ECE 273: Introduction to Digital Logic

Lab <4 >: <Two's Complement ADD/SUB>

By

<Hussein El-Souri >

Lab Instructor: < Zheming Liang >

September 21st, 2016
Fall 2016

Honor Code:

I have neither given nor received unauthorized assistance on this graded report.

X___Hussein El-Souri

Department of Electrical and Computer Engineering
College of Engineering and Computer Science
The University of Michigan-Dearborn

Objective

I aim to design a 3-bit two's complement add/sub logic circuit.

Theory

A truth table was constructed for the full adder circuit. The XOR gate was also essential for this lab.

X	Y	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 1: Full Adder Truth Table

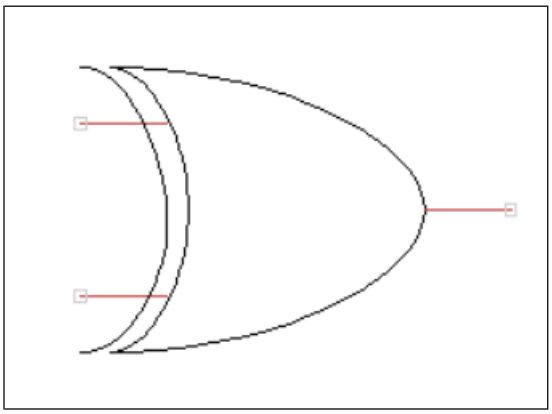

Figure 1: The XOR Logic Gate

x	y	x⊕y
0	0	0
0	1	1
1	0	1
1	1	0

Table 2: XOR Truth Table

Equipment Used

- Software: Xilinx, TinyCAD, PlanAhead
- Hardware: BASYS2 Board

Procedure

First, I use the truth table to derive the equations for each of the outputs. Sum = X ⊕ Y ⊕ Cin

Cout = XCin + YCin + XY

The equation of the sum portrays the importance of the XOR logic gate.

I use these equations in a VHDL module to create a symbol for the full adder. Writing these equations in the VHDL module in the proper context I get the architectural behavior of adder as such:

```
Architecture Behavioral of adder is
begin
Sum <= X XOR Y XOR Cin;
Cout <= (X AND Cin) OR (Y AND Cin) OR (X AND Y);
end Behavioral;
```

I proceed to create a symbol from this module and cascade three adders together since I aim to create a 3-bit full adder. Each full adder has three inputs (X, Y, and Cin) and two outputs (Cout and Sum).I cascade the adders such that I attach a wire between the Cout of the first and Cin in the second and repeat this process between second and third. This leaves the Cin of the first full adder along with the Cout of the last one unattached to anything. Eventually I aim to add X2X1X0 with Y2Y1Y0 and in this case I have no carry in (Cin=0). However, I know that in two's compliment subtraction compliments the input (Y2Y1Y0) and adds 1 to it so I have a carry
in (Cin=1).

I accomplish the following by having a control input (say W) that controls when I am doing addition/subtraction. So, if W=0 I am adding and when W=1 I am subtracting in other words complementing Y through the operation W ⊕ Y (XOR gate with W and Y as inputs. I do this for all occurrences of the input Y (Y0, Y1, Y2) and assign the proper I/O markers such that the first Cin is the same as W and the overflow V= $C_{out} \oplus C_{out-1}$ to have the following schematic:

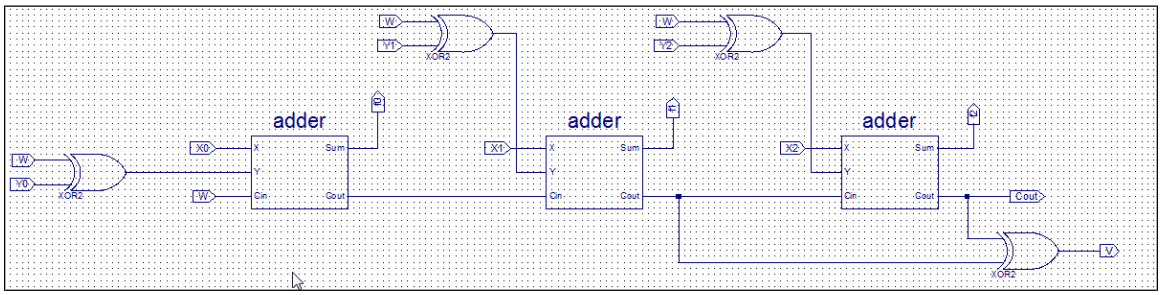

Figure 2: 3-bit full adder schematic

After saving properly I then proceed to create the test bench for this schematic. Since the total number of possible combinations is too many I carefully choose for examples to demonstrate that the adder functions correctly. I write these 4 examples in a test bench code as such:

```
tb : PROCESS
        BEGIN
                W <= '0'; X0 <= '0'; Y0 <= '0' ; X1 <= '1'; Y1 <= '1'; X2 <= '1'; Y2 <= '1';
                WAIT FOR 200ns;
                W <= '0'; X0 <= '0'; Y0 <= '0' ; X1 <= '1'; Y1 <= '0'; X2 <= '1'; Y2 <= '1';
                WAIT FOR 200ns;
                W <= '1'; X0 <= '0'; Y0 <= '0' ; X1 <= '0'; Y1 <= '1'; X2 <= '1'; Y2 <= '1';
                WAIT FOR 200ns;
                W <= '1'; X0 <= '0'; Y0 <= '0' ; X1 <= '1'; Y1 <= '1'; X2 <= '0'; Y2 <= '1';
                WAIT FOR 200ns;
END PROCESS;
```

Writing this code allows us to simulate the circuit after changing the time of the simulation to 800ns resulting in the following:

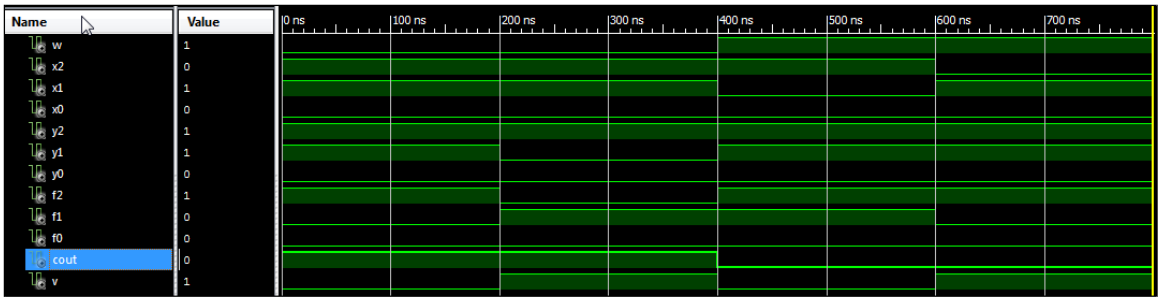

Figure 3: Waveforms of 3-bit full adder

If I look at the waveforms between the 600 and 800ns I see that W=1 so I am preforming subtraction X2=0, X1=1, X0=0, Y2=1, Y1=1 and Y0=0 forming 010 and 110(2+(-5)). So first I have to complement 110 to get 001 and add 1 to get 010 and hence I am preforming 010+010 to get 100 and a carry out of 0 and an overflow which is consistent with the waveforms since f2=1, f1=0, f0=0, Cout=0 and v=1.

I try to confirm these results through hardware by assigning inputs to switches and outputs to LED bulbs. I can do so using the PlanAhead software to produce the following physical constraints:

```
# PlanAhead Generated physical constraints
NET "Cout" LOC = P7;
NET "V" LOC = P6;
NET "W" LOC = P11;
NET "X0" LOC = F3;
NET "X1" LOC = E2;
NET "X2" LOC = N3;
NET "Y0" LOC = K3;
NET "Y1" LOC = B4;
NET "Y2" LOC = G3;
NET "f0" LOC = N4;
NET "f1" LOC = P4;
NET "f2" LOC = G1;
```

Saving theses constraints will allow us to generate a bit file to program onto the BASYS2 board and I will try the example that I discussed in the waveforms above. Firstly, W=1 meaning that switch P11 is switched up (to preform subtraction) also X2=0, X1=1, X0=0 meaning switch N3 is switched down, switch E2 is switched up and switch F3 is switched down whereas Y2=1, Y1=1 and Y0=0 meaning switches G3 and B4 am switched up while switch K3 is switched down. I should observe f2=1, f1=0, f0=0, Cout=0 and v=1 meaning bulb G1 is on while bulbs P4, N4 and P7 are off and finally bulb P6 to be on.

Results

Inspecting the board, I can see the results are as expected and even when trying different examples the circuit shows full functionality.

Conclusion

The 3-bit full adder design was successful both theoretically and practically and I was able to confirm the software design through the examples on the board.