EE417 Advanced Digital Logic with Verilog HDL
L3 Assignment I: BCD to Xcess3 Converter

Biased codes (and Gray codes) are non-weighted codes.
In excess-3 code, numbers are represented as decimal digits, and each digit is represented by four bits as the digit value plus 3 (the "excess" amount): The smallest binary number represents the smallest value (0 – excess).

3. Behavioral Design Assignments

Due Jun 2 at 11:59pm **Points** 100 **Questions** 3 **Time Limit** None

Instructions

The main objective of the assignment is to practice and implement different methods for describing combinational logic in Verilog:

1. In this set of assignments, we will practice how to use the assign operator to define the logical expression for an output. In the structural design with basic primitive logic gates assignment, we instantiated AND, OR, NOT, and XOR gates and used interconnecting wires to implement the switching logic. Using the assign operator allows us to describe the hardware in a more compact code. This is implemented in question 1 converting a BCD to Xcess3 code. This example also introduces you to the Xcess code and covers an overview of the advantages of this code.

2. The assignment also covers the implementation of case structures to implement a hardware design while bypassing the K-map step for finding the SOP (Sum of Products) expressions. Question 2 introduces the students to the 2421 code and uses the case structure to implement the conversion.

3. Displaying 4-bit binary input on two seven-segment-displays covering numbers from 0 to 15. This design requires the implementation of comparators, multiplexers (using the conditional assignment) to have the correct numbers displayed.

For each design, you will create a testbench to verify the correct functionality of the design.

This is a group assignment (groups of 2 students). Each group will submit one document listing the contribution of each member.

Excess-3 codes are unweighted and can be obtained by adding 3 to each decimal digit then it can be represented by using 4 bit binary number for each digit.

Excess-3 code is a self-complementary code. The 1's complement of a binary number can be obtained by replacing 0's with 1's and 1's with 0's. For self-complementary codes, the sum of a binary number and its complement is always equal to decimal 9. This means that the 1's complement of an excess-3 code is the excess-3 code for the 9's complement of the corresponding decimal number.

For example, the excess-3 code for decimal number 5 is 1000 (5+3 = 8) and 1's complement of 1000 is 0111, which is excess-3 code for decimal number 4 (4+3 = 7), and it is the 9's complement of number 5.

The primary advantage of excess-3 coding is that a decimal number can be nines' complemented (for subtraction) just by inverting all bits. Also, when the sum of two excess-3 digits is greater than 9, the carry bit of a 4-bit adder will be set high. This works because, after adding two digits, an "excess" value of 6 results in the sum. Because a 4-bit integer can only hold values 0 to 15, an excess of 6 means that any sum over 9 will overflow (produce a carry out).

Another advantage is that the codes 0000 and 1111 are not used for any digit. A fault in a memory or basic transmission line may result in these codes, as they stay at the same logic level without changing.

Truth Table for BCD (B) to Excess-3 (X)

Decimal value	BCD – 8421 weighted code	Excess-3 code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

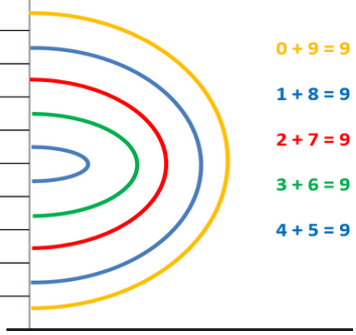
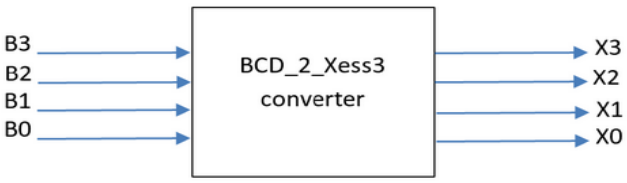


Diagram for pinout of BCD to Excess3 Converter



(a) Use **Gate-level Verilog** code to design the combinational logic for the ***BCD_2_Xess3*** converter module. Show your work including the Karnaugh maps for all output bits. Assume that the inputs will only have digits from 0 to 9. Values 10 to 15 can be considered "don't care" cases. ***Hint: You need to sketch 4 K-maps, one for each bit of the Excess_3_code.***

(b) Create a test-bench ***BCD_2_Xess3_tb*** to test your design. The testbench should display all the possible input binary combinations along with the corresponding Excess-3 code.

Submit one pdf file that includes the Karnaugh maps for the 4 outputs, a screenshot of the Verilog code for *BCD_2_Xess3* and ***BCD_2_Xess3_tb*** typed in Quartus, and *the simulation results generated for the test-bench, Briefly (in one or two sentences) verify the simulation results and include the RTL schematics implementing your switching functions.*

Division of Responsibility

Assignment	Map/ Equations /Code	TestBench
1	Kalin	Jammeh
2	?	?
3	?	?

K-map - bit X3 = (B0B2) + B3 + (B1B2)

B1B0\B3B2	00	01	11	10
00	0	0	X	1
01	0	1	X	1
11	0	1	X	X
10	0	1	X	X

K-map - bit X2 = (B1'B0'B2) + (B0B2') + (B1B2')

B1B0\B3B2	00	01	11	10
00	0	1	X	0
01	1	0	X	1
11	1	0	X	X
10	1	0	X	X

K-map - decimal value

B1B0\B3B2	00	01	11	10	
00	0	4	12	8	x0=B2
01	1	5	13	9	x1=B0
11	3	7	15	11	x2=B3
10	2	6	14	10	x3=B1

K-map - bit X1 = B0'B1' + B0B1 = BO XNOR B1

B1B0\B3B2	00	01	11	10
00	1	1	X	1
01	0	0	X	0
11	1	1	X	X
10	0	0	X	X

K-map - bit X0 = B0'

B1B0\B3B2	00	01	11	10
00	1	1	X	1
01	0	0	X	0
11	0	0	X	X
10	1	1	X	X

Resulting simplified equation

X3 = (B0B2) + B3 + (B1B2)
X2 = (B1'B0'B2) + (B0B2') + (B1B2')
X1 = B0'B1' + B0B1 = BO XNOR B1
X0 = B0'

Module code

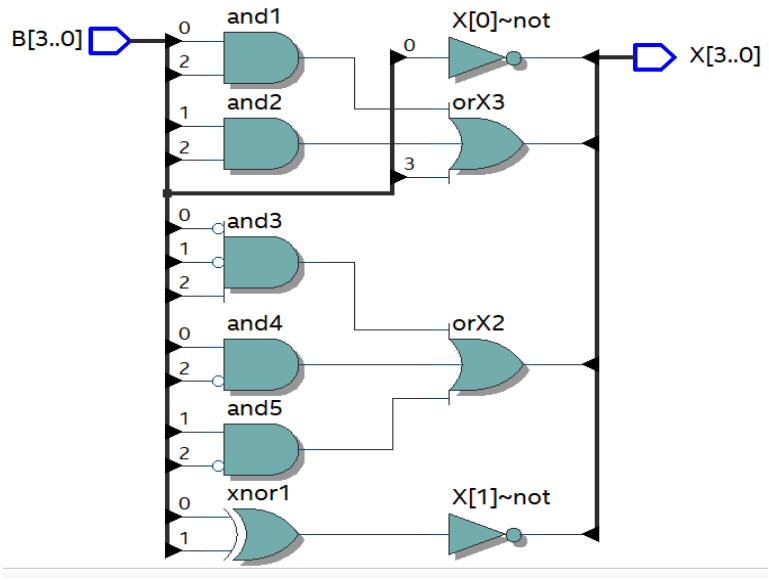
Date: June 01, 2024 BCD_to_Xess3.v Project: BCD_to_Xess3

```
1   // Name:  Ron Kalin, Date: 5-29-24, Design: Lesson 3A1: BCD_to_Xess3
2   // Group: Ron Kalin/Lamin Jammeh _____
3   module BCD_to_Xess3 ( input [3:0] B,
4                         output [3:0] X); //B=BCD, X=Xcess3
5   //output reg sw1;
6   //wire notB0, notB1, notB2, notB3; //declare wires
7   //wire and1X3, and2X3;
8   //wire and1X2, and2X2, and3X2;
9   //wire xnorX1;
10
11  //always @ (sw1)
12  //sw1 = 1'b1; //can change to zero if assign SOP method is desired
13  //if (sw1 == 1) begin //if (switch)
14
15  //  not not0 (notB0, B[0]); //not gates
16  //  not not1 (notB1, B[1]);
17  //  not not2 (notB2, B[2]);
18  //  not not3 (notB3, B[3]);
19
20  //  and and1 (and1X3, B[0], B[2]);  //and gates
21  //  and and2 (and2X3, B[1], B[2]);
22  //  and and3 (and1X2, notB1, notB0, B[2]);
23  //  and and4 (and2X2, B[0],notB2);
24  //  and and5 (and3X2, B[1],notB2);
25
26  //  xnor xnor1 (xnorX1, B[0], B[1]); //xnor gate
27
28  //  or orX3 (X[3], and1X3, B[3], and2X3);  //or gates to give output bits
29  //  or orX2 (X[2], and1X2, and2X2, and3X2);
30  //  assign X[1] = xnorX1;
31  //  assign X[0] = notB0;
32  //end
33
34  //else if (sw1==1'b0) begin
35  //equations below  can be used in lieu of gate logic above
36  //X3 = (B0B2) + B3 + (B1B2)
37  assign X[3] = (B[0] & B[2]) |  B[3] | (B[1] & B[2]);
38  //X2 = (B1'B0'B2) + (B0B2') + (B1B2')
39  assign X[2] = (!B[1]& !B[0] & B[2]) | (B[0] & !B[2]) | (B[1] & !B[2]);
40  //X1 = B0'B1' + B0B1 = BO XNOR B1
41  assign X[1] = B[0] ^~ B[1];
42  //X0 = B0'
43  assign X[0] = !B[0];
```

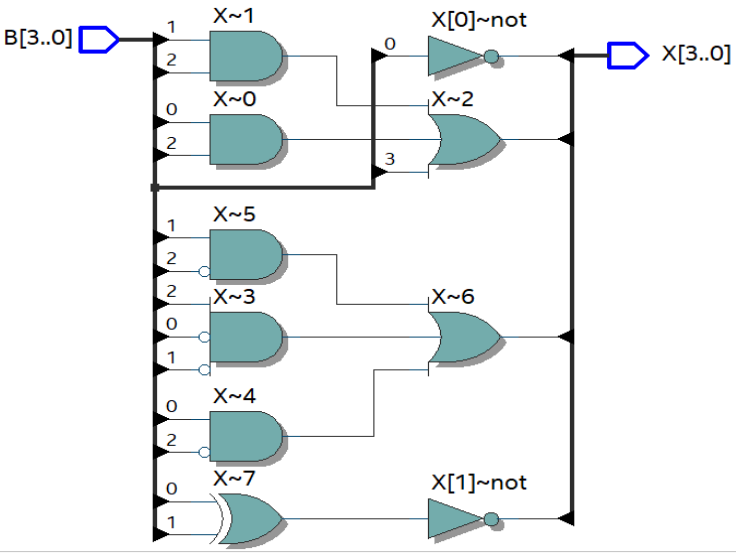
```
44    //end
45    endmodule
46
```

This code was written two ways. Using gate logic (commented out) and using operator SOP logic function equations (chosen, not commented out).

Netlist RTL Viewer Using Gate Logic



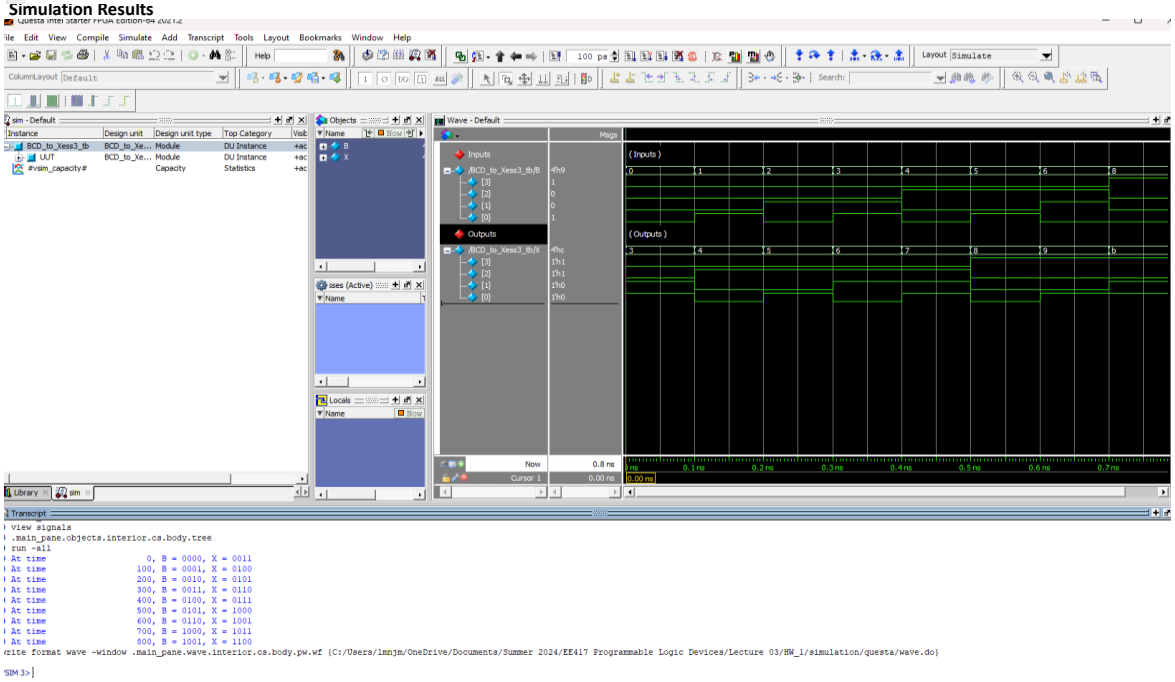
Using assign operator SOP logic function Equations



testbench BDC_to_Xcess3

```
/*--------------------------------------------------------------------------------
Name Lamin Jammeh
CLass: EE417 Summer 2024
Lesson 03 HW Question 1
Group: Ron Kalin/ Lamin Jammeh
Main design was done by Ron Kalin
TestBench was done by Lamin Jammeh
--------------------------------------------------------------------------------*/

//Step1 define a name for the test-bench
module BCD_to_Xess3_tb;

//Step2 define the inputs as registers and outputs as wires
reg [3:0] B;
wire [3:0] X;

//Step3 define the Unit under test UUT with all inputs and outputs
BCD_to_Xess3 UUT (
                        .B(B),
                        .X(X)
                        );

//Step4 open an initial block and define all the possible input conbination the BCD to Xess3 from 0-9
initial
    begin
            B = 4'b0000;                //define the input B as 4 bits with all zero binary at the start
        #100 B = 4'b0001;
        #100 B = 4'b0010;
        #100 B = 4'b0011;
        #100 B = 4'b0100;
        #100 B = 4'b0101;
        #100 B = 4'b0110;
        #100 B = 4'b1000;
        #100 B = 4'b1001;
        end
initial
    begin
            $monitor("At time %t, B = %b, X = %b", $time, B, X);
     end
endmodule
```

Simulation Results



Briefly (in one or two sentences) verify the simulation results and include the RTL schematics implementing your switching functions.
Simulation results verify that Xcess3 is BCD plus 3 or (0011 in binary)