DIGITAL DESIGN

LAB7 COMBINATORIAL CIRCUIT ENCODER, DECODER, MUX, DMUX

2024 FALL TERM @ CSE . SUSETCH

WANGW6@SUSTECH.EDU.CN

LAB7

- Combinational circuit
  - Encoder
  - Decoder
  - Multiplexer
    - The logical expression of Multiplexer
    - Using Multiplexer to implement a combinational circuit
  - Demultiplexer
- Practices

ENCODER(PRIORITY ENCODER)

An encoder is a device that converts information from one format or code to another, for the purposes of **standardization**, **speed** or **compression**.

Priority encoder

```
//4-2 PRI-ENCODER
//INPUT LOW EFFECT
module encoder_pri(
input I0,I1,I2,I3,
output reg[1:0] Y
    );
    always@* begin
        casex( {I3,I2,I1,I0} )
            4'bxxx0: Y=2'b00;
            4'bxx01: Y=2'b01;
            4'bx011: Y=2'b10;
            4'b0111: Y=2'b11;
        endcase
    end
endmodule
```

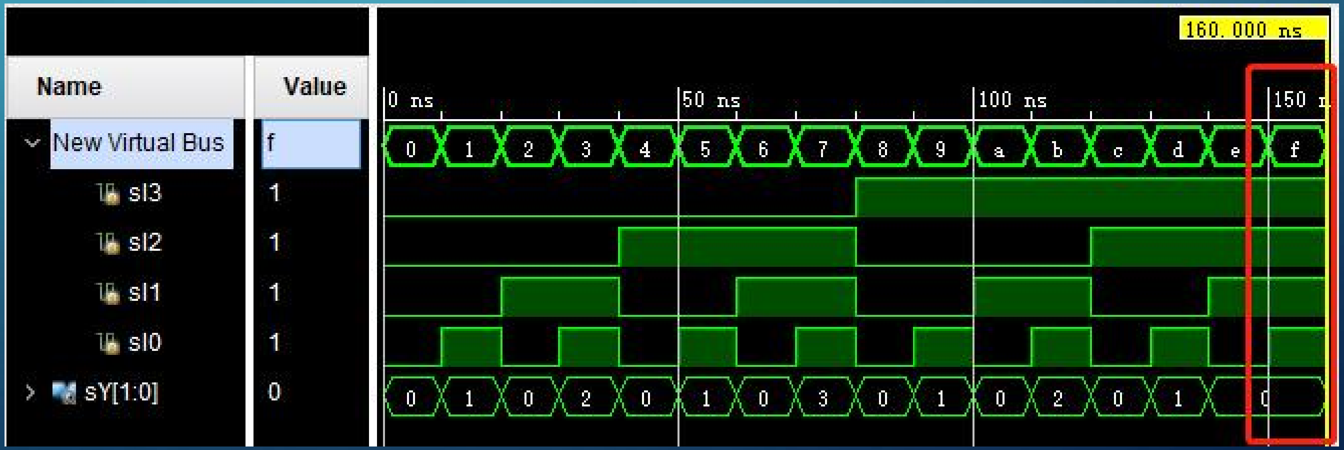
LSB's priority is the highest

The input is low level effective

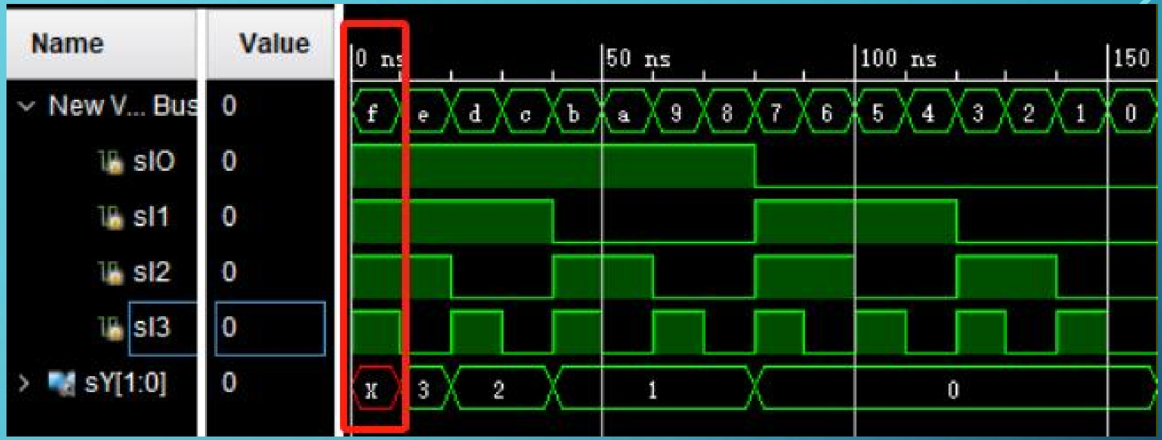
input				output	
I3	I2	I1	I0	Y1	Y0
X	X	X	0	0	0
X	X	0	1	0	1
X	0	1	1	1	0
0	1	1	1	1	1

truth table of 4-2 pri-encoder

```
module encoder_pri_tb( );
reg sI0, sI1,sI2,sI3;
wire [1:0] sY;
encoder_pri u(sI0, sI1,sI2,sI3,sY);
initial begin
    {sI0,sI1,sI2,sI3} = 4'b0000;
    repeat(15)
        #10 {sI0,sI1,sI2,sI3} = {sI0,sI1,sI2,sI3}+1;
    #10 $finish;
end
endmodule
```



NOTE: THE IMPORTANCE OF “DEFAULT”




```
module encoder_pri_tb( );
reg sI0,  sI1, sI2, sI3;
wire [1:0] sY;
encoder_pri u(sI0,  sI1, sI2, sI3, sY);
initial begin
    {sI0, sI1, sI2, sI3} = 4'b0000;
    repeat(15)
        #10 {sI0, sI1, sI2, sI3} = {sI0, sI1, sI2, sI3}+1;
    #10 $finish;
end
endmodule
```

```
//4-2 PRI-ENCODER
//INPUT LOW EFFECT
module encoder_pri(
input I0, I1, I2, I3,
output reg[1:0] Y
    );
    always@* begin
        casex( {I3, I2, I1, I0} )
            4'bxxx0: Y=2'b00;
            4'bxx01: Y=2'b01;
            4'bx011: Y=2'b10;
            4'b0111: Y=2'b11;
        endcase
    end
endmodule
```

```
module encoder_pri_tb( );
reg sI0,  sI1, sI2, sI3;
wire [1:0] sY;
encoder_pri u(sI0,  sI1, sI2, sI3, sY);
initial begin
    {sI0, sI1, sI2, sI3} = 4'b1111;
    repeat(15)
        #10 {sI0, sI1, sI2, sI3} = {sI0, sI1, sI2, sI3}-1;
    #10 $finish;
end
endmodule
```

Same input “ f ”, different output: x and 0 ?

TIPS: Lack of “default” in casex block

DECODER

- In digital electronics, a **binary Decoder** is a combinational logic circuit that converts binary information from the n coded inputs to a maximum of 2^n unique outputs. They are used in a wide variety of applications, including data du-multiplexing, seven segment displays, and memory address decoding.
- There are several types of binary decoders, but in all cases a decoder is an electronic circuit with multiple input and multiple output signals, which converts every unique combination of input states to a specific combination of output states.
- In addition to integer data inputs, some decoders also have one or more "enable" inputs. When the enable input is negated (disabled), all decoder outputs are forced to their inactive states.

DECODER（2-4 DECODER）

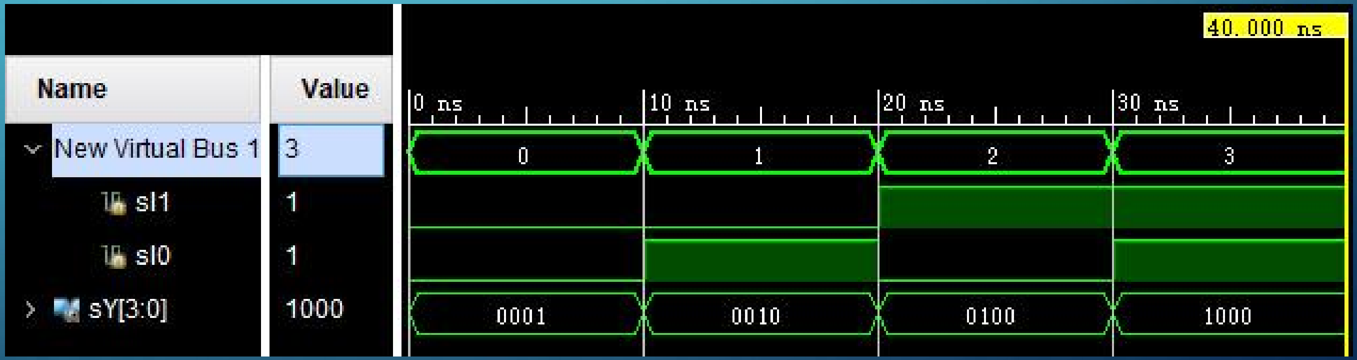
```
//2-4decoder
module decoder(
    input I0,
    input I1,
    output reg [3:0] Y
    );
    always @*
    begin
        case ({I1,I0})
            2'b00: Y=4'b0001;
            2'b01: Y=4'b0010;
            2'b10: Y=4'b0100;
            2'b11: Y=4'b1000;
        endcase
    end
endmodule
```

input		output			
I1	I0	Y3	Y2	Y1	Y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

truth table 2-4 decoder

```
module decoder_tb();
    reg sI0, sI1;
    wire [3:0] sY;

    decoder u(sI0, sI1, sY);
    initial
    begin
        {sI1, sI0} = 0;
        repeat(3) #10 {sI1, sI0} = {sI1, sI0} + 1;
        #10 $finish;
    end
endmodule
```



ONE HOT CODING

- **One hot coding**, also known as one bit effective coding
  - use **n-bit** status register to code **n** states.
  - Each state has its own register bits, and at any time, only one of them is valid.

```
Y=4'b0001;
Y=4'b0010;
Y=4'b0100;
Y=4'b1000;
```

In the previous practice “RPS game”,
using one hot coding as:

rock : 3’b001
paper : 3’b010
scissors : 3’b100

DECODER（2-4 DECODER WITH ENABLE CONTROL)

In addition to integer data inputs, some decoders also have one or more "enable" inputs. When the enable input is negated (disabled), all decoder outputs are forced to their inactive states.

EN low level effective	{ I1, I0 }	Y
1	XX	4'B0000
0	2'B000	4'B0001
0	2'B001	4'B0010
0	2'B010	4'B0100
0	2'B011	4'B1000

```
module decoder_en_2_4(
input I0,I1,
input EN,
output reg [3:0] Y
    );
    always @*  begin
    if(~EN) //low level effecitve
    case( {I1,I0} )
        2'B00: Y=4'B0001;
        2'B01: Y=4'B0010;
        2'B10: Y=4'B0100;
        2'B11: Y=4'B1000;
    endcase
    else
        Y=4'B0000;
    end
endmodule
```

Enable input port

DECODER（3-8 DECODER 1）

- How to implement an 3-8 decoder by using two 2-4 decoders?

Enable input port

EN low level effective	{ I1, I0 }	Y
1	XX	4’B0000
0	2’B000	4’B0001
0	2’B001	4’B0010
0	2’B010	4’B0100
0	2’B011	4’B1000

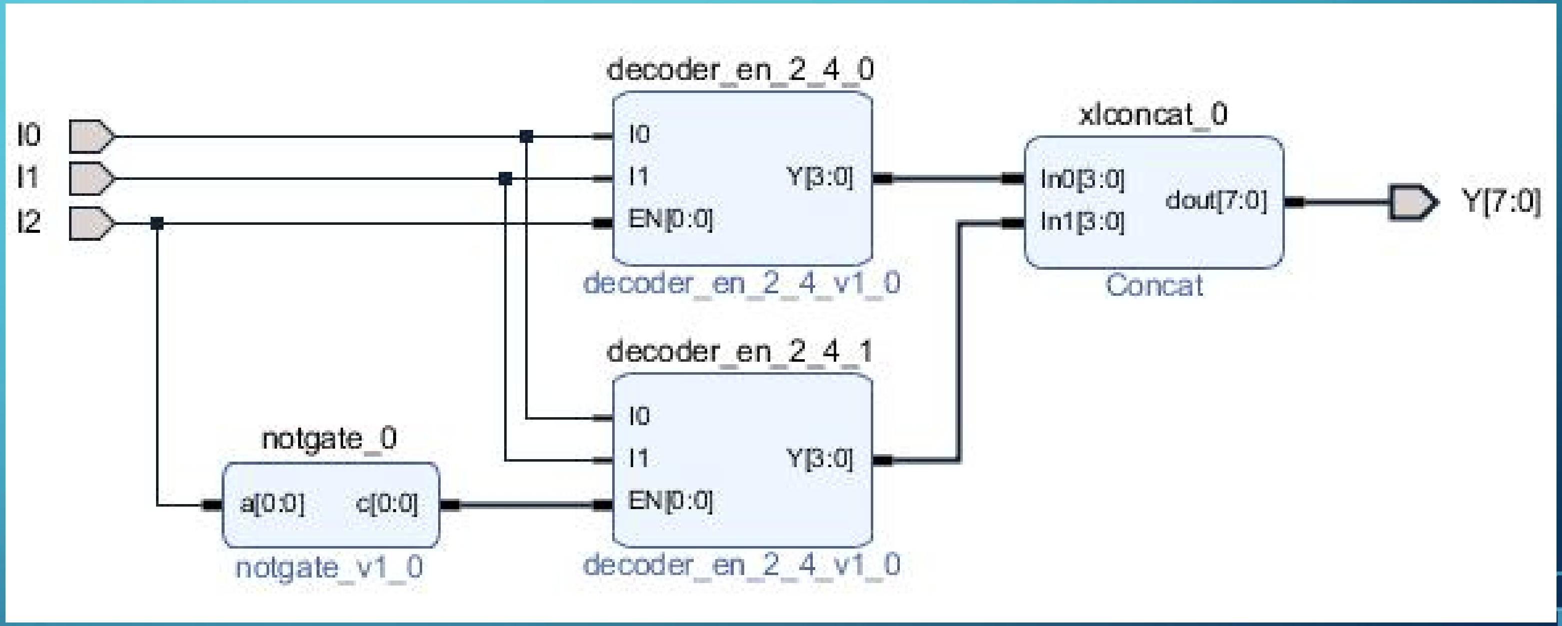
{ I2, I1, I0 }	Y
3’B000	8’B0000_0001
3’B001	8’B0000_0010
3’B010	8’B0000_0100
3’B011	8’B0000_1000
3’B100	8’B0001_0000
3’B101	8’B0010_0000
3’B110	8’B0100_0000
3’B111	8’B1000_0000

{ I2, I1, I0 }	Y
3’B000	8’B0000_0001
3’B001	8’B0000_0010
3’B010	8’B0000_0100
3’B011	8’B0000_1000
3’B100	8’B0001_0000
3’B101	8’B0010_0000
3’B110	8’B0100_0000
3’B111	8’B1000_0000

DECODER （3-8 DECODER 2）

- How to implement an 3-8 decoder by using two 2-4 decoders?

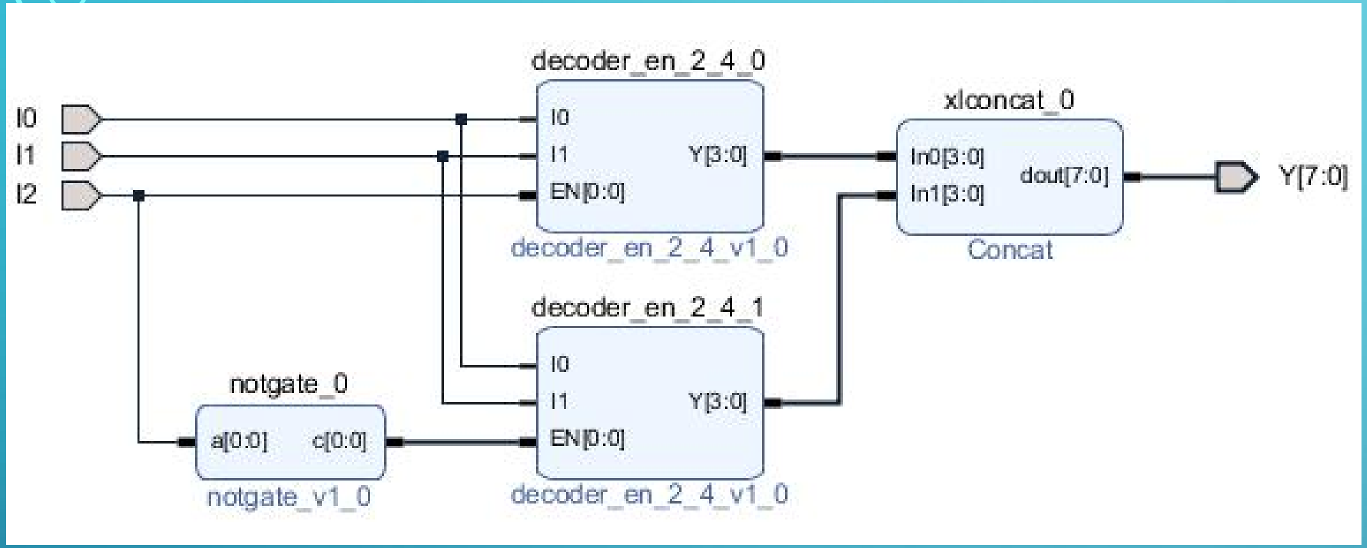
{ I2, I1, I0 }	Y
3'B000	8'B0000_0001
3'B001	8'B0000_0010
3'B010	8'B0000_0100
3'B011	8'B0000_1000
3'B100	8'B0001_0000
3'B101	8'B0010_0000
3'B110	8'B0100_0000
3'B111	8'B1000_0000



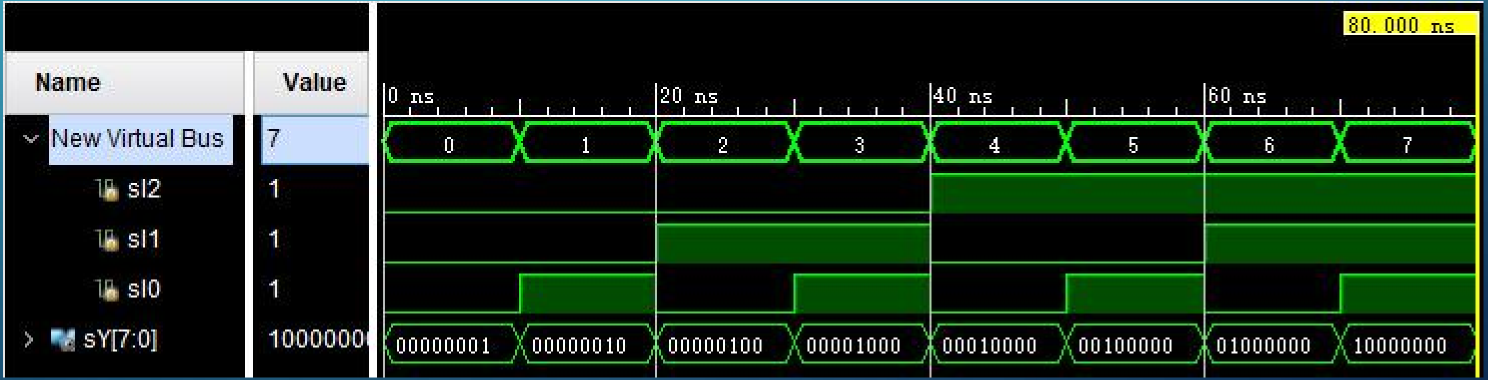
DECODER （3-8 DECODER 3）

- How to implement an 3-8 decoder by using two 2-4 decoders?

{ I2, I1, I0 }	Y
3'B000	8'B0000_0001
3'B001	8'B0000_0010
3'B010	8'B0000_0100
3'B011	8'B0000_1000
3'B100	8'B0001_0000
3'B101	8'B0010_0000
3'B110	8'B0100_0000
3'B111	8'B1000_0000




```
module decoder_3_8(
input I2, I1, I0,
output [7:0]Y
    );
    wire [3:0] Y_7_4, Y_3_0;
    decoder_en_2_4 u0(.EN(I2),.I1(I1),.I0(I0),.Y(Y_3_0));
    decoder_en_2_4 u1(.EN(~I2),.I1(I1),.I0(I0),.Y(Y_7_4));
    assign Y={ Y_7_4 ,Y_3_0 };
endmodule
```



WANGW6@SUSTECH.EDU.CN

DO THE CIRCUIT DESIGN BY DECODER(1)

input		output			
I1	I0	Y3	Y2	Y1	Y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

truth table 2-4 decoder

Sum Of Minterm on Decoder-2-4

Y0 = I1'.I0'
Y1 = I1'.I0
Y2 = I1.I0'
Y3 = I1.I0

```
//2-4decoder
module decoder(
    input I0,
    input I1,
    output reg [3:0] Y
    );
    always @*
    begin
        case ({I1,I0})
            2'b00: Y=4'b0001;
            2'b01: Y=4'b0010;
            2'b10: Y=4'b0100;
            2'b11: Y=4'b1000;
        endcase
    end
endmodule
```

Implement a circuit by Decoder-2-4 and logical gates. F(A,B) = A + A'B

step1:
F(A,B) = A(B+B') + A'B = AB + AB' + A'B

step2:
using Decoder-2-4, A connects to I1, B connects to I0,

setp3:
AB : Y3 , AB': Y2, A'B: Y1

step4:
F(A,B) = Y3 | Y2 | Y1

DO THE CIRCUIT DESIGN BY DECODER(2)

```
//2-4decoder
module decoder(
    input I0,
    input I1,
    output reg [3:0] Y
    );
    always @*
    begin
        case ({I1,I0})
            2'b00: Y=4'b0001;
            2'b01: Y=4'b0010;
            2'b10: Y=4'b0100;
            2'b11: Y=4'b1000;
         endcase
    end
endmodule
```

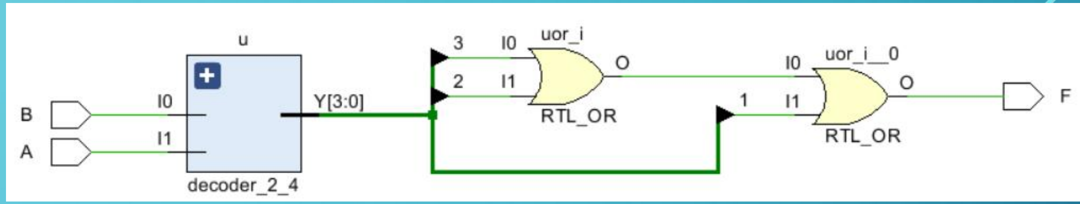
Implement a circuit by Decoder-2-4 and logical gates. F(A,B) = A + A'B

step1:
F(A,B) = A(B+B') + A'B = AB + AB' + A'B

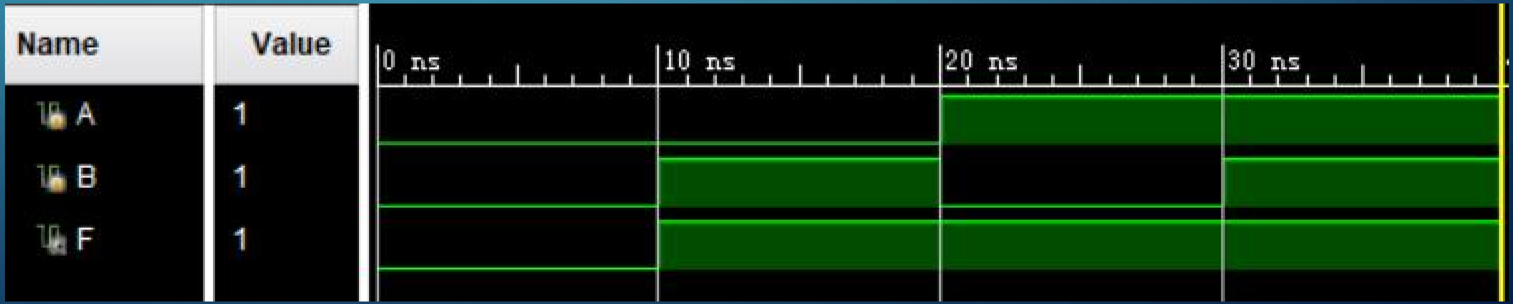
step2:
using Decoder-2-4, A connects to I1, B connects to I0,

setp3:
AB : Y3 , AB': Y2, A'B: Y1

step4:
F(A,B) = Y3 | Y2 | Y1




```
module lab7_demo1(
input A,B,
output F
);
//F(A,B) = A + A'B;
wire [3:0] dout4;
decoder_2_4 u(.I1(A),.I0(B),.Y(dout4));
or uor(F,dout4[3],dout4[2],dout4[1]);
endmodule
```



PRACTICE-1

1. Design a 4-2 Programmable priority encoder in which the bit of input which has the highest priority is determined by another input signal, the priority is successively reduced from this bit to the right.

   1) ports:

      a. Input port X is the encoded object which is encoded to Y, Y is the output port;

      b. Another input port P which is used to indicate the index of the highest priority bit in X. for example: if the value of input which indicate the highest priority is 2, it means the priority bit from high to low is : 2 1 0 3

      Ps: in this circuit, X is 4-bit width, the index of LSB is 0, the index of MSB is 3.

2. Build a testbench, do the simulation and verify the function of your design.

PRACTICE-2

- Implement a 4-16 decoder in two design module:
  - Decoder_4_16_by38 : by two 3-8 decoders in structural manner.
    - You can either modify the provided 3-8 decoder or design 74138 decoder
  - Decoder_4_16_inBe: in behavior manner.

- Do the design and verify the function of your design.
    - Build the testbench , to the simulation.

MULTIPLEXER

- a **Multiplexer** (or MUX) is a device that selects one of several input signals and forwards the selected input to the output**.**
- A multiplexer of **2^n** inputs has **n** select lines. Select lines are used to select one of the input line to be sent to the output.
- Multiplexers are mainly used to increase the amount of data that can be sent over the network within a certain amount of time and bandwidth. A multiplexer is also called a **data selector**.
- Multiplexers can also be used to implement Boolean functions of multiple variables.

MULTIPLEXER(4-TO-1-LINE MULTIPLEXER1)

$$Y = m_0 . D_0 + m_1 . D_1 + m_2 . D_2 + m_3 . D_3$$

Y = (s1'.s0').D0 + (s1'.s0).D1 + (s1.s0').D2+(s1.s0).D3

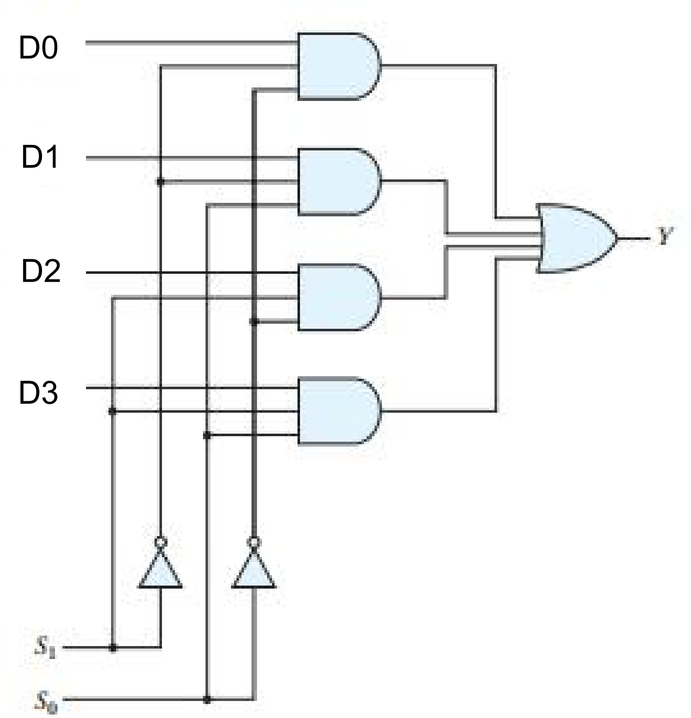
selection input		output
s1	s0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

function table for 4-to-1-line multiplexer

1. There are 4 input data ports(D0,D1,D2,D3), 2 select lines(S1,S0), 1 output port(Y).

The value of output **Y** is determined by the value of select lines and the related input data port.

2. "s1" is the MSB of the select lines, "s0" is the LSB of the select lines.



MULTIPLEXER(4-TO-1-LINE MULTIPLEXER2)

$$Y = m_0 . D_0 + m_1 . D_1 + m_2 . D_2 + m_3 . D_3$$

Y = (s1'.s0').D0 +(s1'.s0).D1+(s1.s0').D2+(s1.s0).D3

selection input		output
s1	s0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

function table for 4-to-1-line multiplexer

1. There are 4 input data ports(D0,D1,D2,D3), 2 select lines(S1,S0), 1 output port(Y).

The value of output Y is determined by the value of select lines and the related input data port.

2. “s1” is the MSB of the select lines, “s0” is the LSB of the select lines.

```
module multiplexer(
input D0, D1, D2, D3,  //data-input
input [1:0] s,  //select
output reg o
    );
    always @ * begin
        case(s)
            2'b00: o = D0;
            2'b01: o = D1;
            2'b10: o = D2;
            2'b11: o = D3;
        endcase
    end
endmodule
```

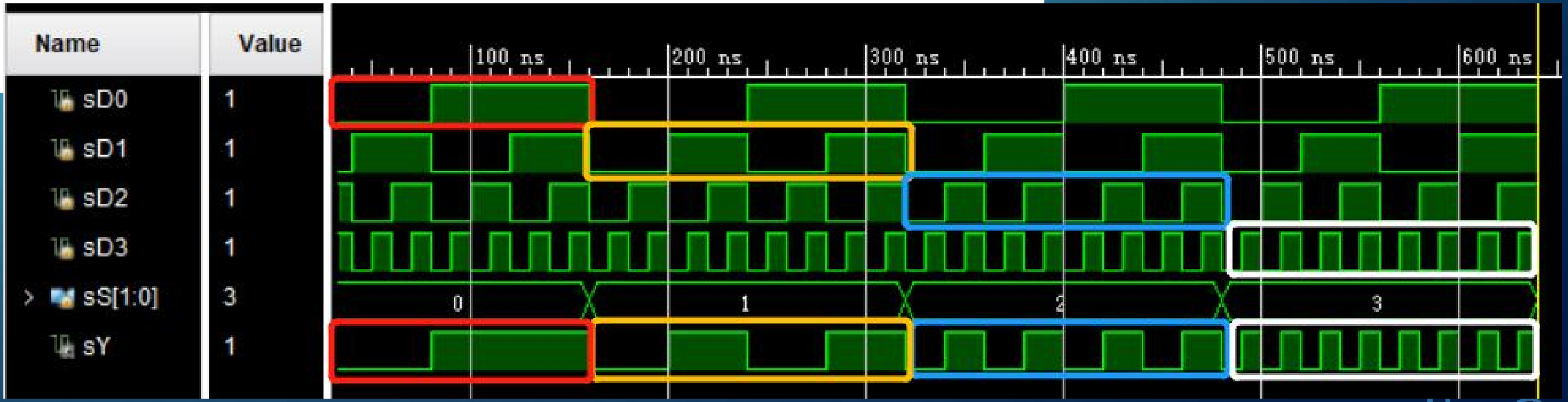
MULTIPLEXER(4-TO-1-LINE MULTIPLEXER3)

$$Y = m_0 . D_0 + m_1 . D_1 + m_2 . D_2 + m_3 . D_3$$

Y = (s1'.s0').D0 +(s1'.s0).D1+(s1.s0').D2+(s1.s0).D3

```verilog
module multiplexer(
input D0, D1, D2, D3,  //data-input
input [1:0] s,  //select
output reg o
    );
    always @ * begin
        case(s)
            2'b00: o = D0;
            2'b01: o = D1;
            2'b10: o = D2;
            2'b11: o = D3;
        endcase
    end
endmodule
```

```verilog
module mux_tb();
    reg sD0, sD1, sD2, sD3;
    reg [1:0]sS;
    wire sY;
    //module multiplexer( input D0, D1, D2, D3,  input [1:0] s,  output reg o);
    multiplexer u(sD0, sD1, sD2, sD3, sS, sY);
    initial begin
        {sS, sD0, sD1, sD2, sD3} = 6'b0;
        repeat(63) #10 {sS, sD0, sD1, sD2, sD3}  = {sS, sD0, sD1, sD2, sD3} + 1;
        #10 $finish;
    end
endmodule
```



selection input		output
s1	s0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

function table for 4-to-1-line multiplexer

MULTIPLEXER(74151:8-TO-1-LINE MULTIPLEXER1)

inputs				output	
EN	S2	S1	S0	Y	W
1	X	X	X	0	1
0	0	0	0	D0	D0'
0	0	0	1	D1	D1'
0	0	1	0	D2	D2'
0	0	1	1	D3	D3'
0	1	0	0	D4	D4'
0	1	0	1	D5	D5'
0	1	1	0	D6	D6'
0	1	1	1	D7	D7'

function table for 74151

1. EN is low level effective.

While EN is effective, the circuit work as a 8-to-1-line multiplexer.

2. There are 8 input data ports, 3 select lines.

The value of output Y is determined by the value of select lines and the related input data port, W is the invert of Y.

3. "s2" is the MSB of the select lines, "s0" is the LSB of the select lines.

```
module multiplexer74151( EN,S2,S1,S0,D7,D6,
D5,D4,D3, D2,D1,D0,Y,W);
    input EN, S2, S1,S0, D7, D6, D5,
     D4, D3, D2, D1, D0;
    output reg Y;
    output W;
    always @*
    if (~EN)
        case ({S2,S1,S0})
            3'b000: Y = D0;
            3'b001: Y = D1;
            3'b010: Y = D2;
            3'b011: Y = D3;
            3'b100: Y = D4;
            3'b101: Y = D5;
            3'b110: Y = D6;
            3'b111: Y = D7;
        endcase
    else
        Y = 1'b0;
    assign W=~Y;
endmodule
```

WANGW6@SUSTECH.EDU.CN

MULTIPLEXER(74151:8-TO-1-LINE MULTIPLEXER2)

inputs				output	
EN	S2	S1	S0	Y	W
1	X	X	X	0	1
0	0	0	0	D0	D0'
0	0	0	1	D1	D1'
0	0	1	0	D2	D2'
0	0	1	1	D3	D3'
0	1	0	0	D4	D4'
0	1	0	1	D5	D5'
0	1	1	0	D6	D6'
0	1	1	1	D7	D7'

function table for 74151

While EN is effective, the logical express of 74151 (about output Y and data inputs and the select lines) is :

S2 is MSB, S0 is LSB

$$Y = m_0.D_0 + m_1.D_1 + m_2.D_2 + m_3.D_3 + m_4.D_4 + m_5.D_5 + m_6.D_6 + m_7.D_7$$

The logical express of 74151 is :

$$Y = EN'.(m_0.D_0 + m_1.D_1 + m_2.D_2 + m_3.D_3 + m_4.D_4 + m_5.D_5 + m_6.D_6 + m_7.D_7)$$

W = Y'

MULTIPLEXER(IMPLEMENT BOOLEAN FUNCTIONS-1)

- Use 74151 implement the following logic function.

$$F(A,B,C) = \boldsymbol{\overline{A}\,\overline{C} + \overline{B}\,\overline{C} + \overline{A}B + BC}$$

$$= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}BC + ABC + \overline{A}BC$$

$$= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + \overline{A}BC + ABC$$

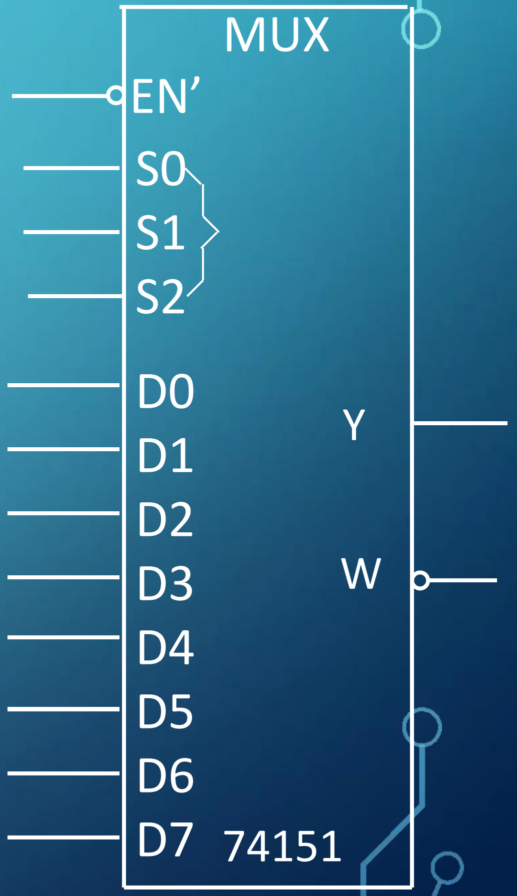
$$= \boldsymbol{\overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\,\overline{C} + ABC}$$

$$=$$

$$= \boldsymbol{m_0.1} + m_1.0 + \boldsymbol{m_2.1} + \boldsymbol{m_3.1} + \boldsymbol{m_4.1} + m_5.0 + m_6.0 + \boldsymbol{m_7.1}$$

While EN is effective, the logical express of 74151 (about output Y and data inputs and the select lines) is :

$$\boldsymbol{Y = m_0.D_0 + m_1.D_1 + m_2.D_2 + m_3.D_3 + m_4.D_4 + m_5.D_5 + m_6.D_6 + m_7.D_7}$$



MULTIPLEXER(IMPLEMENT BOOLEAN FUNCTIONS-2)

$$F(A,B,C) = \overline{A}\overline{C} + \overline{B}\overline{C} + \overline{A}B + BC$$

```
module fun_a_b_c(input A,B,C,output F );
    assign F=( (~A)&(~C) ) | ( (~B)&(~C) )  | ((~A)&B) | (B&C) ;
endmodule
```

$$F(A,B,C) = m_0.1 + m_1.0 + m_2.1 + m_3.1 + m_4.1 + m_5.0 + m_6.0 + m_7.1$$

```
module multiplexer74151( EN,S2,S1,S0,D7,D6,D5,D4,D3, D2,D1,D0,Y,W);
    input EN, S2, S1,S0, D7, D6, D5,
     D4, D3,  D2, D1,  D0;
    output reg Y;
    output W;
    always @*
    if (~EN)
        case ({S2,S1,S0})
            3'b000: Y = D0;
            3'b001: Y = D1;
            3'b010: Y = D2;
            3'b011: Y = D3;
            3'b100: Y = D4;
            3'b101: Y = D5;
            3'b110: Y = D6;
            3'b111: Y = D7;
        endcase
    else
        Y = 1'b0;
    assign W=~Y;
endmodule
```

```
module fun_a_b_c_use_mux(input A,B,C,  output F);
wire sen,sd7,sd6,sd5,sd4,sd3,sd2,sd1,sd0;
wire snf;
assign {sen,sd7,sd6,sd5,sd4,sd3,sd2,sd1,sd0} = 9'b0_1001_1101;

//module multiplexer74151(EN,S2,S1,S0,D7,D6,D5,D4,D3,D2,D1,D0,Y,W);
multiplexer74151 u7411(.EN(sen),
.S2(A),.S1(B),.S0(C),
.D7(sd7),.D6(sd6),.D5(sd5),.D4(sd4),.D3(sd3),.D2(sd2),.D1(sd1),.D0(sd0),
.Y(F),
.W(snf));
endmodule
```

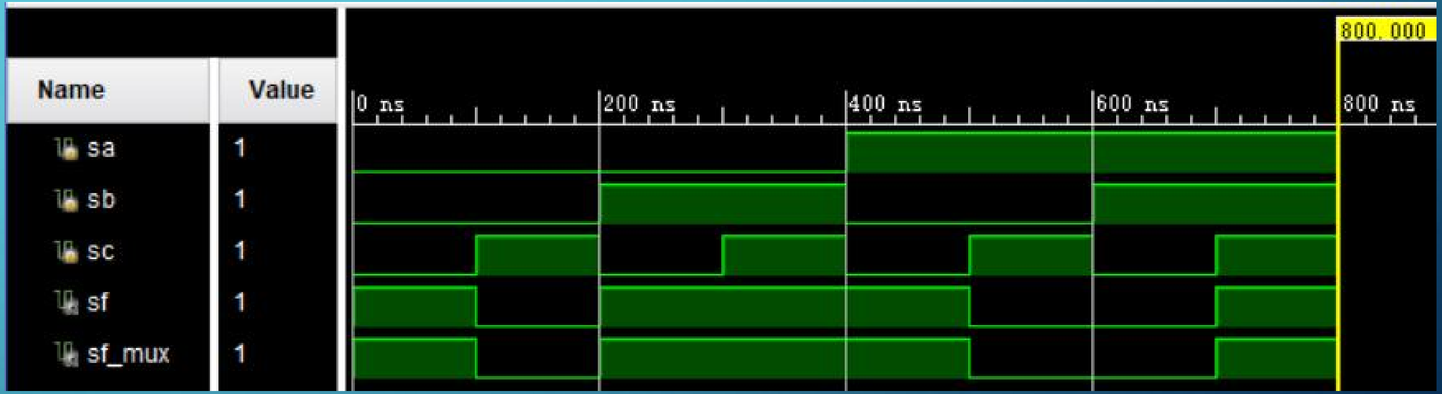
WANGW6@SUSTECH.EDU.CN

MULTIPLEXER(IMPLEMENT BOOLEAN FUNCTIONS-3)

```verilog
module fun_abc_sim( );
reg sa, sb, sc;
wire sf, sf_mux;
//module fun_a_b_c(input A, B, C,  output F  );
fun_a_b_c uf(sa, sb, sc, sf);
//module fun_a_b_c_use_mux(input A, B, C,  output F);
fun_a_b_c_use_mux uf_mux(sa, sb, sc, sf_mux);

initial begin
    {sa, sb, sc} = 3'b000;
    repeat(7) #100  {sa, sb, sc} =  {sa, sb, sc} + 1;
    #100 $finish;
end
endmodule
```

$$F(A,B,C) = \overline{A}\,\overline{C} + \overline{B}\,\overline{C} + \overline{A}B + BC$$



PRACTICE-3

Use 74151(8-to-1-line multiplexer) realize the following logic function

$$F(A,B,C) = \overline{\boldsymbol{A}}\overline{\boldsymbol{C}} + \overline{\boldsymbol{B}}\overline{\boldsymbol{C}} + \overline{\boldsymbol{A}}\boldsymbol{B} + \boldsymbol{B}\boldsymbol{C}$$

- It is asked that A is the LSB of select lines , C is the MSB of select lines.
- Do the design and verify the function of your design.

PRACTICE-4

Use 74151(8-to-1-line multiplexer) realize the following logic function

$Y = A'B'C'D' + BC'D + A'C'D + A'BCD + ACD$

- Do the design and verify the function of your design.
- Create the constraint file, do the synthetic and implementation, generate the bitstream file and program the device, then test on the EGO1 develop board.

DE-MULTIPLEXER

- a **De-multiplexer** (or **De-mux**) is a device taking a single input signal and selecting one of many data-output-lines, which is connected to the single input.

selection input		output			
S1	S0	Y3	Y2	Y1	Y0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

function table of 1-to-4 de-multiplexer
D is the data input

DE-MULTIPLEXER1

selection input		output			
S1	S0	Y3	Y2	Y1	Y0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

function table of 1-to-4 de-multiplexer

D is the data input

```
module demultiplexer(
    input D,
    input [1:0] S,
    output reg Y0,
    output reg Y1,
    output reg Y2,
    output reg Y3
    );
    always@*
    begin
        case (S)
        2'b00: {Y3,Y2,Y1,Y0}={1'b0,1'b0,1'b0,D};
        2'b01: {Y3,Y2,Y1,Y0}={1'b0,1'b0,D,1'b0};
        2'b10: {Y3,Y2,Y1,Y0}={1'b0,D,1'b0,1'b0};
        2'b11: {Y3,Y2,Y1,Y0}={D,1'b0,1'b0,1'b0};
        endcase;
    end
endmodule
```

DE-MULTIPLEXER2

selection input		output			
S1	S0	Y3	Y2	Y1	Y0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

function table of 1-to-4 de-multiplexer

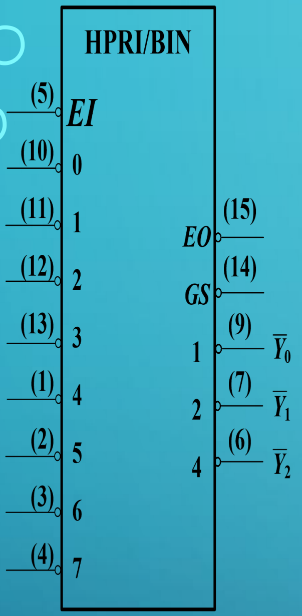
D is the data input

```
module demultiplexer(
    input D,
    input [1:0] S,
    output reg Y0,
    output reg Y1,
    output reg Y2,
    output reg Y3
    );
    always@*
    begin
        case (S)
        2'b00: {Y3,Y2,Y1,Y0}={1'b0,1'b0,1'b0,D};
        2'b01: {Y3,Y2,Y1,Y0}={1'b0,1'b0,D,1'b0};
        2'b10: {Y3,Y2,Y1,Y0}={1'b0,D,1'b0,1'b0};
        2'b11: {Y3,Y2,Y1,Y0}={D,1'b0,1'b0,1'b0};
        endcase;
    end
endmodule
```

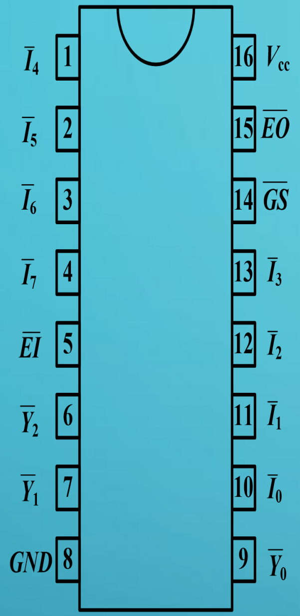
```
module demultiplexer_tb();
    reg [1:0] sS;
    reg sD;
    wire sY0, sY1, sY2, sY3;
    demultiplexer u(sD, sS, sY0, sY1, sY2, sY3);
    initial
    begin
        {sD, sS} = 3'b000;
        repeat(7) #10 {sD, sS} = {sD, sS}+1;
        #10 $finish;
    end
endmodule
```

WANGW6@SUSTECH.EDU.CN

TIPS:ENCODER (74148)



Logic diagram



Pin diagram

74148: 8-3 priority encoder

- The input is low level effective, and the output is 3 bit one's complement.
- HPRI illustrates that the MSB's priority is the highest

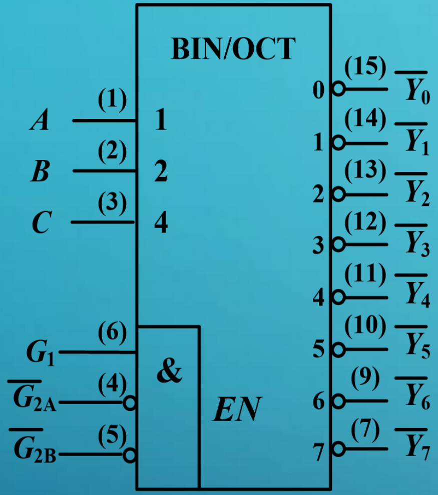
EI: Enable input **EO**: Enable output **GS**: Group select

$\overline{\mathrm{EO}} = \overline{\mathrm{EI}\ \overline{I_0}\,\overline{I_1}\,\overline{I_2}\,\overline{I_3}\,\overline{I_4}\,\overline{I_5}\,\overline{I_6}\,\overline{I_7}}$ $\overline{\mathrm{GS}} = \overline{\mathrm{EI}(I_0 + I_1 + I_2 + I_3 + I_4 + I_5 + I_6 + I_7)}$

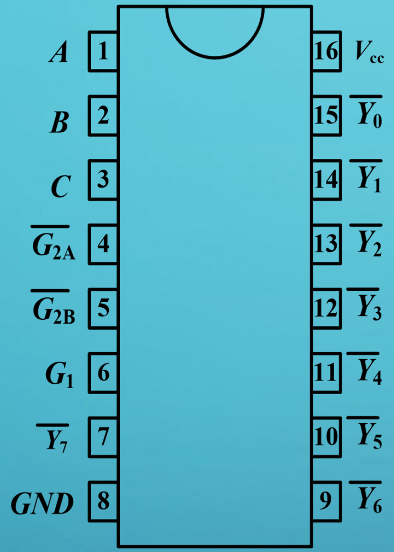
input									output				
EI'	I0'	I1'	I2'	I3'	I4'	I5'	I6'	I7'	Y2'	Y1'	Y0'	GS'	E0'
1	X	X	X	X	X	X	X	X	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	X	X	X	X	X	X	X	0	0	0	0	0	1
0	X	X	X	X	X	X	0	1	0	0	1	0	1
0	X	X	X	X	X	0	1	1	0	1	0	0	1
0	X	X	X	X	0	1	1	1	0	1	1	0	1
0	X	X	X	0	1	1	1	1	1	0	0	0	1
0	X	X	0	1	1	1	1	1	1	0	1	0	1
0	X	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1

truth table of 74148 pri-encoder

TIPS: DECODER (74138)



Logic diagram



Pin diagram

G1	G2A'	G2B'	C	B	A	Y0'	Y1'	Y2'	Y3'	Y4'	Y5'	Y6'	Y7'
0	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	1	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

truth table for 74138 decoder