# DIGITAL DESIGN

LAB8 COMBINATORIAL CIRCUIT: ENCODER, DECODER

2022 SUMMER TERM

# LAB8

- Combinational circuit
  - Encoder
  - Decoder
- Practice

# 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**

| 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 |

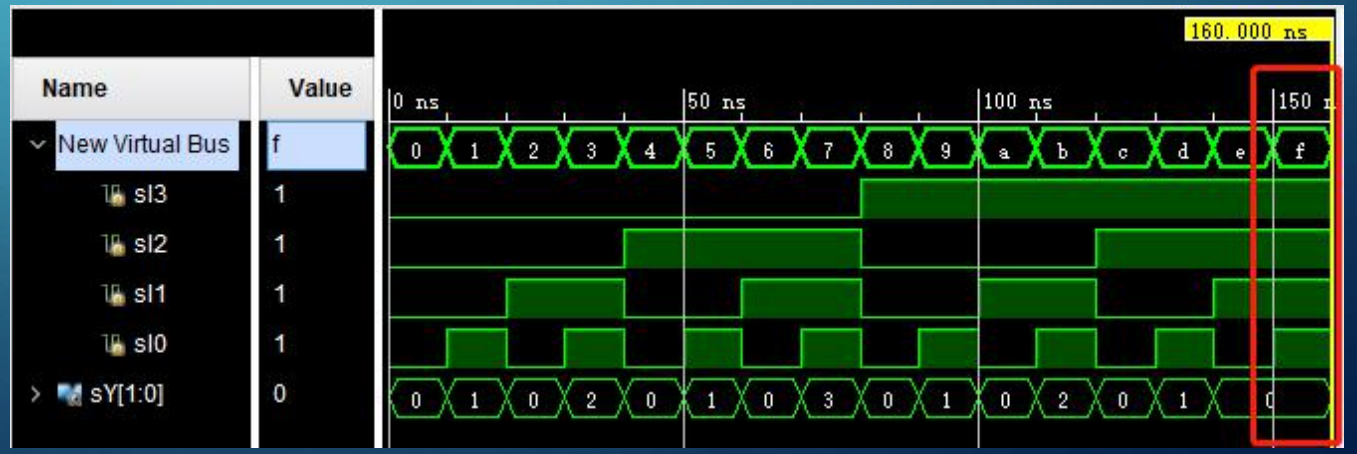
truth table of 4-2 pri-encoder

```
//4-2 priencoder
module encoder(
    input I0,
    input I1,
    input I2,
    input 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
```

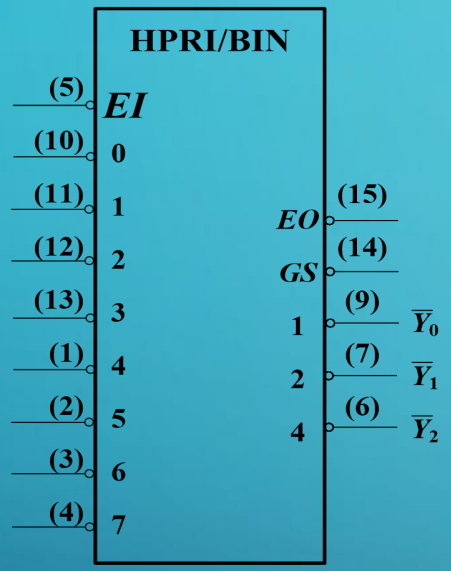
# ENCODER(PRIORITY ENCODER)

```
//4-2 priencoder
module encoder(
    input I0,
    input I1,
    input I2,
    input 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
```

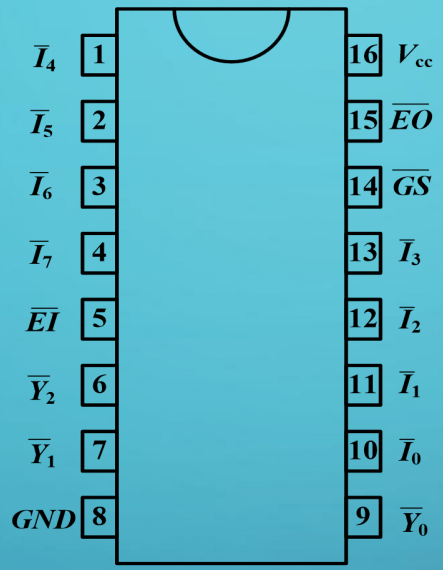
```
module encoder_tb();
    reg sI0, sI1, sI2, sI3;
    wire[1:0] sY;

    encoder u(sI0, sI1, sI2, sI3, sY);
    initial
    begin
        {sI3, sI2, sI1, sI0} = 4'b0000;
        repeat (15)
            #10 {sI3, sI2, sI1, sI0} = {sI3, sI2, sI1, sI0} + 1;
        #10 $finish;
    end
endmodule
```



# ENCODER(74148)




Logic diagram




Pin diagram


Physical photo

- **74148**: 8-3 priority encoder
- The input is low level effective
- The output is 3 bit one's complement.
- HPRI illustrates that the MSB's priority is the highest

# ENCODER(74148)

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

| 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

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

$\overline{\mathrm{EO}} = 0$ : The circuit works, but there is no coding input.

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

$\overline{GS} = 0$ : The circuit works and has coding input.

# 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.

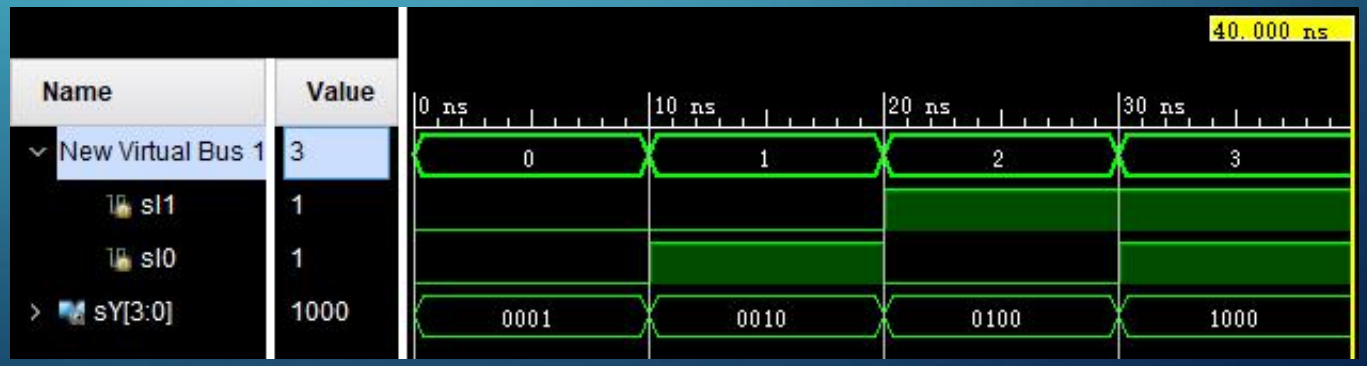
# DECODER（2-4 DECODER）

```
//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

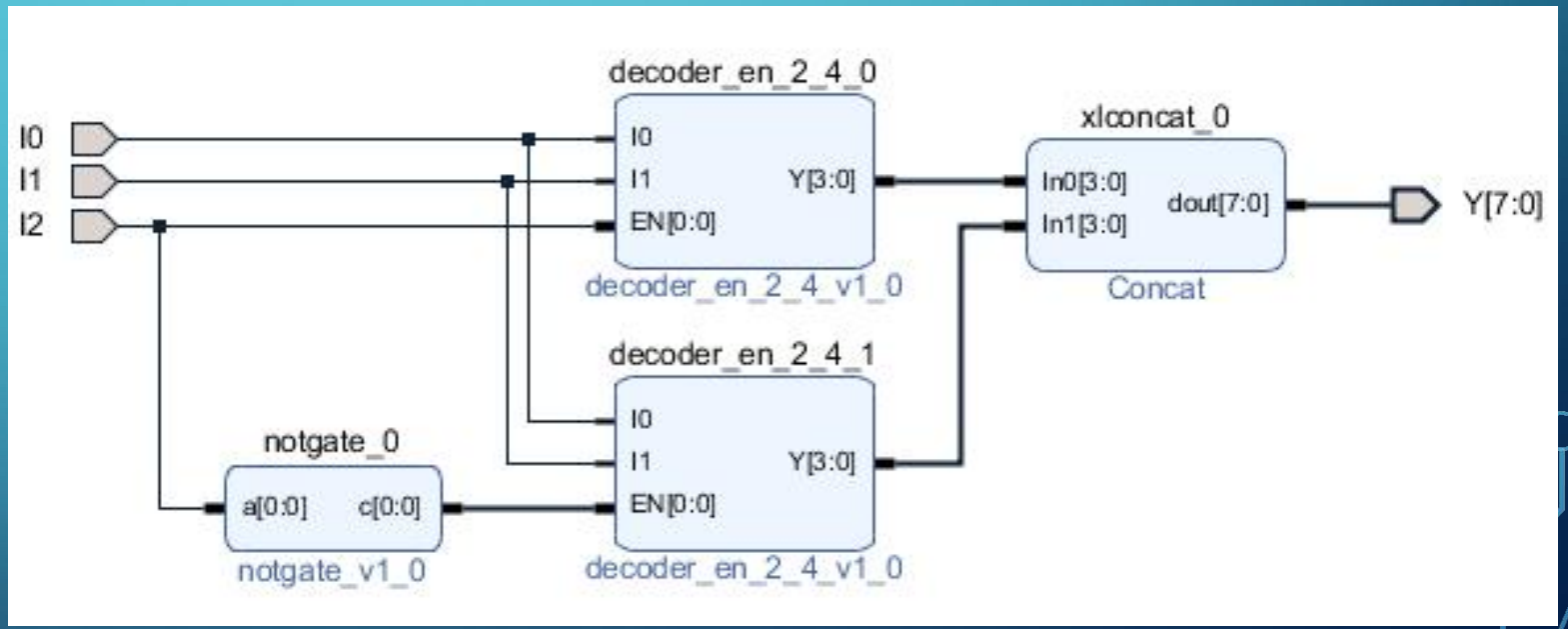
- **One hot coding**, also known as one bit effective coding
  - use n-bit status register to code n states.
  - Each state has its own register bits, and at any time, only one of them is valid.

# DECODER（3-8 DECODER）

Enable input port

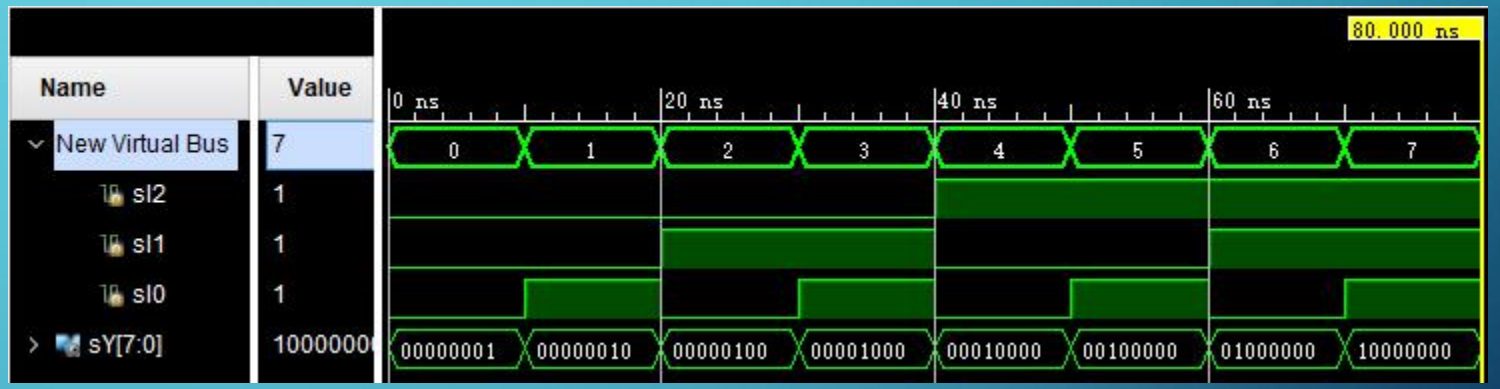
- How to implement an 3-8 decoder by using two 2-4 decoders?

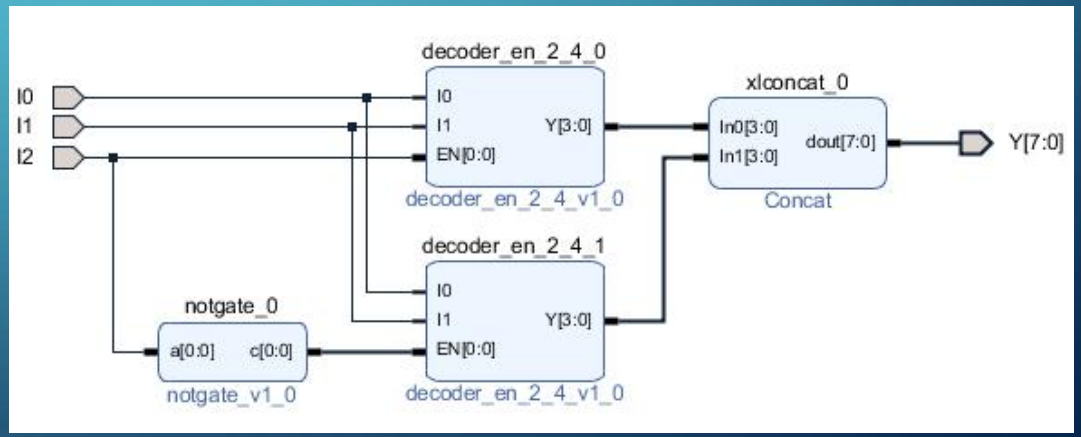
```
module decoder_en #(parameter En_Num = 1)(
    input I0,
    input I1,
    input [En_Num -1: 0]EN,
    output reg [3:0] Y
    );
    always @*
    begin
        if(~EN)//low level effective
        case ({I1,I0})
            2'b00: Y=4'b0001;
            2'b01: Y=4'b0010;
            2'b10: Y=4'b0100;
            2'b11: Y=4'b1000;
         endcase
         else
            Y=4'b0000;
    end
endmodule
```



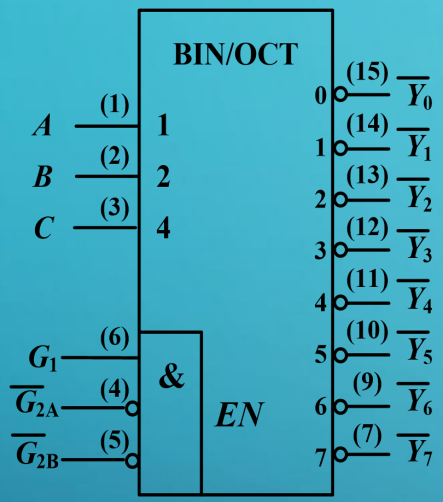
# DECODER（3-8 DECODER）

```
module decoder_3_8_tb();
    reg sI0, sI1, sI2;
    wire [7:0] sY;

    decoder_3_8_wrapper u(sI0, sI1, sI2, sY);
    initial
    begin
        {sI2, sI1, sI0} = 0;
        repeat(7) #10 {sI2, sI1, sI0} = {sI2, sI1, sI0} + 1;
        #10 $finish;
    end
endmodule
```

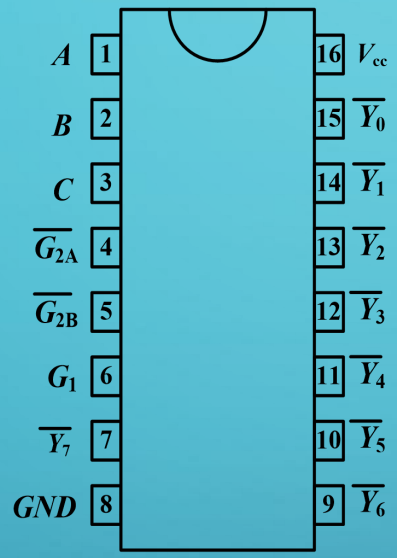


# DECODER（74138）




Logic diagram




Pin diagram

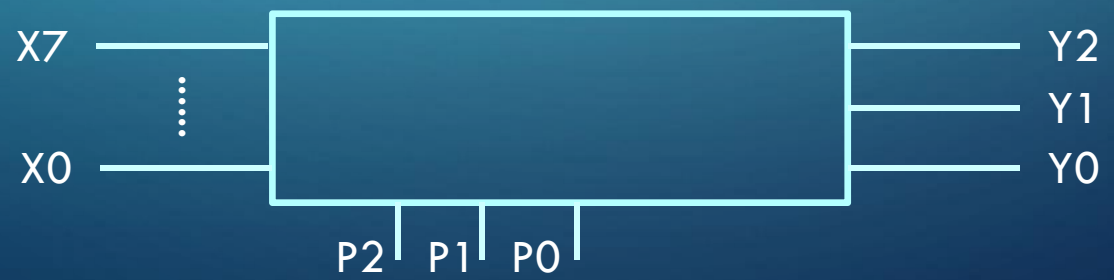

Physical photo

| 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

# PRACTICE 1

1. Design a 8-3 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 in 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
   2) 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 7 6 5 4 3
      Ps: in this circuit, X is 8-bit width, the index of LSB is 0, the index of MSB is 7.
2. Build a testbench, do the simulation and verify the function of your design.



# PRACTICE 2 (AFTER CLASS)

- Implement a 4-16 decoder by two 3-8 decoders.
  - Do the design using both structured design and block design methods.
  - Verify the function of your design.
  - Create the constraint file, do the synthesis and implementation, generate the bitstream file and program the device, then test on the develop board.