Lab 2: Seven-Segment Display

Gabriel Dimas
April 15, 2022
Section C

Description

This lab was meant to introduce students to how to further use the Basys3 Board. Our goal was to use boolean algebra, logic gates, and full adders to enable the lights on one of the four seven segment displays to display hexadecimal values. Namely, to use seven input switches as the basis for a 3-bit adder whose outputs are the inputs for a seven-segment display. This lab served us to better understand Verilog programming, real-life applications to boolean algebra, designing logic circuits, and the wiring of the Basys3 board.

Design

To start the circuit design, we needed to make a truth table. We needed to map the seven input switches to three 3-bit full adders, and then create a Seven Segment converter which takes in those three inputs and outputs a hexadecimal value to the display that is readable. We started out with a design of the inputs[1].

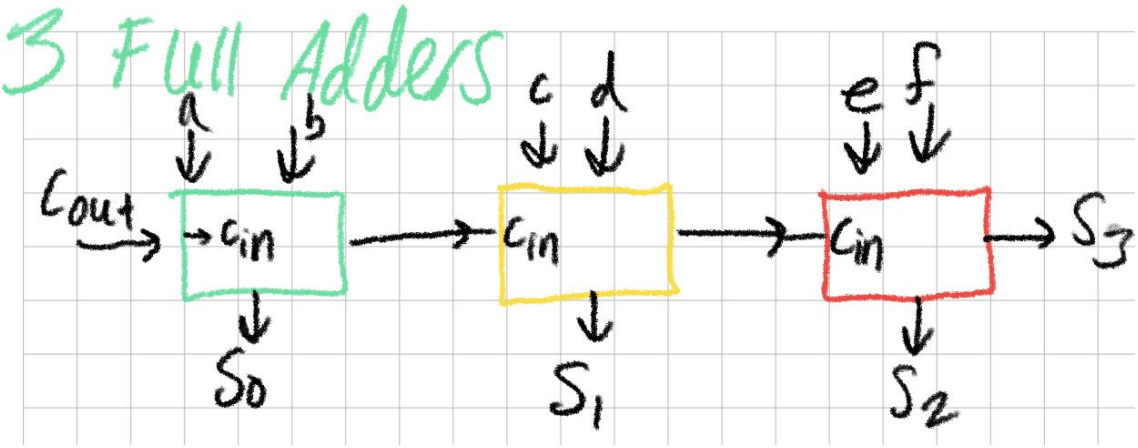


Figure 1

Here is a truth table created for one 3-bit full adder[2]:

Full Adder Truth Table

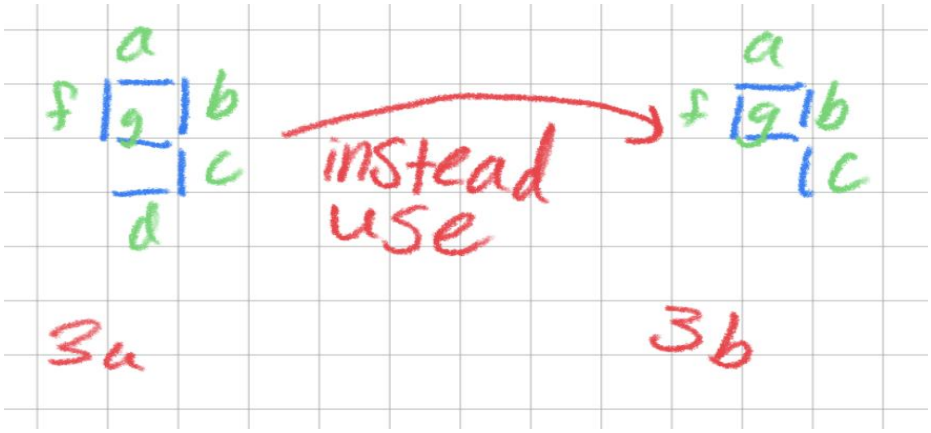
a	b	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 2

[1] See Appendix for the entire document. I created this adder in series to show how they need to be implemented.
[2] Again, See Appendix for the entire document. I created this adder in series to show how they need to be implemented.

Lab 2: Seven-Segment Display

With this design, we can create a truth table for the S0, S1, S2, S3 output values (which are mapped as n3, n2, n1, n0) which are then used as the inputs for the seven segment converter. Bear in mind that, according to the testbench[3], the LEDs needed to represent a '9' as lights a, b, c, f, g (which excludes the bottom LED from displaying, similar to ab upside down 'd').



Figures 3a and 3b

This is a different representation of '9' but still gives off the same meaning. After we created this truth table, we needed to design and implement three 3-bit adders in series according to the mapping in this figure[4]:

Switch	sw6	sw5	sw4	sw3	sw2	sw1	sw0
Input	b_2	b_1	b_0	a_2	a_1	a_0	c_{in}

Figure 4

The inputs of one full adder are the inputs in0, in1, and carry in. The outputs of one full adder are the out1 and the carry out (which is the carry-in for the next full adder in series). When we put three of these in series with each other, we get this figure[5] (mapped in inputs/output as the figure above):

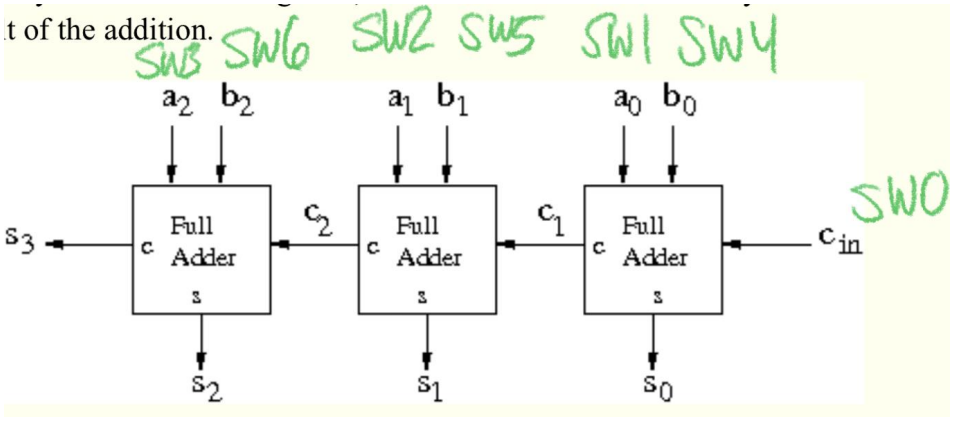


Figure 5

[3] More information about the testbench on the **Testing and Simulation** section.
[4] Courtesy of the lab document https://classes.soe.ucsc.edu/cse100/Spring22/lab/lab2/lab2.html.
[5] Courtesy of the lab document https://classes.soe.ucsc.edu/cse100/Spring22/lab/lab2/lab2.html.

Lab 2: Seven-Segment Display

The mapping for the seven-segment display was also done with a truth table according to the segments that needed to be lit for a specific hex value. For example, representing a '4' needed the LEDs b, c, f, g to be low[6] (although the truth table shows high because of simplicity, on the Verilog code, it is low).

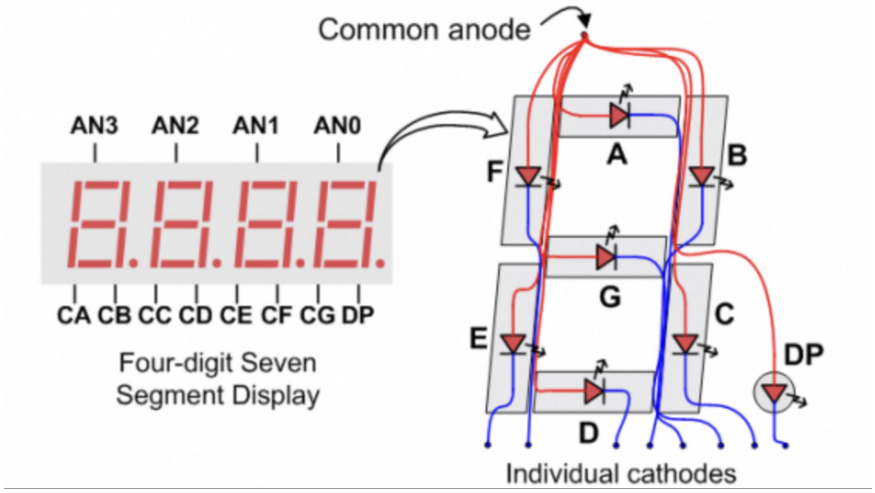


Figure 6

We also needed to bear in mind that the seven segment display was active low, so if we needed to light an LED, we would supply a low signal[7]. This goes for all the lights on the segment board.

After this, we needed to design seven segment converter. This converter takes the inputs from the series of full adders and maps them into the binary values needed to light up the lights. We take the inputs of n3, n2, n1, n0 (in that order) and find which exact value it would represent from the truth table. For example, if our n values correspond to 0110, then our light schematic would be a=0, b=1, c=0, d=0, e=0, f=0, which represents a '6' on the seven segment display. Here is the truth table for how I mapped the n values and segments values[8].

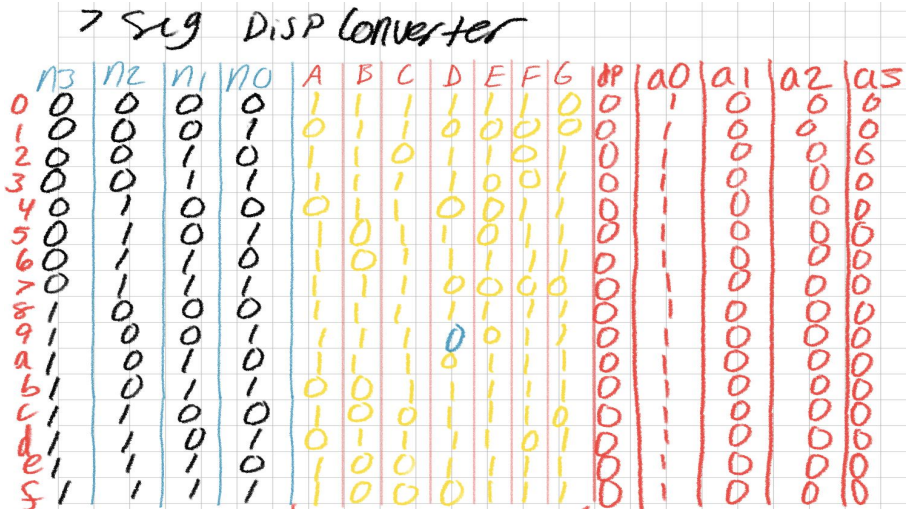


Figure 7

[6] Photo taken from Basys3 Board Reference Manual:
https://digilent.com/reference/programmable-logic/basys-3/reference-manual?redirect=1
[7] Photo taken from Basys3 Board Reference Manual:
https://digilent.com/reference/programmable-logic/basys-3/reference-manual?redirect=1
[8] See **Appendix** for the entire document. I created this truth table to see how to light up the seven-segment display.

Lab 2: Seven-Segment Display

After we finally got our truth tables and mappings of values, we were able to write our program in the Verilog software. We started by creating a module for the full adder and connected three of them in series using the **wire** keyword[9]. Here are those Verilog-generated schematics as well as a snippet of code used[10].

```
wire w0, w1;    //src3
wire f0, f1, f2, f3;
assign AN0 = 0, AN1 = 0, AN2 = 0, AN3 = 0, DP = 0;

Full_Adder ad1 (.c_in(sw0), .in0(sw1), .in1(sw4), .s(f3), .c_out(w0));  //src2
Full_Adder ad2 (.c_in(w0), .in0(sw2), .in1(sw5), .s(f2), .c_out(w1));
Full_Adder ad3 (.c_in(w1), .in0(sw3), .in1(sw6), .s(f1), .c_out(f0));

SegDisp_Convert SevSeg_Convert(.a(f0), .b(f1), .c(f2), .d(f3),
 .A(CA), .B(CB), .C(CC), .D(CD), .E(CE), .F(CF), .G(CG));
```

Figure 8

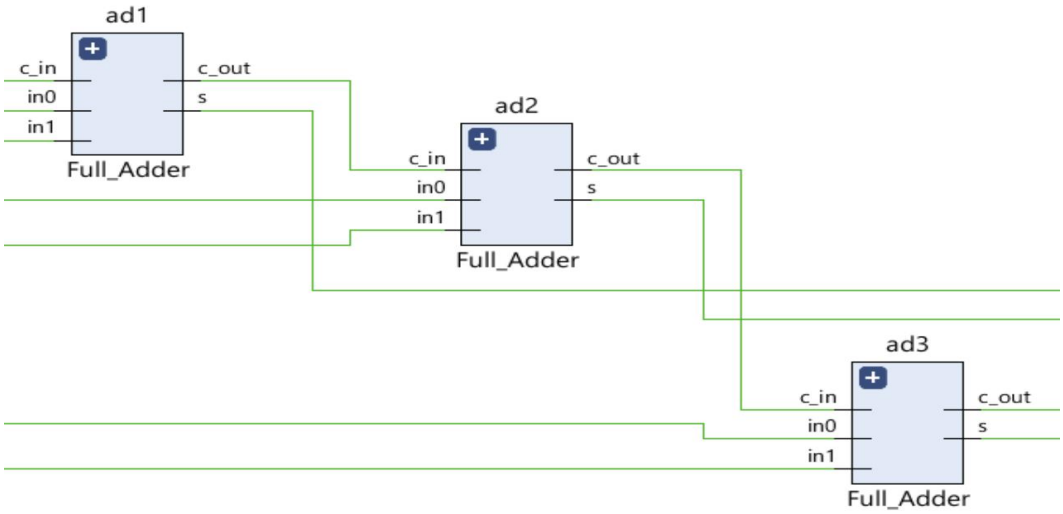


Figure 9

After creating this and connecting three full adders together, create the seven-segment display converter. We took the four output wires from our **Full_Adder** module and used them as the 4-bit inputs to our converter (see Figure 7). For example, here is a snippet of how the value of 'a' on the seven-segment display would be displayed. It is dependent directly on how the four inputs behaved[11].

```
assign A = !a&!b&!c&d | //0001
           !a&b&!c&!d | //0100
           a&!b&c&d   | //1011
           a&b&!c&d   ; //1101
```

Figure 10

[9] **Wire** keyword knowledge came from external source: ttps://www.chipverify.com/verilog/verilog-if-else-if

[10] Adding 3 Fuller Adders in series came from https://classes.soe.ucsc.edu/cse100/Spring22/lab/hierarchy/hierarchy.html

[11] See **Appendix** for the rest of the Verilog code

Lab 2: Seven-Segment Display

This sort of format is the same for LEDs ‘b’ through ‘g’, where ‘a’-’d’ are the full adder outputs and ‘A’-’G’ are the seven-segment LED outputs. And here is the Verilog schematic of the entire module with the full adders and the Seven-Segment Converter.

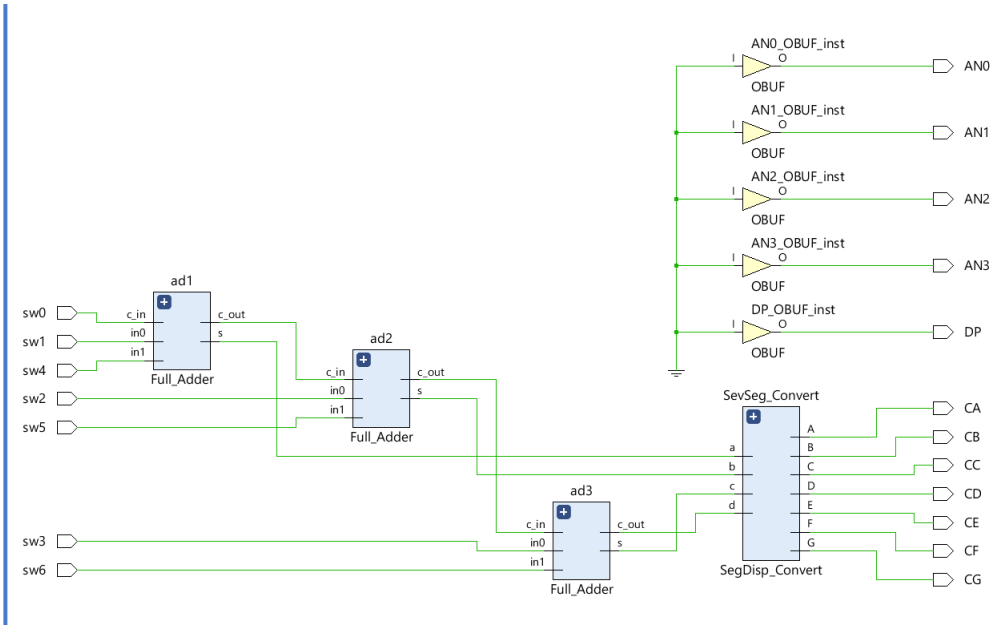


Figure 11

Testing & Simulation

To test our output, we used the provided testbench[12] given by our professor. For this to work, we needed to manipulate the testbench so that its parameters matched the ones created in our original top module. Although, we ran into a problem during this stage. When we got our full adders and seven-segment converter to work, we ran into a problem with the simulation stage. We found that the hexadecimal values displayed correctly for all values ‘0’ - ‘8’ and ‘A’ - ‘F’, except for ‘9’. During our simulation and many trials, we kept getting this same output for the testbench simulation, but it worked out flawlessly on the Basys3 Board (i.e the board would correctly display a ‘9’ for the correct inputs). We thought maybe this was an issue with our full adder. So we went back and made a testbench for our full adder. We found there were no errors. We then went to our top module and looked at the wired connection from the full adders to the seven-segment converter. We also found there was no problem there.

But then I looked at my board and noticed that there is more than one way to represent a ‘9’ digit. When first creating the seven-segment converter, we did so with the previous recollection of a display lighting the LEDs ‘a’, ‘b’, ‘c’, ‘d’, ‘f’, ‘g’, which gives the user that ‘9’ look. But then I realized that you can also represent a ‘9’ without the ‘d’ LED lit, so it would just look like ‘a’, ‘b’, ‘c’, ‘f’, ‘g’, which takes the form of an upside-down ‘d’ (see **Figure 3**). After some trials and errors, I was able to add this change to the seven-segment converter. All I needed to do was look at my truth table and add a line for row 9 in the converter for ‘D’ (see the blue ‘0’ in the middle of **Figure 7**). After making this change, the simulation was able to work as intended, and the updates were made to the Basys3 board to work as intended, which now reflected this upside-down ‘d’ (see **Figures 3a and 3b**).

[12]See **Appendix** for entire test bench. Here is where the source is located: https://classes.soe.ucsc.edu/cse100/Spring22/lab/simulate/ug900-vivado-logic-simulation-minimized.pdf

Lab 2: Seven-Segment Display

Results

All-in-all we were finally able to get the correct output. With that slight correct of displaying a different '9' we were finally able to achieve our goal. In addition to that, we were able to add additional test cases to the provided testbench in order to test the hex values 'A' - 'F'. This part also allowed us to ensure that our full adders and converter were working properly. After some trials and errors and the creation of some testbenches for some of the modules, the lab was able to come together and work seemlessly. All modules output their intended values, and the displays output their intended values as well.

Questions

1. Used Pin: V17, sw[0], V16 sw[1], W16 sw[2], W17 sw[3], W15 sw[4], V15 sw[5], W14 sw[6]. These pins are used as inputs for the **Full_Adder** module, which is fed to the **Seven-Segment_Converter** Module[13], which were then outputs to the Seven-Segment Display. Using the constraint file, the top module was mapped to the outputs of the constraint file, which were then used in the test benches built to test the module's functionality.
2. The conclusion is that sw[0] would be the longest path from input to output because sw[0] is the initial carry-in bit before we get to any actual output. For example, if three full adders are in series, then then the sw[0] would be the first to evaluate. Because of this, the entirety of the full adders need to be evaluated before we can get any type of output (i.e the 4-bit output we need for the converter stage).
3. For an n-bit adder, the longest path would be $2^n - 1$ because for any n inputs, we would also need to mind the carry in bit. In the case of this lab, the input was 2-bits, so $2^2 - 1 = 3$, which is how many equations were used: assign s = c_in^ in0^ in1;
4. There are $2 * 2 * 2 * 2 * 2 * 2 * 2 = 128$ possible inputs for this adder. For this lab, about 15 of 128 or 11.718% were tested for accuracy of 100% (± a few percent for computational errors with float values)

Conclusion

Working with the seven-segment display, there are many applications to it. Namely, they are able to have hexadecimal values assigned to them and are able to work well with multiple synchronized together, especially when there are multiple full adders together. There are also many applications to full adders. In this lab, three series of full adders were used to add six bits of information together. This real-world application works in today's daily life with smartphones, calculators, and timers, just to name a few. In addition to full adders, there is a better understanding of how Seven Segment Converters are used to convert 4 bits of binary into seven bits for the seven-segment display.

It's important to know how wires in the Basys3 are implemented when using Verilog software in conjunction. When working with wires, the programmer needs to keep in mind that these wires are inputs or outputs and that they are not variables. Variables have a place in memory, wires only hold signals and are not used for any type of holder. While working with Verilog, it's also important to know that test benches use register **reg** as outputs to simulate a

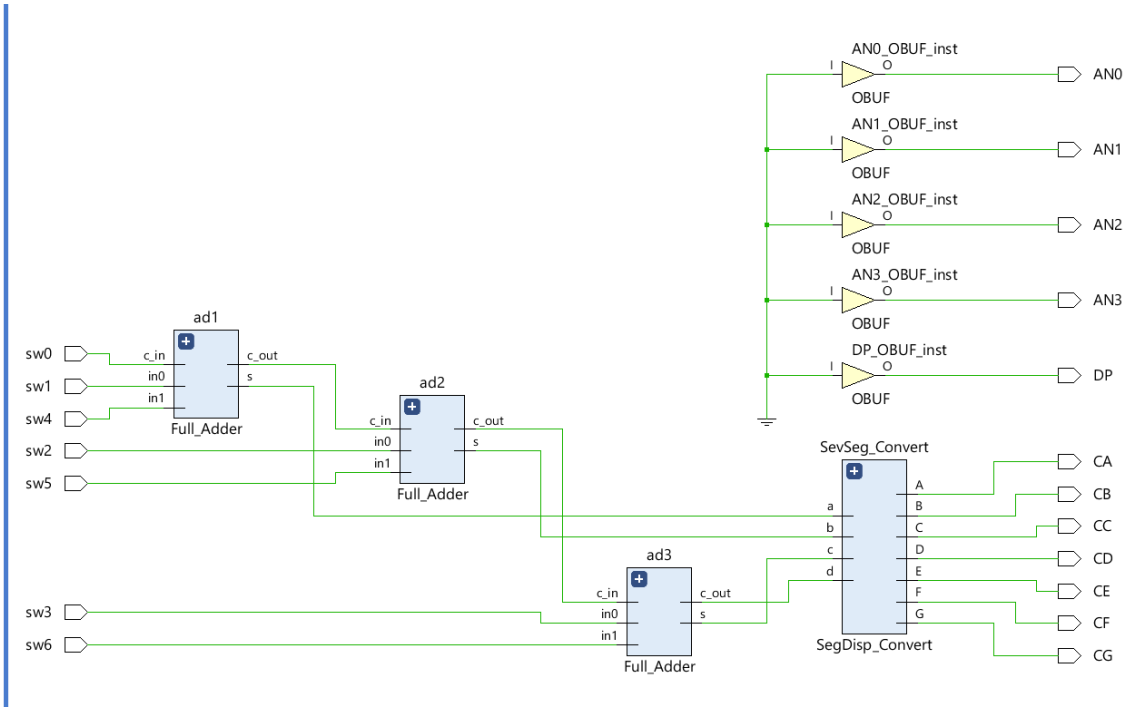
[13] See Appendix for full code

Lab 2: Seven-Segment Display

design. All-in-all, seven-segment converters are used to convert 4-bit inputs to seven LEDs which show numbers in a hexadecimal value, and how they are used for real-world applications.

Appendix

Schematic For Full Adder and Seven Segment Converter **Figure 11**

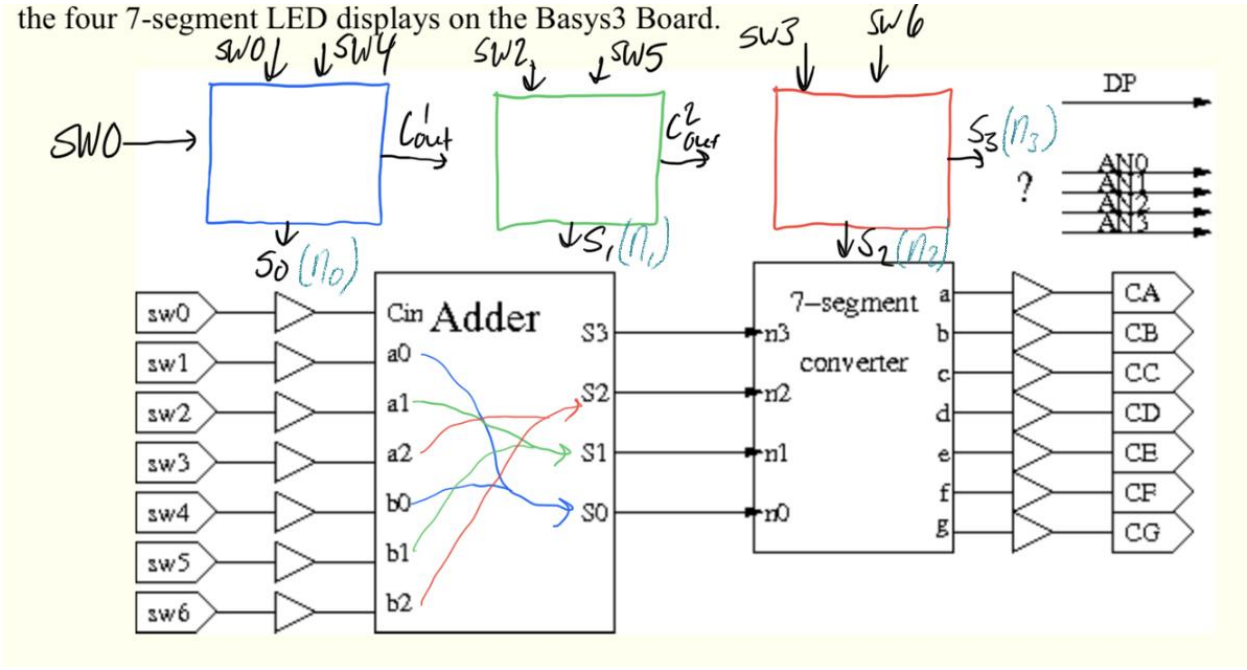


Top Header Module

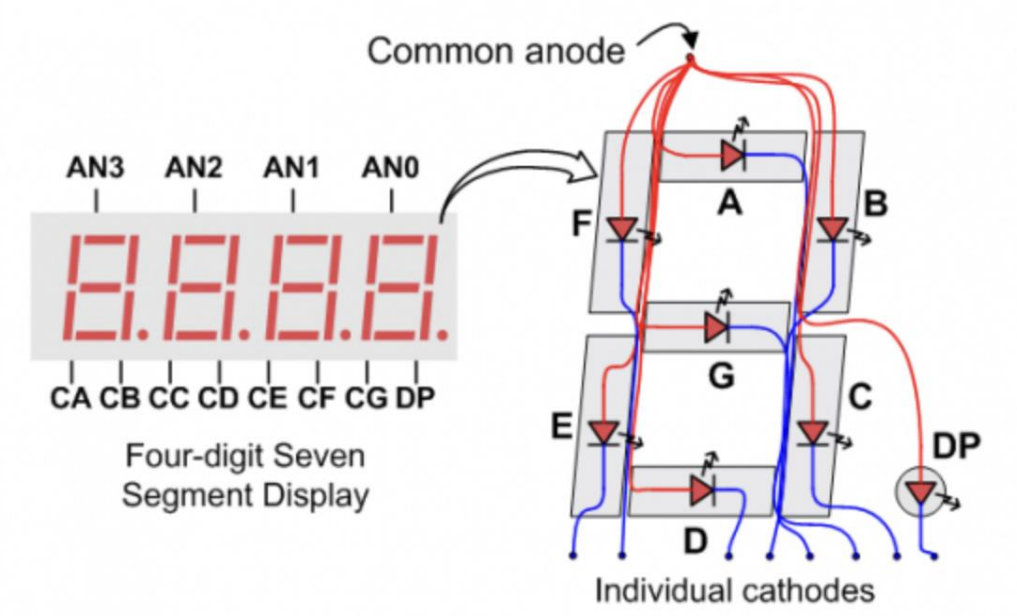
```
22
23  module top_adder(
24      input sw0,
25      input sw1,
26      input sw2,
27      input sw3,
28      input sw4,
29      input sw5,
30      input sw6,
31      output CA, output CB, output CC, output CD, output CE, output CF, output CG, output DP,
32      output AN0, output AN1, output AN2, output AN3);
33
34      wire w0, w1;    //src3
35      wire f0, f1, f2, f3;
36      assign AN0 = 0, AN1 = 0, AN2 = 0, AN3 = 0, DP = 0;
37
38      Full_Adder ad1 (.c_in(sw0), .in0(sw1), .in1(sw4), .s(f3), .c_out(w0));  //src2
39      Full_Adder ad2 (.c_in(w0), .in0(sw2), .in1(sw5), .s(f2), .c_out(w1));
40      Full_Adder ad3 (.c_in(w1), .in0(sw3), .in1(sw6), .s(f1), .c_out(f0));
41
42      SegDisp_Convert SevSeg_Convert(.a(f0), .b(f1), .c(f2), .d(f3),
43       .A(CA), .B(CB), .C(CC), .D(CD), .E(CE), .F(CF), .G(CG));
44
45  endmodule
46
```

Lab 2: Seven-Segment Display

Full Adder and Segment Converter



Seven Segment Diagram

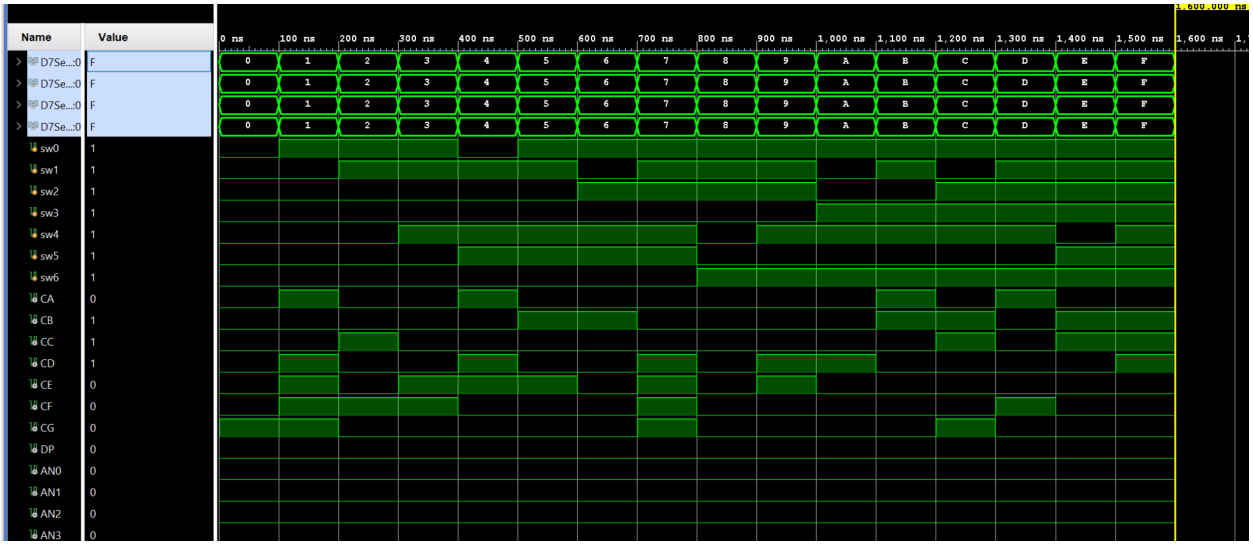


Lab 2: Seven-Segment Display

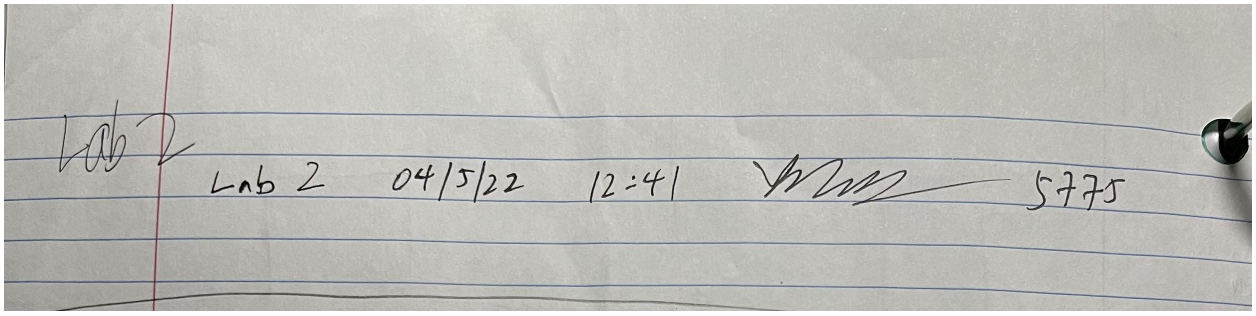
How to create an instance of a module[14].

```
wire t0, t1;

mux2to1 m1 (.s(s0), .i0(i0), .i1(i1), .y(t0));
mux2to1 m2 (.s(s0), .i0(i2), .i1(i3), .y(t1));
mux2to1 m3 (.s(s1), .i0(t0), .i1(t1), .y(f));

endmodule
```

Seven Segment Testbench Output



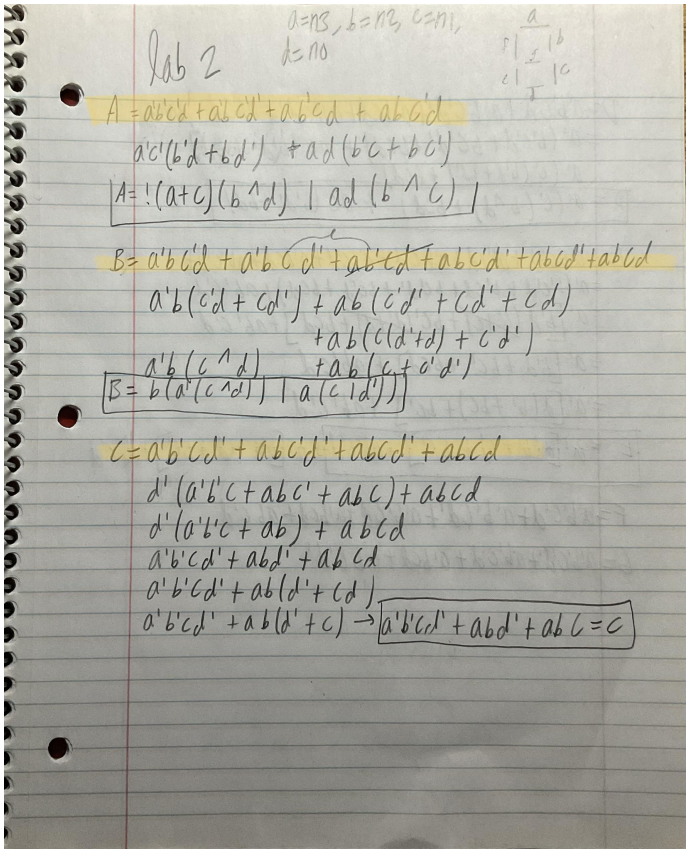
Code For Check Off: 5775

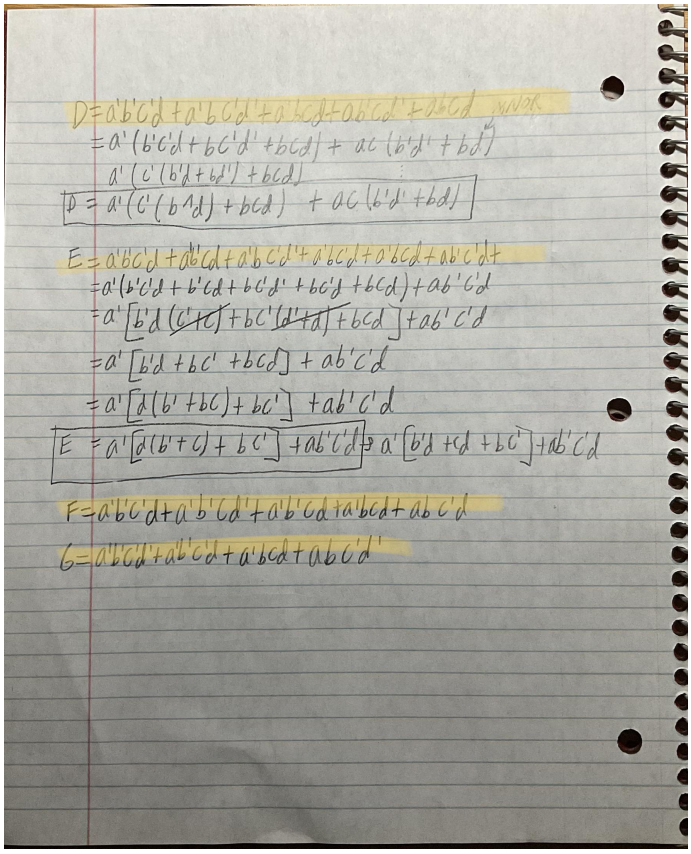


[14] Source: https://inst.eecs.berkeley.edu/~cs150/Documents/Nets.pdf

Lab 2: Seven-Segment Display

Functions for Seven-Segment Display

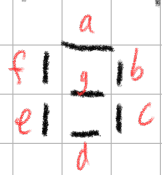




The next pages are the supplementary documents. Here are **Figures 1, 2, 3a, 3b and 7.** Additionally below that, the Verilog code shown in **Figures 8, 9, 10, and 11.**

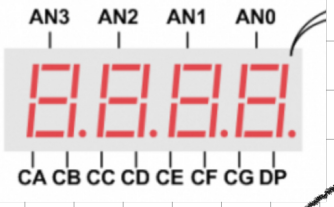
Full Adder Truth Table

a	b	Cin	S	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



Supplimentary to Lab 2

selector



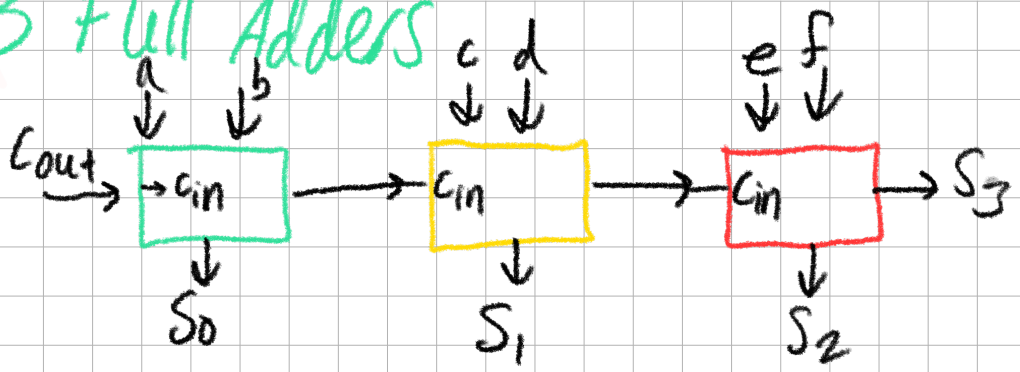
digit to disp

7 Seg Disp Converter

n3	n2	n1	n0	A	B	C	D	E	F	G	dp	a0	a1	a2	a3
0	0	0	0	1	1	1	1	1	1	0	0	1	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0
2	0	0	1	0	1	0	1	1	0	1	0	1	0	0	0
3	0	0	1	1	1	1	1	0	0	1	0	1	0	0	0
4	0	1	0	0	1	1	0	0	1	1	0	1	0	0	0
5	0	1	0	1	0	1	1	0	1	1	0	1	0	0	0
6	0	1	1	0	0	1	1	1	1	1	0	1	0	0	0
7	0	1	1	1	1	1	0	0	0	0	0	1	0	0	0
8	1	0	0	1	1	1	1	1	1	1	0	1	0	0	0
9	1	0	0	1	1	1	0	0	1	1	0	1	0	0	0
a	1	0	1	0	1	1	0	1	1	1	0	1	0	0	0
b	1	0	1	1	0	0	1	1	1	1	0	1	0	0	0
c	1	1	0	0	1	0	0	1	1	0	0	1	0	0	0
d	1	1	0	1	0	1	1	1	0	1	0	1	0	0	0
e	1	1	1	0	0	0	1	1	1	1	0	1	0	0	0
f	1	1	1	1	0	0	0	1	1	1	0	1	0	0	0

only need

3 Full Adders



Seven Switches

↓

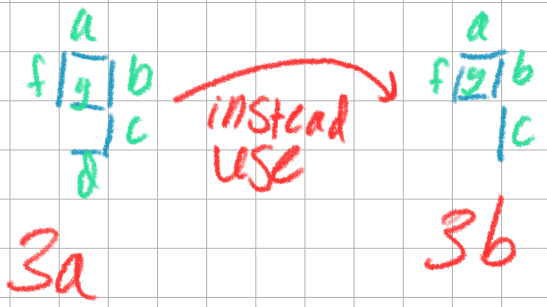
3 Full adders (7 input)(4 out)

↓

4bit → 7 bit Converter

↓

display



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:

module top_adder(
    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);

    wire w0, w1;    //src3
    wire f0, f1, f2, f3;
    assign AN0 = 0, AN1 = 1, AN2 = 1, AN3 = 1, DP = 1;

    Full_Adder ad1 (.c_in(sw0), .in0(sw1), .in1(sw4), .s(f3), .c_out(w0));  //src2
    Full_Adder ad2 (.c_in(w0), .in0(sw2), .in1(sw5), .s(f2), .c_out(w1));
    Full_Adder ad3 (.c_in(w1), .in0(sw3), .in1(sw6), .s(f1), .c_out(f0));

    SegDisp_Convert SevSeg_Convert(.a(f0), .b(f1), .c(f2), .d(f3),
     .A(CA), .B(CB), .C(CC), .D(CD), .E(CE), .F(CF), .G(CG))
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:

module SegDisp_Convert(input a, input b, input c, input d,
    output A,
    output B,
    output C,
    output D,
    output E,
    output F,
    output G
    );
    assign A = !a&!b&!c&d | //0001
               !a&b&!c&!d | //0100
               a&!b&c&d   | //1011
               a&b&!c&d   ; //1101

    assign B = !a&b&!c&d  |
               !a&b&c&!d  |
               a&!b&c&d   |
               a&b&!c&!d  |
               a&b&c&!d   |
               a&b&c&d    ;

    assign C = !a&!b&c&!d |
               a&b&!c&!d  |
               a&b&c&!d   |
               a&b&c&d    ;

    assign D = !a&!b&!c&d |
```

```
            !a&b&!c&!d |
            !a&b&c&d   |
            a&!b&c&!d  |
            a&b&c&d    |
            a&!b&!c&d  ; //newly added

   assign E = !a&!b&!c&d |
            !a&!b&c&d  |
            !a&b&!c&!d |
            !a&b&!c&d  |
            !a&b&c&d   |
            a&!b&!c&d  ;

   assign F = !a&!b&!c&d |
            !a&!b&c&!d |
            !a&!b&c&d  |
            !a&b&c&d   |
            a&b&!c&d   ;

   assign G = !a&!b&!c&!d|
            !a&!b&!c&d |
            !a&b&c&d   |
            a&b&!c&!d  ;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/01/2022 08:21:15 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 c_in,
    input in0,
    input in1,
    output s,
    output c_out
    );
    assign s = c_in^ in0^ in1;  //^ = xor
    assign c_out = (in0 & in1) | (in0 & c_in) | (in1 &
```

```
c_in);
endmodule
```

```
// CSE 100/L Sprign 2022
// This is a testbench for the entire Lab 2 Project.
// If the top level module in your Lab 2 project is named
"top_lab2"
// and you used the suggested names for its inputs/outputs
then
// then it will run without modification.  Otherwise follow
the instructions
// in the comments marked "TODO" to modify this testbench
to conform to your project.
`timescale 1ns/1ps

module lab2_7Seg_testbench();

  wire [7:0] D7Seg3, D7Seg2, D7Seg1, D7Seg0;
  reg sw0, sw1, sw2, sw3, sw4, sw5, sw6;
  wire CA, CB, CC, CD, CE, CF, CG, DP, AN0, AN1, AN2, AN3;



//=========================Start interface your toplevel here
=========================
// TODO: replace "top_lab2" with the name of your top level
Lab 2 module.

  top_adder UUT (
    .sw0(sw0), .sw1(sw1), .sw2(sw2),.sw3(sw3), .sw4(sw4),
.sw5(sw5), .sw6(sw6),
    .CA(CA), .CB(CB), .CC(CC), .CD(CD), .CE(CE), .CF(CF),
.CG(CG), .DP(DP),
    .AN0(AN0), .AN1(AN1), .AN2(AN2), .AN3(AN3)
  );
// TODO: In the three lines above, make sure the pin names
```

```
match the names
// used for the inputs/outputs of your top level module.
For example, if you
// used "cin" rather than "sw0" in yout top level module,
then replace ".sw0(sw0)" with ".cin(sw0)"

//=========================Stop interface your toplevel here
=========================



  show_7segDisplay  showit (
    .seg({CG, CF, CE, CD, CC, CB, CA}),
    .DP(DP), .AN0(AN0), .AN1(AN1), .AN2(AN2), .AN3(AN3),
    .D7Seg0(D7Seg0), .D7Seg1(D7Seg1), .D7Seg2(D7Seg2),
.D7Seg3(D7Seg3)
  );


  // Start sequential portion
  initial
  begin

    sw0=1'b0;
    sw1=1'b0;
    sw2=1'b0;
    sw3=1'b0;
    sw4=1'b0;
    sw5=1'b0;
    sw6=1'b0;
  // sum is 0
    //-------------  Current Time:  0ns
    #100;  //This advances time by 100 units (ns in this
```

```
case)
      sw0 = 1'b1;
  // sum is 1
  // -------------  Current Time:  100ns
    #100;
    sw1 = 1'b1;
  // sum is 2
  // -------------  Current Time:  200ns
    #100;
    sw4 = 1'b1;
  // sum is 3
  // -------------  Current Time:  300ns
    #100;
    sw0 = 1'b0;
    sw5 = 1'b1;
  // sum is 4
  // -------------  Current Time:  400ns
    #100;
    sw0 = 1'b1;
  // sum is 5
  // -------------  Current Time:  500ns
    #100;
    sw1 = 1'b0;
    sw2 = 1'b1;
  // sum is 6
  // -------------  Current Time:  600ns
    #100;        //was 200
    sw1 = 1'b1;
  // sum is 7
  // -------------  Current Time:  700ns
    #100;
    sw4 = 1'b0;
    sw5 = 1'b0;
```

```
  sw6 = 1'b1;
// sum is 8
// -------------  Current Time:  800ns
  #100;
  sw4 = 1'b1;
// sum is 9
// -------------  Current Time:  900ns
  #100;
  sw1 = 1'b0;
  sw2 = 1'b0;
  sw3 = 1'b1;
// sum is 10a
// -------------  Current Time:  1000ns
  #100;
  sw1 = 1'b1;
// sum is 11b
// -------------  Current Time:  1100ns
  #100;
  sw1 = 1'b0;
  sw2 = 1'b1;
// sum is 12c
// -------------  Current Time:  1200ns
  #100;
  sw1 = 1'b1;
// sum is 13d
// -------------  Current Time:  1300ns
  #100;
  sw4 = 1'b0;
  sw5 = 1'b1;
// sum is 14e
// -------------  Current Time:  1400ns
  #100;
  sw4 = 1'b1;
```

```
  // -------------  Current Time:  1500ns
    #100;
    // TODO: complete this testbentch so that all 16 hex
values are generated
  end
endmodule




//==============Do not edit below this line
==========================================
module show_7segDisplay (
  input [6:0] seg,
  input DP,AN0,AN1,AN2,AN3,
  output reg [7:0] D7Seg0, D7Seg1, D7Seg2,D7Seg3);

  reg [7:0] val;

  always @(AN0 or val)
  begin
    if (AN0 == 0) D7Seg0 <= val;
    else if (AN0 == 1) D7Seg1 <= " ";
    else D7Seg0 <= 8'bX;   //  non-blocking assignment
  end

  always @(AN1 or val)
  begin
    if (AN1 == 0) D7Seg1 <= val;
    else if (AN1 == 1) D7Seg1 <= " ";
    else D7Seg1 <= 8'bX;   //  non-blocking assignment
  end
```

```
always @(AN2 or val)
begin
  if (AN2 == 0) D7Seg2 <= val;
  else if (AN2 == 1) D7Seg2 <= " ";
  else D7Seg2 <= 8'bX;   //  non-blocking assignment
end

always @(AN3 or val)
begin
  if (AN3 == 0) D7Seg3 <= val;
  else if (AN3 == 1) D7Seg3 <= " ";
  else D7Seg3 <= 8'bX;   //  non-blocking assignment
end

always @(seg)
case (seg)
7'b0111111:
     val = "-";
7'b1111111:
     val = " ";
7'b1000000:
     val = "0";
7'b1111001:
     val = "1";
7'b0100100:
     val = "2";
7'b0110000:
     val = "3";
7'b0011001:
     val = "4";
7'b0010010:
     val = "5";
7'b0000010:
```

```
      val = "6";
  7'b1111000:
      val = "7";
  7'b0000000:
      val = "8";
  7'b0011000:
      val = "9";
  7'b0001000:
      val = "A";
  7'b0000011:
      val = "B";
  7'b1000110:
      val = "C";
  7'b0100001:
      val = "D";
  7'b0000110:
      val = "E";
  7'b0001110:
      val = "F";
  default:
      val = 8'bX;
  endcase
endmodule
```