Lab 2
Simon Carballo
1618309
10/23/2020
Section D

Write Up

- **Description:** In this lab we were asked to create a functional 7-segment Display on the Basys3 board that can change values using serven switches. In order to create this, we were tasked with designing a 4-bit adder, and a seven-segment-converter to put together using wires.

- **Design:** For this design we have the Top Level, with 2 subsections including the 4-bit adder and the seven-segment converter. Inside of the 4-bit adders are 3 Full adders (n-bit = (n-1)Full_Adders) that change the input of the converter depending on the active switches.
    - 4-bit Adder:
      First let's look at a normal Full Adder:
        - Truth Table of a Full Adder:

a	b	c	Sum	Cout
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

        - As we see in the truth table we add the sum of a, b, and cin, and when there is a value over 1, there would be an overflow representing Cout.
        - We get this equation for the Sum and Cout according to the truth table:
            - Sum = !a!bc+!ab!c+a!b!c+abc
              !c(!ab+a!b)+c(!a!b+ab)
              !c(a xor b)+c(a xand b)
              !c(x)+c(1)
              c ^ x

c ^ a ^ b

Thus with Simplification we get: Sum = c ^ a ^ b

- Cout = !abc+a!bc+ab!c+abc

  c(!ab+a!b)+ab(!c+c)

  c(a ^ b)+ab

  Thus with Simplification we get: Cout = c(a ^ b) + ab

- With this knowledge of how a Full Adder works we can create it in Verilog with the proper boolean expressions. In order to find how many Full Adders we will need we use the expression: n-bit = (n-1)Full_Adders.
  - 4-Bit Adder:

    4-bit = (4-1)Full_Adders

    3Full_Adders
- By wiring the Cout of each adder into the next we create a ripple-carry adder with the sums being the outputs necessary for our next subsection (n0,n1,n2,n3).

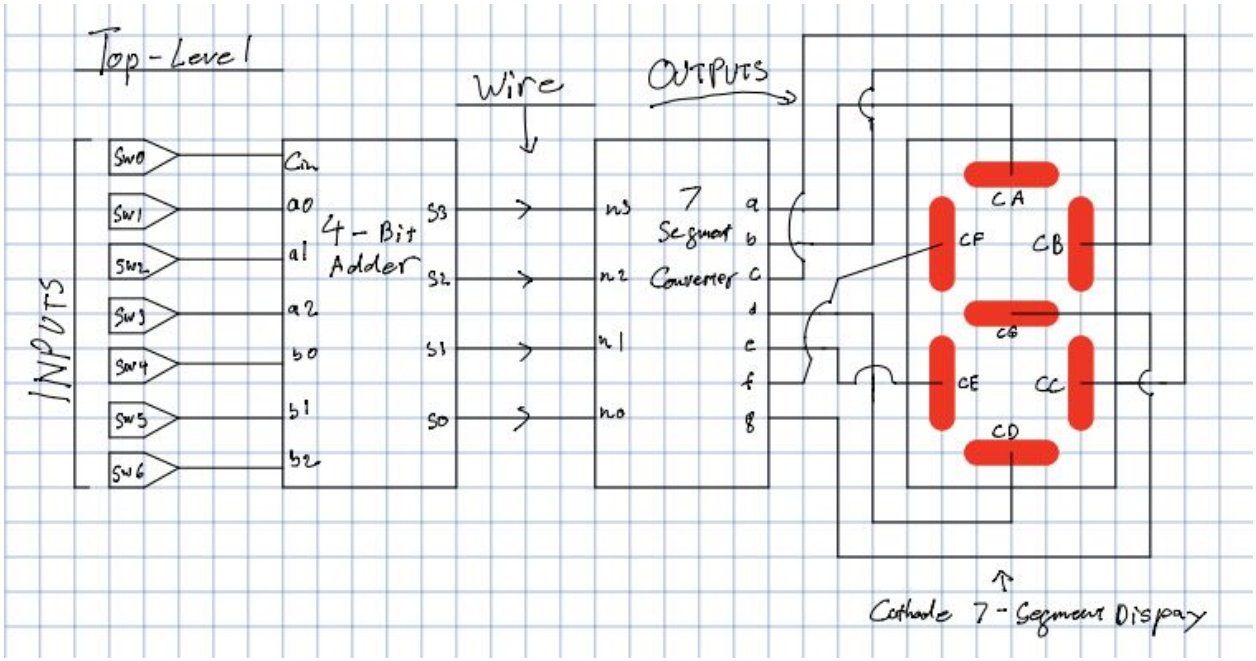
- <u>Seven-Segment Converter:</u>
  First let's take a look at the truth table:

n3(d)	n2(c)	n1(b)	n0(a)	a	b	c	d	e	f	g
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

- This is a 4-variable truth table for the 16hex values available on a seven segment display. CA - CG represents each segment of the Cathode Segment Display. There is a DP segment but that will be unused for this design. In order to figure out active high and lows of each segment we go through each hex value and determine whether the segment will be on or off. For example, when we want the value "1" we would need to have the segments 'CB' and 'CC' on therefore making them the lows(0s) for a cathode diode.
- With this truth table we can acquire a boolean expression for each segment to make the 16Hex values (n0,n1, n2, n3 == a, b, c, d):
    - a = a!b!c!d + !abc!d + ab!cd + a!bcd
    - b = a!bc!d + !abc!d + ab!cd + !a!bcd + !abcd + abcd
    - c = !ab!c!d + !a!bcd + !abcd + abcd
    - d = a!b!c!d + !a!bc!d + abc!d + a!b!cd + !ab!cd + abcd
    - e = a!b!c!d + ab!c!d + !a!bc!d + a!bc!d + abc!d + a!b!cd
    - f = a!b!c!d + !ab!c!d + ab!c!d + abc!d + a!bcd
    - g = !a!b!c!d + a!b!c!d + abc!d + !a!bcd
- We can input this boolean expression into the converter by taking in the 4 outputs and wiring each output to its respective segment. We will do this in the Top_Level Diagram as shown below.

- <u>Top-Level:</u>



- As you can see in the diagram, the Top-Level connects all the components we designed to enable the 7-segment display to properly function with the 7 switches as inputs.

- **Testing & Simulation:**
  In order to test my design I used two different resources. The first resource is by running a simulation directly from the Vivado application. Using the provided testbench file, I entered in the 16 different cases spaced between each 100ns. By spacing out each test by 100ns, I can clearly distinguish each switch change made, and the corresponding highs and lows of the output. Another source was by programming the bitstream to the Basys3 board. This method is a more efficient method of testing, as there is not too much going on with the Top_Level and there are 7 switches to interact with to get different values.

- **Results:**
  - Document which pins of the FPGA you used and how they were connected to the switches and 7-segment displays.
    - As shown above in the top_level diagram I used switches sw[6:0] as the inputs to the adder and wires t[3:0] to connect the 4-bit adder to the 7-segment converter. The converter outputs CA - CG to the seven segment display. In order to activate the displays AN0 is set to 0, while AN1, 2, 3, and DP are set to 1.
  - Determine the longest path from any input to any output in your 3-bit adder. (i.e. the highest number of gates that any input goes through before reaching an output.)

        - The longest path for an input to reach it's. output in a 4-bit adder is 9 logic gates.
    - For an *n*-bit adder determine the length of the longest path from any input to any output (in terms of *n*).
        - Path = n-bit * 3gates
            - The longest path lies with the Cin which adds 3 gates (AND, OR, OR) per adder.
    - Calculate the number of possible input values (7-bit vectors) for your adder? Give the percentage of these that you tested in your simulation.
        - In a 7-bit vector there are 128 possible input values/combinations which stems from the equation 2^n-bit
            - Since we only did 15 tests (15/128)*100 = 11.72% of tests were conducted on the simulation

- **Conclusion**
    - From this lab I learned how to manage the hierarchy in Verilog, which helped with designing the Top_Level by only calling components to connect together with wires. I learned how to create n-bit ripple carry adders, and how to create a seven segment converter. If I was to do this lab again, I would definitely set a better schedule for myself. I ended up turning the lab in one day late because I didn't ask enough questions on the Tuesday section. This caused me to get stuck on understanding how to use xilinx rather than understanding the main design of this lab. Although, I turned the lab in one day late, I definitely had very cleanly written code and would only need to use Buses to optimize this design.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/13/2020 06:08:33 PM
// Design Name:
// Module Name: Full_Adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Full_Adder(
    input A,
    input B,
```

```
    input Cin,
    output S,
    output Cout
    );

    assign S = A ^ (B ^ Cin);
    assign Cout = (B & Cin)|(A & Cin)|(A & B);

endmodule
```

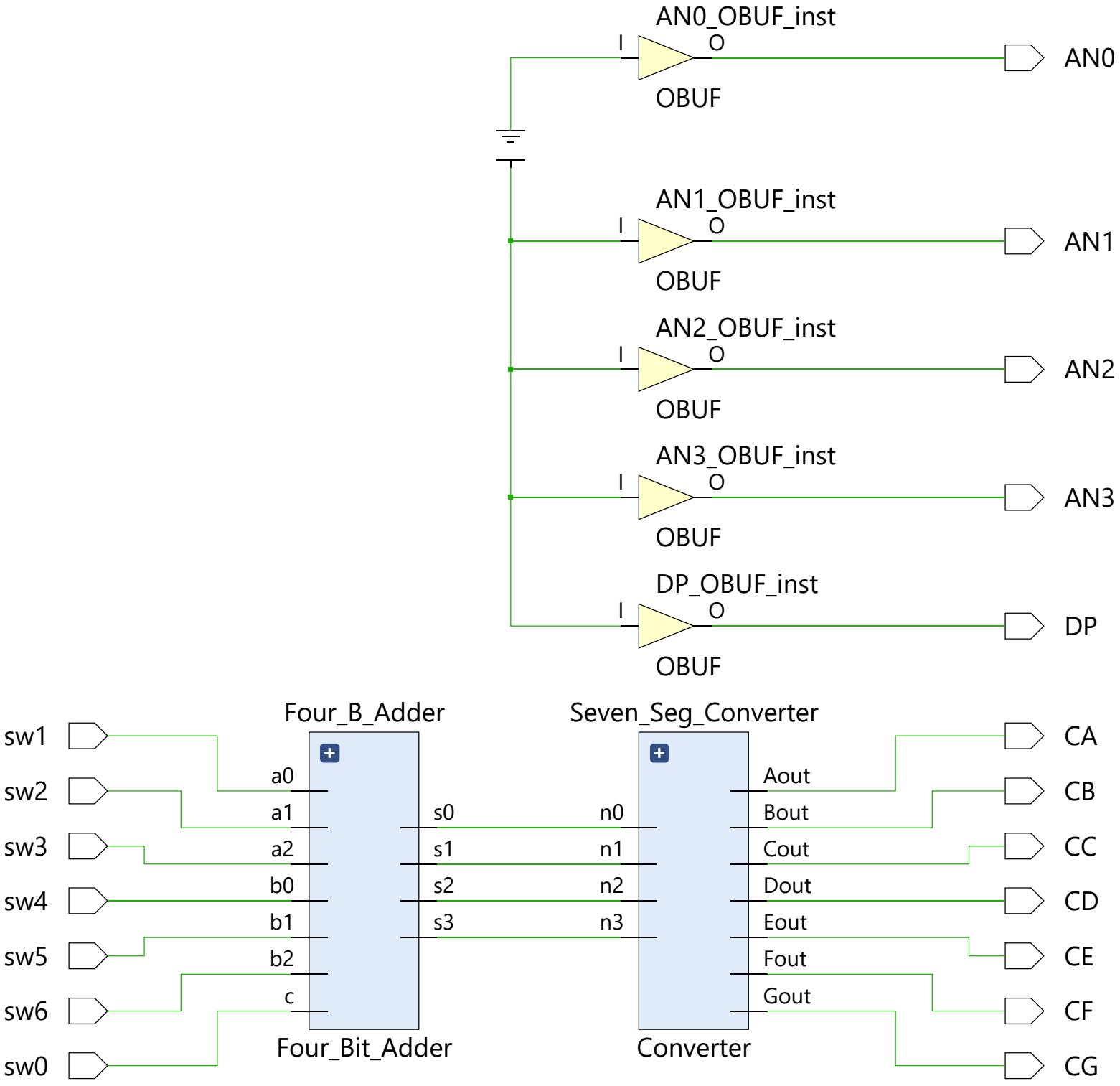
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/13/2020 11:14:19 PM
// Design Name:
// Module Name: 4-Bit_Adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Four_Bit_Adder(
    input c,
    input a0,
```

```
    input b0,
    input a1,
    input b1,
    input a2,
    input b2,
    output s0,
    output s1,
    output s2,
    output s3
    );

    wire t0, t1;

    Full_Adder Adder1(.A(a0), .B(b0), .Cin(c),
.S(s0), .Cout(t0));
    Full_Adder Adder2(.A(a1), .B(b1),
.Cin(t0), .S(s1), .Cout(t1));
    Full_Adder Adder3(.A(a2), .B(b2),
.Cin(t1), .S(s2), .Cout(s3));



endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/14/2020 11:41:22 PM
// Design Name:
// Module Name: Converter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Converter(
    input n0,
    input n1,
```

```
    input n2,
    input n3,
    output Aout,
    output Bout,
    output Cout,
    output Dout,
    output Eout,
    output Fout,
    output Gout
    );

    assign Aout = (n0 & ~n1 & ~n2 & ~n3)|(~n0
& ~n1 & n2 & ~n3)|(n0 & n1 & ~n2 & n3)|(n0 &
~n1 & n2 & n3);
    assign Bout = (n0 & ~n1 & n2 & ~n3)|(~n0 &
n1 & n2 & ~n3)|(n0 & n1 & ~n2 & n3)|(~n0 & ~n1
& n2 & n3)|(~n0 & n1 & n2 & n3)|(n0 & n1 & n2
& n3);
    assign Cout = (~n0 & n1 & ~n2 & ~n3)|(~n0
& ~n1 & n2 & n3)|(~n0 & n1 & n2 & n3)|(n0 & n1
& n2 & n3);
    assign Dout = (n0 & ~n1 & ~n2 & ~n3)|(~n0
& ~n1 & n2 & ~n3)|(n0 & n1 & n2 & ~n3)|(n0 &
~n1 & ~n2 & n3)|(~n0 & n1 & ~n2 & n3)|(n0 & n1
& n2 & n3);
    assign Eout = (n0 & ~n1 & ~n2 & ~n3)|(n0 &
n1 & ~n2 & ~n3)|(~n0 & ~n1 & n2 & ~n3)|(n0 &
```

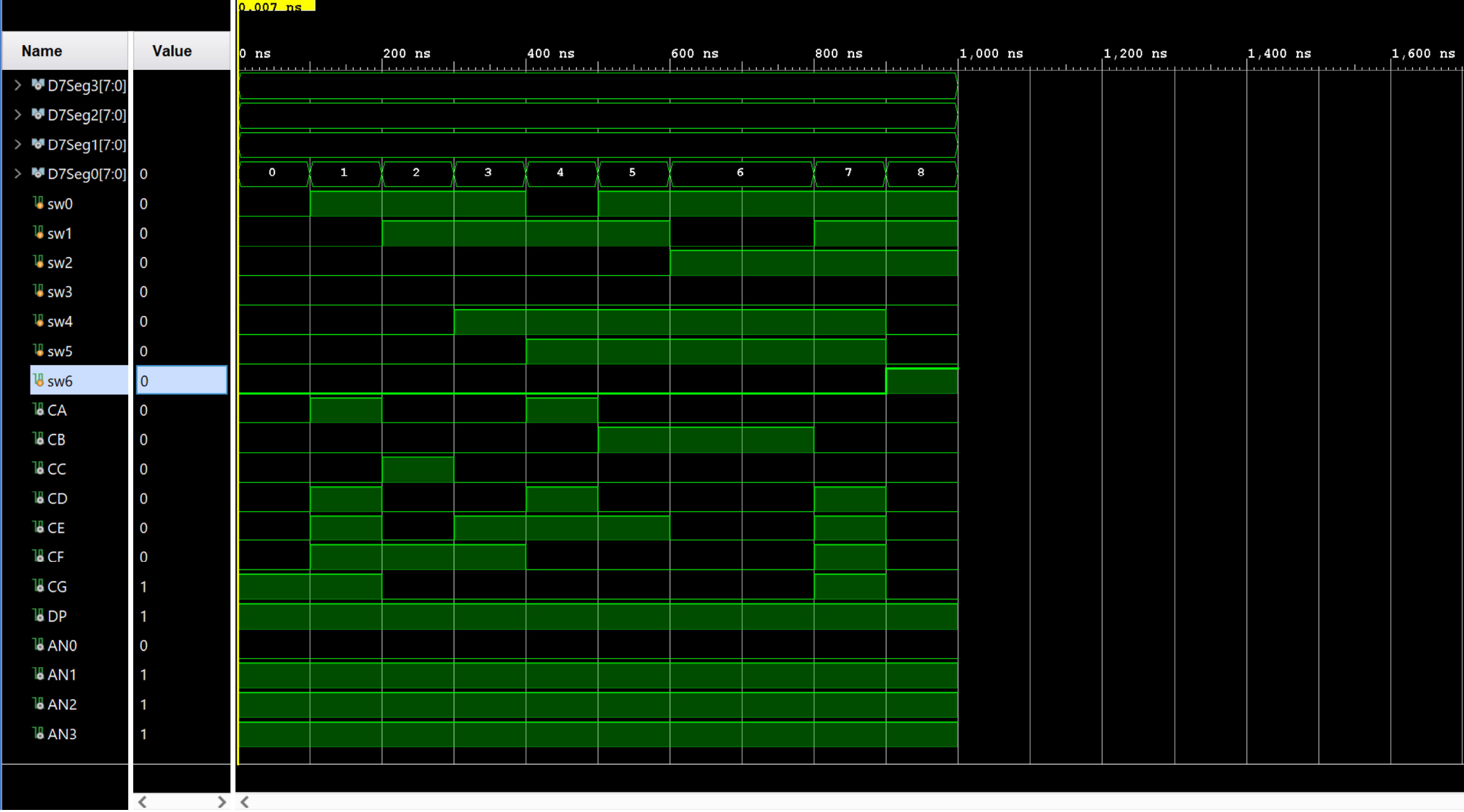
```
~n1 & n2 & ~n3)|(n0 & n1 & n2 & ~n3)|(n0 & ~n1
& ~n2 & n3);
    assign Fout = (n0 & ~n1 & ~n2 & ~n3)|(~n0
& n1 & ~n2 & ~n3)|(n0 & n1 & ~n2 & ~n3)|(n0 &
n1 & n2 & ~n3)|(n0 & ~n1 & n2 & n3);
    assign Gout = (~n0 & ~n1 & ~n2 & ~n3)|(n0
& ~n1 & ~n2 & ~n3)|(n0 & n1 & n2 & ~n3)|(~n0 &
~n1 & n2 & n3);


endmodule
```


AN0_OBUF_inst
I
O
OBUF
AN0
AN1_OBUF_inst
I
O
OBUF
AN1
AN2_OBUF_inst
I
O
OBUF
AN2
AN3_OBUF_inst
I
O
OBUF
AN3
DP_OBUF_inst
I
O
OBUF
DP
Four_B_Adder
sw1
sw2
sw3
sw4
sw5
sw6
sw0
a0
a1
a2
b0
b1
b2
c
s0
s1
s2
s3
Four_Bit_Adder
Seven_Seg_Converter
n0
n1
n2
n3
Aout
Bout
Cout
Dout
Eout
Fout
Gout
Converter
CA
CB
CC
CD
CE
CF
CG

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/15/2020 01:19:26 AM
// Design Name:
// Module Name: Main
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Main(
    input sw0,
    input sw1,
```

```
input sw2,
input sw3,
input sw4,
input sw5,
input sw6,
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 DP = 1;
assign AN0 = 0;
assign AN1 = 1;
assign AN2 = 1;
assign AN3 = 1;

wire t0, t1, t2, t3;
```

```
    Four_Bit_Adder Four_B_Adder(.c(sw0),
.a0(sw1), .b0(sw4), .a1(sw2), .b1(sw5),
.a2(sw3), .b2(sw6), .s0(t0), .s1(t1), .s2(t2),
.s3(t3));
    Converter Seven_Seg_Converter(.n0(t0),
.n1(t1), .n2(t2), .n3(t3), .Aout(CA),
.Bout(CB), .Cout(CC), .Dout(CD), .Eout(CE),
.Fout(CF), .Gout(CG));

endmodule
```


0.007 ns
Name
Value
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
1,400 ns
1,600 ns
D7Seg3[7:0]
D7Seg2[7:0]
D7Seg1[7:0]
D7Seg0[7:0] 0
sw0 0
sw1 0
sw2 0
sw3 0
sw4 0
sw5 0
sw6 0
CA 0
CB 0
CC 0
CD 0
CE 0
CF 0
CG 1
DP 1
AN0 0
AN1 1
AN2 1
AN3 1
0
1
2
3
4
5
6
7
8