30 Days of RTL Coding

-By T Tushar Shenoy

Day 30

Problem Statement: Implementing 4 BIT Code converter in Structural Style.

Theory:

A 4-bit binary code converter is a circuit that can convert a 4-bit binary code represent from one form to another.

The excess-3 code (or XS3) is a non-weighted code used to express code used to express decimal numbers. It is a self-complementary binary coded decimal (BCD) code. 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.

BCD code or Binary coded Decimal codes. It is a numeric weighted binary codes, where every digit of a decimal number is expressed by a separate group of 4-bits. There are various BCD codes like 8421, 2421, 5211, etc. The BCD code is also known as the 8421 code.

Gray code is a non-weighted code. The successive gray code differs in one bit position only that means it is a unit distance code. It is also referred as cyclic code. It is not suitable for arithmetic operations. It is the most popular of the unit distance codes. It is also a reflective code.

Binary code in electronics refers to the representation of data and information using only two digits, 0 and 1. In digital electronics, binary codes are used to store, process and transmit information in computers and other digital devices. The binary system uses only two digits, 0 and 1, to represent all types of data, including numbers, letters, and special characters. The binary code is converted into electrical signals that can be processed and manipulated by electronic circuits. The binary system is a fundamental concept in digital electronics and forms the basis of modern computing.

Binary to Gray code conversion: The Binary to Gray code conversion involves converting a binary number into a gray code number. The gray code is a non-weighted code where only one bit changes between consecutive values. The conversion is performed by XORing the binary number with its right shift by 1 position.

BCD to Excess-3 conversion: BCD to Excess-3 conversion involves converting a BCD number into an Excess-3 number. In the Excess-3 code, the decimal equivalent of each code is 3 more than the BCD code. For example, if the BCD code is 1001, its Excess-3 code will be 1100.

Gray code to Binary conversion: The Gray code to Binary conversion involves converting a gray code number into its equivalent binary number. The conversion in general is performed by starting with the least significant bit (LSB) and XORing it with the previous bit. This process is repeated for each bit, starting from the LSB, to get the equivalent binary number.

Excess-3 to BCD conversion: The Excess-3 to BCD conversion involves converting an Excess-3 code into its equivalent BCD code. The conversion in general is performed by subtracting 3 from each digit in the Excess-3 code. For example, if the Excess-3 code is 0100, its BCD code will be 0001.

DESIGN AND IMPLEMENTATION

A 4-bit binary code converter can be designed and implemented using different techniques such as hardware circuitry or software algorithms.

The Circuit is designed using multiplexer (MUX) and de-multiplexer (DEMUX) along with Logic gates like OR, AND, NOT and XOR.

NOTE: The outputs are shorted through a Diode, in Verilog code I have considered different Outputs as shorting them was causing an Issue.

CIRCUIT DIAGRAM / BLOCK DIAGRAM

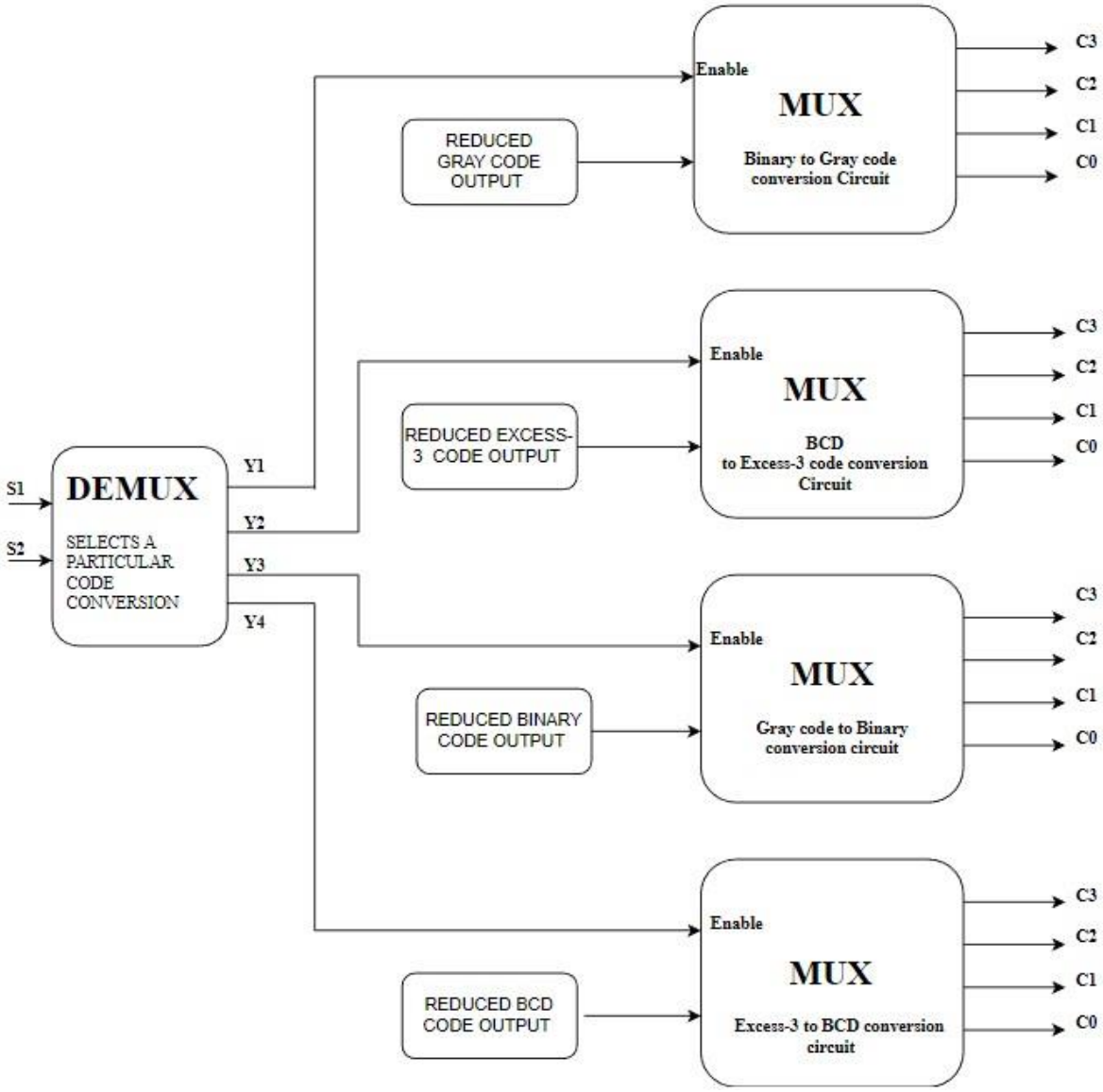


Fig: Block diagram of the Code Converter Circuit

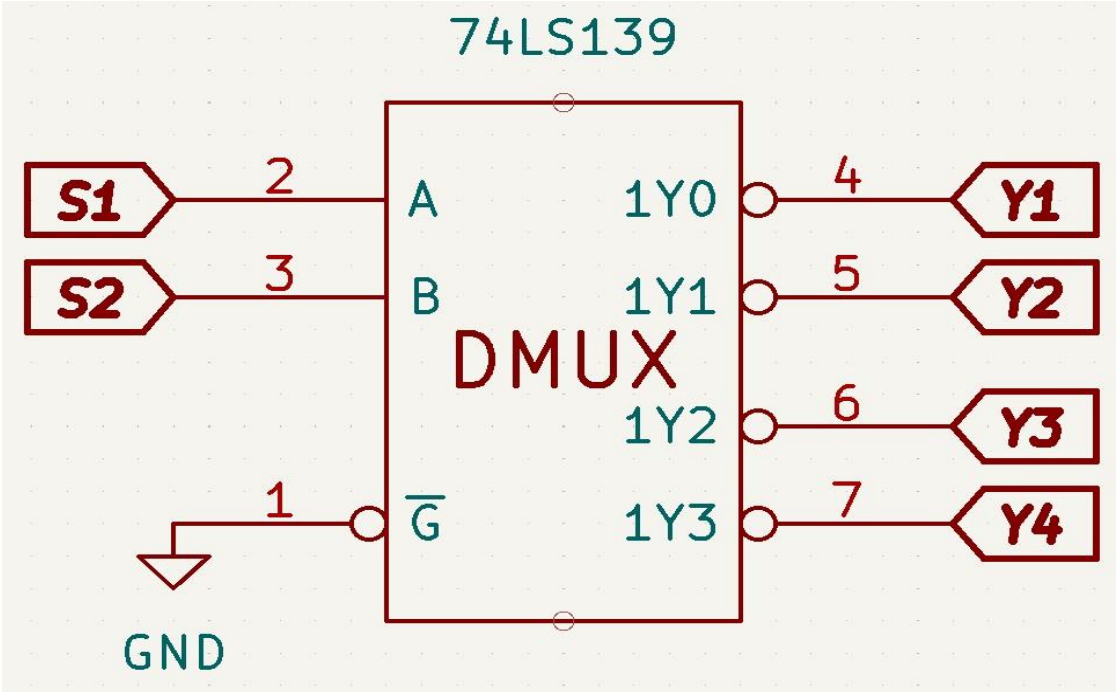


Fig: De-multiplexer Circuit

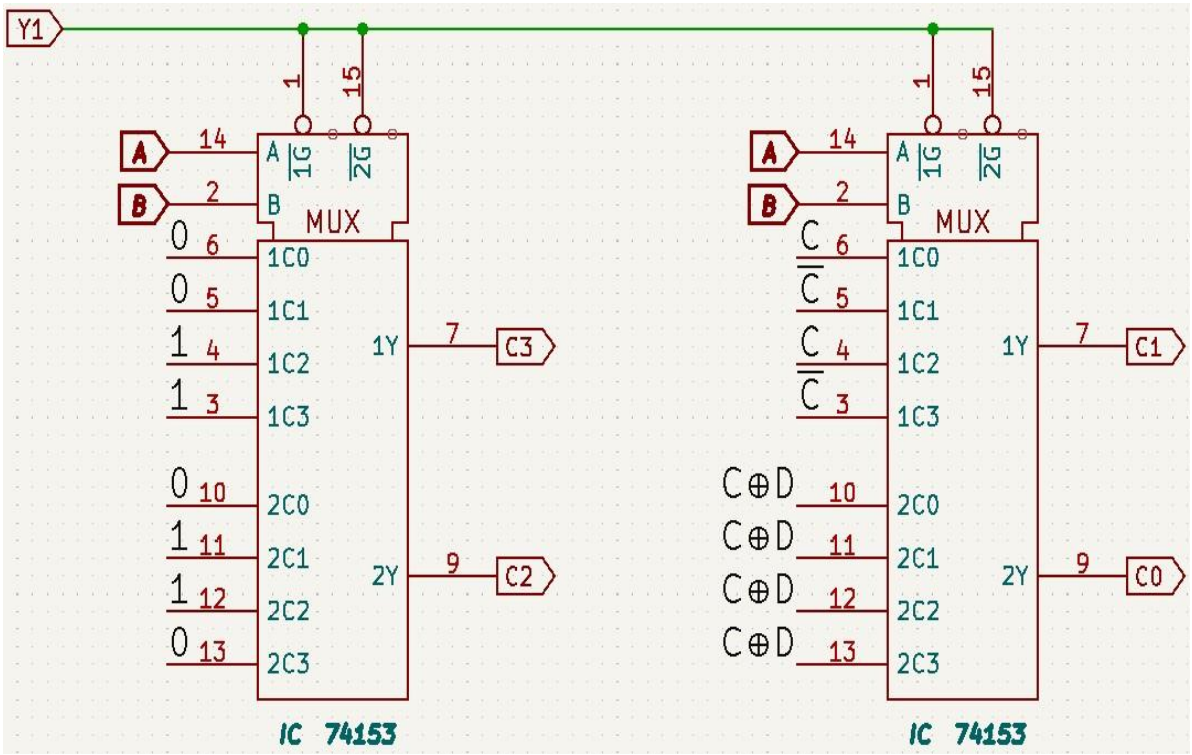


Fig: Binary to Gray code conversion Circuit

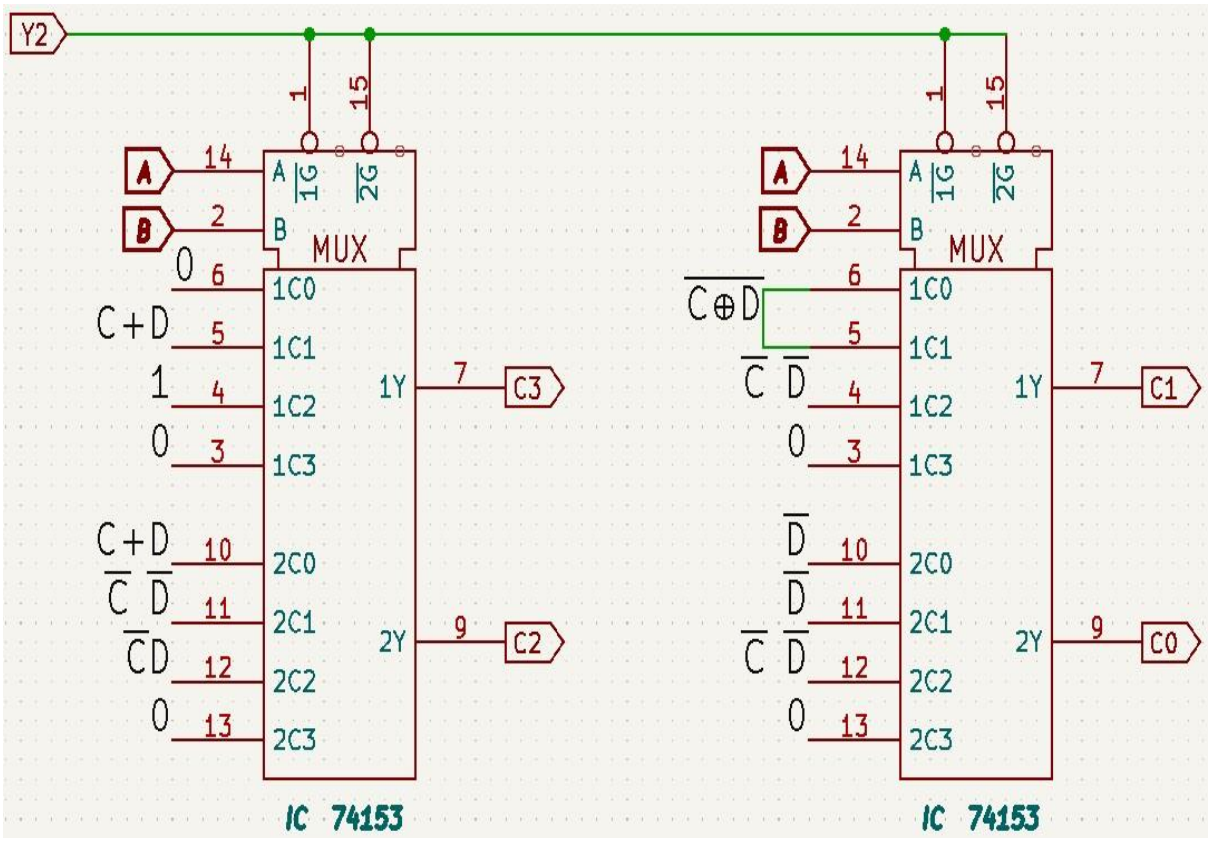


Fig: BCD to Excess-3 code conversion Circuit

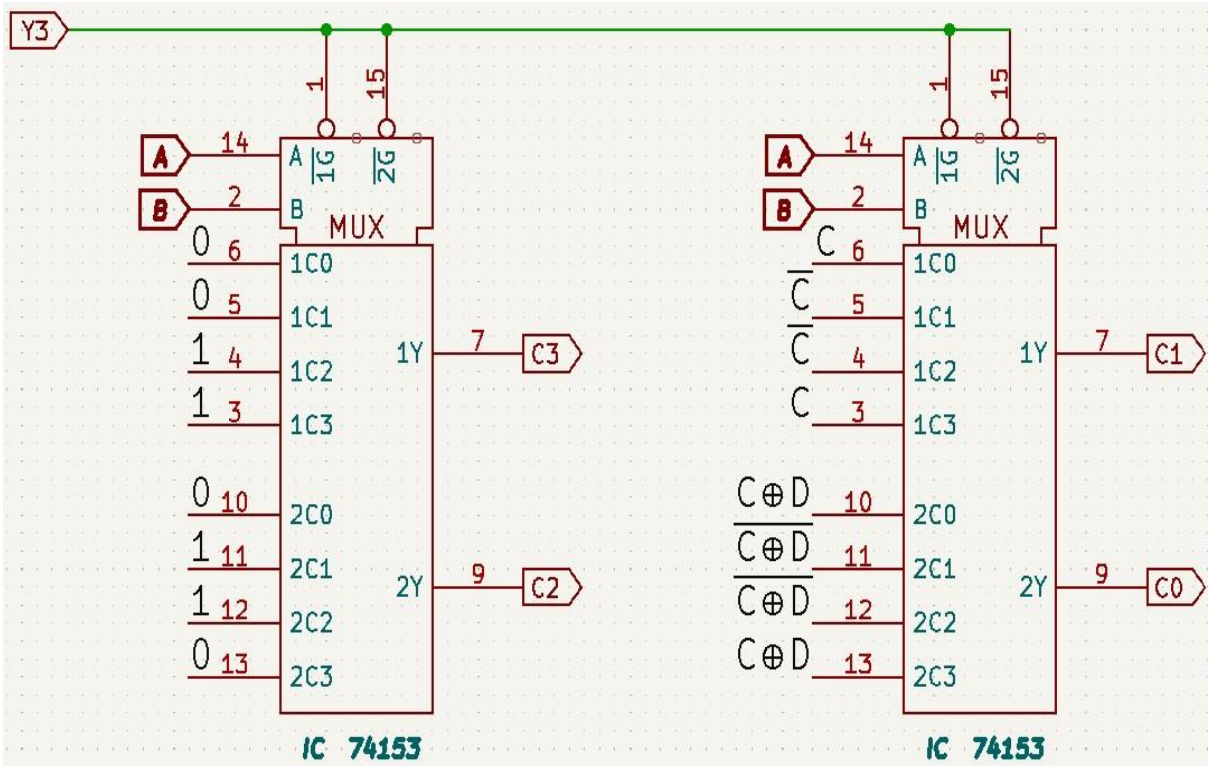


Fig:Gray code to Binary conversion Circuit

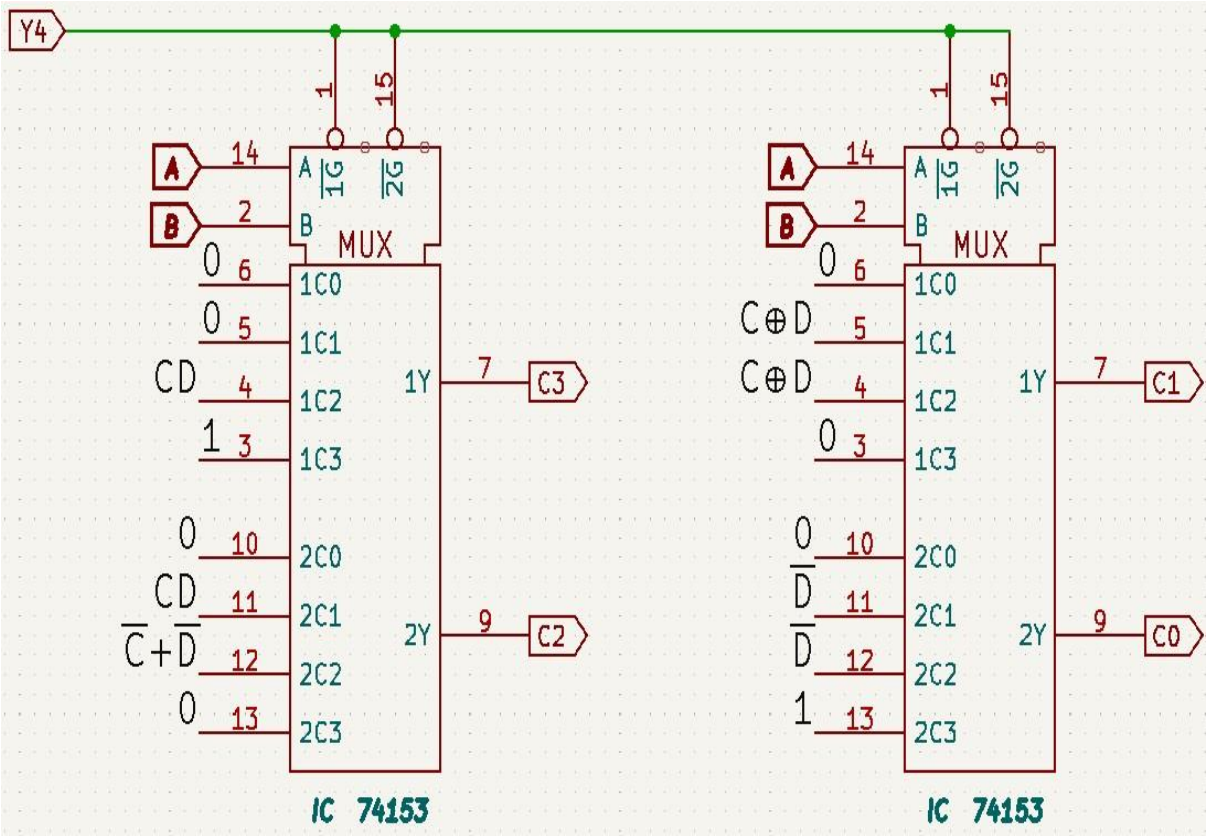


Fig: Excess-3 code to BCD conversion Circuit

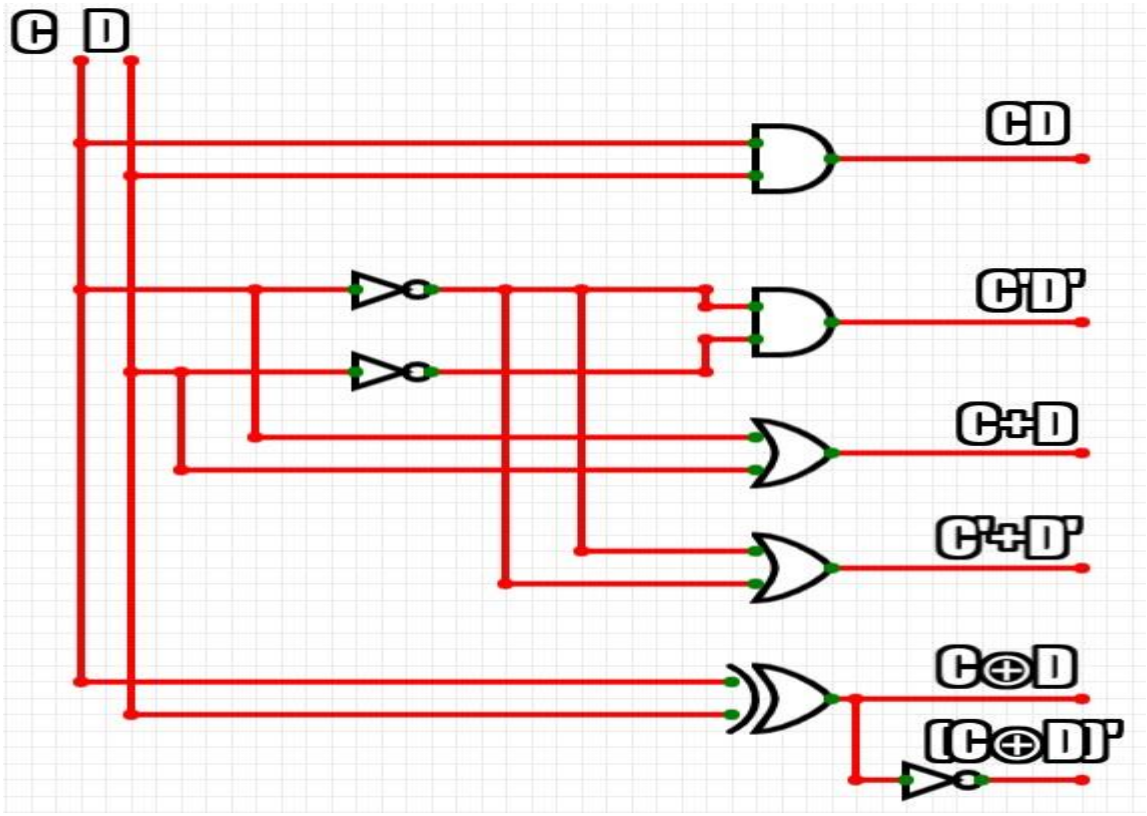


Fig: Logic Diagram using Circuitverse

ICs Specifications

1) IC 74153 (Dual 4-input multiplexer)

The multiplexer, shortened to "MUX" is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. IC 74153 is a multiplexer IC (Integrated Circuit) that allows the user to select one of four inputs and send it to the output. It has four data inputs (A0, A1, A2, A3), a selection input (S1, S0) to choose one of the four inputs, and an output (Y).

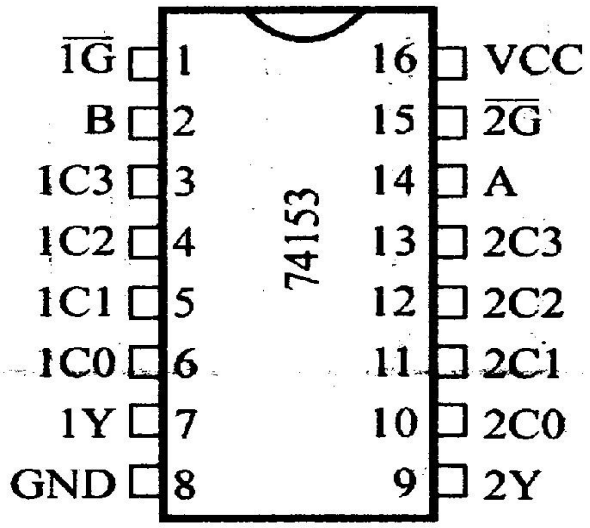


Fig: Pin Diagram of IC 74153

2) IC 74139 (Dual 2-to-4 line Decoder)

IC 74139 is a dual 2-to-4 line decoder/de-multiplexer integrated circuit. It has two inputs (A, B), two selection inputs (S1, S0), and four outputs (Y0, Y1, Y2, Y3). The device is used to convert two binary inputs into four outputs, where only one output is active at a time based on the selection inputs.

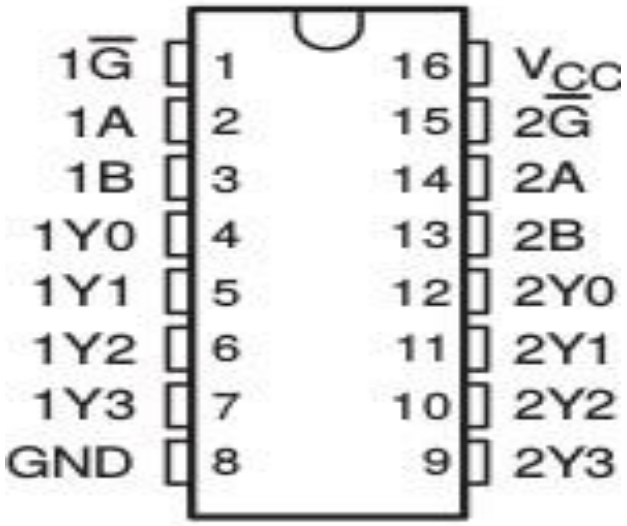


Fig: Pin Diagram of IC 74139

3) IC 7432 (Quad OR Gate)

IC 7432 is a logic gate IC which consist of four OR Gates. The OR gate performs logical OR operation. The OR gates come in form of DIP package ICs. Each gate has three terminal two inputs and one output. The OR gate outputs a high (1) signal if one or both of its inputs are high (1), and outputs a low (0) signal if both inputs are low (0). The inputs of the OR gate are labeled A and B, and the output is labeled Y.

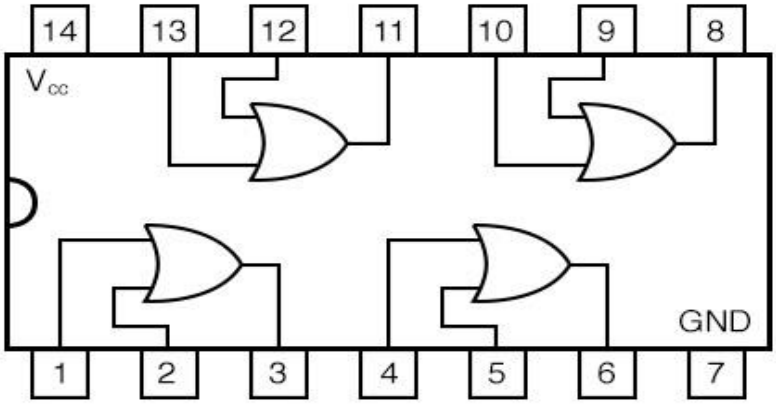


Fig: Pin Diagram of IC 7432

4) IC 7486 (Quad XOR Gate)

The IC 7486 is a quad 2-input XOR gate integrated circuit. It consists of four independent gates, each of which performs the logical exclusive OR (XOR) function. The XOR gate outputs a high (1) signal if one, but not both, of its inputs are high (1), and outputs a low (0) signal if both inputs are either high (1) or low (0). The inputs of the XOR gate are labeled A and B, and the output is labeled Y.

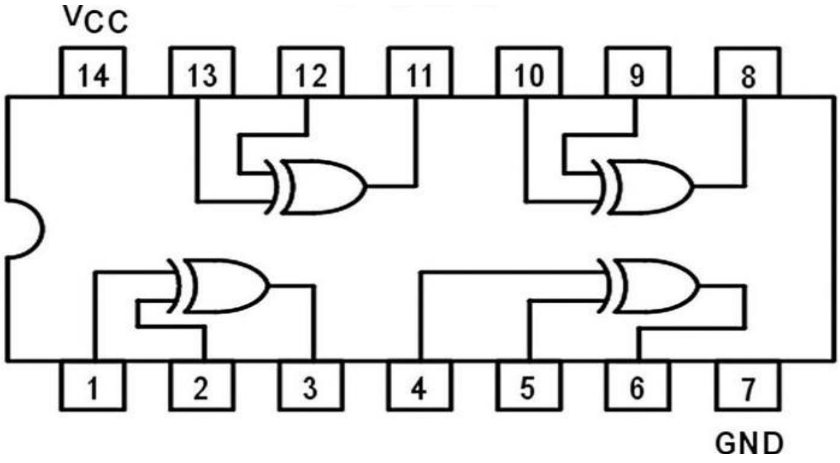


Fig: Pin Diagram of IC 7486

5) IC 7408 (Quad 2 input AND gate)

The IC 7408 is a quad 2-input AND gate integrated circuit. It consists of four independent gates, each of which performs the logical AND function. The AND gate outputs a high (1) signal if both of its inputs are high (1), and outputs a low (0) signal if one or both inputs are low (0). The inputs of the AND gate are labeled A and B, and the output is labeled Y.

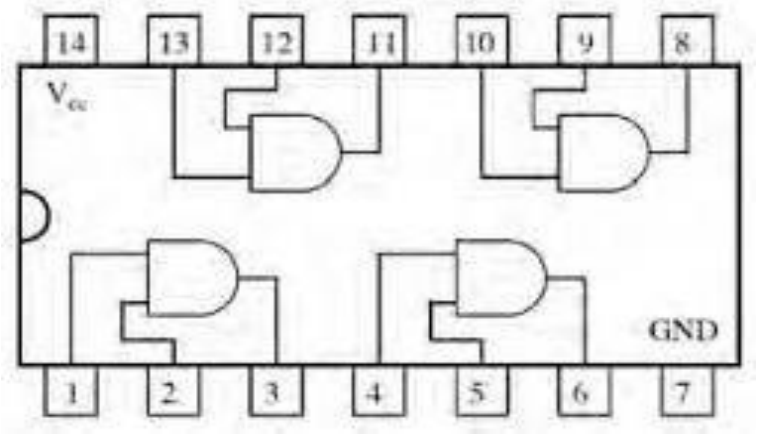


Fig: Pin Diagram of IC 7408

6) IC 7404 (Hex Inverter)

The IC 7404 is a hex inverter integrated circuit. It contains six independent inverter gates, each of which performs the logical NOT function. The NOT function, also known as inversion, outputs a low (0) signal if its input is high (1), and outputs a high (1) signal if its input is low (0). The input of the inverter gate is labeled A, and the output is labeled Y.

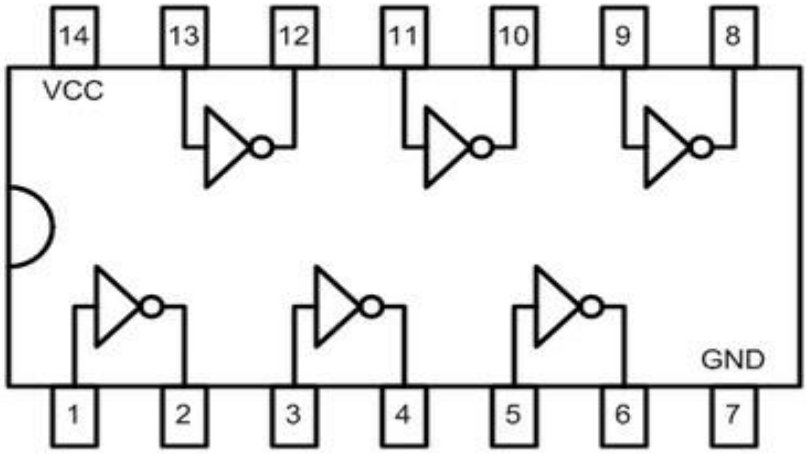


Fig: Pin Diagram of IC 7404

Truth Table

Table: Binary to Gray Code

BINARY INPUT				GRAY CODE OUTPUT				REDUCED GRAY CODE OUTPUT			
A	B	C	D	C_3	C_2	C_1	C_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0	0	0	C	$C \oplus D$
0	0	0	1	0	0	0	1				
0	0	1	0	0	0	1	1				
0	0	1	1	0	0	1	0				
0	1	0	0	0	1	1	0	0	1	$\overline{C}$	$C \oplus D$
0	1	0	1	0	1	1	1				
0	1	1	0	0	1	0	1				
0	1	1	1	0	1	0	0				
1	0	0	0	1	1	0	0	1	1	C	$C \oplus D$
1	0	0	1	1	1	0	1				
1	0	1	0	1	1	1	1				
1	0	1	1	1	1	1	0				
1	1	0	0	1	0	1	0	1	0	$\overline{C}$	$C \oplus D$
1	1	0	1	1	0	1	1				
1	1	1	0	1	0	0	1				
1	1	1	1	1	0	0	0				

Table: BCD to Excess-3

BCD INPUT				EXCESS-3 OUTPUT				REDUCED EXCESS-3 OUTPUT			
A	B	C	D	C_3	C_2	C_1	C_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	1	1	0	C+D	$\overline{C \oplus D}$	$\overline{D}$
0	0	0	1	0	1	0	0				
0	0	1	0	0	1	0	1				
0	0	1	1	0	1	1	0				
0	1	0	0	0	1	1	1	C+D	$\overline{C}\,\overline{D}$	$\overline{C \oplus D}$	$\overline{D}$
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	$\overline{C}$	$\overline{C}D$	$\overline{C}\,\overline{D}$	$\overline{C}\,\overline{D}$
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	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				

Table: Gray Code to Binary

GRAY CODE INPUT				BINARY OUTPUT				REDUCED BINARY OUTPUT			
A	B	C	D	C_3	C_2	C_1	C_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0	0	0	C	C⊕D
0	0	0	1	0	0	0	1				
0	0	1	0	0	0	1	1				
0	0	1	1	0	0	1	0				
0	1	0	0	0	1	1	1	0	1	$\overline{C}$	$\overline{C \oplus D}$
0	1	0	1	0	1	1	0				
0	1	1	0	0	1	0	0				
0	1	1	1	0	1	0	1				
1	0	0	0	1	1	1	1	1	1	$\overline{C}$	$\overline{C \oplus D}$
1	0	0	1	1	1	1	0				
1	0	1	0	1	1	0	0				
1	0	1	1	1	1	0	1				
1	1	0	0	1	0	0	0	1	0	C	C⊕D
1	1	0	1	1	0	0	1				
1	1	1	0	1	0	1	1				
1	1	1	1	1	0	1	0				

Table: Excess-3 to BCD

EXCESS-3 INPUT				BCD OUTPUT				REDUCED BCD OUTPUT			
A	B	C	D	C_3	C_2	C_1	C_0	C_3	C_2	C_1	C_0
0	0	0	0	X	X	X	X	0	0	0	0
0	0	0	1	X	X	X	X				
0	0	1	0	X	X	X	X				
0	0	1	1	0	0	0	0				
0	1	0	0	0	0	0	1	0	CD	C⊕D	$\overline{D}$
0	1	0	1	0	0	1	0				
0	1	1	0	0	0	1	1				
0	1	1	1	0	1	0	0				
1	0	0	0	0	1	0	1	CD	$\overline{C}$+$\overline{D}$	C⊕D	$\overline{D}$
1	0	0	1	0	1	1	0				
1	0	1	0	0	1	1	1				
1	0	1	1	1	0	0	0				
1	1	0	0	1	0	0	1	1	0	0	1
1	1	0	1	X	X	X	X				
1	1	1	0	X	X	X	X				
1	1	1	1	X	X	X	X				

Code Conversion Table

Demultiplexer Select Lines		Code Conversion	
S_2 (MSB)	S_1 (LSB)	FROM	TO
0	0	Binary Code	Gray Code
0	1	BCD	Excess-3 Code
1	0	Gray Code	Binary Code
1	1	Excess-3 Code	BCD

WORKING

The 4-bit binary code converter Circuit using Demultiplexer (DEMUX) and Multiplexer (MUX) works as follows:

- Initially the DEMUX is used to select a Particular Conversion, the output of the DEMUX is applied to the enable of the MUX. The DEMUX is used to select a particular binary conversion out of 4 different conversion. The selection lines of the DEMUX determine the Conversion which has to be performed.

- The inputs A, B, C, D is given once after selecting the conversion which is needed to be performed.

- The logic Gates Like OR, NOT, AND, XOR are Used to Perform Operations on inputs C and D and the outputs of the Logic Gates is then applied to the inputs of the MUX.

- The MUX is used to map the Logic gates input to output and the conversion to gray code or Excess-3 code or binary is performed. The selection lines of the MUX that is A and B determine the mapping of the binary code to the gray code or Excess-3 code.

- The outputs of the MUX represent the converted code in the desired format.

- The Converted output is displayed on the 7 Segment displays along with the LEDs.

Verilog Code:

```
//Verilog Code for NOT gate IC IC7404
module IC7404(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14);
input P1,P3,P5,P9,P11,P13;
input P7,P14; //Supply
output P2,P4,P6,P8,P10,P12;

assign P2=~P1;
assign P4=~P3;
assign P6=~P5;
assign P8=~P9;
assign P10=~P11;
assign P12=~P13;

endmodule

//Verilog Code for AND gate IC IC7408
module IC7408(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14);
input P1,P2,P4,P5,P9,P10,P12,P13;
input P7,P14;//SUPPLY
output P3,P6,P8,P11;

assign P3=P1&P2;
assign P6=P4&P5;
assign P8=P9&P10;
assign P11=P12&P13;
endmodule
```

```
//Verilog Code for OR gate IC IC7432
module IC7432(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14);
input P1,P2,P4,P5,P9,P10,P12,P13;
input P7,P14;//SUPPLY
output P3,P6,P8,P11;

assign P3=P1|P2;
assign P6=P4|P5;
assign P8=P9|P10;
assign P11=P12|P13;

endmodule


//Verilog Code for XOR gate IC
module IC7486(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14);
input P1,P2,P4,P5,P9,P10,P12,P13;
input P7,P14;//SUPPLY
output P3,P6,P8,P11;

assign P3=P1^P2;
assign P6=P4^P5;
assign P8=P9^P10;
assign P11=P12^P13;

endmodule
```

```
//Verilog Code for DEMUX IC IC74139
module
IC74139(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14,P15,P16);
input P1,P2,P3,P13,P14,P15;
input P8,P16;//SUPPLY
output reg P4,P5,P6,P7,P9,P10,P11,P12;

//P1 and P15 are the active low enable pins
//P2,P3 and P14,P13 are the two select lines
//P4,P5,P6,P7 are the DEMUX 1 outputs
//P9,P10,P11,P12 are the DEMUX 2 outputs

always@(*)
begin
   if(~P1)
   begin
      case({P3,P2})
       2'b00:{P4,P5,P6,P7}=4'b0111;
       2'b01:{P4,P5,P6,P7}=4'b1011;
       2'b10:{P4,P5,P6,P7}=4'b1101;
       2'b11:{P4,P5,P6,P7}=4'b1110;
      endcase
     end
     else if(~P15)
      begin
       case({P13,P14})
        2'b00:{P12,P11,P10,P9}=4'b0111;
        2'b01:{P12,P11,P10,P9}=4'b1011;
        2'b10:{P12,P11,P10,P9}=4'b1101;
```

```
        2'b11:{P12,P11,P10,P9}=4'b1110;
       endcase
    end
end
endmodule



//Verilog Code for MUX IC IC74153
module
IC74153(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14,P15,P16);
input P1,P2,P3,P4,P5,P6,P10,P11,P12,P13,P14,P15;
input P8,P16;//SUPPLY
output reg P7,P9;

//P1 and P15 are the active low enable pins
//P2 and P14 are the two select lines
//P6,P5,P4,P3 are the MUX 1 inputs
//P10,P11,P12,P13 are the MUX 2 inputs

always@(*)
begin
  if(~(P1|P15))
  begin
       if(P14==1'b0&P2==1'b0)
       begin
            P7=P6;
            P9=P10;
       end
```

```
        if(P14==1'b0&P2==1'b1)
        begin
            P7=P5;
            P9=P11;
        end

        if(P14==1'b1&P2==1'b0)
        begin
            P7=P4;
            P9=P12;
        end

        if(P14==1'b1&P2==1'b1)
        begin
            P7=P3;
            P9=P13;
        end
    end
end
endmodule
```

```
//Verilog Code for 4 Bit Code Converter in Structural Style

//Here the outputs are considered as different because there was an issue when outputs of all the Muxs were shorted.

//In Practically Built Circuits the Outputs are shorted using a Diode so the Issue has been resolved there.

//o0 output of Binary to Gray Code Converter.

//o1 BCD to Excess-3 Code Converter.

//o2 Gray Code to Binary Converter.

//o3 Excess-3 to BCD Converter.


module Code_Converter_4Bit(A,B,C,D,S2,S1,o0,o1,o2,o3);

input A,B,C,D;

input S2,S1;

output [3:0]o0,o1,o2,o3;

reg GND=1'b0,VCC=1'b1;

wire [3:0]de_mux;

wire nC,nD,andCD,nCandD,nCandnD,orCD,nCornD,xorCD,nxorCD;


IC74139
demux(.P1(GND),.P2(S1),.P3(S2),.P4(de_mux[0]),.P5(de_mux[1]),.P6(de_mux[2]),.P7(de_mux[3]),.P8(GND),.P16(VCC));//Unused Pins P9,P10,P11,P12,P13,P14,P15


//Binary to Gray Code Converter

IC74153
mux1(.P1(de_mux[0]),.P2(B),.P3(1'b1),.P4(1'b1),.P5(1'b0),.P6(1'b0),.P7(o0[3]),.P8(GND),.P9(o0[2]),.P10(1'b0),.P11(1'b1),.P12(1'b1),.P13(1'b0),.P14(A),.P15(de_mux[0]),.P16(VCC));

IC74153
mux2(.P1(de_mux[0]),.P2(B),.P3(nC),.P4(C),.P5(nC),.P6(C),.P7(o0[1]),.P8(GND),.P9(o0[0]),.P10(xorCD),.P11(xorCD),.P12(xorCD),.P13(xorCD),.P14(A),.P15(de_mux[0]),.P16(VCC));
```

//BCD to Excess-3 Code Converter

IC74153
mux3(.P1(de_mux[1]),.P2(B),.P3(1'b0),.P4(1'b1),.P5(orCD),.P6(1'b0),.P7(o1[3]),.P8(GND),.P9(o1[2]),.P10(orCD),.P11(nCandnD),.P12(nCandD),.P13(1'b0),.P14(A),.P15(de_mux[1]),.P16(VCC));

IC74153
mux4(.P1(de_mux[1]),.P2(B),.P3(1'b0),.P4(nCandnD),.P5(nxorCD),.P6(nxorCD),.P7(o1[1]),.P8(GND),.P9(o1[0]),.P10(nD),.P11(nD),.P12(nCandnD),.P13(1'b0),.P14(A),.P15(de_mux[1]),.P16(VCC));

//Gray Code to Binary Converter

IC74153
mux5(.P1(de_mux[2]),.P2(B),.P3(1'b1),.P4(1'b1),.P5(1'b0),.P6(1'b0),.P7(o2[3]),.P8(GND),.P9(o2[2]),.P10(1'b0),.P11(1'b1),.P12(1'b1),.P13(1'b0),.P14(A),.P15(de_mux[2]),.P16(VCC));

IC74153
mux6(.P1(de_mux[2]),.P2(B),.P3(C),.P4(nC),.P5(nC),.P6(C),.P7(o2[1]),.P8(GND),.P9(o2[0]),.P10(xorCD),.P11(nxorCD),.P12(nxorCD),.P13(xorCD),.P14(A),.P15(de_mux[2]),.P16(VCC));

//Excess-3 to BCD Converter

IC74153
mux7(.P1(de_mux[3]),.P2(B),.P3(1'b1),.P4(andCD),.P5(1'b0),.P6(1'b0),.P7(o3[3]),.P8(GND),.P9(o3[2]),.P10(1'b0),.P11(andCD),.P12(nCornD),.P13(1'b0),.P14(A),.P15(de_mux[3]),.P16(VCC));

IC74153
mux8(.P1(de_mux[3]),.P2(B),.P3(1'b0),.P4(xorCD),.P5(xorCD),.P6(1'b0),.P7(o3[1]),.P8(GND),.P9(o3[0]),.P10(1'b0),.P11(nD),.P12(nD),.P13(1'b1),.P14(A),.P15(de_mux[3]),.P16(VCC));

IC7404
notgate(.P1(C),.P2(nC),.P3(D),.P4(nD),.P5(xorCD),.P6(nxorCD),.P7(GND),.P14(VCC));//Unused Pins P8,P9,P10,P11,P12,P13

IC7432
orgate(.P1(C),.P2(D),.P3(orCD),.P4(nC),.P5(nD),.P6(nCornD),.P7(GND),.P14(VCC));//Unused Pins P8,P9,P10,P11,P12,P13

IC7486 xorgate(.P1(C),.P2(D),.P3(xorCD),.P7(GND),.P14(VCC));//Unused Pins P4,P5,P6,P8,P9,P10,P11,P12,P13

IC7408
andgate(.P1(nC),.P2(nD),.P3(nCandnD),.P4(nC),.P5(D),.P6(nCandD),.P7(GND),.P8(andCD),.P9(C),.P10(D),.P14(VCC));//Unused Pins P11,P12,P13

endmodule

Testbench Code:

```
//Testbench Code for 4 Bit Binary Code Converter.
module Code_Converter_4Bit_tb();

reg A,B,C,D;
reg S2,S1;
wire [3:0]o0,o1,o2,o3;
Code_Converter_4Bit
dut(.A(A),.B(B),.C(C),.D(D),.S2(S2),.S1(S1),.o0(o0),.o1(o1),.o2(o2),.o3(o3)
);

initial begin
   {A,B,C,D}=4'b1111;     //Initializing the inputs to 1111

// Performs Binary to Gray code
    {S2,S1}=2'b00;
      stimulus();
// Performs BCD to Excess-3 code
    {S2,S1}=2'b01;
      stimulus();
// Performs Gray Code to Binary
    {S2,S1}=2'b10;
      stimulus();
// Performs Excess-3 code to Binary
    {S2,S1}=2'b11;
     stimulus();
     $finish;
end

task stimulus;
```

```
    integer i;
        for(i=0;i<16;i=i+1)
            begin
                {A,B,C,D}={A,B,C,D}+1'b1;
                #5;
            end
endtask
endmodule
```

Schematic:

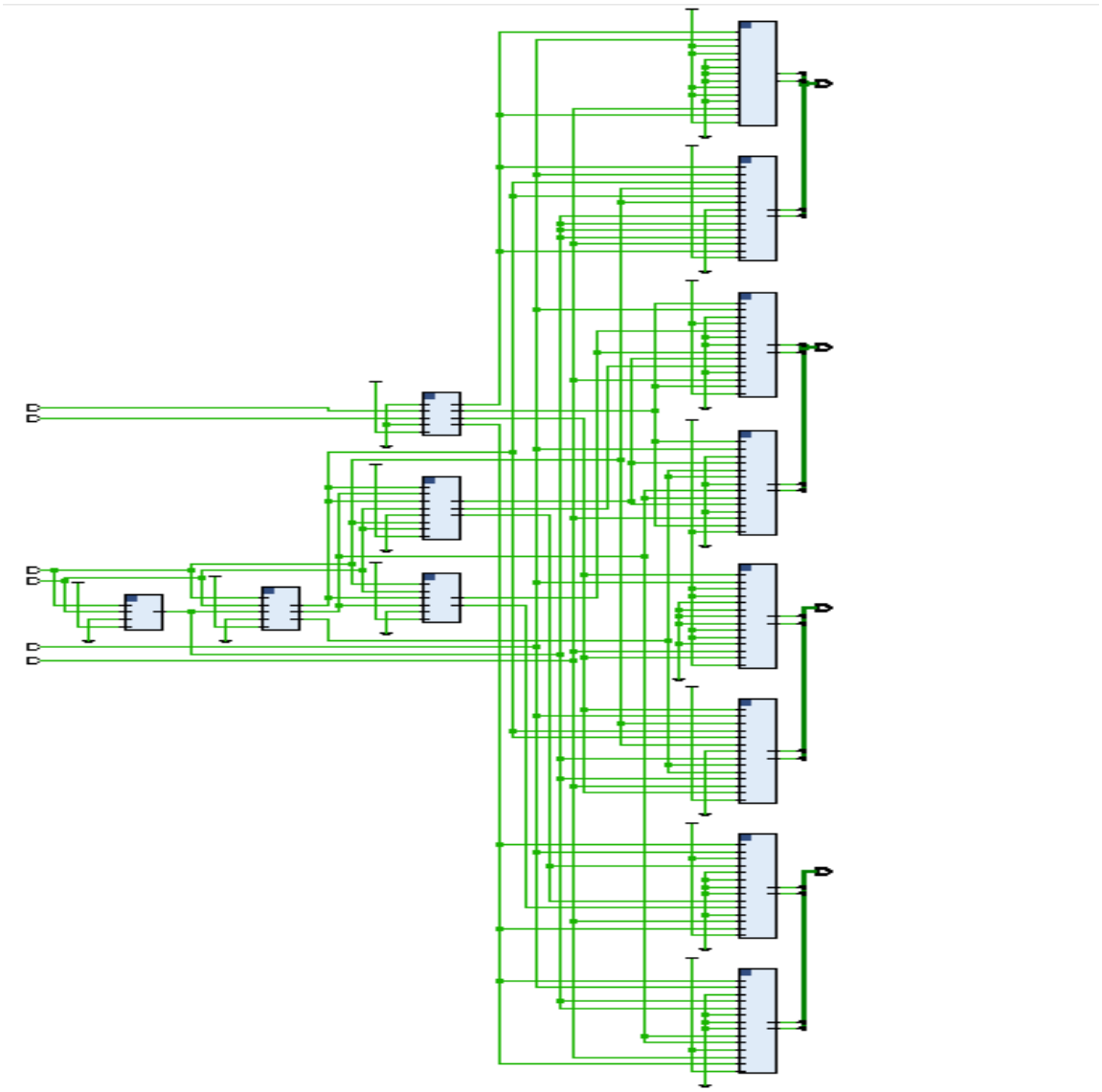
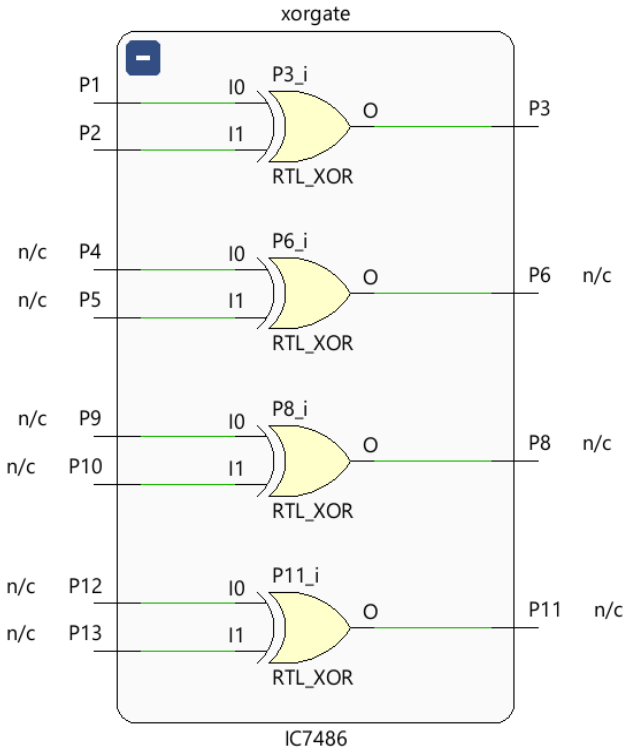
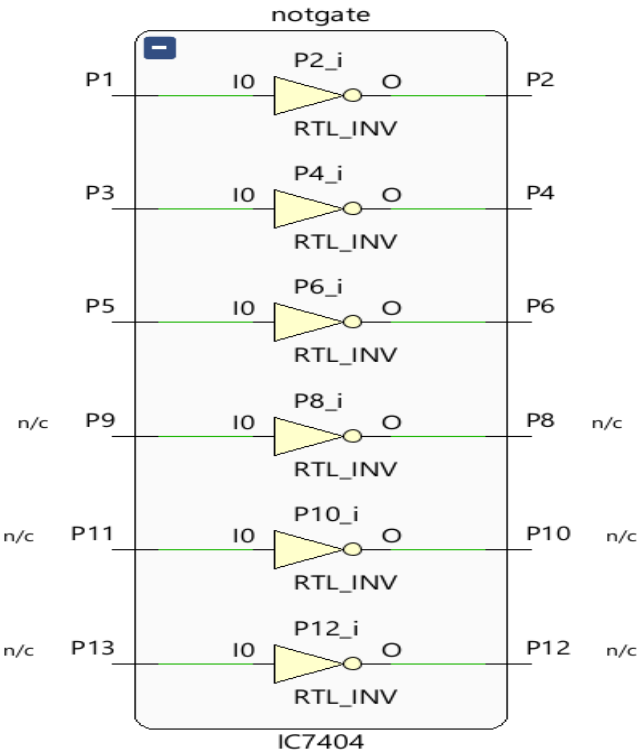
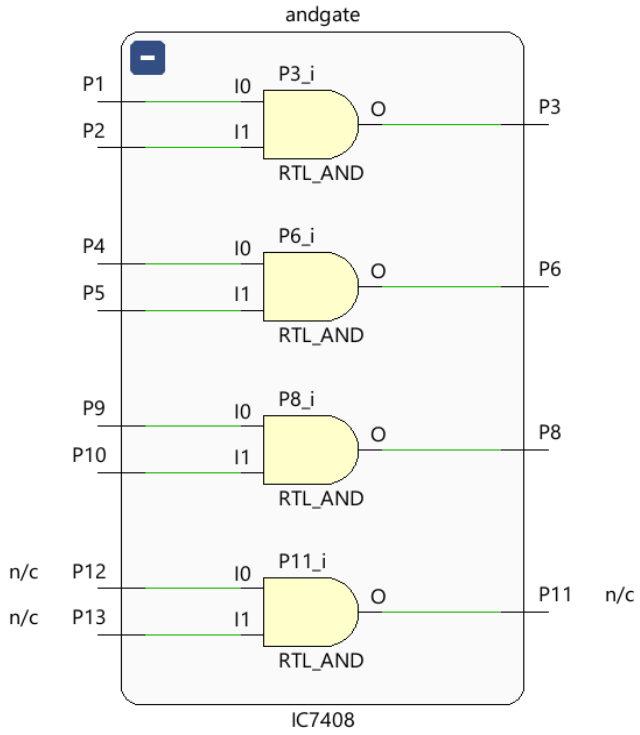


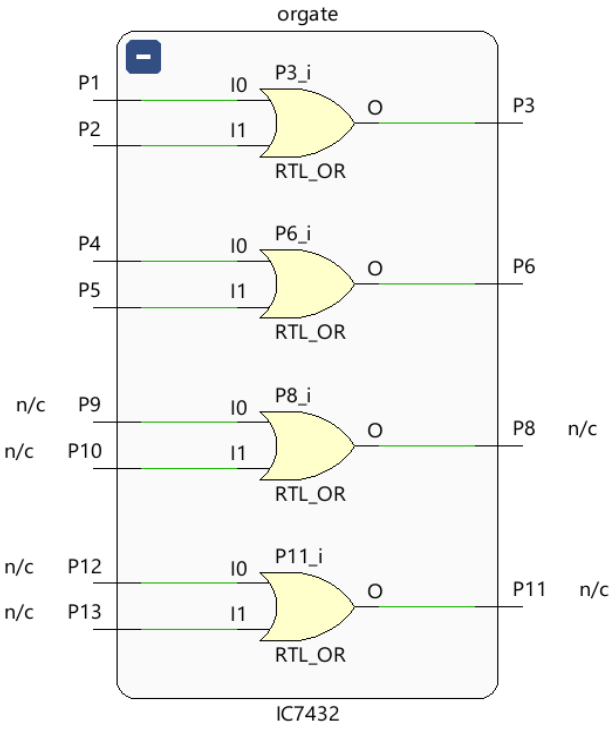
FIG:4 Bit Binary Code Converter

FIG: ICs


xorgate
P1
P2
I0
I1
P3_i
O
P3
RTL_XOR
n/c
P4
n/c
P5
P6_i
P6
n/c
n/c
P9
n/c
P10
P8_i
P8
n/c
n/c
P12
n/c
P13
P11_i
P11
n/c
IC7486


notgate
P1
P2_i
P2
RTL_INV
P3
P4_i
P4
n/c
P5
P6_i
P6
n/c
n/c
P9
P8_i
P8
n/c
n/c
P11
P10_i
P10
n/c
n/c
P13
P12_i
P12
n/c
IC7404


andgate
P1
P2
P3_i
P3
RTL_AND
P4
P5
P6_i
P6
P9
P10
P8_i
P8
n/c
n/c
P12
n/c
P13
P11_i
P11
n/c
IC7408


orgate
P1
P2
P3_i
P3
RTL_OR
P4
P5
P6_i
P6
n/c
n/c
P9
P10
P8_i
P8
n/c
n/c
P12
n/c
P13
P11_i
P11
n/c
IC7432

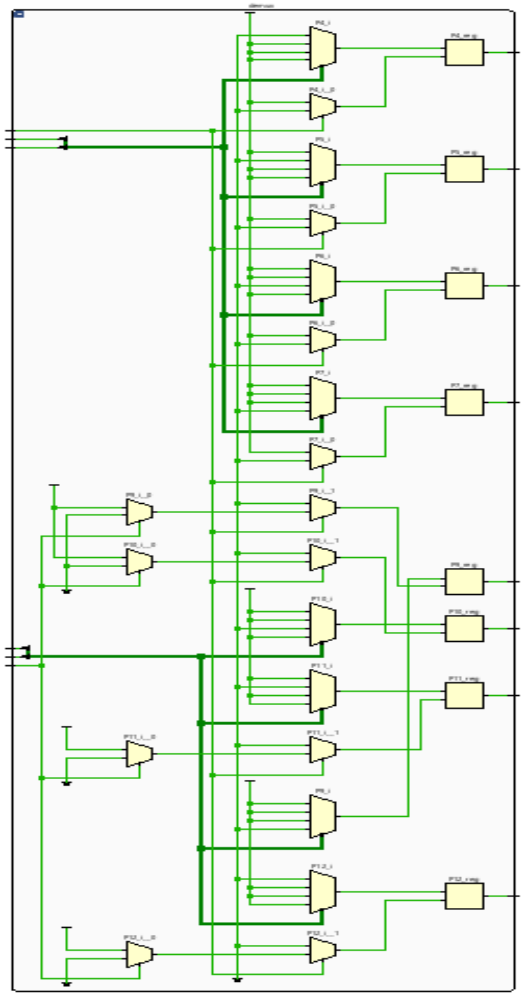


FIG: IC 74139

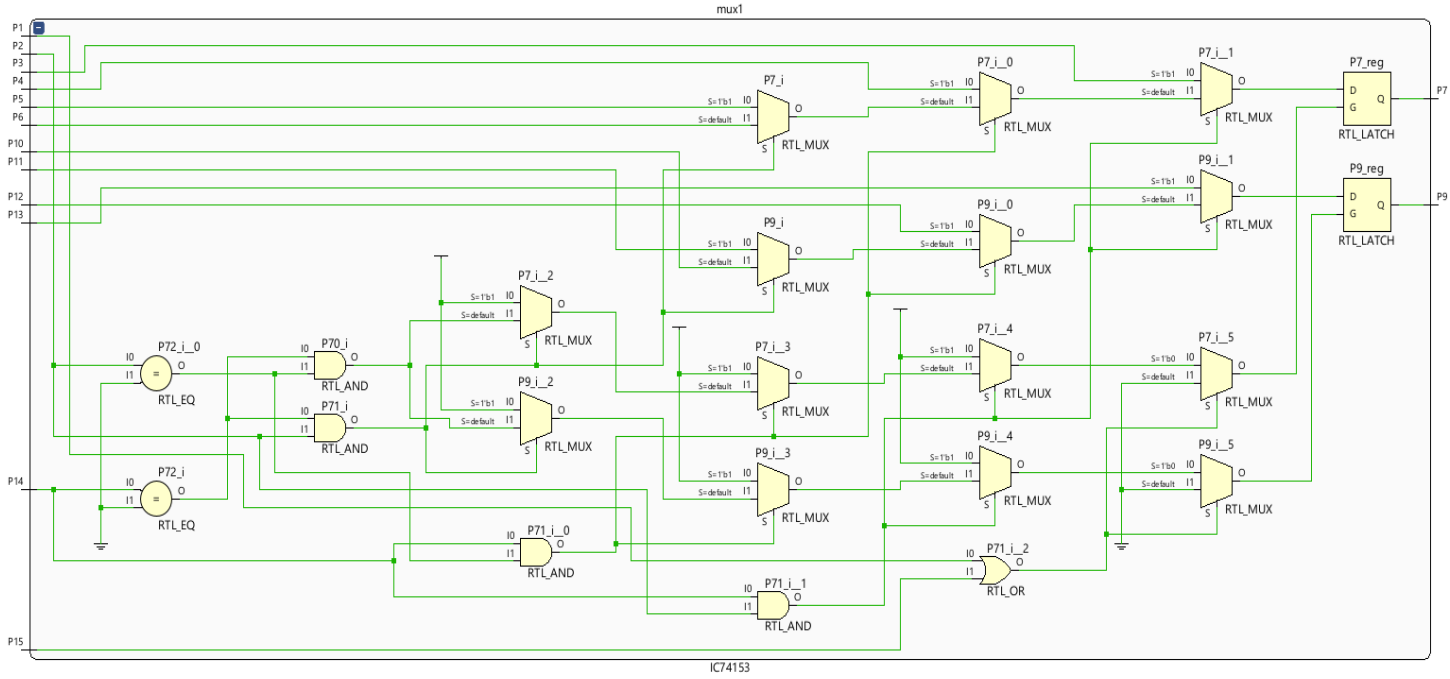
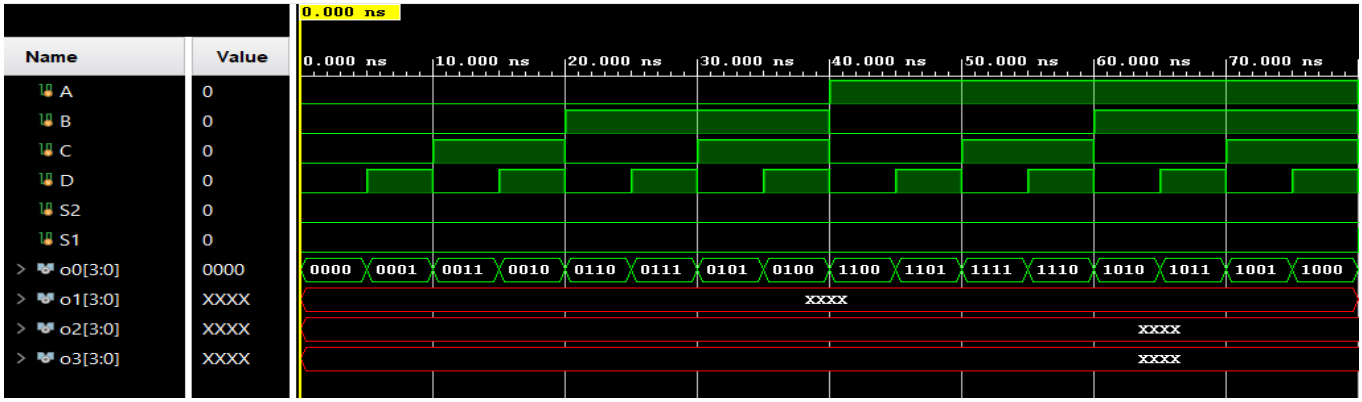


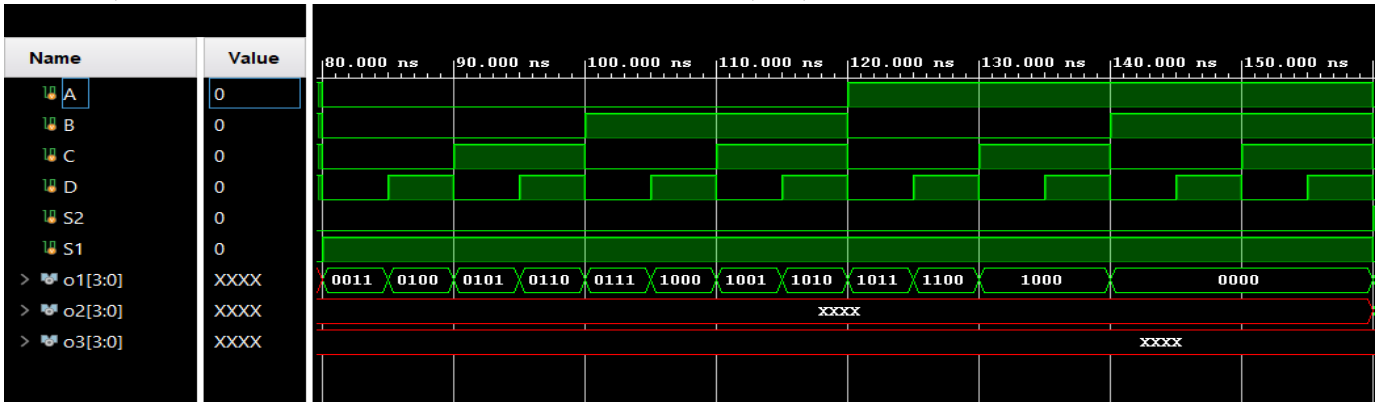
FIG: IC 74153

Simulation Output:

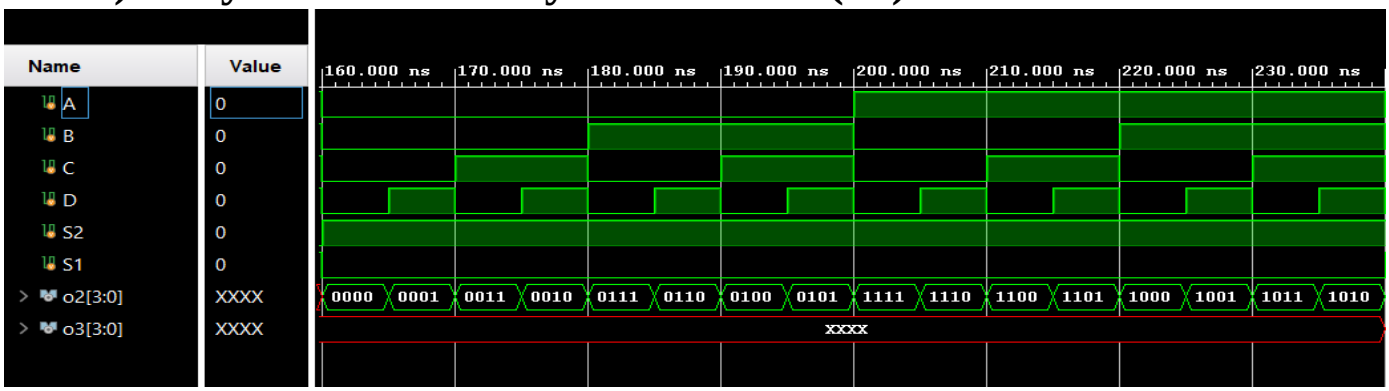
1) Binary to Gray Code Conversion (o0)



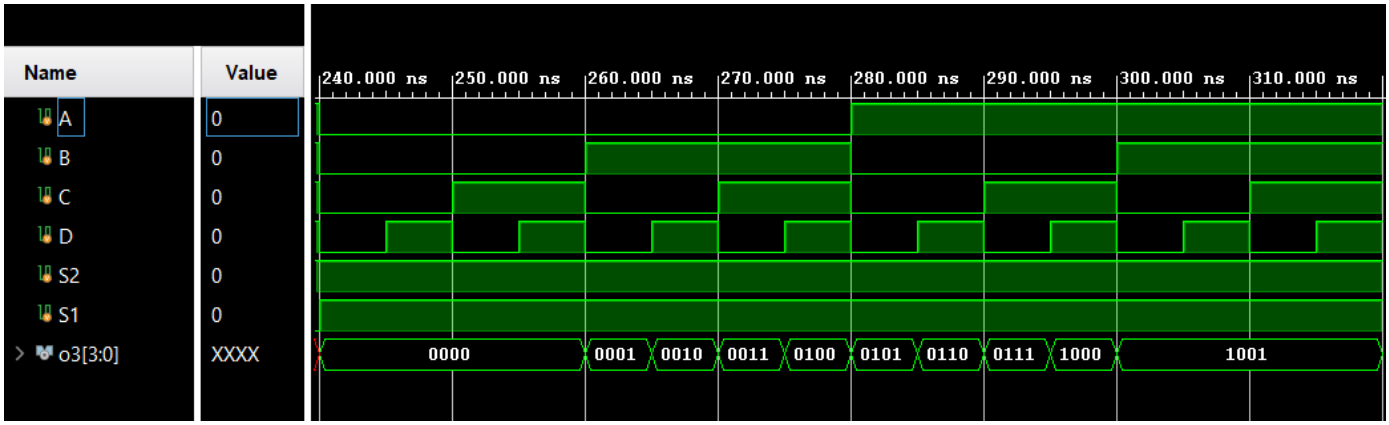
2) BCD to Excess-3 Conversion (o1)



3) Gray Code to Binary Conversion (o2)



4) Excess-3 to BCD Conversion (o3)



GitHub Repository URL:

https://github.com/tusharshenoy/RTL-Day-30-4-BIT-Binary-Code-Converter