***LAB 02: Numbers and Displays***

**Task 1: Hex Display via Switches**

File: lab02part01.v, B\_7SegDec.v

FPGA: DONE.

Code:

MAIN

module lab02part01(

input [7:0]SW,

output [6:0]HEX1, HEX0

);

/\*

As we only have 10 switches available, so using only 2 dispalys (8 switches)

\*/

B\_7SegDec num1(SW[7:4], HEX1[6:0]);

B\_7SegDec num2(SW[3:0], HEX0[6:0]);

endmodule

TESTBENCH

// VERIFICATION

module testBenchL2P1;

reg [7:0]sw;

wire [6:0]h0, h1;

lab02part01 siml1p1(sw, h1, h0);

initial begin

#100; sw = 8'b00000000;

#100; sw = 8'b00000001;

#100; sw = 8'b00000010;

#100; sw = 8'b00000011;

#100; sw = 8'b00000100;

#100; sw = 8'b00000101;

#100; sw = 8'b00000110;

#100; sw = 8'b00000111;

#100; sw = 8'b00001000;

#100; sw = 8'b00001001;

#100; sw = 8'b00001010;

#100; sw = 8'b00010000;

#100; sw = 8'b00100000;

#100; sw = 8'b00110000;

#100; sw = 8'b01000000;

#100; sw = 8'b01010000;

#100; sw = 8'b01100000;

#100; sw = 8'b01110000;

#100; sw = 8'b10000000;

#100; sw = 8'b10010000;

end

endmodule

7 SEGMENT DECODER

module B\_7SegDec(

input [3:0] X,

output [6:0] Y

);

assign Y[6] = ((~X[3] & ~X[2] & ~X[1]) |(X[2] & X[1] & X[0]));

assign Y[5] = ((~X[2] & X[1])|(X[1] & X[0])|(~X[3] & ~X[2] & X[0]);

assign Y[4] = ((X[0])|(X[2] & ~X[1]));

assign Y[3] = ((X[2] &~X[1] & ~X[0])|(X[2] & X[1] & X[0])|(~X[3] & ~X[2] & ~X[1] & X[0]));

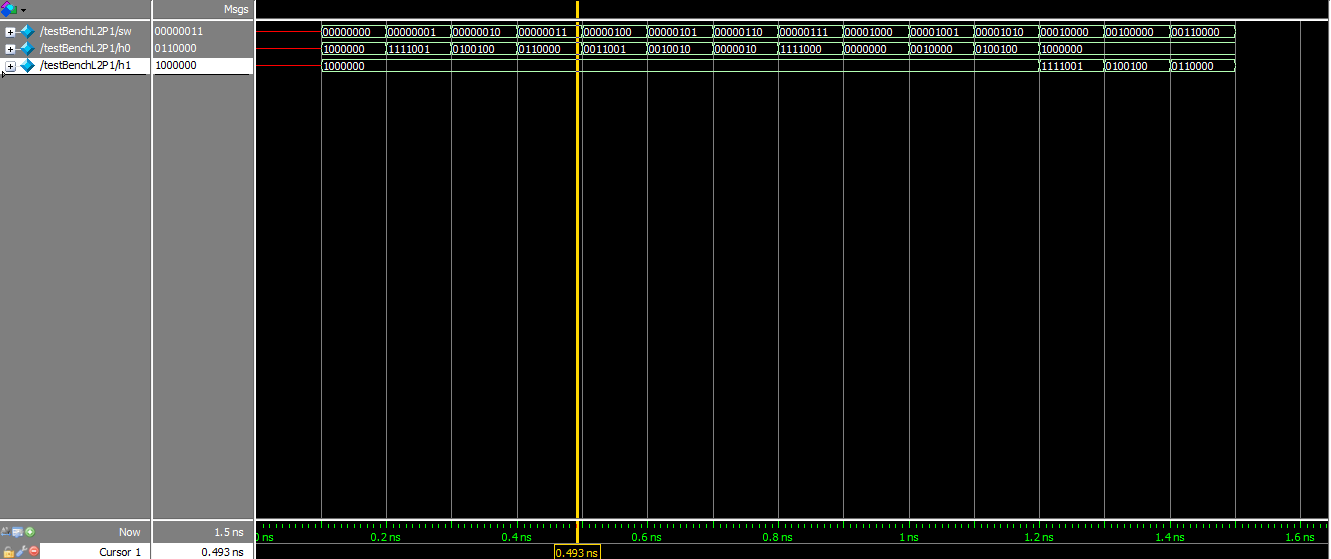
assign Y[2] = (~X[2] & X[1] & ~X[0]);

assign Y[1] = ((X[2] & ~X[1] & X[0])|(X[2] & X[1] & ~X[0]));

assign Y[0] = ((X[2] & ~X[1] & ~X[0])|(~X[3] & ~X[2] & ~X[1] & X[0]));

endmodule

Output:



**Task 2: Decimal Digit Separation**

File: lab02part02.v, B\_7SegDec.v, lab02part02\_circuitA.v, B\_1xMUX2to1.v, B\_is2Digit.v, lab02part02\_table\_circuitA.xlsx

FPGA: X

Code:

MAIN

module lab02part02(

input [3:0]SW,

output [6:0]HEX1, HEX0

);

wire Z;

wire [2:0]A;

wire [3:0]M;

B\_is2Digit isgth9(SW[3:0], Z);

B\_7SegDec digit1({3'b000,Z}, HEX1[6:0]);

lab02part02\_circuitA adjst(SW[2:0], A[2:0]);

B\_1xMUX2to1 m3(Z, SW[3], 1'b0, M[3]);

B\_1xMUX2to1 m2(Z, SW[2], A[2], M[2]);

B\_1xMUX2to1 m1(Z, SW[1], A[1], M[1]);

B\_1xMUX2to1 m0(Z, SW[0], A[0], M[0]);

B\_7SegDec digit0(M[3:0], HEX0[6:0]);

endmodule

TESTBENCH

//VERIFICATION

module testBenchl2p2;

reg [3:0]sw;

wire [6:0]h1, h0;

lab02part02 testp2(sw, h1, h0);

initial begin

sw = 4'b0000; #100;

sw = 4'b0001; #100;

sw = 4'b0010; #100;

sw = 4'b0011; #100;

sw = 4'b0100; #100;

sw = 4'b0101; #100;

sw = 4'b0110; #100;

sw = 4'b0111; #100;

sw = 4'b1000; #100;

sw = 4'b1001; #100;

sw = 4'b1010; #100;

sw = 4'b1011; #100;

sw = 4'b1100; #100;

sw = 4'b1101; #100;

sw = 4'b1110; #100;

sw = 4'b1111; #100;

end

endmodule

Is2Digit

module B\_is2Digit(

input [3:0]num,

output z

);

/\*

Checks if the given 4-bit number is greater than 9.

\*/

assign z = (num[3] & (num[2] | num[1] | num[0]));

endmodule

circuitA

module lab02part02\_circuitA(

input [2:0]num,

output [2:0]out

);

/\*

2 AB

1 B'

0 C

\*/

assign out[2] = (num[2] & num[1]);

assign out[1] = (~num[1]);

assign out[0] = (num[0]);

endmodule

1xMUX2to1

module B\_1xMUX2to1(

input S, X, Y,

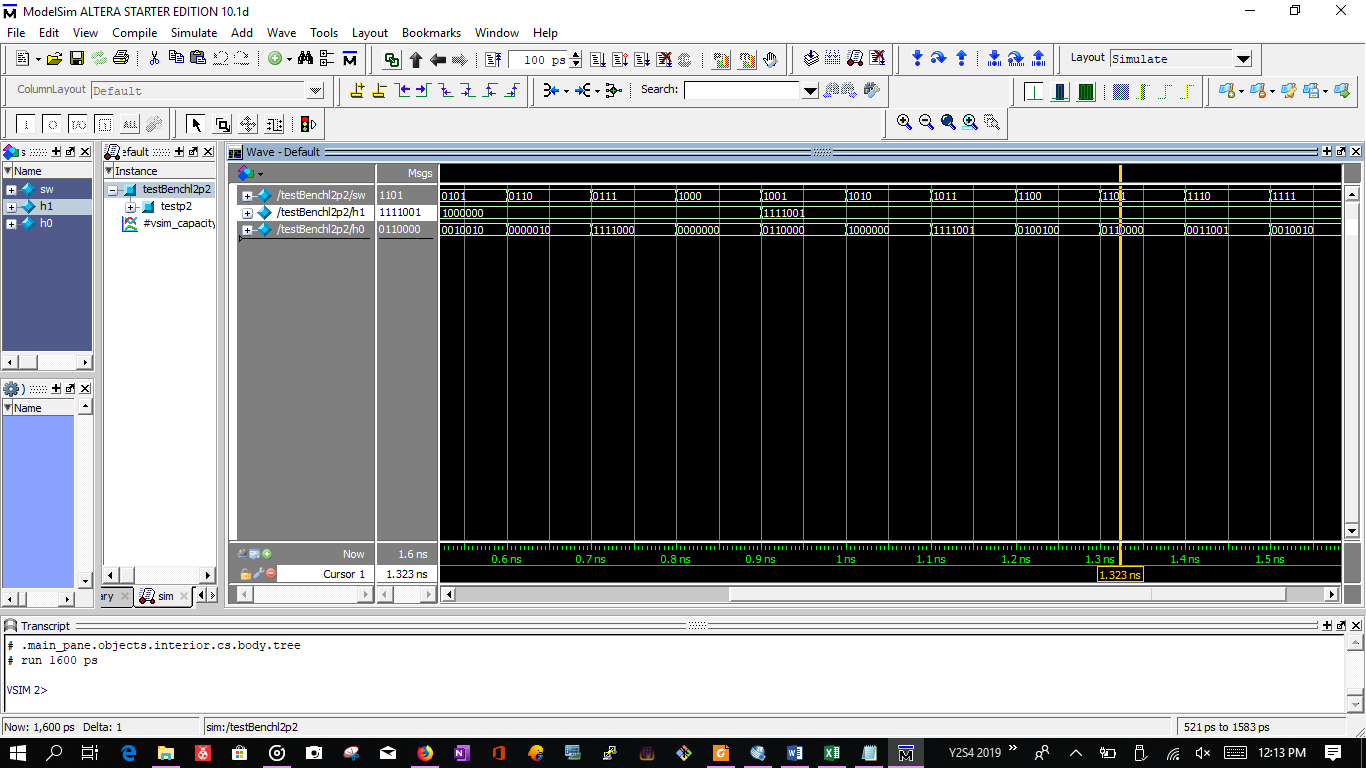
output M

);

assign M = ((~S & X) | (S & Y));

endmodule

Simulation:



**Task 3: 4-Bit Ripple Full Adder**

File: lab02part03.v, B\_4xFullAdderRipple, B\_1xHalfAdder.v, lab02part03\_tableHA.txt

FPGA: X

Code:

MAIN

module lab02part03(

input [8:0]SW,

output [8:0]LEDR,

output [4:0]LEDG

);

B\_4xFullAdderRipple addAB(SW[8], SW[7:4], SW[3:0], LEDG[4], LEDG[3:0]);

assign LEDR[8:0] = SW[8:0];

endmodule

TESTBENCH

// VERIFICATION

module testBenchl1p3;

reg [8:0]sw;

wire [8:0]lr;

wire [4:0]lg;

lab02part03 verifyl2p3(sw, lr, lg);

initial begin

sw={1'b0, 4'b0000, 4'b0000}; #100;

sw={1'b0, 4'b0000, 4'b0001}; #100;

sw={1'b0, 4'b0001, 4'b0000}; #100;

sw={1'b1, 4'b0010, 4'b0001}; #100;

sw={1'b1, 4'b0100, 4'b0001}; #100;

sw={1'b1, 4'b1000, 4'b1000}; #100;

sw={1'b1, 4'b1111, 4'b0000}; #100;

end

endmodule

FULL ADDER 4-BIT RIPPLE

module B\_4xFullAdderRipple(

input CIN,

input [3:0]A, B,

output COUT,

output [3:0]S

);

wire [2:0]carryAhead;

B\_1xFullAdder sum1(A[0], B[0], CIN, carryAhead[0], S[0]);

B\_1xFullAdder sum2(A[1], B[1], carryAhead[0], carryAhead[1], S[1]);

B\_1xFullAdder sum3(A[2], B[2], carryAhead[1], carryAhead[2], S[2]);

B\_1xFullAdder sum4(A[3], B[3], carryAhead[2], COUT, S[3]);

Endmodule

FULL ADDER 1-BIT

module B\_1xFullAdder(

input A, B, C,

output cot, sum

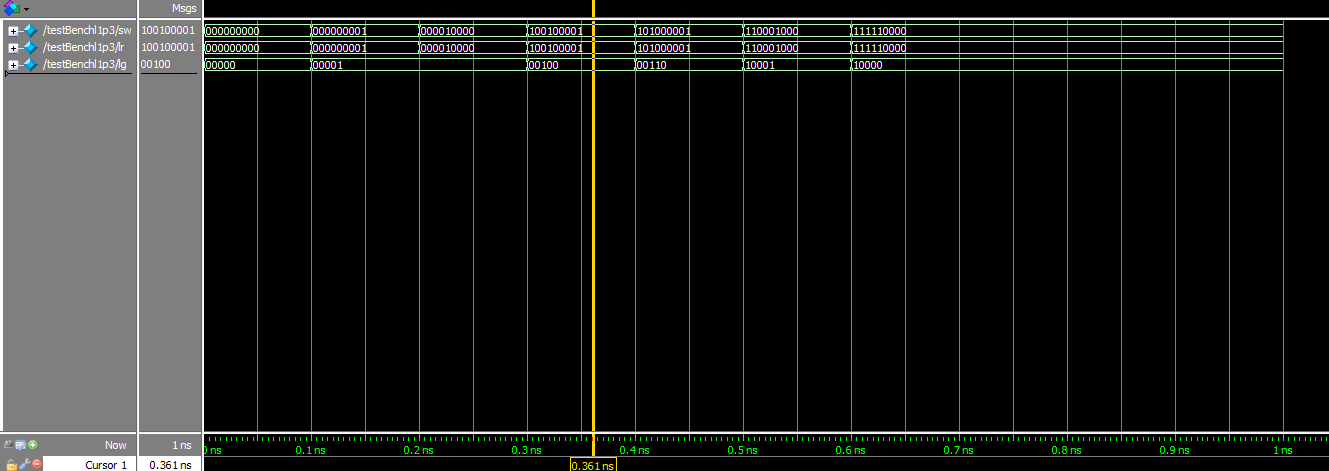
);

assign cot = ((A & B) | (B & C) | (C & A));

assign sum = ((~A & ~B & C) | (~A & B & ~C) | (A & ~B & ~C) | (A & B & C));

endmodule

Simulation:



**Task 4: 4-bit BCD Adder**

File: lab02part04.v, B\_4xComp.v, B\_8xComp.v, B\_4xMUX2to1.v, lab02part04\_table.xlsx

FPGA: X

Code:

MAIN

module lab02part04(

input [8:0]SW,

output [4:0]LEDG, //BCD Sum

output [8:0]LEDR, //Output

output [6:0]HEX3, HEX2, HEX1, HEX0

);

//Showing IN on HEX

B\_7SegDec showA(SW[7:4], HEX3[6:0]);

B\_7SegDec showB(SW[3:0], HEX2[6:0]);

//adding the BCD Numbers

wire [4:0]adderSum;

B\_4xFullAdderRipple addAB(SW[8], SW[7:4], SW[3:0], adderSum[4], adderSum[3:0]);

assign LEDR[8:0] = SW[8:0]; //LED Rep of IN

assign LEDG[4:0] = adderSum[4:0];//LED Rep of OUT

//Z determination

wire L, E, G;

B\_8xComp comparator({3'b000,adderSum}, {8'h10}, L, E, G);

B\_7SegDec digitTens({3'b000, G}, HEX1[6:0]);

//Final SUM

wire [3:0]adjSum, BCDSumS0;

lab02part04\_cA adjst(adderSum[3:0], adjSum[3:0]);

B\_4xMUX2to1 selS(G, adderSum[3:0], adjSum[3:0], BCDSumS0[3:0]);

B\_7SegDec digitUnits(BCDSumS0[3:0], HEX0[6:0]);

endmodule

TESTBENCH

// VERIFICATION

module testBenchl2p4;

reg [8:0]sw;

wire [4:0]lg;

wire [8:0]lr;

wire [6:0]h3,h2,h1,h0;

lab02part04 verifyl2p4(sw, lg, lr, h3, h2, h1, h0);

initial begin

sw = {1'b0, 4'b0000, 4'b0000}; #100;

sw = {1'b0, 4'b0001, 4'b0001}; #100;

sw = {1'b0, 4'b0001, 4'b1001}; #100;

sw = {1'b0, 4'b1000, 4'b1000}; #100;

sw = {1'b1, 4'b1001, 4'b1001}; #100;

end

endmodule

CIRCUIT A

module lab02part04\_cA(

input [3:0]X,

output [3:0]Y

);

assign Y[3] = ((~X[3] & X[1]));

assign Y[2] = ((X[3] & ~X[1]) | (X[2] & X[1]));

assign Y[1] = (~X[1]);

assign Y[0] = X[0];

endmodule

8-BIT COMPARATOR

module B\_8xComp(

input [7:0]numA, numB,

output LT, EQ, GT

);

wire [2:0]resCompL;

B\_4xComp compL(1'b0, 1'b1, 1'b0, numA[3:0], numB[3:0], resCompL[2], resCompL[1], resCompL[0]);

B\_4xComp compH(resCompL[2], resCompL[1], resCompL[0], numA[7:4], numB[7:4], LT, EQ, GT);

endmodule

4-BIT COMPARATOR (CASCADED)

module B\_4xComp(

input Ll, El, Gl,

input [3:0]numA, numB,

output L, E, G

);

wire x0, x1, x2, x3; //XNOR

wire Lh, Eh, Gh;

assign x0 = ((numA[0] & numB[0]) | (~numA[0] & ~numB[0]));

assign x1 = ((numA[1] & numB[1]) | (~numA[1] & ~numB[1]));

assign x2 = ((numA[2] & numB[2]) | (~numA[2] & ~numB[2]));

assign x3 = ((numA[3] & numB[3]) | (~numA[3] & ~numB[3]));

assign Eh = (x0 & x1 & x2 & x3);

assign Gh = ((numA[3] & ~numB[3]) | (x3 & numA[2] & ~numB[2]) |

(x3 & x2 & numA[1] & ~numB[1]) | (x3 & x2 & x1 & numA[1] & ~numB[1]));

assign Lh = ((~numA[3] & numB[3]) | (x3 & ~numA[2] & numB[2]) |

(x3 & x2 & ~numA[1] & numB[1]) | (x3 & x2 & x1 & ~numA[1] & numB[1]));

assign E = (Eh & El);

assign L = ((Eh & Ll) | Lh);

assign G = ((Eh & Gl) | Gh);

endmodule

4-BIT MUX 2x1

module B\_4xMUX2to1(

input S,

input [3:0]numA, numB,

output [3:0]M

);

B\_1xMUX2to1 m3(S, numA[3], numB[3], M[3]);

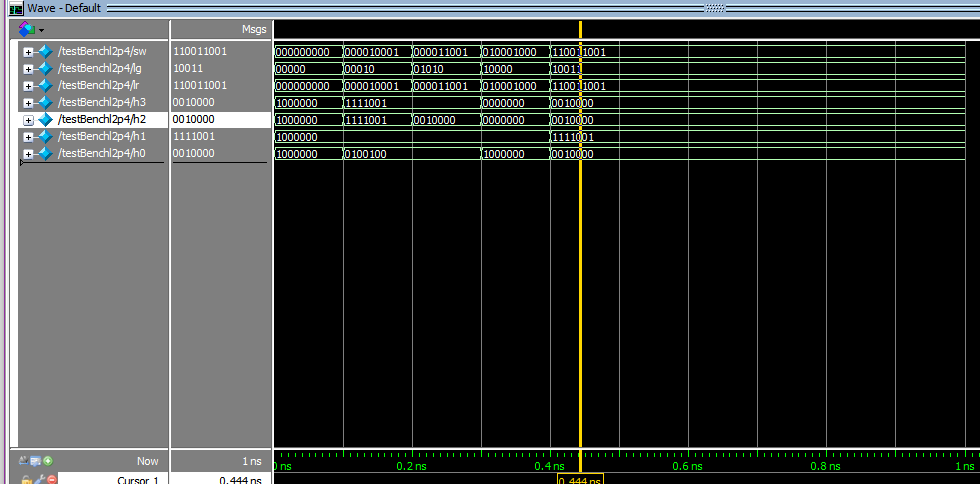
B\_1xMUX2to1 m2(S, numA[2], numB[2], M[2]);

B\_1xMUX2to1 m1(S, numA[1], numB[1], M[1]);

B\_1xMUX2to1 m0(S, numA[0], numB[0], M[0]);

endmodule

Simulation:



**Task 5: 2-Digit BCD Adder**

File: lab02part05.v, B\_2dBCDAdder.v, B\_1dBCDAdder, B\_4xFullAdderRipple.v, B\_8xBin2BCD.v

FPGA: X

Code:

MAIN

module lab02part05(

input [7:0]numA,

input [7:0]numB,

output [11:0]sumBCD

);

assign sumBCD[11:9] = 3'b000;

B\_2dBCDAdder get2DigitBCD(1'b0, numA[7:0], numB[7:0], sumBCD[8], sumBCD[7:0]);

endmodule

TESTBENCH

//VERIFICATION

module testBenchl2p5;

reg [7:0]a;

reg [7:0]b;

wire [3:0]h, t, u;

lab02part05 testl2p5(a, b, {h, t, u});

initial begin

a = {4'b0001, 4'b0001}; b = {4'b0001, 4'b0001}; #100;

a = {4'b1000, 4'b0001}; b = {4'b0001, 4'b0001}; #100;

a = {4'b1001, 4'b1001}; b = {4'b1001, 4'b1001}; #100;

a = {4'b0001, 4'b0000}; b = {4'b0001, 4'b0000}; #100;

end

endmodule

2-DIGIT BCD ADDER

module B\_2dBCDAdder(

input CIN,

input [7:0]numA,

input [7:0]numB,

output COUT,

output [7:0]sumBCD

);

wire carry0;

B\_1dBCDAdder add0( CIN, numA[3:0], numB[3:0], carry0, sumBCD[3:0]);

B\_1dBCDAdder add1(carry0, numA[7:4], numB[7:4], COUT, sumBCD[7:4]);

endmodule

1-DIGIT BCD ADDER

module B\_1dBCDAdder(

input CIN,

input [3:0] numA, numB,

output COUT,

output [3:0] sumBCD

);

wire [4:0]sumBin;

wire [9:0]bcdMid;

B\_4xFullAdderRipple add0(CIN, numA[3:0], numB[3:0], sumBin[4], sumBin[3:0]);

B\_8xBin2BCD convertSum({3'b000, sumBin[4:0]}, bcdMid[9:0]);

assign sumBCD[3:0] = bcdMid[3:0];

assign COUT = bcdMid[4];

endmodule

Logisim Simulation of Testbench:



**Task 7: Binary to BCD Converter**

File: lab02part07.v, B\_8xBin2BCD.v, B\_add3BCD.v

TruthTable: lab02part04\_table [#47-#65]

FPGA: X

Code:

MAIN

module lab02part07(

input [5:0]SW,

output [6:0]HEX1, HEX0

);

wire [9:0]numBCD; //10-bit [0~255]

B\_8xBin2BCD getBCD({2'b00, SW[5:0]}, numBCD[9:0]);

B\_7SegDec showTens(numBCD[7:4], HEX1[6:0]);

B\_7SegDec showUnts(numBCD[3:0], HEX0[6:0]);

Endmodule

TESTBENCH

module tBl2p7;

reg [5:0]sw;

wire [6:0]h1, h0;

lab02part07 testl2p7(sw, h1, h0);

initial begin

sw = 6'b001000; #100;

sw = 6'b010000; #100;

sw = 6'b100000; #100;

sw = 6'b111111; #100;

end

endmodule

8-BIT BINARY TO BCD

module B\_8xBin2BCD(

input [7:0]numBin,

output [9:0]numBCD

);

/\*

8-BIT BINARY TO BCD CONVERTER

-----------------------------

[+] Using Double Dabble

[+] Flow: https://johnloomis.org/ece314/notes/devices/binary\_to\_BCD/bcd04.png

\*/

wire [3:0]add3Out[6:0];

B\_add3BCD C1({1'b0, numBin[7:5]}, add3Out[6][3:0]);

B\_add3BCD C2({add3Out[6][2:0], numBin[4]}, add3Out[5][3:0]);

B\_add3BCD C3({add3Out[5][2:0], numBin[3]}, add3Out[4][3:0]);

B\_add3BCD C4({add3Out[4][2:0], numBin[2]}, add3Out[3][3:0]);

B\_add3BCD C5({add3Out[3][2:0], numBin[1]}, add3Out[2][3:0]);

B\_add3BCD C6({1'b0, add3Out[6][3], add3Out[5][3], add3Out[4][3]}, add3Out[1][3:0]);

B\_add3BCD C7({add3Out[1][2:0], add3Out[3][3]}, add3Out[0][3:0]);

assign numBCD[9] = add3Out[1][3];

assign numBCD[8:5] = add3Out[0][3:0];

assign numBCD[4:1] = add3Out[2][3:0];

assign numBCD[0] = numBin[0];

endmodule

module testBenchbin2bcd;

reg [7:0]binx;

wire [9:0]bcdy;

B\_8xBin2BCD testconv(binx, bcdy);

initial begin

binx=8'b00001111; #100;

binx=8'b11111111; #100;

end

endmodule

ADD-3 BLOCK

module B\_add3BCD(

input [3:0]numIN,

output [3:0]numOT

);

/\*

Adds 3 if input > 4.

Y3 | A + BC + BD

Y2 | AD + BC'D'

Y1 | B'C + CD + AD'

Y0 | AD' + A'B'D + BCD'

\*/

assign numOT[3] = ((numIN[3]) | (numIN[2] & numIN[1]) | (numIN[2] & numIN[0]));

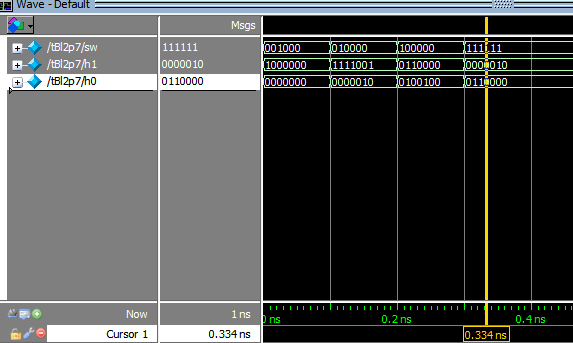
assign numOT[2] = ((numIN[3] & numIN[0]) | (numIN[2] & ~numIN[1] & ~numIN[0]));

assign numOT[1] = ((~numIN[2] & numIN[1]) | (numIN[1] & numIN[0]) | (numIN[3] & ~numIN[0]));

assign numOT[0] = ((numIN[3] & ~numIN[0]) | (~numIN[3] & ~numIN[2] & numIN[0]) | (numIN[2] & numIN[1] & ~numIN[0]));

endmodule

Simulation:

t