**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:

Put some bullshit in here about how we drive the display and how each part of the display is setup to correspond to a particular instance or something.

Show a picture of the display along w/ labels.

Talk about how our truth table will determine our equations

Talk about how what is in this truth table will turn into a few different sum of products equations.

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.

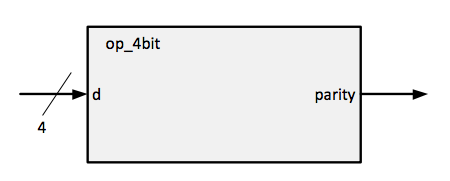


Figure : Block diagram for op\_4bit module

Additionally we will want to implement a testing scheme based on Figure 2. 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.

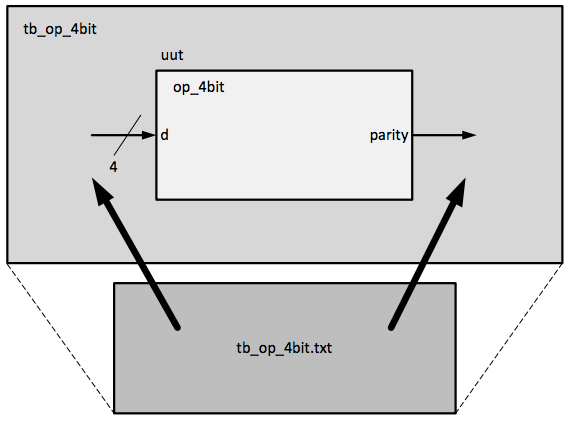


Figure : Testing methods

**Procedure:**

Create the Truth Table and Formula

1. Create a truth table for our module (figure 3).
2. Calculate the F parity bit that will make the sum odd.
3. Use the results of the truth table to create a Sum of Products Equation (figure 4).

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 lab2 bit party project in ~/Xilinx/lab2/ directory.
4. Modify the op\_4bit.v source code to implement your algorithm by navigating to
   * View: Implementation
     + xc3s500e-4f6320
       - lab2\_top\_io\_wrapper
5. Save all files.

Prepare for Testing the Design

1. Modify the tb\_op\_4bit.txt testing to suit the needs of a 4 bit odd parity test by navigating to

* View: Implementation

1. Modify the tb\_op\_4bit.txt file to contain all possible input bit combinations.
2. Modify the tb\_op\_4bit.txt file to contain all expected output bit combinations.
3. Save all files.

Test the Design with iSim

1. Switch to Simulation mode by clicking on
   1. View:Simulation
      1. Xc3s500e-4fg320.
      2. Tb\_op\_4bit
2. Run iSim simulator by clicking on
   1. iSim Simulator
   2. 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
   1. Implementation
      1. Xc3s500e-4g320
         1. Lab2\_top\_io\_wrapper
            1. Implement design
            2. 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 4 bit odd parity generator.

Truth Table (figure 3):

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **BCD IN** | 4 | 2 | 1 | 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] |
| 0 | x | x | x | x | x | x | x | x | x | x | x |
| 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 |

Create Equation that Satisfies the Truth Table:

Each bit of the seg\_out bus will need its own equation utilizing an AND 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.

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:**

Talk about how there were multiple ways to do this, and that your way was kind of the more confusing way.

Also put some bullshit in here about how this showed you how important testing would be, as if there were problems you would have to spend tons of time hunting them down manually… maybe you sould even say some things about how you did initially do this and that now you realize how bad of an idea it was.

Also put some stuff in there abot how you made the truth table and which unix commands you used.

Also maybe talk about how you now just edit the code locally and then upload it to the machine because the remote access is pretty slow and pretty shitty.

**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\_op\_4bit.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