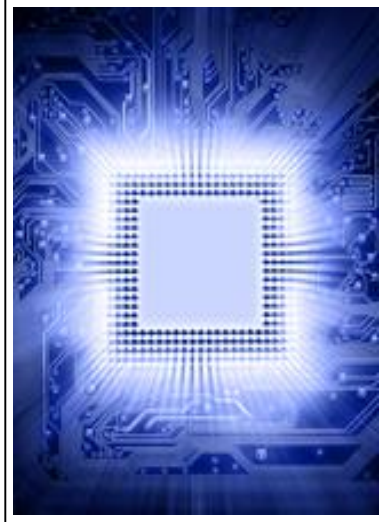
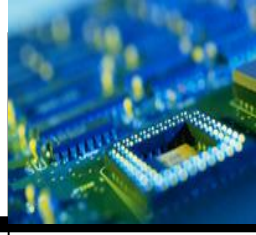
第六章 组合逻辑设计实践

武港山

南京大学人工智能学院



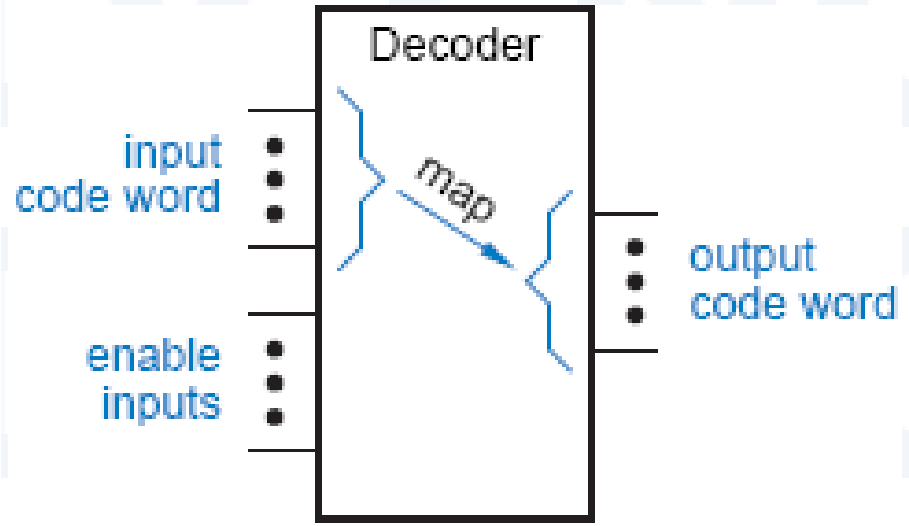
主要内容



- 设计文档编制标准
- 电路的定时
- 组合PLD的内部结构
  - 译码器
  - 编码器
  - 三态器件
  - 多路复用器
  - 校验电路
  - 比较器
  - 加法器、减法器和ALU
  - 组合乘法器

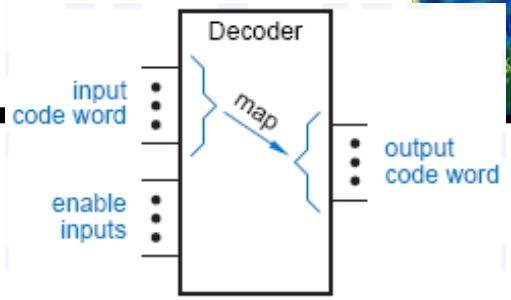
3.1 译码器 Decoder

- 多输入、多输出的逻辑电路。
  - 一般来说，输入比输出位数少，输入码字和输出码字之间一一对应。
- 一般结构：



- 最常用的输入编码是n位二进制编码。
- 最常用的输出编码是m中取1码。

3.1 译码器 Decoder

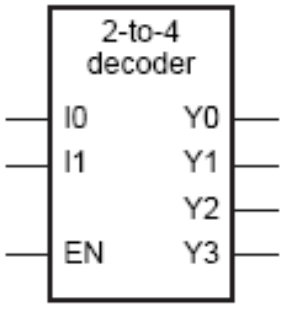


- 设计思路：
  - 真值表
    - 输入和输出是一一对应的，可以直接推导出真值表。
  - 逻辑图
    - 地址译码结构：生成所有最小项供输出使用。
  - 器件综合
    - 组件。

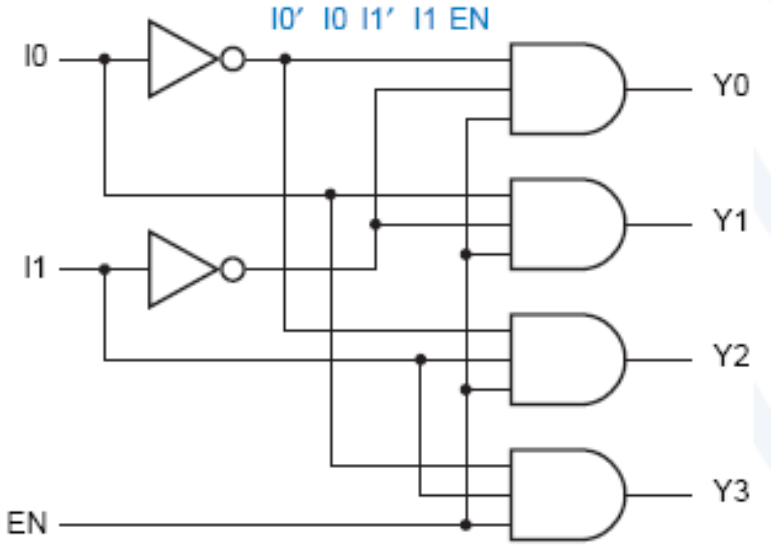
3.1 译码器 Decoder

- 二进制译码器：
  - 输入：n位二进制编码
  - 输出：2^n中取1码
- 2-4译码器：
  - 真值表
  - 逻辑图
  - 器件符号

Inputs			Outputs			
EN	I1	I0	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



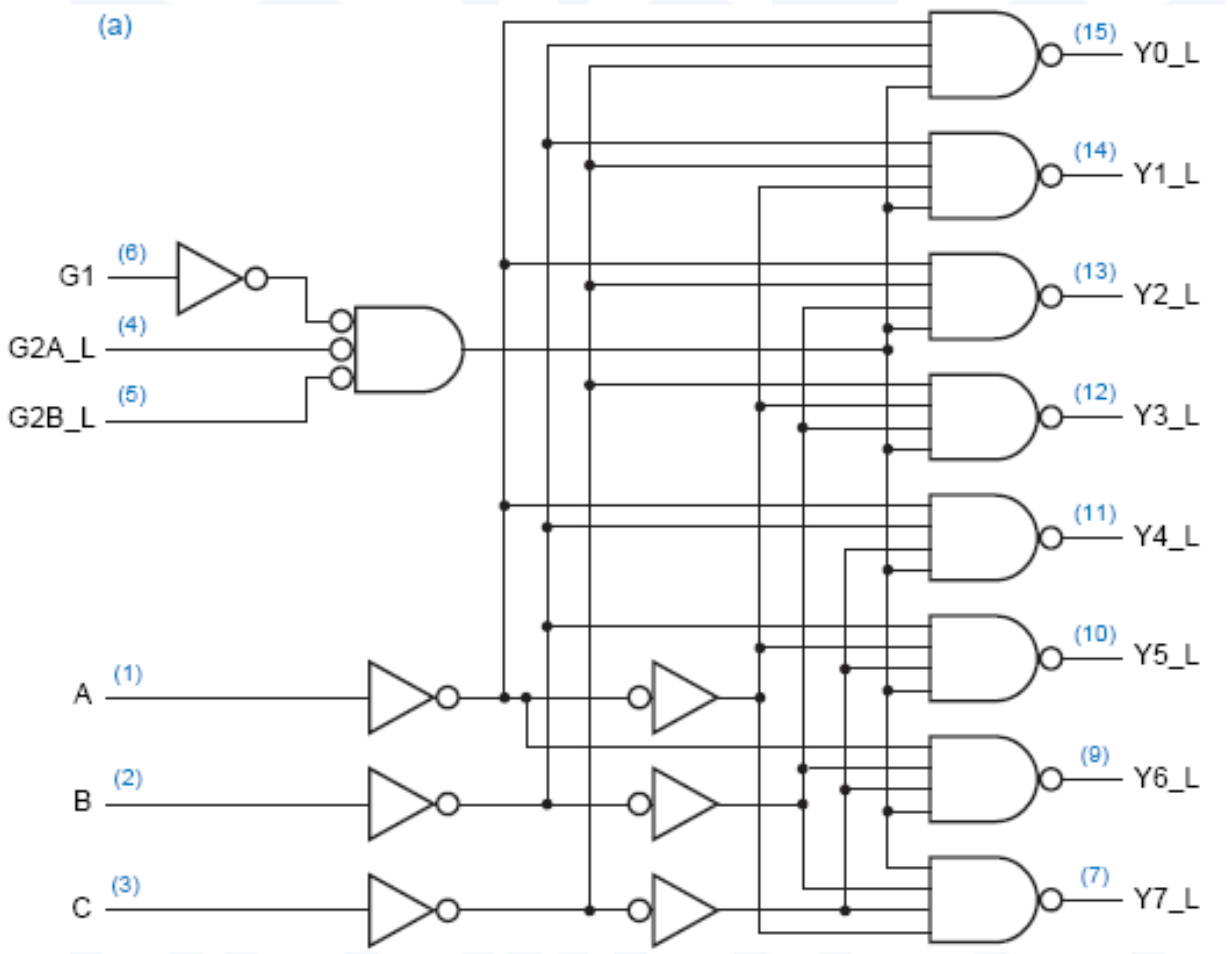
(a)



(b)

3.1 译码器 Decoder

- 3-8译码器：
  - 逻辑图
  - 真值表
  - 器件符号



3.1 译码器 Decoder

- 3-8译码器：
  - 逻辑图
  - 真值表
  - 器件符号

(b) 74x138

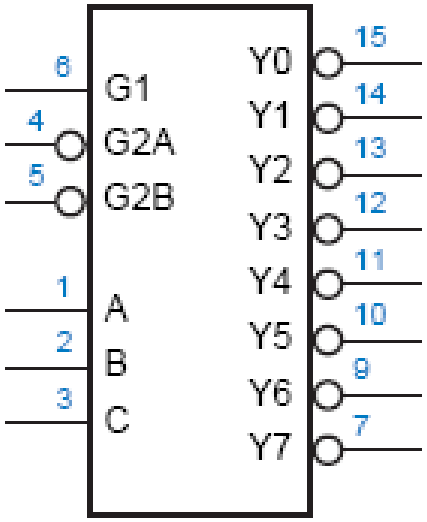


Table 5-7 Truth table for a 74x138 3-to-8 decoder.

Inputs						Outputs							
G1	G2A_L	G2B_L	C	B	A	Y7_L	Y6_L	Y5_L	Y4_L	Y3_L	Y2_L	Y1_L	Y0_L
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	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1

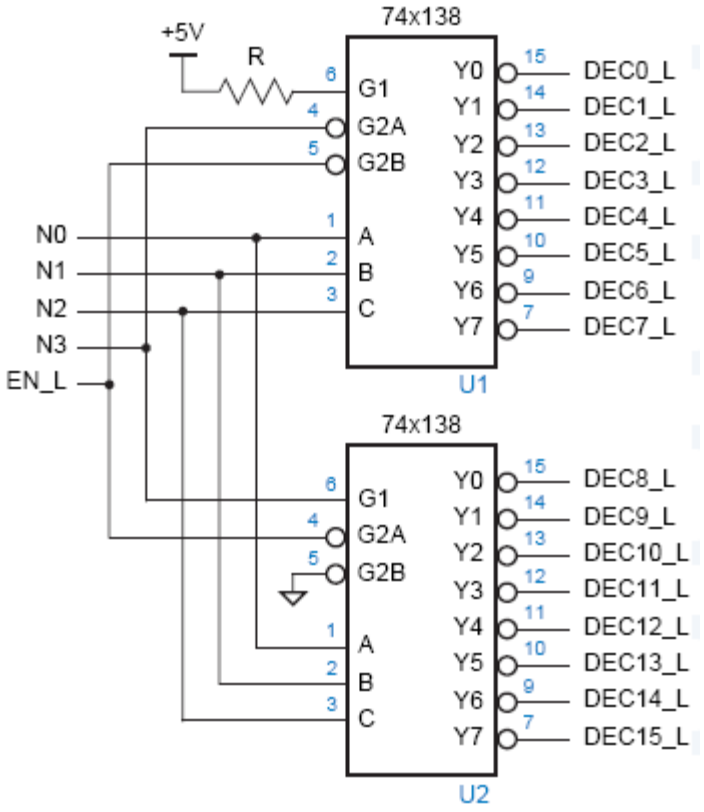
$$Y5 = \underbrace{G1 \cdot G2A \cdot G2B}_{\text{enable}} \cdot \underbrace{C \cdot B' \cdot A}_{\text{select}}$$

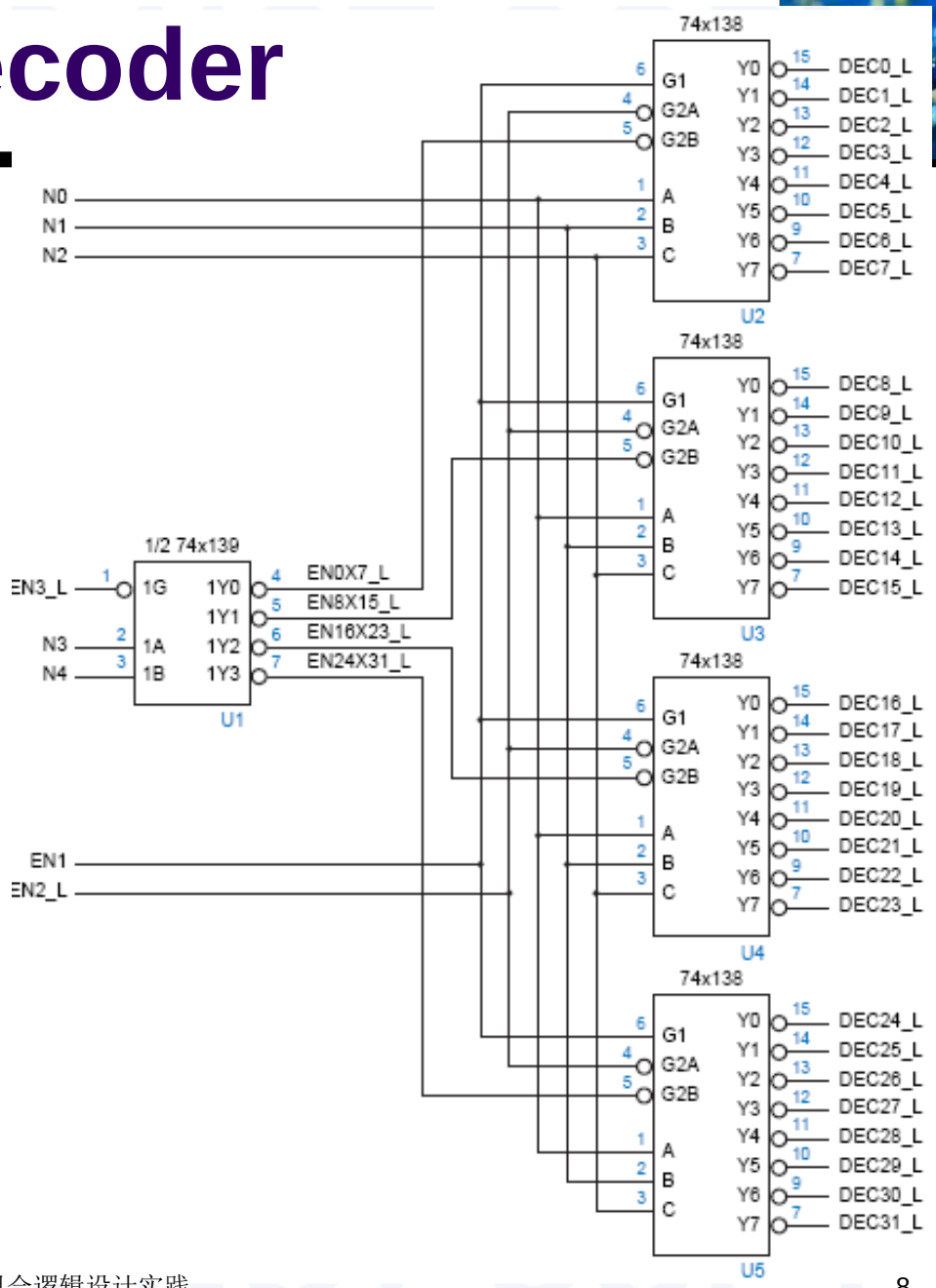
$$G2A = G2A\_L'$$
$$G2B = G2B\_L'$$
$$Y5 = Y5\_L'$$

$$Y5\_L = Y5' = (G1 \cdot G2A\_L' \cdot G2B\_L' \cdot C \cdot B' \cdot A)'$$
$$= G1' + G2A\_L + G2B\_L + C' + B + A'$$

3.1 译码器 Decoder

- 级联二进制译码器：
  - 4-16译码器
  - 5-32译码器

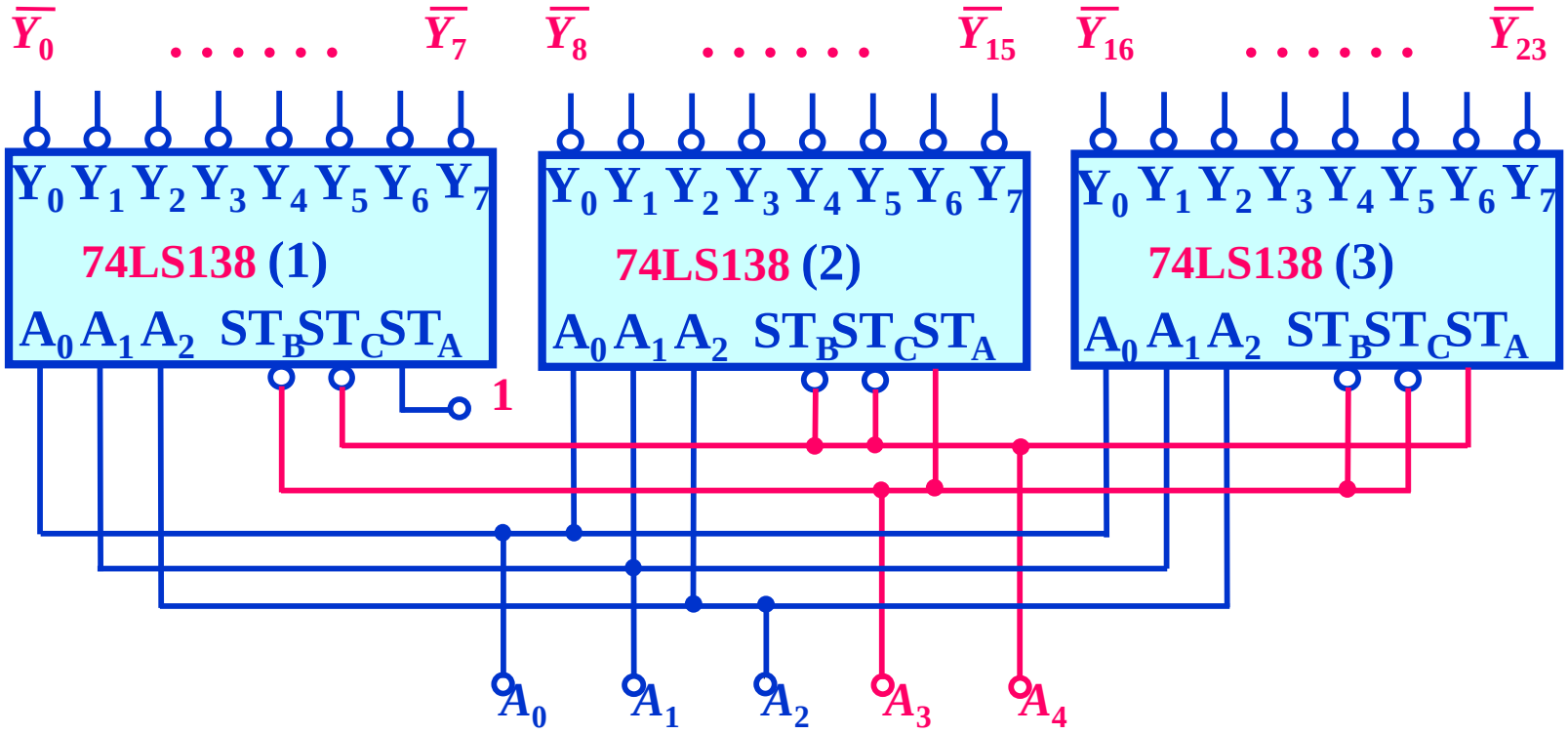




三片 3 线- 8 线

5 线 - 24 线

A_4	A_3	（1）	（2）	（3）	输 出
0	0	工	禁	禁	$\overline{Y}_0 \sim \overline{Y}_7$
0	1	禁	工	禁	$\overline{Y}_8 \sim \overline{Y}_{15}$
1	0	禁	禁	工	$\overline{Y}_{16} \sim \overline{Y}_{23}$
1	1	禁	禁	禁	全为 1



3.1 译码器 Decoder

- 用Verilog实现译码器：
  - 结构型

表6-20　图6-32译码器的结构型Verilog模块

```
module Vr2to4dec(I0, I1, EN, Y0, Y1, Y2, Y3);
  input I0, I1, EN;
  output Y0, Y1, Y2, Y3;
  wire NOTI0, NOTI1;

  INV U1 (NOTI0, I0);
  INV U2 (NOTI1, I1);
  AND3 U3 (Y0, NOTI0, NOTI1, EN);
  AND3 U4 (Y1,    I0, NOTI1, EN);
  AND3 U5 (Y2, NOTI0,    I1, EN);
  AND3 U6 (Y3,    I0,    I1, EN);
endmodule
```

3.1 译码器 Decoder

- 用Verilog实现译码器：
  - 数据流型

表6-21　类似74x138 3-8译码器的功能型Verilog模块

```
module Vr74x138a(G1, G2A_L, G2B_L, A, Y_L);
  input G1, G2A_L, G2B_L;
  input [2:0] A;
  output [0:7] Y_L;
  reg [0:7] Y_L;

  always @ (G1 or G2A_L or G2B_L or A) begin
    if (G1 & ~G2A_L & ~G2B_L)
      case (A)
        0: Y_L = 8'b01111111;
        1: Y_L = 8'b10111111;
        2: Y_L = 8'b11011111;
        3: Y_L = 8'b11101111;
        4: Y_L = 8'b11110111;
        5: Y_L = 8'b11111011;
        6: Y_L = 8'b11111101;
        7: Y_L = 8'b11111110;
        default: Y_L = 8'b11111111;
      endcase
    else Y_L = 8'b11111111;
  end
endmodule
```

3.1 译码器 Decoder

- 用Verilog实现译码器：
  - 有效电平处理

表6-22 对有效电平进行处理的可维护方法的Verilog模块

```
module Vr74x138b(G1, G2A_L, G2B_L, A, Y_L);
  input G1, G2A_L, G2B_L;
  input [2:0] A;
  output [0:7] Y_L;
  reg G2A,G2B;
  reg [0:7] Y_L, Y;

  always @ (G1 or G2A_L or G2B_L or A or Y) begin
    G2A = ~G2A_L;  // Convert inputs
    G2B = ~G2B_L;
    Y_L = ~Y;      // Convert outputs
    if (G1 & G2A & G2B)
      case (A)
        0: Y = 8'b10000000;
        1: Y = 8'b01000000;
        2: Y = 8'b00100000;
        3: Y = 8'b00010000;
        4: Y = 8'b00001000;
        5: Y = 8'b00000100;
        6: Y = 8'b00000010;
        7: Y = 8'b00000001;
        default: Y = 8'b00000000;
      endcase
    else Y = 8'b00000000;
  end
endmodule
```

3.1 译码器 Decoder

- 用Verilog实现译码器：
  - 分层定义

```
module Vr74x138c(G1, G2A_L, G2B_L, A, Y_L);
  input G1, G2A_L, G2B_L;
  input [2:0] A;
  output [0:7] Y_L;
  wire G2A,G2B;
  wire [0:7] Y;

  assign G2A = ~G2A_L;  // Convert inputs
  assign G2B = ~G2B_L;
  assign Y_L = ~Y;      // Convert outputs
  Vr3to8deca U1 (G1, G2A, G2B, A, Y);
endmodule
```

```
module Vr3to8deca(G1, G2, G3, A, Y);
  input G1, G2, G3;
  input [2:0] A;
  output [0:7] Y;
  reg [0:7] Y;

  always @ (G1 or G2 or G3 or A) begin
    if (G1 & G2 & G3)
      case (A)
        0: Y = 8'b10000000;
        1: Y = 8'b01000000;
        2: Y = 8'b00100000;
        3: Y = 8'b00010000;
        4: Y = 8'b00001000;
        5: Y = 8'b00000100;
        6: Y = 8'b00000010;
        7: Y = 8'b00000001;
        default: Y = 8'b00000000;
      endcase
    else Y = 8'b00000000;
  end
endmodule
```

3.1 译码器 Decoder

- 用Verilog实现译码器：
  - 分层定义

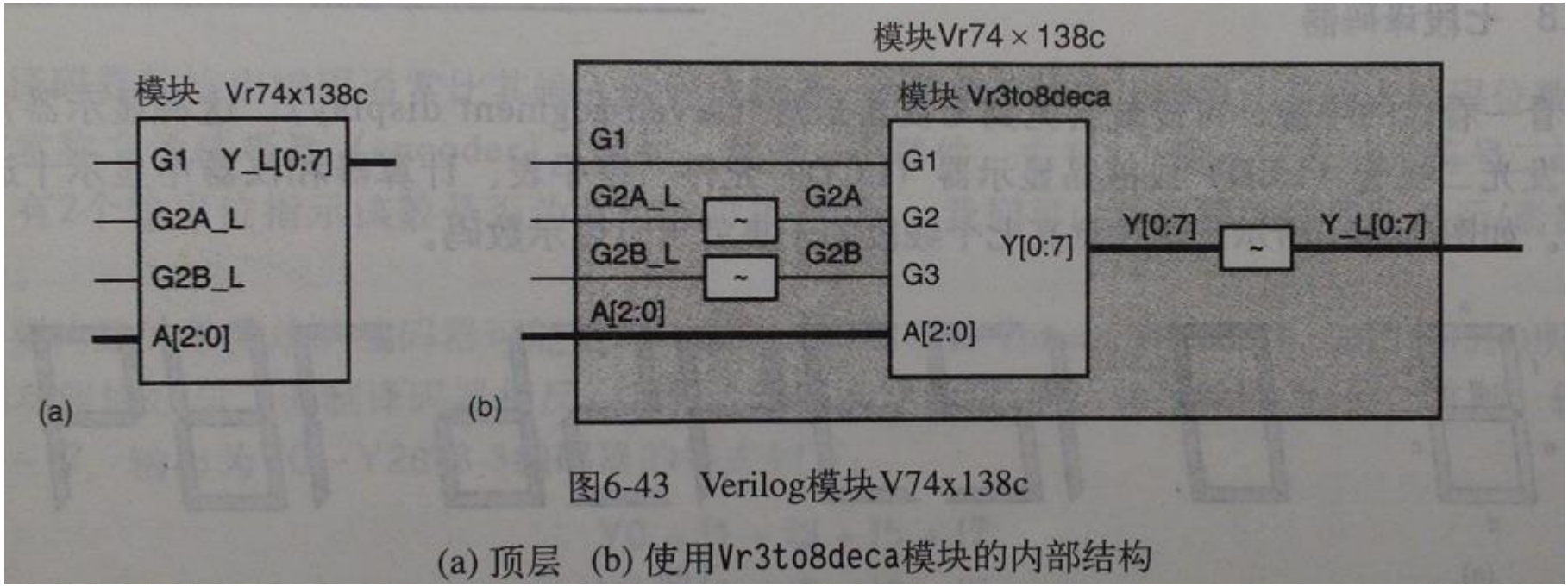


图6-43 Verilog模块V74x138c

(a) 顶层 (b) 使用Vr3to8deca模块的内部结构

3.1 译码器 Decoder

- 用Verilog实现译码器：
  - 行为描述型

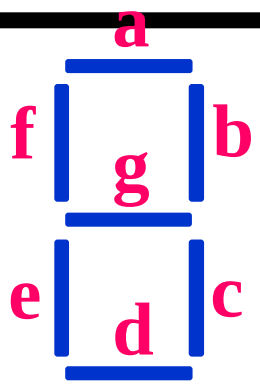
表6-25 对3-8译码器的Verilog行为定义

```
module Vr3to8decb(G1, G2, G3, A, Y);
  input G1, G2, G3;
  input [2:0] A;
  output [0:7] Y;
  reg [0:7] Y;
  integer i;

  always @ (G1 or G2 or G3 or A) begin
    Y = 8'b00000000;
    if (G1 & G2 & G3)
      for (i=0; i<=7; i=i+1)
        if (i == A) Y[i] = 1;
  end
endmodule
```

七段显示译码器

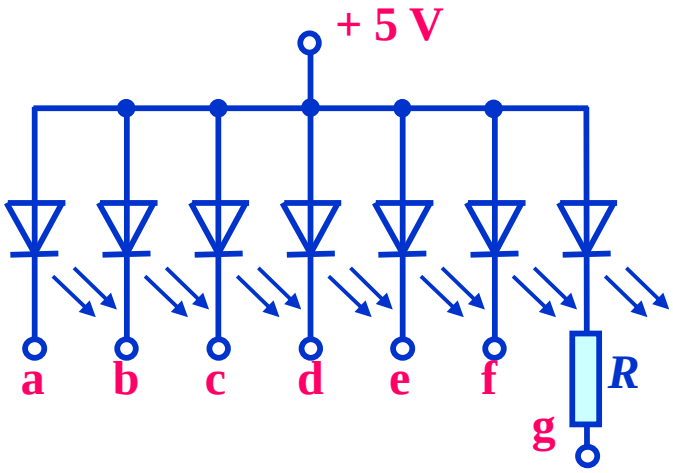
数码显示器

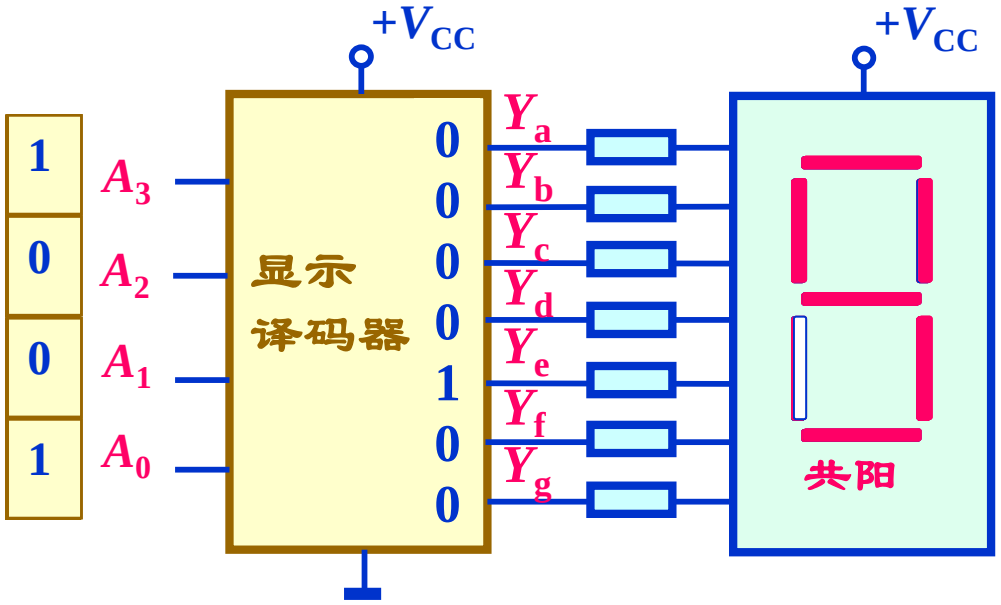
- 半导体显示(LED)
- 液晶显示(LCD)



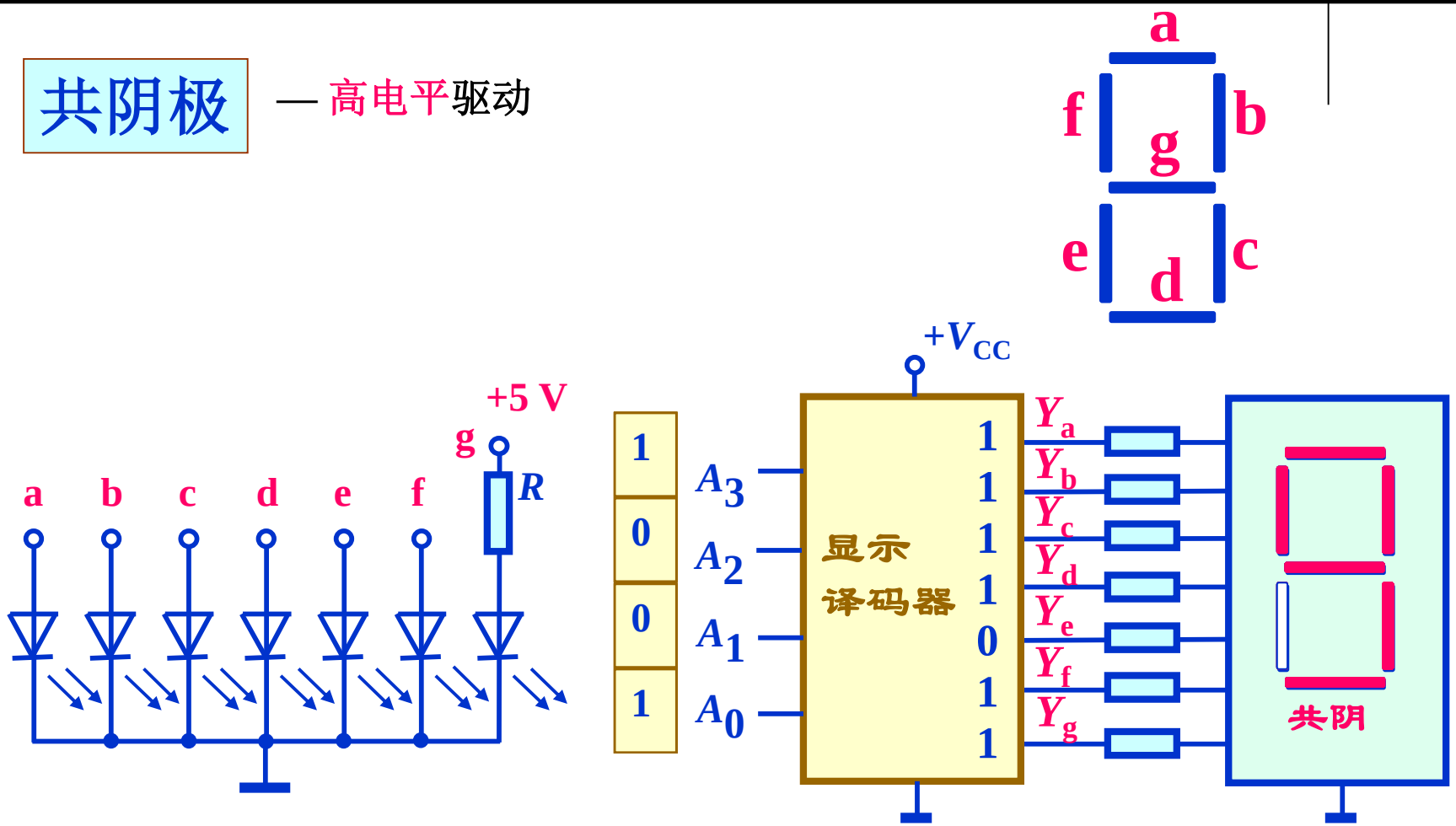
每字段是一只发光二极管

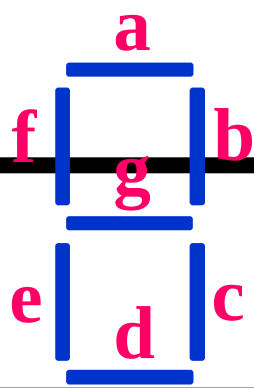
共阳极 — 低电平驱动





共阴极 — 高电平驱动

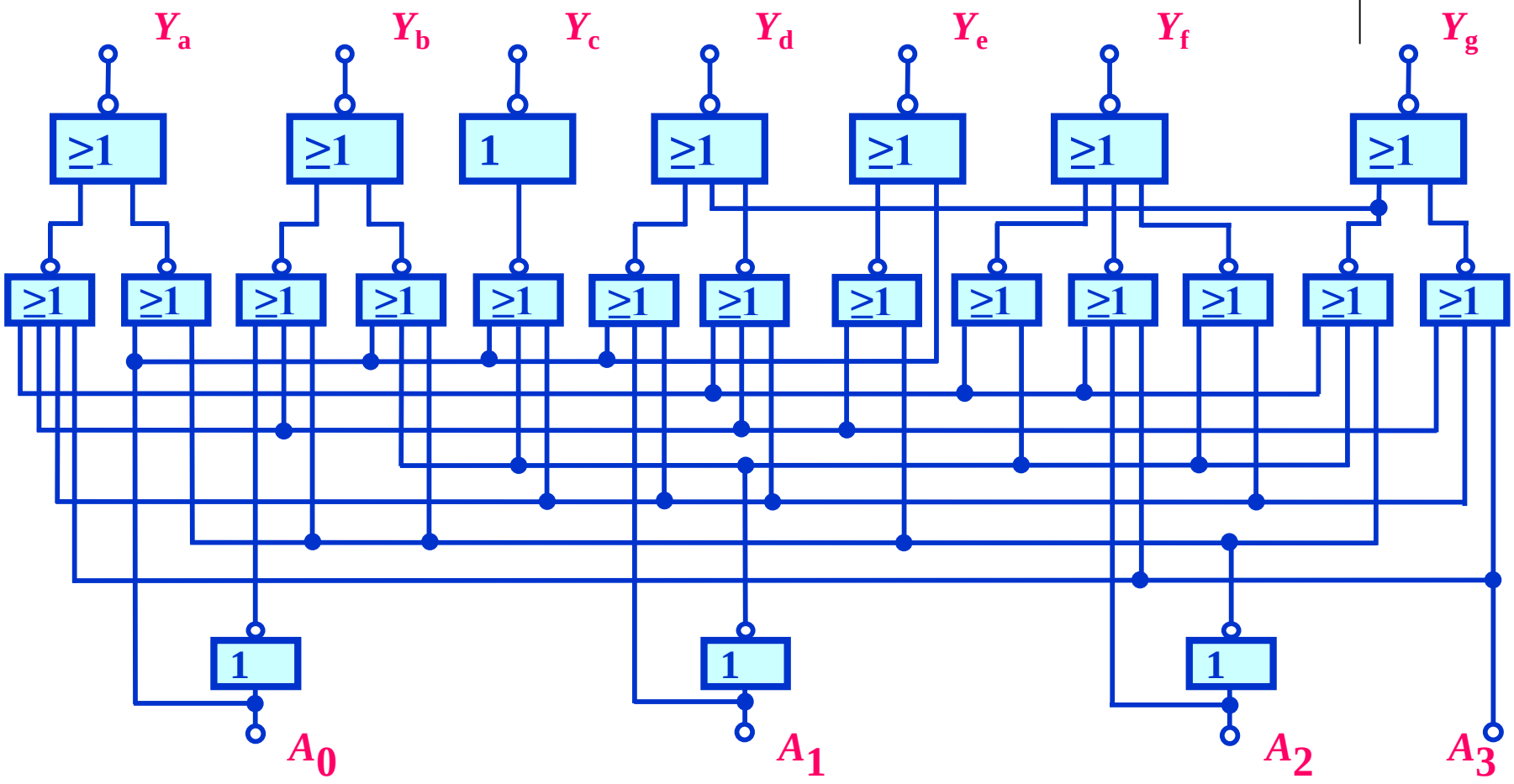

a
f
b
g
e
c
d
+5 V
g
R
a b c d e f
+V_{CC}
1
0
0
1
A_3
A_2
A_1
A_0
显示
译码器
1
1
1
1
0
1
1
Y_a
Y_b
Y_c
Y_d
Y_e
Y_f
Y_g
共阴



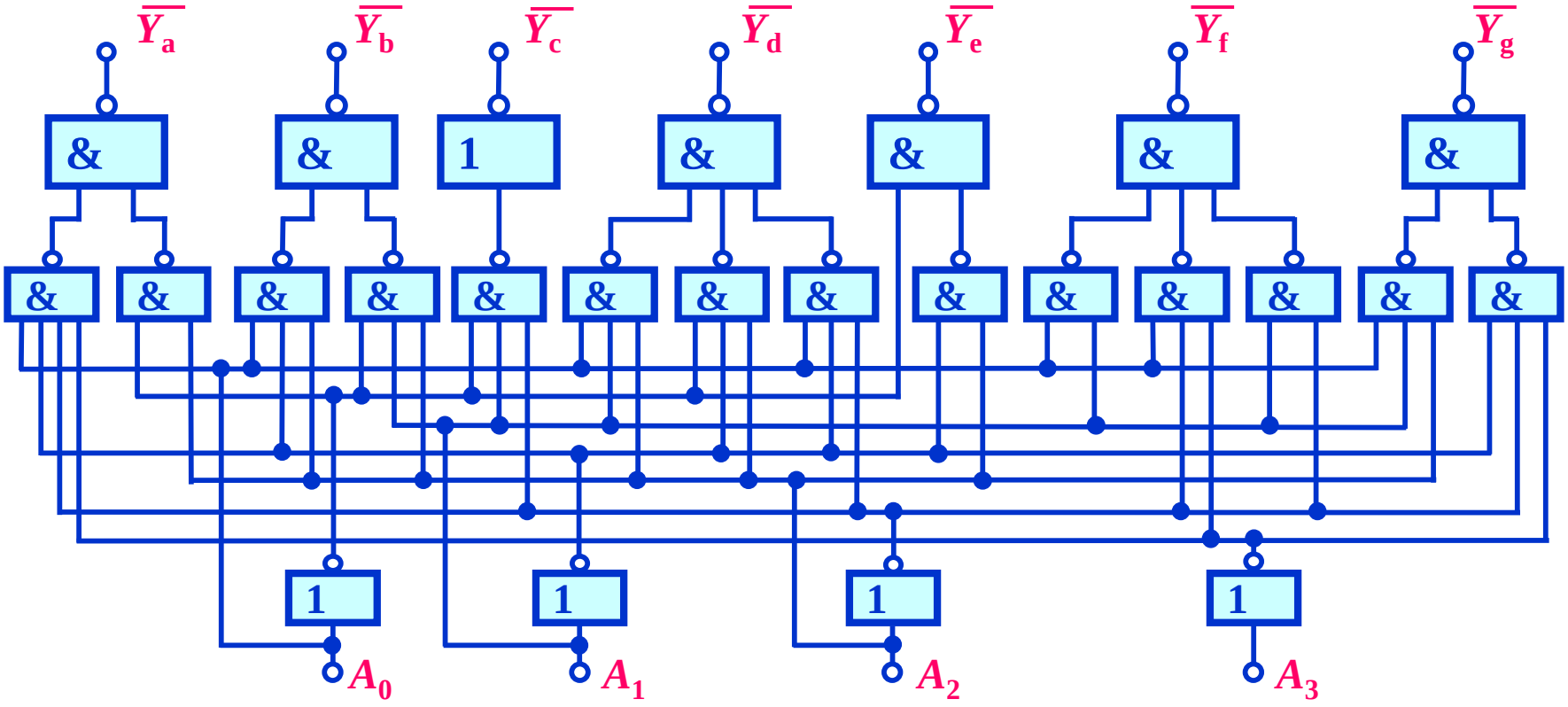
真值表

A3 A2 A1 A0	a b c d e f g (共阳极)	a b c d e f g (共阴极)
0000	0000001	1111110
0001	1001111	0110000
0010	0010010	1101101
0011	0000110	1111001
0100	1001100	0110011
0101	0100100	1011011
0110	0100000	1011111
0111	0001111	1110000
1000	0000000	1111111
1001	0000100	1111011

驱动共阴极数码管的电路—输出高电平有效



驱动共阳极数码管的电路—输出低电平有效



3.1 译码器 Decoder

- 七段显示译码器Verilog实现：

```
module Vr7seg(A, B, C, D, EN,
              SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG);
  input A, B, C, D, EN;
  output SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG;
  reg SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG;
  reg [1:7] SEGS;

  always @ (A or B or C or D or EN) begin
    if (EN)
      case ({D,C,B,A})
// 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')
        7:  SEGS = 7'b1110000;  // 7
        8:  SEGS = 7'b1111111;  // 8
        9:  SEGS = 7'b1110011;  // 9 (no 'tail')
        default SEGS = 7'bx;
      endcase
    else SEGS = 7'b0;
    {SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG} = SEGS;
  end
endmodule
```

3.2 编码器

- 译码器的输出编码通常比其输入编码位数多。
- 如果器件的输出编码比其输入编码位数少，则常常称它为**编码器（encoder）**。
- 最常见的就是2^n-n编码器或二进制编码器

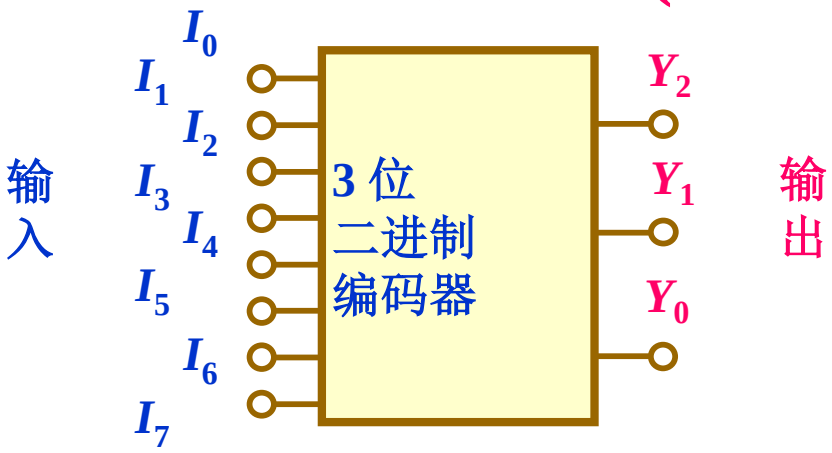
分类：
- 二进制编码器 $2^n \rightarrow n$
- 二—十进制编码器 $10 \rightarrow 4$

或
- 普通编码器
- 优先编码器

二进制编码器

用 n 位二进制代码对 $N = 2^n$ 个信号进行编码的电路

1. 3 位二进制编码器(8 线- 3 线)



$I_0 \sim I_7$ 是一组互相排斥的输入变量，任何时刻只能有一个端输入有效信号。

编码表

输 入	输 出		
	Y_2	Y_1	Y_0
I_0	0	0	0
I_1	0	0	1
I_2	0	1	0
I_3	0	1	1
I_4	1	0	0
I_5	1	0	1
I_6	1	1	0
I_7	1	1	1

函数式

$$Y_2 = I_4 + I_5 + I_6 + I_7$$
$$Y_1 = I_2 + I_3 + I_6 + I_7$$
$$Y_0 = I_1 + I_3 + I_5 + I_7$$

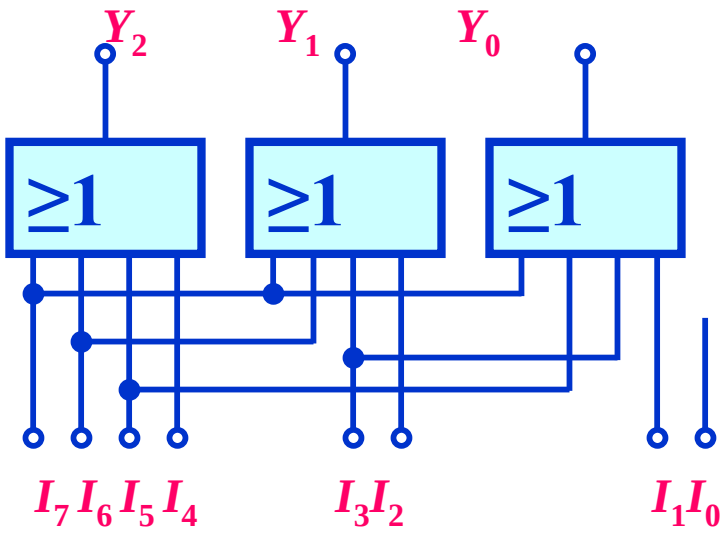
函数式

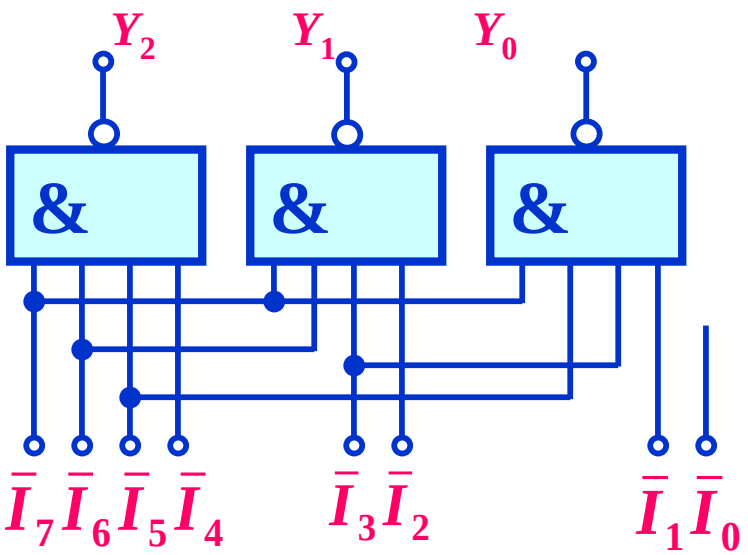
$$Y_2 = I_4 + I_5 + I_6 + I_7 = \overline{\bar{I}_4 \cdot \bar{I}_5 \cdot \bar{I}_6 \cdot \bar{I}_7}$$

$$Y_1 = I_2 + I_3 + I_6 + I_7 = \overline{\bar{I}_2 \cdot \bar{I}_3 \cdot \bar{I}_6 \cdot \bar{I}_7}$$

$$Y_0 = I_1 + I_3 + I_5 + I_7 = \overline{\bar{I}_1 \cdot \bar{I}_3 \cdot \bar{I}_5 \cdot \bar{I}_7}$$

逻辑图 — 用或门实现 — 用与非门实现





3 位二进制优先编码器

优先编码： 允许几个信号同时输入，但只对优先级别最高的进行编码。优先顺序：$I_7 \rightarrow I_0$

编码表

输入								输出		
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
1	×	×	×	×	×	×	×	1	1	1
0	1	×	×	×	×	×	×	1	1	0
0	0	1	×	×	×	×	×	1	0	1
0	0	0	1	×	×	×	×	1	0	0
0	0	0	0	1	×	×	×	0	1	1
0	0	0	0	0	1	×	×	0	1	0
0	0	0	0	0	0	1	×	0	0	1
0	0	0	0	0	0	0	1	0	0	0

函数式

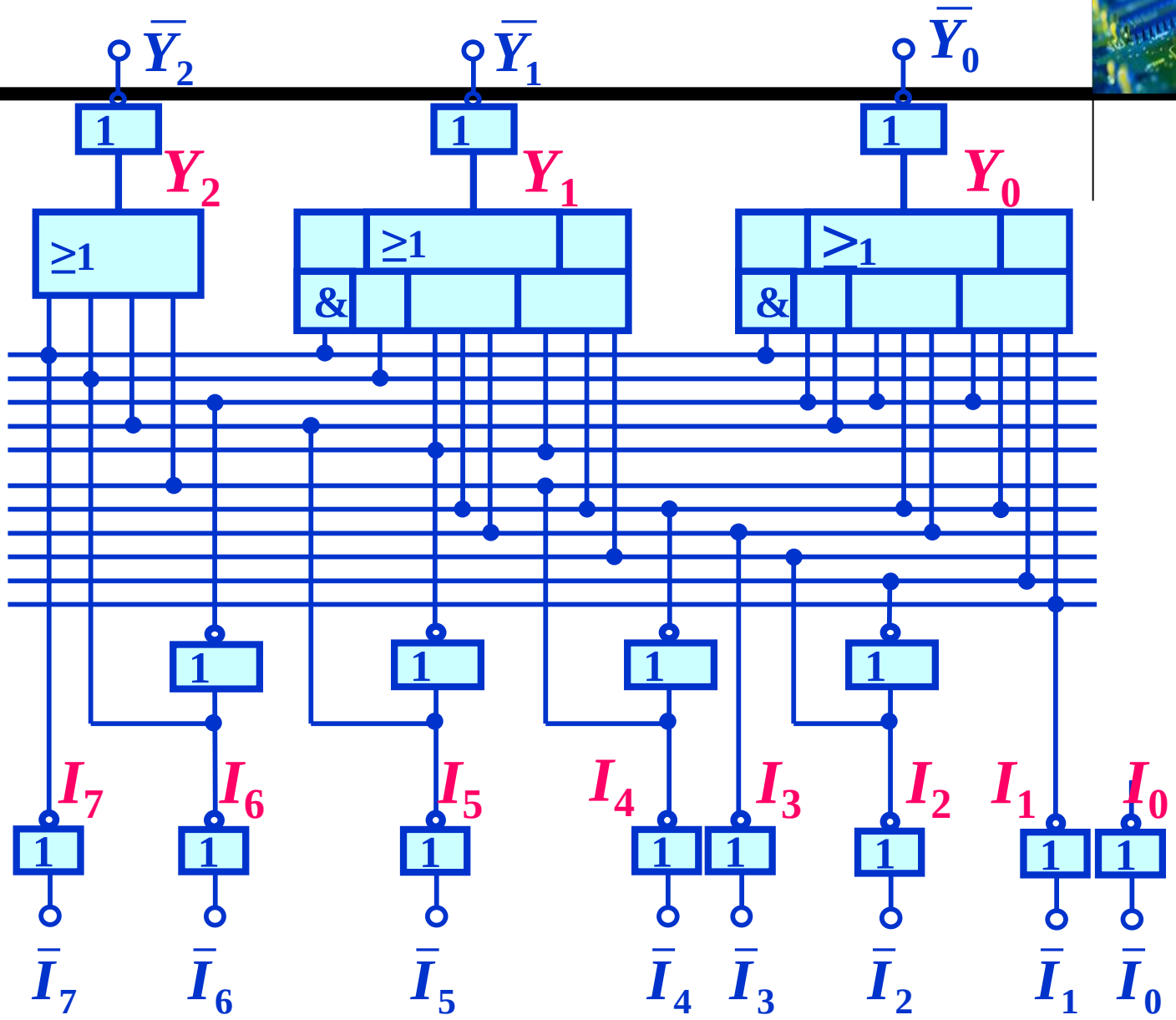
$$Y_2 = I_7 + I_6 + I_5 + I_4$$

$$Y_1 = I_7 + I_6 + \overline{I}_5\overline{I}_4 I_3 + \overline{I}_5\overline{I}_4 I_2$$

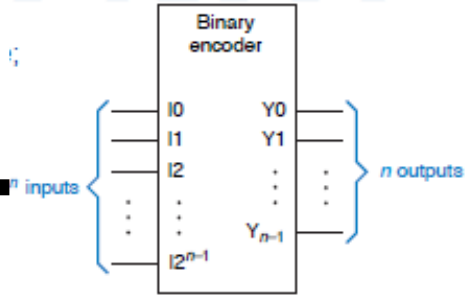
$$Y_0 = I_7 + \overline{I}_6 I_5 + \overline{I}_6\overline{I}_4 I_3 + \overline{I}_6\overline{I}_4\overline{I}_2 I_1$$

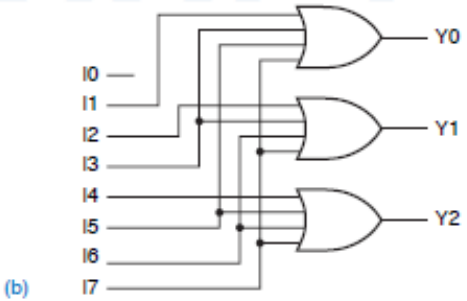
逻辑图

输入输出为反变量



3.2 编码器





- 优先编码器74x148
  - 低电平有效
  - 支持级联

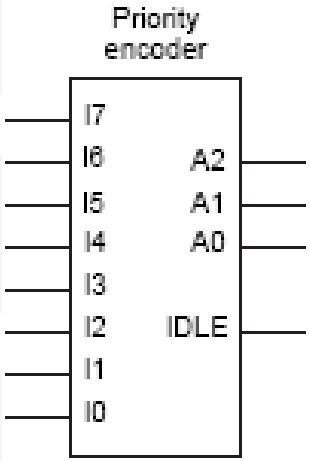


Figure 5-47
Logic symbol for a generic 8-input priority encoder.

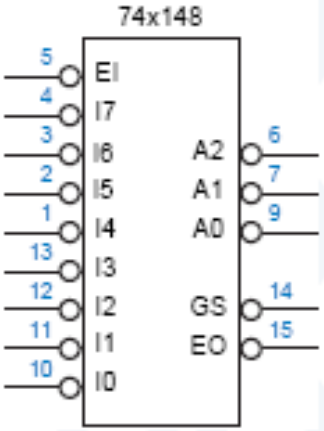


Figure 5-48
Logic symbol for the 74x148 8-input priority encoder.

Table 5-22 Truth table for a 74x148 8-input priority encoder.

Inputs									Outputs				
/EI	/I0	/I1	/I2	/I3	/I4	/I5	/I6	/I7	/A2	/A1	/A0	/GS	/EO
1	x	x	x	x	x	x	x	x	1	1	1	1	1
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
0	1	1	1	1	1	1	1	1	1	1	1	1	0

3.2 编码器

- **74x148**实现

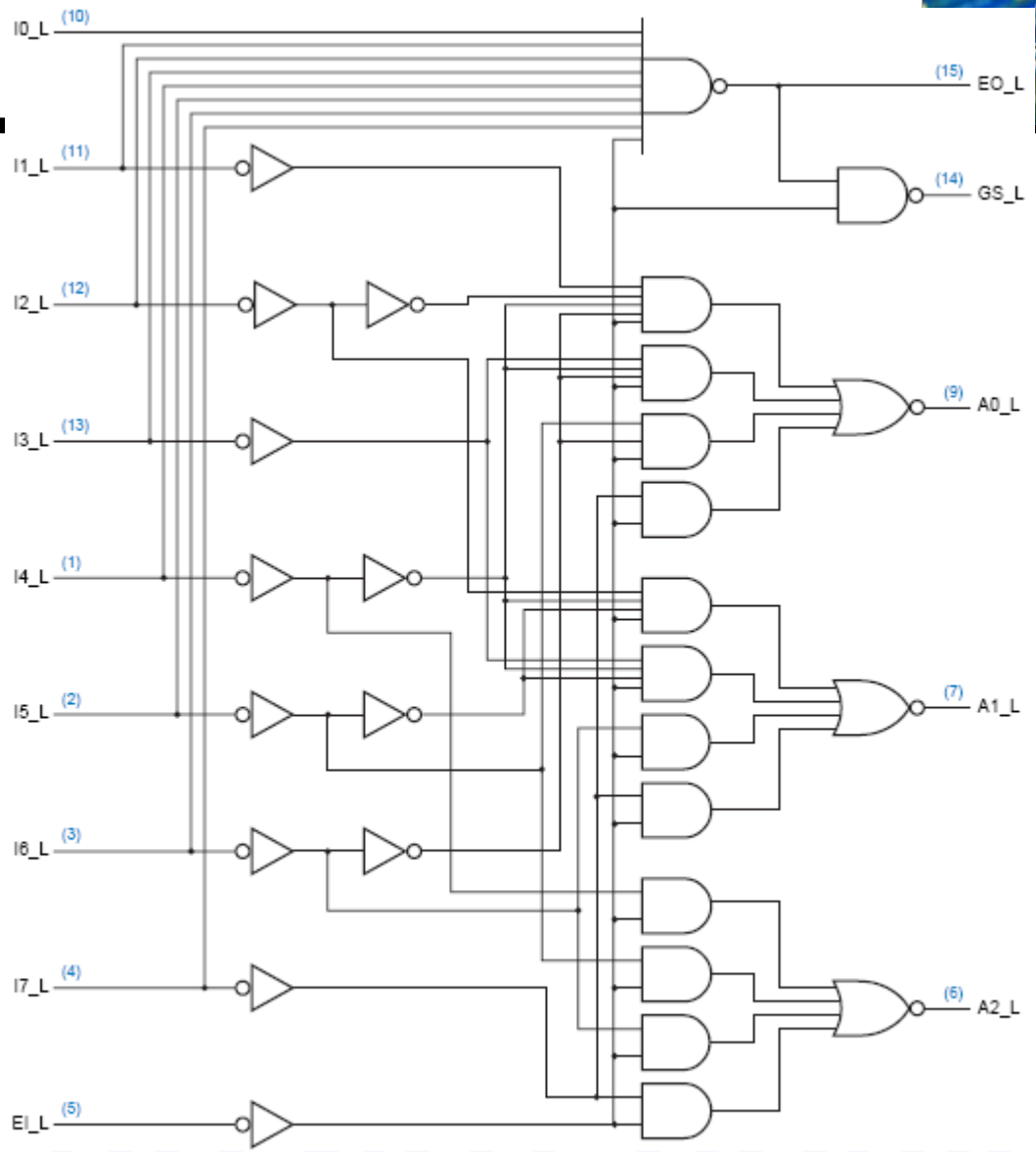
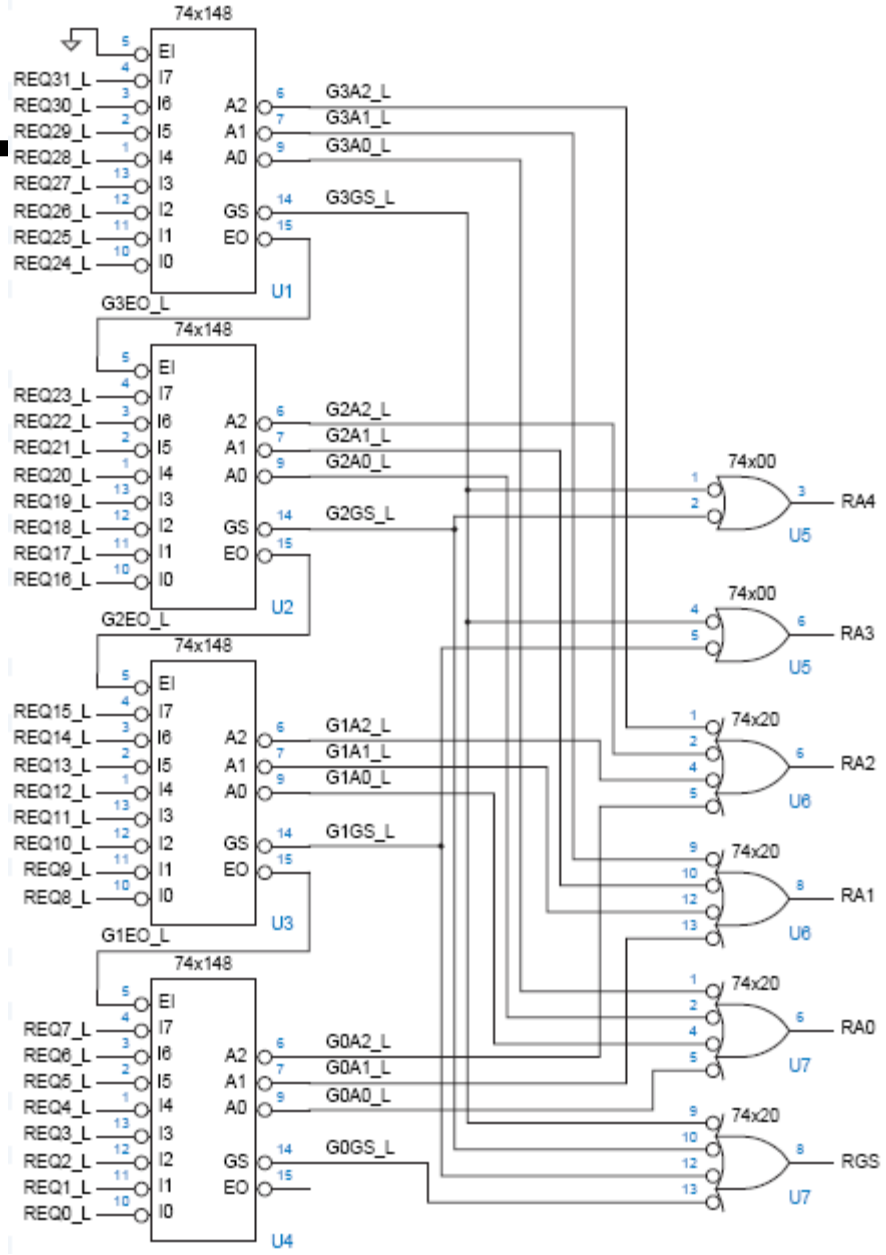


Figure 5-49 Logic diagram for the 74x148 8-input priority encoder, including pin numbers for a standard 16-pin dual in-line package.

3.2 编码器

- 74x148级联



3.2 编码器

- Verilog实现

换。很容易将它修改，使其利用不同优先级顺序、不同输入数字或增加更多的功能（
出次最高优先级），在DDPPonline的Xcver.3节中做了探讨。

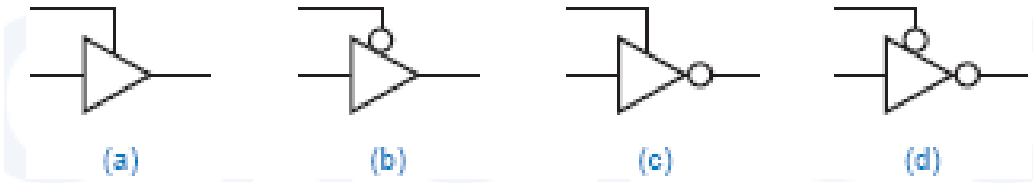
表6-31　类似74x148 8输入优先级编码器的行为描述型Verilog模块

```
module Vr74x148(EI_L, I_L, A_L, EO_L, GS_L);
  input EI_L;
  input [7:0] I_L;
  output [2:0] A_L;
  output EO_L, GS_L;
  reg [7:0] I;
  reg [2:0] A, A_L;
  reg EI, EO_L, EO, GS_L, GS;
  integer j;

  always @ (EI_L or EI or I_L or I or A or EO or GS) begin
    EI = ~EI_L; I  = ~I_L;                  // convert inputs
    EO_L = ~EO; GS_L = ~GS; A_L = ~A;  // convert outputs
    EO = 1; GS = 0; A = 0;              // default output values
    begin
      if (EI==0) EO = 0;
      else for (j=0; j<=7; j=j+1) // check low priority first
             if (I[j]==1)
               begin GS = 1; EO = 0; A = j; end
    end
  end
endmodule
```

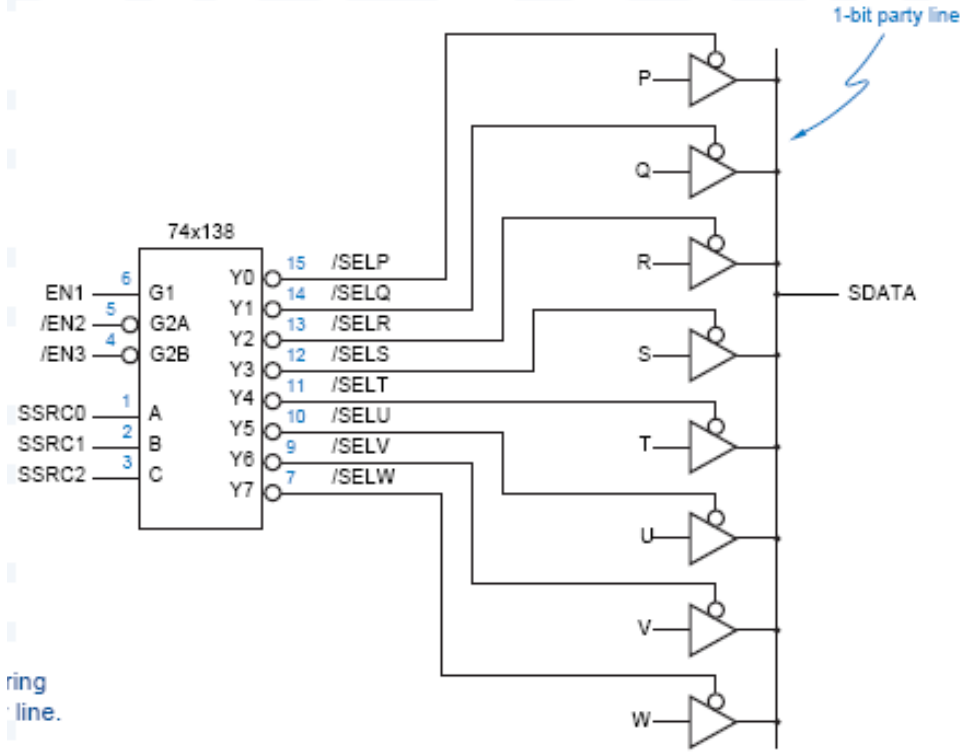
3.3 三态器件

- 三态缓冲器

Various three-state buffers: (a) noninverting, active-high enable; (b) non-inverting, active-low enable; (c) inverting, active-high enable; (d) inverting, active-low enable.

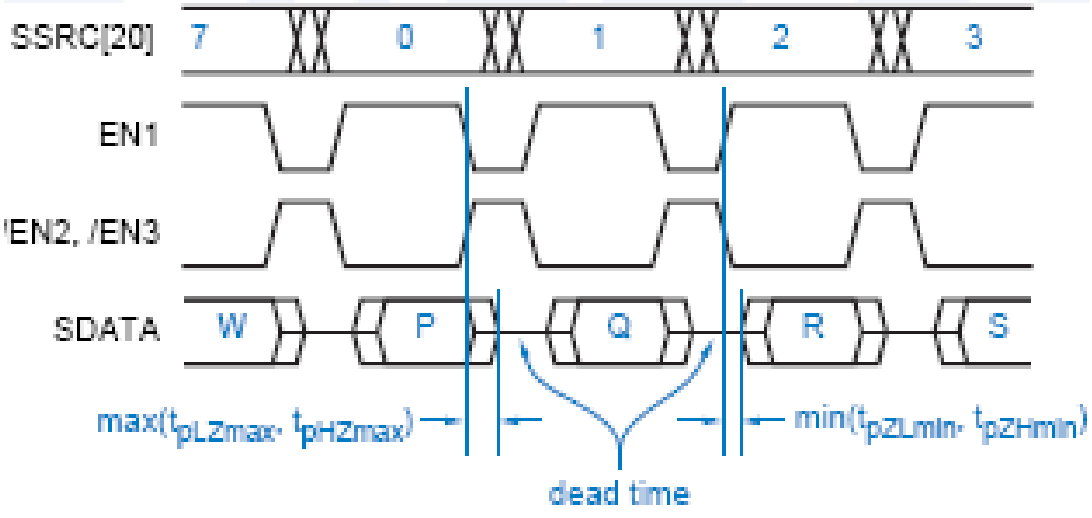


- 三态器件允许多个信号源共享单个线路。



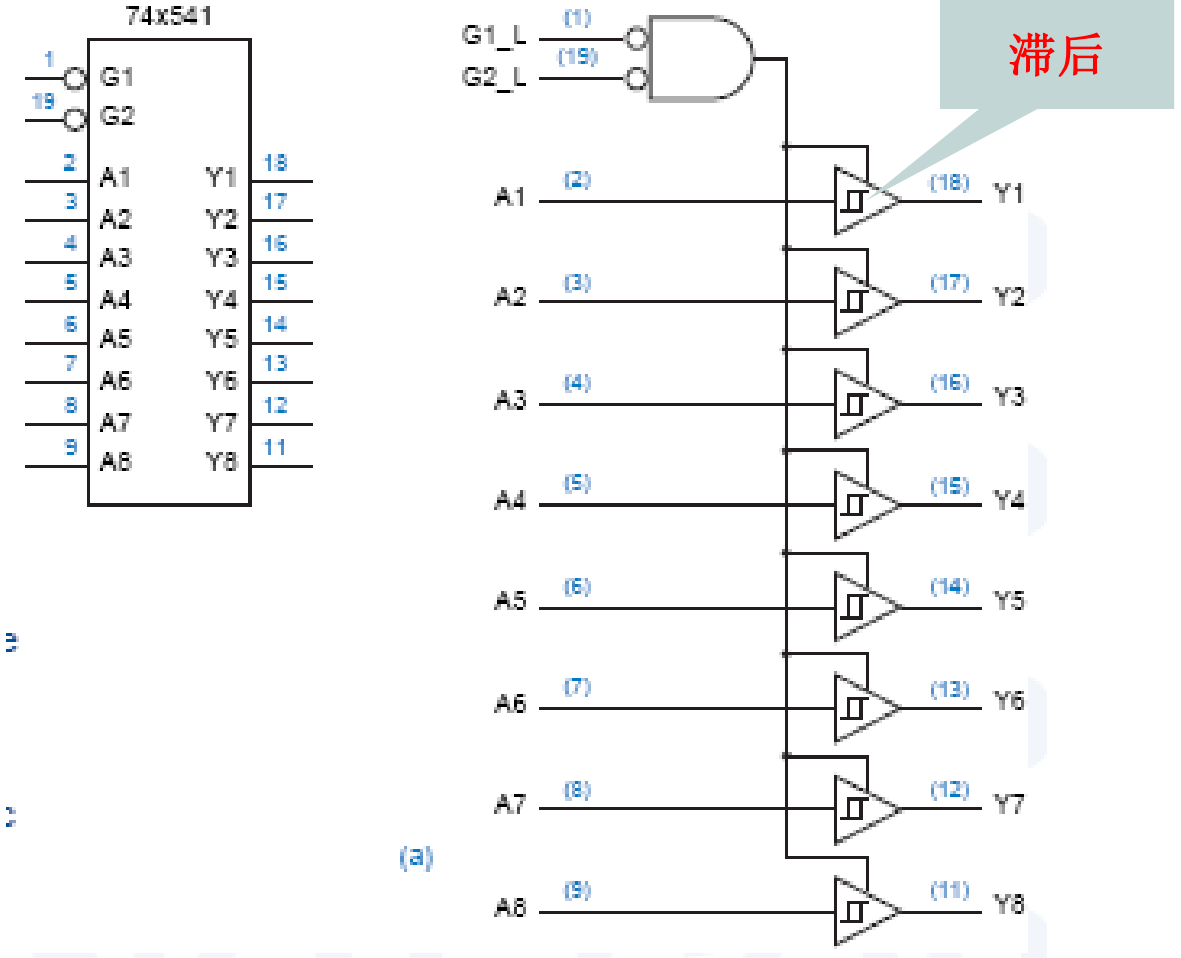
3.3 三态器件

- 三态缓冲器
  - 三态门进入和离开高阻态的延迟时间不同，会造成输出线路上值的混乱，造成同时驱动的冲突现象。
  - 解决的方法：设计控制逻辑，保证一段**截止时间**（dead time），这段时间不应该有任何器件驱动同线。



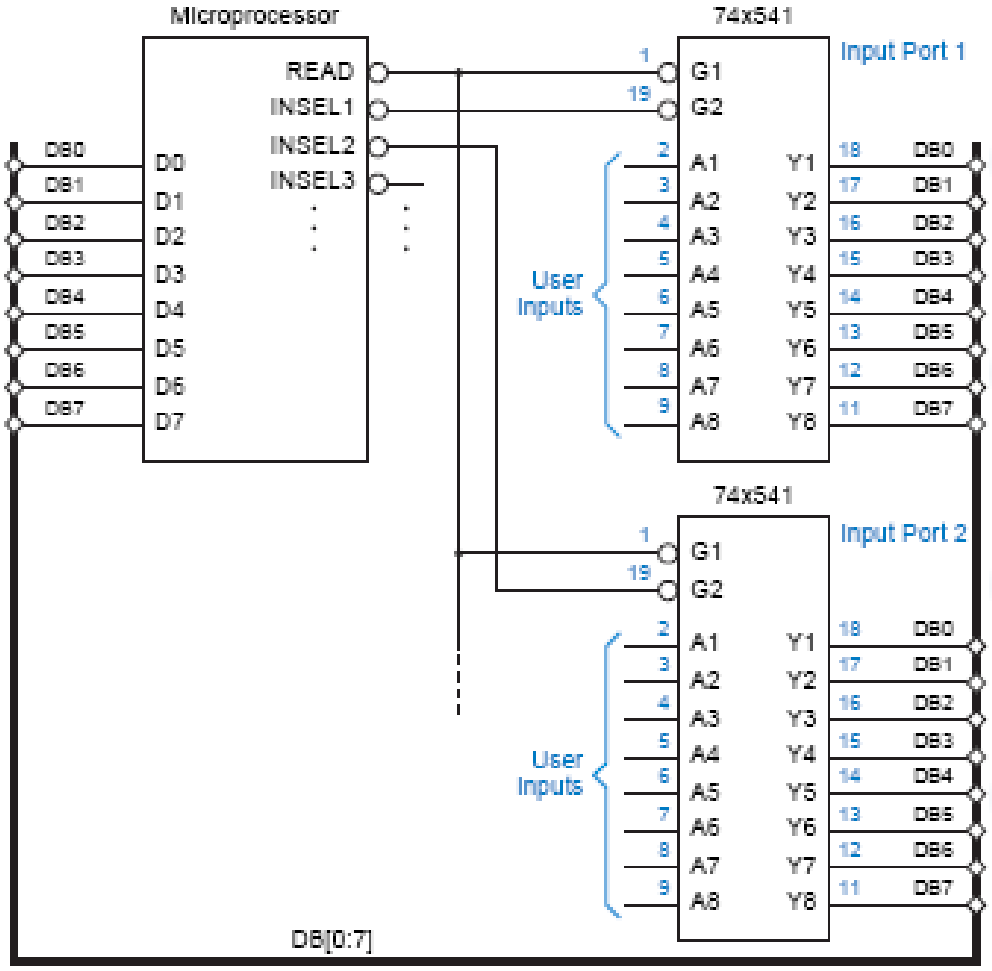
3.3 三态器件

- 标准MSI三态缓冲器



3.3 三态器件

- 标准MSI三态缓冲器



3.3 三态器件

- 标准MSI三态缓冲器——总线收发器
  - DIR控制方向
  - G_L控制使能
  - 使能控制无效，可能两条总线需要分别处理事情。

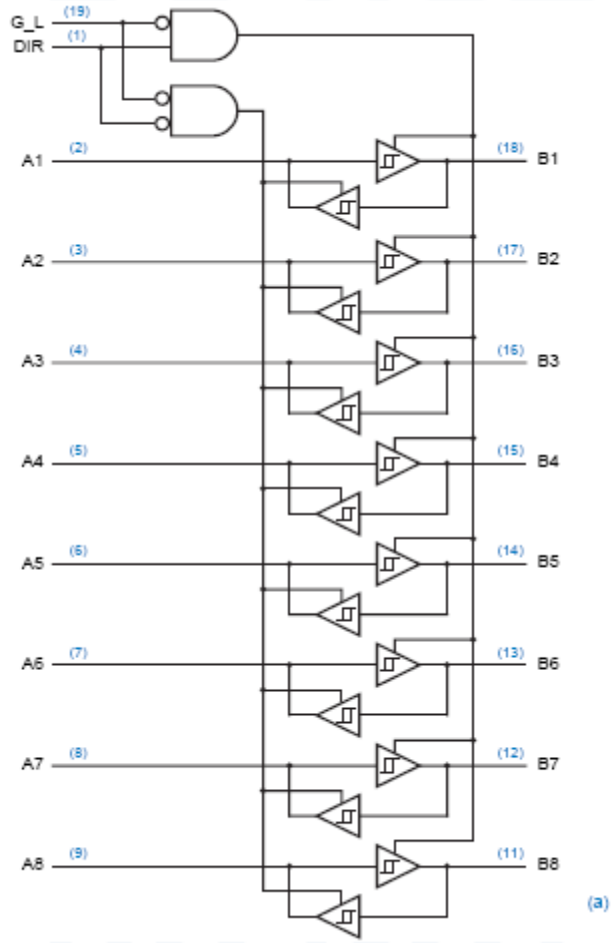
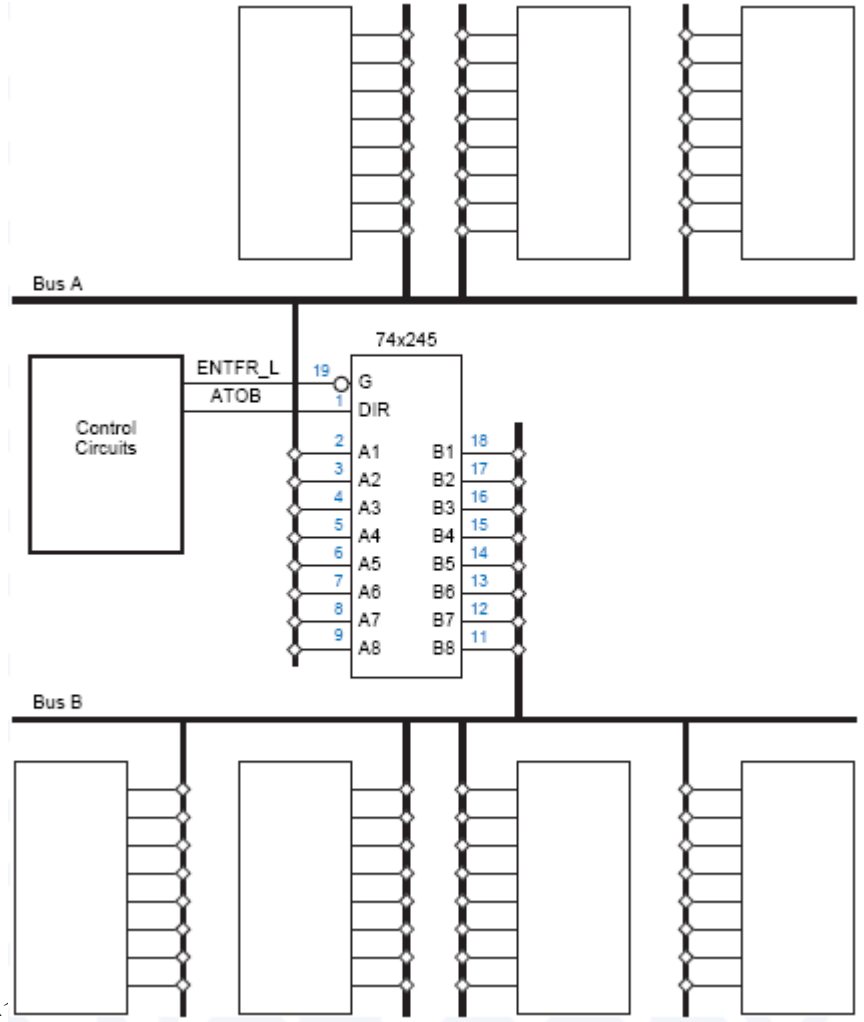


Figure 5-58
The 74x245 octal three-state transceiver: (a) logic diagram; (b) traditional logic symbol.

3.3 三态器件

- 标准MSI三态缓冲器——总线收发器
  - 双向总线控制应用

ENTFR_L	ATOB	*Operation*
0	0	Transfer data from a source on bus B to a destination on bus A.
0	1	Transfer data from a source on bus A to a destination on bus B.
1	x	Transfer data on buses A and B independently.



3.3 三态器件

- 标准MSI三态缓冲器
  - Ve
  - S[2
  - S[1
  - 其

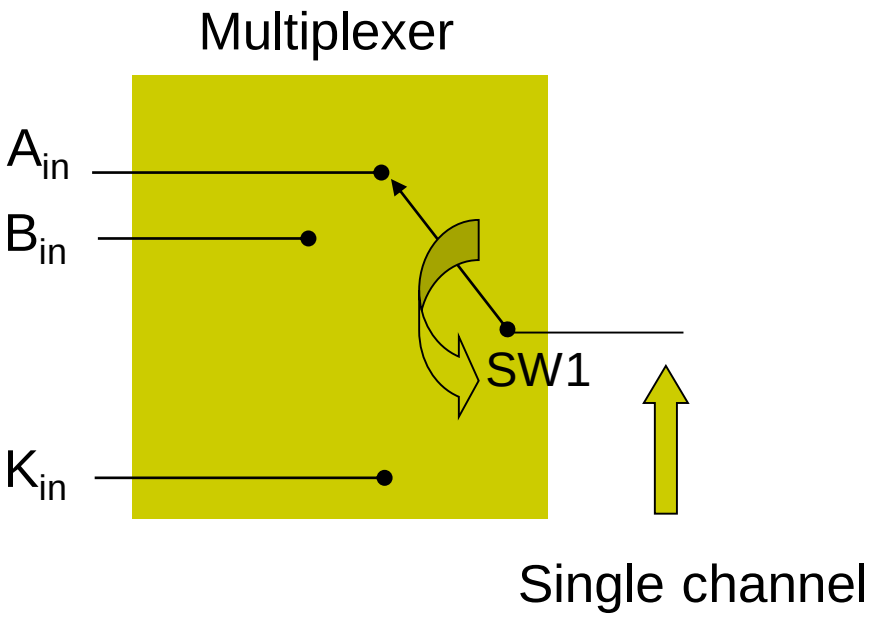
选总线不被驱动（尽管输出被使能）的逻辑。

表6-41　4路2位总线收发器的Verilog模块

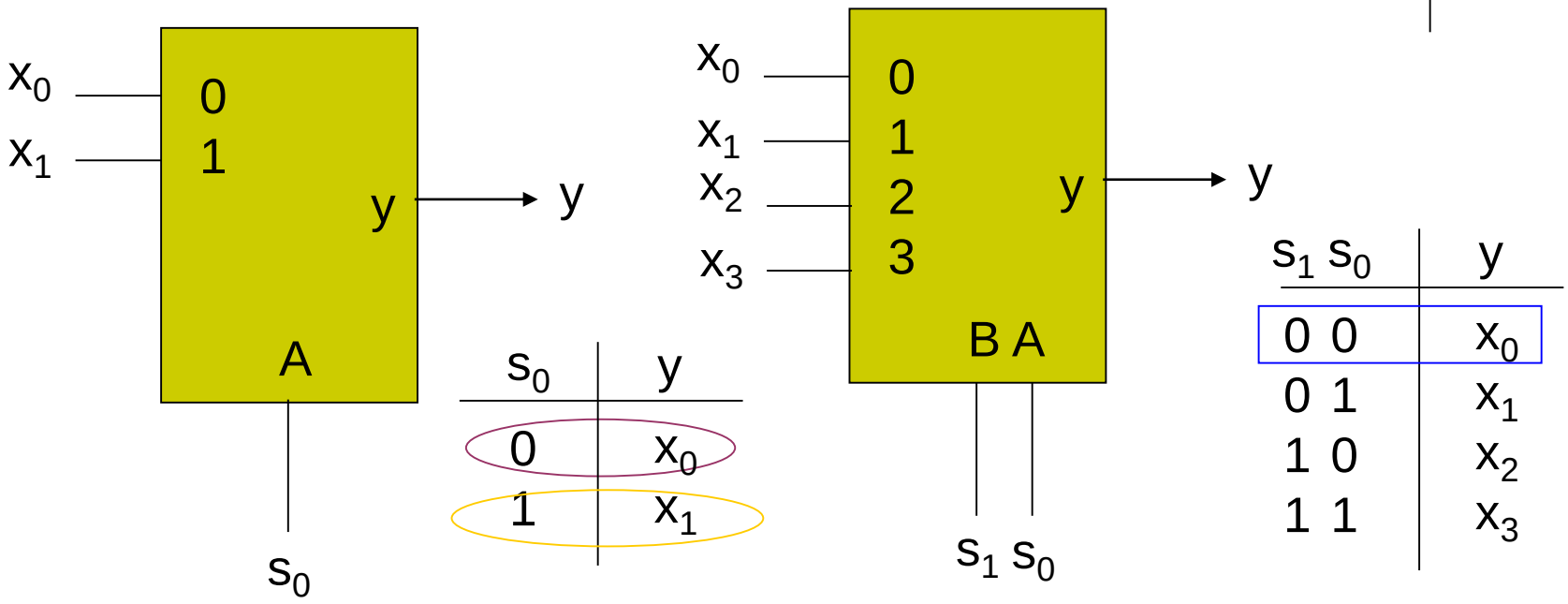
```
module VrXcvr4x8(A, B, C, D, S, AOE_L, BOE_L, COE_L, DOE_L, MOE_L);
  input [2:0] S;
  input AOE_L, BOE_L, COE_L, DOE_L, MOE_L;
  inout [1:8] A, B, C, D;
  reg [1:8] ibus;

  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])
      0: ibus = A;
      1: ibus = B;
      2: ibus = C;
      3: ibus = D;
    endcase
  end
  assign A = ((~AOE_L & ~MOE_L) && (S[2:0] != 4)) ? ibus : 8'bz;
  assign B = ((~BOE_L & ~MOE_L) && (S[2:0] != 5)) ? ibus : 8'bz;
  assign C = ((~COE_L & ~MOE_L) && (S[2:0] != 6)) ? ibus : 8'bz;
  assign D = ((~DOE_L & ~MOE_L) && (S[2:0] != 7)) ? ibus : 8'bz;
endmodule
```

3.4、数据选择器(多路选择器)

- 又称为多路转换器或多路开关(Multiplexers, MUX)
  - 一种多路输入，单路输出的逻辑构件。
  - 其输出等于一路输入，取决于控制信号。



2-to-1 multiplexer

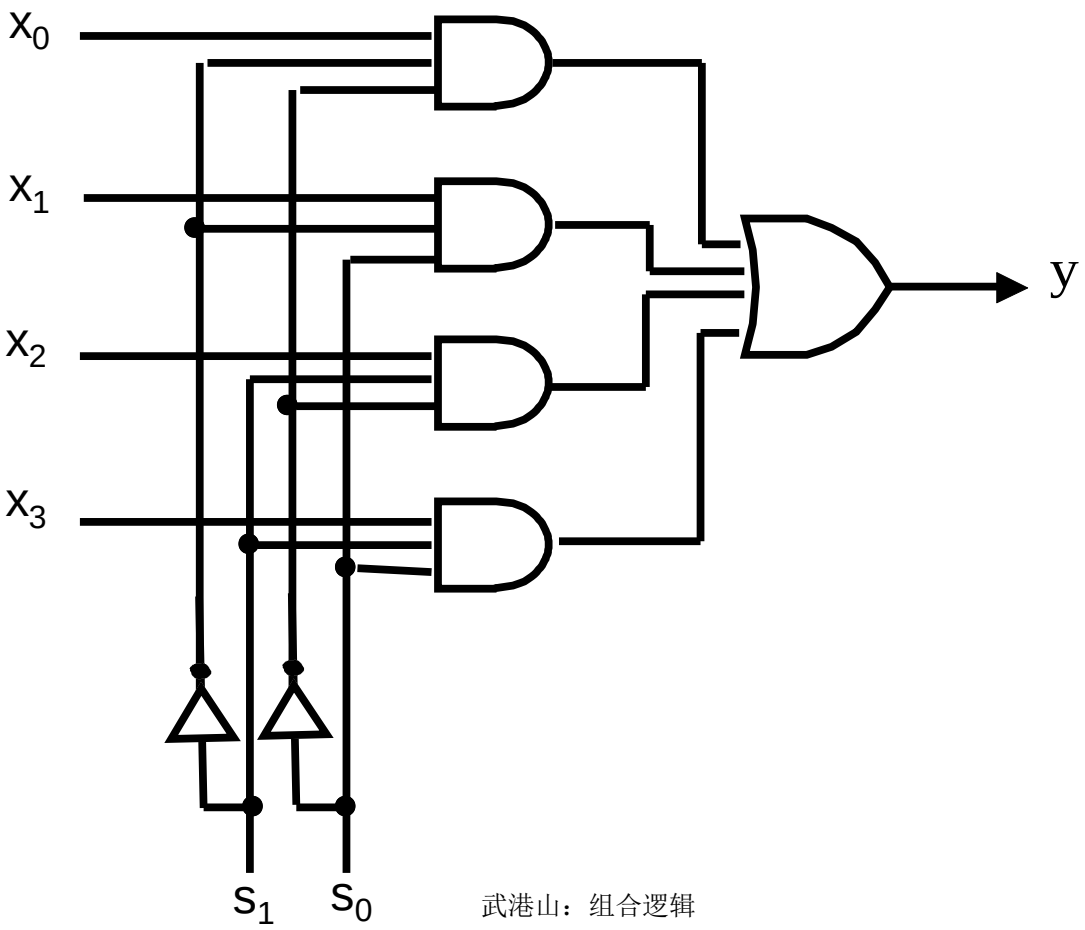


$$y = x_0\overline{s_0} + x_1 s_0$$

$$y = \overline{s_1}\,\overline{s_0} x_0 + \overline{s_1} s_0 x_1 + s_1 \overline{s_0} x_2 + s_1 s_0 x_3$$

4-to-1 MUX realization

$$y = \overline{s_1}\,\overline{s_0}x_0 + \overline{s_1}s_0x_1 + s_1\overline{s_0}x_2 + s_1s_0x_3$$



16-to-1 MUX

MUX 0

s_3	s_2	s_1	s_0	y
0	0	0	0	x_0
0	0	0	1	x_1
0	0	1	0	x_2
0	0	1	1	x_3
0	1	0	0	x_4
0	1	0	1	x_5
0	1	1	0	x_6
0	1	1	1	x_7

MUX 1

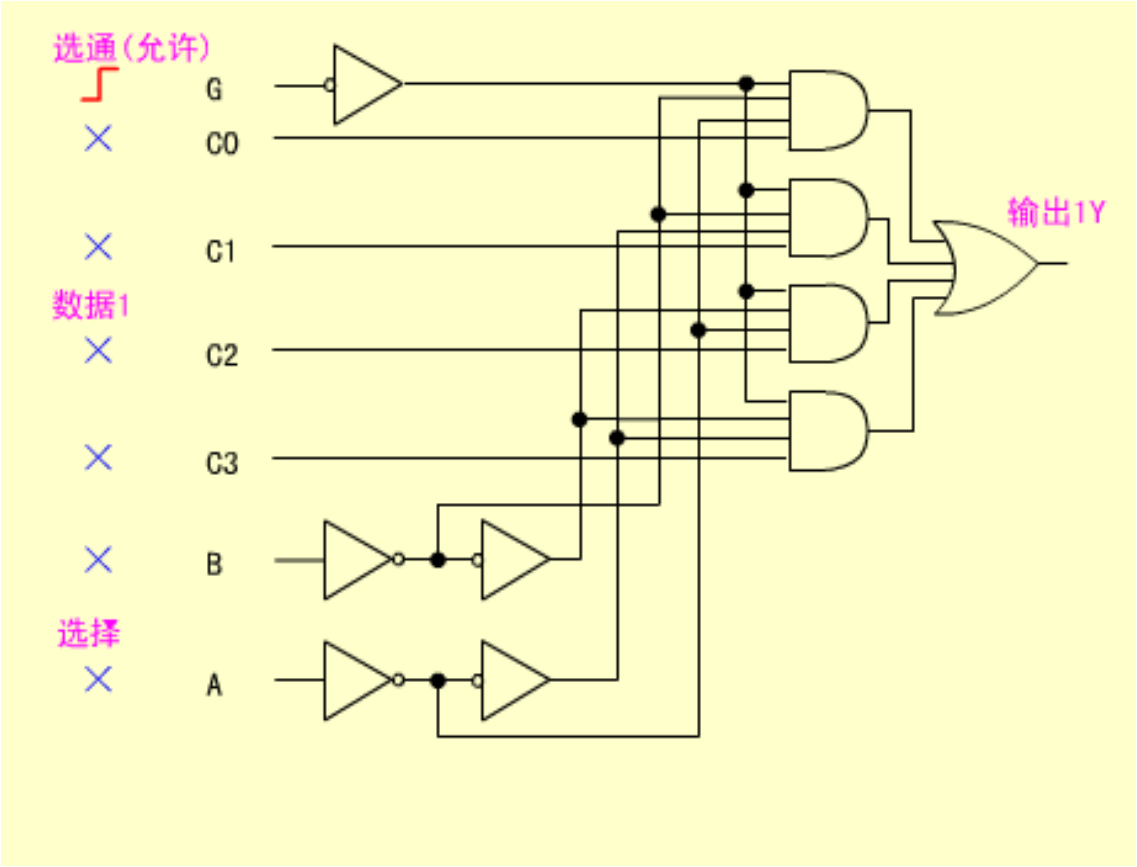
MUX 2

s_3	s_2	s_1	s_0	y
1	0	0	0	x_8
1	0	0	1	x_9
1	0	1	0	x_{10}
1	0	1	1	x_{11}
1	1	0	0	x_{12}
1	1	0	1	x_{13}
1	1	1	0	x_{14}
1	1	1	1	x_{15}

MUX 3

3.4、数据选择器

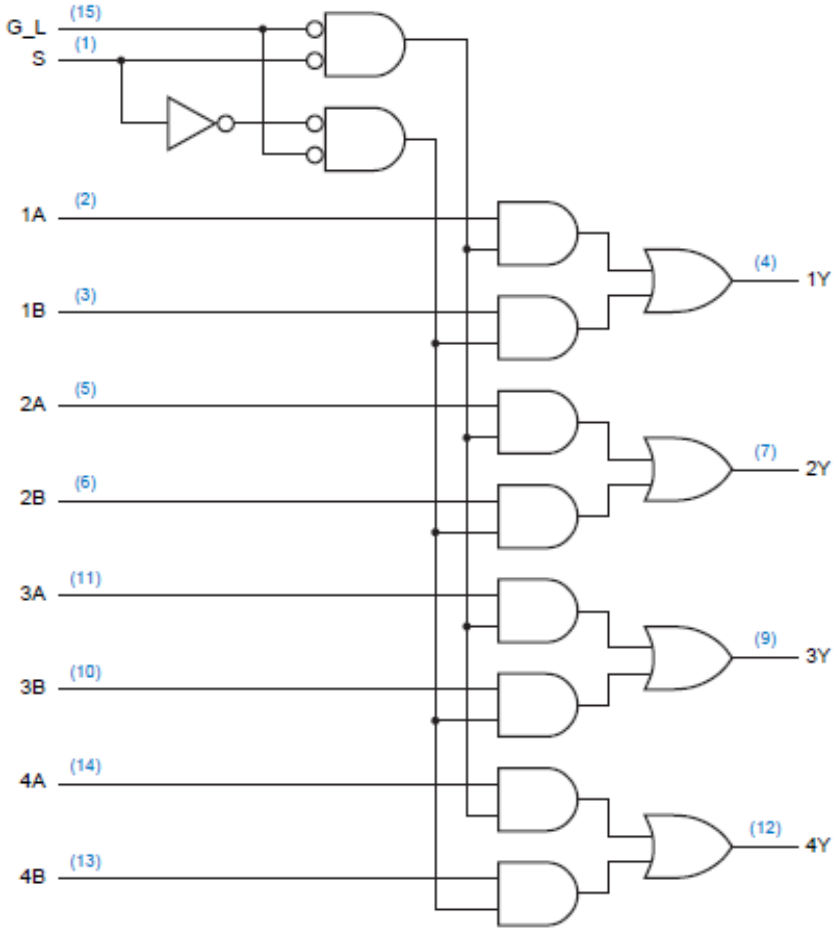
- 74LS153的逻辑结构

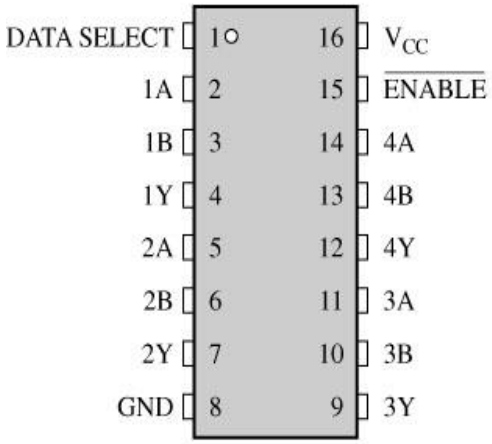


2、数据选择器

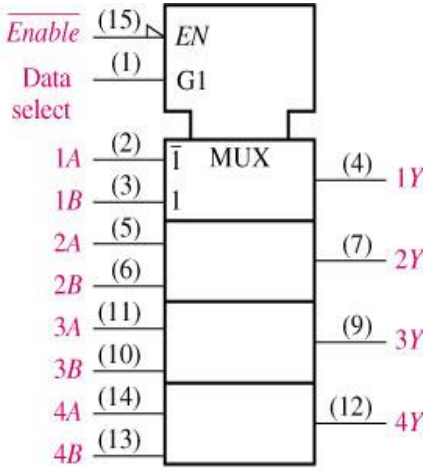
- 芯片介绍
  - 74151：8-to1
  - 74150：16-to-1
  - 74153：dual(2组) 4-to-1
  - 74157：quad (4组) 2-to-1

74157 2-1 MUX





(a) Pin diagram



(b) Logic symbol

74151 8-to-1 MUX

Inputs				Outputs	
EN_L	C	B	A	Y	Y_L
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′

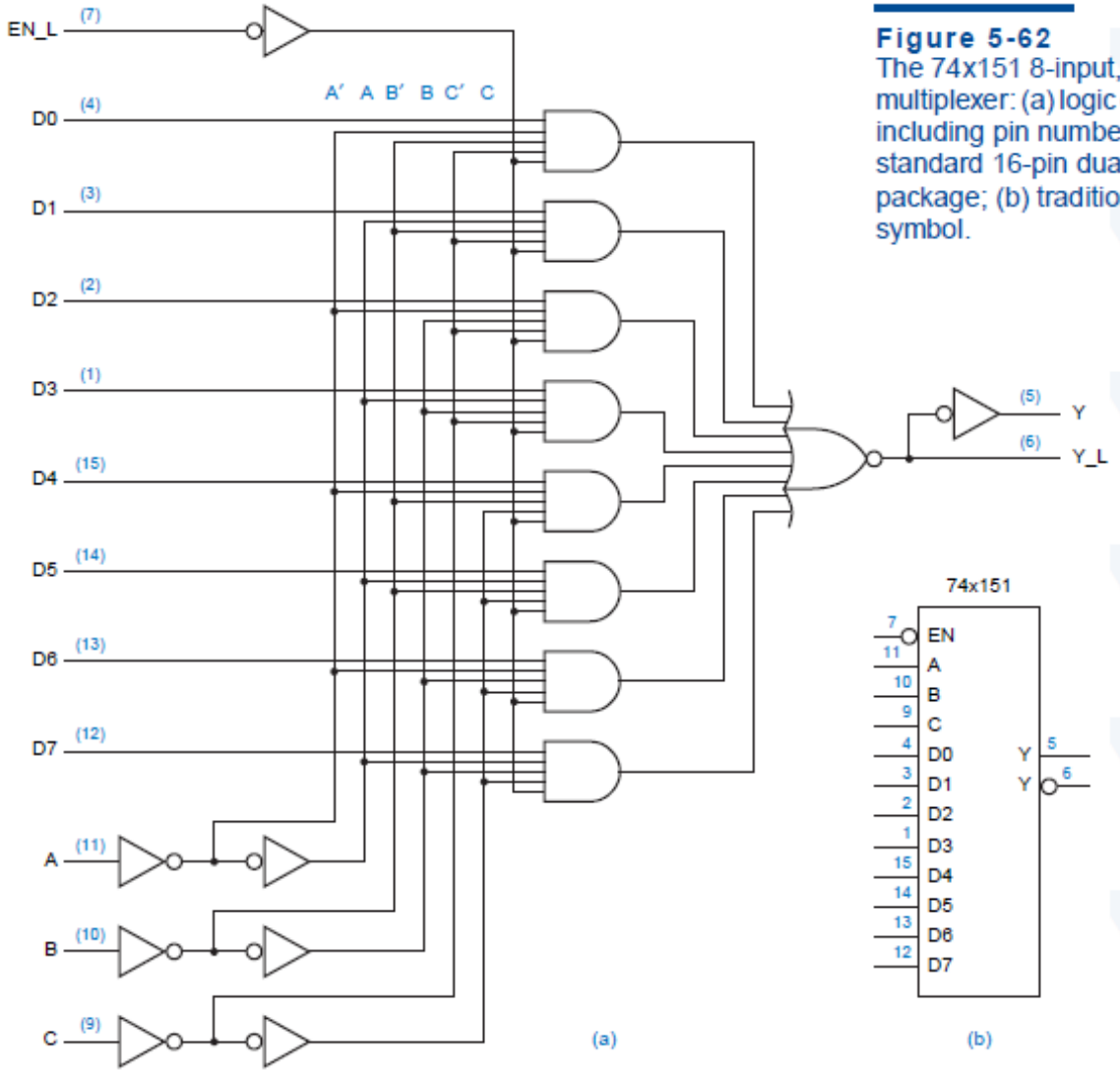


Figure 5-62
The 74x151 8-input, 1-bit multiplexer: (a) logic diagram, including pin numbers for a standard 16-pin dual in-line package; (b) traditional logic symbol.

MUX的扩展

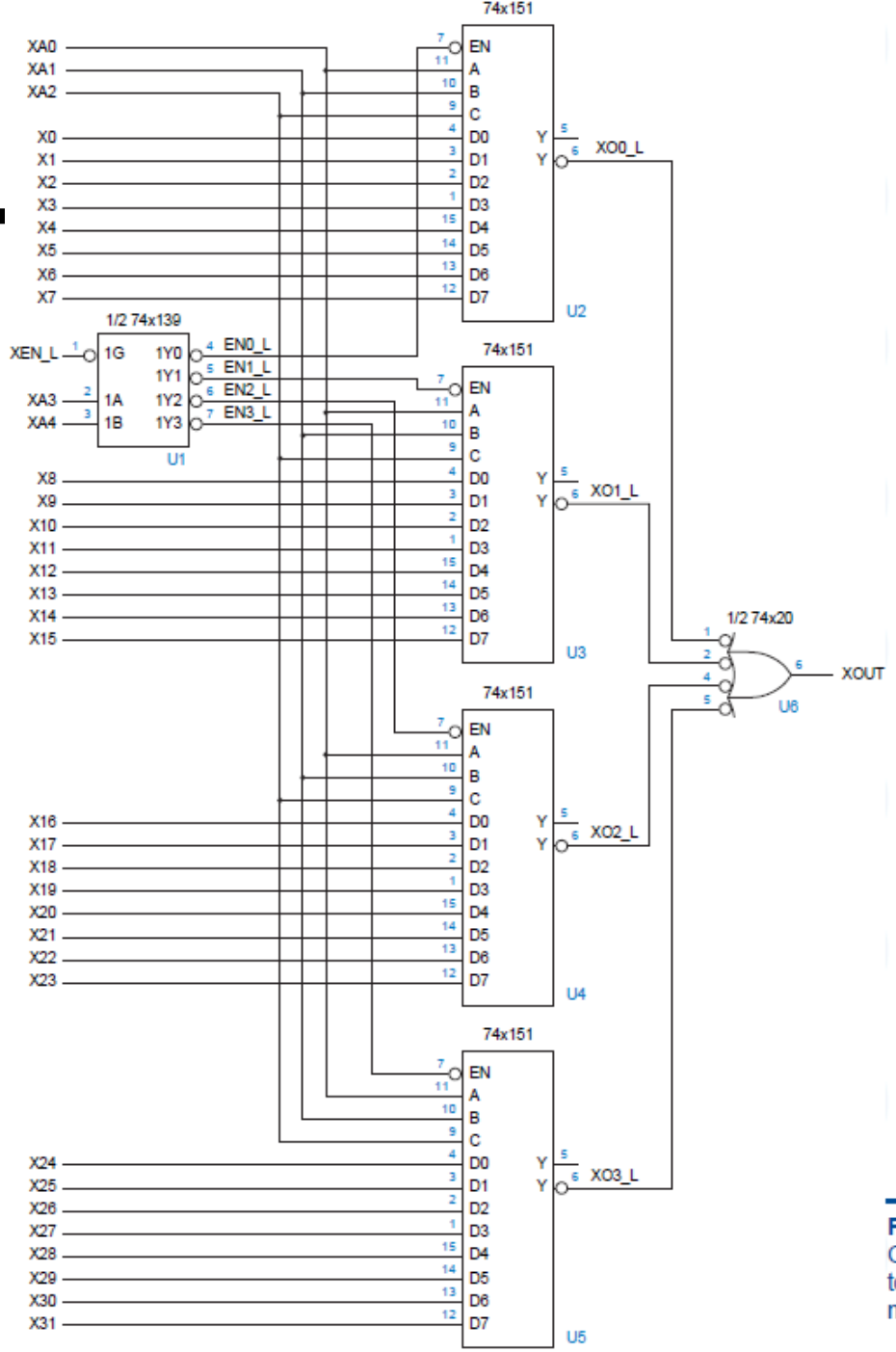


Figure 5-65
Combining 74x151s to make a 32-to-1 multiplexer.

3.4、数据选择器

- 假设是n个输入变量，则需要

$$m = \log_2^n$$

- 输出函数y,为所有选择变量最小项mi与上相应数据输入线Di之和。

$$y = \sum_{i=0}^{n} m_i D_i$$

- 以74LS153为例介绍数据选择器的逻辑结构
  - 74LS153是四选一多路开关，即一个芯片内集成了两个同样结构的四选一多路开关

多路选择器的推广应用

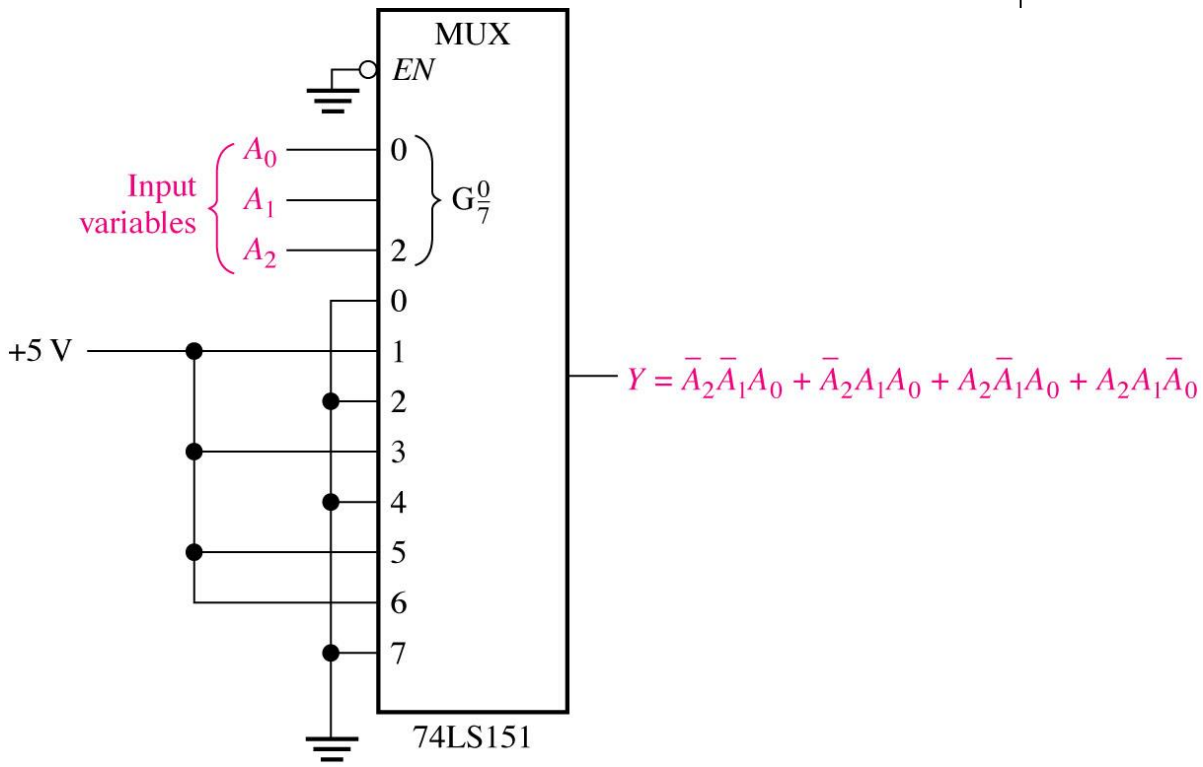
- 多路选择器除完成对多路数据进行选择的基本功能外，
- 在逻辑设计中主要用来实现各种逻辑函数功能。
- 用多路选择器实现分时多路转换电路。(将并行输入的数据转换成串行输出)。

Use MUX to Realize Function

- 方法I：用具有n个选择变量的MUX实现n个变量的函数。
  - 将函数的n个变量依次连接到MUX的n个选择变量端，并将函数表示成最小项之和的形式。若函数表达式中包含最小项m_i，则相应MUX的D_i接1，否则D_i接0。

Example1 Use 74151(8 to 1MUX) to Realize function: $f(A0,A1,A2) = \sum m(1,3,5,6)$

m	$A_2A_1A_0$	D_i
0	0 0 0	0
1	0 0 1	1
2	0 1 0	0
3	0 1 1	1
4	1 0 0	0
5	1 0 1	1
6	1 1 0	1
7	1 1 1	0



方法简单，但并不经济，因为MUX的数据输入端未能得到充分利用。对于具有n个变量的逻辑函数，完全可以用少于n个选择变量的MUX实现。利用数据输入线定义变量。

- 方法Ⅱ：用具有n-1个选择控制变量的MUX实现n个变量函数功能
  - 即从函数的n个变量中任n-1个作为MUX的选择控制变量，并根据所选定的选择控制变量将函数变换成$F=\sum m_i D_i$的形式，以确定各数据输入D_i。假定剩余变量为X，则D_i的取值只可能是0、1、X或X’四者之一。

N-1 selector value

- Example：Use 4 to 1 MUX to realize F（A，B，C）=∑m(1,3,5,6)

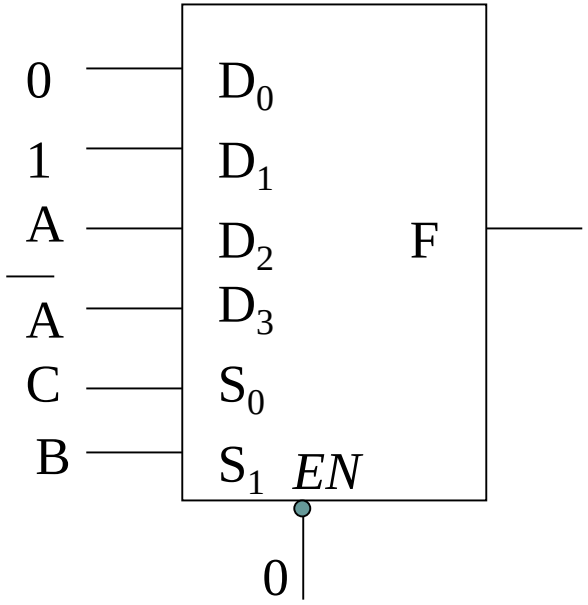
1 选择B、C为选择变量，A为数据输入。将函数表达式展开成选择变量最小项表示形式。列出选择器的功能表，确定每条数据输入线的值。画出逻辑电路图

$$f = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}C + AB\overline{C}$$

$$= \overline{B}\,\overline{C} \bullet 0 + \overline{B}C(A + \overline{A}) + B\overline{C}A + BC\overline{A}$$

选择 BC	输入 A	输出 F
00	0	0
01	1	1
10	A	1
11	$\overline{A}$	1

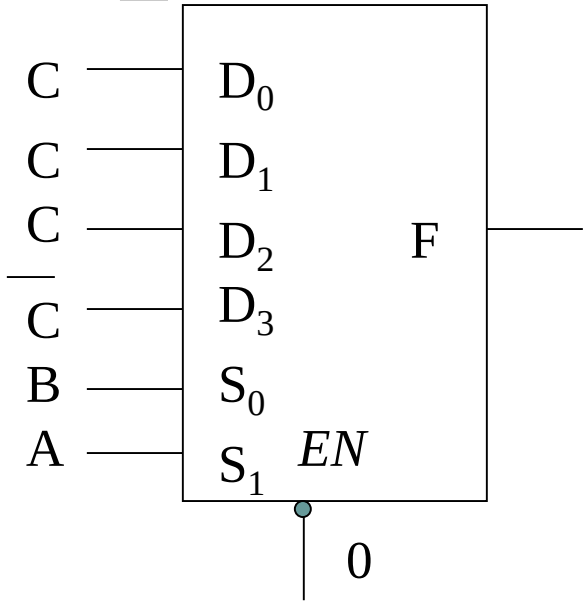
功能表



2 选择AB为选择变量，C为数据输入，画出卡诺图，标出实现函数的最小项，确定每种选择情况下，数据输入线的值（0,1，原变量，反变量）。$F(A,B,C)=\sum m(1,3,5,6)$

输入 C	选择 AB 00	01	10	11
0	0	2	4	6
1	1	3	5	7
D_i				

卡诺图实现表



思考：1什么情况下，数据输入线值为0，什么时候为1？

2在增加逻辑门电路控制的情况下，能否用更少的选择变量来实现布尔函数？

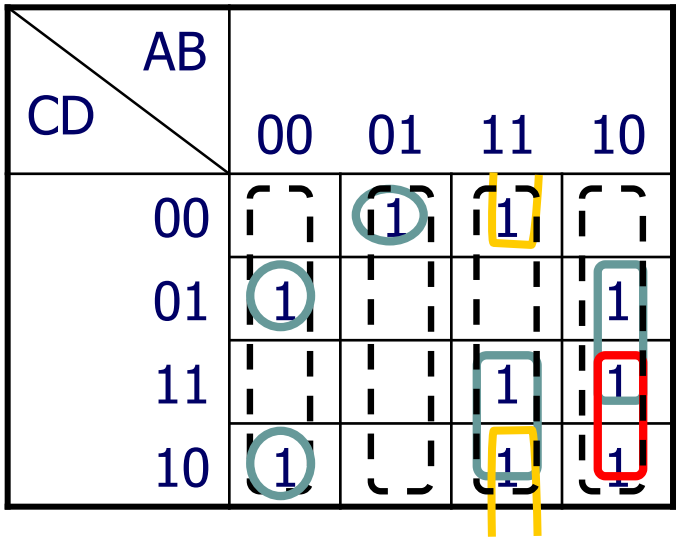
- 例： 用4路选择器实现4变量逻辑函数的功能，函数式为

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

1、选择2个变量作为选择变量，其余两个变量用为数据输入值，以选择变量为基准，画出函数的卡诺图。

2、选择变量的每一列作为子卡诺图，对输入变量进行化简，确定每一种选择情况下的输入值D_i。

3、用基本逻辑门电路实现数据输入值。

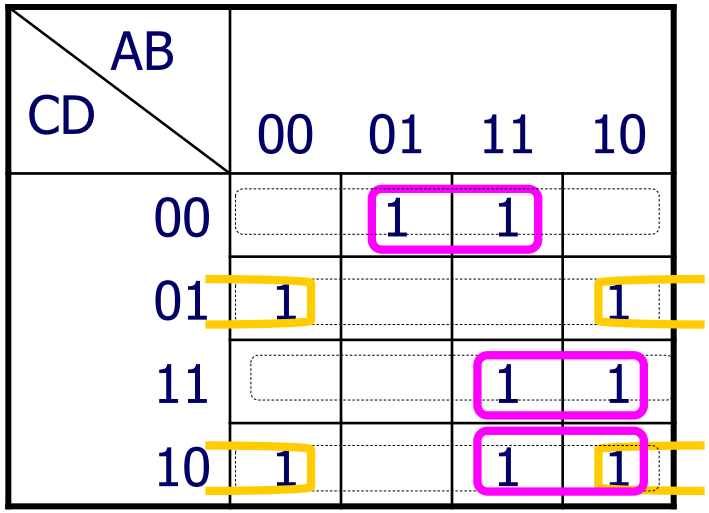


1假设AB为选择输入，则数据输入线的值为：

D0=C'D+CD'=C⊕D　　D1=C'·D'

D2=C+D　　D3=C+D'

2假设CD为选择输入，AB为数据输入，则数据线的值为：



D0=B

D1=B’

D2=A+B’

D3=A

设定不同的选择变量可取得不同导致化简的结果。

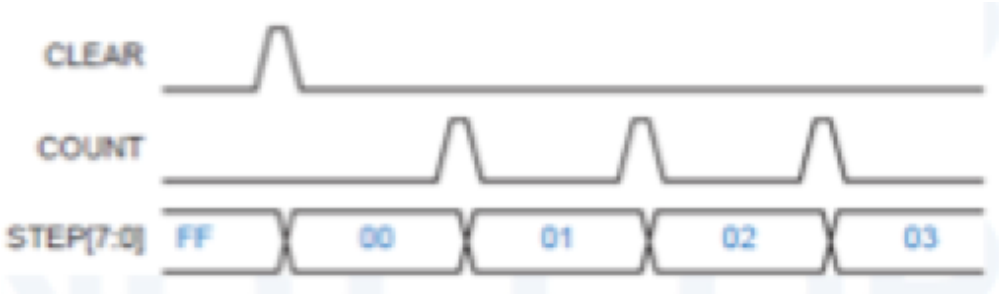
更多变量的函数，先对函数表达式化简，消去变量，得到最简式后，再设置选择变量和数据输入变量。

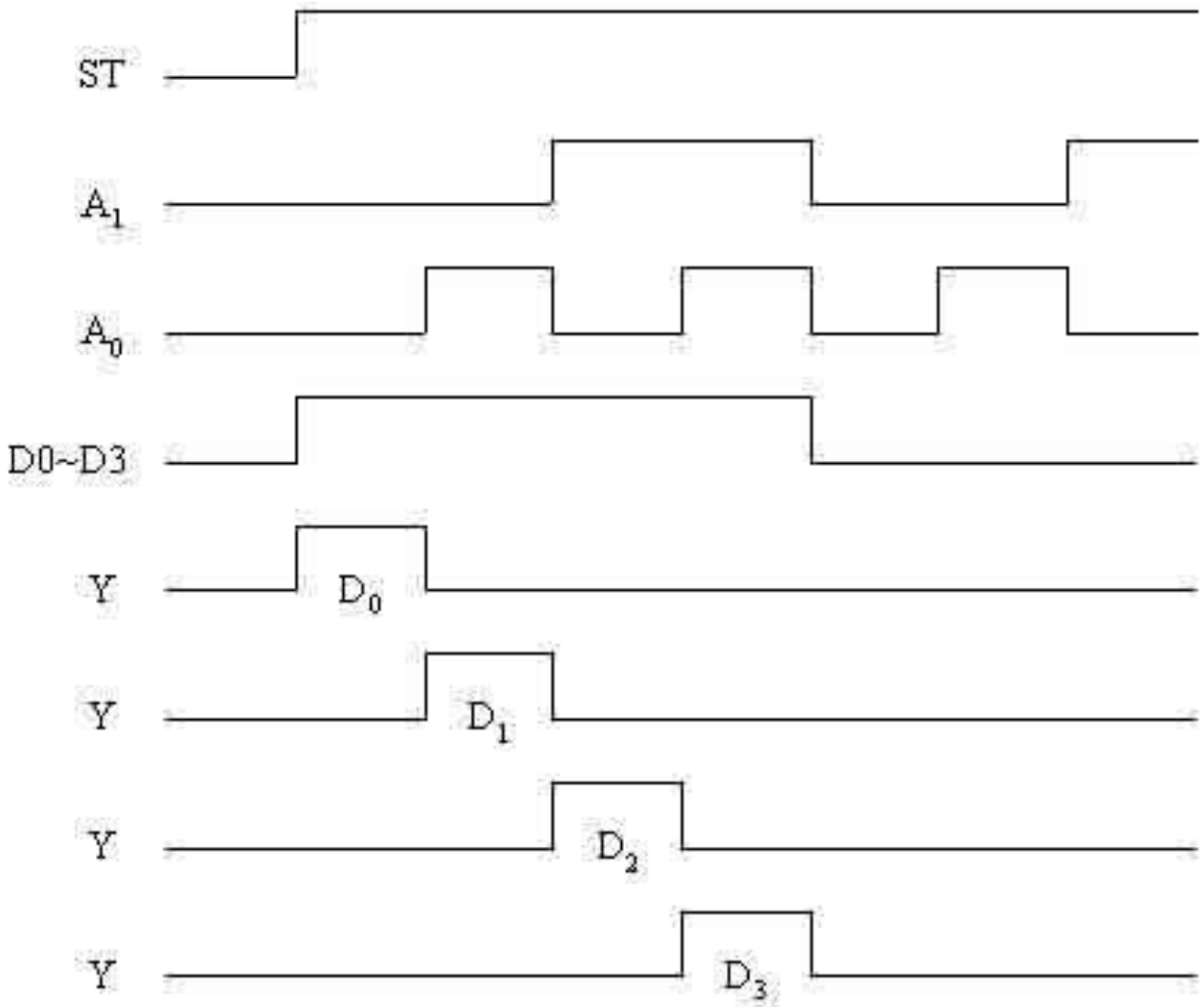
2、数据选择器

- 【例】用4路选择器74LS253构成分时多路转换器
  - 把并行处理的数据放在Di端上。
  - 在地址端上，周期性的循环加载

    00->01->10->11

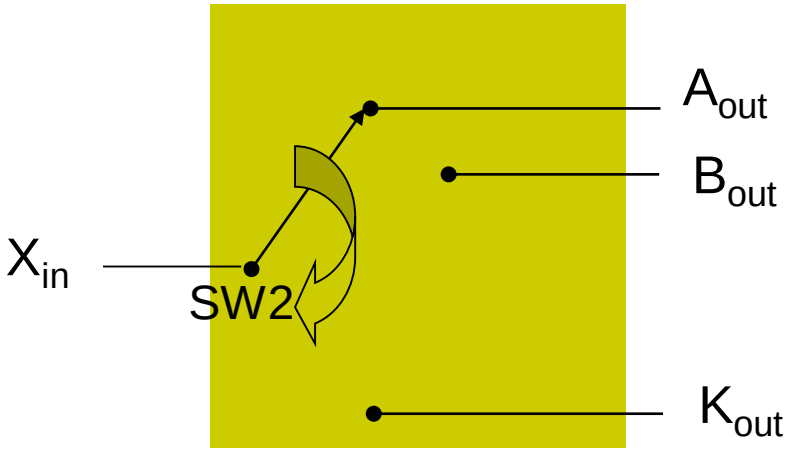
  如此，在输出端上，顺序地送出原先并行的数据。



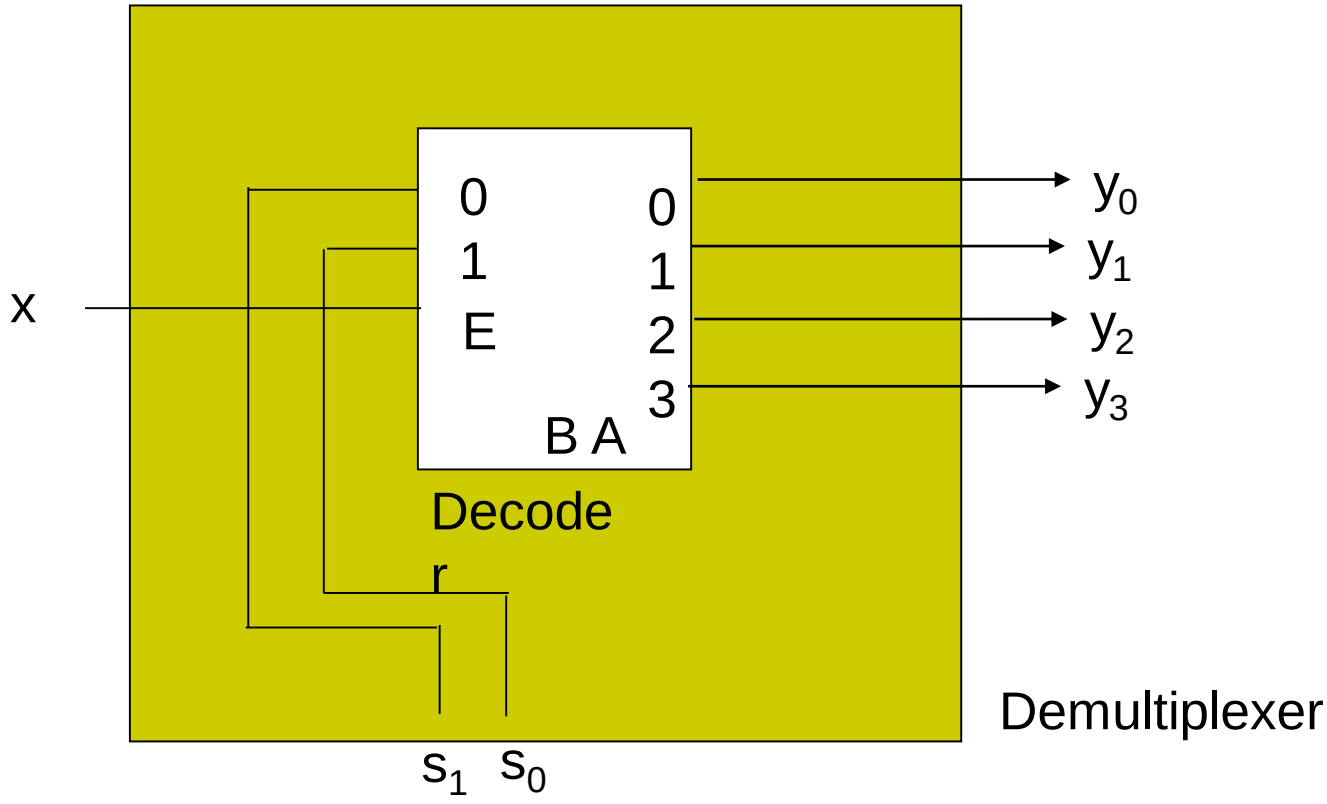

ST
A1
A0
D0~D3
Y
D0
Y
D1
Y
D2
Y
D3

3.5 数据分配器(Demultiplexers)

- 其功能和多路数据选择器相反。
- 是一种单路输入，多路输出的逻辑构件。
  - 从哪一端输出依赖于当时的地址控制端输入。



DEMUX的实现



3.5 数据分配器

- N个输出，需要m个地址变量标识

$$m = \log_2^n$$

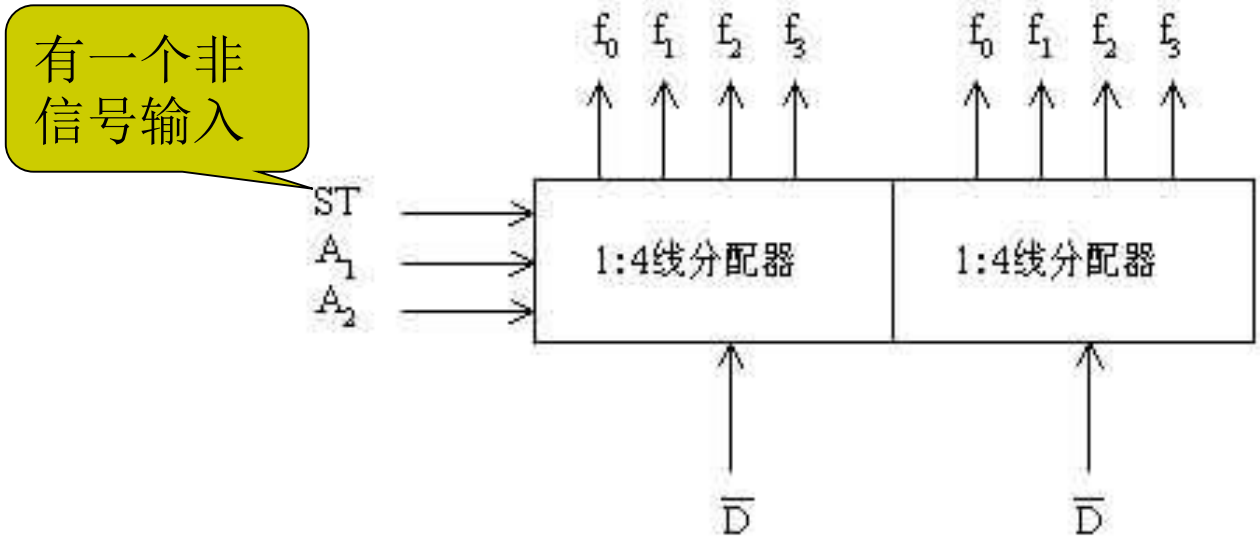
- 每个输出Yi，都是一个最小项表达式和输入变量的乘积。

$$Yi = D\ mi$$

- 逻辑结构: (74LS155)
  - 74LS155是双1：4线数据分配器，结构见框图所示。外部标明了两个独立数据分配器的数据输入、输出线和控制信号线。

3.5 数据分配器

- 当地址输入A1A0=00，且使能控制ST有效时，数据输入发送到f0输出端；当地址输入A1A0=01，且使能控制ST有效时，数据输入发送到f1输出端；依次类推。



3.5 数据分配器

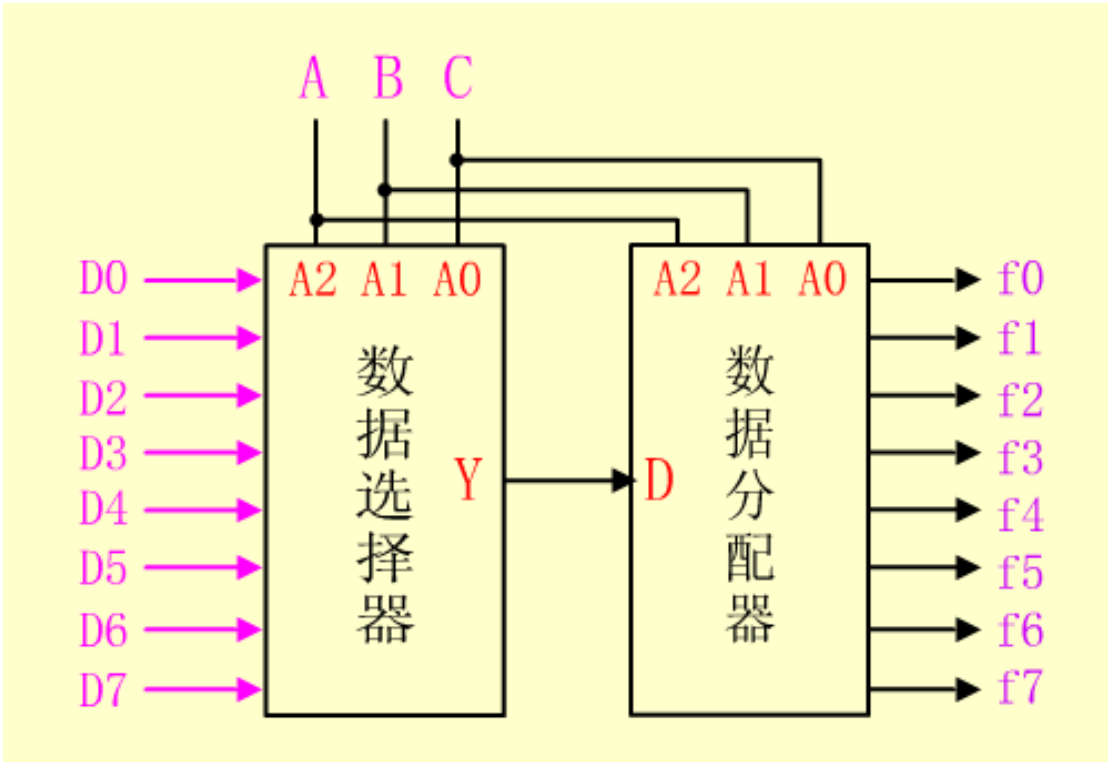
- 功能表

输入			输出				输入			输出			
1ST	$1A_1$	$1A_0$	$1f_0$	$1f_1$	$1f_2$	$1f_3$	$\overline{2ST}$	$2A_1$	$2A_0$	$2f_0$	$2f_1$	$2f_2$	$2f_3$
0	×	×	1	1	1	1	1	×	×	1	1	1	1
1	0	0	1D	1	1	1	0	0	0	2D	1	1	1
1	0	1	1	1D	1	1	0	0	1	1	2D	1	1
1	1	0	1	1	1D	1	0	1	0	1	1	2D	1
1	1	1	1	1	1	1D	0	1	1	1	1	1	2D

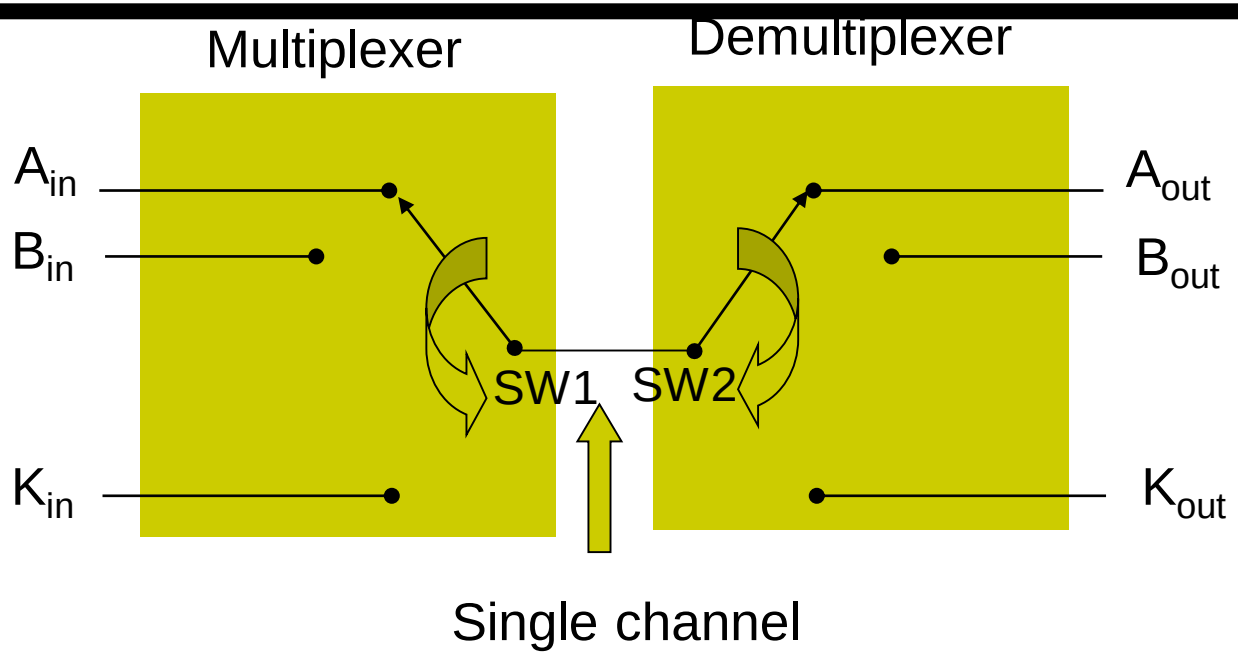
将ST和ST’连在一起作为地址输入端A2，两个数据输入端连在一起作为数据输入，则芯片74LS155可以组成一个1：8线数据分配器。

3.5 数据分配器

- 【例】利用DMUX和MUX设计一个实现8路数据传输的逻辑电路。



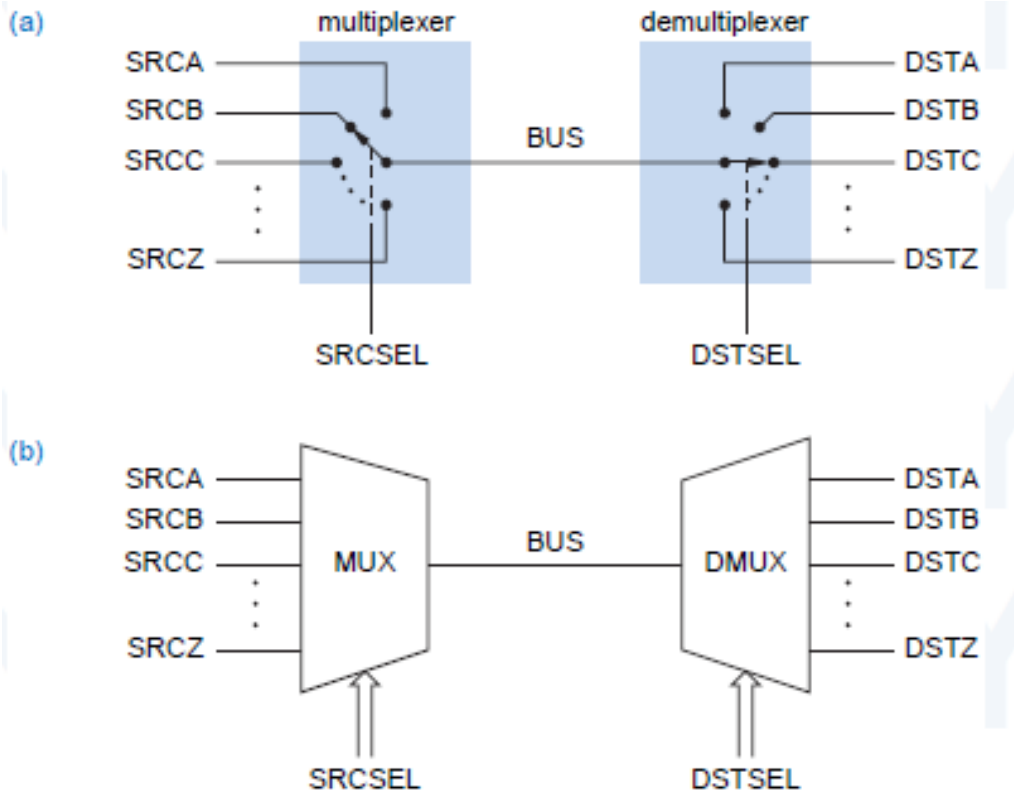
K-channel multiplexing/demultiplexing system



多路分配器常与多路选择器联用，以实现多通道数据分时传送。通常在发送端由MUX将各路数据分时送上公共传输线(总线)，接收端再由DEMUX将公共线上的数据适时分配到相应的输出端。

3.5 多路选择器/分配器的应用

- 大扇出处理
  - 每个输出端增加一个三态缓冲。
- 常用符号
  - a 开关等效
  - b 框图符号



3.5 多路选择器/分配器的应用

- Verilog实现
  - 数据流
  - 行为型

6 用Verilog实现多路复用器

多路复用器很容易用Verilog来描述。在数据流形式中，可以使用一系列条件操作符(

所要求的功能，表6-51是一个4输入8位多路复用器的Verilog模块

在行为结构体中，可以使用一条case语句。例如，表6-52是对同一个mux4in8b实体使用always程序块和case语句的一个模块。

表6-52 4输入8位多路复用器的行为型Verilog模块

```
module Vrmux4in8bc(YOE_L, EN_L, S, A, B, C, D, Y);
  input YOE_L, EN_L;
  input [1:0] S;
  input [1:8] A, B, C, D;
  output [1:8] Y;
  reg [1:8] Y;

 always @ (YOE_L or EN_L or S or A or B or C or D) begin
   if (~YOE_L == 1'b0) Y = 8'bz;
   else 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
```

在Verilog模块中，很容易对选择标准进行定制。例如，表6-53是对一个特定的4输入

复用器的行为型程序，它的选择标准是表6-46。

个mux4in8b实