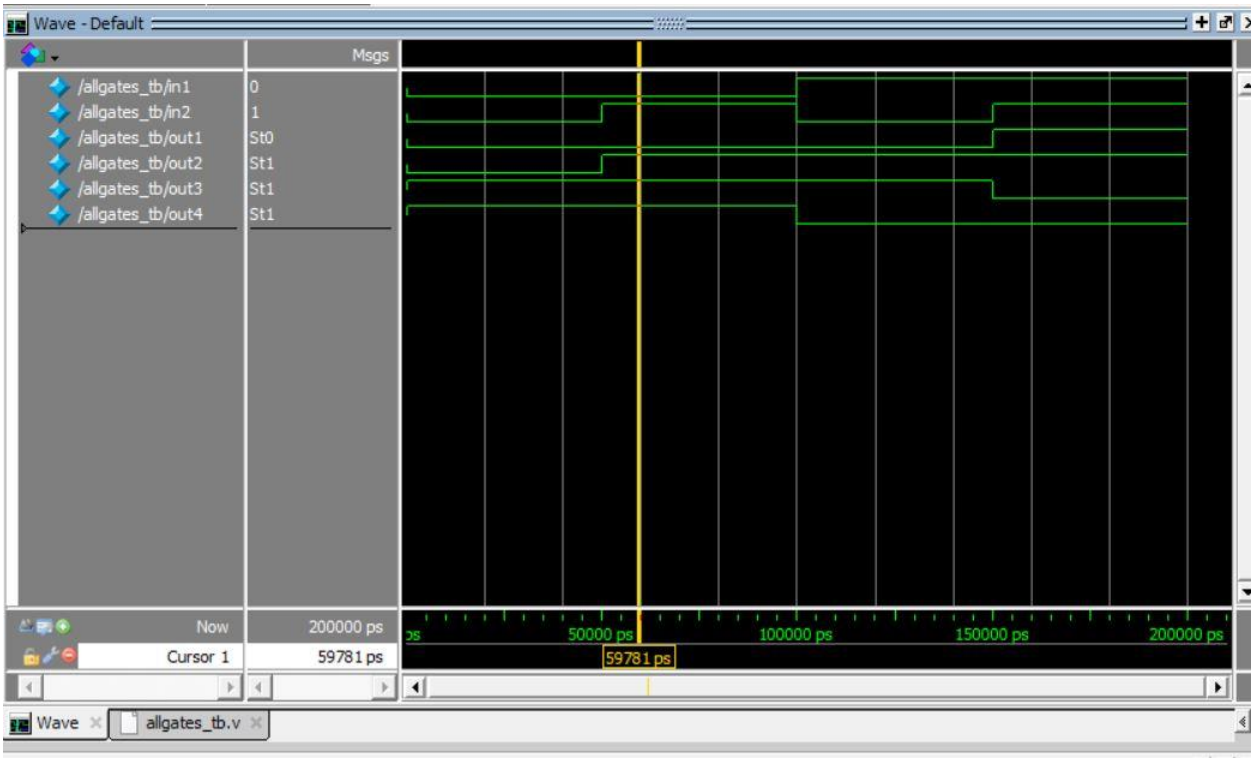

Figure 3. Waveform of the Four 7400 TTL Series Gates

Conclusion:

In conclusion, when designing the structural module in Verilog there will be declared inputs and outputs, while also putting them in a proper order when being used with a primitive. The primitives must have the desired output variable name first followed by the proper inputs. This part of the lab took me the longest because I was having issues with displaying the waveform.

Part 2. Exclusive-OR gate

Description:

The objective of this part of the lab is to build the EXCLUSIVE-OR gates in Multisim and the structural Verilog module. While also displaying its waveform.

Problem Definition:

There are two input EXCLUSICE-OR gates, write a structural model to implement the function. Then combine both gates into one module and Multisim diagram. Show that each circuit satisfies the truth table.

Engineering Data:

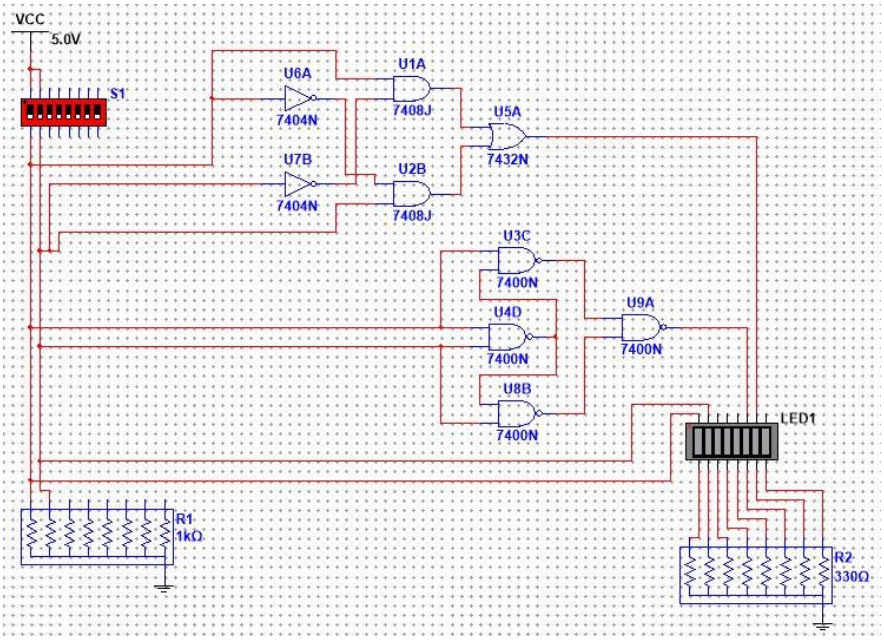

Figure 4. Multisim Diagram of XOR Gate (Regular Circuit and NAND Way)

```
module xorgate(x,y,f,f2);
        input x,y;
        output f,f2;

        //Exclusive-Or gate
        wire x,y,f;
        wire notxout,notyout,and1out,and2out;

        or or1(f,and1out,and2out);
        and and1(and1out,x,notyout);
        and and2(and2out,y,notxout);
        not notx(notxout,x);
        not noty(notyout,y);

        //NAND Way Exclusive-Or gate
        wire nand1out,nand2out,nand3out;

        nand nand1(f2,nand1out,nand3out);
        nand nand2(nand1out,x,nand2out);
        nand nand3(nand2out,x,y);
        nand nand4(nand3out,nand2out,y);
endmodule
```

Figure 5. Structural Verilog for Regular Circuit and NAND Way XOR Gate

Figure 6. XOR Gate Waveform for Both Regular Circuit and NAND Way

There are two ways to write an EXCLUSIVE-OR gate; the regular circuit way and the NAND way. The regular circuit way includes NOT, AND, and OR gates, it can also include just one NOT gate with a NAND and an OR gate. While the NAND way can be built with four NAND gates.

Conclusion:

To recap the EXCLUSIVE-OR gate, there are two ways of setting it up; the regular circuit and NAND way. Both will give you the same output and follows the truth table. Overall this part of the lab took me 20 minutes.

Part 3. Reducing a Boolean Equation

Description:

The objective of this part of the lab is to create a truth table to from the reduced Boolean expression. Then create a module using both the structural and dataflow in Verilog to display the Waveform.

Problem Definition:

Using standard AND, OR, and NOT symbols reduce the following equation;

$$F = /((/C + /D) \cdot (/A + B + /D))$$

In Dataflow the original equation will look like;

$$\sim((\sim C \mid \sim D) \,\&\, (\sim A \mid B \mid \sim D).$$

Engineering Data:

$$F = \sim((\sim C \mid \sim D) \,\&\, (\sim A \mid B \mid \sim D))$$
$$F = \sim(\sim C \mid \sim D) \mid \sim(\sim A \mid B \mid \sim D)$$
$$F = (\sim\sim C \,\&\, \sim\sim D) \mid (\sim\sim A \,\&\, \sim B \,\&\, \sim\sim D)$$
$$F = (C \,\&\, D) \mid (A \,\&\, \sim B \,\&\, D)$$

or

$$F = D \,\&\, (C \mid A \,\&\, \sim B)$$

INPUTS				OUTPUTS	
A	B	C	D	F	uF
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	1	1

Figure 7. Truth Table Using Reduced Equation (F) and Original Equation (uF)

```
//Dataflow Modeling
module equation(A,B,C,D,uF,F,);
        input A,B,C,D;
        output uF,F;

        wire A,B,C,D,uF,F;

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

endmodule
```

```
//Structural Modeling
module equation_structural(A,B,C,D,F);
        input A,B,C,D;
        output F;

        wire and3out,and2out,and1out,notB;

        not b_inv(notB,B);
        and and_BA(and1out,notB,A);
        and and_CD(and2out,C,D);
        and and_ABD(and3out,and1out,D);
        or or_final(f,and2out,and3out);

endmodule
```

Figure 8. Structural and Dataflow Modeling for Boolean Equation

Figure 9. Waveform for Reduced (F) and Original (uF) Boolean Expression

Conclusion:

Overall after this part of the lab, using dataflow modeling is a shorter way of creating modules for Boolean expressions over structural modeling. It is important to follow the laws of Boolean reduction in order to get a simplified expression. This part took me about 30 minutes to complete.

Part 4. Reducing Truth Table

Description:

The objective of this part is to create Boolean expressions from a truth table, while also creating a complete schematic diagram for the circuit using the four integrated circuits.

Problem Definition:

Use the truth table provided and use K-maps, Boolean algebra, and Laws to design the final equations for both F1 and F2. Then afterwards build the designs in Multisim and Verilog.

Input				Output	
A	B	C	D	F1	F2
0	0	0	0	0	1
0	0	0	1	1	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1
1	1	1	1	0	0

Figure 10. Truth Table for Part 4 of Lab

Engineering Data:

AB\CD	00	01	11	10
00	0	**1**	0	0
01	0	**1**	0	0
11	0	**1**	0	0
10	**1**	**1**	0	0

Figure 11. K-Map for F1 Outputs

F1 = (~C&D) | (A&~B&~D)
= ~C&((A~B)|D)

AB\CD	00	01	11	10
00	**1**	**1**	**1**	**1**
01	0	**1**	0	**1**
11	0	**1**	0	**1**
10	0	**1**	**1**	**1**

Figure 12. K-Map for F2 Outputs

F2 = (~A&~B) | (~C&D) | (C&~D) | (A&~B&D)
= ~B&(~A | (A&D)) | (~C&D) | (C&~D)
= ~B&(~A | D) | (~C&D) | (C&~D)
= ~B&~(A | ~D) | (~C&D) | (C&~D)

```
module Equation_Outputs(A,B,C,D,F1,F2);
        input A,B,C,D;
        output F1,F2;

        assign F1 = (~C&((A&~B)|D));
        assign F2 = ~B&~(A|~D)|(~C&D)|(C&~D);

endmodule
```

Figure 13. Verilog for Both Outputs in Dataflow

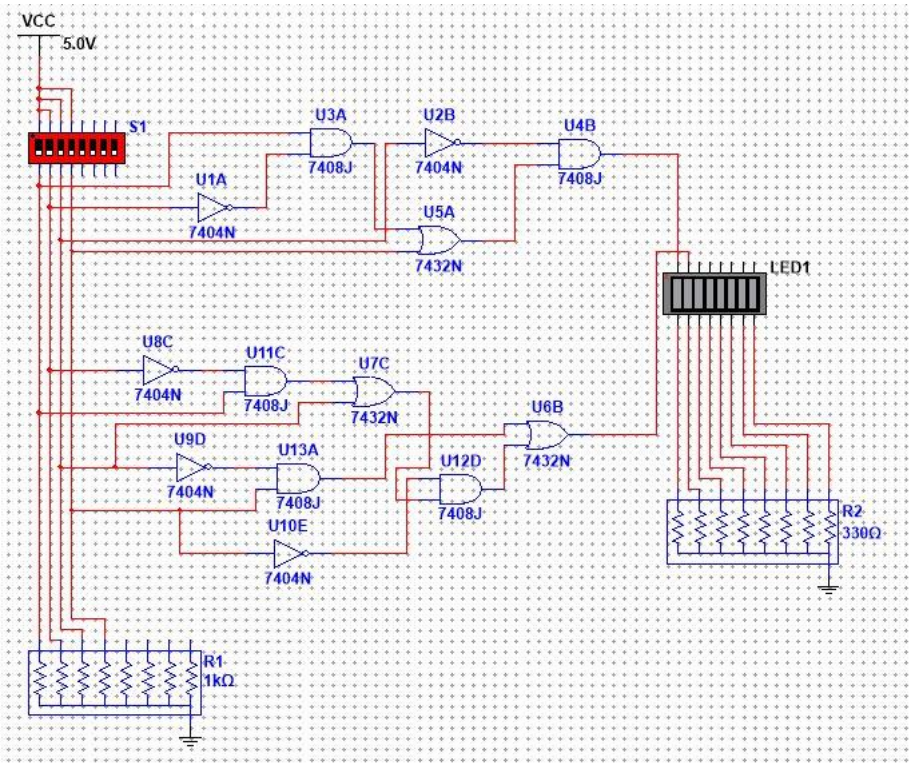

Figure 14. Schematic of F1 and F2 in Multisim

Figure 15. Waveform of F1 and F2 Outputs

When writing the Verilog module it seemed better and more efficient to write the equations in Dataflow, when Structural would've been multiple lines. Over the K-Maps were created by taking the outputs that have high signal only, then using the CD over AB to get my groups.

Conclusion:

This part of the lab took me a while, about 45 minutes because of writing the Verilog module in Dataflow. In order to receive the correct outputs the expression must be written correctly or else there will be errors.

Lab Questions:

#1: Is there an easier way to write an EXCLUSIVE-OR function still using primitives and structural modeling? Explain in the conclusion portion of your report.

Yes there is another easier way to write an EXLUSIVE-OR function using primitives and structural modeling. The circuit is called the Equivalent Circuit XOR Gate, which includes an OR and NAND gate going into an AND gate.

$$(A + B)(\overline{AB})$$

#2: If you really needed to use EXCLUSIVE-OR gates could you buy a 7400 Series TTL Integrated Circuit (IC)? If so, what is the TTL part number and explain and draw the pin out of the device

The TTL number of a single 2-input EXCLUSIVE-OR is 7486, and the pinout for it is below.