Lab 3 – Four bit, Seven Segment Decoder

Name: Robert Bara

Section #: 003

Date: 02/06/2020

Summary/Abstract

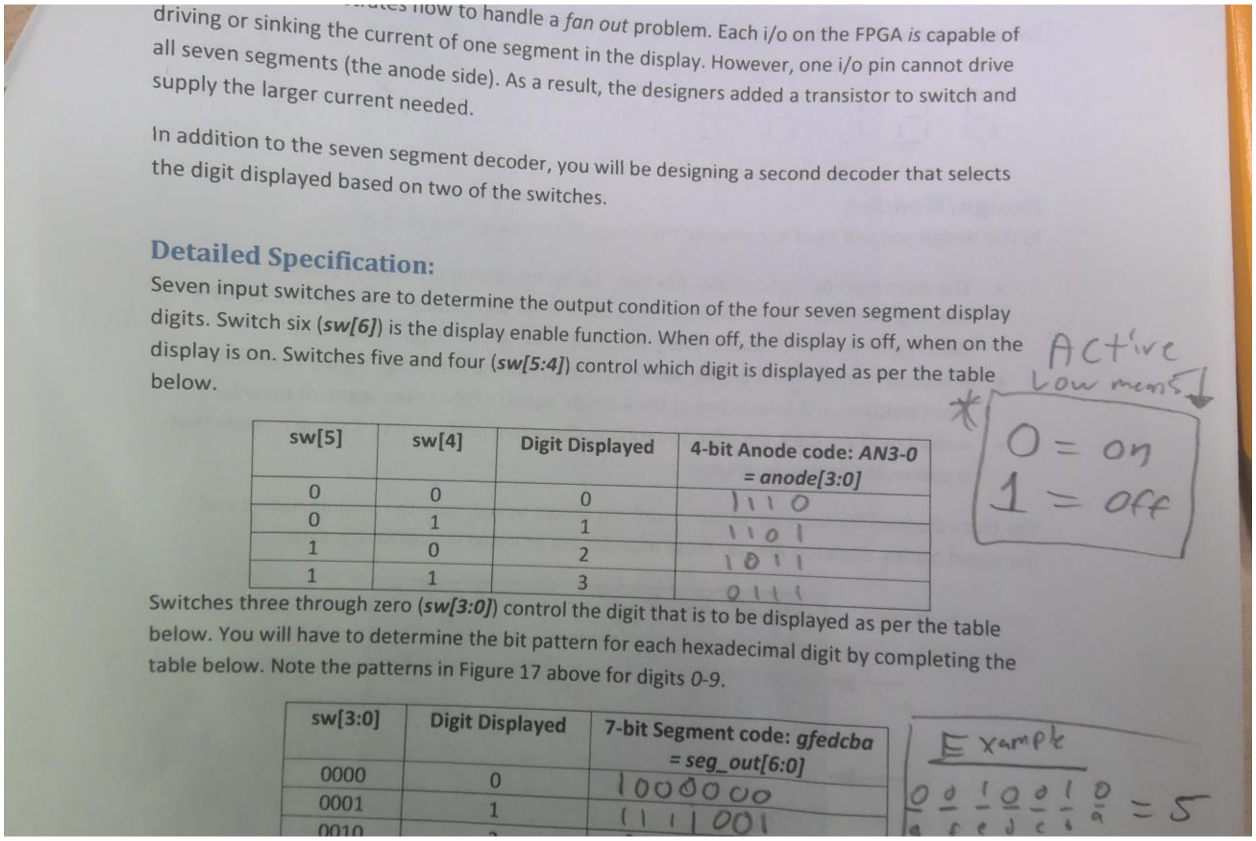
The idea of this lab is to create a seven-segment decoder by converting 4-bit hexadecimal code to a corresponding 7-bit code. This will be done by creating two modules and instantiating them into the lab3_decoder module. The running the .bit file will then allow the board to display hexadecimal characters 0-9, a-f on the anodes and cathodes of basys3 hardware board by using switches 0:6 and the corresponding truth tables.

Introduction

For this lab, essential background information includes knowledge of number systems such as binary to hexadecimal as well as the Verilog syntax for case statements, if statements, and instantiation. The four-bit, seven segment decoder is essential to future labs because it introduces the 7-segment display on the basys3 board which will be used for future labs, as well as applying Verilog case statements and always blocks, which can make the conditions from truth tables a lot easier and more efficient. It is useful to refer to the basys3 board's reference manual to understand how each switch will correspond to the Verilog code. The display is an active high, however the anodes and cathodes are an active low, meaning in the truth tables generated, 0=on and 1=off. Switches (0-3) will be used to describe the hexadecimal code, switches (4-5) will dictate which anode will be lit up, and switch 6 will turn the display on and off.

Procedure

Keeping in mind the active high vs active low as described in the introduction, the following truth tables can be generated for the 4-bit anode code and the 7-bit segment code:



how to handle a *fan out* problem. Each i/o on the FPGA *is* capable of driving or sinking the current of one segment in the display. However, one i/o pin cannot drive all seven segments (the anode side). As a result, the designers added a transistor to switch and supply the larger current needed.

In addition to the seven segment decoder, you will be designing a second decoder that selects the digit displayed based on two of the switches.

Detailed Specification:

Seven input switches are to determine the output condition of the four seven segment display digits. Switch six (***sw[6]***) is the display enable function. When off, the display is off, when on the display is on. Switches five and four (***sw[5:4]***) control which digit is displayed as per the table below.

Active low means ↓

*

$\boxed{\begin{array}{l} 0 = \text{on} \\ 1 = \text{off} \end{array}}$

sw[5]	sw[4]	Digit Displayed	4-bit Anode code: *AN3-0 = anode[3:0]*
0	0	0	1110
0	1	1	1101
1	0	2	1011
1	1	3	0111

Switches three through zero (***sw[3:0]***) control the digit that is to be displayed as per the table below. You will have to determine the bit pattern for each hexadecimal digit by completing the table below. Note the patterns in Figure 17 above for digits *0-9*.

sw[3:0]	Digit Displayed	7-bit Segment code: *gfedcba = seg_out[6:0]*
0000	0	1000000
0001	1	1111001
0010	2	

Example

$\underset{g}{0}\,\underset{f}{0}\,\underset{e}{1}\,\underset{d}{0}\,\underset{c}{0}\,\underset{b}{1}\,\underset{a}{0} = 5$

The truth table for the 7-bit segment code functions a bit strangely, using the reference from the lab manual,

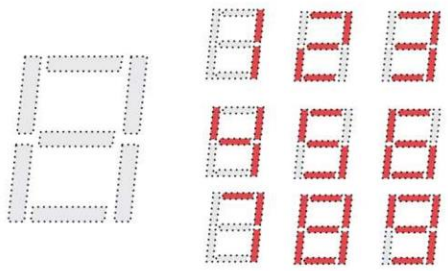
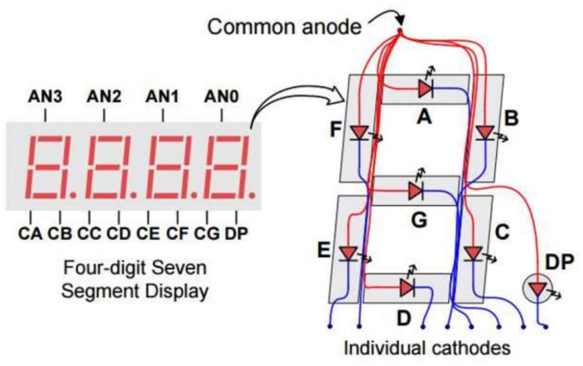
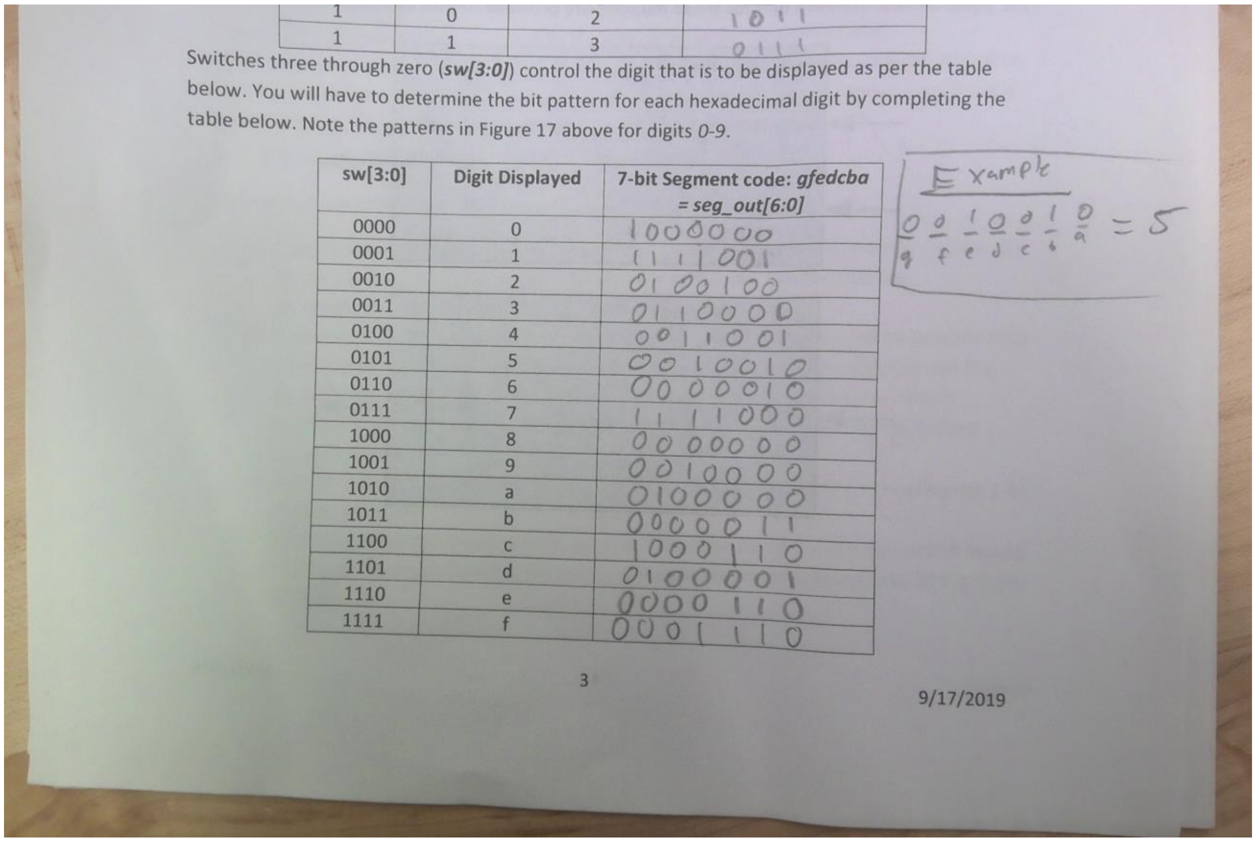


Figure 17. An un-illuminated seven-segment display and nine illumination patterns corresponding to decimal digits.



The cathodes have to be correspond with a visual representation of what the hexadecimal numbers and letters will appear as on the board, therefore the 7-bit segment code should be written in the order of cathodes as: GFEDCBA as shown in the truth table below:



1	0	2	1011
1	1	3	0111

Switches three through zero (***sw[3:0]***) control the digit that is to be displayed as per the table below. You will have to determine the bit pattern for each hexadecimal digit by completing the table below. Note the patterns in Figure 17 above for digits *0-9*.

sw[3:0]	Digit Displayed	7-bit Segment code: *gfedcba* = *seg_out[6:0]*
0000	0	1000000
0001	1	1111001
0010	2	0100100
0011	3	0110000
0100	4	0011001
0101	5	0010010
0110	6	0000010
0111	7	1111000
1000	8	0000000
1001	9	0010000
1010	a	0100000
1011	b	0000011
1100	c	1000110
1101	d	0100001
1110	e	0000110
1111	f	0001110

Example

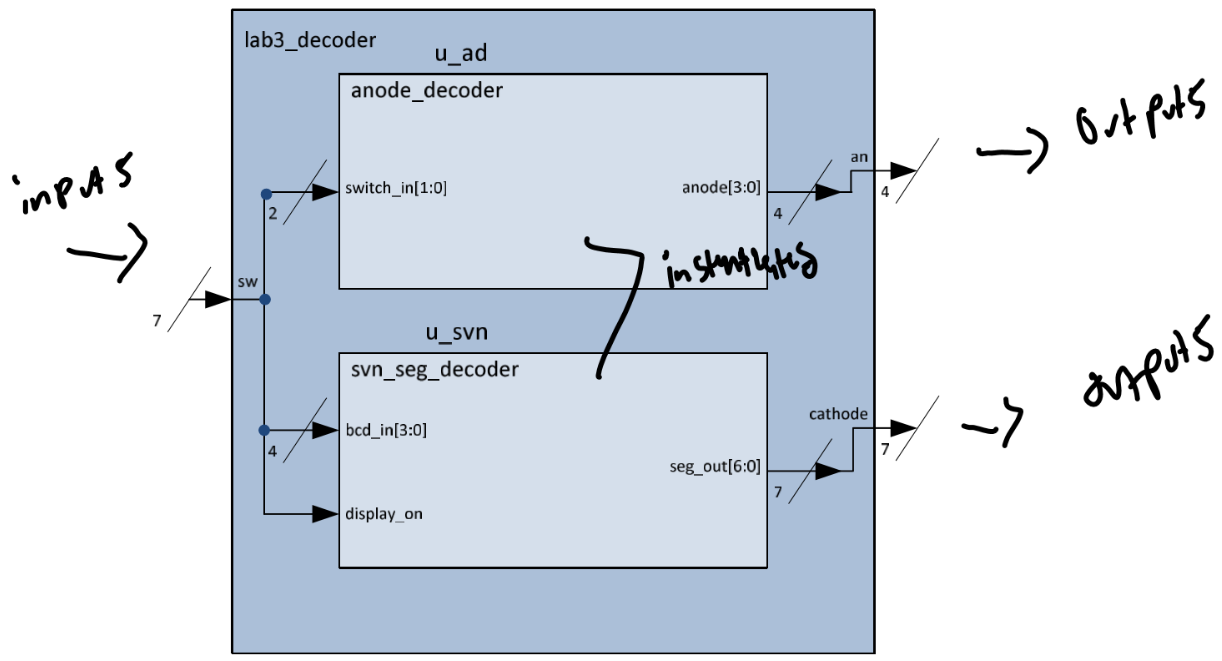
0	0	1	0	0	1	0	= 5
g	f	e	d	c	b	a	

3

9/17/2019

Upon doing this, I entered the truth tables into the corresponding test benches into Verilog and began constructing my code. The design works by instantiating 2 modules within a the larger lab3_decoder module, using this I was able to code the anode_decoder and svn_seg_decoder with no mismatches since I also established the rules for the testbench by inputting the truth table, therefore as long as my syntax and logic flow is correct, the code follows the schematic below and the simulations run 100% correct, which can then be checked using GTKWave and uploaded into the Basys3 board for hardware implementation.

Block Diagram

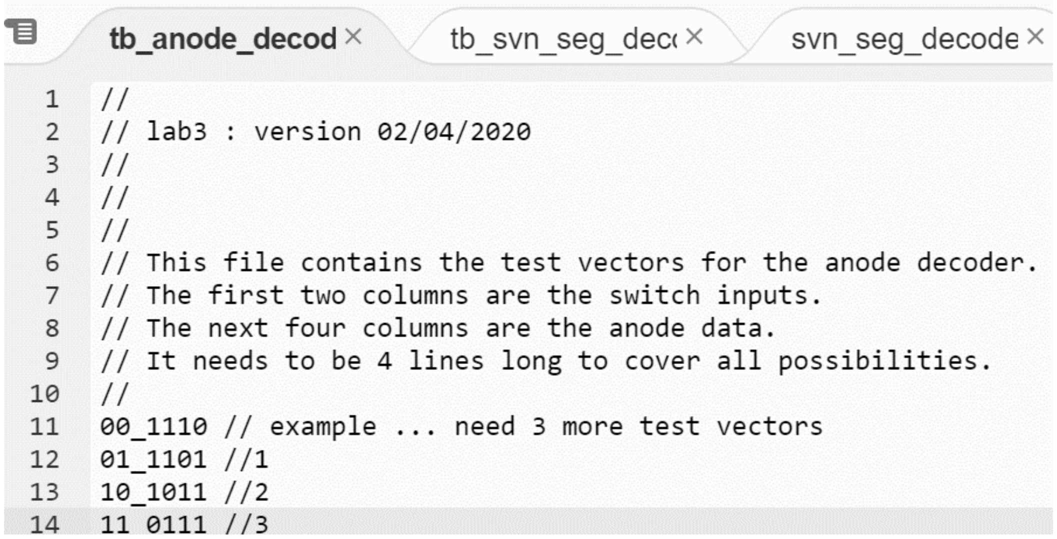


Results

Overall my experience with the lab turned out to be a success. After verifying the truth table and implementing it into the testbenches, the hardest part for me was understanding the rules of the Verilog case statements. Originally, I was putting case 1, case 2, etc. in front of the inputs but quickly realized that is not necessary. The only syntax errors I received was from not including the default originally, but then it worked fine. I also originally was using an else statement after my if statement in the svn_seg_decoder module but I realized I could just initialize when the segment is off before the if statement. Instantiating the gates proved to be straight forward after being given the example for the anode_decoder instantiation. I also checked my code using the GTKWave debugger but will not include this since it is optional for this lab. My simulations ran with no mismatches and the hardware implementation lit up with the corresponding truth table combinations.

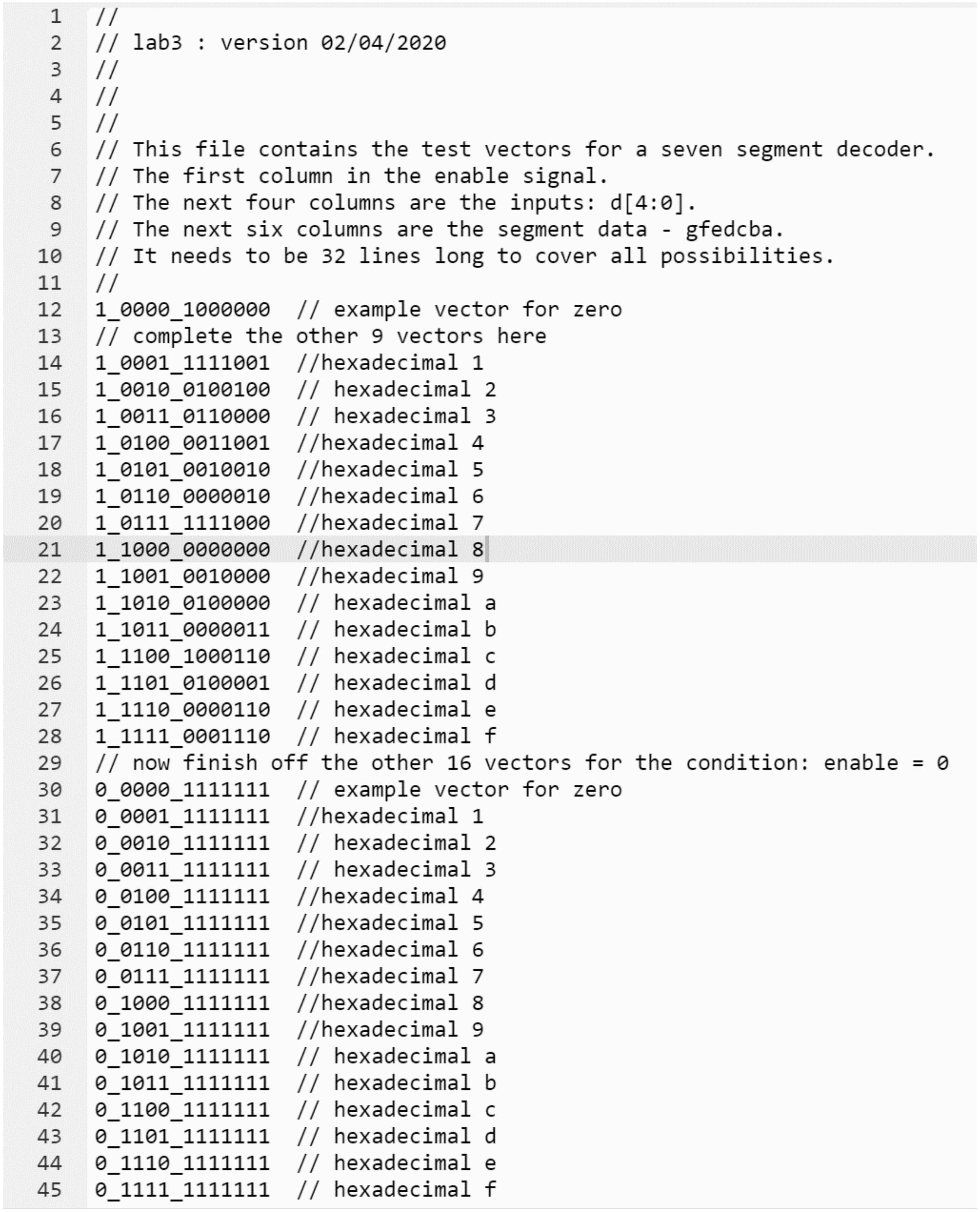
Design Code

Here is the testbench for the anode_decoder:



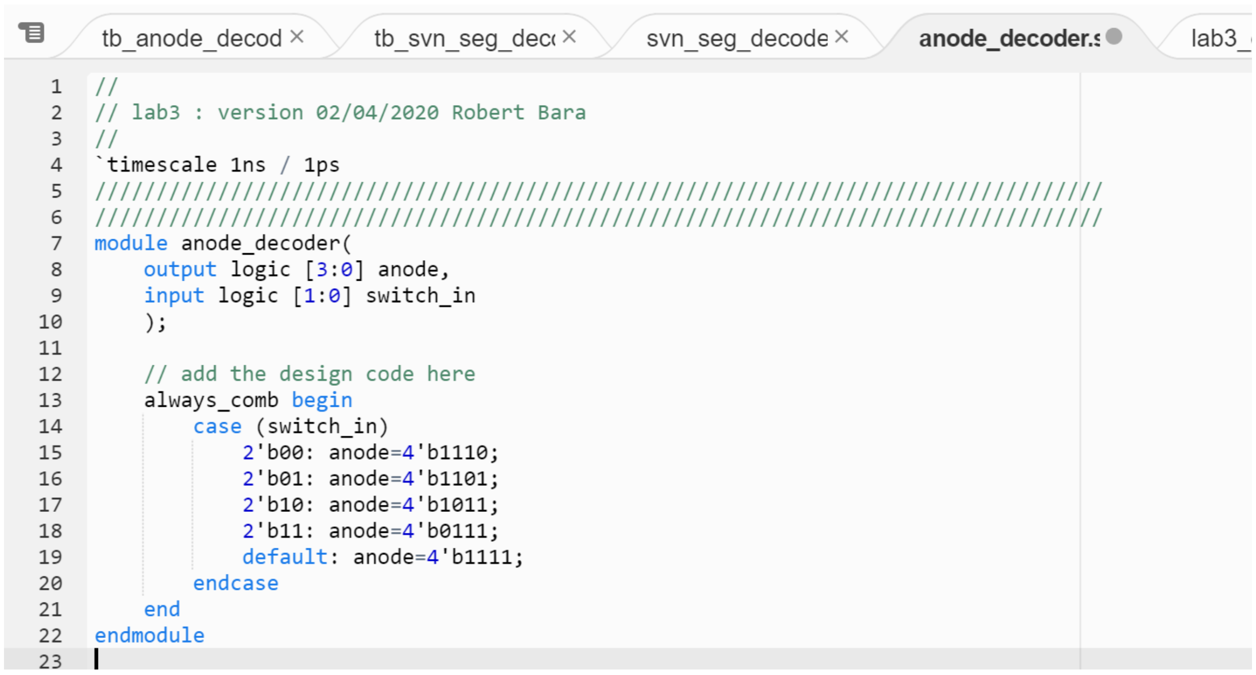
```
//
// lab3 : version 02/04/2020
//
//
//
// This file contains the test vectors for the anode decoder.
// The first two columns are the switch inputs.
// The next four columns are the anode data.
// It needs to be 4 lines long to cover all possibilities.
//
00_1110 // example ... need 3 more test vectors
01_1101 //1
10_1011 //2
11 0111 //3
```

The testbench for the svn_seg_decoder:
Since it is an active low design, 7’b1111111 is all off



```
//
// lab3 : version 02/04/2020
//
//
//
// This file contains the test vectors for a seven segment decoder.
// The first column in the enable signal.
// The next four columns are the inputs: d[4:0].
// The next six columns are the segment data - gfedcba.
// It needs to be 32 lines long to cover all possibilities.
//
1_0000_1000000  // example vector for zero
// complete the other 9 vectors here
1_0001_1111001  //hexadecimal 1
1_0010_0100100  // hexadecimal 2
1_0011_0110000  // hexadecimal 3
1_0100_0011001  //hexadecimal 4
1_0101_0010010  //hexadecimal 5
1_0110_0000010  //hexadecimal 6
1_0111_1111000  //hexadecimal 7
1_1000_0000000  //hexadecimal 8
1_1001_0010000  //hexadecimal 9
1_1010_0100000  // hexadecimal a
1_1011_0000011  // hexadecimal b
1_1100_1000110  // hexadecimal c
1_1101_0100001  // hexadecimal d
1_1110_0000110  // hexadecimal e
1_1111_0001110  // hexadecimal f
// now finish off the other 16 vectors for the condition: enable = 0
0_0000_1111111  // example vector for zero
0_0001_1111111  //hexadecimal 1
0_0010_1111111  // hexadecimal 2
0_0011_1111111  // hexadecimal 3
0_0100_1111111  //hexadecimal 4
0_0101_1111111  //hexadecimal 5
0_0110_1111111  //hexadecimal 6
0_0111_1111111  //hexadecimal 7
0_1000_1111111  //hexadecimal 8
0_1001_1111111  //hexadecimal 9
0_1010_1111111  // hexadecimal a
0_1011_1111111  // hexadecimal b
0_1100_1111111  // hexadecimal c
0_1101_1111111  // hexadecimal d
0_1110_1111111  // hexadecimal e
0_1111_1111111  // hexadecimal f
```

Code for the anode_decoder using Verilog case statements:

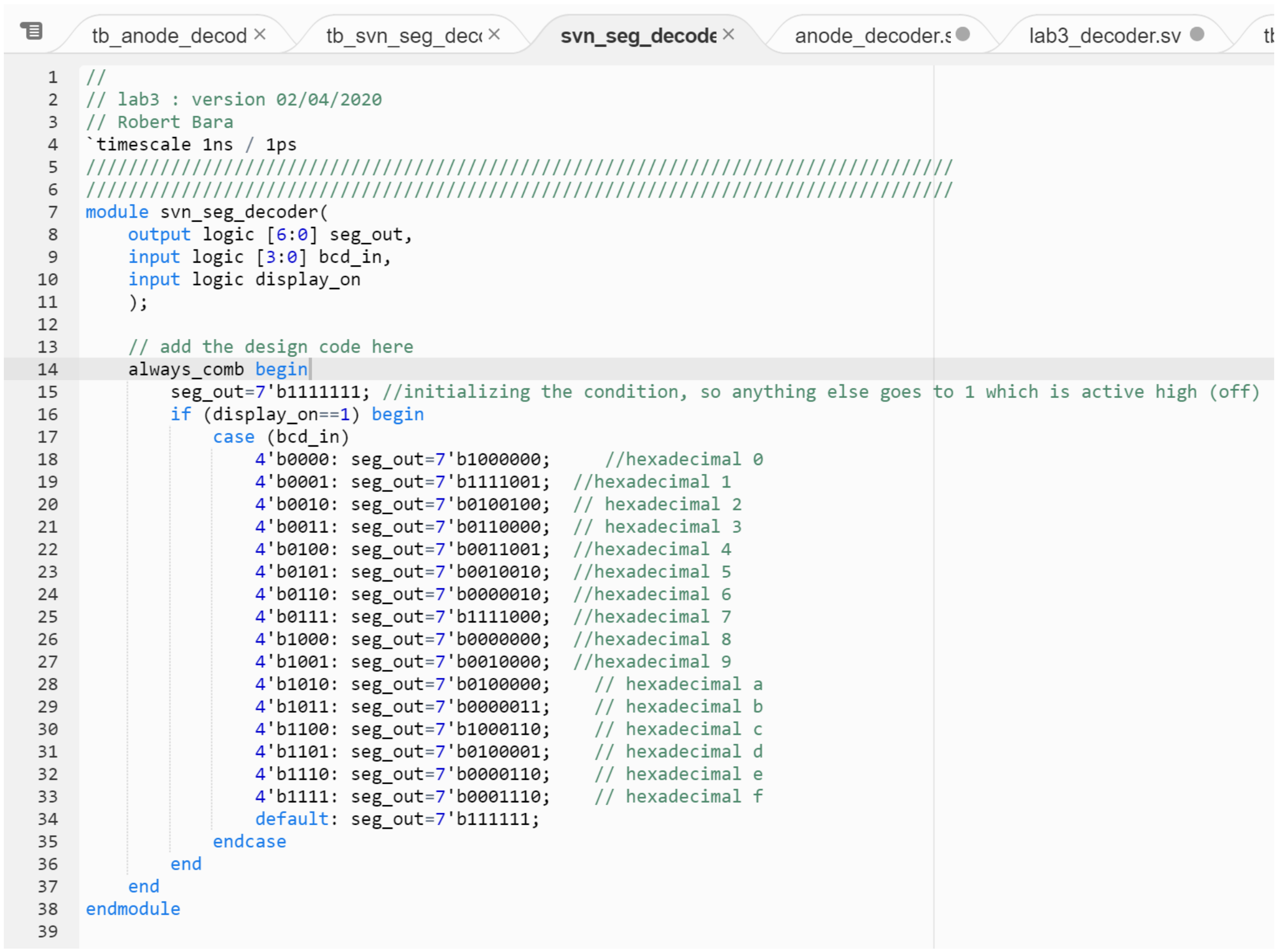


```
//
// lab3 : version 02/04/2020 Robert Bara
//
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////
module anode_decoder(
    output logic [3:0] anode,
    input logic [1:0] switch_in
    );

    // add the design code here
    always_comb begin
        case (switch_in)
            2'b00: anode=4'b1110;
            2'b01: anode=4'b1101;
            2'b10: anode=4'b1011;
            2'b11: anode=4'b0111;
            default: anode=4'b1111;
        endcase
    end
endmodule

```

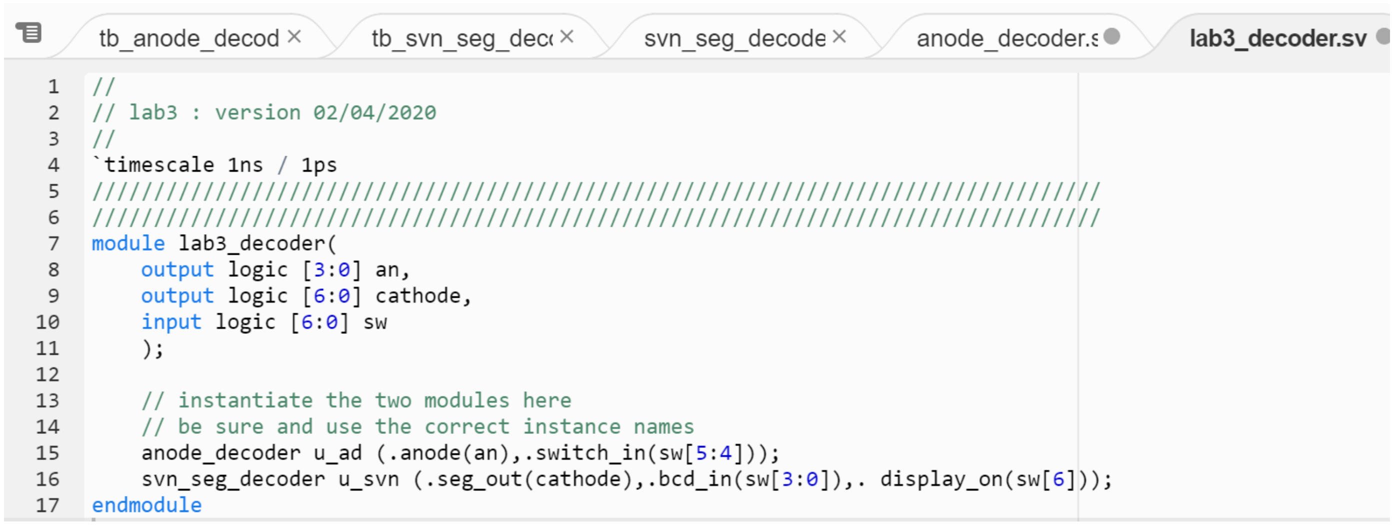
Code for the svn_seg_decoder using case and if statements:



```
//
// lab3 : version 02/04/2020
// Robert Bara
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////
module svn_seg_decoder(
    output logic [6:0] seg_out,
    input logic [3:0] bcd_in,
    input logic display_on
    );

    // add the design code here
    always_comb begin
        seg_out=7'b1111111; //initializing the condition, so anything else goes to 1 which is active high (off)
        if (display_on==1) begin
            case (bcd_in)
                4'b0000: seg_out=7'b1000000;     //hexadecimal 0
                4'b0001: seg_out=7'b1111001;  //hexadecimal 1
                4'b0010: seg_out=7'b0100100;  // hexadecimal 2
                4'b0011: seg_out=7'b0110000;  // hexadecimal 3
                4'b0100: seg_out=7'b0011001;  //hexadecimal 4
                4'b0101: seg_out=7'b0010010;  //hexadecimal 5
                4'b0110: seg_out=7'b0000010;  //hexadecimal 6
                4'b0111: seg_out=7'b1111000;  //hexadecimal 7
                4'b1000: seg_out=7'b0000000;  //hexadecimal 8
                4'b1001: seg_out=7'b0010000;  //hexadecimal 9
                4'b1010: seg_out=7'b0100000;    // hexadecimal a
                4'b1011: seg_out=7'b0000011;    // hexadecimal b
                4'b1100: seg_out=7'b1000110;    // hexadecimal c
                4'b1101: seg_out=7'b0100001;    // hexadecimal d
                4'b1110: seg_out=7'b0000110;    // hexadecimal e
                4'b1111: seg_out=7'b0001110;    // hexadecimal f
                default: seg_out=7'b111111;
            endcase
        end
    end
endmodule
```

Instantiating both modules within the lab3_decoder module:

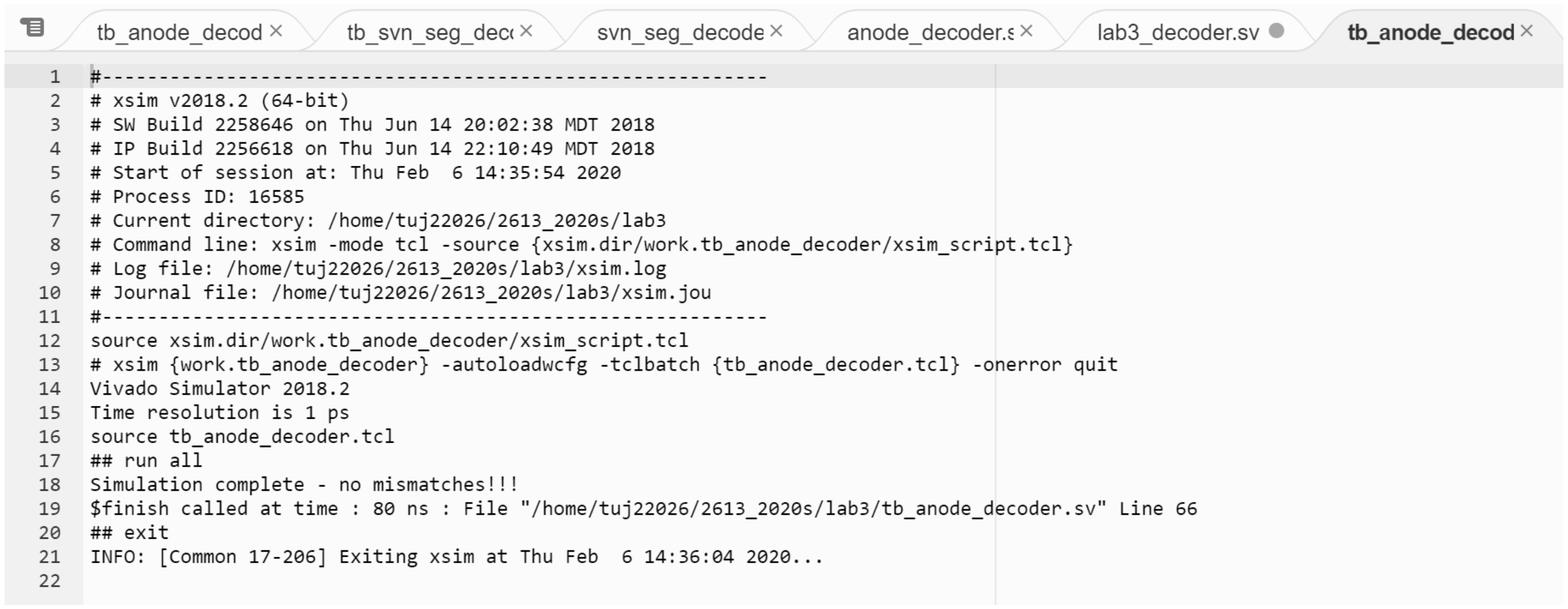


```
//
// lab3 : version 02/04/2020
//
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////
module lab3_decoder(
    output logic [3:0] an,
    output logic [6:0] cathode,
    input logic [6:0] sw
    );

    // instantiate the two modules here
    // be sure and use the correct instance names
    anode_decoder u_ad (.anode(an),.switch_in(sw[5:4]));
    svn_seg_decoder u_svn (.seg_out(cathode),.bcd_in(sw[3:0]),. display_on(sw[6]));
endmodule
```

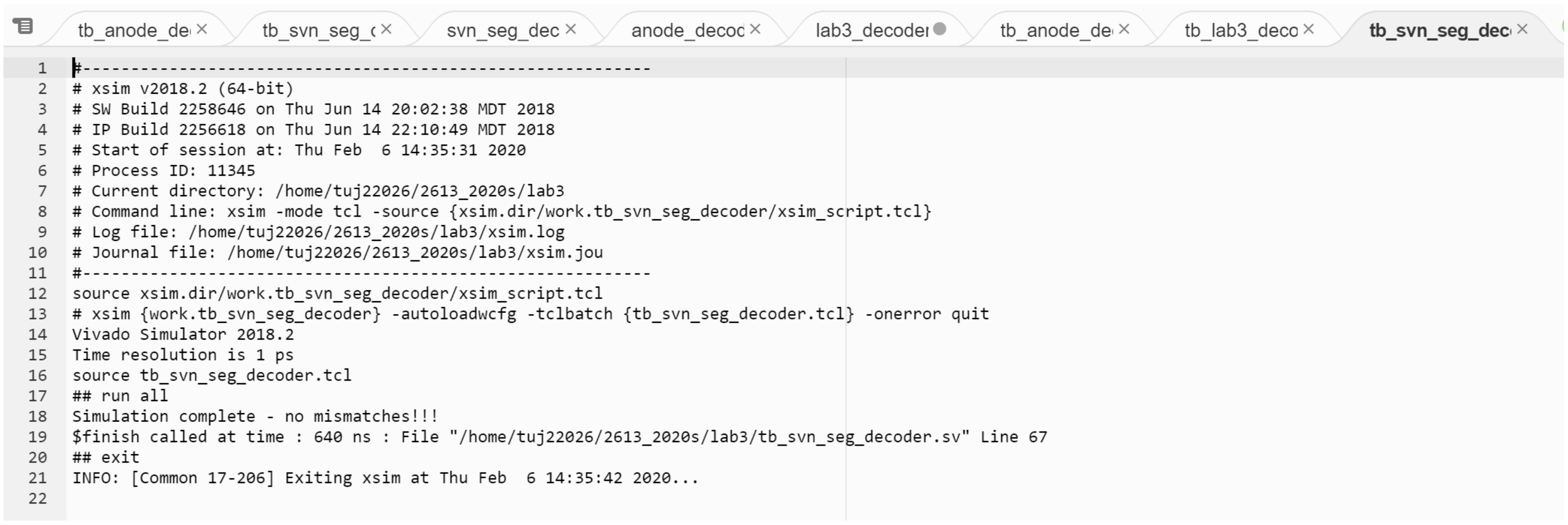
Simulation Results

Simulation file of the anode_decoder module with no mismatches:



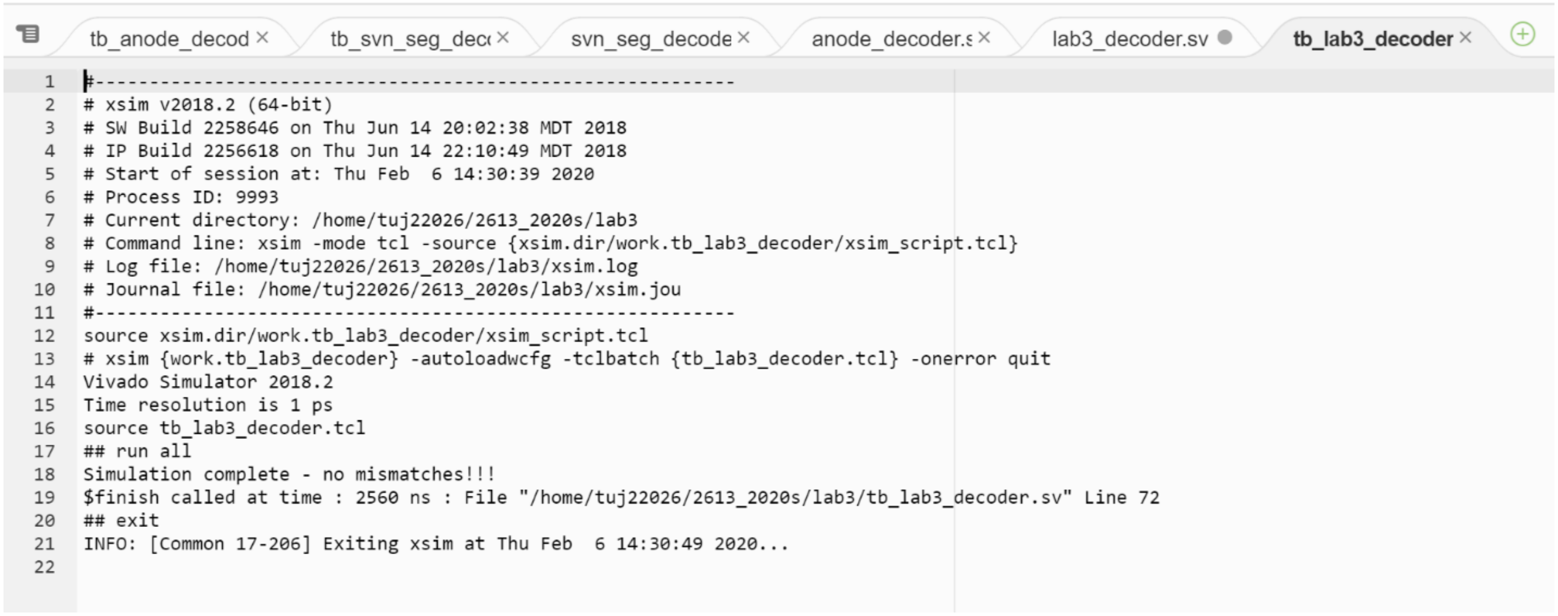
```
#-----------------------------------------------------------
# xsim v2018.2 (64-bit)
# SW Build 2258646 on Thu Jun 14 20:02:38 MDT 2018
# IP Build 2256618 on Thu Jun 14 22:10:49 MDT 2018
# Start of session at: Thu Feb  6 14:35:54 2020
# Process ID: 16585
# Current directory: /home/tuj22026/2613_2020s/lab3
# Command line: xsim -mode tcl -source {xsim.dir/work.tb_anode_decoder/xsim_script.tcl}
# Log file: /home/tuj22026/2613_2020s/lab3/xsim.log
# Journal file: /home/tuj22026/2613_2020s/lab3/xsim.jou
#-----------------------------------------------------------
source xsim.dir/work.tb_anode_decoder/xsim_script.tcl
# xsim {work.tb_anode_decoder} -autoloadwcfg -tclbatch {tb_anode_decoder.tcl} -onerror quit
Vivado Simulator 2018.2
Time resolution is 1 ps
source tb_anode_decoder.tcl
## run all
Simulation complete - no mismatches!!!
$finish called at time : 80 ns : File "/home/tuj22026/2613_2020s/lab3/tb_anode_decoder.sv" Line 66
## exit
INFO: [Common 17-206] Exiting xsim at Thu Feb  6 14:36:04 2020...
```

Simulation file of the svn_seg_decoder module with no mismatches:



```
#-----------------------------------------------------------
# xsim v2018.2 (64-bit)
# SW Build 2258646 on Thu Jun 14 20:02:38 MDT 2018
# IP Build 2256618 on Thu Jun 14 22:10:49 MDT 2018
# Start of session at: Thu Feb  6 14:35:31 2020
# Process ID: 11345
# Current directory: /home/tuj22026/2613_2020s/lab3
# Command line: xsim -mode tcl -source {xsim.dir/work.tb_svn_seg_decoder/xsim_script.tcl}
# Log file: /home/tuj22026/2613_2020s/lab3/xsim.log
# Journal file: /home/tuj22026/2613_2020s/lab3/xsim.jou
#-----------------------------------------------------------
source xsim.dir/work.tb_svn_seg_decoder/xsim_script.tcl
# xsim {work.tb_svn_seg_decoder} -autoloadwcfg -tclbatch {tb_svn_seg_decoder.tcl} -onerror quit
Vivado Simulator 2018.2
Time resolution is 1 ps
source tb_svn_seg_decoder.tcl
## run all
Simulation complete - no mismatches!!!
$finish called at time : 640 ns : File "/home/tuj22026/2613_2020s/lab3/tb_svn_seg_decoder.sv" Line 67
## exit
INFO: [Common 17-206] Exiting xsim at Thu Feb  6 14:35:42 2020...
```

Simulation file of the lab3_decoder module with no mismatches:


tb_anode_decod × tb_svn_seg_dec × svn_seg_decode × anode_decoder.s × lab3_decoder.sv ● tb_lab3_decoder ×

```
#-----------------------------------------------------------
# xsim v2018.2 (64-bit)
# SW Build 2258646 on Thu Jun 14 20:02:38 MDT 2018
# IP Build 2256618 on Thu Jun 14 22:10:49 MDT 2018
# Start of session at: Thu Feb  6 14:30:39 2020
# Process ID: 9993
# Current directory: /home/tuj22026/2613_2020s/lab3
# Command line: xsim -mode tcl -source {xsim.dir/work.tb_lab3_decoder/xsim_script.tcl}
# Log file: /home/tuj22026/2613_2020s/lab3/xsim.log
# Journal file: /home/tuj22026/2613_2020s/lab3/xsim.jou
#-----------------------------------------------------------
source xsim.dir/work.tb_lab3_decoder/xsim_script.tcl
# xsim {work.tb_lab3_decoder} -autoloadwcfg -tclbatch {tb_lab3_decoder.tcl} -onerror quit
Vivado Simulator 2018.2
Time resolution is 1 ps
source tb_lab3_decoder.tcl
## run all
Simulation complete - no mismatches!!!
$finish called at time : 2560 ns : File "/home/tuj22026/2613_2020s/lab3/tb_lab3_decoder.sv" Line 72
## exit
INFO: [Common 17-206] Exiting xsim at Thu Feb  6 14:30:49 2020...
```

Hardware Implementation

Upon uploading my code, actually flipping the switches was a little bit difficult since the labels on each switch are small and hard to see, but nonetheless my design worked and I was checked off by Sivan during the lab period on 02/06/2020 at about 3:10pm.

Conclusion

This lab solidified Verilog syntax, specifically the use of writing case statements for me which I think will be useful for the upcoming exams and future labs. My truth tables matched the criteria of the active low vs active high design and by looking at the display cathodes I was able to construct the correct 7-bit segments needed to display one hexadecimal character on the desired anode. Coding the individual modules and instantiating them produced simulations with no mismatches and the .bit file correctly transformed the block design into working hardware by using the basys3 board.