Lab 4: Convert 8-Bit Binary to BCD Display

Sign-Off Sheet

Date:

Student 1: _____ **ECE mailbox:** _____

Student 2: _____ **ECE mailbox:** _____

YOU ARE RESPONSIBLE TO COMPLETE ALL THE ASSIGNMENTS IN THE CHECK LIST BELOW IN ORDER TO GET FULL CREDIT FOR THE LAB…

After getting this sheet signed-off by the instructor/TA, please INDIVIDUALLY upload it on CANVAS to receive the GRADE!!

Check List	
Assignments	**TA Sign-off**
Pre-Lab (**MUST** be completed before the start of the lab) 1. Complete Add3 module 2. Complete BinarytoBCD module 3. Simulation Read Lab Write-up, Watch: *https://youtu.be/YM8s4SfHBPU*	
Lab part **Create Source File for the following Modules:** **i. BCD7SEG** **ii. Slow Clock** **iii. 2-bit Counter** **iv. 4 to 1 Mux** **v. Decoder** **vi. Top Module (put the system together)**	
Project File(s) Upload on CANVAS (Import all files in one folder, zip it, upload it) Upload your project file (individually) | Sign-up Sheet Upload the Writing Assignment (Debugging and Troubleshooting errors FAQ's)	

Due Date: 02/24/2020 [Sec C02]
02/25/2020 [Sec 01]

Both Students MUST be present at Sign-off for any and all parts!!

Prelab

Watch: *https://youtu.be/YM8s4SfHBPU*

1. Binary Coded Decimal (BCD) number system

You should now be familiar with the Binary, Decimal and Hexadecimal Number System. If we view single digit values for hex, the numbers 0 – F, they represent the values 0 - 15 in decimal. Often, we wish to use a binary equivalent of the decimal system. This system is called Binary Coded Decimal or BCD. Each decimal digit 0~9 is represented by 4 binary bits 0000 to 1001. In BCD number system, the binary patterns 1010 through 1111 do not represent valid BCD numbers, and cannot be used. For example, a decimal number 264_{10} can be represented as BCD numbers 001001100100_{BCD}.

2	6	4_{10}
0010	0110	0100_{BCD}

To convert from BCD to decimal, simply reverse the process as an example below:

1001	1000	0011_{BCD}
9	8	3_{10}

As you can see, BCD number system is designed for the convenience of showing decimal number using binary data. We will learn how to do that in this prelab.

2. Binary to BCD Converter

In this prelab, we will design a combinational circuit to convert an 8-bit binary number to 12-bit BCD. Why 12-bit for the BCD? The 8-bit binary number can present an integer from 0 to 255. Therefore, we need 3 decimal digits. As we see from Section 1, each decimal digit is represented by a 4-bit BCD. Therefore, we need 3*4 = 12-bit BCD.

The BCD number is particularly useful when we try to display numbers in decimal format. A binary to BCD converter will be used frequently in our future labs to display numbers. After all, people are so used to the decimal number format.

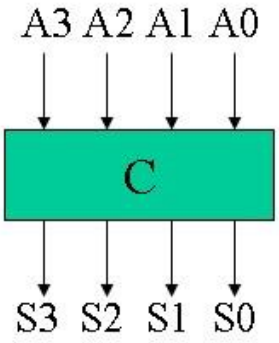
We begin to design the binary to BCD convert by introducing the **"shift and add-3 algorithm"**.

i. Shift the binary number left one bit.
ii. If 8 shifts have taken place, the BCD number is in the *Hundreds*, *Tens*, and *Units* column.
iii. If the binary value in any of the BCD columns is 5 or greater, add 3 to that value in that BCD column.
iv. Go to 1.

For example, let's convert FF (11111111) into BCD format by using the above algorithm.

Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1
Shift 1			1	1 1 1 1	1 1 1
Shift 2			1 1	1 1 1 1	1 1
Shift 3			1 1 1	1 1 1 1	1
Add 3			1 0 1 0	1 1 1 1	1
Shift 4		1	0 1 0 1	1 1 1 1	
Add 3		1	1 0 0 0	1 1 1 1	
Shift 5		1 1	0 0 0 1	1 1 1	
Shift 6		1 1 0	0 0 1 1	1 1	
Add 3		1 0 0 1	0 0 1 1	1 1	
Shift 7	1	0 0 1 0	0 1 1 1	1	
Add 3	1	0 0 1 0	1 0 1 0	1	
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		

Below is the truth table for the add-3 module:



A3	A2	A1	A0	S3	S2	S1	S0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

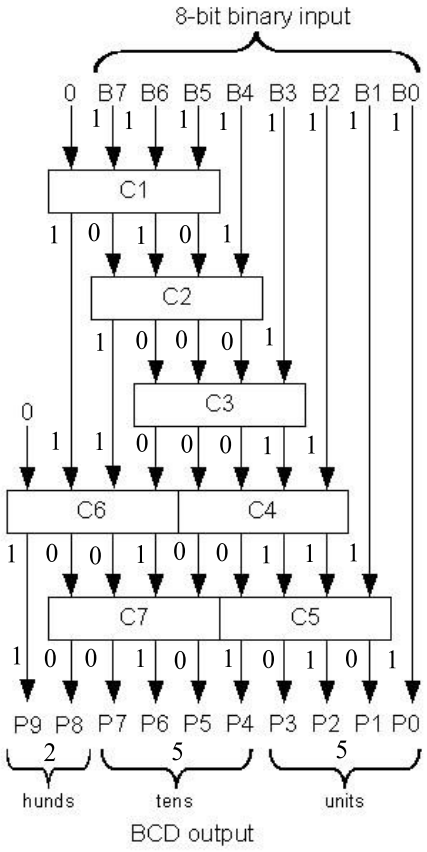
We can derive the logic expression using K-map, as we did in Lab 3, however, we'll choose behavioral model here. In Verilog, we can also simply list the truth table and the synthesis tool will analyze and implement it in an optimized form. Here is a Verilog module for this truth table.

```
module add3(
input [3:0] in,     //Four inputs
output reg [3:0] out);          //Declaring output as registers
always @ (in)          //Describing an event that should happen to input when certain condition is met.
case (in)     //The case  statement is a decision instruction that executes the statement.
        4'b0000: out <= 4'b0000;        //Here when all four input (in) bits (A3,A2,A1,A0) are Zeros (0000),
                                                out is 0000 (S3,S2,S1,S0)

        4'b0001: out <= 4'b0001;
        4'b0010: out <= 4'b0010;
        4'b0011: out <= 4'b0011;
        4'b0100: out <= 4'b0100;
        4'b0101: out <= 4'b1000; ////Here when all four input (in) bits (A3,A2,A1,A0) are 0101 (binary value
                                        is 5 or greater, we add 3), out is 8, 1000 (S3,S2,S1,S0).
```

Complete the code for 6, 7, 8, and 9.
Follow the truth table.

```
        default: out <= 4'b0000;

         endcase
        endmodule
```

The block diagram of the binary-to-BCD converter module is shown here.



//The 8-bit binary number can present an integer from 0 to 255. When all bits are high, it should display 255. C_{sign} (block in the figure) represents Add 3 module.

//Here we have two-bits for hundreds because the value cannot exceed 2, and therefore we can generate 2 by two bits only.

Here we have 7 add-3 components. To skip all the details, here is a structural Verilog module corresponding to the logic diagram.

module binary_to_BCD(

input [7:0] A, //These refers to B0 to B7 as shown above in the circuit.
output [3:0] ONES, //We need 4 bits to display a digit for ones as it could go from 0 to 9. output [3:0] TENS, //We need 4 bits to display a digit for tens as it could go from 0 to 9.
output [1:0] HUNDREDS); //We need to bits to display a digit for hundreds as it could go from 0 to 2.

wire [3:0] c1,c2,c3,c4,c5,c6,c7; //Declaring data lines coming out of each add 3 module as wires.

wire [3:0] d1,d2,d3,d4,d5,d6,d7; //Declaring data lines going into each add 3 module as wires.

//Follow the Block Diagram

assign d1 = {1'b0,A[7:5]}; //LMB is 0. A[7:5] -->Inputs refer to B7, B6, B5 going into C1

```
assign d2 = {c1[2:0],A[4]};     //Inputs going into C2.
                                //Inputs going into C3.
                                //Inputs going into C4.
                                //Inputs going into C5.
assign d6 = {1'b0,c1[3],c2[3],c3[3]};          //LMB is 1. Inputs going into C6.
assign d7 = {c6[2:0],c4[3]};            //Inputs going into C7.

add3 m1(d1,c1);          //Using add3 module that you created above to perform
                         operation on csign and dsign on all 7 modules.
add3 m2(d2,c2);
```

Complete the code…d3, d4, d5

Complete the code… add 5 more add3 modules…

```
assign ONES = {c5[2:0],A[0]};  //four bits that will make-up ones.

assign TENS = {c7[2:0],c5[3]};  //four bits that will make-up tens.

assign HUNDREDS =                    ; //two bits that will make-up hundreds,
finish the line, follow the block diagram.

endmodule
```

3. Simulate the binary-to-BCD converter in Vivado

The aformentioned binary-to-BCD converter will be used in Lab 4. In this prelab, you need to perform the following:

1. Create a new project using Xilinx Vivado software. All ECE lab computers are installed with Xilinx software packages. Alternatively, you can download the Xilinx webpack free. Please note that the software download can take very long due to its size;
2. Add the given Verilog module binary_to_BCD to the project;
3. Use create an appropriate testbench instantiating the module;
4. Edit the testbench by add different 8-bit stimulus input;
5. Use the simulation waveform to verify if the output data in BCD format is correct. Since the ONES, TENS, and HUNDRES are vector format, in simulation waveform you can click the right button to select view data in unsigned decimal format. So each will be display as a decimal digit 0~9. This may help you to read the BCD output data.
6. Save your project.

Show the lab /instructor/TA that your binary-to-BCD converter is working in Vivado.