Utilizing the Seven Segment Display on a Nexys2 Board

Vincent Martin
TUID: 913012274
ECE 2613
Lab #: 3 (9/13/2012)

Introduction:
The objective of this lab is to utilize our understanding of Verilog and the Sum of Products method to create a Verilog module that will use input based on switches to drive the seven segment display built into our Xilinx test board, test it, and finally implement it on the board itself.

The Theory of our Seven Segment Display:
To drive the display we will need to create a module that has two inputs. One being a 4 bit bus that will represent the 4 switches we are using to to create binary numbers. The second input will be the 1 bit status of a 5th switch. These inputs will result in an output of 7 bits into the seg_out bus.

When the 5th switch is on and the remaining 4 switches are set the end user will be able to control what gets sent to the seven segment display components by the seg_out bus. Otherwise we will see no display if the 5th switch is off.

Applying the Theory to Hardware:

In order to transfer our understanding of theory to our Nexys2 hardware board we will have to write a Verilog code module that represents the block diagram seen in figure 1.

Module Description:

- Input
  - bcd_in: 4 bit mapped to switches 0 to 3 on the board.
  - display_on: 1 bit mapped to switch 4 on the board.
- Output
  - seg_out: 7 bit mapped to each particular segment of the seven segment display as seen in figure 2.

Figure 1 : Block diagram for the svn_seg_decoder

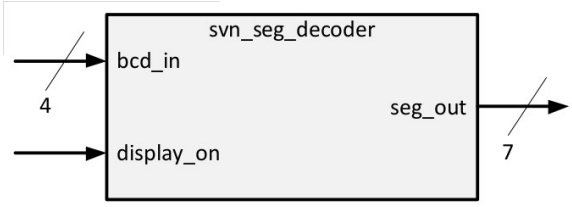
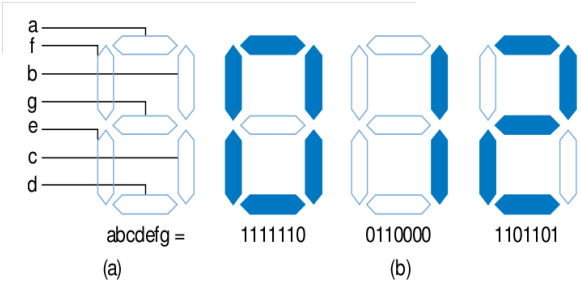
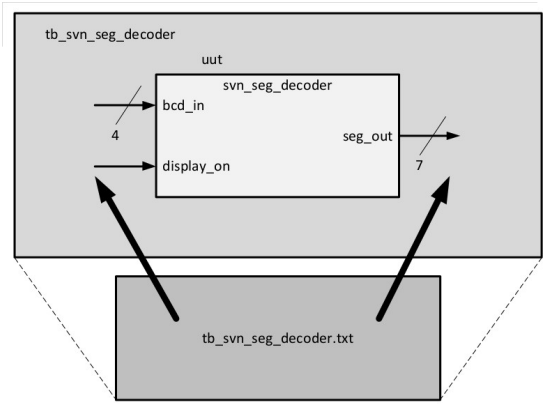


Figure 2 : Seven Segment Display Diagram



Additionally we will want to implement a testing module based on Figure 3. This scheme will utilize a .txt file, based on our truth table, which will allow us to test all of our expected outcomes. This text file will be included in the lab report.

Figure 3: Testing methods



Procedure:

Create the Truth Table and Formula

1. Create a truth table for our module (figure 4).
2. Calculate the equations that will result in seg_out lighting the appropriate segments in regards to the input of bcd_in and display_on.
3. Use the results of the truth table to create a Equations found below in Figure 5.

Implement the Design in Software

1. Make a secure connection to electro9.eng.temple.edu using the no machine client.
2. Once terminal opens on the local workstation type the ‘remote_xilinx.sh’ shell command to launch the ISE development environment.
3. Open the lab3 seven segment project in ~/Xilinx/lab3/ directory.
4. Modify the svn_seg_out.v source code to implement your algorithm by navigating to
    - View: Implementation
        - ¬ xc3s500e-4f6320
5. Save all files.

Prepare for Testing the Design

1. Modify the tb_svn_seg_out.txt testing to suit the needs of our truth table by navigating to
    - View: Implementation
2. Modify the tb_svn_seg_out.txt file to contain all possible input bit combinations.
3. Modify the tb_svn_seg_out.txt to contain all expected output bit combinations.
4. Save all files.

Test the Design with iSim

1. Switch to Simulation mode by clicking on
    a. View:Simulation
        i. Xc3s500e-4fg320.
        ii. Tb_svn_seg_out
2. Run iSim simulator by clicking on
    a. iSim Simulator
    b. Right click Simulate Behavioral Model and then run.
3. Once iSim runs, verify that the Mismatch—index messages match what you are expecting in your test bench text file.
4. If the results are not what you expect either edit your module code or your test bench code and then attempt to test again.
5. If the results are what you expected move on to the Compile to .bit file step.

Compile to .bit file

1. Compile to .bit file by navigating to
    a. Implementation

i. Xc3s500e-4g320
    1. Lab3_top_io_wrapper
        a. Implement design
        b. Generation programming file

Transfer .bit file to Board

1. Use your favorite network transfer program to move the .bit file from the development server to your local workstation.
2. Plug the board into USB port.
3. Launch the Digilent Adept application on your local workstation.
4. Click the config tab.
5. Click on browse by the PROM icon.
6. Select your transferred .bit file.
7. Click program.
8. Once complete press the reset button on the board.
9. Test your outcome physically on the board to make sure that it matches expectations.

Results:

Below you will find the truth table and equation representing our seven segment display. The results on the physical board matched what was expected from the truth table.

Figure 4: Truth Table

Display	BCD IN				g	f	e	d	c	b	a
Display_ON	bcd_in[3]	bcd_in[2]	bcd_in[1]	bcd_in[0]	seg_out[6]	seg_out[5]	seg_out[4]	seg_out[3]	seg_out[2]	seg_out[1]	seg_out[0]
1	0	0	0	0	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	0	0	0
1	0	0	1	0	1	0	1	1	0	1	1
1	0	0	1	1	1	1	1	1	0	0	1
1	0	1	0	0	1	1	1	0	1	0	0
1	0	1	0	1	1	1	0	1	1	0	1
1	0	1	1	0	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	0	0	0
1	1	0	0	0	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	0	1
1	1	0	1	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0

Create Equation that Satisfies the Truth Table:

Each bit of the seg_out bus will need its own equation utilizing an AND statement and then a sum of products. Each of these equations will have to be determined based on what is in the truth table and then converted to Verilog code. The Verilog code can be found in the source code section of this lab report.

Below are the equations that we will use to control each of the segments in the seven segment display. When these are equal to '1' the board will then light up that particular segment. The combination of these will form the number we request with our bcd_in switches.

Figure 5: Verilog Equations

```
Seg_out[6] =
        (Display_On)
        (
        (bcd_in[3]' bcd_in[2]' bcd_in[1] bcd_in[0]') +
         (bcd_in[3]'  bcd_in[2]'  bcd_in[1] bcd_in[0]) +
         (bcd_in[3]`  bcd_in[2]` bcd_in[1]` bcd_in[0]`) +
         (bcd_in[3]`  bcd_in[2]  bcd_in[1]` bcd_in[0]) +
         (bcd_in[3]`  bcd_in[2]  bcd_in[1]  bcd_in[0]`) +
         (bcd_in[3]   bcd_in[2]` bcd_in[1]` bcd_in[0]`) +
         (bcd_in[3]   bcd_in[2]` bcd_in[1]` bcd_in[0])
        )

seg_out[5] =

        (display_on)
        (
        (bcd_in[3]'  bcd_in[2]'  bcd_in[1]'   bcd_in[0]') +
        (bcd_in[3]'  bcd_in[2]'  bcd_in[1]'    bcd_in[0]) +
        (bcd_in[3]'  bcd_in[2]'   bcd_in[1]    bcd_in[0]) +
        (bcd_in[3]'   bcd_in[2]  bcd_in[1]'   bcd_in[0]') +
        (bcd_in[3]'   bcd_in[2]  bcd_in[1]'    bcd_in[0]) +
        (bcd_in[3]'   bcd_in[2]   bcd_in[1]   bcd_in[0]') +
        (bcd_in[3]'   bcd_in[2]   bcd_in[1]    bcd_in[0]) +
        (bcd_in[3]  bcd_in[2]'  bcd_in[1]'  bcd_in[0]') +
        (bcd_in[3]  bcd_in[2]'  bcd_in[1]'    bcd_in[0])
        )
```

```
seg_out[4] =
      (display_on)
      (
      (bcd_in[3]’  bcd_in[2]’   bcd_in[1]’   bcd_in[0]’) +
      (bcd_in[3]’  bcd_in[2]’   bcd_in[1]’    bcd_in[0]) +
      (bcd_in[3]’  bcd_in[2]’    bcd_in[1]   bcd_in[0]’) +
      (bcd_in[3]’  bcd_in[2]’    bcd_in[1]    bcd_in[0]) +
      (bcd_in[3]’   bcd_in[2]   bcd_in[1]’   bcd_in[0]’) +
      (bcd_in[3]’   bcd_in[2]    bcd_in[1]    bcd_in[0]) +
      (bcd_in[3]  bcd_in[2]’   bcd_in[1]’   bcd_in[0]’) +
      (bcd_in[3]  bcd_in[2]’   bcd_in[1]’    bcd_in[0])
      )

seg_out[3] =
      (display_on)
      (
      (bcd_in[3]’  bcd_in[2]’   bcd_in[1]’   bcd_in[0]’) +
      (bcd_in[3]’  bcd_in[2]’    bcd_in[1]   bcd_in[0]’) +
      (bcd_in[3]’  bcd_in[2]’    bcd_in[1]    bcd_in[0]) +
      (bcd_in[3]’   bcd_in[2]   bcd_in[1]’    bcd_in[0]) +
      (bcd_in[3]’   bcd_in[2]    bcd_in[1]   bcd_in[0]’) +
      (bcd_in[3]’   bcd_in[2]    bcd_in[1]    bcd_in[0]) +
      (bcd_in[3]  bcd_in[2]’   bcd_in[1]’   bcd_in[0]’) +
      (bcd_in[3]  bcd_in[2]’   bcd_in[1]’    bcd_in[0])
      )

seg_out[2] =
      (display_on)
      (
      (bcd_in[3]’  bcd_in[2]’   bcd_in[1]’   bcd_in[0]’) +
      (bcd_in[3]’   bcd_in[2]   bcd_in[1]’   bcd_in[0]’) +
      (bcd_in[3]’   bcd_in[2]   bcd_in[1]’    bcd_in[0]) +
      (bcd_in[3]’   bcd_in[2]    bcd_in[1]   bcd_in[0]’) +
      (bcd_in[3]  bcd_in[2]’   bcd_in[1]’   bcd_in[0]’) +
      (bcd_in[3]  bcd_in[2]’   bcd_in[1]’    bcd_in[0])
      )

seg_out[1] =
      (display_on)
      (
      (bcd_in[3]’  bcd_in[2]’   bcd_in[1]’   bcd_in[0]’) +
      (bcd_in[3]’  bcd_in[2]’    bcd_in[1]   bcd_in[0]’) +
      (bcd_in[3]’   bcd_in[2]    bcd_in[1]   bcd_in[0]’) +
      (bcd_in[3]  bcd_in[2]’   bcd_in[1]’   bcd_in[0]’)
```

```
        )

 seg_out[0] =
        (display_on)
        (
        (bcd_in[3]’  bcd_in[2]’   bcd_in[1]’   bcd_in[0]’) +
        (bcd_in[3]’  bcd_in[2]’    bcd_in[1]   bcd_in[0]’) +
        (bcd_in[3]’  bcd_in[2]’    bcd_in[1]    bcd_in[0]) +
        (bcd_in[3]’   bcd_in[2]   bcd_in[1]’    bcd_in[0]) +
        (bcd_in[3]’   bcd_in[2]    bcd_in[1]   bcd_in[0]’) +
        (bcd_in[3]  bcd_in[2]’   bcd_in[1]’   bcd_in[0]’) +
        (bcd_in[3]  bcd_in[2]’   bcd_in[1]’    bcd_in[0])
        )
```

Discussion:

This was a good step up from the previous lab. I feel that this lab gave me a better feel for how things are connected together in this language. Also, it gave me experience in a slightly more complex system design.

In particular the experience has cemented in my mind how all of this runs in parallel. It did not matter what order any of my assign statements were in because they are essentially all acted on at the identical time within which would be controlled by a clock setting within the chip.

Initially I had problems getting the lights to work properly. I now know that this was because I was being lazy. I should have not tried to implement a design before its test bed module is complete. What I thought would be easy to test manually turned into quite a lot of labor until I utilized the test bed. In the future I will always do the test bed first.

The svn_seg_module could have been designed in a few different ways. One of those was to assign a wire instance to a name, for example 'p#' and then assign that each particular combination of bcd_in. This would have saved me some typing and saved me the effort of having huge blocks of verilog code.

Source Code:
Please see attached documents.

SVN_SEG_DECODER.V module code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Vinceco
// Engineer: Vincent Martin
//
// Create Date:    18:16:21 09/11/2012
// Design Name:    seven segment decoder module
// Module Name:    svn_seg_decoder
// Project Name:   lab 03 seven segment decoder
// Target Devices:  xilinx board
// Tool versions:
// Description: Take in 4 bits as a descriptor of the number you want to show and also
// 1bit as an on/off switch and then output the appropriate signal to drive
// the seven segment display.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module svn_seg_decoder(
    input [3:0] bcd_in,
    input display_on,
    output [6:0] seg_out
    );

/*
___Note about how this was done___
I think that it might have been easier to create a variable for each particular combination possibility of our bcd_in[3] through bcd_in[0] and do the logic that way. So that I could make all of my crazy assign seg_out[#] logic blocks smaller.

This would have made it a little more easy to read I think, however, this way works just as well. But it can be kind of confusing to read and error prone to type.
```

```
*/

// code to calculate our output goes here.

assign seg_out[6] =

        (display_on) &
        (
        (~bcd_in[3] & ~bcd_in[2] &   bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] & ~bcd_in[2] &   bcd_in[1]  &  bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &  ~bcd_in[1]  &  bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &   bcd_in[1]  & ~bcd_in[0]) |
        ( bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
        ( bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  &  bcd_in[0])
        );


assign seg_out[5] =

        (display_on) &
        (
        (~bcd_in[3] & ~bcd_in[2] & ~bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] & ~bcd_in[2] & ~bcd_in[1]  &  bcd_in[0]) |
        (~bcd_in[3] & ~bcd_in[2] &  bcd_in[1]  &  bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] & ~bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] & ~bcd_in[1]  &  bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &  bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &  bcd_in[1]  &  bcd_in[0]) |
        ( bcd_in[3] & ~bcd_in[2] & ~bcd_in[1]  & ~bcd_in[0]) |
        ( bcd_in[3] & ~bcd_in[2] & ~bcd_in[1]  &  bcd_in[0])

        );

assign seg_out[4] =
        (display_on) &
        (
        (~bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  &  bcd_in[0]) |
        (~bcd_in[3] & ~bcd_in[2] &   bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] & ~bcd_in[2] &   bcd_in[1]  &  bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &   bcd_in[1]  &  bcd_in[0]) |
        ( bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
```

```
        ( bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  &  bcd_in[0])

        );

assign seg_out[3] =
        (display_on) &
        (
        (~bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] & ~bcd_in[2] &   bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] & ~bcd_in[2] &   bcd_in[1]  &  bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &  ~bcd_in[1]  &  bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &   bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &   bcd_in[1]  &  bcd_in[0]) |
        ( bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
        ( bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  &  bcd_in[0])
        );

assign seg_out[2] =
        (display_on) &
        (
        (~bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &  ~bcd_in[1]  &  bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &   bcd_in[1]  & ~bcd_in[0]) |
        ( bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
        ( bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  &  bcd_in[0])
        );

assign seg_out[1] =
        (display_on) &
        (
        (~bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] & ~bcd_in[2] &   bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &   bcd_in[1]  & ~bcd_in[0]) |
        ( bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0])
        );

assign seg_out[0] =
        (display_on) &
        (
        (~bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] & ~bcd_in[2] &   bcd_in[1]  & ~bcd_in[0]) |
        (~bcd_in[3] & ~bcd_in[2] &   bcd_in[1]  &  bcd_in[0]) |
        (~bcd_in[3] &  bcd_in[2] &  ~bcd_in[1]  &  bcd_in[0]) |
```

```
        (~bcd_in[3] &  bcd_in[2] &   bcd_in[1]  & ~bcd_in[0]) |
        ( bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  & ~bcd_in[0]) |
        ( bcd_in[3] & ~bcd_in[2] &  ~bcd_in[1]  &  bcd_in[0])
        );



endmodule
```

tb_svn_seg_module.txt

```
//
// lab3 : version 09/06/2012
//
// This file contains the test vectors for the
// 7 segment decoder
// The first column is the input display_on signal
// The next four columns are the inputs: bcd_in[3:0]
// The next 7 columns are the signals to the display:
// seg_out[6:0], representing the g,f,e,d,c,b,a segments.
//
// This needs to be 32 lines long to cover all possibilities
//
/* This is what they gave me, but I am going to just comment it all
out and then start over with my own generated file.

1_0000_0111111
1_0001_0110000

I basically copied this data from the excel sheet truth table and then
used
tr -d '\t' <test.txt >> tb_svn_seg_decoder.txt
to strip it of the tab delimiters.
*/


1_0000_0111111
1_0001_0110000
1_0010_1011011
1_0011_1111001
1_0100_1110100
1_0101_1101101
1_0110_1101111
1_0111_0111000
1_1000_1111111
1_1001_1111101
1_1010_0000000
1_1011_0000000
1_1100_0000000
1_1101_0000000
1_1110_0000000
1_1111_0000000
0_0000_0000000
```

0_0001_0000000
0_0010_0000000
0_0011_0000000
0_0100_0000000
0_0101_0000000
0_0110_0000000
0_0111_0000000
0_1000_0000000
0_1001_0000000
0_1010_0000000
0_1011_0000000
0_1100_0000000
0_1101_0000000
0_1110_0000000
0_1111_0000000