Lab 4 Submission

**BCD-to-Seven-Segment Decoder**

CPE 133 - 03

Michael Hegglin

Jonathan Skelly

**Executive Summary**

We designed a BCD-to-seven segment decoder to output digital numbers on a Digilent board LED display when specific switches are flipped. We implemented this circuit through Verilog.



This is the BCD-to-seven segment decoder black box diagram.

**Questions:**

**1. If you were not able to use a decoder in this experiment, how many concurrent signal assignments would you have needed to implement the segments portion of the seven segment display? Briefly explain.**

* Without the use of a decoder, 8 signal assignments would've had to have been made for every number to be displayed (7 segments and 1 decimal LED per number), leading to 10 combinations of 8 signals, or 80 assignments in total.

**2. Draw a black box diagram showing the four 7-segment display device(s), anodes, LEDs, segments, switches, and buttons on the development board. Be sure to carefully labeled inputs and outputs on your diagram.**



**3. Similar to 7-segment displays, there are also 14-segment displays out there in the real world. Briefly describe the main purpose served by 14-segment displays.**

* 14 segment displays show letters as well as numbers.

**4. This lab activity required that you use a generic decoder. You could have modeled this decoder using one of three possible types of concurrent statements. Provide the code for a generic decoder using one of the statements you did not use in this lab activity.**

always @(bcd)

begin

if (BCD == 0) segs = 8'b00000011;

else if (BCD == 1) segs = 8'b10011111;

else if (BCD == 2) segs = 8'b00100101;

else if (BCD == 3) segs = 8'b00001101;

else if (BCD == 4) segs = 8'b10011001;

else if (BCD == 5) segs = 8'b01001001;

else if (BCD == 6) segs = 8'b01000001;

else if (BCD == 7) segs = 8'b00011111;

else if (BCD == 8) segs = 8'b00000001;

else if (BCD == 9) segs = 8'b00011001;

else: segs = 8'b11111111;

end

**5. One of the important design approaches in modeling digital circuits is to use a LUT (decoder) whenever possible. Briefly describe why this is a good approach.**

* It is a good approach to use a LUT because it is basically a truth table where you can see the logical behaviors of your circuit. This makes it very easy to see the behavior of your circuit based off the inputs. It is also faster to look it up in a table from memory, then to do all the logic with the circuit.

**Design Problems:**

**1. Non-standard decoders are essentially LUTs. As you know from computer programming, often times using a LUT for calculations is a great idea. For this problem, show the code for a 4-input decoder that outputs the square of the input. Consider both input and output to be unsigned binary numbers. Use as few signals for the output as possible but still be able to represent the largest possible value for the output.**

**BBD of the BCD-to-squared value decoder.**

**Code:**

input [3:0] bcd;

output reg [7:0] sq;

always @(bcd)

begin

case (bcd)

0: sq = 8'b00000000;

1: sq = 8'b00000001;

2: sq = 8'b00000100;

3: sq = 8'b00001001;

4: sq = 8'b00010000;

5: sq = 8'b00011001;

6: sq = 8'b00100100;

7: sq = 8'b00110001;

endcase

End

**Source Code:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company: DogsWithJobs

// Engineer: Hegglin/Skelly

//

// Create Date: 10/05/2018 12:25:07 PM

// Design Name: LED Displayer

// Module Name: Lab\_4\_source

// Project Name: BCD-to-Seven-Segment Decoder

// Target Devices: Digilent Board

// Tool Versions: Verilog

// Description: Output digital numbers to an LED display when specific // switches are flipped

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module Lab\_4\_source(bcd, segs, an, switches);

input [3:0] bcd;

input [3:0] switches;

output reg [7:0] segs;

output [3:0] an;

assign an = switches;

always @(bcd)

begin

case (bcd)

0: segs = 8'b00000011;

1: segs = 8'b10011111;

2: segs = 8'b00100101;

3: segs = 8'b00001101;

4: segs = 8'b10011001;

5: segs = 8'b01001001;

6: segs = 8'b01000001;

7: segs = 8'b00011111;

8: segs = 8'b00000001;

9: segs = 8'b00011001;

default: segs = 8'b11111111;

endcase

end

endmodule