Lab 2-Encoder Design Four bit, Error Correction Code (ECC) using Hamming(7,4) Code
Lab 4-Four bit, Error Correction Code (ECC) using Hamming(7,4) Code Decoding

Robert Bara

Section #: 003

Date: 2/13/2020

Summary/Abstract (10 points)

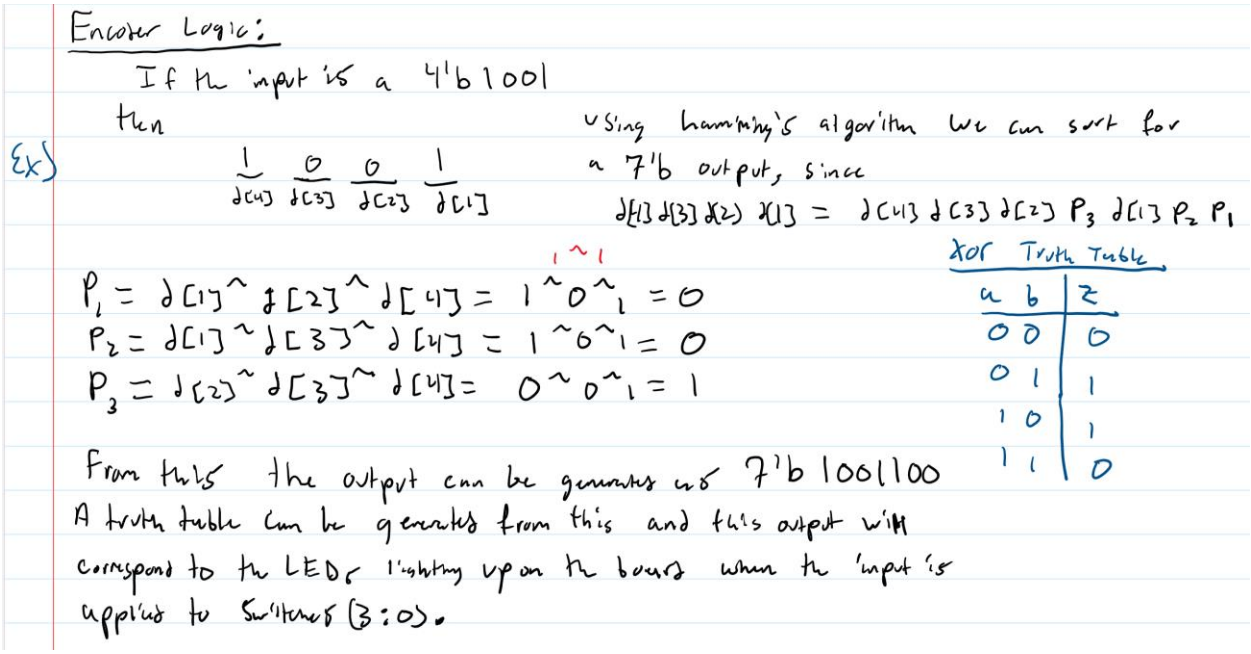
The idea of Lab 2 and Lab 4 were to understand the hamming sorting algorithm which uses parity to encode 4-bits to 7-bits or decode 7-bits to 4-bits. After designing and instantiating both modules, the encoder will take an input of 4-bits and display a combination of 7 LEDs corresponding to the truth table output, or the decoder will take the 7-bits and decode them to display which element was the bad bit by lighting up an LED, and display a hexadecimal digit on the Cathodes which will allow us to understand which bit is the bad bit.

Introduction (10 points)

Relevant information for this lab includes knowledge of parity since the encoder and decoder compare a combination of digits by using XOR. Further information on this process can be seen in the procedure. Verilog syntax for these labs can be accomplished using assign statements and instantiation. Using an encoder or decoder is extremely useful for design because there can be 128 combinations if a user enters a 4digit code into an interface and depending on their combination a unique outcome can be displayed or activated, or vice versa if it is a 7digit code or depending on the outcome a code needs to be generated.

Procedure (15 points)

The first step of both the encoder and decoder design is to understand hamming's algorithm. The process of taking a 4'b input and encoding a 7'b output is described below:



[figure 1: Encoder Logic Process]

A truth table can then be generated from this process as seen in the Lab 2 mannual:

d[4:1]	e[7:1]
4'b0000	7'b0000000
4'b0001	7'b0000111
4'b0010	7'b0011001
4'b0011	7'b0011110
4'b0100	7'b0101010
4'b0101	7'b0101101
4'b0110	7'b0110011
4'b0111	7'b0110100
4'b1000	7'b1001011
4'b1001	7'b1001100
4'b1010	7'b1010010
4'b1011	7'b1010101
4'b1100	7'b1100001
4'b1101	7'b1100110
4'b1110	7'b1111000
4'b1111	7'b1111111

[Figure 1b: Hamming Encoder Truth Table]

Similarly, the process for the decoder is as follows and the truth table can be developed which should match the truth table of the encoder:

Decoder Logic:

If the 7'b input reads 7'b 0 0 0 0 1 0 1

then Hamming claims 7'b d[7] d[6] d[5] d[4] d[3] d[2] d[1]

And the Bad Bits can be calculated as follows:

bad_bit[3] = d[7] ^ d[6] ^ d[5] ^ d[4] = 0

0 ^ 0 ^ 0 ^ 0

bad_bit[2] = d[7] ^ d[6] ^ d[5] ^ d[2] = 1

0 ^ 0 ^ 0 ^ 0

bad_bit[1] = d[7] ^ d[5] ^ d[3] ^ d[1] = 0

0 ^ 0 ^ 1 ^ 1

XOR Truth Table

a	b	z
0	0	0
0	1	1
1	0	1
1	1	0

Since bad_bit is bad_bit[2], this gets flipped from the original input

				bad_bit[1]		bad_bit[2]	bad_bit[3]
e[7:1]	d[4]	d[3]	d[2]	p3	d[2]	p2	p1
h_code[7:1]	h_code[7]	h_code[6]	h_code[5]	h_code[4]	h_code[3]	h_code[2]	h_code[1]
input	0	0	0	0	1	0	1
original	0	0	0	0	1	1	1

- The 7'b input of the decoder
- The 7'b output of the encoder

From this the Decoder tells us which Led will be turned on which represents the binary of the original 4'b input of the encoder.

A 1'b0 is then concatenated in front of bad_bit which will be 4'b0010 and this will display on the anode which bit is bad by converting from binary to an active low display as we did in Lab 3.

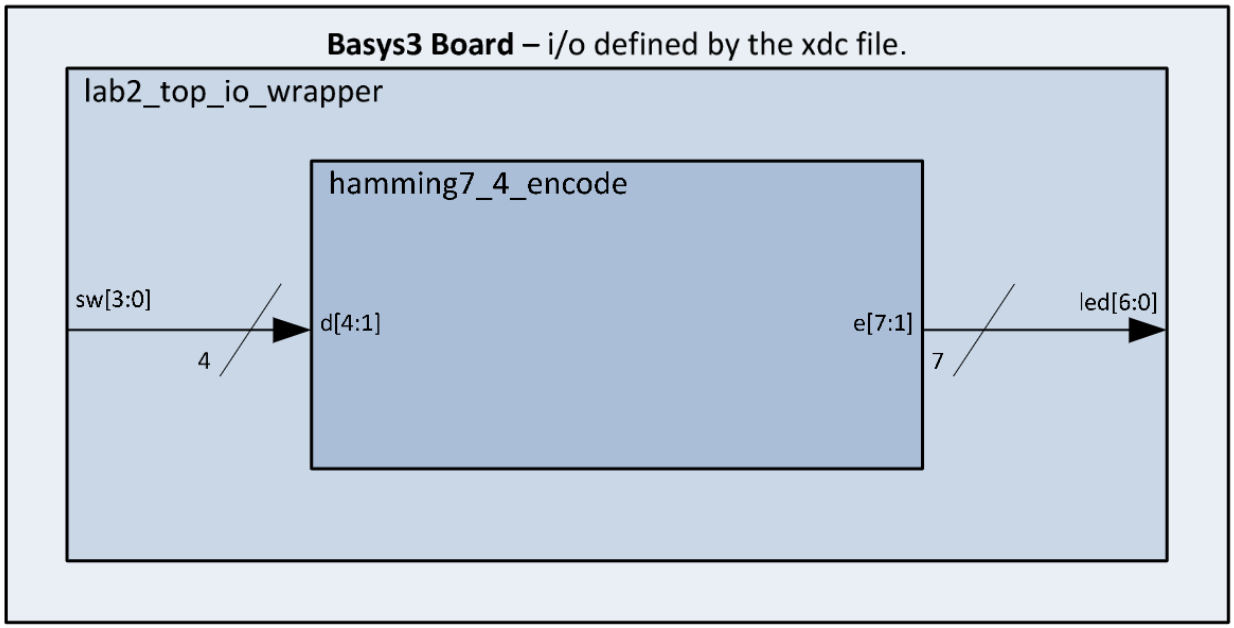
[Figure 2: Hamming Decoder Logic]

h_code[7:1]	decode[4:1]
7'b0000000	4'b0000
7'b0000111	4'b0001
7'b0011001	4'b0010
7'b0011110	4'b0011
7'b0101010	4'b0100
7'b0101101	4'b0101

h_code[7:1]	decode[4:1]
7'b0110011	4'b0110
7'b0110100	4'b0111
7'b1001011	4'b1000
7'b1001100	4'b1001
7'b1010010	4'b1010
7'b1010101	4'b1011
7'b1100001	4'b1100
7'b1100110	4'b1101
7'b1111000	4'b1110
7'b1111111	4'b1111

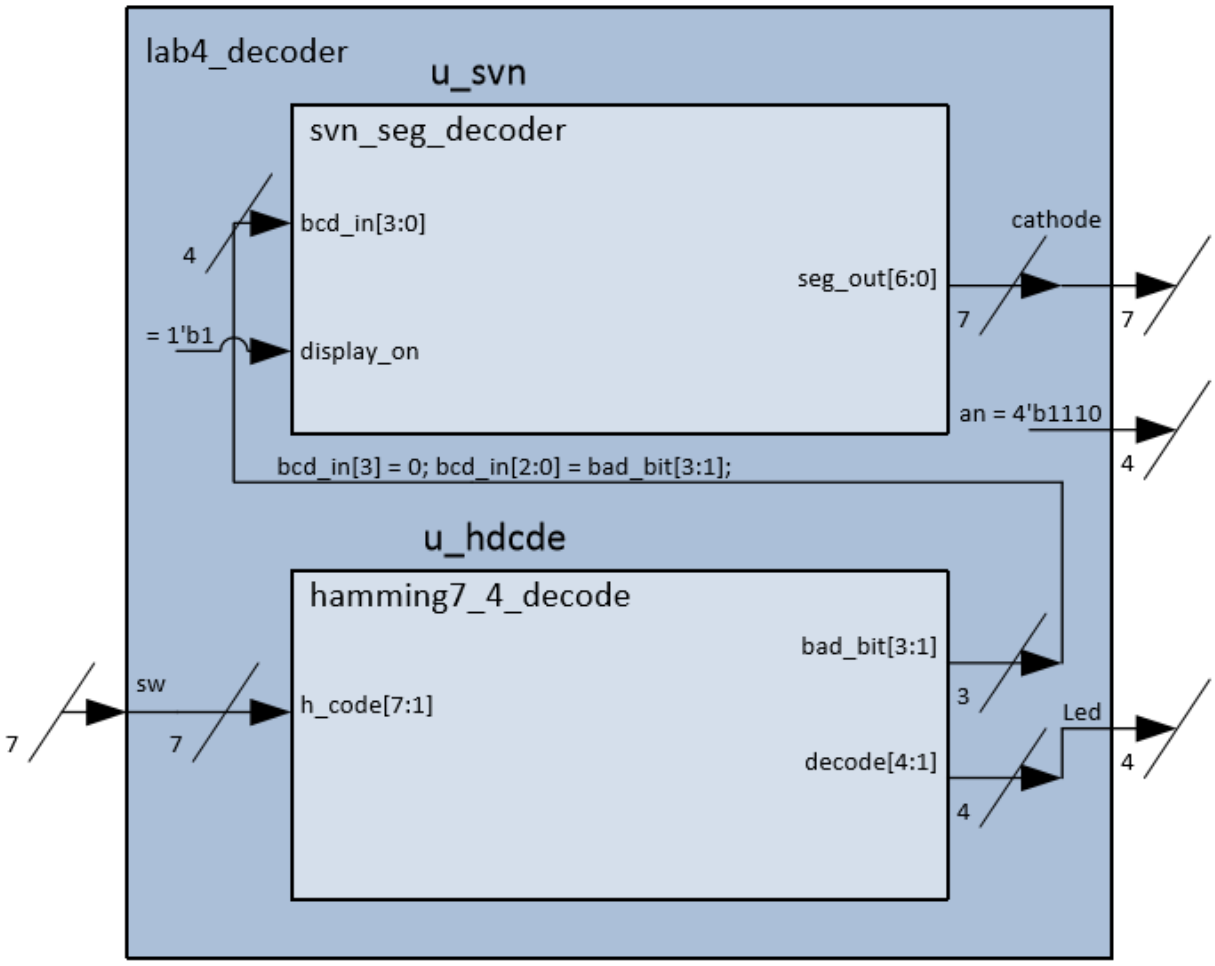
[Figure 2b: The Hamming Decoder Truth Table]

It is also worth mentioning how the modules are wired to the board by examining the block diagrams given by the lab manuals. The encoder takes in 4bits and outputs 7bits by running the code based on the logic that was described above and uses the Basys3 Board to generate the output by using switches (3:0):



[Figure 3: Block Diagram for the Encoder]

The Decoder works slightly different since it instantiates another module to display the original code through the LEDs and display which bit is bad by using the anodes:



[Figure 4: Block Diagram for the Decoder]

The simulation process is 100% correct since I can prove that these truth tables are equivalent, and both are uploaded into their respective testbenches and the hardware implementation proves to be correct.

Results (45 points – see sub-sections)

These labs helped me understand how to instantiate better because both of my code designs for each lab were pretty simply with maybe a slight syntax error here or there when using the assign statements, however instantiating the modules in lab 4 is what solidified my understanding of how to properly transfer a block diagram into Verilog by using . and parentheses. I was able to get 0 mismatches and my design matched the hardware implementation.

Design Code (15 points)

Lab 2 Design Code:

```
1  //
2  // lab2 : version 01/28/2020
3  //Robert Bara
4  `timescale 1ns / 1ps
5  module hamming7_4_encode(
6      output logic [7:1] e,
7      input logic [4:1] d
8      );
9      logic p1,p2,p3;
10     assign p1 = d[1] ^ d[2] ^ d[4];
11     assign p2 = d[1] ^ d[3] ^ d[4];
12     assign p3 = d[2] ^ d[3] ^ d[4];
13     assign e[7:1] = {d[4],d[3],d[2],p3,d[1],p2,p1};
14
15
16 endmodule
17
```

[Figure 5: Hamming Encoder Using Assign Statements and Xor Operator]

Lab 4 Design Code:

hamming7_4_de × | lab4_decoder.sv × | svn_seg_decode ×

```
1  //
2  // lab4 : version 02/11/2020 Robert Bara
3  //
4  `timescale 1ns / 1ps
5  //////////////////////////////////////////////////////////////////////////////////
6  //////////////////////////////////////////////////////////////////////////////////
7  module hamming7_4_decode(
8      output logic [3:1] bad_bit,
9      output logic [4:1] decode,
10     input logic [7:1] h_code
11     );
12
13     // insert your code here
14     assign bad_bit[3]=h_code[7]^h_code[6]^h_code[5]^h_code[4];
15     assign bad_bit[2]=h_code[7]^h_code[6]^h_code[3]^h_code[2];
16     assign bad_bit[1]=h_code[7]^h_code[5]^h_code[3]^h_code[1];
17     assign decode={h_code[7], h_code[6], h_code[5], h_code[3]};
18
19 endmodule
20
```

[Figure 6: Hamming Decoder using Assign Statements and Xor Operator]

hamming7_4_de × | lab4_decoder.sv × | svn_seg_decode ×

```
1  //
2  // lab4 : version 02/11/2020 Robert Bara
3  //
4  `timescale 1ns / 1ps
5  //////////////////////////////////////////////////////////////////////////////////
6  //////////////////////////////////////////////////////////////////////////////////
7  module lab4_decoder(
8      output logic [3:0] led,
9      output logic [3:0] an,
10     output logic [6:0] cathode,
11     input logic [6:0] sw
12     );
13
14     // insert your code here
15     logic [2:0] bad_bit;
16     assign an=4'b1110;
17     hamming7_4_decode u_hdcde(.decode(led),.bad_bit,.h_code(sw));
18     svn_seg_decoder u_svn (.seg_out(cathode),.display_on(1'b1),.bcd_in({1'b0,bad_bit[2:0]}));
19 endmodule
20
```

[Figure 6b: Instantiating the Decoder Module to Top Module and Svn Seg Decoder Module, (Block Diagram)]

Simulation Results (15 points)

Simulation for Lab 2:

bash - "tuj22026(× | lab2/hamming7_. × | lab2/hamming7_. × | lab2/tb_hamming × | lab2/lab2_top_io. ×

Run | Command: lab2/hamming7_4_encode.sim | Runner: simulate | CWD | ENV

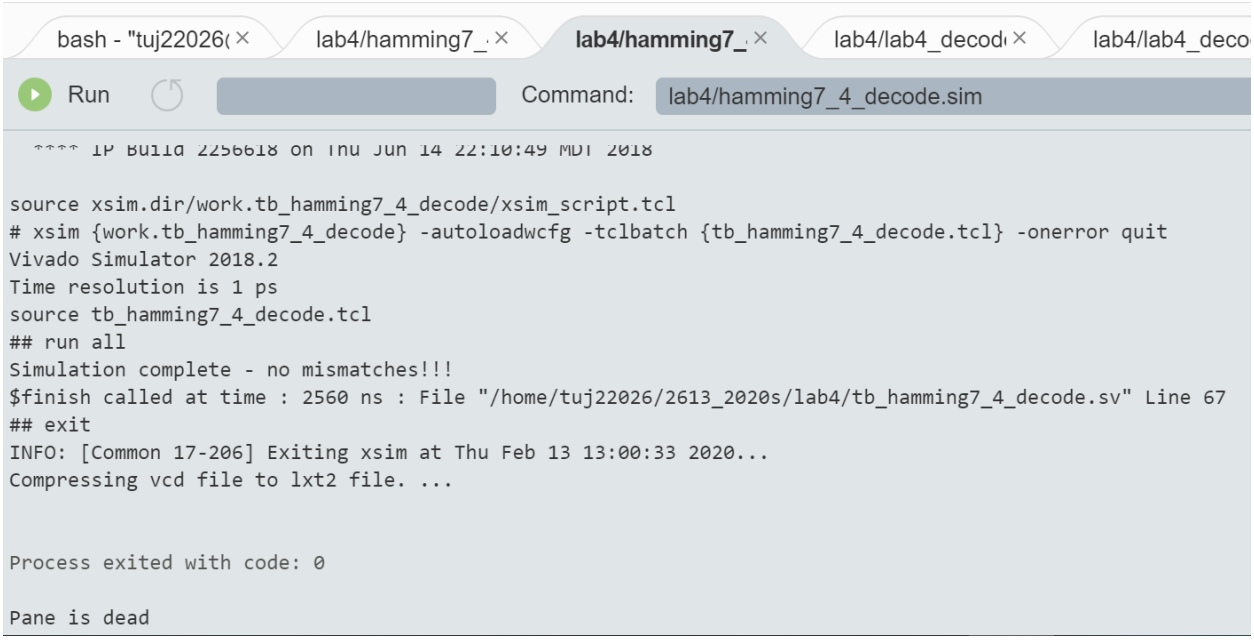
```
source /home/tuj22026/2613_2020s/lab2/xsim.dir/work.tb_hamming7_4_encode/webtalk/xsim_webtalk.tcl -notrace
INFO: [Common 17-186] '/home/tuj22026/2613_2020s/lab2/xsim.dir/work.tb_hamming7_4_encode/webtalk/usage_statistics_ext_xsim.xml' has been successfully sent to Xil
inx on Thu Jan 30 09:44:14 2020. For additional details about this file, please refer to the WebTalk help file at /data/courses/ece_2612/Xilinx/Vivado/2018.2/doc
/webtalk_introduction.html.
INFO: [Common 17-206] Exiting Webtalk at Thu Jan 30 09:44:14 2020...

****** 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
    ** Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.

source xsim.dir/work.tb_hamming7_4_encode/xsim_script.tcl
# xsim {work.tb_hamming7_4_encode} -autoloadwcfg -tclbatch {tb_hamming7_4_encode.tcl} -onerror quit
Vivado Simulator 2018.2
Time resolution is 1 ps
source tb_hamming7_4_encode.tcl
## run all
Simulation complete - no mismatches!!!
$finish called at time : 320 ns : File "/home/tuj22026/2613_2020s/lab2/tb_hamming7_4_encode.sv" Line 65
## exit
INFO: [Common 17-206] Exiting xsim at Thu Jan 30 09:44:25 2020...


Process exited with code: 0
Pane is dead
```

Simulation for Lab 4:



[Figure 7b: Decoder Simulation]

[Figure 7c: Instantiation of Modules Simulation]

Hardware Implementation (10 points)

For Lab 2, my design was demonstrated to Sivan on Thursday 1/30/2020 at 1:30pm during the Lab period.

For Lab 4, my design was demonstrated to Sivan on Thursday 2/13/20 at 2:15pm during the Lab period.

Conclusion (10 points)

Using assign statements and instantiation, I was able to successfully build a hamming encoder and decoder. I used the algorithm to create a series of xor statements which correspond to the truth tables created by parity and upon simulation I was able to upload it to the Basys3 Board and represent both the 4 bits and 7 bits depending on which code was active.