

# 数字逻辑设计

高翠芸

School of Computer Science

gaocuiyun@hit.edu.cn

# Unit 7 组合逻辑元件

---

- 多路复用器(multiplexers)
- 三态器件(Three-state Buffer)
- 译码器(Decoders)
- 编码器(Encoders)
- 奇偶校验器
- 比较器
- 利用MSI设计组合逻辑电路
- 只读存储器(ROM)

# Unit 7 组合逻辑元件

---

- 多路复用器(multiplexers)
- 三态器件(Three-state Buffer)
- 译码器(Decoders)
- 编码器(Encoders)
- 奇偶校验器
- 比较器
- 利用MSI设计组合逻辑电路
- 只读存储器(ROM)

# Design with MSI blocks

---

- ① 了解各类典型集成电路芯片的功能、外特性；
- ② 学会查阅器件资料；
- ③ 能灵活运用，完成最佳设计。

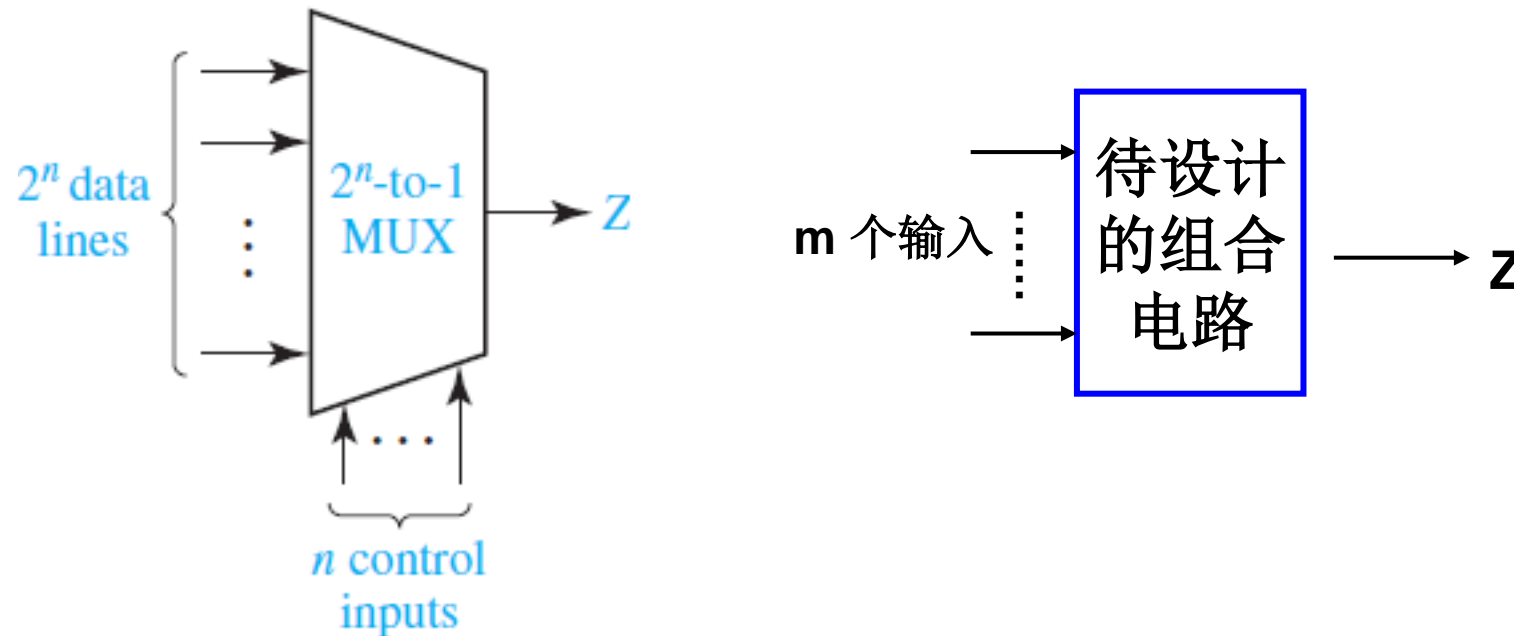
MSI blocks {

- Multiplexers
- Decoders

# 1.用数据选择器来实现组合逻辑电路

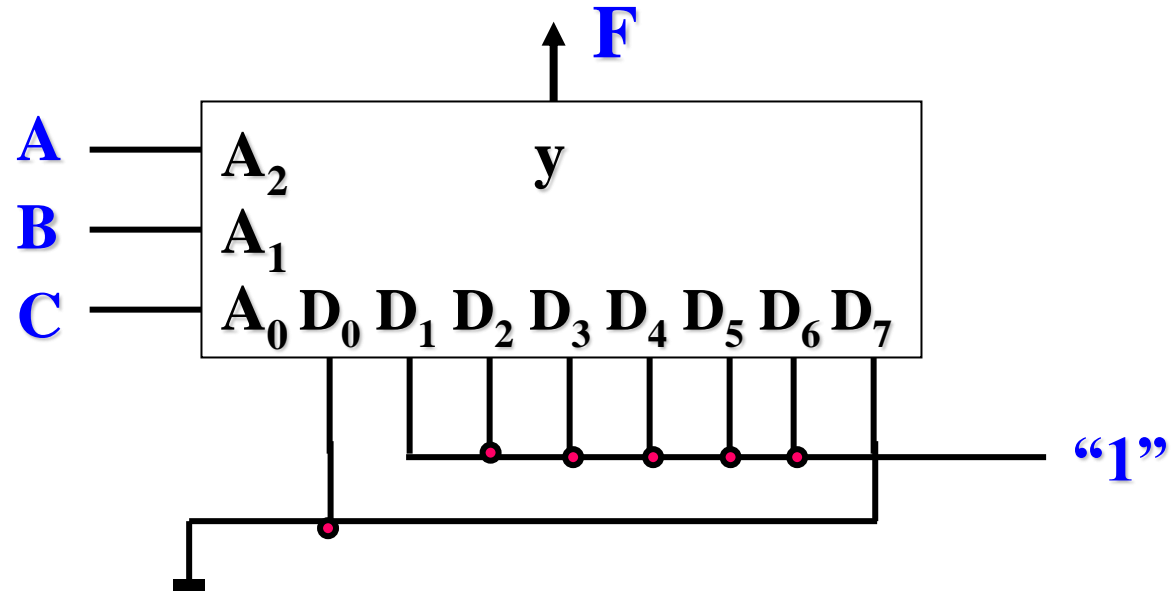
(1)  $m = n$

数据选择器的控制端个数  
组合电路的输入变量个数



Use 8 to 1 MUX realize  $F = A\bar{B} + \bar{A}C + B\bar{C}$

$A_2A_1A_0$	$y$
0 0 0	$D_0$
0 0 1	$D_1$
0 1 0	$D_2$
0 1 1	$D_3$
1 0 0	$D_4$
1 0 1	$D_5$
1 1 0	$D_6$
1 1 1	$D_7$



A	BC			
	00	01	11	10
0	0	1	1	1
1	1	1	0	1

K.Map of F

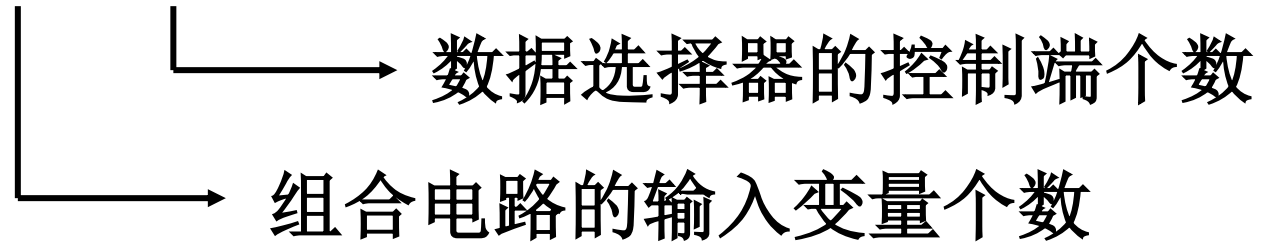
$A_2$	$A_1A_0$			
	00	01	11	10
0	$D_0$	$D_1$	$D_3$	$D_2$
1	$D_4$	$D_5$	$D_7$	$D_6$

K.Map of MUX



# 1.用数据选择器来实现组合逻辑电路

(2)  $m = n+1$



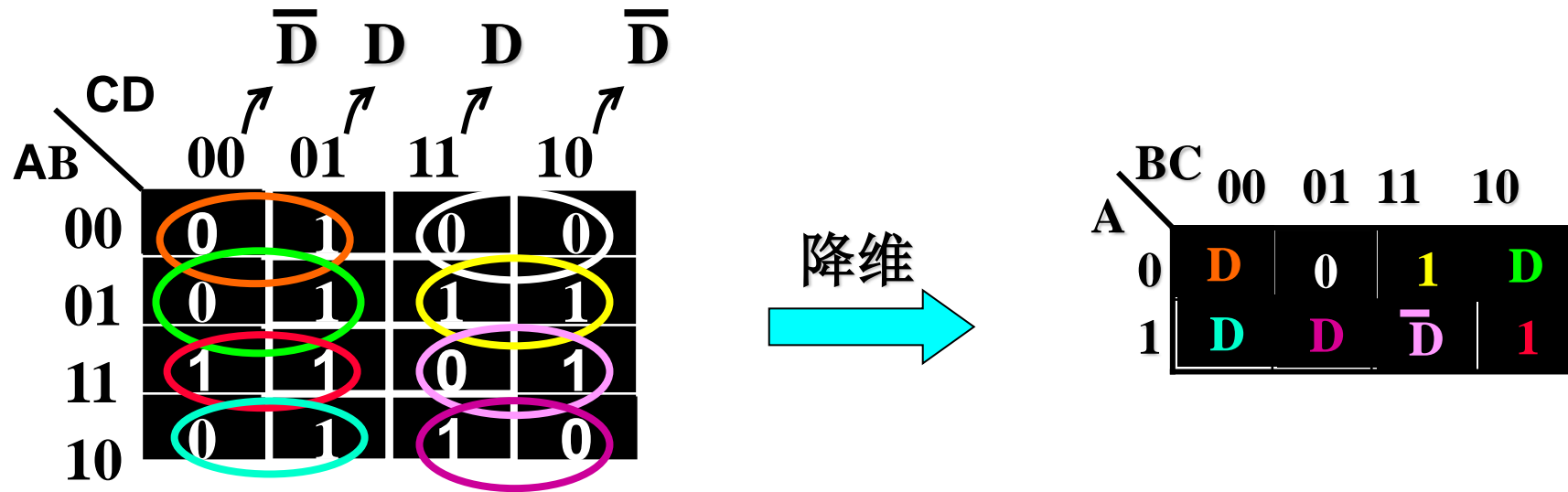
**Method:** 降维

利用 8 to 1 MUX 设计组合逻辑：

$$F(A,B,C,D)=\sum m(1,5,6,7,9,11,12,13,14)$$

# 1.用数据选择器来实现组合逻辑电路

$$F(A,B,C,D)=\sum m(1,5,6,7,9,11,12,13,14)$$



$$f(x_1x_2\dots x_i\dots x_n)$$

$$= x_i \cdot f(x_1x_2\dots \mathbf{1} \dots x_n) + \overline{x_i} \cdot f(x_1x_2\dots \mathbf{0} \dots x_n)$$

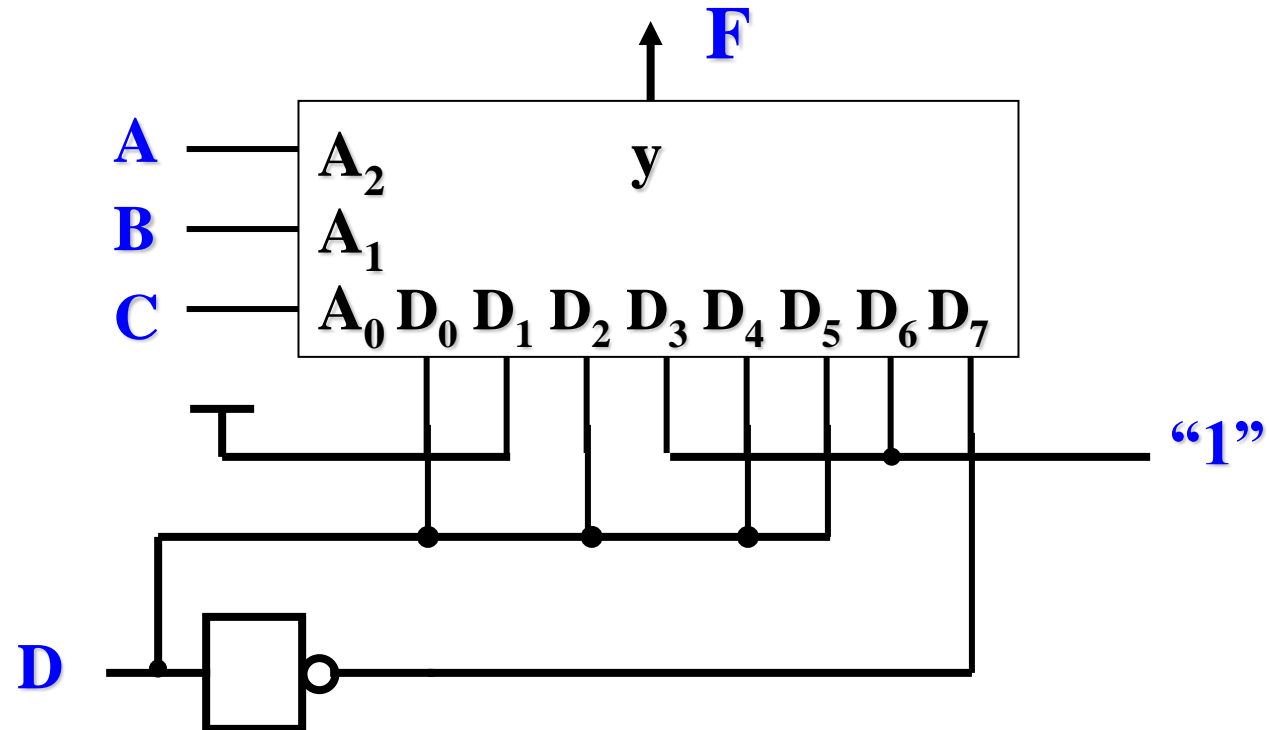


$A_2$	$A_1 A_0$			
	00	01	11	10
0	$D_0$	$D_1$	$D_3$	$D_2$
1	$D_4$	$D_5$	$D_7$	$D_6$

K.Map of MUX

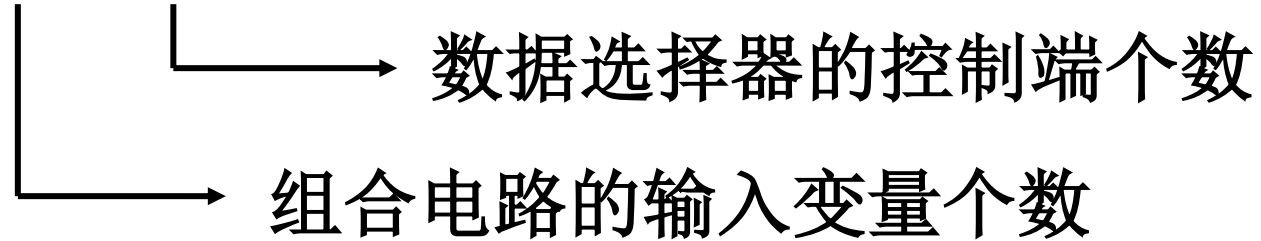
$A$	$BC$			
	00	01	11	10
0	$D$	0	1	$D$
1	$D$	$D$	$\overline{D}$	1

K.Map of F



# 1.用数据选择器来实现组合逻辑电路

(3)  $m = n+2$



**Method: 降维**

**Use 4-to-1 MUX realize :**

$$F(A,B,C,D)=\sum m(0,1,5,6,7,9,10,14,15)$$

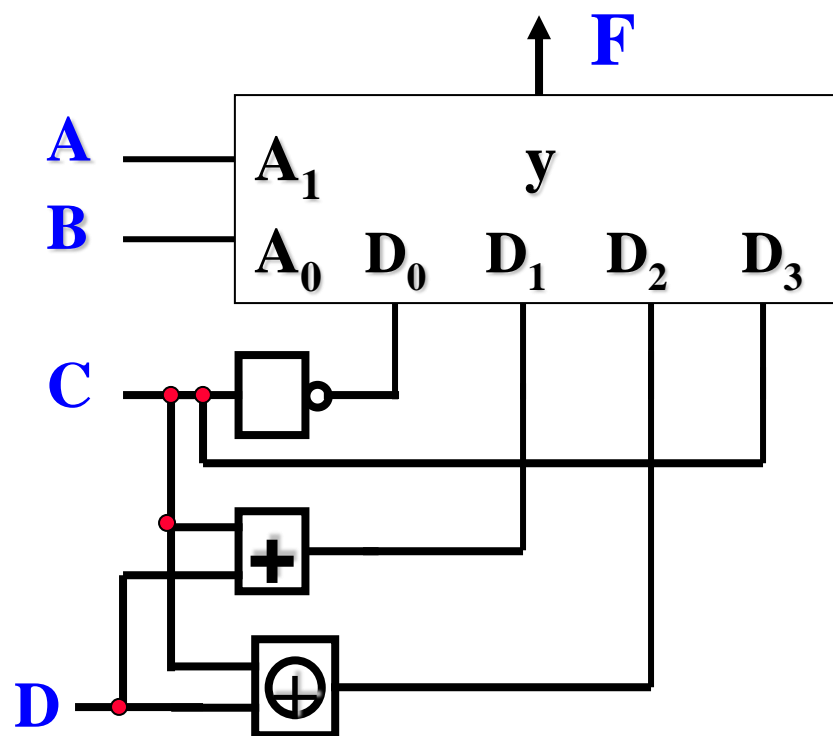
		$\bar{D} \quad D \quad D \quad \bar{D}$			
		$\nearrow \nearrow \nearrow \nearrow$			
CD	AB	00	01	11	10
00		1	1	0	0
01		0	1	1	1
11		0	0	1	1
10		0	1	0	1

降维

		BC			
		00	01	11	10
A	0	1	0	1	D
	1	D	$\bar{D}$	1	0

降维

		B	
		0	1
A	0	$\bar{C}$	$C+D$
	1	$C \oplus D$	C

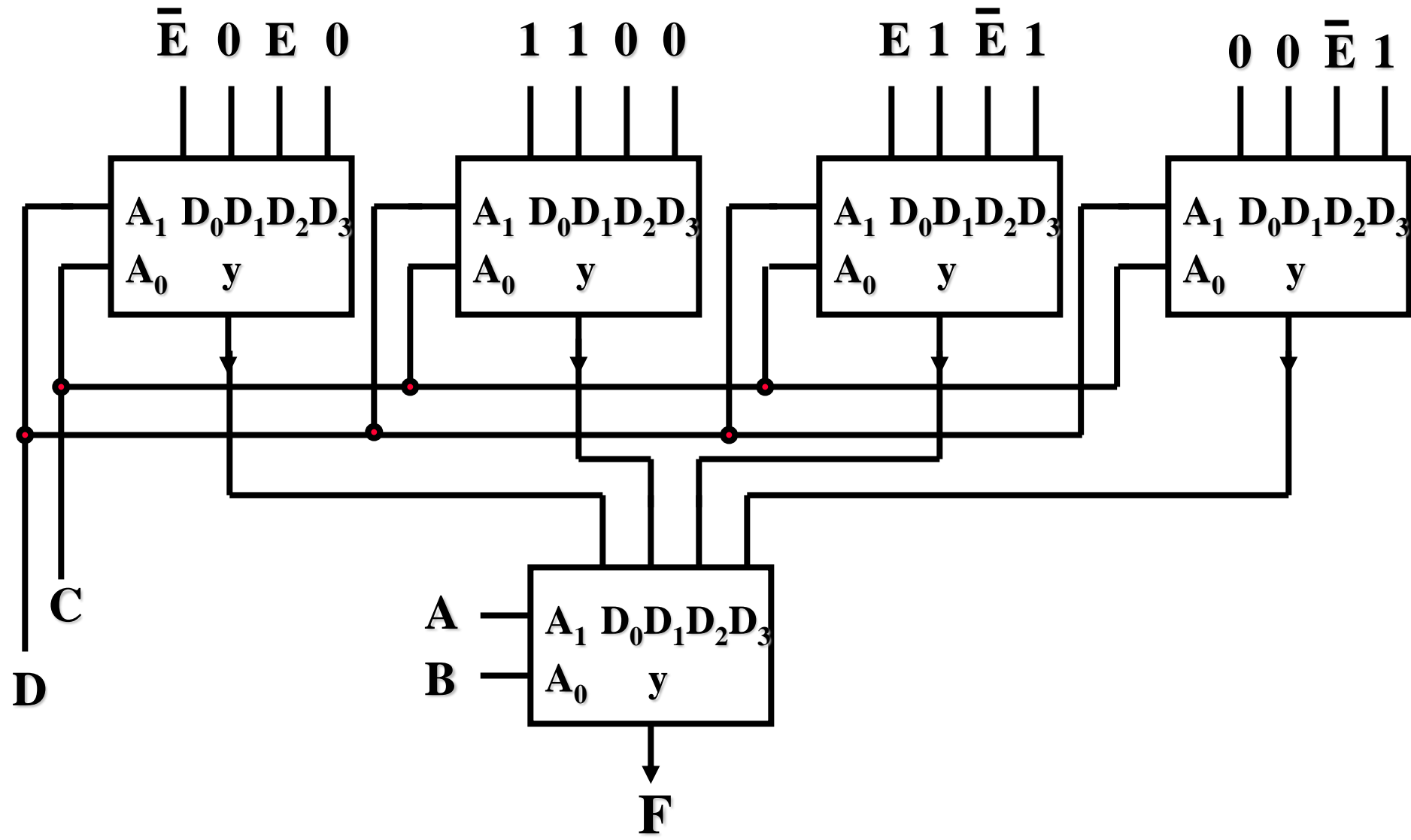


# 利用4-to-1 MUX 设计组合逻辑

$$F(A,B,C,D,E) = \sum m(0,5,8,9,10,11,17,18,19,20,22,23,28,30,31)$$

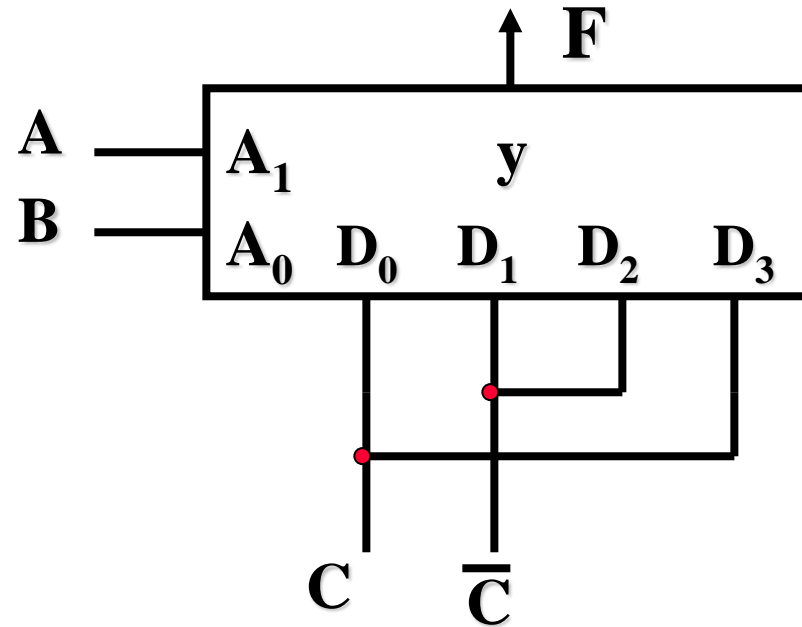
A B	C D	E	F	
00	0 0	0	1	$\bar{E}$
		1	0	
	0 1	0	0	0
		1	0	
	1 0	0	0	E
		1	1	
01	0 0	0	1	1
		1	1	
	0 1	0	1	1
		1	1	
	1 0	0	0	0
		1	0	
	1 1	0	0	0
		1	0	

A B	C D	E	F	
10	0 0	0	0	E
		1	1	
	0 1	0	1	1
		1	1	
	1 0	0	1	$\bar{E}$
		1	0	
11	0 0	0	0	0
		1	0	
	0 1	0	0	0
		1	0	
	1 0	0	1	$\bar{E}$
		1	0	
	1 1	0	1	1
		1	1	



# 1.用数据选择器来实现组合逻辑电路

Write the expression of F



$$F = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

## 2.用数据选择器来实现组合逻辑电路

Use 4-to-1 MUX realize

$$F(A,B,C,D)=\sum m(1,2,4,9, 10,11,12,14,15)$$

**Method:** 降维

从函数的多个输入变量中选出2个作为MUX的选择控制变量。原则上讲，这种选择是任意的，但选择合适时可使设计简化。

① Choose A and B

# ① Choose A and B

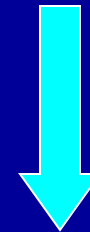
CD AB		$\bar{D}$	D	D	$\bar{D}$
		00	01	11	10
00	0	1	0	1	
01	1	0	0	0	
11	1	0	1	1	
10	0	1	1	1	

降维

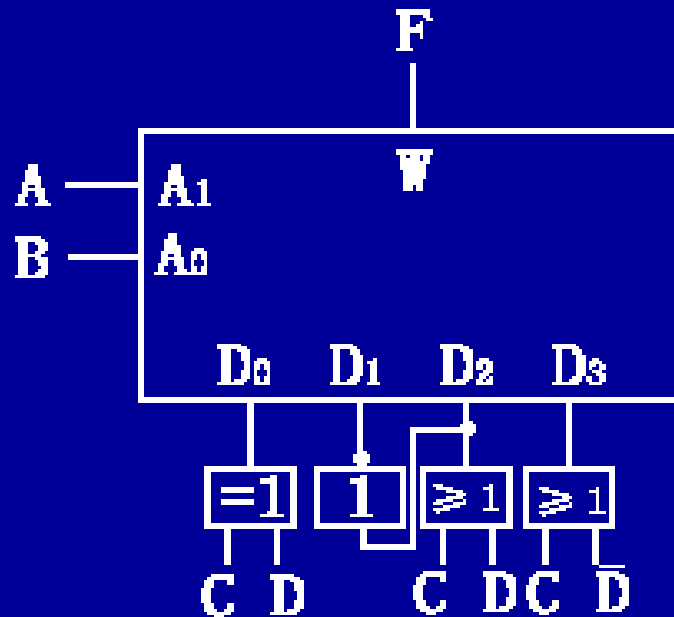


A \ BC		00	01	11	10
		D	$\bar{D}$	0	$\bar{D}$
0	D	$\bar{D}$	0	$\bar{D}$	
1	D	1	1	$\bar{D}$	

降维



A \ B		0	1
		$C \oplus D$	$\overline{C}\overline{D}$
0	$C + D$	$C + \overline{D}$	



(b)



## ② Choose B and C

$\bar{D}$   $D$   $D$   $\bar{D}$

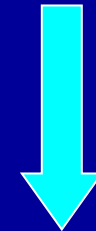
AB \ CD	00	01	11	10
00	0	1	0	1
01	1	0	0	0
11	1	0	1	1
10	0	1	1	1

降维

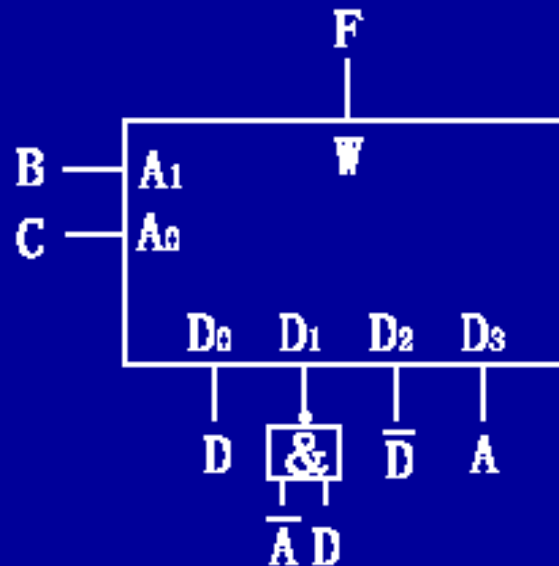


A \ BC	00	01	11	10
0	$D$	$\bar{D}$	0	$\bar{D}$
1	$D$	1	1	$\bar{D}$

降维



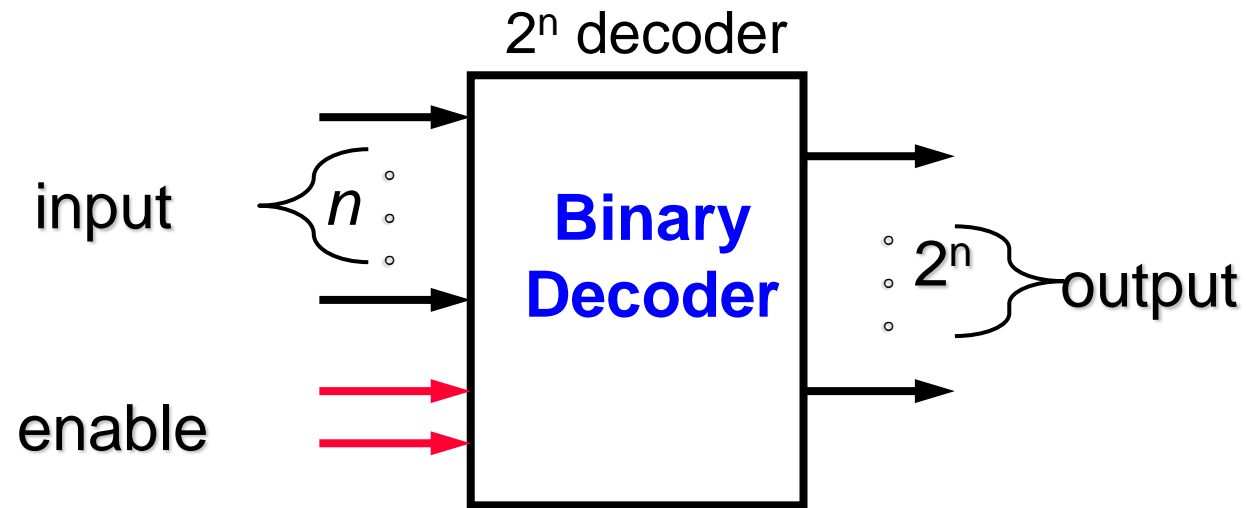
B \ C	0	1
0	$D$	$A + \bar{D}$
1	$\bar{D}$	$A$



(d)

## 2.用译码器来实现组合逻辑电路

MSI blocks {  
▪ Multiplexers  
▪ Decoders



$$y_i = m_i, \quad i = 0 \text{ to } 2^n - 1 \quad (\text{noninverted outputs})$$

$$y_i = m_i' = M_i, \quad i = 0 \text{ to } 2^n - 1 \quad (\text{inverted outputs})$$

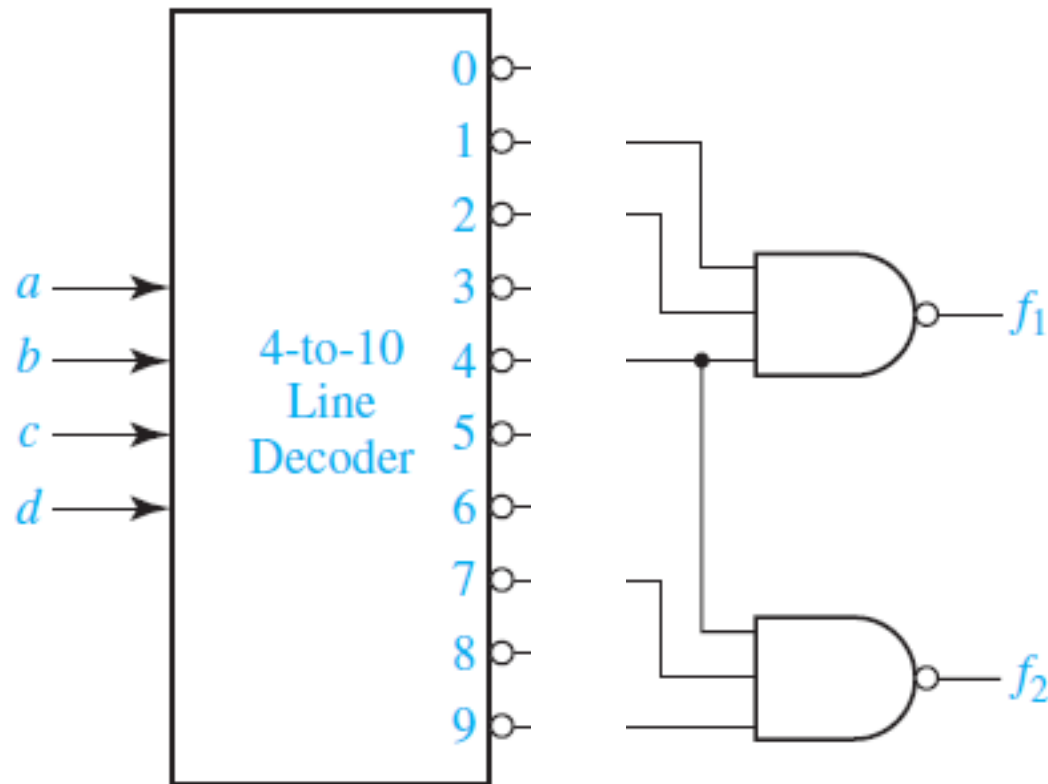
## 2.用译码器来实现组合逻辑电路

$$f_1(a, b, c, d) = m_1 + m_2 + m_4$$

$$f_2(a, b, c, d) = m_4 + m_7 + m_9$$

$$f_1 = (m'_1 m'_2 m'_4)'$$

$$f_2 = (m'_4 m'_7 m'_9)'$$



# 利用 74LS138 设计 1-bit FA

$a_i$	$b_i$	$C_{i-1}$	$S_i$	$C_i$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

74138功能表

使能端			输入			译码输出							
$G_1$	$G_{2A}$	$G_{2B}$	C	B	A	$Y_0$	$Y_1$	$Y_2$	$Y_3$	$Y_4$	$Y_5$	$Y_6$	$Y_7$
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	1	0	1	1
1	0	0	1	0	1	1	1	1	1	1	1	0	1
1	0	0	1	1	0	1	1	1	1	1	1	1	0
1	0	0	1	1	1	1	1	1	1	1	1	1	0

$$y_i = \overline{m}_i$$

$$S_i = \sum (1, 2, 4, 7) = \overline{m}_1 \overline{m}_2 \overline{m}_4 \overline{m}_7$$

$$c_{i-1} = \sum (3, 5, 6, 7) = \overline{m}_3 \overline{m}_5 \overline{m}_6 \overline{m}_7$$

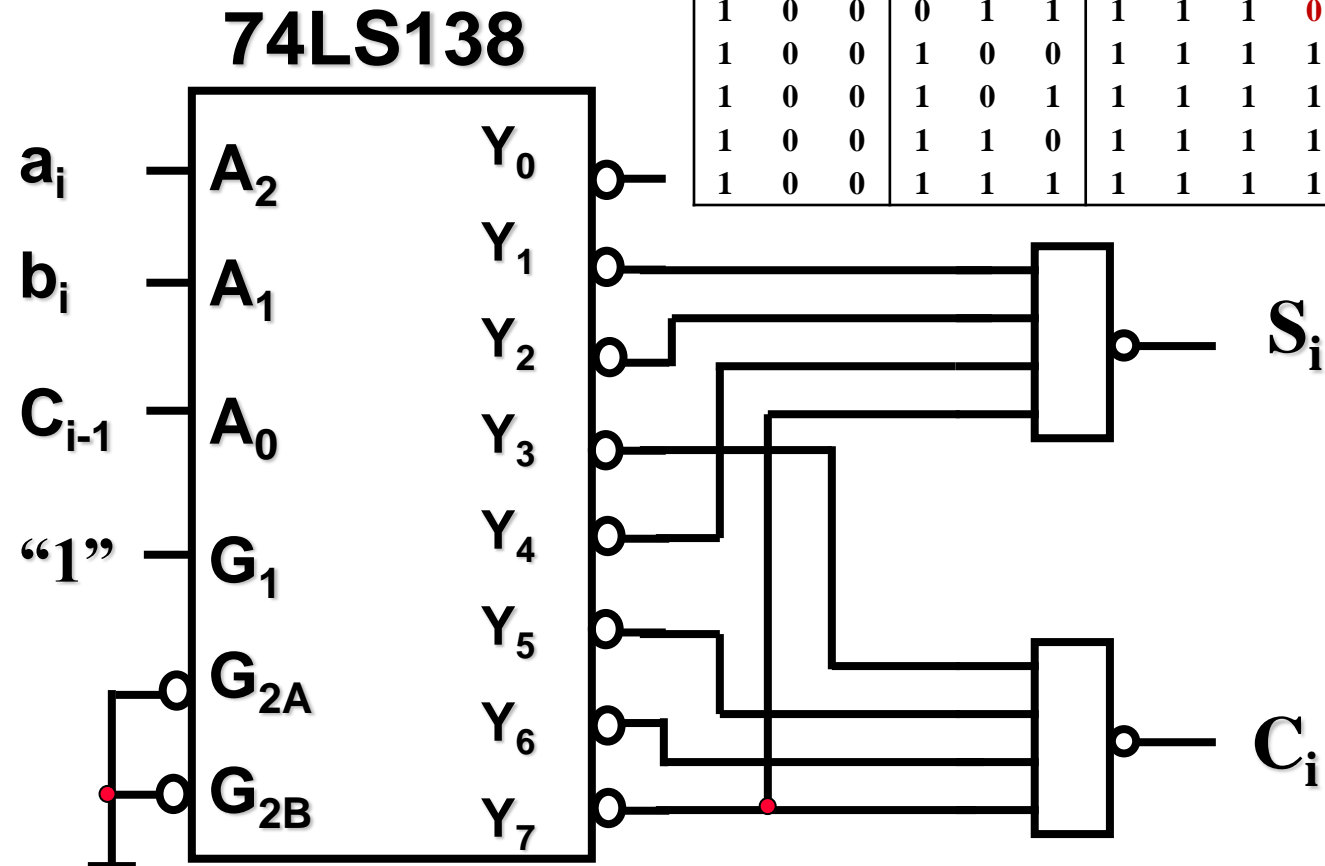
# 74138功能表

$$y_i = \overline{m_i}$$

$$S_i = \sum (1,2,4,7) = \overline{m_1} \overline{m_2} \overline{m_4} \overline{m_7}$$

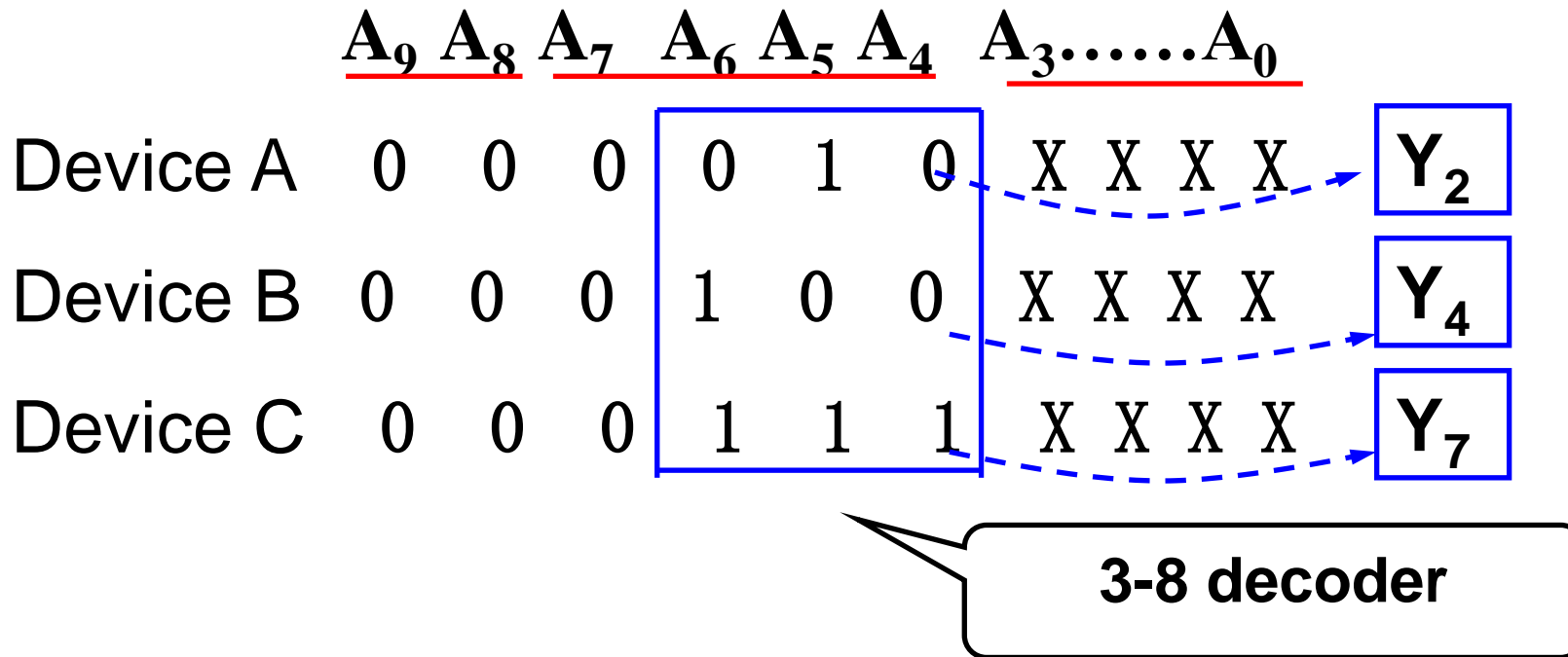
$$c_{i-1} = \sum (3,5,6,7) = \overline{m_3} \overline{m_5} \overline{m_6} \overline{m_7}$$

使能端			输入			译码输出							
$G_1$	$G_{2A}$	$G_{2B}$	C	B	A	$Y_0$	$Y_1$	$Y_2$	$Y_3$	$Y_4$	$Y_5$	$Y_6$	$Y_7$
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



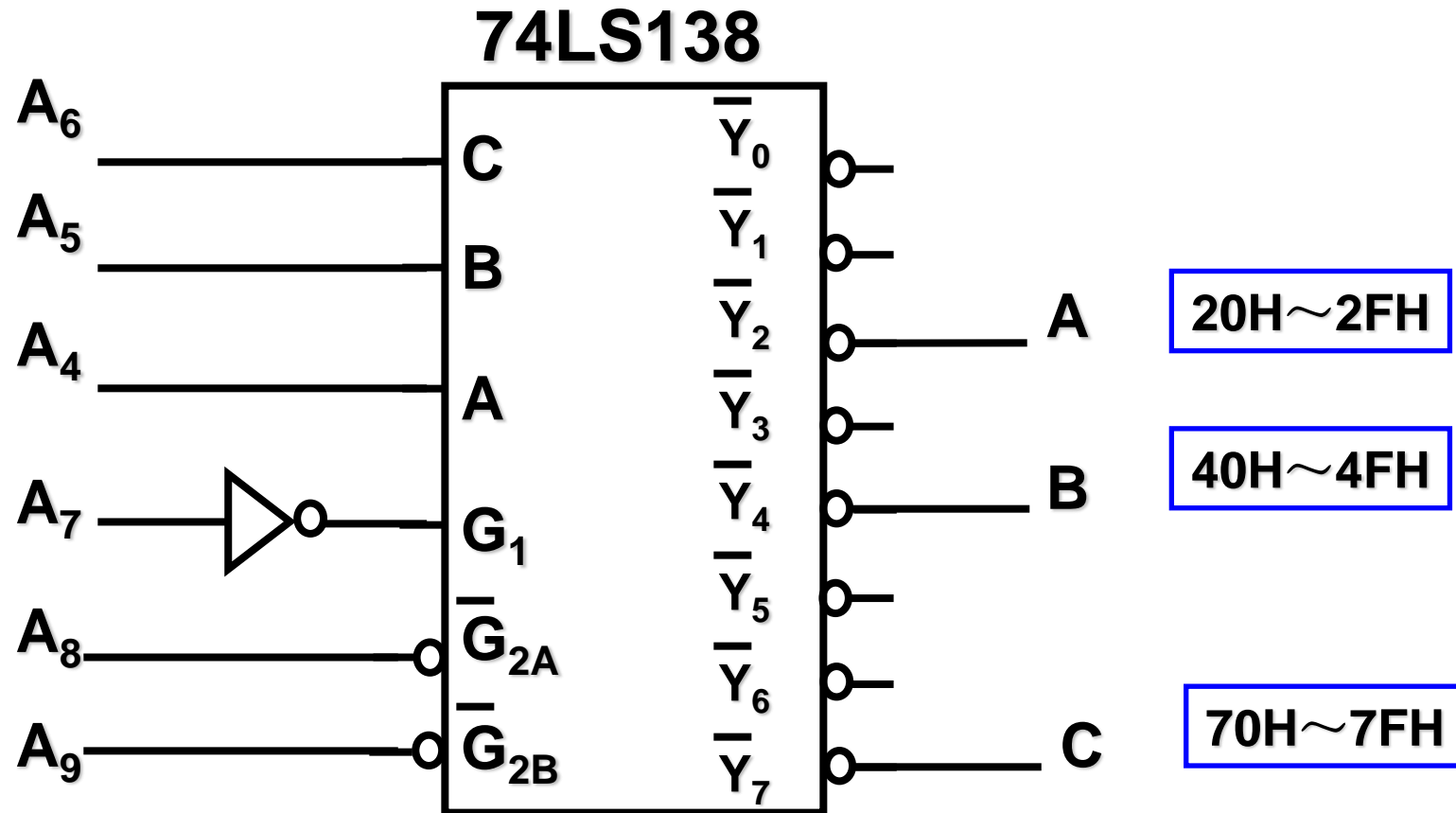
## 2.用译码器来实现组合逻辑电路

Address lines  $A_9 A_8 \dots A_0$  will be used to select devices A, B and C. The address assignment is 20H~2FH, 40H~4FH and 70H~7FH respectively, design the *address decoder* (地址译码器)。



## 2.用译码器来实现组合逻辑电路

Circuit



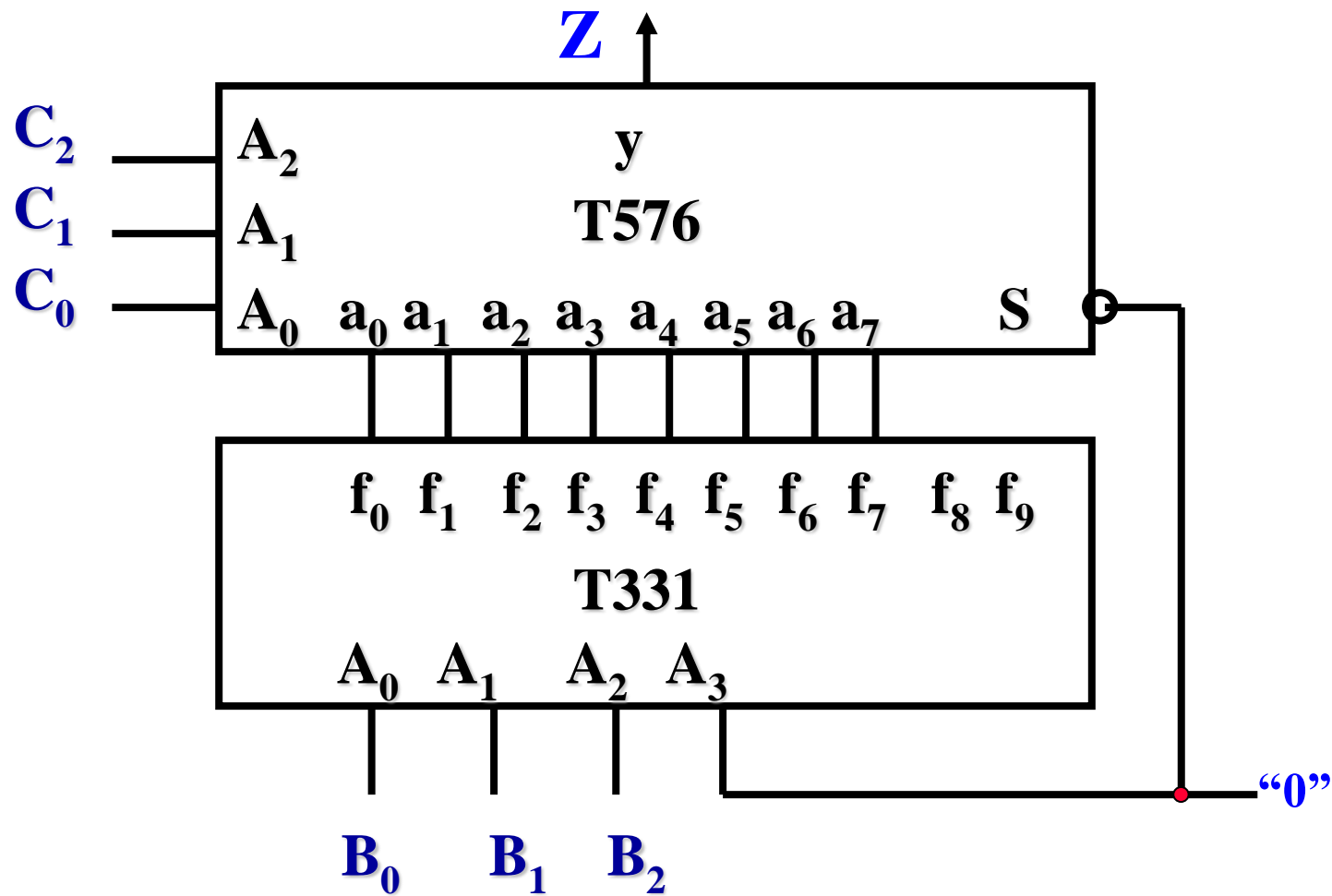
利用 **8-to -1 MUX**以及 **4-10**线译码器设计一个能实现**2组3位**数码等值比较的电路。

$A_3$	$A_2$	$A_1$	$A_0$	$f_0$	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$	$f_8$	$f_9$
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

8-to -1 MUX

$S$	$A_2$	$A_1$	$A_0$	$y$
1	×	×	×	0
0	0	0	0	$a_0$
0	0	0	1	$a_1$
0	0	1	0	$a_2$
0	0	1	1	$a_3$
0	1	0	0	$a_4$
0	1	0	1	$a_5$
0	1	1	0	$a_6$
0	1	1	1	$a_7$

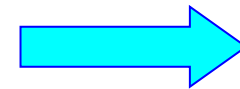




if:  $B_2B_1B_0 = 110$ , then  $f_6 = a_6 = 0$

if:  $C_2C_1C_0 = 110$ , then  $y = a_6 = 0$

if:  $C_2C_1C_0 = 111$ , then  $y = a_7 = 1$



If  $B_2B_1B_0 = C_2C_1C_0$ ,  
Then  $Z=0$

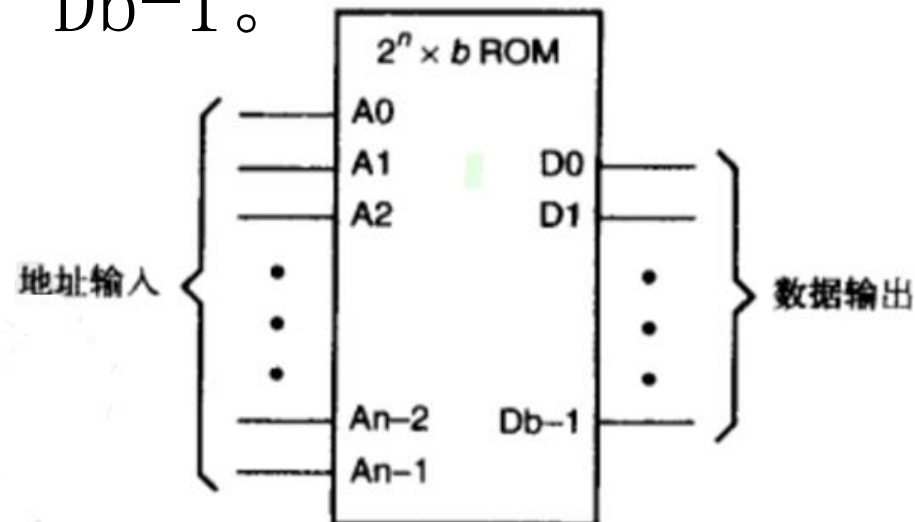
# Unit 7 组合逻辑元件

---

- 多路复用器(multiplexers)
- 三态器件(Three-state Buffer)
- 译码器(Decoders)
- 编码器(Encoders)
- 奇偶校验器
- 比较器
- 利用MSI设计组合逻辑电路
- 只读存储器(ROM)

# ROM (Read-Only Memory)

- ROM是一种具有 $n$ 个输入 $b$ 个输出的组合逻辑电路。
- 输入被称为地址输入 (address input)，通常命名为  $A_0, A_1, \dots, A_{n-1}$ 。
- 输出被称为数据输出 (data output)，通常命名为  $D_0, D_1, \dots, D_{b-1}$ 。



一个  $2^n \times b$  ROM 的基本结构

# ROM和真值表

- ROM “存储” 了一个n输入、b输出的组合逻辑功能的真值表。
- 一个3输入、4输出的组合功能的真值表，可以被存储在一个 $2^3 * 4$  (8 \* 4) 的只读存储器中。忽略延迟，ROM的数据输出总是等于真值表中由地址输入所选择的那行输出位。

输入			输出			
A2	A1	A0	D3	D2	D1	D0
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

一个3输入4输出组合逻辑函数的真值表  
(具有输出极性控制的2-4译码器)

# FPGA (Field Programmable Gate Array )

---

- 即现场可编程门阵列
- FPGA能完成任何数字逻辑功能，上至高性能计算，下至简单的74系列电路，也常用于ASIC流片前的原型验证。



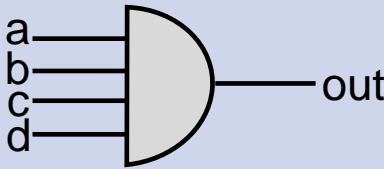
嵌入式硬件算法加速



协处理器

# FPGA中的查找表 (LUT)

- 例：使用LUT实现一个4与门电路逻辑功能

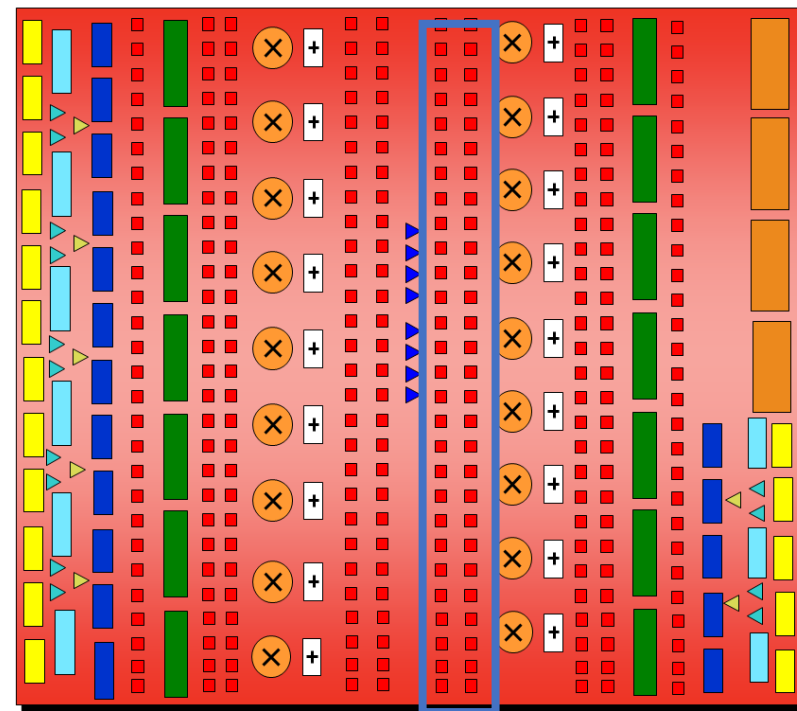
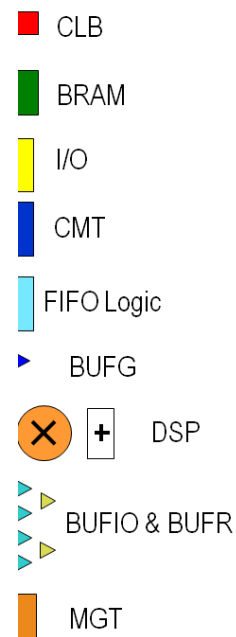
实际逻辑电路		LUT的实现方式	
			
a、b、c、d	逻辑输出	地址	RAM中存储的内容
0000	0	0000	0
0001	0	0001	0
.....	0	.....	0
1111	1	1111	1

LUT本质就是RAM，主流的FPGA是5输入或6输入LUT

A,B,C,D由FPGA芯片的管脚输入后进入可编程连线，然后作为地址线连到LUT，LUT中已经事先写入了所有可能的逻辑结果，通过地址查找到相应的数据然后输出，这样组合逻辑就实现了。

# FPGA内部结构

- 内部资源分类:
- 逻辑资源: CLB 、块存储 ( block ram) 、DSP等;
- 连接资源: 可编程互联线 (PI)、输入输出块 (IOB) 等;
- 其他资源: 全局时钟网络、时钟管理模块等
- 高端FPGA还会集成ARM核、PCIE核等。
- 资源分布采用ASMBL架构, 相同资源以列方式排布。



# Unit 7 组合逻辑元件

---

Verilog

- 
- 多路复用器(multiplexers)
  - 三态器件(Three-state Buffer)
  - 译码器(Decoders)
  - 编码器(Encoders)
  - 奇偶校验器
  - 比较器
  - 只读存储器(ROM)



## 2输入8位多路复用器的数据流型Verilog模块

---

- 多路复用器在**数据流形式**中，可以使用一系列条件操作符(?:)来提供所要求的功能。

```
module Vrmux2in8b_d(EN_L, S, D0, D1, Y);  
    input EN_L, S;  
    input [1:8] D0, D1;  
    output [1:8] Y;  
  
    assign Y = (~EN_L == 1'b0) ? 8'b0 : (  
        (S == 1'd0) ? D0 : (  
            (S == 1'd1) ? D1 : 8'bx));  
endmodule
```

# 采用多重if语句实现多路复用器

---

- 多路复用器的行为化描述。
- 采用一系列多重if语句，一个if语句对应一个选择输入值

```
module Vrmux2in8b_b(EN_L, S, D0, D1, Y);  
    input EN_L, S;  
    input [1:8] D0, D1;  
    output reg [1:8] Y;  
  
    always @ (*) begin  
        if (~EN_L == 1'b0) Y = 8'b0;  
        else if (S == 1'b0) Y = D0;  
        else if (S == 1'b1) Y = D1;  
        else Y = 8'bx;  
    end  
endmodule
```

# 采用**case**语句的4输入8位多路复用器

---

- 一个选择输入值对应一个**case**语句

- 更易读也更好维护。

```
module Vrmux4in8b(EN_L, S, A, B, C, D, Y);  
    input EN_L;  
    input [1:0] S;  
    input [1:8] A, B, C, D;  
    output reg [1:8] Y;
```

```
    always @ (EN_L or S or A or B or C or D) begin  
        if (~EN_L == 1'b0) Y = 8'b0;  
        else case (S)  
            2'd0: Y = A;  
            2'd1: Y = B;  
            2'd2: Y = C;  
            2'd3: Y = D;  
            default: Y = 8'bx;  
        endcase  
    end  
endmodule
```

# 用Verilog实现三态输出

- Verilog为高阻态内设了位数据值“z”，所以它很容易指定三态输出

```
module Vr74x541(G1_L, G2_L, A, Y);  
    input G1_L, G2_L;  
    input [1:8] A;  
    output [1:8] Y;  
  
    assign Y = (~G1_L & ~G2_L) ? A : 8'bz;  
endmodule
```

与74\*541类似的8位三态驱动器的Verilog模块，只是把三态端口用作输出

```
module Vr74x245(G_L, DIR, A, B);  
    input G_L, DIR;  
    inout [1:8] A, B;           //输出端口可作输入  
  
    assign A = (~G_L & ~DIR) ? B : 8'bz;  
    assign B = (~G_L & DIR) ? A : 8'bz;  
endmodule
```

与74\*245类似的8位三态驱动器的Verilog模块

# 用Verilog实现三态输出

inout端口的另一个应用是4路8位总线收发器

```
module VrXcvr4x8(A, B, C, D, S, AOE_L, BOE_L, COE_L, DOE_L, MOE_L);
    input [2:0] S;
    input AOE_L, BOE_L, COE_L, DOE_L, MOE_L;
    inout [1:8] A, B, C, D;
    reg [1:8] ibus;

    always @ (A or B or C or D or S) begin
        if (S[2] == 0) ibus = {4{S[1:0]}};
        else case (S[1:0])
            2'b00: ibus = A;
            2'b01: ibus = B;
            2'b10: ibus = C;
            2'b11: ibus = D;
        endcase
    end

    assign A = ((~AOE_L & ~MOE_L) && (S[2:0] != 3'b100)) ? ibus : 8'bz;
    assign B = ((~BOE_L & ~MOE_L) && (S[2:0] != 3'b101)) ? ibus : 8'bz;
    assign C = ((~COE_L & ~MOE_L) && (S[2:0] != 3'b110)) ? ibus : 8'bz;
    assign D = ((~DOE_L & ~MOE_L) && (S[2:0] != 3'b111)) ? ibus : 8'bz;
endmodule
```

4路8位总线收发器的Verilog模块

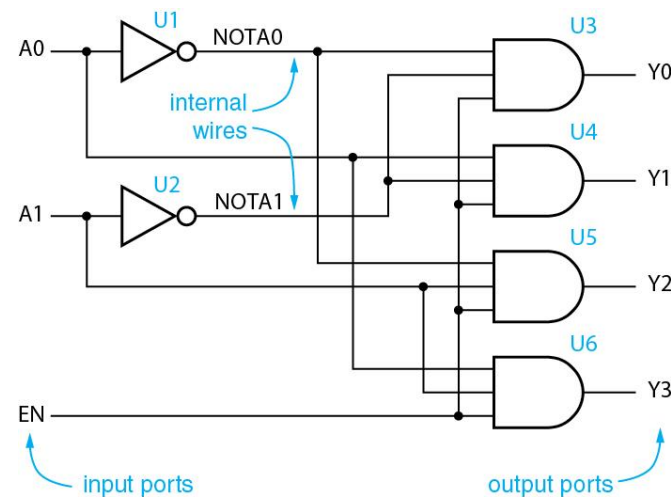
# 用Verilog实现2-4译码器——1

- Verilog设计译码器方法有很多
- 最原始的方法是编写等效于译码器逻辑电路的结构性程序。

```
module Vr2to4dec_s(A0, A1, EN, Y0, Y1, Y2, Y3);  
  input A0, A1, EN;  
  output Y0, Y1, Y2, Y3;  
  wire NOTA0, NOTA1;
```

```
  not U1 (NOTA0, A0);  
  not U2 (NOTA1, A1);  
  and U3 (Y0, NOTA0, NOTA1, EN);  
  and U4 (Y1, A0, NOTA1, EN);  
  and U5 (Y2, NOTA0, A1, EN);  
  and U6 (Y3, A0, A1, EN);  
endmodule
```

缺点：不易于理解和维护



Inputs			Outputs			
EN	A1	A0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

# 用Verilog实现2-4译码器——2

- 使用Verilog的行为化描述。
- 采用了一个**always**语句，其敏感信号列表包括译码器所有输入。
- 将输出变量声明为**reg**类型，故而在过程块中对输出变量赋值。
- 用一个**if**语句来测试使能输入。
  - 如果**EN**为0，所有输出都设置为0；
  - 当**EN**有效时，译码器的功能。

```
module Vr2to4dec_b1(A0, A1, EN, Y0, Y1, Y2, Y3);  
    input A0, A1, EN;  
    output reg Y0, Y1, Y2, Y3;  
  
    always @ (A0, A1, EN)  
        if (EN==0) {Y3,Y2,Y1,Y0} = 4'b0000;  
        else  
            case ({A1,A0})  
                2'b00: {Y3,Y2,Y1,Y0} = 4'b0001;  
                2'b01: {Y3,Y2,Y1,Y0} = 4'b0010;  
                2'b10: {Y3,Y2,Y1,Y0} = 4'b0100;  
                2'b11: {Y3,Y2,Y1,Y0} = 4'b1000;  
                default: {Y3,Y2,Y1,Y0} = 4'b0000;  
            endcase  
    endmodule
```

# 用Verilog实现2-4译码器——3

---

- 使用2-4二进制译码器的行为化风格Verilog模块

- 能够能更好地捕捉到译码器的行为特性。

```
module Vr2to4dec_b2(A0, A1, EN, Y0, Y1, Y2, Y3);  
  input A0, A1, EN;  
  output reg Y0, Y1, Y2, Y3;  
  reg [3:0] IY;  
  integer i;  
  always @ (A0 or A1 or EN) begin  
    IY = 4'b0000;      // Default outputs all 0  
    if (EN==1)         // If enabled...  
      for (i=0; i<=3; i=i+1) // set out bit i where i equals {A1,A0}  
        if (i == {A1,A0}) IY[i] = 1;  
    {Y3,Y2,Y1,Y0} = IY; // Copy internal bit-vector variable to outputs  
  end  
endmodule
```



# 用Verilog实现2-4译码器——4

---

- 最简练的行为化模型: 在将输出位初始化为全0之后, 其只需要索引 {A1, A0} 将IY为设置为1

```
module Vr2to4dec_b3(A0, A1, EN, Y0, Y1, Y2, Y3);
```

```
    input A0, A1, EN;
```

```
    output reg Y0, Y1, Y2, Y3;
```

```
    reg [3:0] IY;
```

```
    always @ (*) begin
```

```
        IY = 4'b0000;           // Default outputs all 0
```

```
        if (EN==1) IY[{A1,A0}] = 1; // Set selected output bit if enabled
```

```
        {Y3,Y2,Y1,Y0} = IY;      // Copy internal variable to output
```

```
    end
```

```
endmodule
```

## 2-4译码器的测试平台

- 设计一个测试平台来确保设计的正确性。
- 对于只有三个输入的译码器，就有八个不同的输入组合，这个测试平台采用一个变量*i*来逐一检查这些组合。

```
`timescale 1 ns / 100 ps
module Vr2to4dec_tb () ;
    reg A0s, A1s, ENs;
    wire Y0s, Y1s, Y2s, Y3s;
    integer i, errors;
    reg [3:0] expectY;

    Vr2to4dec_d UUT ( .A0(A0s),.A1(A1s),.EN(ENs), // Instantiate unit under test (UUT)
                      .Y0(Y0s),.Y1(Y1s),.Y2(Y2s),.Y3(Y3s) );

    initial begin
        errors = 0;
        for (i=0; i<=7; i=i+1) begin
            {ENs, A1s, A0s} = i;                // Apply test input combination
            #10 ;
            expectY = 4'b0000;                  // Expect no outputs asserted if EN = 0
            if (ENs==1) expectY[{A1s,A0s}] = 1'b1; // Else output {A1,A0} should be asserted
            if ({Y3s,Y2s,Y1s,Y0s} !== expectY) begin
                $display("Error: EN A1A0 = %b %b%b, Y3Y2Y1Y0 = %b%b%b%b",
                        ENs, A1s, A0s, Y3s, Y2s, Y1s, Y0s);
                errors = errors + 1;
            end
        end
        $display("Test complete, %d errors",errors);
    end
endmodule
```

# 用Verilog实现七段译码器

```
module Vr7segdec(DIG, EN, SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG);
  input [3:0] DIG;    input EN;
  output reg SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG;
  reg [1:7] SEGS;
  always @ (DIG or EN) begin
    if (EN)
      case (DIG)      // Segment patterns  abcdefg
        0: SEGS = 7'b1111110; // 0
        1: SEGS = 7'b0110000; // 1
        2: SEGS = 7'b1101101; // 2
        3: SEGS = 7'b1111001; // 3
        4: SEGS = 7'b0110011; // 4
        5: SEGS = 7'b1011011; // 5
        6: SEGS = 7'b0011111; // 6 (no 'tail') //    6: SEGS = 7'b1011111; // 6 ('tail' included)
        7: SEGS = 7'b1110000; // 7
        8: SEGS = 7'b1111111; // 8
        9: SEGS = 7'b1110011; // 9 (no 'tail') //    9: SEGS = 7'b1111011; // 9 ('tail' included)
        // 10: SEGS = 7'b1110111; // A //    11: SEGS = 7'b0011111; // b
        // 12: SEGS = 7'b1001110; // C //    13: SEGS = 7'b0111101; // d
        // 14: SEGS = 7'b1001111; // E //    15: SEGS = 7'b1000111; // F
        default: SEGS = 7'bxxxxxxx;
      endcase
    else SEGS = 7'b0000000;
    {SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG} = SEGS;
  end
endmodule
```

# 七段译码器的Verilog测试平台

---

```
`timescale 1ns / 100ps
module Vr7seg_tb ();
    reg EN;
    reg [3:0] DIG;
    wire SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG;
    integer i;

    Vr7segE UUT (.DIG(DIG), .EN(EN), .SEGA(SEGA), .SEGB(SEGB), .SEGC(SEGC),
                .SEGD(SEGD), .SEGE(SEGE), .SEGF(SEGF), .SEGG(SEGG) );

    initial
    begin
        EN = 1; // Enable all
        for (i=0; i<16; i=i+1)
            begin
                DIG = i;
                #5
                $write("Iteration %0d\n", i);
                if (SEGA) $write(" __\n"); else $write("\n");
                if (SEGF) $write("|"); else $write(" ");
                if (SEGG) $write(" __"); else $write(" ");
                if (SEGB) $write("|\n"); else $write("\n");
                if (SEGE) $write("|"); else $write(" ");
                if (SEGD) $write(" __"); else $write(" ");
                if (SEGC) $write("|\n"); else $write("\n");
                #5 ;
            end
        $write("Done\n");
    end
endmodule
```

# 用Verilog实现优先编码器

---

用Verilog对优先编码器的行为建模，可以有多种方式。一种方式是采用多重Verilog if语句序列

```
module Vr8inprior2(I, A, IDLE);
    input [7:0] I;
    output reg [2:0] A;
    output reg IDLE;
    always @ (I) begin
        IDLE = 0;
        if (I[7]) A = 3'd7;
        else if (I[6]) A = 3'd6;
        else if (I[5]) A = 3'd5;
        else if (I[4]) A = 3'd4;
        else if (I[3]) A = 3'd3;
        else if (I[2]) A = 3'd2;
        else if (I[1]) A = 3'd1;
        else if (I[0]) A = 3'd0;
        else begin A = 3'd0; IDLE = 1;
        end
    end
end
endmodule
```

# 采用for循环的8输入实现优先编码器

---

```
module Vr8inprior3(I, A, IDLE);  
    input [7:0] I;  
    output reg [2:0] A;  
    output reg IDLE;  
    integer j;  
  
    always @ (I) begin  
        IDLE = 1; A = 0;    // default output values  
        for (j=0; j<=7; j=j+1) // check low priority first  
            if (I[j]==1) begin IDLE = 0; A = j; end  
    end  
endmodule
```

# 采用case语句的8输入优先编码器

---

```
module Vr8inprior4(I, A, IDLE);
    input [7:0] I;
    output reg [2:0] A;
    output reg IDLE;
    always @ (I) begin
        IDLE = 1; A = 0;      // default output values
        case(1'b1)
            I[7]: begin IDLE = 0; A = 7; end // Highest priority (as first
            I[6]: begin IDLE = 0; A = 6; end // choice in case statement)
            I[5]: begin IDLE = 0; A = 5; end
            I[4]: begin IDLE = 0; A = 4; end
            I[3]: begin IDLE = 0; A = 3; end
            I[2]: begin IDLE = 0; A = 2; end
            I[1]: begin IDLE = 0; A = 1; end
            I[0]: begin IDLE = 0; A = 0; end
        endcase
    end
endmodule
```

# 8输入优先编码器模块的测试平台

---

```
`timescale 1ns / 100ps
module Vr8inprior_tb();
    reg [7:0] I;
    wire [2:0] A;
    wire IDLE;
    integer ii, errors;
    Vr8inprior6 UUT ( .I(I), .A(A), .IDLE(IDLE) );
    initial begin
        errors = 0;
        for (ii=0; ii<256; ii=ii+1) begin
            I = ii;
            #10 ;
            if (
                // Identify all error cases
                ( (I==8'b0) && (IDLE!=1'b1) ) // Should be idle
                || ( (I>8'b0) && (IDLE===1'b1) ) // Should not be idle
                || ( (I>8'b0) && (I<2**A) ) // I should be at least 2**A
                || ( (I>8'b0) && (I>=2**(A+1)) ) // but less than 2**(A+1)
            )
            begin
                errors = errors+1;
                $display("Error: I=%b, A=%b, IDLE=%b",I,A,IDLE);
            end
        end
        $display("Test done, %d errors\n",errors);
        $stop(1);
    end
endmodule
```



# 用Verilog实现异或门和奇偶校验电路

---

```
module Vrxor3(A, B, C, Y);  
  input A, B, C;  
  output Y;  
  
  assign Y = A ^ B ^ C;  
endmodule
```

一个3输入异或器件的数据流型Verilog模块

```
module Vrparity9(I, ODD);  
  input [1:9] I;  
  output reg ODD;  
  integer j;  
  
  always @ (I) begin  
    ODD = 1'b0;  
    for (j = 1; j <= 9; j = j+1)  
      if (I[j]) ODD = ~ODD;  
  end  
endmodule
```

9输入奇偶校验电路的行为化Verilog模块

# 用Verilog实现9输入奇偶校验电路

---

```
module Vrparity9s(I, EVEN, ODD);  
    input [1:9] I;  
    output EVEN, ODD;  
    wire Y1, Y2, Y3, Y3N;  
  
    Vrxor3 U1 (I[1], I[2], I[3], Y1);  
    Vrxor3 U2 (I[4], I[5], I[6], Y2);  
    Vrxor3 U3 (I[7], I[8], I[9], Y3);  
    assign Y3N = ~Y3;  
    Vrxor3 U4 (Y1, Y2, Y3, ODD);  
    Vrxor3 U5 (Y1, Y2, Y3N, EVEN);  
endmodule
```

# 用Verilog实现比较器——1

---

8位数值比较器的行为化Verilog模块的第一次尝试。

问题：按照惯例，编译器会“推理出一个锁存器”，用来保留原先的条件输出值，所以即使不用存储原先的值，推理出的锁存器也还是会增加最后综合出来的电路规模和延迟。

```
module Vr8bitcmp_xi(P, Q, PGTQ, PEQQ, PLTQ);  
    input [7:0] P, Q;  
    output reg PGTQ, PEQQ, PLTQ;  
  
    always @ (P or Q)  
        if (P == Q)  
            begin PGTQ = 1'b0; PEQQ = 1'b1; PLTQ = 1'b0; end  
        else if (P > Q)  
            begin PGTQ = 1'b1; PEQQ = 1'b0; PLTQ = 1'b0; end  
        else if (P < Q)  
            begin PGTQ = 1'b0; PEQQ = 1'b0; PLTQ = 1'b1; end  
endmodule
```

# 用Verilog实现比较器——2

8位数值比较器的行为化Verilog模块的第二次尝试。确保所有情况都有一个值赋给输出。虽然最后的else语句决不会被用到，但可以避免锁存器的问题。

问题：三个比较结果是互斥的但并没有体现出来，需要更多有效的芯片资源

```
module Vr8bitcmp_xc(P, Q, PGTQ, PEQQ, PLTQ);
  input [7:0] P, Q;
  output reg PGTQ, PEQQ, PLTQ;

  always @ (P or Q)
    if (P == Q)
      begin PGTQ = 1'b0; PEQQ = 1'b1; PLTQ = 1'b0; end
    else if (P > Q)
      begin PGTQ = 1'b1; PEQQ = 1'b0; PLTQ = 1'b0; end
    else if (P < Q)
      begin PGTQ = 1'b0; PEQQ = 1'b0; PLTQ = 1'b1; end
    else
      begin PGTQ = 1'bx; PEQQ = 1'bx; PLTQ = 1'bx; end
endmodule
```

# 用Verilog实现比较器——3

- 纠正过的8位数值比较器的行为化Verilog模块，采用数据流风格。

```
module Vr8bitcmp(P, Q, PGTQ, PEQQ, PLTQ);  
    input [7:0] P, Q;  
    output reg PGTQ, PEQQ, PLTQ;  
  
    always @ (P or Q)  
        if (P == Q)  
            begin PGTQ = 1'b0; PEQQ = 1'b1; PLTQ = 1'b0; end  
        else if (P > Q)  
            begin PGTQ = 1'b1; PEQQ = 1'b0; PLTQ = 1'b0; end  
        else  
            begin PGTQ = 1'b0; PEQQ = 1'b0; PLTQ = 1'b1; end  
    endmodule
```

# 64位数值比较器

采用9个8位数值比较器构成的64位数据比较器的层次型结构化Verilog模块

```
module Vr64bitcmp_sh(P, Q, PGTQ, PEQQ, PLTQ);  
    input [63:0] P, Q;  
    output PGTQ, PEQQ, PLTQ;  
    wire [7:0] GT, EQ, LT;  
  
    Vr8bitcmp_kh U1(P[7:0], Q[7:0], GT[0], EQ[0], LT[0]);  
    Vr8bitcmp_kh U2(P[15:8], Q[15:8], GT[1], EQ[1], LT[1]);  
    Vr8bitcmp_kh U3(P[23:16], Q[23:16], GT[2], EQ[2], LT[2]);  
    Vr8bitcmp_kh U4(P[31:24], Q[31:24], GT[3], EQ[3], LT[3]);  
    Vr8bitcmp_kh U5(P[39:32], Q[39:32], GT[4], EQ[4], LT[4]);  
    Vr8bitcmp_kh U6(P[47:40], Q[47:40], GT[5], EQ[5], LT[5]);  
    Vr8bitcmp_kh U7(P[55:48], Q[55:48], GT[6], EQ[6], LT[6]);  
    Vr8bitcmp_kh U8(P[63:56], Q[63:56], GT[7], EQ[7], LT[7]);  
    Vr8bitcmp_kh U9(GT, LT, PGTQ, PEQQ, PLTQ);  
endmodule
```

# N位比较器的测试平台

---

```
`timescale 1 ns / 100 ps
module VrNbitcmp_tb();
    parameter N = 64;    // Input width of comparator UUT
    parameter SEED = 1; // Set a different pseudorandom seed here if desired
    reg [N-1:0] P, Q;
    wire PGTQ, PEQQ, PLTQ;
    integer ii, errors;

    Vr64bitcmp_sh UUT ( .P(P), .Q(Q), .PGTQ(PGTQ), .PEQQ(PEQQ), .PLTQ(PLTQ) );

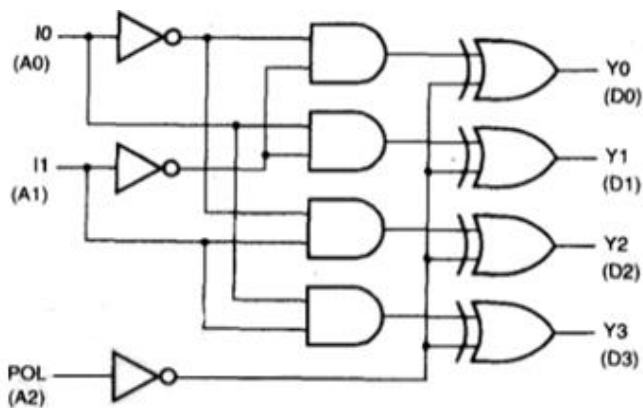
    initial begin
        errors = 0;
        P = $random(SEED); // Set pattern based on seed parameter
        for (ii=0; ii<10000; ii=ii+1) begin
            P = $random; Q = $random;
            #10 ;
            if ( (PGTQ) !== (P>Q) ||
                (PLTQ) !== (P<Q) ||
                (PEQQ) !== (P==Q) ) begin
                errors = errors + 1;
                $display("P=%b(%0d), Q=%b(%0d), PGTQ=%b, PEQQ=%b, PLTQ=%b", P, P, Q, Q, PGTQ, PEQQ, PLTQ);
            end;
        end
        $display("Test done, %0d errors", errors);
    end
endmodule
```

# Backup

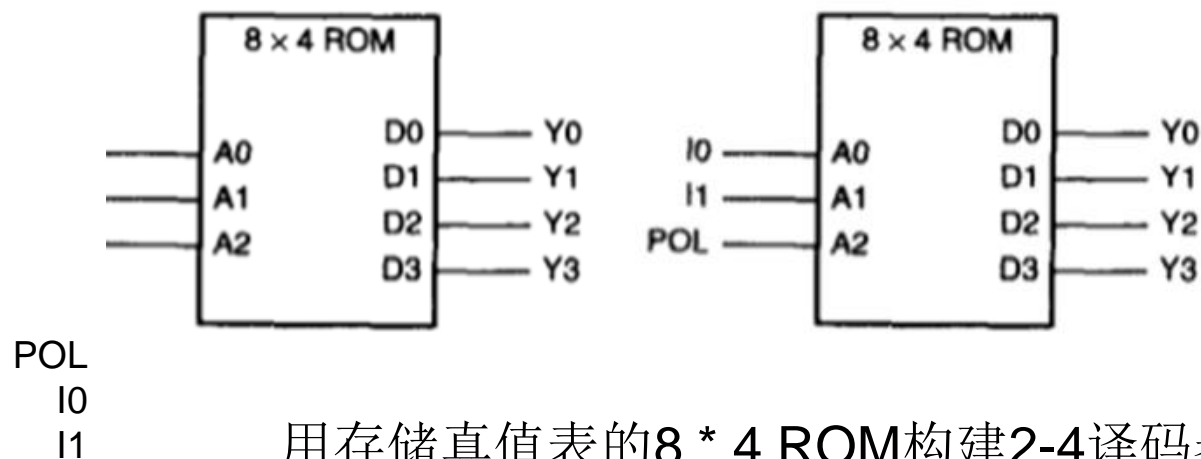


# 用ROM实现组合逻辑函数

- 两种不同的方式来构建译码器：
  - 使用分立的门
  - 用包含真值表的8 \* 4 ROM
- 使用ROM的物理实现并不是唯一的。



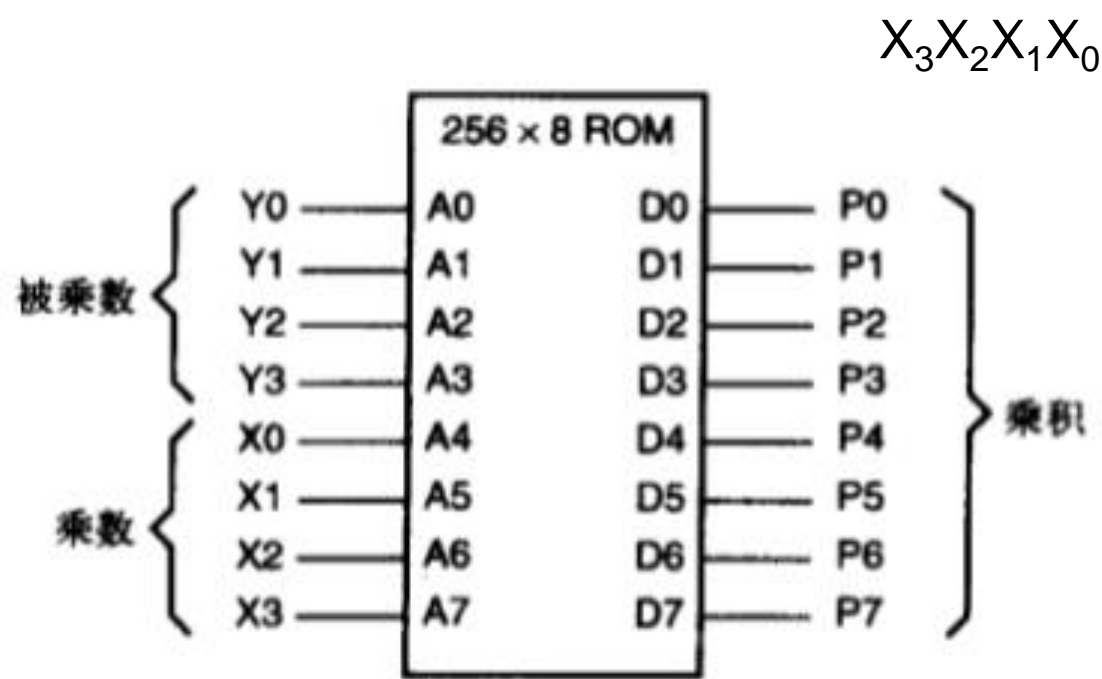
具有输出极性控制的2-4译码器



用存储真值表的8 \* 4 ROM构建2-4译码器

# 用ROM实现4\*4无符号二进制数乘法

- 多少种组合？
- 乘积最多为几位？



地址	$Y_3Y_2Y_1Y_0$															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
10	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
20	00	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
30	00	03	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
40	00	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
50	00	05	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
60	00	06	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
70	00	07	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
80	00	08	10	18	20	28	30	38	40	48	50	58	60	68	70	78
90	00	09	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A0	00	0A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B0	00	0B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C0	00	0C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D0	00	0D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E0	00	0E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F0	00	0F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	D1

# 基于ROM的设计方法的优点

- 通常可以用高级程序语言来计算存储在ROM中的内容。

```
#include <stdio.h>

/*Procedure to print d as a hex digit. */
void PrintHexDigit(int d)
{
    if (d<10) printf("%c", '0'+d);
    else printf("%c", 'A'+d-10);
}

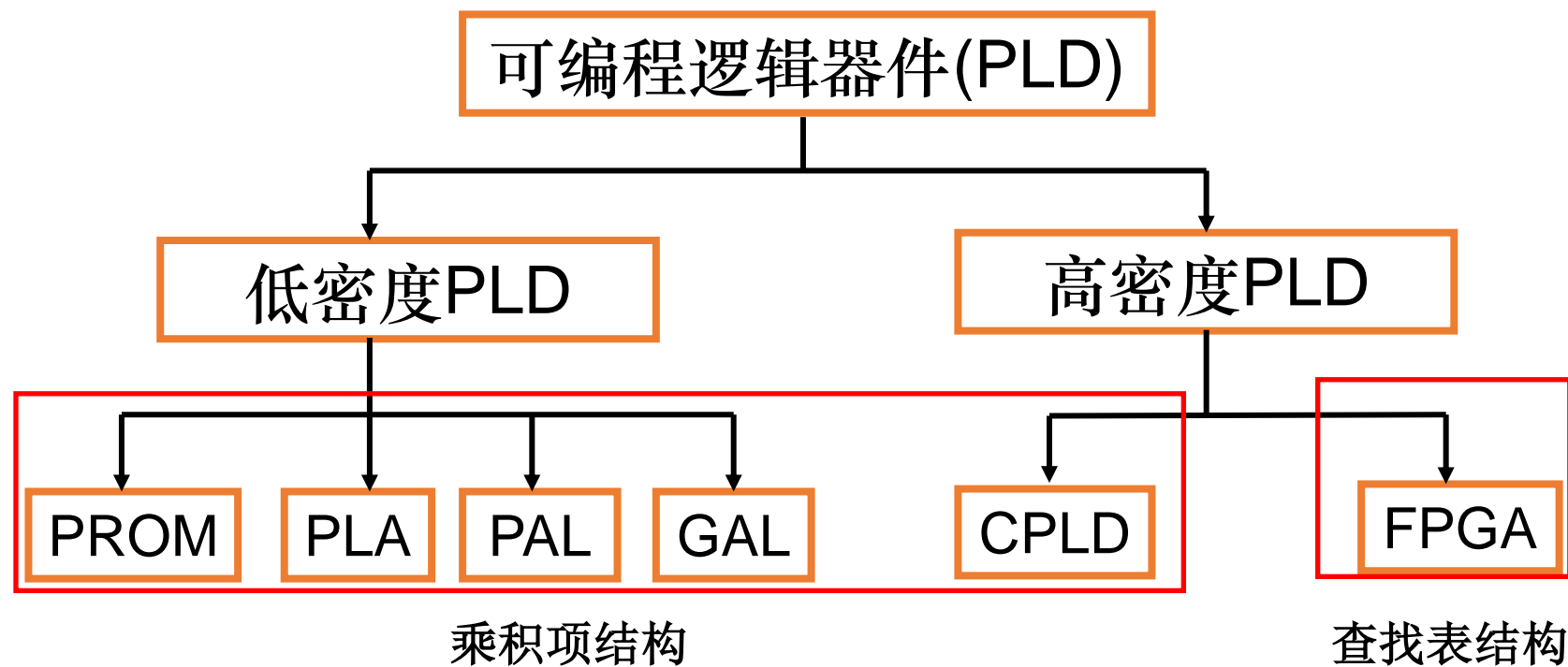
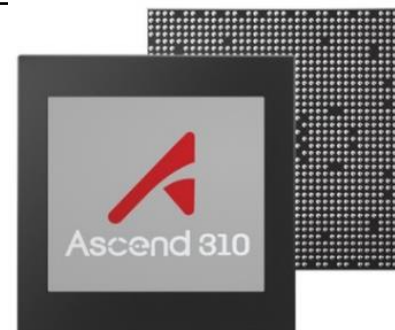
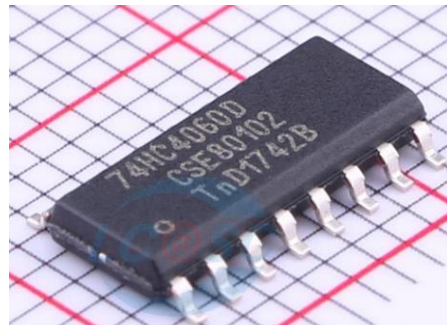
/*Procedure to print i as two hex digits. */
void PrintHex2(int i)
{
    PrintHexDigit((i / 16) % 16);
    PrintHexDigit(i % 16);
}

void main()
{
    int x, y;

    for (x=0; x<=15; x++) {
        PrintHex2(x*16); printf(":");
        for (y=0; y<=15; y++) {
            printf(" ");
            PrintHex2(x*y);
        }
        printf("\n");
    }
}
```

# 可编程逻辑器件

- 固定逻辑器件



# 乘积项结构

