MARIO PALACIOS

LAB COURSE (CpE 64) SECTION #20

WENDESDAY

INSTRUCTOR: EMMANUEL DUPART

PART 1. ADDER WITH CONSTANT

Description:

The objective of this part of the lab is to practice designing more combinational logic circuits and get more experience with equations, K-maps, and Boolean Algebra.

Problem Description:

Design a four-bit adder that accepts any 4-bit input and adds a 4-bit binary constant to the input. Create truth tables, use K-maps to find equations and create a circuit schematic diagram. There should be a total of five equations for the five outputs of the adder. Use DeMorgan's theorem and circuit sharing between outputs. When creating schematic design in Multisim use the 4 gates available and the six inverter ICs. Record the outputs in a truth table.

Engineering Data:

INPUTS				ADDER OUTPUTS				
A	B	C	D	Sum4	Sum3	Sum2	Sum1	Sum0
0	0	0	0	0	0	1	1	1
0	0	0	1	0	1	0	0	0
0	0	1	0	0	1	0	0	1
0	0	1	1	0	1	0	1	0
0	1	0	0	0	1	0	1	1
0	1	0	1	0	1	1	0	0
0	1	1	0	0	1	1	0	1
0	1	1	1	0	1	1	1	0
1	0	0	0	1	1	1	1	1
1	0	0	1	1	0	0	0	0
1	0	1	0	1	0	0	0	1
1	0	1	1	1	0	0	1	0
1	1	0	0	1	0	0	1	1
1	1	0	1	1	0	1	0	0
1	1	1	0	1	0	1	0	1
1	1	1	1	1	0	1	1	0

Figure 1. Truth Table for ADDER with CONSTANT (0111)[7]

Red	Blue	Purple	Green	Black
A	$\bar{A}(D+C+B)+A\bar{B}\bar{C}\bar{D}$	$B(D+C)+\bar{B}\bar{C}\bar{D}$	$\bar{C}\bar{D}+CD$	$\bar{D}$

Figure 2. Simplified Boolean Expressions

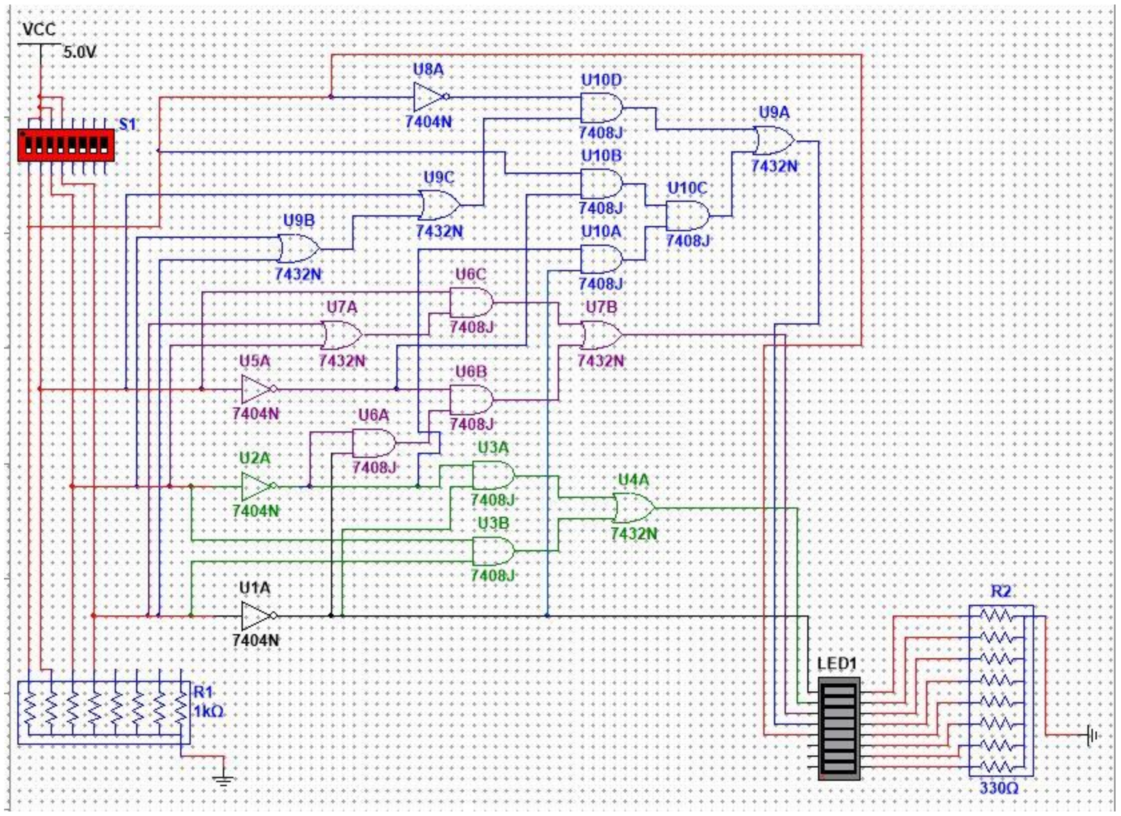


Figure 3. Circuit Schematic for 5 Boolean Equations

Conclusion:

In conclusion creating an adder is possible if a K-map is made from which a truth table is needed. This part was great for practicing my skills for combinational logic. This part of the lab took me about fifteen minutes to complete.

PART 2. ADDER WITH CONSTANT CONTINUED

Description:

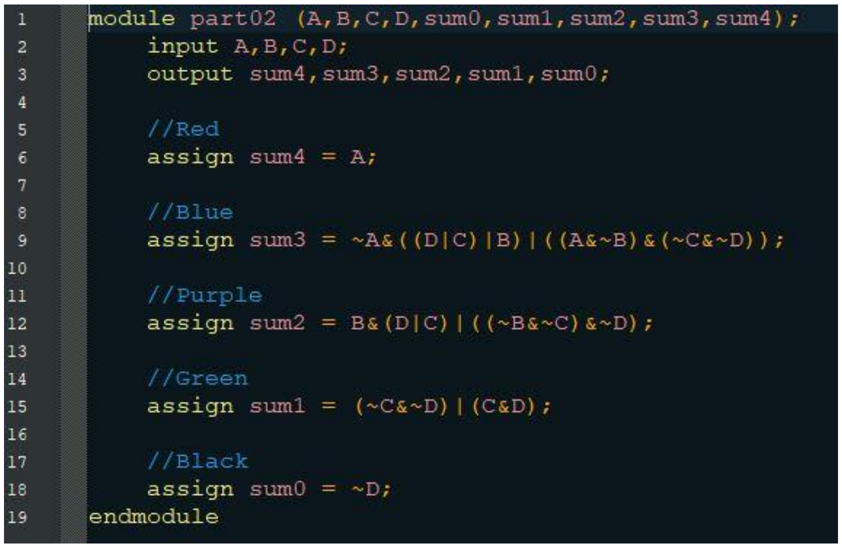
The object of this part is to practice Verilog writing techniques and testbench creating knowledge.

Problem Description:

Using the five Boolean equations created in the previous part write a Data Flow model description in Verilog HDL. Then test the Verilog solution on the FPGA board. Create a testbench and see if the waveform matches with the predicted results done in the previous part.

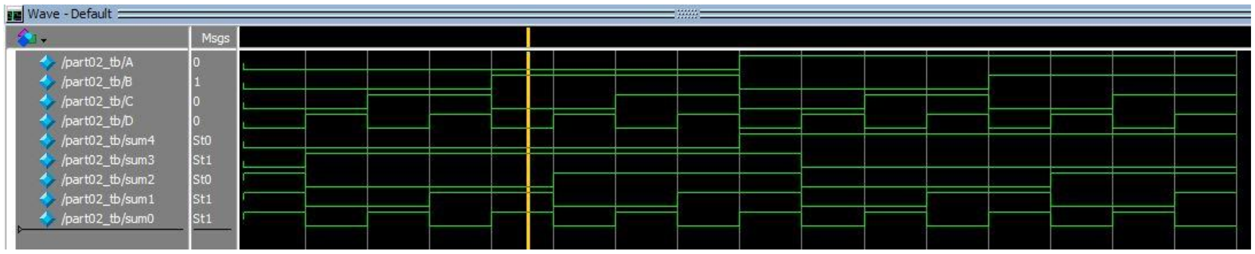
Engineering Data:

The Boolean equations that were derived from the truth table and K-maps can be seen on Figure 2 in the first part.



```verilog
module part02 (A,B,C,D,sum0,sum1,sum2,sum3,sum4);
    input A,B,C,D;
    output sum4,sum3,sum2,sum1,sum0;

    //Red
    assign sum4 = A;

    //Blue
    assign sum3 = ~A&((D|C)|B)|((A&~B)&(~C&~D));

    //Purple
    assign sum2 = B&(D|C)|((~B&~C)&~D);

    //Green
    assign sum1 = (~C&~D)|(C&D);

    //Black
    assign sum0 = ~D;
endmodule
```

Figure 4. Verilog in Data Flow Description for Boolean Equations



```
/part02_tb/A      0
/part02_tb/B      1
/part02_tb/C      0
/part02_tb/D      0
/part02_tb/sum4   St0
/part02_tb/sum3   St1
/part02_tb/sum2   St0
/part02_tb/sum1   St1
/part02_tb/sum0   St1
```

Figure 5. Waveform from Boolean Equations

Conclusion:

Overall refining my skills in writing Verilog in Data Flow description allowed me to create short and precise code. This part took me about fifteen minutes to complete

PART 3. FULL 4-BIT ADDER

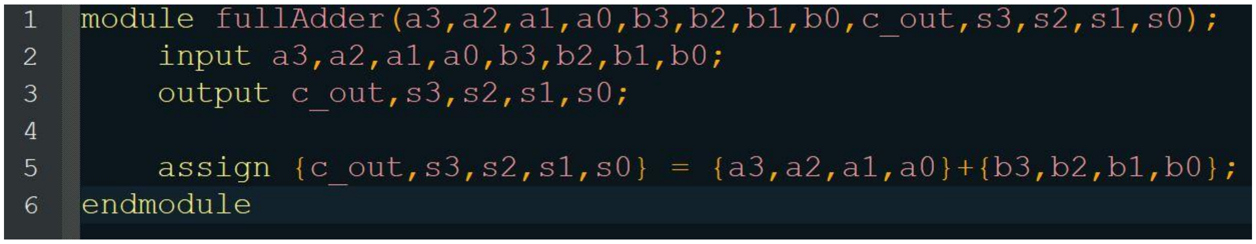
Description:

To create a full 4-bit adder that adds any 4 bits to any other 4 bits.

Problem Description:

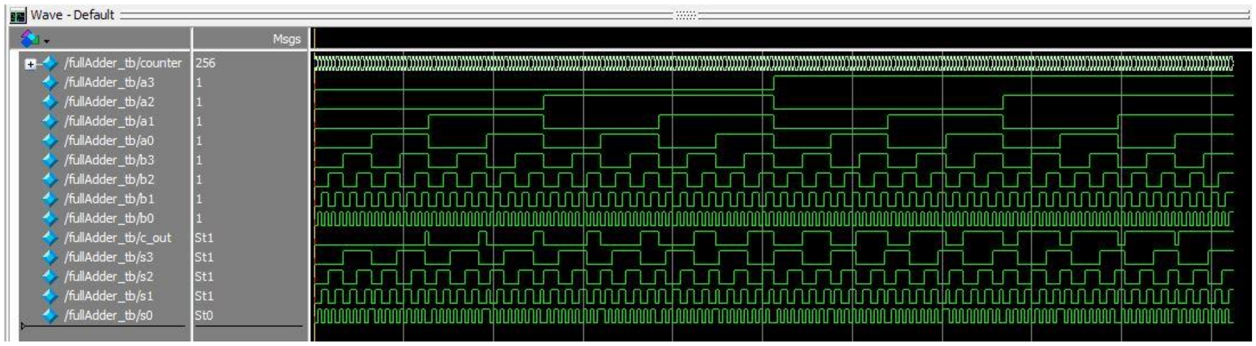
Have the two 4-bit numbers be interpreted as unassigned, and call the two numbers A and B. The bits under A will be called a3, a2, a1, and a0; same goes for the bits under B b3, b2, b1, and b0. The adder will have five outputs called s4, s3, s2, s1, s0.

Engineering Data:



```
module fullAdder(a3,a2,a1,a0,b3,b2,b1,b0,c_out,s3,s2,s1,s0);
    input a3,a2,a1,a0,b3,b2,b1,b0;
    output c_out,s3,s2,s1,s0;

    assign {c_out,s3,s2,s1,s0} = {a3,a2,a1,a0}+{b3,b2,b1,b0};
endmodule
```

Figure 6. Verilog Code for Full 4-Bit Adder in Structural Modeling



```
/fullAdder_tb/counter  256
/fullAdder_tb/a3       1
/fullAdder_tb/a2       1
/fullAdder_tb/a1       1
/fullAdder_tb/a0       1
/fullAdder_tb/b3       1
/fullAdder_tb/b2       1
/fullAdder_tb/b1       1
/fullAdder_tb/b0       1
/fullAdder_tb/c_out    St1
/fullAdder_tb/s3       St1
/fullAdder_tb/s2       St1
/fullAdder_tb/s1       St1
/fullAdder_tb/s0       St0
```

Figure 7. Waveform for 4-Bit Adder

Conclusion:

In conclusion the 4-Bit adder allowed me to visualize how this can play a big role in ALUs. I was able to practice more on my Structural modeling for code in Verilog. Overall this part of the lab took me about 15 minutes to complete because it was a continuation of the previous section.

PART 4. COMPARATOR USING GATES

Description:

To design a simple combinational circuit, 2-bit comparator, and to gain insight into IC chip minimization instead of gate minimization in implementing the combinational circuit.

Problem Description:

Create a 4-input, 3-circuit that compares 2-bit unsigned numbers. Call the 4-inputs a1, a0, b1, and b0; where the most significant bit will be a1/b1 and the least significant a0/b0. We are doing this because it compares the binary number a1a0 with the binary number b1b0. Afterwards draw a circuit diagram with only NAND gates on Multisim.

Engineering Data:

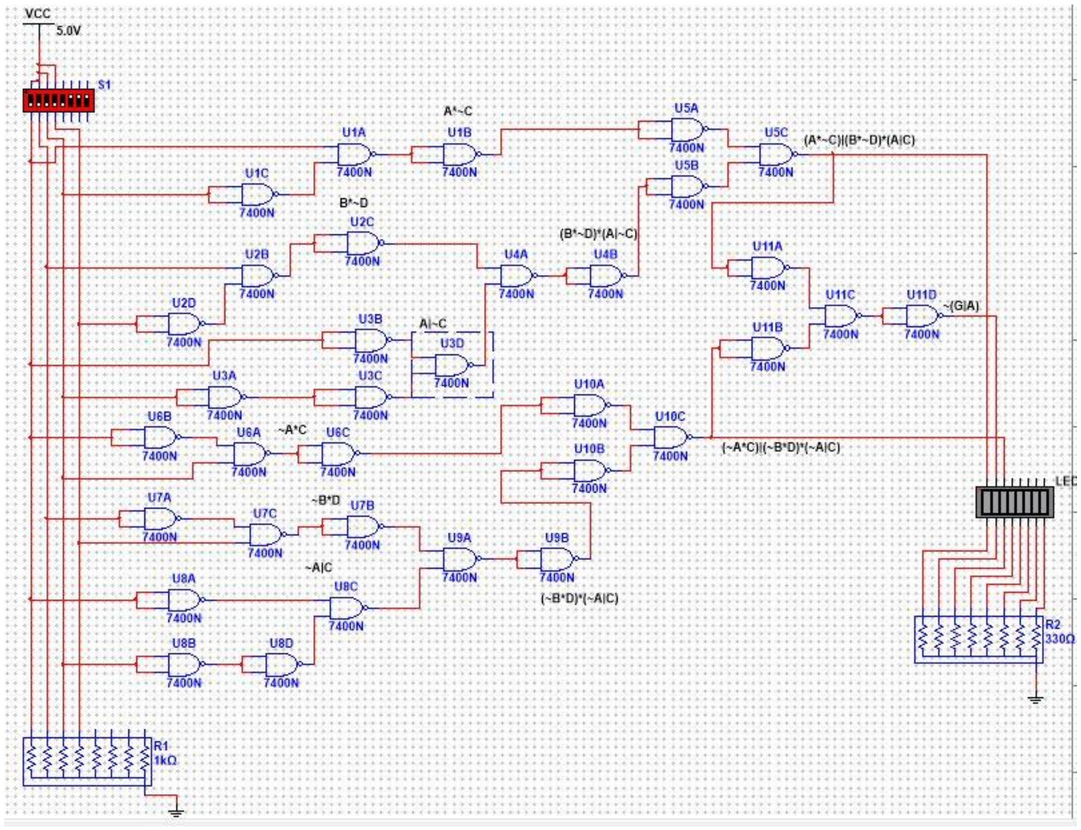


Figure 8. Multisim of NAND Comparator

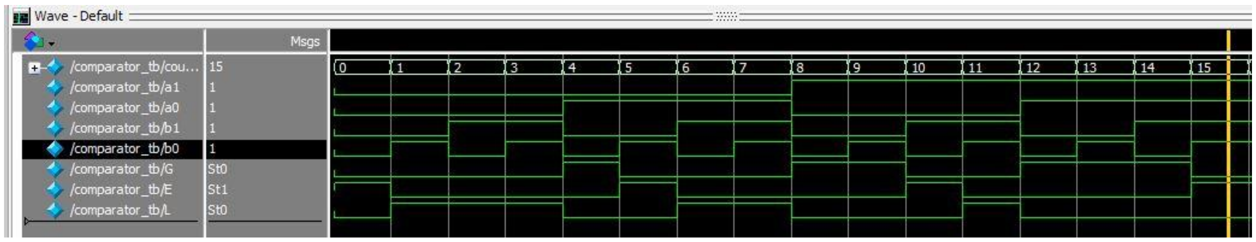
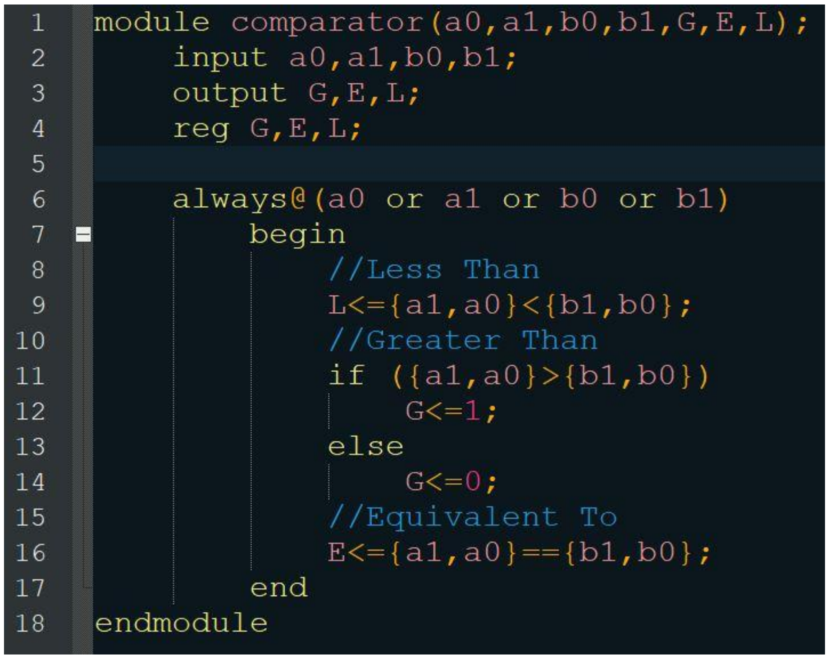


Figure 9. Waveform for Comparator



```
module comparator(a0,a1,b0,b1,G,E,L);
    input a0,a1,b0,b1;
    output G,E,L;
    reg G,E,L;

    always@(a0 or a1 or b0 or b1)
        begin
            //Less Than
            L<={a1,a0}<{b1,b0};
            //Greater Than
            if ({a1,a0}>{b1,b0})
                G<=1;
            else
                G<=0;
            //Equivalent To
            E<={a1,a0}=={b1,b0};
        end
endmodule
```

Figure 10. Verilog for Comparator in "Behavioral" Modeling

Conclusion:

The 2-bit comparator take in two bits and compares them with one another, then it separates which two bits are greater, less than, or equal. A comparator is the important building block for CPUs and microcontrollers. This part of the lab took me about 30 minutes to complete, but afterwards I came out with new found knowledge.