Noah Khan

April 15, 2022

4-Bit Hexadecimal Display

Description:

For this lab, I designed a 3-bit adder with a carry-in and displayed the result to the seven-segment display on the Basys 3 board. I created 3 files: a top-level file, full adder file and a seven-segment converter file. The full adder file creates a full adder, hence the name. The top adder created 3 instances of the full adder, thus creating a 3-bit adder. The seven-segment converter file sets up logic to turn on the LEDs on the board.

Methods:

I started this lab by creating a truth table for the full adder. Afterwards, I created the boolean expressions for the table. Then, I created the full adder file with 3 inputs and 2 outputs, where the outputs were assigned to the boolean expressions.

Once the full adder file was completed, I worked on the seven-segment converter. I started by creating a truth table with 4 inputs (n3 - n0) and 7 outputs (a - g). Where the input represented a binary number and the output controlled which LEDs were turned on to display a number from 0 - F. After carefully reading the Basys 3 Reference Manual, I learned which letters (outputs) represented which LED. I had to keep in mind that this board was active low when creating this truth table. Once the truth table was complete, I created boolean expressions. After completing the truth table and the boolean expressions, I created a seven-segment converter file with 4 inputs and 7 outputs. Where the outputs were assigned to the expressions.

Once the full adder and seven segment converter files were created, I created a top-level file named “top_adder.” Going based off of the first diagram in the lab manual, I created this file to have 7 inputs and 12 outputs. Where the inputs were c_in, a0-a3, and b0-b3 and the outputs were CA-CG, AN0-AN3, and DP. I created 3 instances of the full adder and 1 instance of the seven-segment converter. Following diagram 3 in the lab manual, I created 2 wires (c1 and c2) to connect the full adders. Referring to diagram 1 in the lab manual, I created 4 switch wires (sw0, sw1, sw2, sw4) to go from the 3-bit adder to the seven-segment converter. I connected each wire to the appropriate ports once the wires, inputs and outputs were established. This was done by referring to diagrams 1 and 3 from the lab manual. I assigned AN0 to 0 and the rest to 1. I also assigned DP to 0.

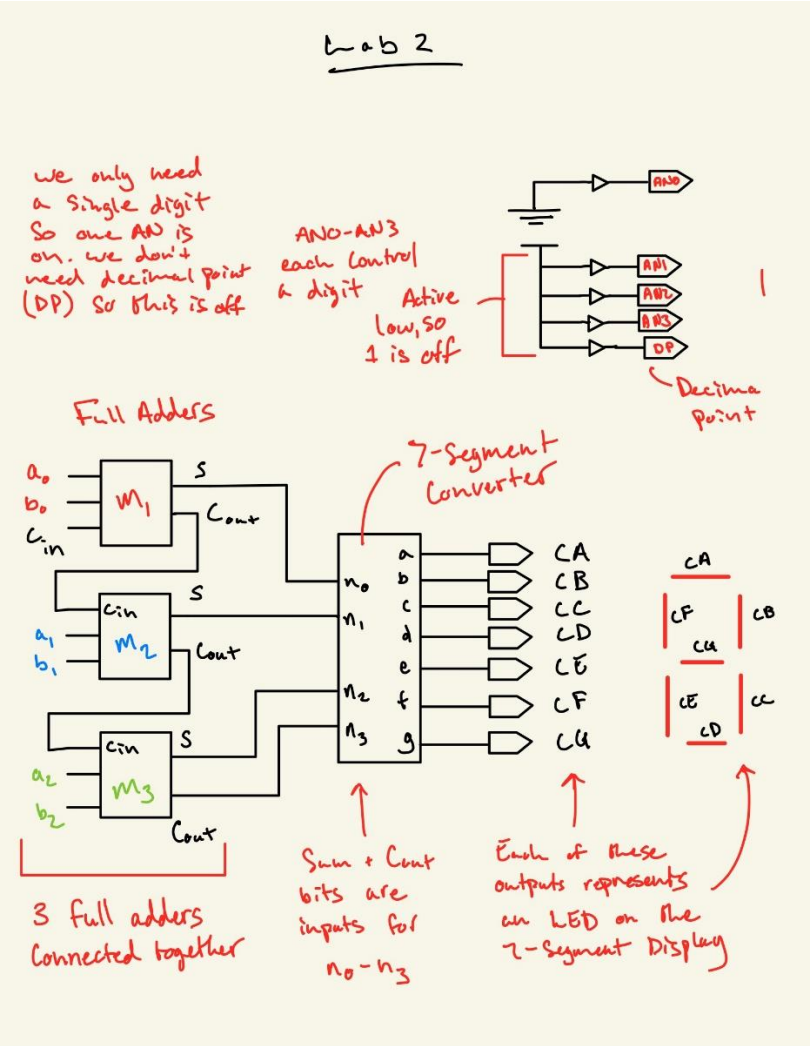
After all these files were created, I changed my constraints file. For the switches, I changed sw0 - sw6 to c_in, a0-a3, and b0-b3. For the 7-segment display portion, I added CC - CG, DP, and AN0 - AN4.

The next step I took was to download the testbench file and add it to the project. I followed the "TO-DO" portion of the in this file, which instructed me to change a few variable names. Afterwards, I added the rest of the test cases.

Once everything was set, I ran a behavioral simulation. The simulation looked good, so I generated a bitstream. Then, I programmed the board with the file.

Results:

Design:



In the diagram above, the full adders are connected in sequence, so the carryout bit is fed into the neighboring adder as a carry in bit. The third adder feeds the carryout bit into the seven-segment converter. All of the adders have a sum bit which go directly into the seven-segment converter.

Once the seven-segment receives the bits, it uses logic to display a character form 0 – F on the seven-segment display on the Basys 3 board.

Full Adder Truth Table

a	**b**	**c**	**s**	**C_{out}**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Seven-Segment Converter Truth Table

n_3	**n_3**	**n_3**	**n_3**	**CA**	**CB**	**CC**	**CD**	**CE**	**CF**	**CG**
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	1	0	0
1	0	1	0	0	0	0	1	0	0	0
1	0	1	1	1	1	0	0	0	0	0
1	1	0	0	0	1	1	0	0	0	1
1	1	0	1	1	0	0	0	0	1	0
1	1	1	0	0	1	1	0	0	0	0
1	1	1	1	0	1	1	1	0	0	0

The seven-segment converter uses the logic in the truth table above. That is, the inputs represent a hex number. The right side, or the outputs, will display that digit on the seven-segment display.

This is done by turning off certain LEDs on the display. Since this system is active low 1 will represent off and 0 will represent on. From this truth table, I derived Boolean expressions (can be found in the appendix).

Testing & Simulation:

I tested my design by using the behavioral simulation in Vivado. Before running this test, I triple checked my table and expressions to make sure everything was logically sound. There were no corner cases. There were a few hurdles I had to overcome while completing this lab. The main ones were figuring out the system was active low, creating additional test cases, figuring out how to properly instantiate modules, and understanding which wire goes to which port.

I had to approach this lab mutinously once I found out that this system was active low. One small careless mistake could have set me back substantially.

Creating additional test cases was a challenge for me because I did not know how the full adders were behaving in order to get the digits. I thought I was supposed to do ordinary binary addition.

In the top-level file, I had to create instances of the full adder module and seven-segment converter. What I didn't know was that I was using ports and wires. For example, it would like something like:

```
fulladder m1 ( .a(a0) … )
```

I did not know the ".a" represented the port and the "a0" was the wire. Once I figured this out, following diagram 1 in the lab manual was easier.

Lab Questions:

Document which pins of the FPGA you used and how they were connected to the switches and 7-segment displays.

I used the pins attached to switches 7-0. On the board they are named W14, V15, W15, W17, W16, V16, and V16. Each pin represented a bit, when the pin was low (0), it would interact with a segment on the 7-segment display. A hex digit would display depending on the switches that were on.

The longest path from any input to any output in the 3-bit adder.

The longest path in a single adder would be the a0 input. This input goes through 5 gates. In terms of the whole 3-bit adder, I would say a0 would still be the longest path because it contributes to the carryout bit which goes into the second full adder. This carryout bit also contributes to another carryout bit that will be fed into the last full adder, which will be an input for the seven-segment converter. In total it will travel through 12 gates.

For an n-bit adder determine the length of the longest path from any input to any output (in terms of n).

For an n bit adder, the length of the longest path would be 2^3 which equals 8.

Calculate the number of possible input values (7-bit vectors) for your adder? Give the percentage of these that you tested in your simulation.

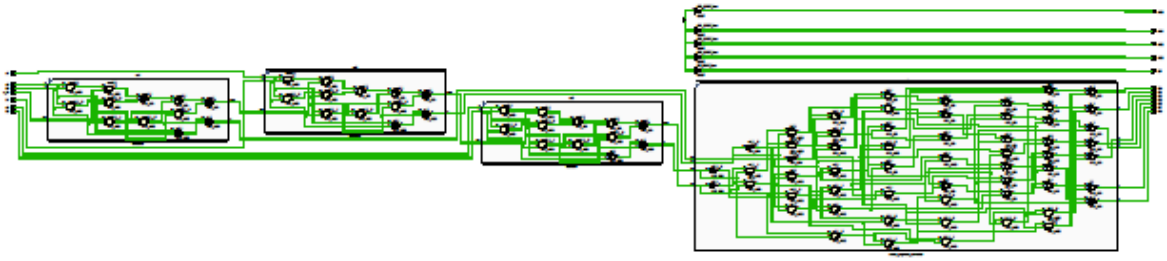
The number of possible input values would be 128. This number was obtained by taking 2 to the power of 7. 11% was tested with the simulation.

Conclusion:

From this lab, I learned how to use hierarchy by using modules and simulating my design using the Vivado simulator. I created a full adder module and created 3 instances of this module in my top-level design file. I also created an instance of the seven-segment converter. These instances had access to the inputs/outputs of the modules. The constraints file uses the inputs/outputs from the top-level file and does not interact with the lower-level files. Once I had all my modules created, I ran a behavioral simulation to test if my logic was sound. In addition, I have a better idea of the 3-bit adder's functionality and how the seven-segment display works.

Appendix:

Elaborated Design Top-Level



Full Adder Logic

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/05/2022 04:34:42 PM
// Design Name:
// Module Name: fulladder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module fulladder(
    input a,
    input b,
    input cin,
    output cout,
    output s
    );

    assign s = a ^ b ^ cin;
    assign cout = (~a & b & cin) | (a & ~b & cin) | (a & b & ~cin) | (a & b & cin);

endmodule
```

Seven-Segment

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/05/2022 03:40:02 PM
// Design Name:
// Module Name: seven_segment_converter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module seven_segment_converter(
    input n3,
    input n2,
    input n1,
    input n0,
    output a,
    output b,
    output c,
    output d,
    output e,
    output f,
    output g
    );

    assign a = (~n3 & ~n2 & ~n1 & n0) | (~n3 & n2 & ~n1 & ~n0) | (n3 & ~n2 & n1 &
n0) | (n3 & n2 & ~n1 & n0);
    assign b = (~n3 & n2 & ~n1 & n0) | (~n3 & n2 & n1 & ~n0) | (n3 & ~n2 & n1 & n0)
| (n3 & n2 & ~n1 & ~n0) | (n3 & n2 & n1 & ~n0) | (n3 & n2 & n1 & n0);
    assign c = (~n3 & ~n2 & n1 & ~n0) | (n3 & n2 & ~n1 & ~n0) | (n3 & n2 & n1 & ~n0)
| (n3 & n2 & n1 & n0);
    assign d = (~n3 & ~n2 & ~n1 & n0) | (~n3 & n2 & ~n1 & ~n0) | (~n3 & n2 & n1 &
n0) | (n3 & ~n2 & ~n1 & n0) | (n3 & ~n2 & n1 & ~n0) | (n3 & n2 & n1 & n0);
    assign e = (~n3 & ~n2 & ~n1 & n0) | (~n3 & ~n2 & n1 & n0) | (~n3 & n2 & ~n1 &
~n0) | (~n3 & n2 & ~n1 & n0) | (~n3 & n2 & n1 & n0) | (n3 & ~n2 & ~n1 & n0);
```

```
    assign f = (~n3 & ~n2 & ~n1 & n0) | (~n3 & ~n2 & n1 & ~n0) | (~n3 & ~n2 & n1 &
n0) | (~n3 & n2 & n1 & n0) | (n3 & n2 & ~n1 & n0);
    assign g = (~n3 & ~n2 & ~n1 & ~n0) | (~n3 & ~n2 & ~n1 & n0) | (~n3 & n2 & n1 &
n0) | (n3 & n2 & ~n1 & ~n0);
endmodule
```

Top Adder Logic

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/05/2022 04:37:26 PMDesign Name:
// Module Name: top_adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module top_adder(
    input a0,
    input b0,
    input a1,
    input b1,
    input a2,
    input b2,
    input c_in,

    output CA,
    output CB,
    output CC,
    output CD,
    output CE,
    output CF,
    output CG,
    output DP,
    output AN0,
    output AN1,
    output AN2,
    output AN3
    );

    assign AN0 = 1'b0;
    assign AN1 = 1'b1;
```

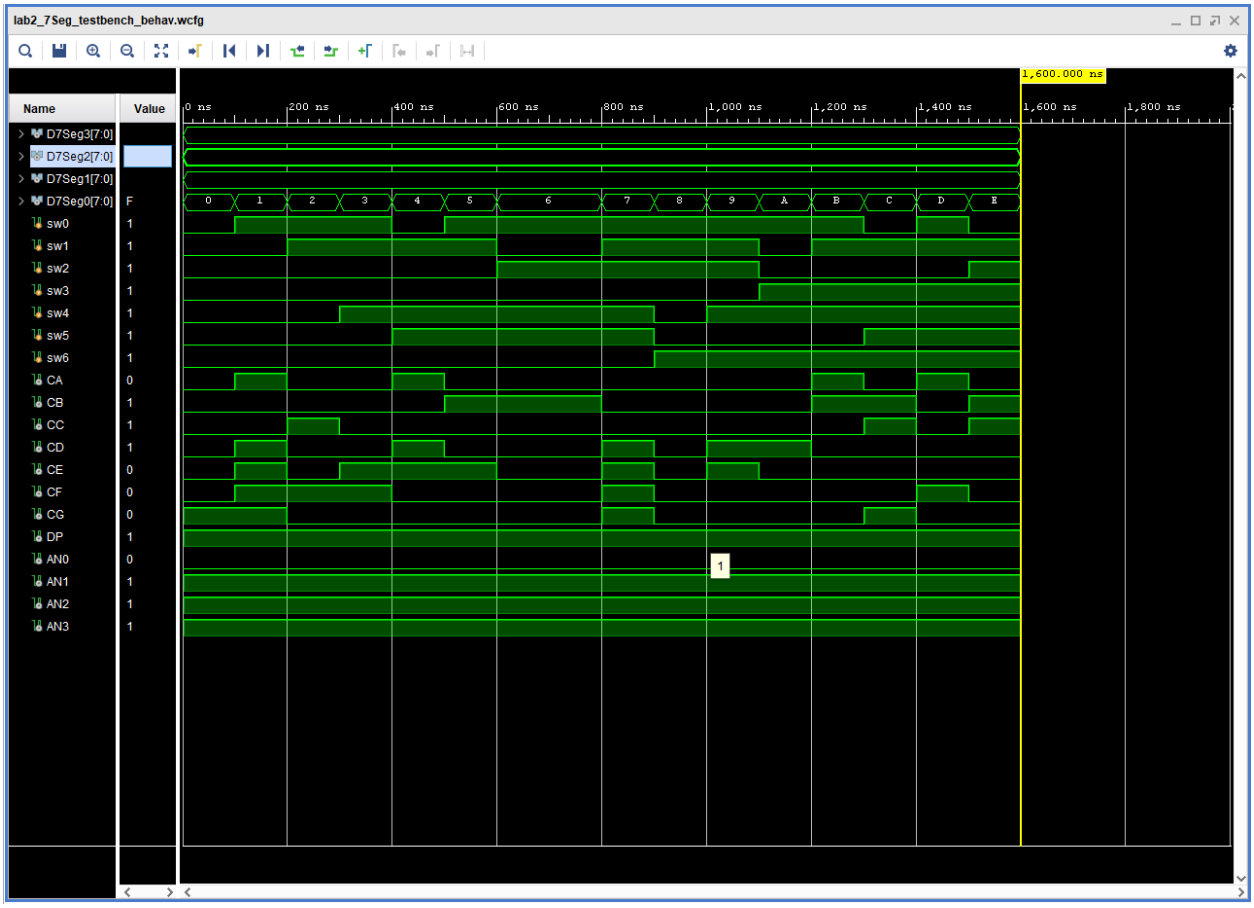
```
    assign AN2 = 1'b1;
    assign AN3 = 1'b1;
    assign DP = 1'b1;

    wire c1, c2;
    wire s0, s1, s2, s3;

    fulladder m1 (.a(a0), .b(b0), .cin(c_in), .cout(c1), .s(s0));
    fulladder m2 (.a(a1), .b(b1), .cin(c1), .cout(c2), .s(s1));
    fulladder m3 (.a(a2), .b(b2), .cin(c2), .cout(s3), .s(s2));
    seven_segment_converter m4 (.n3(s3), .n2(s2), .n1(s1), .n0(s0), .a(CA), .b(CB),
.c(CC), .d(CD), .e(CE), .f(CF), .g(CG)); //, .dp(DP), .an0(AN0), .an1(AN1),
.an2(AN2), .an3(AN3));

endmodule
```

Simulation


lab2_7Seg_testbench_behav.wcfg
Name
Value
D7Seg3[7:0]
D7Seg2[7:0]
D7Seg1[7:0]
D7Seg0[7:0]
F
sw0
1
sw1
1
sw2
1
sw3
1
sw4
1
sw5
1
sw6
1
CA
0
CB
1
CC
1
CD
1
CE
0
CF
0
CG
0
DP
1
AN0
0
AN1
1
AN2
1
AN3
1
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
1,400 ns
1,600 ns
1,800 ns
1,600.000 ns
0
1
2
3
4
5
6
7
8
9
A
B
C
D
E