

第4章 组合逻辑电路

概 述

4.1 组合逻辑电路分析

4.2 组合逻辑电路设计

4.3 组合逻辑电路的冒险现象

数字系统

逻辑电路

组合逻辑电路

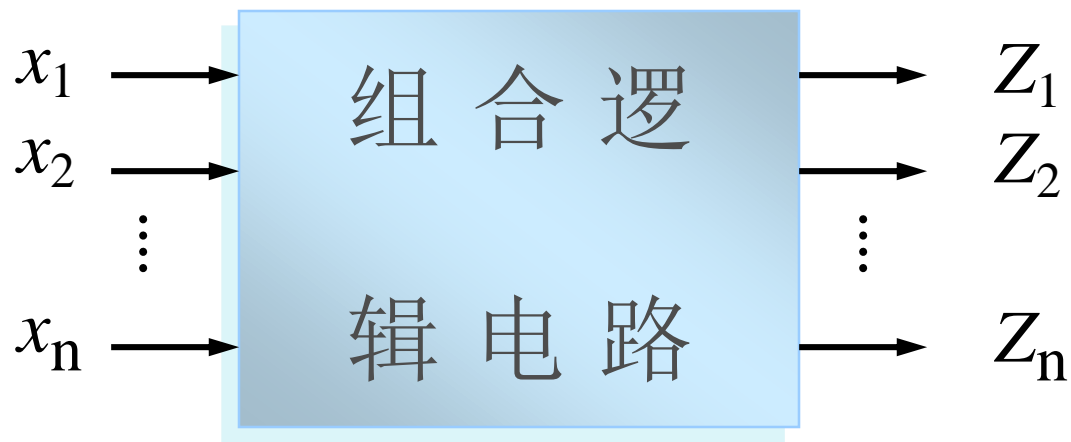
时序逻辑电路

组合逻辑电路的特点

结构特点：基本上由门电路组成；只有从输入端到输出端的直接通路，而没有从输出端到输入端的反馈回路；电路中不包含具有记忆功能的存储元件

逻辑特点：任何时刻电路的输出仅仅取决于该时刻的输入信号，而与这一时刻输入信号作用之前电路原来所处的状态无关

组合逻辑电路的一般框图描述



$$Z_1 = f_1(x_1, x_2, \dots, x_n)$$

$$Z_2 = f_2(x_1, x_2, \dots, x_n)$$

$$\vdots$$

$$Z_n = f_n(x_1, x_2, \dots, x_n)$$

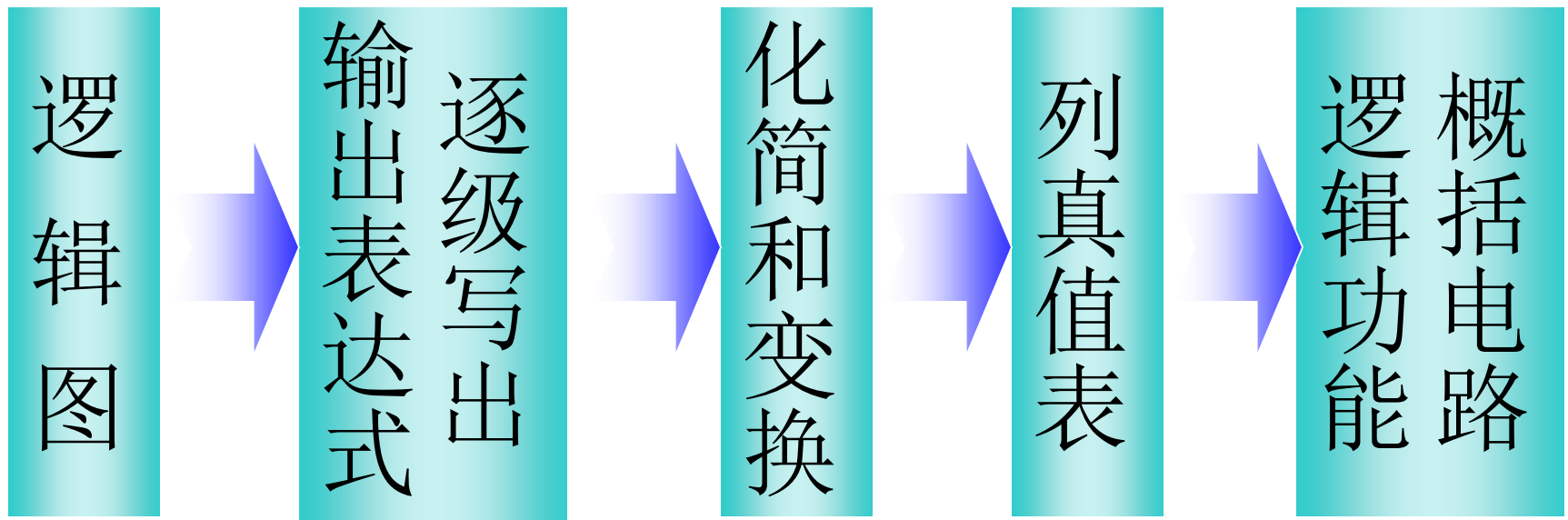
4.1 组合逻辑电路分析

分析目的

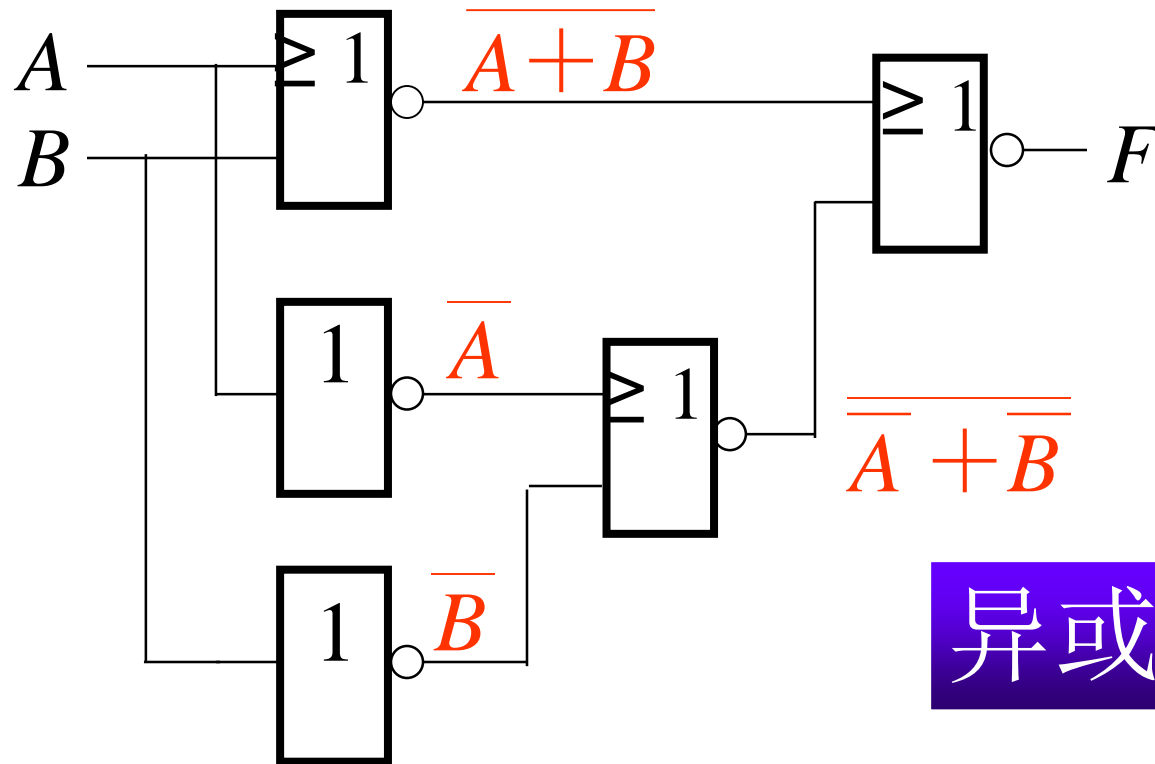
根据给定逻辑电路，找出该电路的逻辑功能
组合逻辑电路分析是建立在逻辑代数基础上的，大部分分析步骤与逻辑函数的运算、化简和变换有关

常见的典型组合逻辑电路有：编码器、译码器、数据选择/分配器、全加器、数值比较器、奇偶产生/校验器等

组合逻辑电路分析的一般步骤



组合逻辑电路分析例

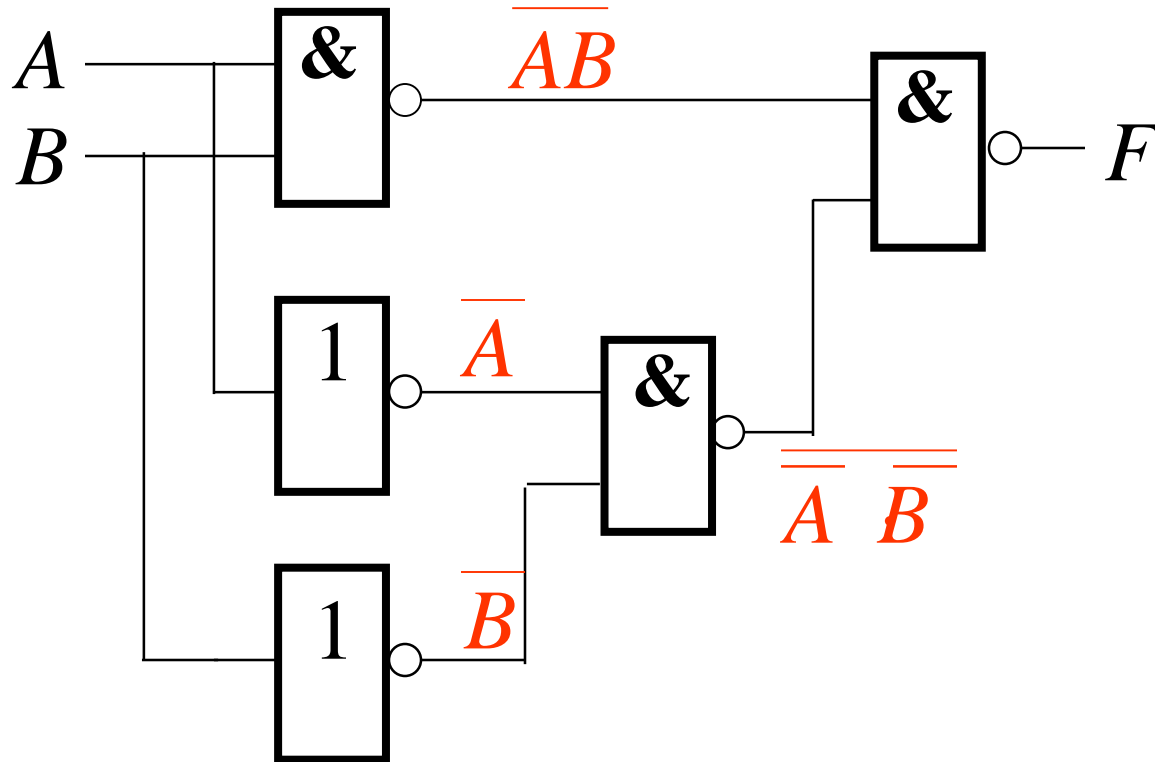


异或逻辑

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

(真值表略)

组合逻辑电路分析例



$$F = \overline{\overline{AB}} \cdot \overline{\overline{\overline{A} \overline{B}}} = AB + \overline{A} \overline{B}$$

(真值表略)

同或逻辑

4.1.1 全加器

加法运算规律：

$$\begin{array}{r} 1\ 1\ 0\ 1 \\ +) 1\ 0\ 0\ 1 \\ \hline 1\ 0\ 0\ 1 \\ 1\ 0\ 1\ 1\ 0 \end{array} \quad \left. \begin{array}{l} \\ \end{array} \right\} \begin{array}{l} \text{加数} \\ \text{进位和} \end{array}$$

逢二进一

各位相加时实际上是两个加数和低位来的进位三个数相加

各位加法运算产生的结果都是本位和和向高位的进位

半加运算和半加器

两个加数和低位来的进位相加求和的运算，称为全加运算。

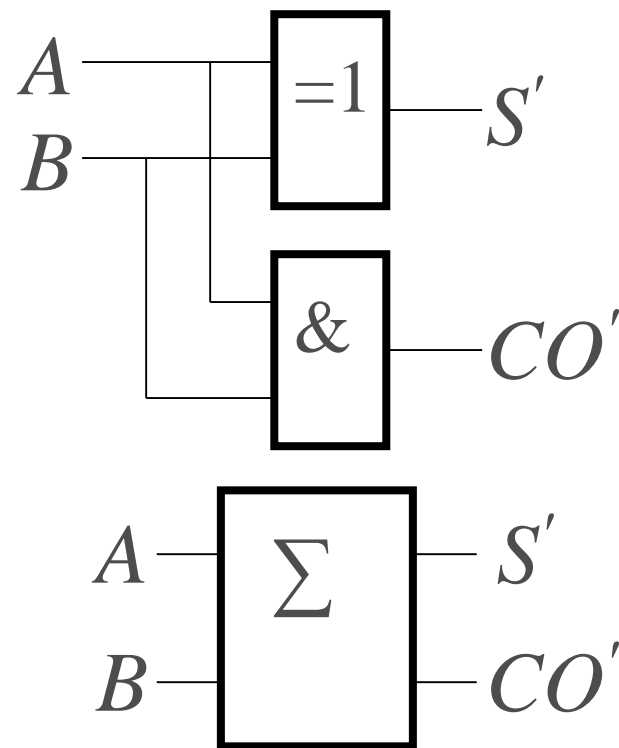
仅仅是两个加数相加求和，不考虑低位进位，称为半加运算。半加运算产生“半加和”和“半加进位”，半加运算是一种过渡性的不完整加法运算。能完成半加运算的电路称为半加器

半加器

列出半加运算的真值表，进而得出半加器的逻辑函数表达式和逻辑图

输 入		输 出	
A	B	S	CO
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$\begin{cases} S' = A\bar{B} + \bar{A}B = \underline{A \oplus B} \\ CO' = AB \end{cases} \quad \text{异或}$$



半加器逻辑符号

全加器 (考虑进位!)

全加器的真值表

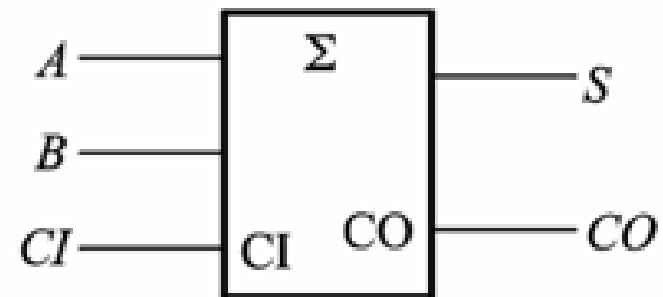
CI_{i-1}	A_i	B_i	S_i	CO_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

全加器的表达式

$$S_i = A_i \oplus B_i \oplus CI_{i-1}$$

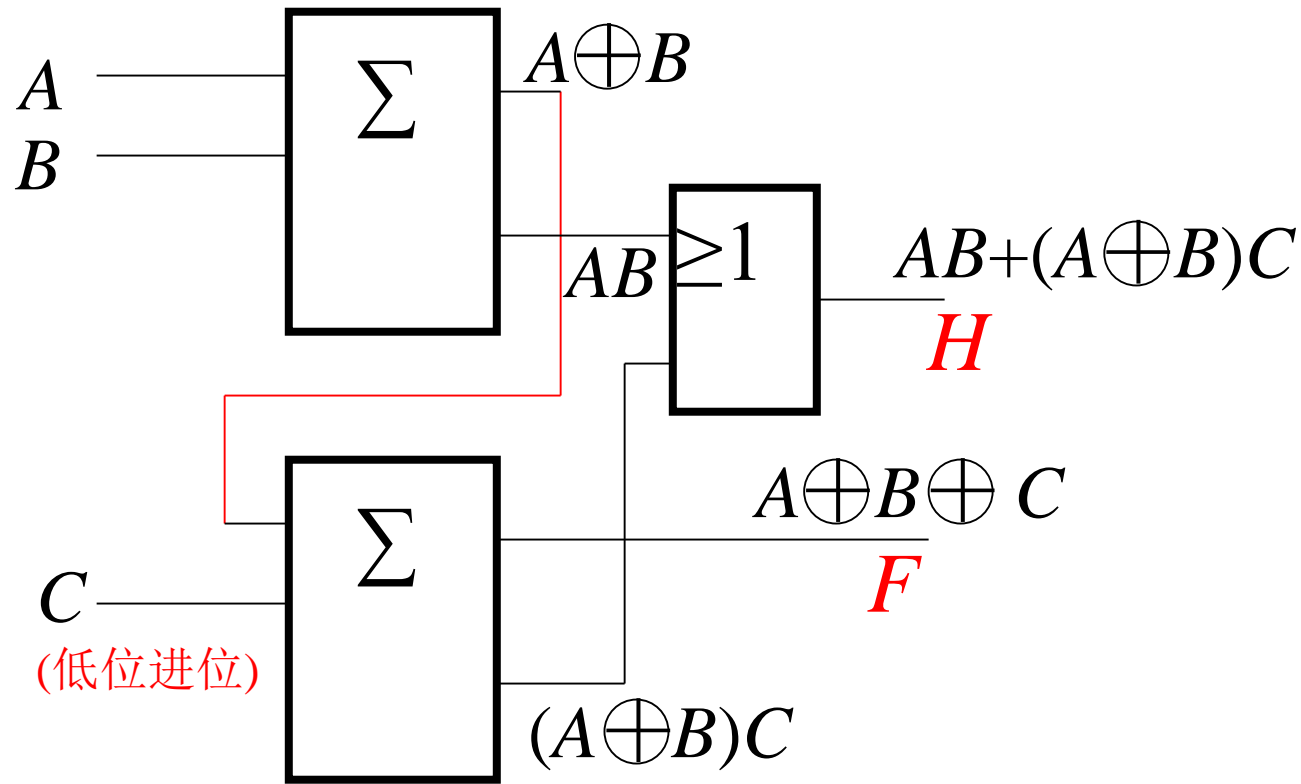
$$CO_i = A_i B_i + (A_i \oplus B_i) CI_{i-1}$$

$$= A_i B_i + A_i \bar{B}_i CI_{i-1} + \bar{A}_i B_i CI_{i-1}$$



一位全加器的逻辑符号

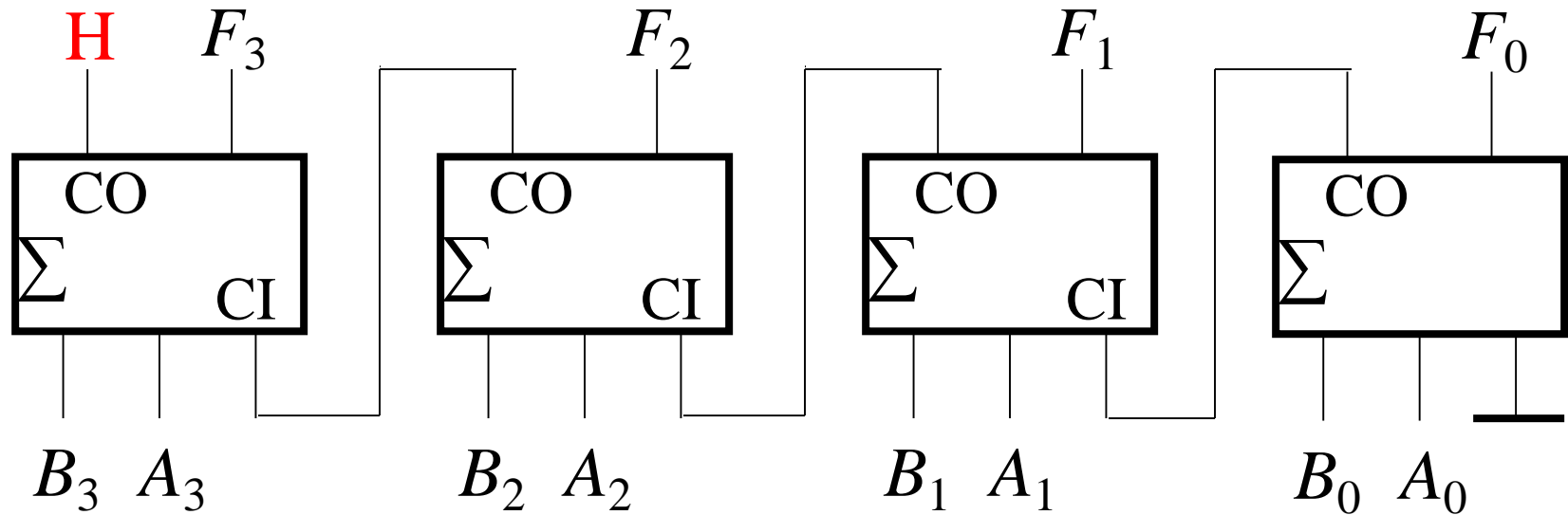
用半加器构成全加器



本位和 $F = A \oplus B \oplus C$

高位进位 $H = AB + (A \oplus B)C$


逐位进位全加器(串行进位加法器)



实现无低位进位的四位二进制数加法： $A+B$

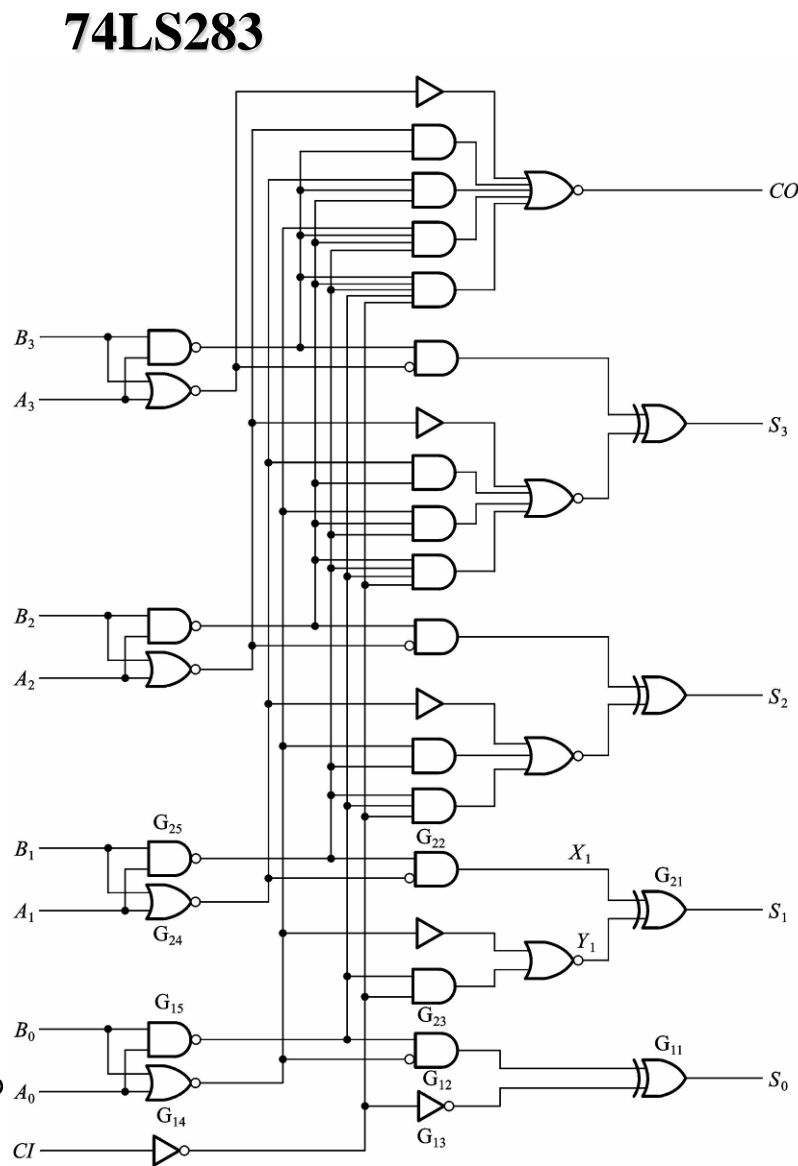
逐位进位加法器结构简单、便于理解，但由于逐级运算需要消耗许多时间，因此运算速度较慢，实用的全加器是超前进位全加器

超前进位全加器

基本原理：加到第 i 位的进位输入信号是**两个加数第 i 位以前各位**（ $0 \sim i-1$ ）的函数，可在相加前由A, B两数确定。

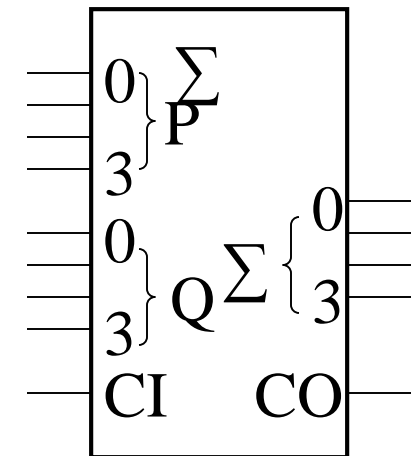
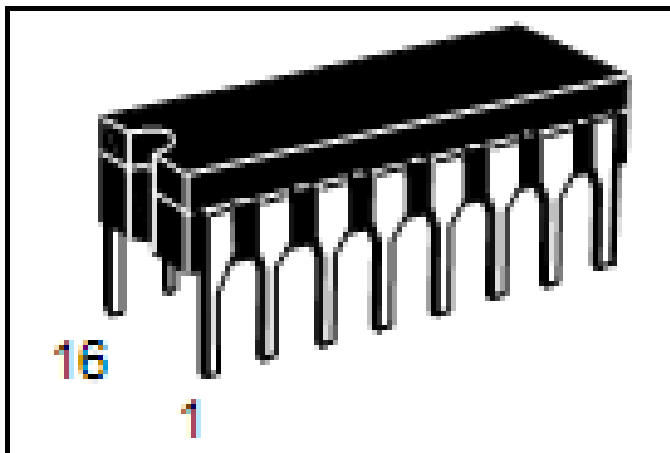
优点：快，每1位的**和**及最后的**进位**基本同时产生。

缺点：电路复杂。



超前进位全加器

常用的中规模超前进位全加器集成电路有：
CT54/74283, CT54S/74283,
CT54LS/74LS283, CC4008
等



4位全加器逻辑符号

4.1.2 编码器

赋予每个二元码序列一个固定的含义，称为编码。能够实现编码操作的电路称为编码器。编码器的作用是将一系列信号状态转换成二进制代码

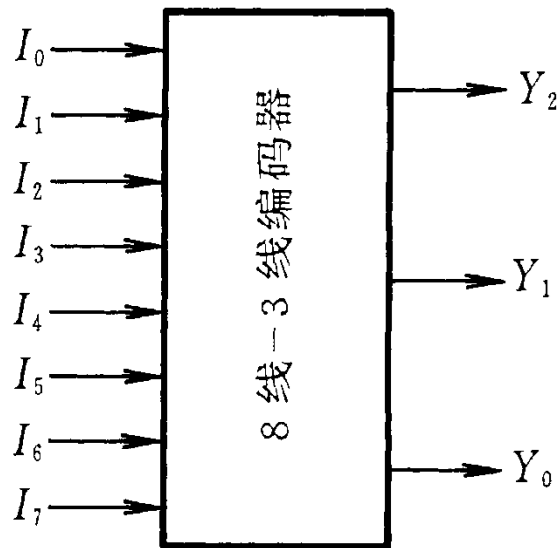
如果需要编码的信息量是 N ，二进制代码的码长是 n 位，则应满足关系：

$$2^n \geq N \geq 2^{n-1}$$

常见的编码器有普通编码器和优先编码器

一、普通编码器

- 特点：任何时刻只允许输入一个编码信号。
- 例：3位二进制普通编码器



输 入								输 出		
I ₀	I ₁	I ₂	I ₃	I ₄	I ₅	I ₆	I ₇	Y ₂	Y ₁	Y ₀
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

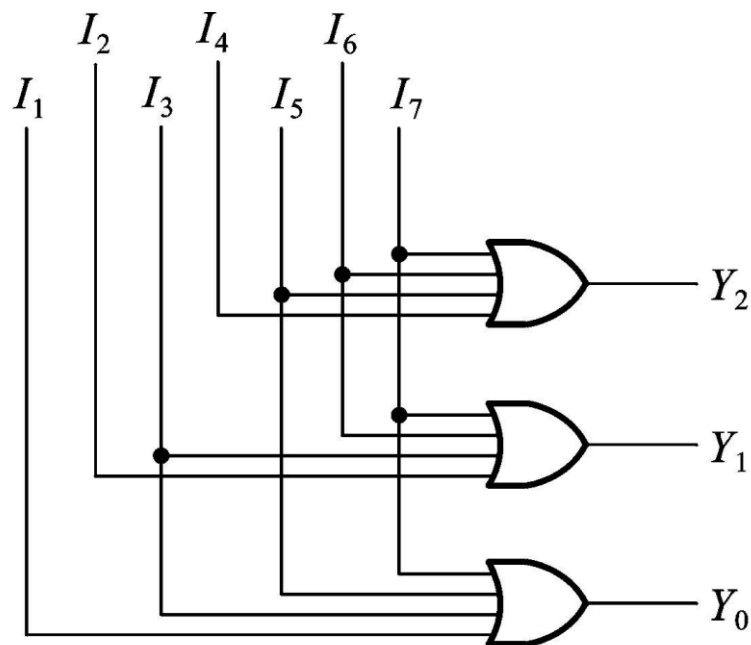
$$\begin{aligned}
 Y_2 = & \overline{I_7} \overline{I_6} \overline{I_5} \overline{I_4} \overline{I_3} \overline{I_2} \overline{I_1} I_0 + \overline{I_7} \overline{I_6} \overline{I_5} \overline{I_4} I_3 \overline{I_2} \overline{I_1} I_0 \\
 & + \overline{I_7} \overline{I_6} \overline{I_5} \overline{I_4} I_3 I_2 \overline{I_1} I_0 + \overline{I_7} \overline{I_6} \overline{I_5} \overline{I_4} I_3 I_2 I_1 I_0
 \end{aligned}$$

利用无关项化简，得：

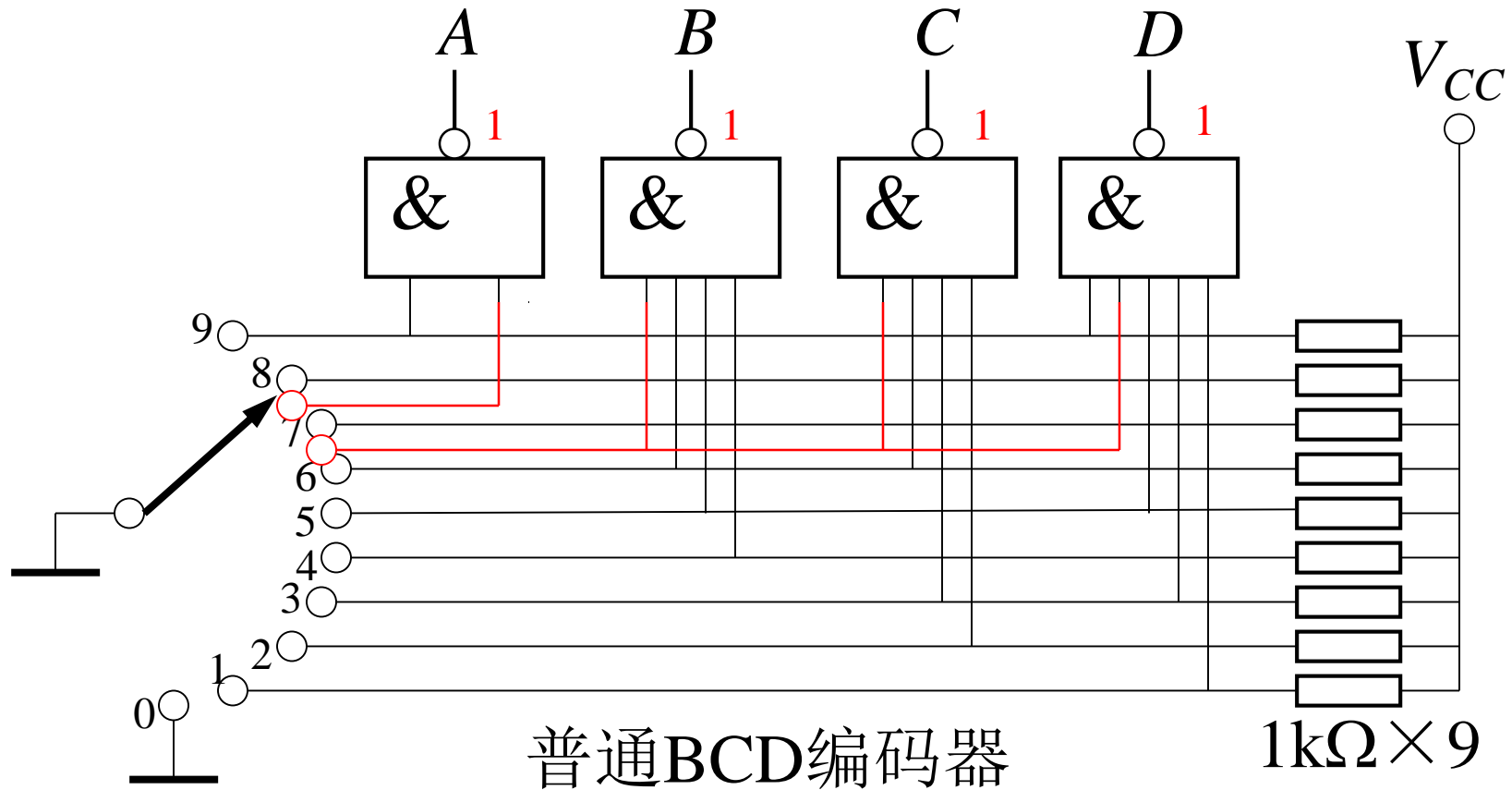
$$Y_2 = I_4 + I_5 + I_6 + I_7$$

$$Y_1 = I_2 + I_3 + I_6 + I_7$$

$$Y_0 = I_1 + I_3 + I_5 + I_7$$



任何时刻只允许输入一个编码信号



当开关切换时，可能出现有两个输入同时要求编码，编码器将输出错码。

二、优先编码器 (HPRI/BIN)

优先编码器允许同时输入两个以上的编码信号，但只对其中优先权最高的一个进行编码，而对其它输入不作任何响应。（以下介绍的优先编码器是以输入端的下标编号数值最大的优先级别最高）

常用中规模的优先编码器有：8线—3线优先编码器CT54/74148、CT54LS/74LS148、CC4532，10线—4线优先编码器CT54/74147、CT54LS/74LS147、CC40147等

例：8线-3线优先编码器

$$Y_2 = I_7 + \overline{I_7} I_6 + \overline{I_7} \overline{I_6} I_5 + \overline{I_7} \overline{I_6} \overline{I_5} I_4$$

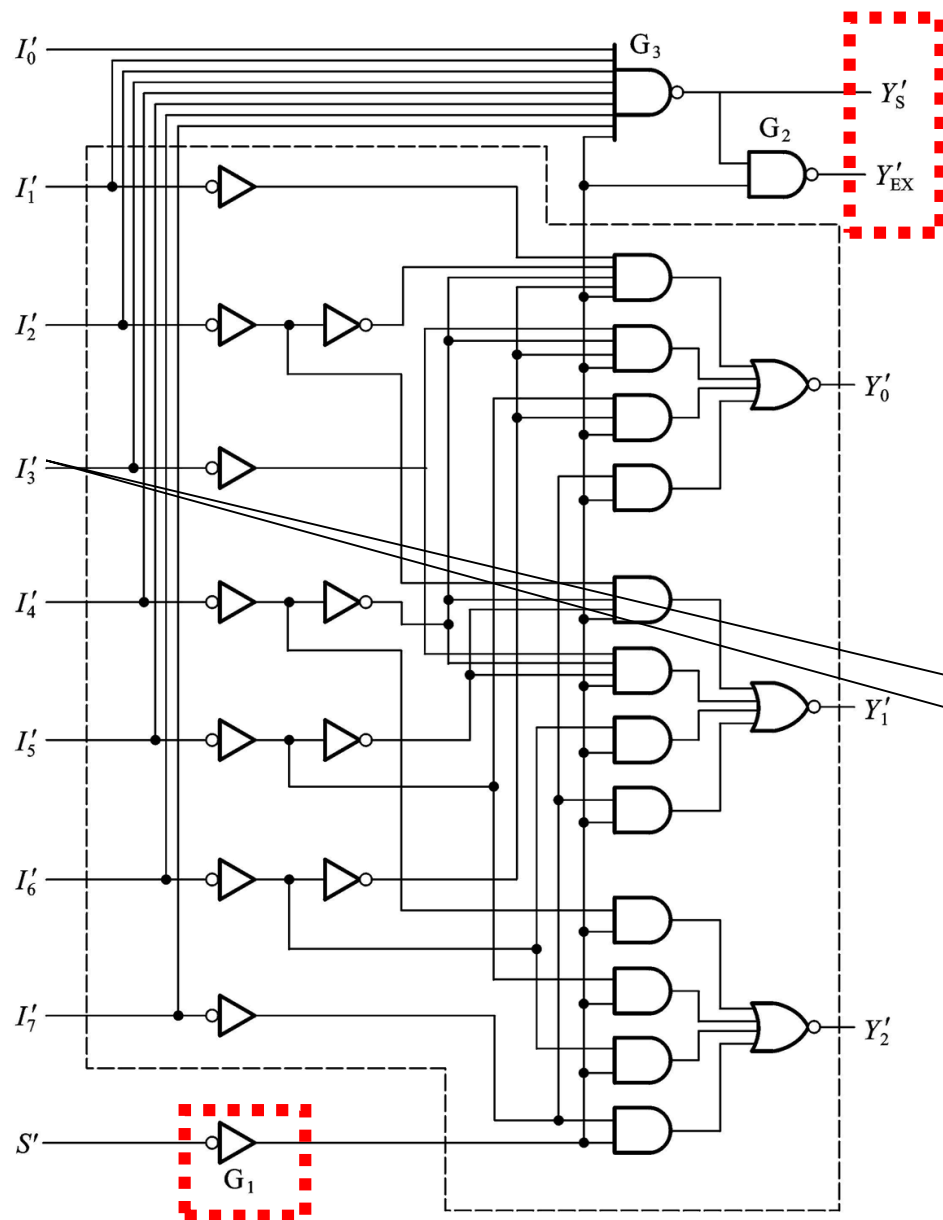


$$\mathbf{Y}_2 = \mathbf{I}_7 + \mathbf{I}_6 + \mathbf{I}_5 + \mathbf{I}_4$$

$$A + \overline{A} B = A + B$$

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

实例： 74HC148



低电平

$$Y_2' = (I_7 + I_6 + I_5 + I_4)'$$

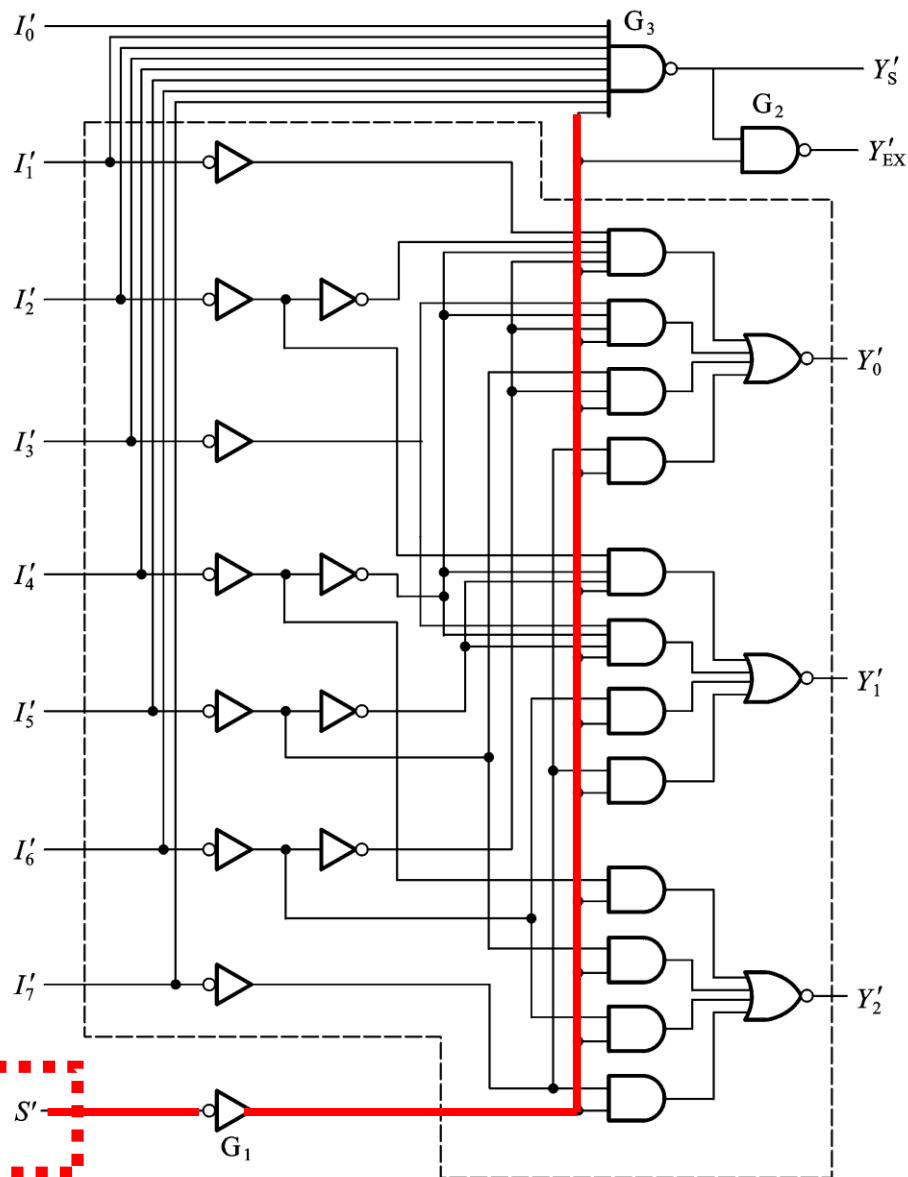
选通信号

$$Y_2' = [(I_7 + I_6 + I_5 + I_4)S]'$$

$$Y_2' = [(I_7 + I_6 + I_5 + I_4)S]'$$

$$Y_1' = [(I_7 + I_6 + I_5 I_4' I_3' + I_2 I_4' I_5')S]'$$

$$Y_0' = [(I_7 + I_6 I_5 + I_3 I_4' I_6' + I_1 I_2 I_4' I_6')S]'$$



选通信号

附加输出信号

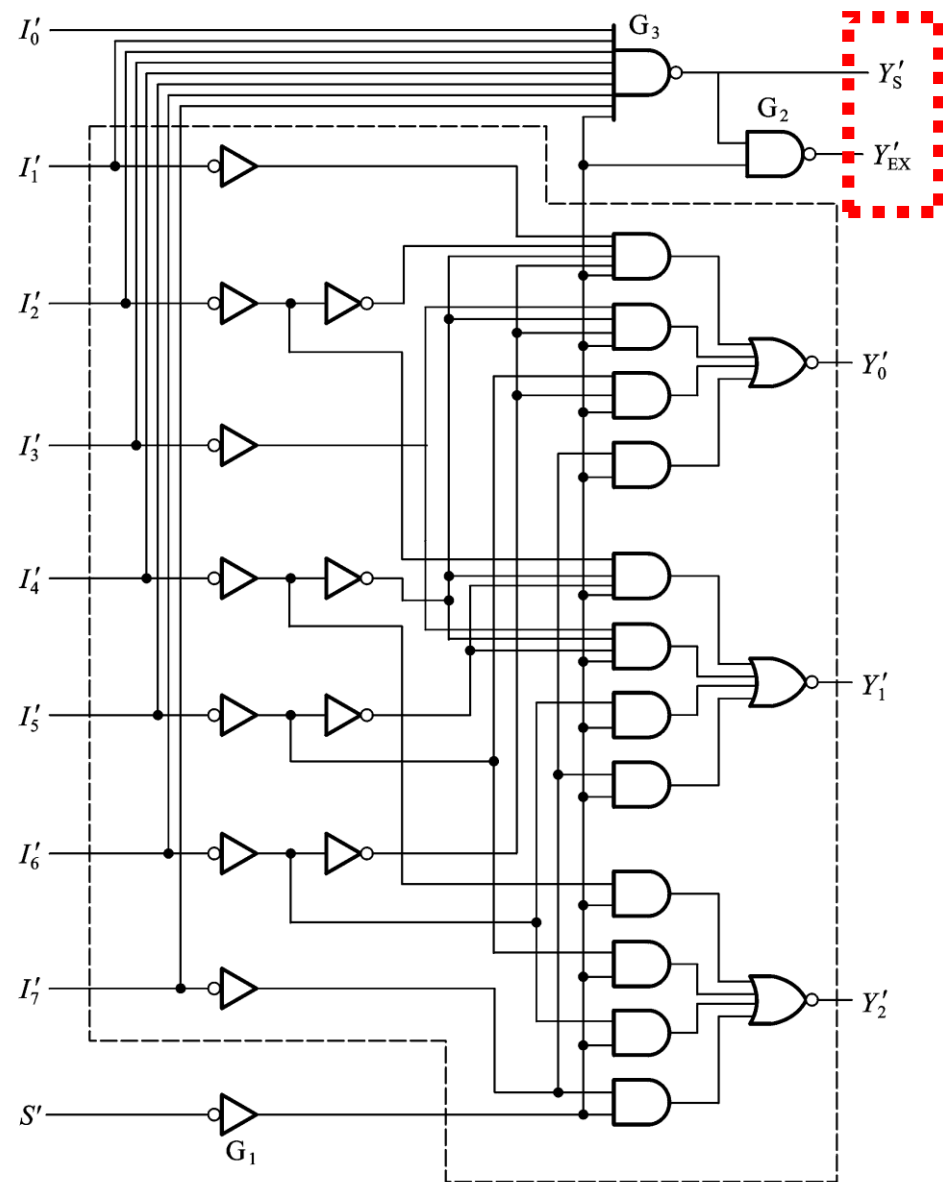
$$Y'_S = (I'_7 I'_6 I'_5 I'_4 I'_3 I'_2 I'_1 I'_0 S)'$$

$$Y'_{EX} = [(I'_7 I'_6 I'_5 I'_4 I'_3 I'_2 I'_1 I'_0 S)' S']$$

$$= [(I_7 + I_6 + I_5 + I_4 + I_3 + I_2 + I_1 + I_0) S']$$

Y's	Y'EX	状态
1	1	不工作(S'=1)
0	1	工作，但无输入
1	0	工作，且有输入
0	0	不可能出现

解决了当输出全为零时，是否有编码输入的问题。



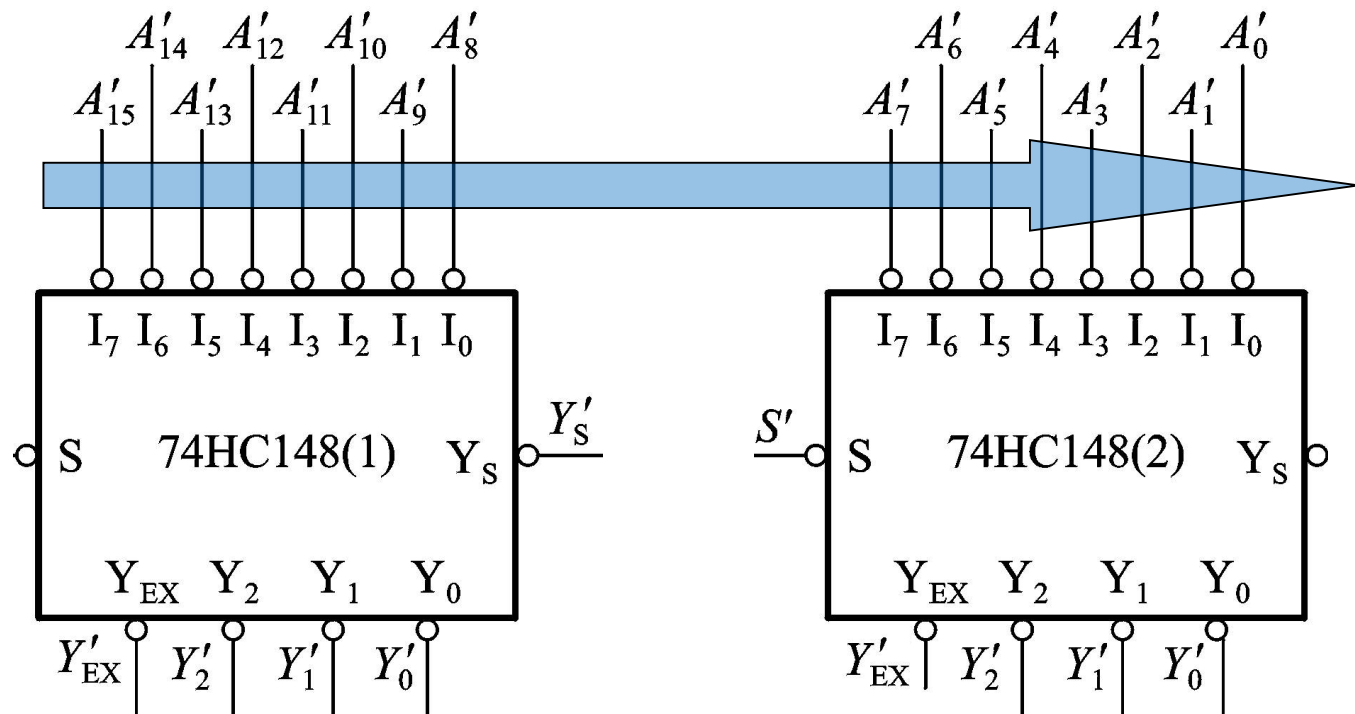
控制端扩展功能举例：

➤ 例： 用两片8线-3线优先编码器

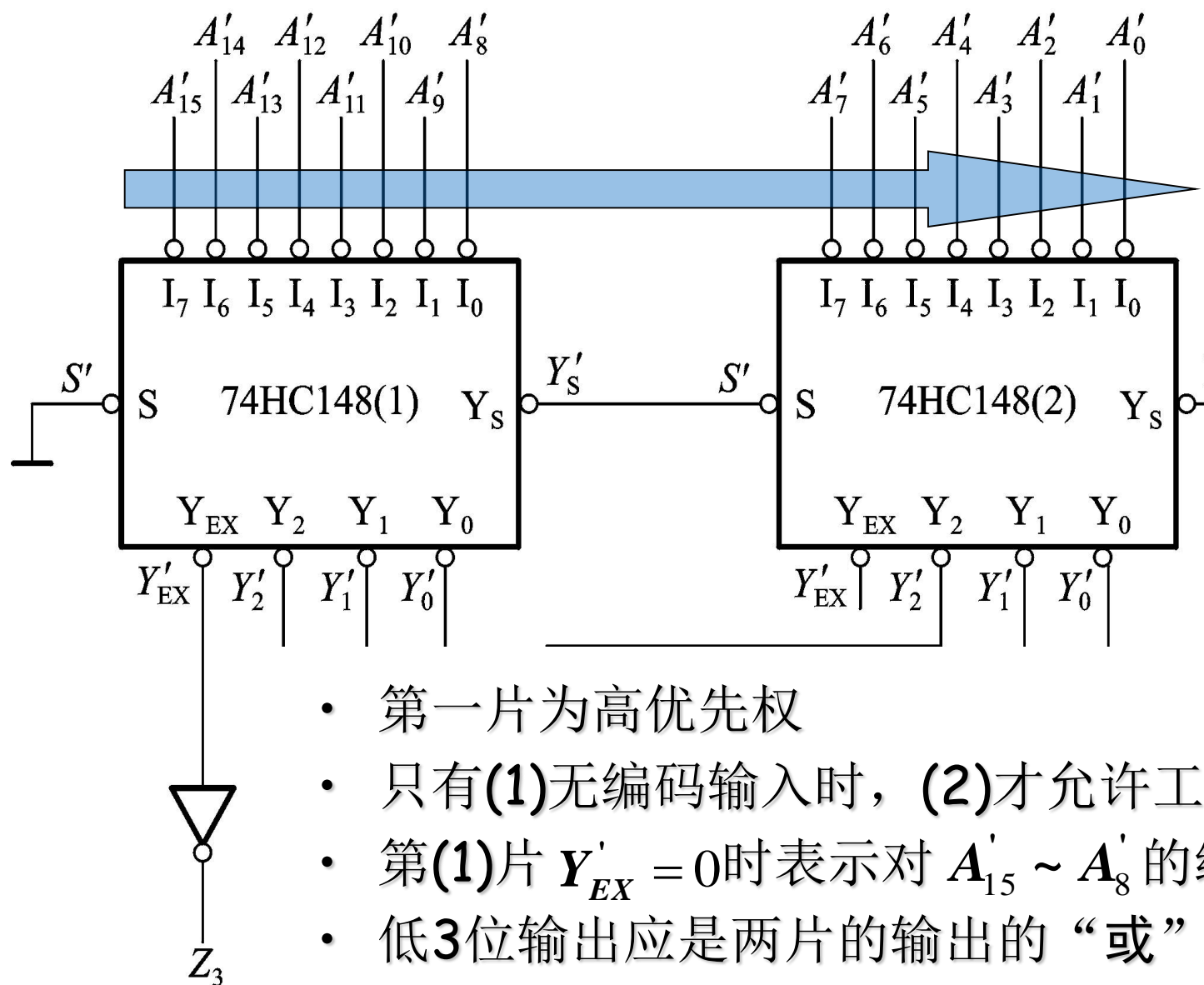


16线-4线优先编码器

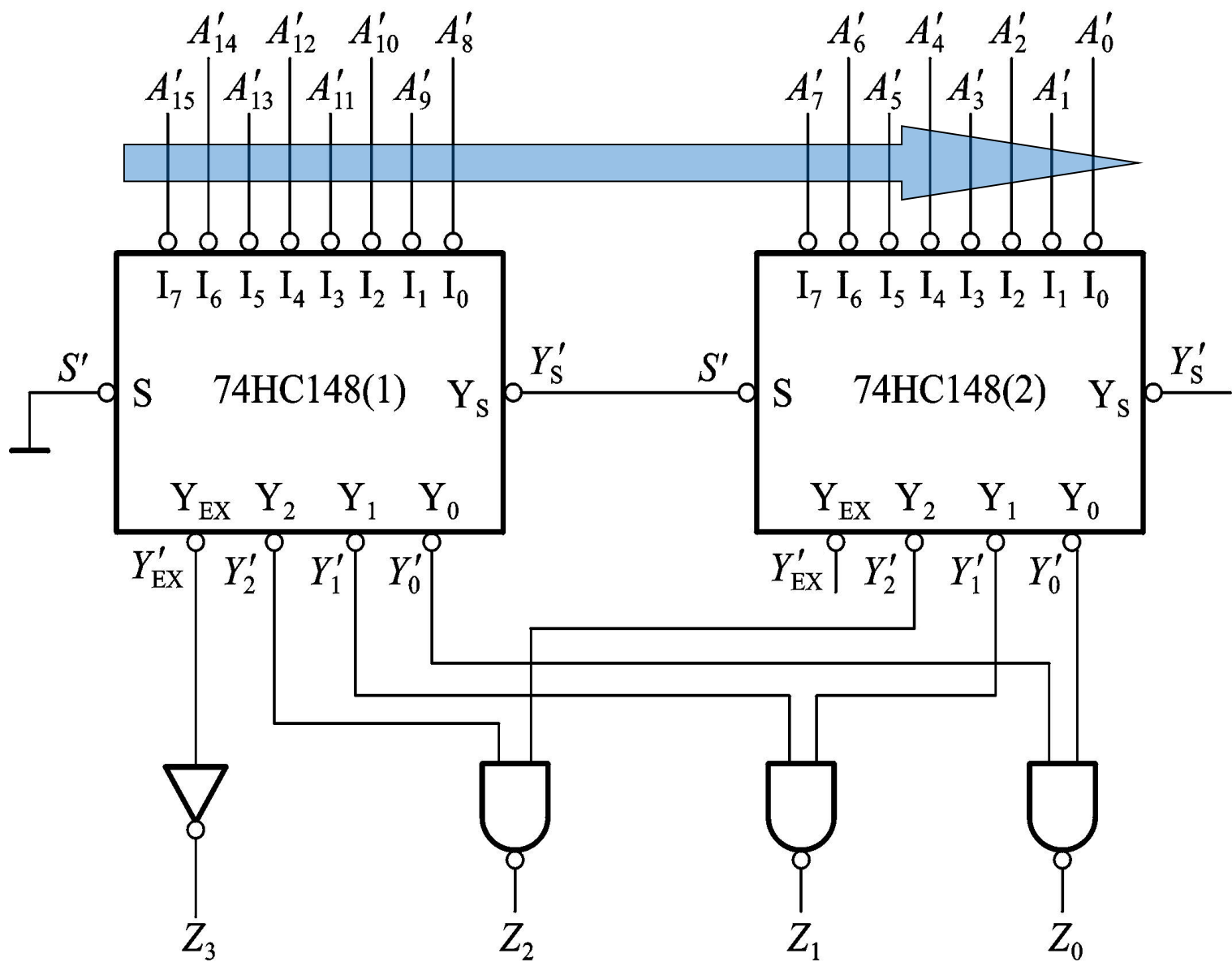
其中， A'_{15} 的优先权最高 · · ·

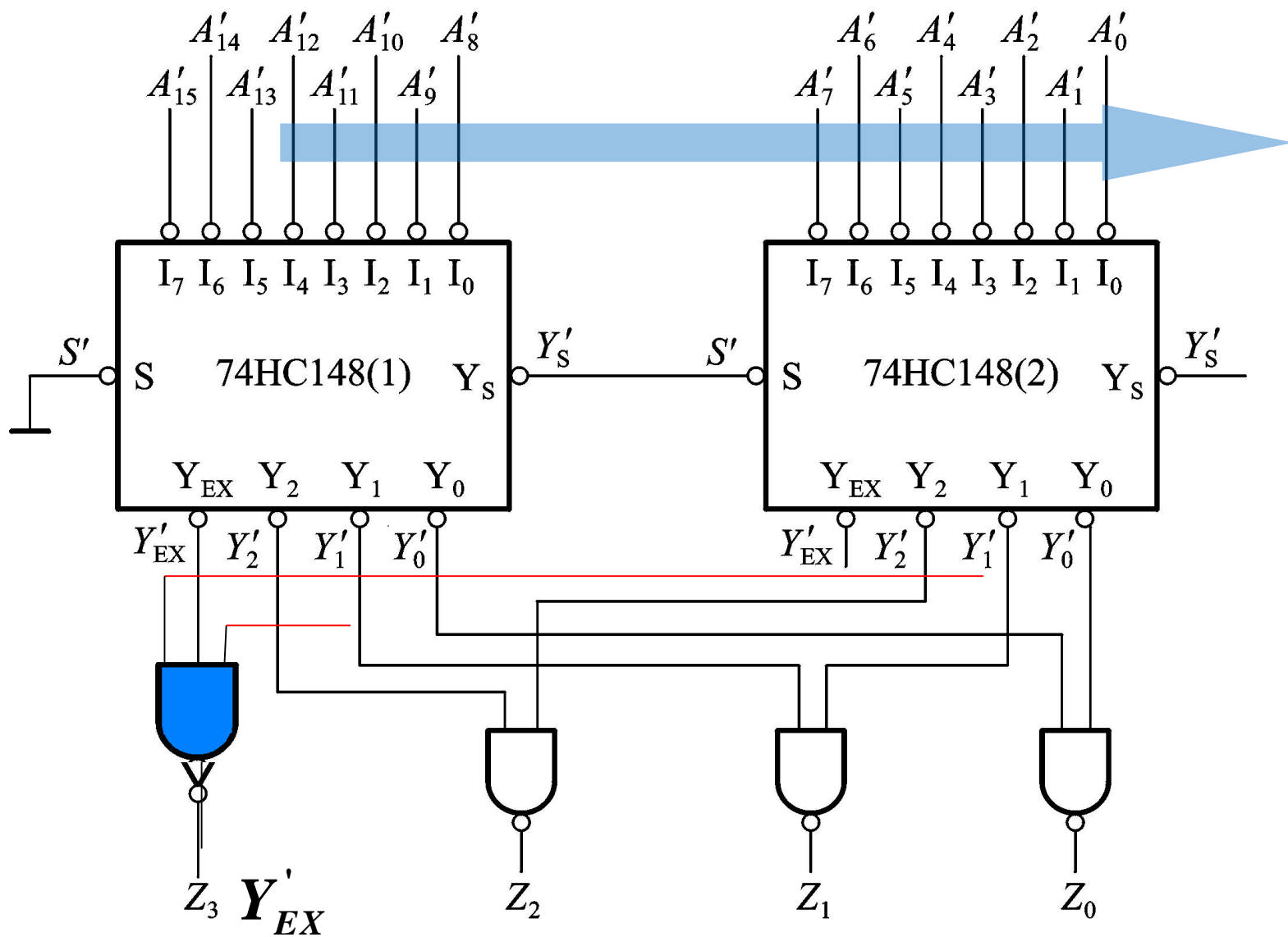


$Y's$	Y'_{EX}	状态
1	1	不工作($S'=1$)
0	1	工作, 但无输入
1	0	工作, 且有输入
0	0	不可能出现



- 第一片为高优先权
- 只有(1)无编码输入时，(2)才允许工作
- 第(1)片 $Y'_{EX} = 0$ 时表示对 $A'_{15} \sim A'_8$ 的编码
- 低3位输出应是两片的输出的“或”





是否是编码输出

4.1.3 译码器

译码是编码的逆过程，即将编码时赋予每个二进制代码原来的含义“翻译”出来，在相应的输出端以事先规定的电平输出

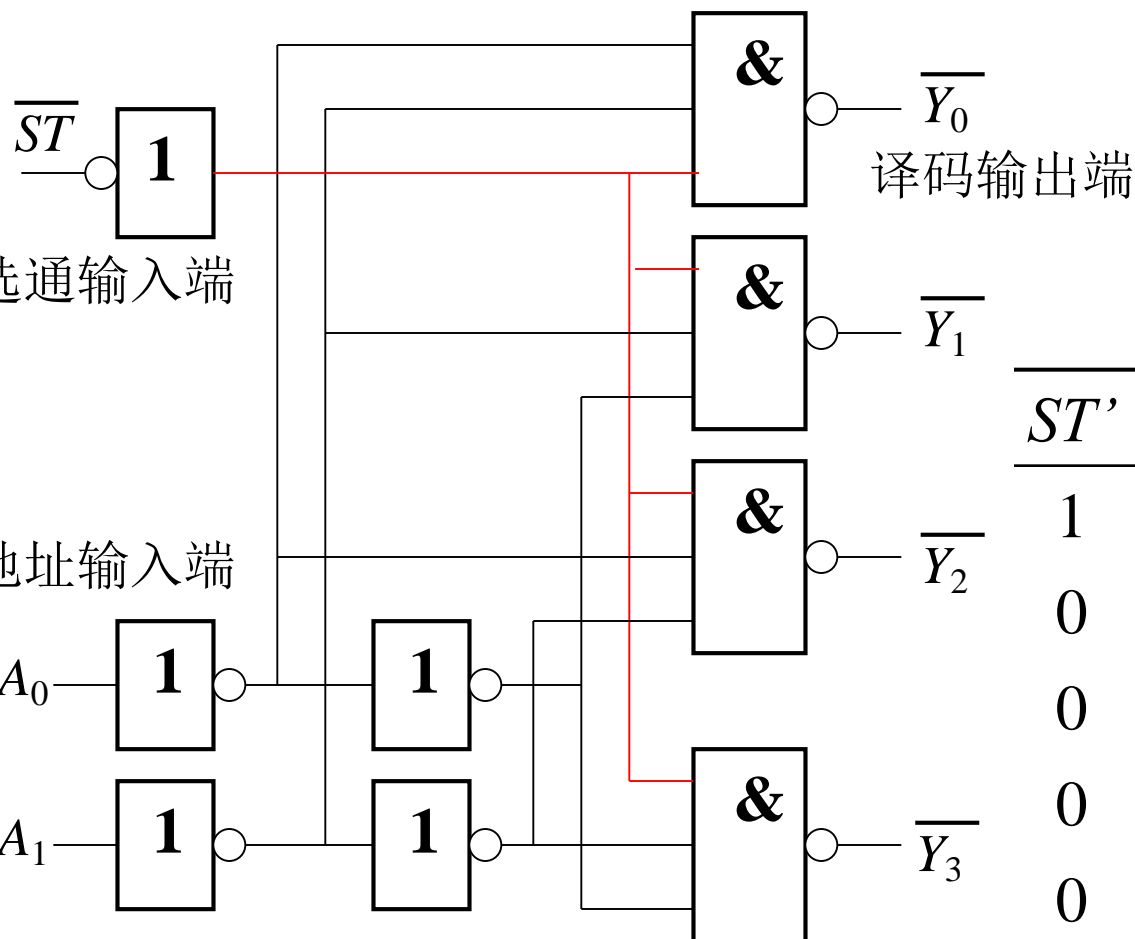
常见的译码器有：二进制译码器(变量译码器)、二—十进制译码器、显示译码器等

常用中规模集成译码器有：双2线—4线译码器CT54S/74S139、CT54LS/74LS139、3线—8线译码器CT54S/74S138、CT54LS/74LS138、CC74HC138，4线—16线译码器CT54/74154、CT54LS/74LS154、CC74HC154，4线—10线译码器CT54/7442、CT54S/74S42、CT54LS/74LS42等

2线—4线译码器(BIN/OCT)

$$\left\{ \begin{array}{l} \overline{Y_0} = \overline{\overline{A_1} \overline{A_0} \overline{ST}} \\ \overline{Y_1} = \overline{\overline{A_1} A_0 \overline{ST}} \\ \overline{Y_2} = \overline{A_1 \overline{A_0} \overline{ST}} \\ \overline{Y_3} = \overline{A_1 A_0 \overline{ST}} \end{array} \right.$$

ST'	A_1	A_0	Y'_3	Y'_2	Y'_1	Y'_0
1	×	×	1	1	1	1
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

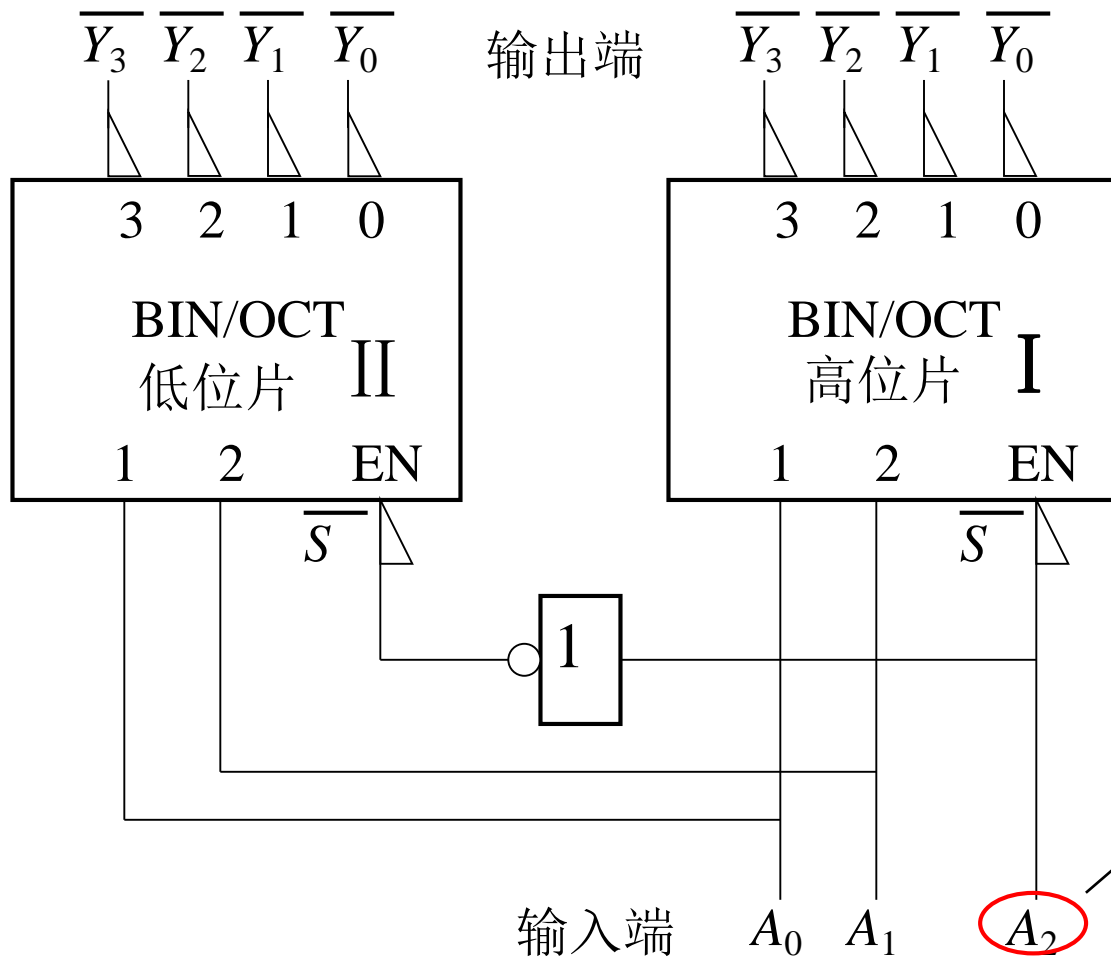


2线—4线译码器逻辑图



2线—4线译码器扩展应用

4.1.3 译码器

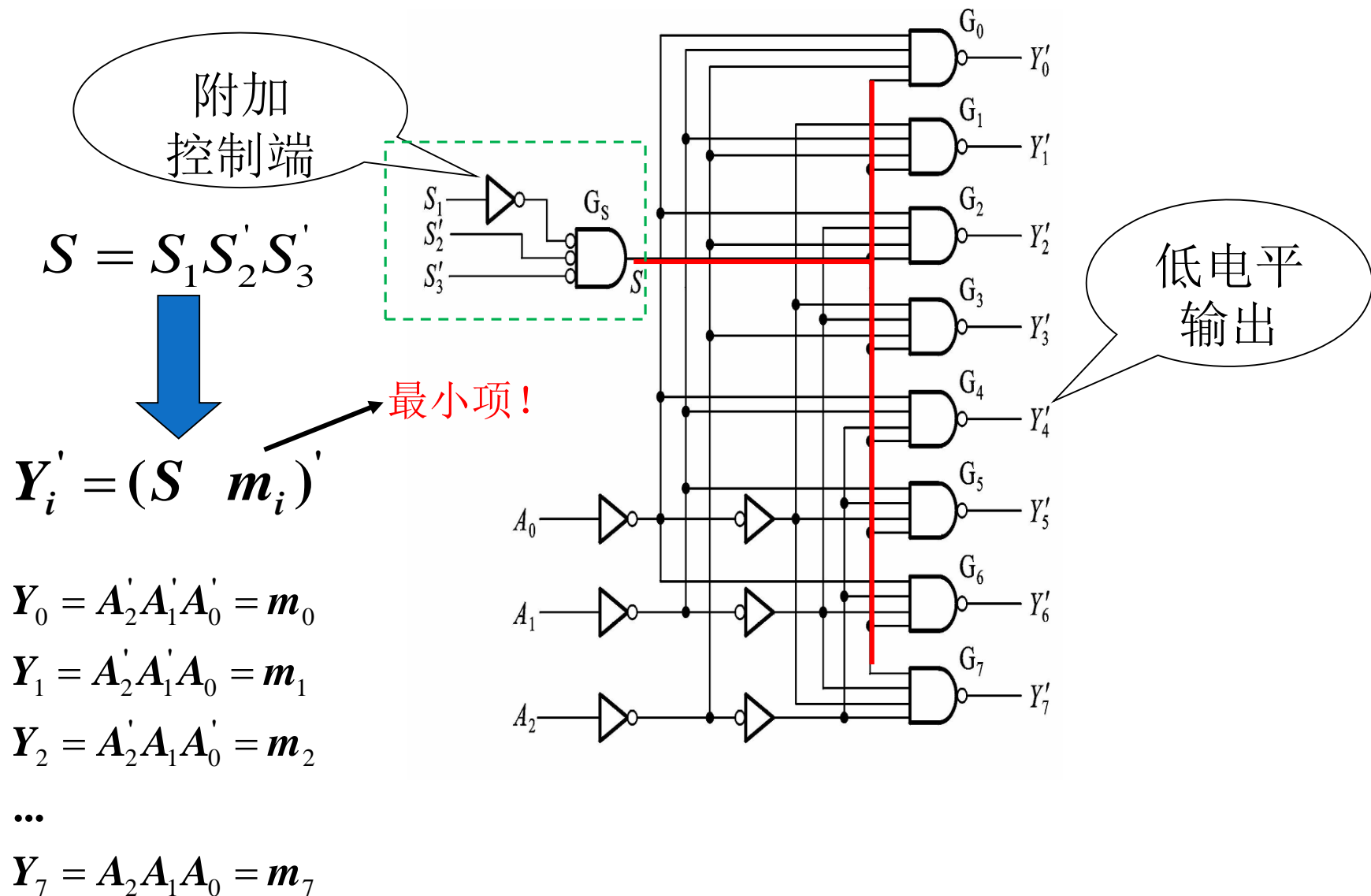


A2	A1	A0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

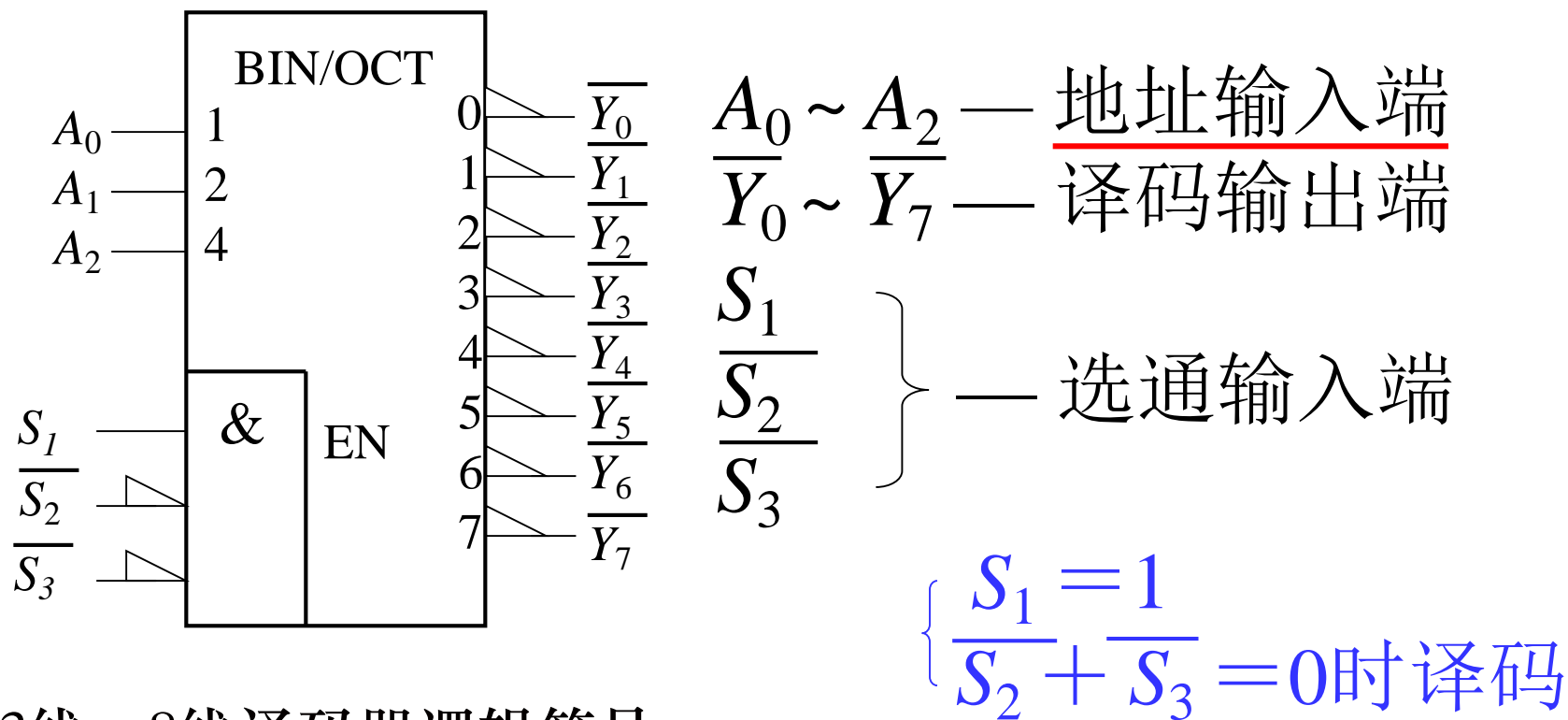
2线—4线译码器扩展构成3线—8线译码器

CT54/74138

3线-8线译码器(BIN/OCT)



3线—8线译码器(BIN/OCT)



3线—8线译码器逻辑符号

如果把 S_1 作为数据输入端，同时令 $S_2' = S_3' = 0$ ，那么从 S_1 来的数据只能通过由 $A_2A_1A_0$ 所指定的那一根输出线送出去。

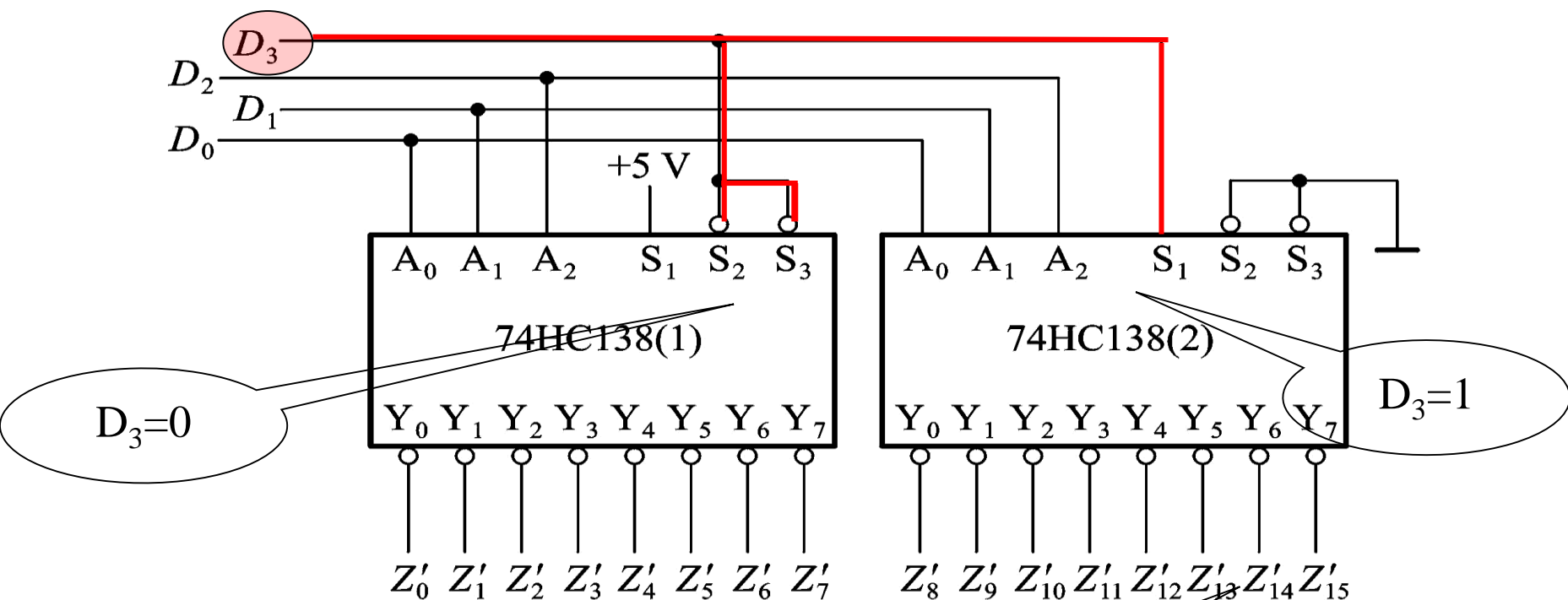
3线-8线译码器CT54/74138

3线-8线译码器CT54/74138真值表

S_1	$\overline{S_2+S_3}$	A_2	A_1	A_0	$\overline{Y_0}$	$\overline{Y_1}$	$\overline{Y_2}$	$\overline{Y_3}$	$\overline{Y_4}$	$\overline{Y_5}$	$\overline{Y_6}$	$\overline{Y_7}$
×	1	×	×	×	1	1	1	1	1	1	1	1
0	×	×	×	×	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1	1	1
1	0	1	0	0	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	1	1	0	1	1
1	0	1	1	0	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	1	0

► 利用附加控制端进行扩展

例：用74HC138（3线—8线译码器）扩展4线—16线译码器



译码条件:

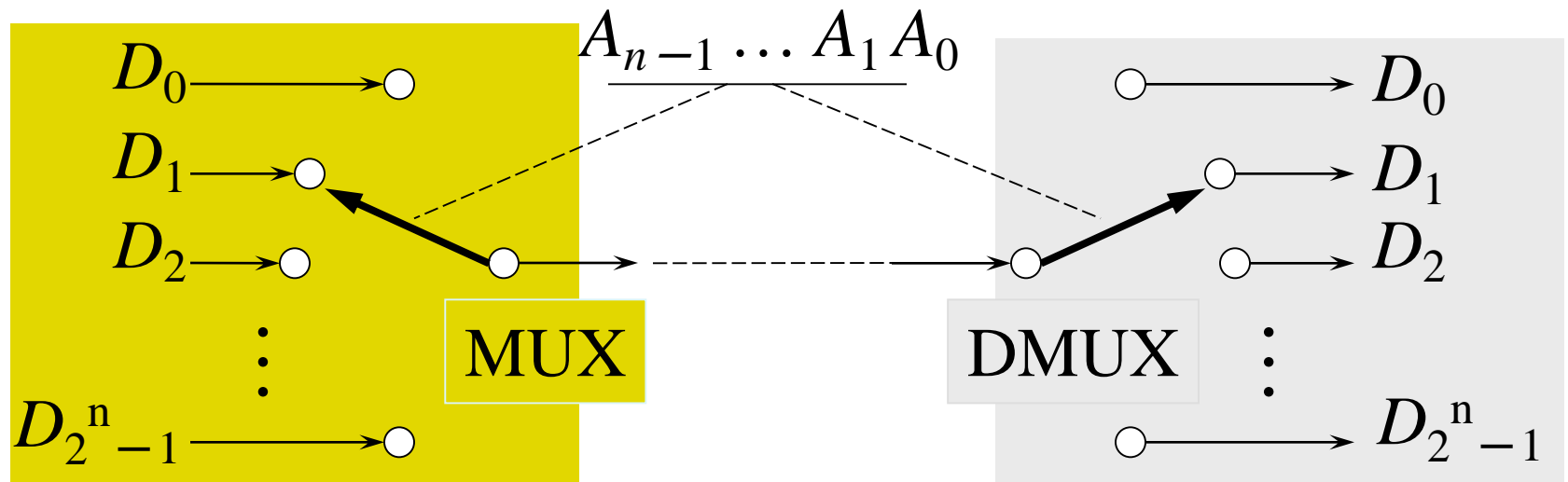
$$S_1 = 1$$

$$S_2 + S_3 = 0$$

$$Z'_i = m'_i$$


4.1.4 数据选择 / 分配器

按 n 位地址码从 2^n 路输入数据通道中选择一个数据传送到输出端上的电路称为数据选择器(MUX); 按 n 位地址码将一路输入数据分送到 2^n 个数据输出端上的电路称为数据分配器(DMUX)。数据选择器和数据分配器联用可实现多路数据的分时传送



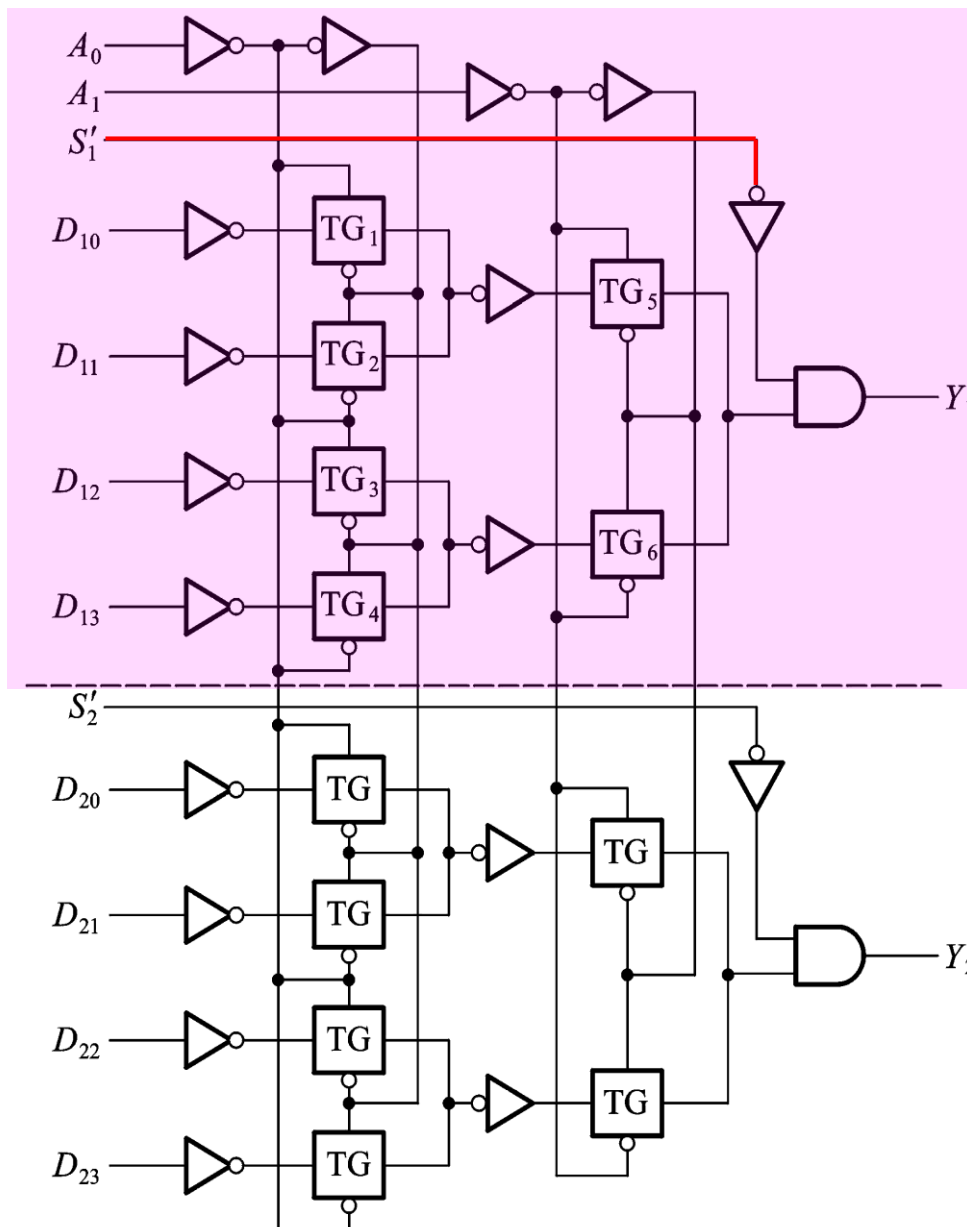
数据选择器

2^n 选1数据选择器的一般表达式为

$$Y = \sum_{i=0}^{2^n-1} m_i \cdot D_i$$


例如：8选1数据选择器的表达式为

$$Y = \bar{A}_2 \bar{A}_1 \bar{A}_0 D_0 + \bar{A}_2 \bar{A}_1 A_0 D_1 + \bar{A}_2 A_1 \bar{A}_0 D_2 + \bar{A}_2 A_1 A_0 D_3 + \\ + A_2 \bar{A}_1 \bar{A}_0 D_4 + A_2 \bar{A}_1 A_0 D_5 + A_2 A_1 \bar{A}_0 D_6 + A_2 A_1 A_0 D_7$$

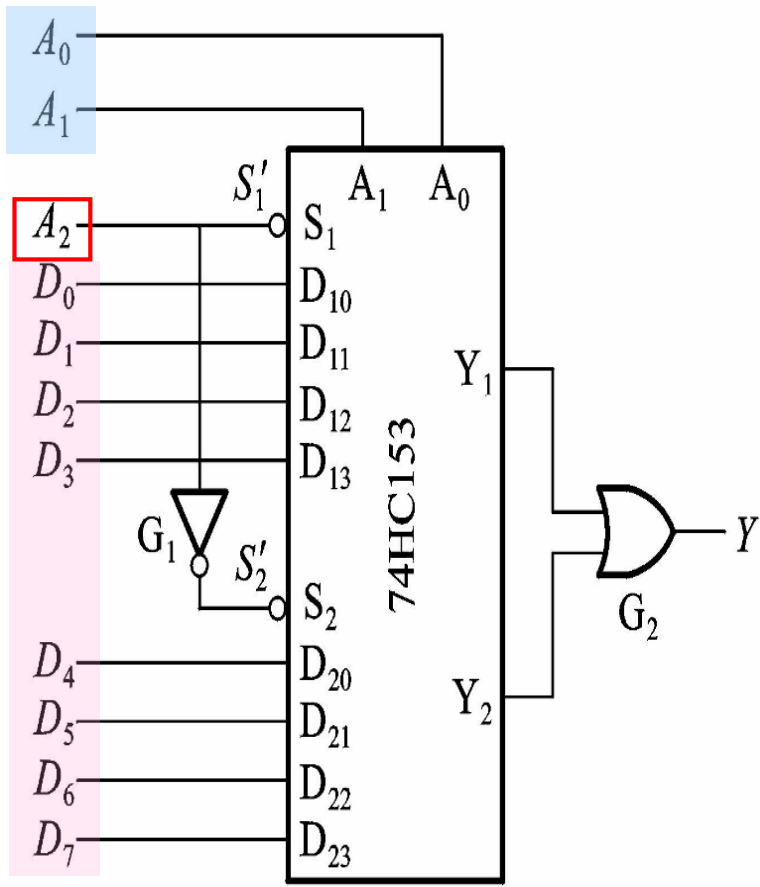


➤ 例：“双四选一”
74HC153，分析其中的一个“四选一”

$$Y_1 = S_1 [D_0(A_1'A_0') + D_1(A_1'A_0) + D_2(A_1A_0') + D_3(A_1A_0)]$$

S_1'	A_1	A_0	Y_1
1	X	X	0
0	0	0	D_{10}
0	0	1	D_{11}
0	1	0	D_{12}
0	1	1	D_{13}

例：用两个“四选一”接成“八选一”



➤ “四选一”只有2位地址输入，从四个输入中选中一个“八选一”的八个数据需要3位地址代码指定其中任何一个

	A ₁	A ₀	Y ₁
S ₁ '	X	X	0
0	0	0	D ₁₀
0	0	1	D ₁₁
0	1	0	D ₁₂
0	1	1	D ₁₃

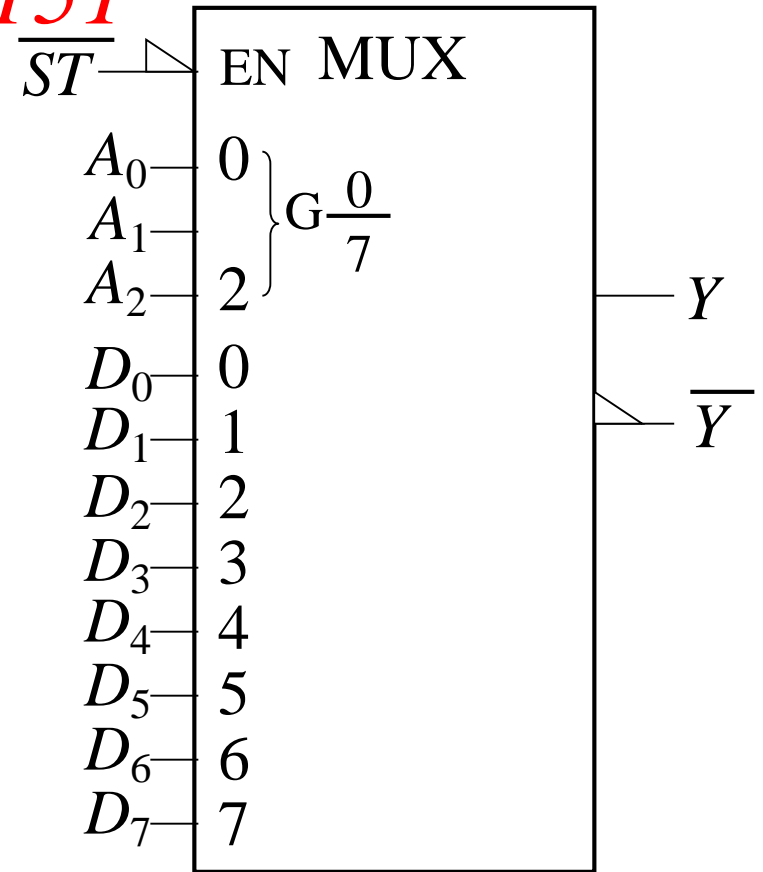
利用S'作为第3位地址输入端

$$\begin{aligned}
 Y = & (A_2' A_1' A_0') D_0 + (A_2' A_1' A_0) D_1 + (A_2' A_1 A_0') D_2 + (A_2' A_1 A_0) D_3 \\
 & + (A_2 A_1' A_0') D_4 + (A_2 A_1' A_0) D_5 \\
 & + (A_2 A_1 A_0') D_6 + (A_2 A_1 A_0) D_7
 \end{aligned}$$

8选1数据选择器 74HC151

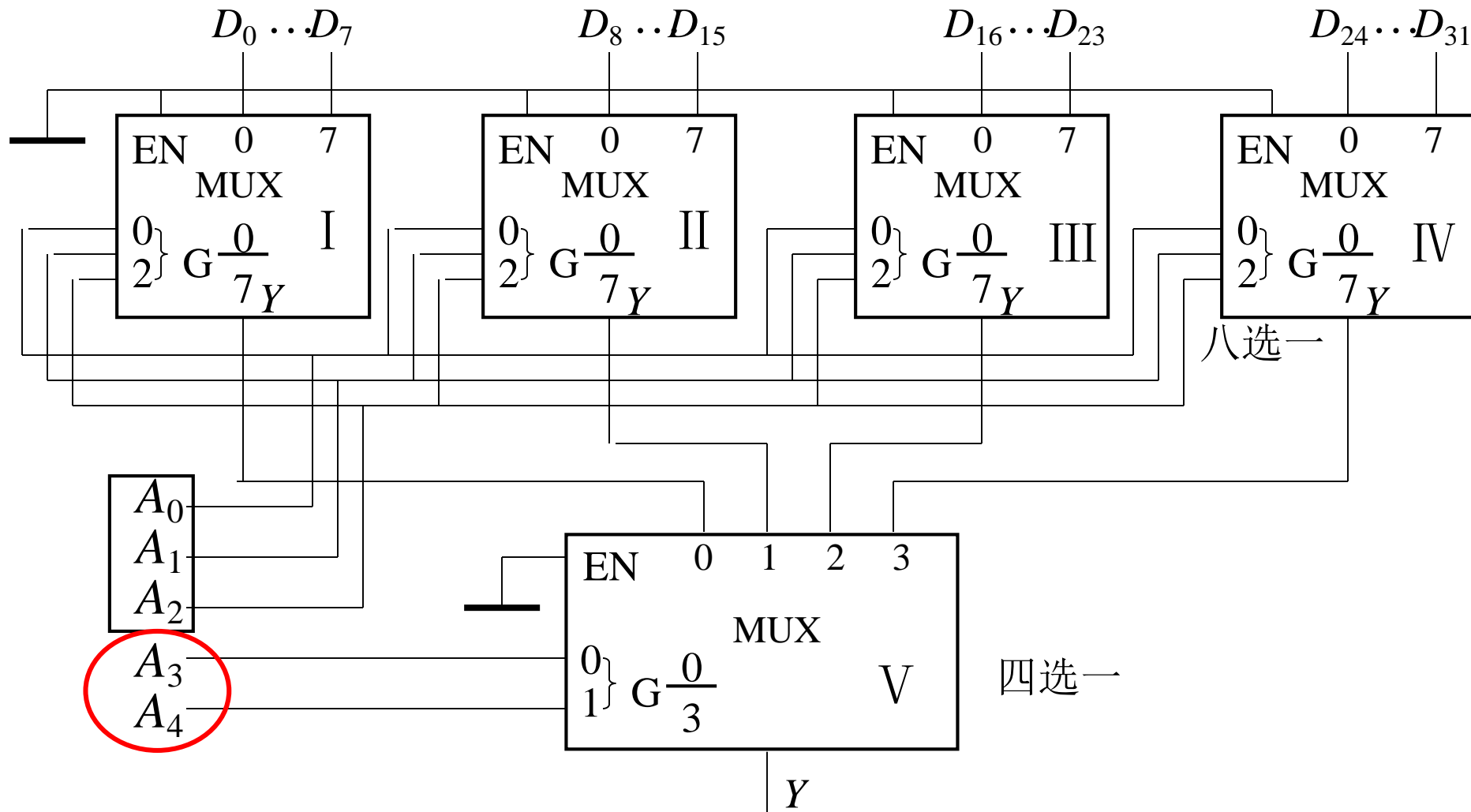
8选1数据选择器真值表

\overline{ST}	A_2	A_1	A_0	Y	\overline{Y}
1	×	×	×	0	0
0	0	0	0	D_0	$\overline{D_0}$
0	0	0	1	D_1	$\overline{D_1}$
0	0	1	0	D_2	$\overline{D_2}$
0	0	1	1	D_3	$\overline{D_3}$
0	1	0	0	D_4	$\overline{D_4}$
0	1	0	1	D_5	$\overline{D_5}$
0	1	1	0	D_6	$\overline{D_6}$
0	1	1	1	D_7	$\overline{D_7}$



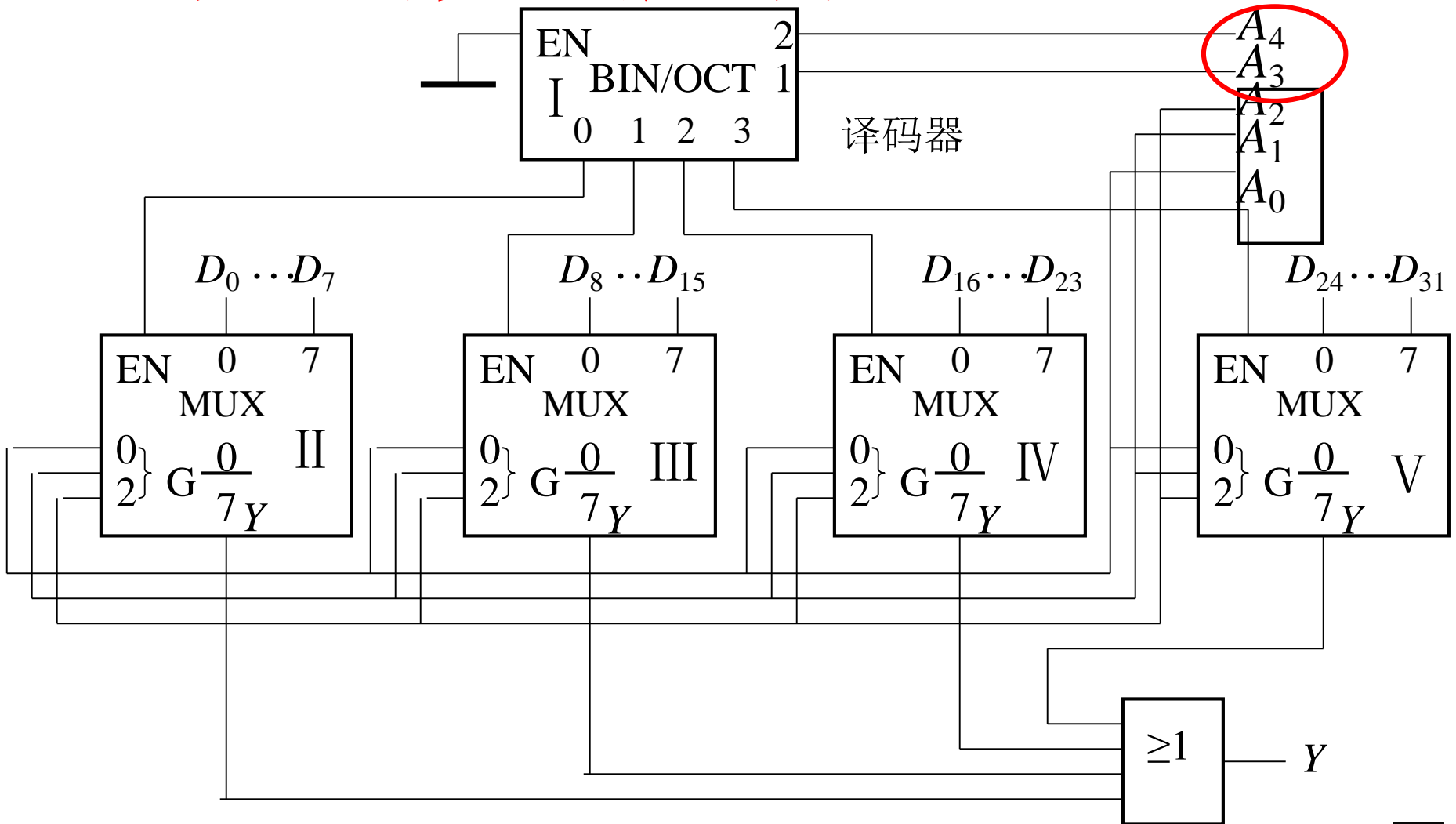
8选1数据选择器逻辑符号

8选1数据选择器扩展应用一



8选1扩展成32选1数据选择器（位扩展）

8选1数据选择器扩展应用二

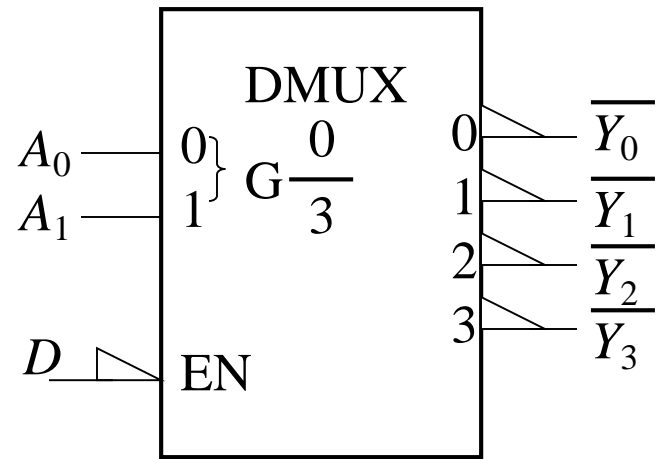


8选1扩展成32选1数据选择器的另一种结构（字扩展）



数据分配器

数据分配器实际上就是译码器，区别仅在于译码器中EN端的作用是选通控制，而在数据分配器中则是作为数据输入端，因此凡是需要使用数据分配器时，都采用译码器，所以集成电路产品手册上根本找不到数据分配器



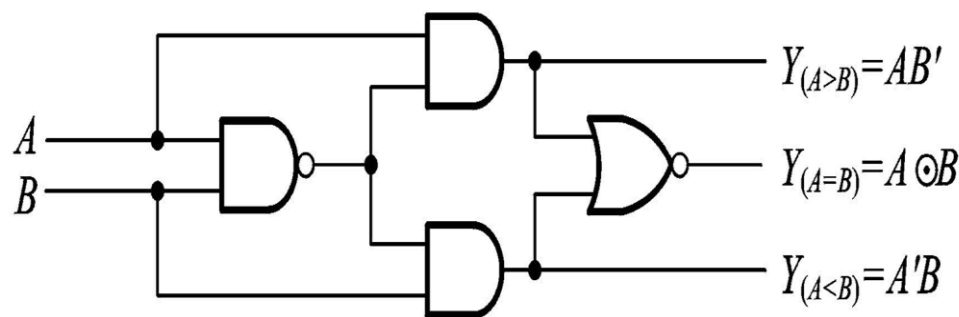
1分4数据分配器逻辑符号

1路-4路数据分配器（如74LS139）、
1路-8路数据分配器（74LS138）



4.1.5 数值比较器

具有比较两个数字数值的大小或判断是否相等的电路称为数值比较器。



输 入		输 出		
A	B	$A > B$	$A = B$	$A < B$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

* $A > B$ ($A = 1, B = 0$) 则 $AB' = 1, \therefore Y_{(A>B)} = AB'$

* $A < B$ ($A = 0, B = 1$) 则 $A'B = 1, \therefore Y_{(A<B)} = A'B$

* $A = B$ (A, B 同为 0 或 1),

$$\therefore Y_{(A=B)} = (A \oplus B)'$$

1位数值比较器

多位数值比较器

原则：从高位比起，只有高位相等，才比较下一位。

比较 $A_3A_2A_1A_0$ 和 $B_3B_2B_1B_0$

$$Y_{(A<B)} = A_3'B_3 + (A_3 \oplus B_3)' A_2'B_2 + (A_3 \oplus B_3)' (A_2 \oplus B_2)' A_1'B_1 \\ + (A_3 \oplus B_3)' (A_2 \oplus B_2)' (A_1 \oplus B_1)' A_0'B_0$$

$$Y_{(A=B)} = (A_3 \oplus B_3)' (A_2 \oplus B_2)' (A_1 \oplus B_1)' (A_0 \oplus B_0)'$$

$$Y_{(A>B)} = (Y_{(A<B)} + Y_{(A=B)})'$$

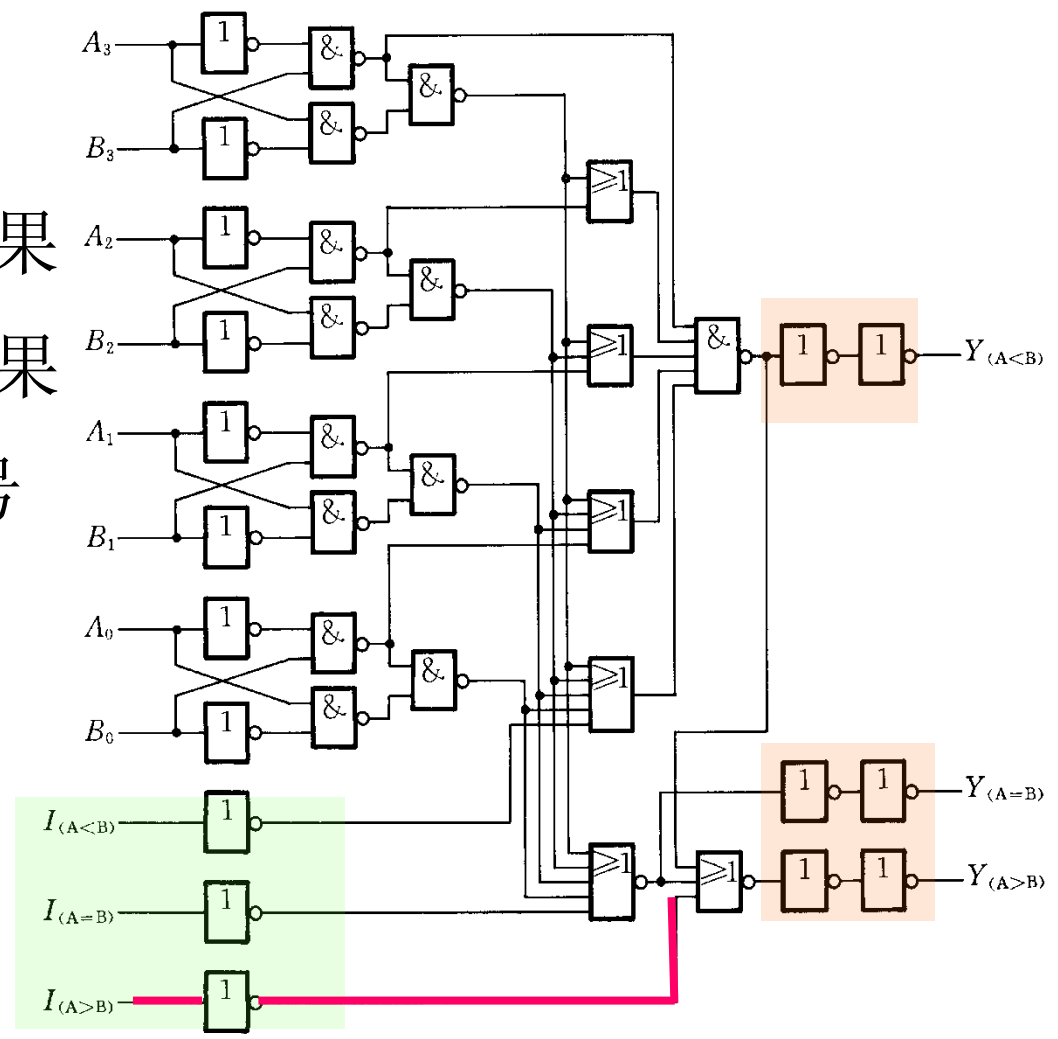
4位二进制比较器74HC85

$I_{(A<B)}$, $I_{(A=B)}$ 和 $I_{(A>B)}$ 为附加端，用于扩展

$I_{(A<B)}$, 来自低位的比较结果

$I_{(A=B)}$, 来自低位的比较结果

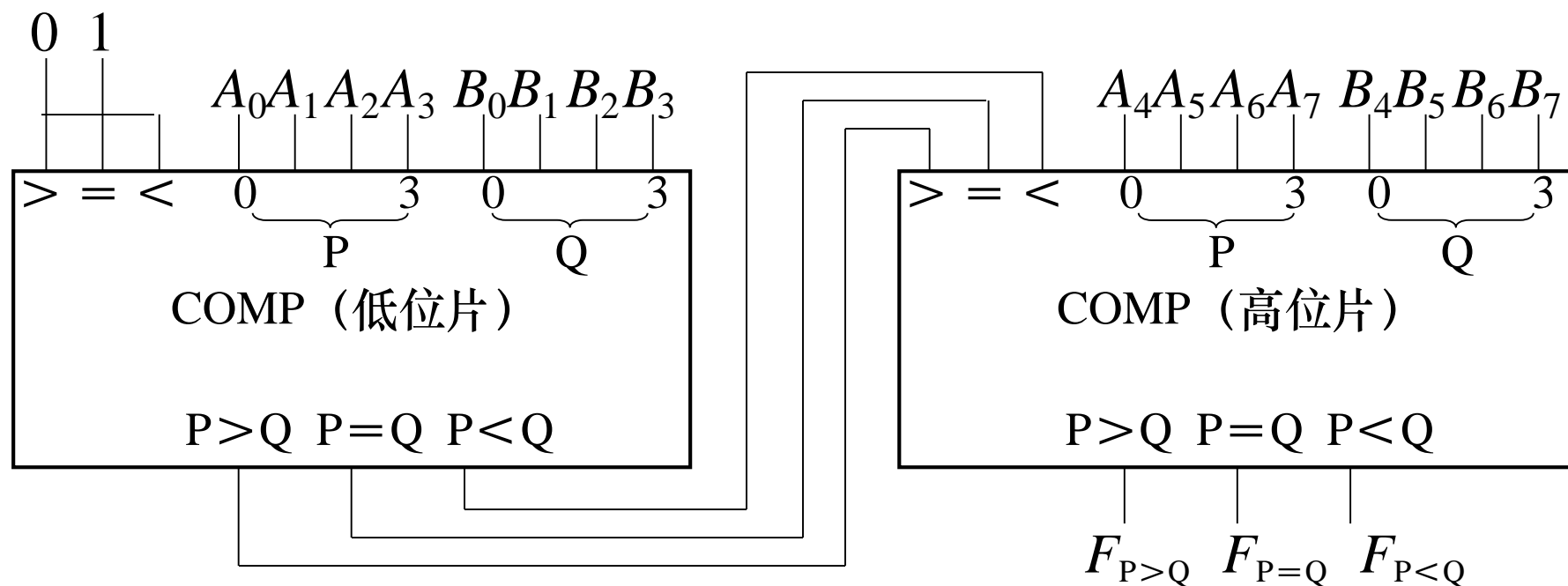
$I_{(A>B)}$, $A > B$ 输出允许信号



4位数值比较器真值表

输 入										输 出			
A_3	B_3	A_2	B_2	A_1	B_1	A_0	B_0	$A>B$	$A=B$	$A<B$	$F_{A>B}$	$F_{A=B}$	$F_{A<B}$
$A_3>B_3$		×	×	×	×	×	×	×	×	×	1	0	0
$A_3<B_3$		×	×	×	×	×	×	×	×	×	0	0	1
$A_3=B_3$		$A_2>B_2$		×	×	×	×	×	×	×	1	0	0
$A_3=B_3$		$A_2<B_2$		×	×	×	×	×	×	×	0	0	1
$A_3=B_3$		$A_2=B_2$		$A_1>B_1$		×	×	×	×	×	1	0	0
$A_3=B_3$		$A_2=B_2$		$A_1<B_1$		×	×	×	×	×	0	0	1
$A_3=B_3$		$A_2=B_2$		$A_1=B_1$		$A_0>B_0$		×	×	×	1	0	0
$A_3=B_3$		$A_2=B_2$		$A_1=B_1$		$A_0<B_0$		×	×	×	0	0	1
$A_3=B_3$		$A_2=B_2$		$A_1=B_1$		$A_0=B_0$		1	0	0	1	0	0
$A_3=B_3$		$A_2=B_2$		$A_1=B_1$		$A_0=B_0$		0	1	0	0	1	0
$A_3=B_3$		$A_2=B_2$		$A_1=B_1$		$A_0=B_0$		0	0	1	0	0	1

数值比较器扩展应用



4位数值比较器扩展构成8位数值比较器

高位片有比较结果，由高位片输出

高4位相等时，由低位片比较，高位片根据低位片比较结果($>$ 、 $=$ 、 $<$)决定比较结果，由高位片输出

4.2 组合逻辑电路设计

组合逻辑电路设计是组合逻辑电路分析的逆过程。根据给定的逻辑功能，设计出能够实现这些功能的逻辑电路。

设计组合逻辑电路时可供选用的数字电路器件有：小规模集成门电路（SSI）、中规模数字集成电路（MSI）、存储器（ROM）、可编程逻辑器件（PLD）

4.2.1 采用小规模集成器件的 组合逻辑电路设计

采用小规模集成器件设计组合逻辑电路是一种传统的、规范的、经典的方法

设计时应从经济指标、工作速度、功耗等方面综合考虑，以期得到所谓“最小化”电路。“最小化”电路不一定是“最佳化”电路，只是为满足工程需要而提出的，要求设计时电路使用器件的种类和数目尽可能少；器件间的连线尽可能简单，门电路级数尽可能少；从而达到满足工作速度要求、减少功耗、提高可靠性的目的

采用SSI设计组合逻辑电路的一般步骤

- ❖ 分析设计要求 :要求详尽
- ❖ 列写真值表 :强调正确
- ❖ 写出逻辑函数表达式
- ❖ 化简、变换表达式 :力求简洁
- ❖ 画出逻辑图
- ❖ 实验验证或仿真

设计从文字描述出发，最终得到满足功能要求的逻辑图，其中每一个步骤也应当符合逻辑解释

采用SSI设计组合逻辑电路例

例1 设计一个三人提案表决电路

解 (1)分析题意：

当两人或两人以上同意提案时，

提案可获得通过

设三人分别是 A 、 B 、 C ，

同意提案为1，不同意提案为0，

设提案是 P ，

提案通过为1，不通过为0

4.2.1 采用SSI的组合逻辑电路设计

(2)列写真值表

A	B	C	P
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(3)写出逻辑函数表达式

$$P = \bar{A} B C + A \bar{B} C + A B \bar{C} + A B C$$

4.2.1 采用SSI的组合逻辑电路设计

(4)化简和变换表达式

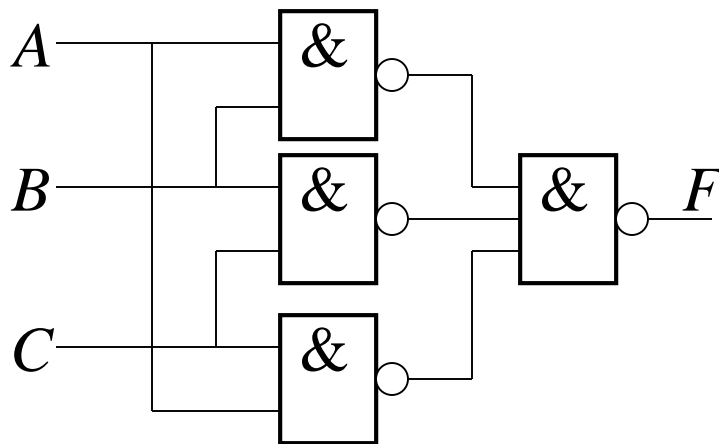
C \ AB				
	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$P = AB + BC + AC$$

$$= \overline{\overline{AB + BC + AC}}$$

$$= \overline{\overline{AB} \cdot \overline{BC} \cdot \overline{AC}}$$

(5)画出逻辑图



4.2.1 采用SSI的组合逻辑电路设计

进一步对表达式进行变换

$$P = AB + BC + AC$$

$$= A(B + C) + BC$$

$$= (A + BC)(B + C)$$

$$A + BC = (A + B)(A + C)$$

$$= (A + B)(A + C)(B + C)$$

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

$$= \overline{\overline{A + B} + \overline{A + C} + \overline{B + C}}$$

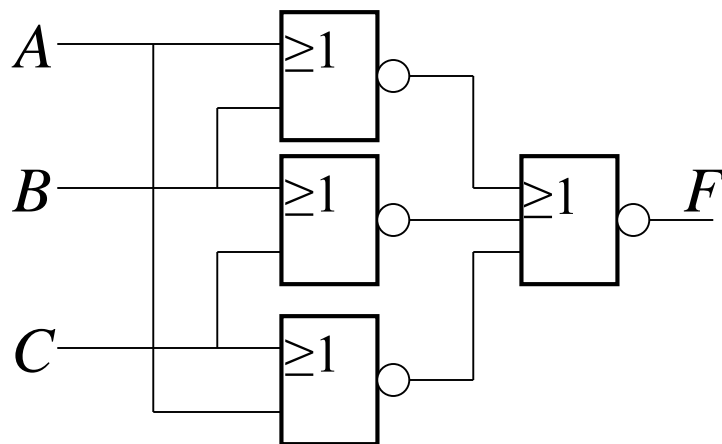
—或非—或非表达式

$$= \overline{\overline{A} \overline{B} + \overline{A} \overline{C} + \overline{B} \overline{C}}$$

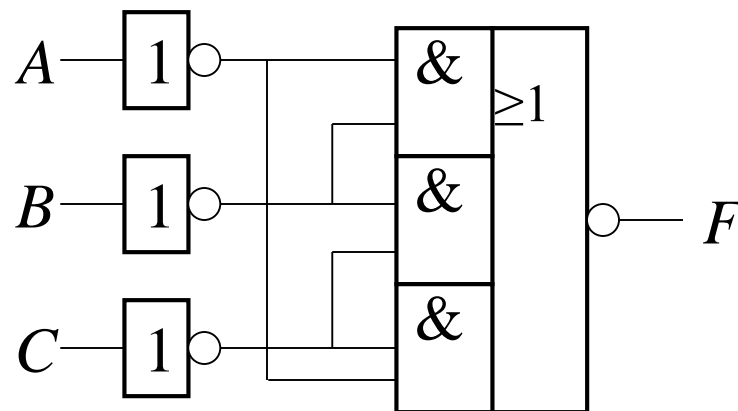
—与或非表达式

对应可以用或非门和与或非门实现

4.2.1 采用SSI的组合逻辑电路设计



用与非门实现



用与或非门实现

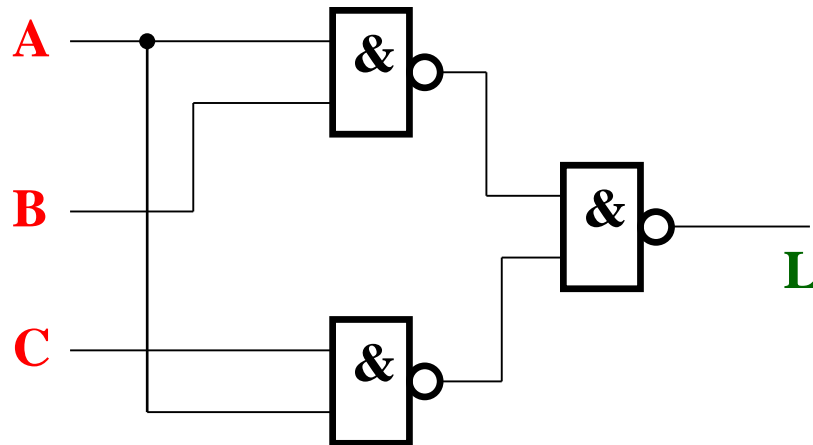
从本例可见，一个设计任务可以有多种实现方案，从而得到不同的组合逻辑电路，究竟采用什么方法，需根据具体情况而定

4.2.1 采用SSI的组合逻辑电路设计

➤ 若三人中的A有否决权，即A不赞成，就不能通过，又应如何实现呢？

$$L = AB + AC$$

$$L = \overline{\overline{AB} \overline{AC}}$$



4.2.1 采用SSI的组合逻辑电路设计

例2 用与非门实现逻辑函数

$$F(A, B, C, D) = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14)$$

解 用卡诺图对函数进行化简，得

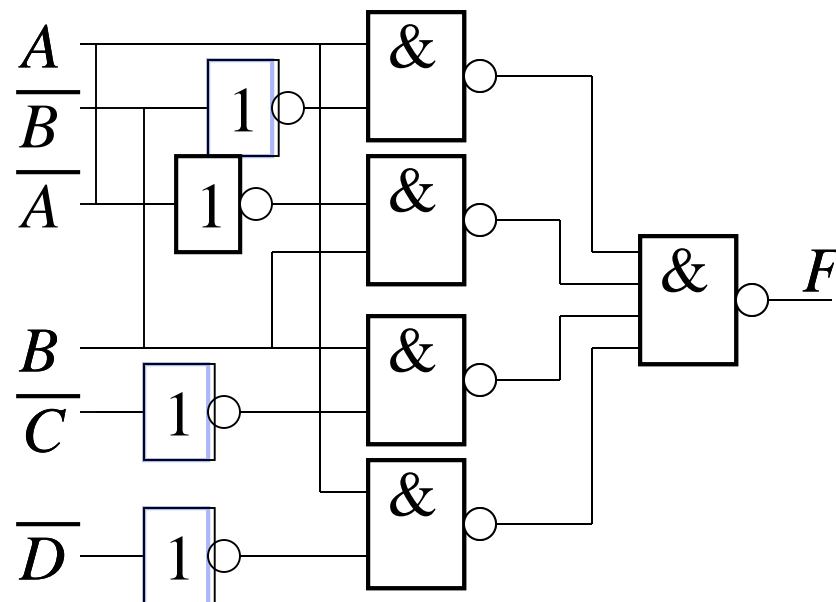
$CD \backslash AB$					
		00	01	11	10
00			1	1	1
01			1	1	1
11			1		1
10			1	1	1

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

4.2.1 采用SSI的组合逻辑电路设计

经变换可得到
与非—与非表达式

$$\begin{aligned} F &= A\bar{B} + \bar{A}B + B\bar{C} + A\bar{D} \\ &= \overline{\overline{A\bar{B} + \bar{A}B + B\bar{C} + A\bar{D}}} \\ &= \overline{\overline{A\bar{B}} \cdot \overline{\bar{A}B} \cdot \overline{B\bar{C}} \cdot \overline{A\bar{D}}} \end{aligned}$$



从变换后的表达式可见，需要5个与非门才能实现该函数，而且输入变量中有原变量，还有反变量，该如何处理这些反变量？

输入端添加非门求反，则共需要9个门
寻求新的途径 — 再变换表达式

4.2.1 采用SSI的组合逻辑电路设计

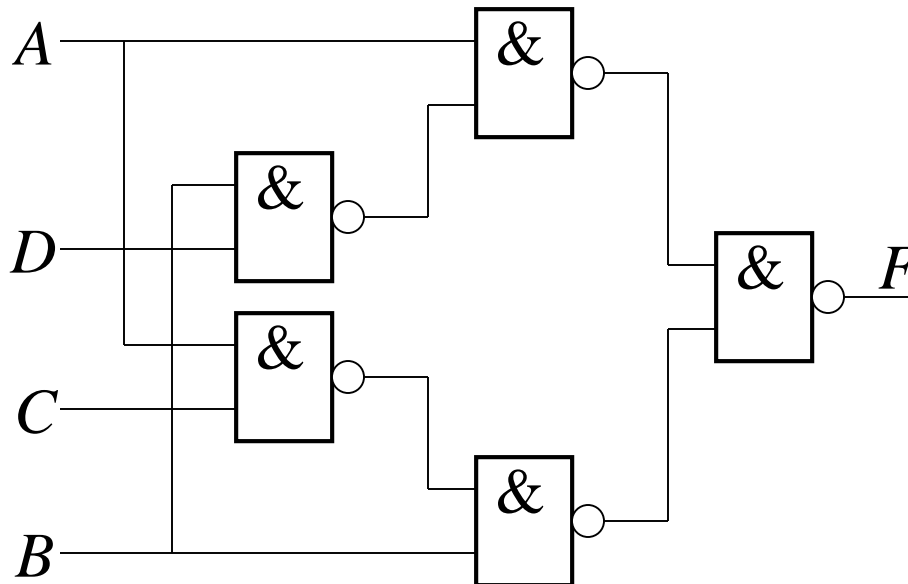
再经变换得到又一种与非—与非表达式

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

$$= A(\bar{B} + \bar{D}) + B(\bar{A} + \bar{C})$$

$$= A\bar{B}\bar{D} + B\bar{A}\bar{C}$$

$$= \overline{\overline{A\bar{B}\bar{D}}} \cdot \overline{\overline{B\bar{A}\bar{C}}} \quad \text{需5个与非门实现,但无反变量}$$



4.2.1 采用SSI的组合逻辑电路设计

再经变换还可得到另一种与非—与非表达式

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

$$= A\bar{B} + \bar{A}B + B\bar{C} + A\bar{D} + B\bar{D} + A\bar{C}$$

$$= A(\bar{B} + \bar{C} + \bar{D}) + B(\bar{A} + \bar{C} + \bar{D})$$

$$= \overline{\overline{A\bar{B}C\bar{D}} \cdot \overline{B\bar{A}C\bar{D}}}$$

$$= \overline{\overline{AABCD} \cdot \overline{BABCD}}$$

仅需用4个与非门实现

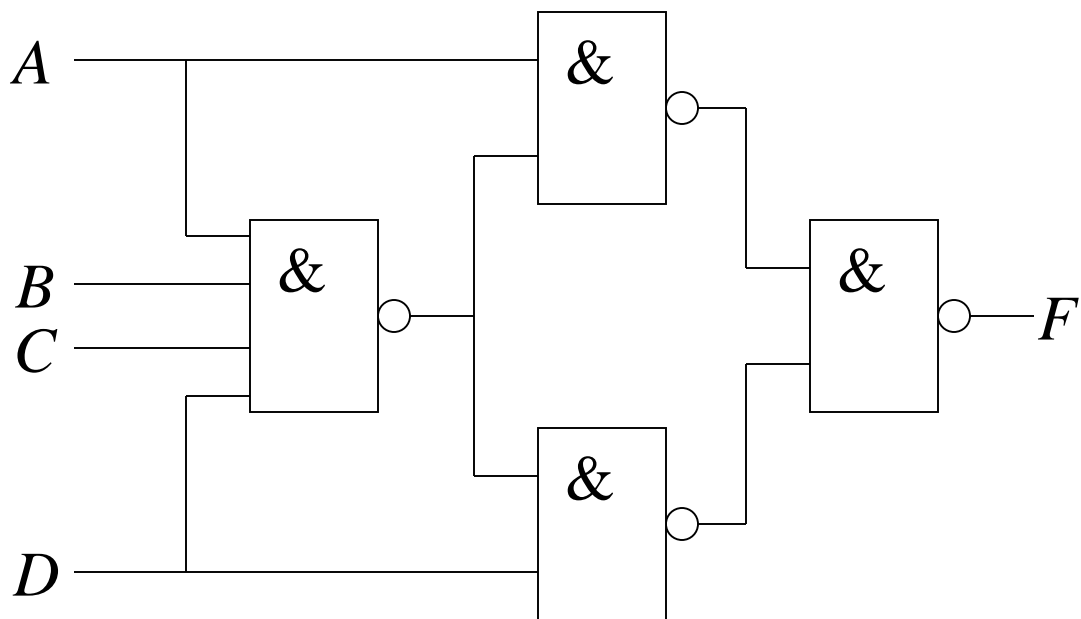
这里将尾部因子用所谓“尾部替代因子”替代，从而减少了使用的门的数量

$$\overline{A\bar{B}C\bar{D}} = A(\bar{A} + \overline{BCD}) = A\overline{A\bar{B}C\bar{D}}$$

$$\overline{B\bar{A}C\bar{D}} = B(\bar{B} + \overline{ACD}) = B\overline{A\bar{B}C\bar{D}}$$

4.2.1 采用SSI的组合逻辑电路设计

可以说得到了一个最小化电路



4.2.1 采用SSI的组合逻辑电路设计

例3 用或非门实现逻辑函数

$$F(A, B, C, D) = \sum m(0, 5, 7, 11, 12, 13, 15)$$

解 求原函数的对偶函数的与非—与非表达式

$$\overline{F}(A, B, C, D) = \sum m(1, 2, 3, 4, 6, 8, 9, 10, 14)$$

$$F^*(A, B, C, D)$$

$$= \sum m(14, 13, 12, 11, 9, 7, 6, 5, 1)$$

最小项表达式的对偶函数的标准与或表达式的获得方法

$CD \backslash AB$		AB			
		00	01	11	10
00				1	
01	1	1	1	1	1
11			1		1
10			1	1	

4.2.1 采用SSI的组合逻辑电路设计

写出对偶函数的最简与或表达式并化简、变换

$$\begin{aligned}F^* &= \overline{C}D + \overline{A}BC + AB\overline{D} + A\overline{B}D \\&= \overline{C}D + \overline{A}BC + AB\overline{D} + ABC\overline{C} + BC\overline{D} + A\overline{B}D \\&= D\overline{C}\overline{D} + BC\overline{A}\overline{D} + AB\overline{C}\overline{D} + A\overline{B}D \\&= D\overline{C}\overline{D} + BC\overline{A}\overline{B}\overline{D} + AB\overline{C}\overline{D} + AD\overline{A}\overline{B}\overline{D} \\&= \overline{\overline{D}\overline{C}\overline{D}} \cdot \overline{\overline{B}\overline{C}\overline{A}\overline{B}\overline{D}} \cdot \overline{\overline{A}\overline{B}\overline{C}\overline{D}} \cdot \overline{\overline{A}\overline{D}\overline{A}\overline{B}\overline{D}}\end{aligned}$$

最后得到原函数的或非—或非表达式

$$\begin{aligned}F &= (F^*)^* \\&= \overline{\overline{D + C + D + B + C + A + B + D + A + B + C + D + A + D + A + B + D}}\end{aligned}$$

用七个或非门可以实现
该逻辑函数(逻辑图略)

4.2.1 采用SSI的组合逻辑电路设计

例4 用与门和异或门实现逻辑函数

$$F(A, B, C, D) = \Sigma m(0, 1, 6, 7, 8, 9, 11, 12, 13)$$

解 求函数 F 的最简异或表达式

当 $AB=0$ 时:

$$\begin{aligned} A+B &= A \oplus B \oplus (A B) \\ &= A \oplus B \end{aligned}$$

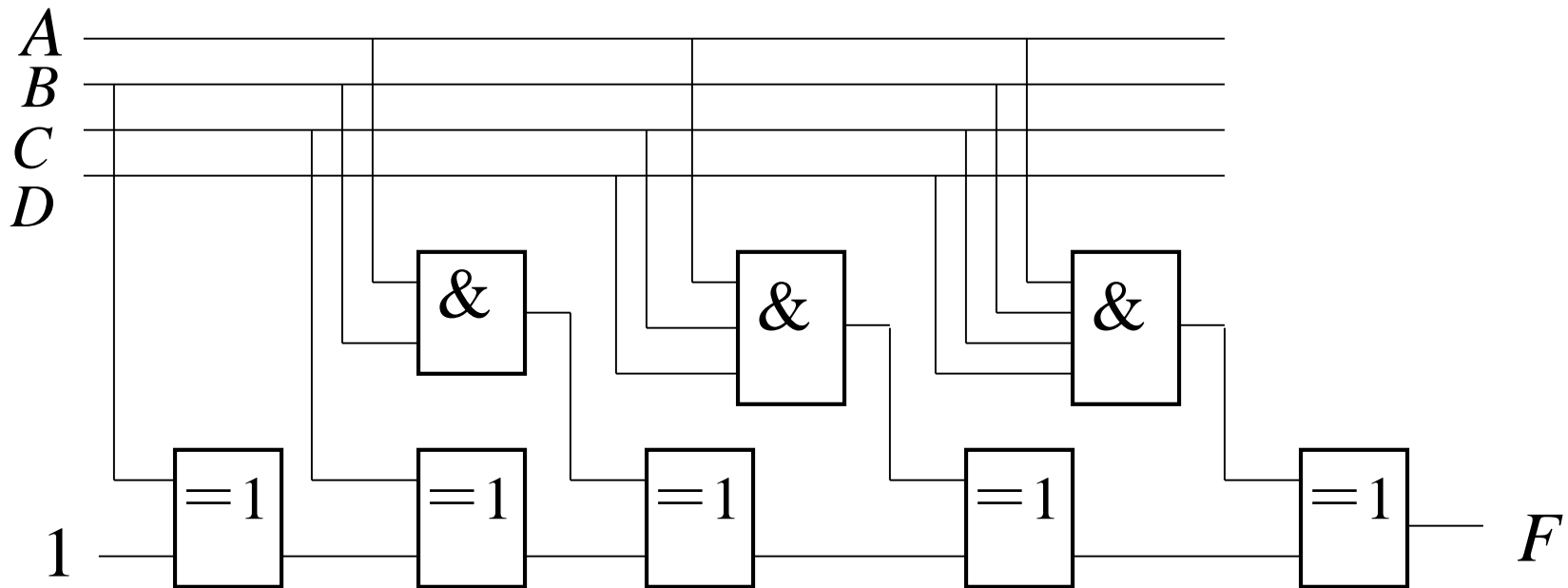
若卡诺图中的两个圈不重叠则这两个圈对应的乘积项之积必定为0

于是可以从函数的积之和表达式
写出异或表达式

		AB			
		00	01	11	10
CD	00	1		1	1
	01	1		1	1
	11		1		1
	10		1		

4.2.1 采用SSI的组合逻辑电路设计

$$\begin{aligned} F &= A\bar{C} + \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}CD \\ &= A\bar{C} \oplus \bar{A}\bar{B}\bar{C} \oplus \bar{A}BC \oplus A\bar{B}CD \\ &= A(1 \oplus C) \oplus (1 \oplus A)(1 \oplus B)(1 \oplus C) \oplus (1 \oplus A)BC \oplus ACD(1 \oplus B) \\ &= A \oplus AC \oplus 1 \oplus A \oplus B \oplus AB \oplus C \oplus AC \oplus BC \oplus ABC \oplus BC \oplus \\ &\quad \oplus ABC \oplus ACD \oplus ABCD \\ &= 1 \oplus B \oplus C \oplus AB \oplus ACD \oplus ABCD \end{aligned}$$



用异或门和与门实现的逻辑图

4.2.2 采用中规模集成器件

实现组合逻辑函数

采用MSI实现组合逻辑函数的特点

器件的名称仅仅表示其基本逻辑功能，可以扩展开发出更多的应用，用MSI设计电路可省去许多繁琐的设计过程，减少甚至避免设计错误，改善电路性能

用MSI实现组合逻辑电路的基本方法是对比法，将待实现的逻辑函数表达式与选用MSI器件的表达式进行对比

采用MSI实现组合逻辑函数的方法

- ❖ 将待实现的逻辑函数表达式进行变换，尽可能使其变换成与MSI器件的表达式完全相同的形式或类似的形式
- ❖ 将它们的表达式进行对比，若两者完全一致，则使用这种器件最为简便；若两者仅仅部分相同则需根据具体情况适当处理：器件有多余输入端时可空闲不用，器件容量不足时需要扩展后再应用
- ❖ 数据选择器常用于实现单输出逻辑函数，译码器则多用于实现多输出逻辑函数

用MUX实现组合逻辑函数

MUX的逻辑函数表达式是
$$Y = \sum_{i=0}^{2^n-1} m_i \cdot D_i$$

任意组合逻辑函数的表达式可以写成
$$F = \sum_{i=0}^{2^n-1} a_i \cdot m_i$$

对比后不难发现它们的共同之处。显然，用数据选择器实现组合逻辑函数比较方便

用具有 n 个地址输入端的数据选择器实现 m 变量组合逻辑函数时，可能有三种情况： $n = m$ ， $n > m$ ，和 $n < m$ ，以下举例分别进行讨论

☆ 当 $n=m$ 时的设计

例1 用8选1数据选择器实现函数

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

解

实际做对比时，往往并不需要对比表达式，而是将MUX和函数的卡诺图进行对比，一般步骤是：

- 确定数据选择器的全部地址输入端和函数的全部变量的对应连接关系
- 数据选择器卡诺图各方格中的 D_i 与函数卡诺图各方格的值（0或1）按位置对应相等
- 按照对比结果完成逻辑图

4.2.1 采用MSI实现组合逻辑函数

A_2A_1					
A_0		00	01	11	10
	0	D_0	D_2	D_6	D_4
1	D_1	D_3	D_7	D_5	

MUX卡诺图

AB					
C		00	01	11	10
	0	0	1	1	1
1	1	1	1	0	1

逻辑函数卡诺图

比较的结果是：

$$A_2=A, \quad A_1=B, \quad A_0=C$$

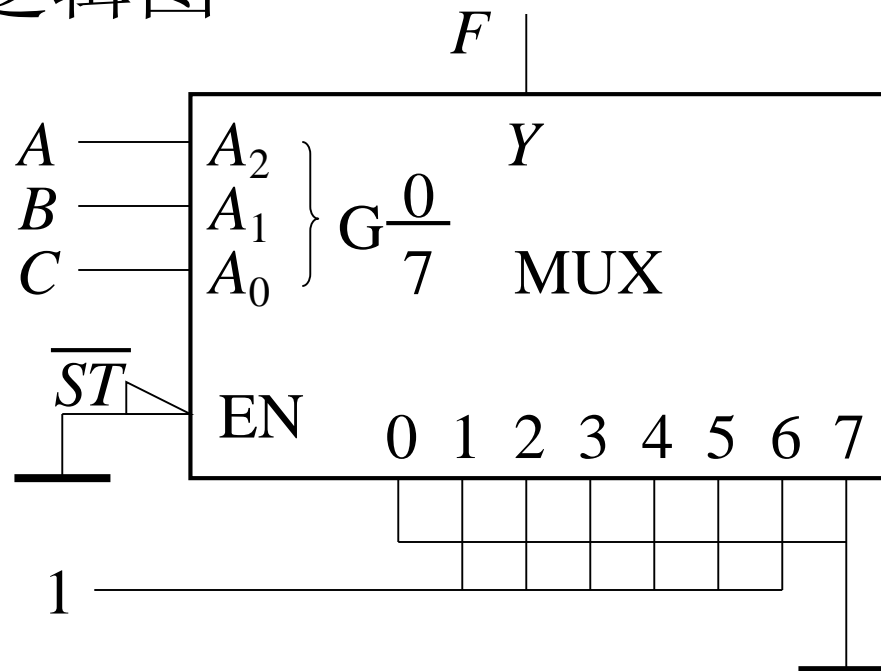
$$D_0=0, \quad D_1=1, \quad D_2=1, \quad D_3=1$$

$$D_4=1, \quad D_5=1, \quad D_6=1, \quad D_7=0$$

这就是说，将输入变量加到地址端，MUX的数据输入端按函数卡诺图中各方格的值对应相连

4.2.1 采用MSI实现组合逻辑函数

画出逻辑图



从本例的设计过程可见，当 $m=n$ 时并不需要将函数化简为最简表达式，只需直接将输入变量加到地址端，MUX的数据输入端则按卡诺图中各方格的值（0或1）对应相连

4.2.1 采用MSI实现组合逻辑函数

☆ 当 $n < m$ 时的设计

由于数据选择器地址端的个数少于函数变量个数，表面看似无法直接利用器件实现函数。但是可以通过以下两种方法解决，分别予以介绍

扩展法—以增加器件数量为前提，利用 EN 端的作用实现扩展

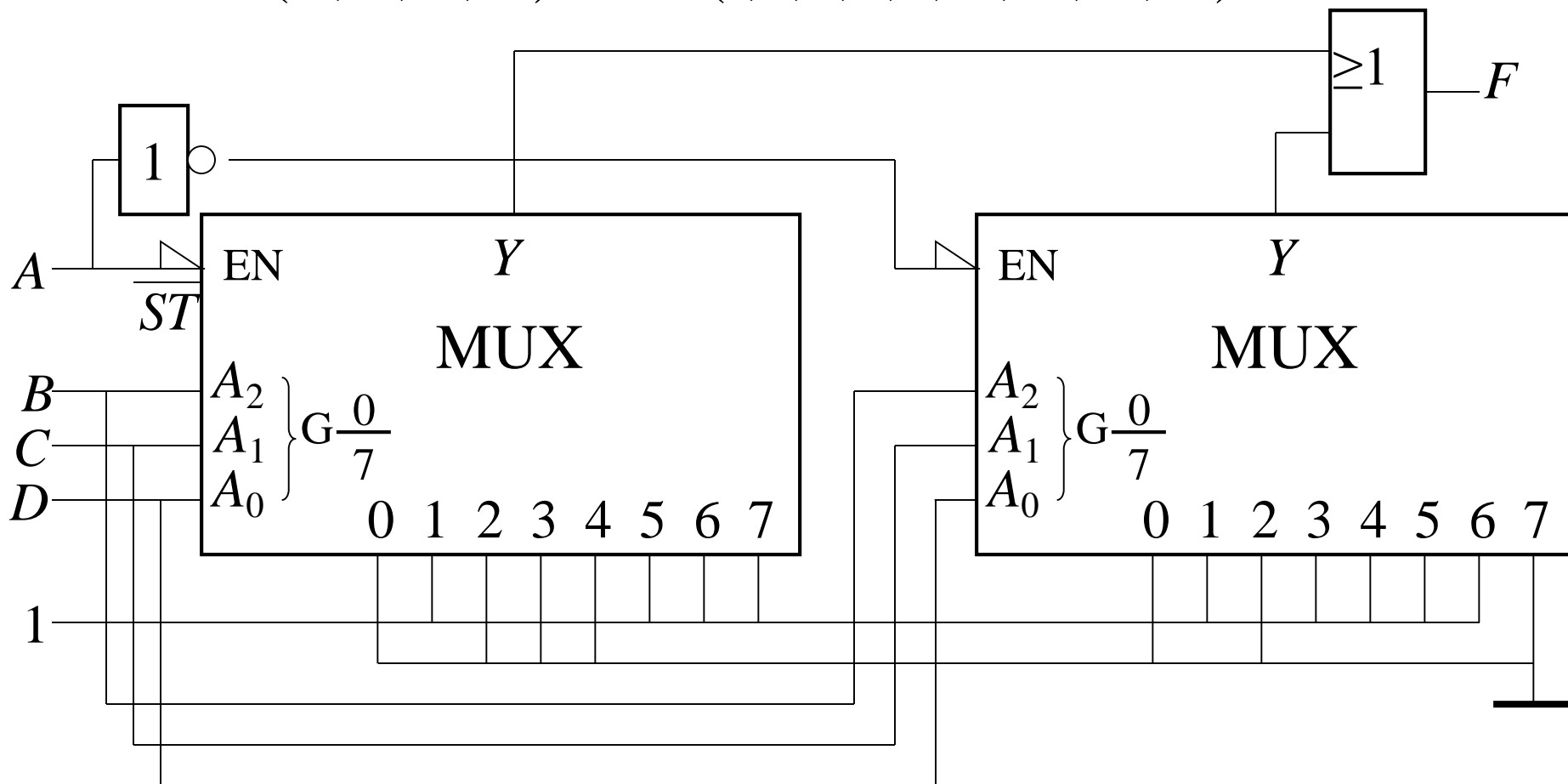
降维图法—采用“软”的方法，减少函数卡诺图中外围变量的个数，从而使之与数据选择器的地址端的个数相对应（略）

4.2.1 采用MSI实现组合逻辑函数

☆ 当 $n < m$ 时的设计

例2 用8选1数据选择器实现函数(用扩展法)

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



讨论：故障报警系统

某实验室有红、黄两个故障指示灯，用来指示三台设备的工作情况。当只有一台设备有故障时，黄灯亮；有两台设备有故障时，红灯亮；只有当三台设备都发生故障时，才会使红、黄两个故障指示灯同时点亮。

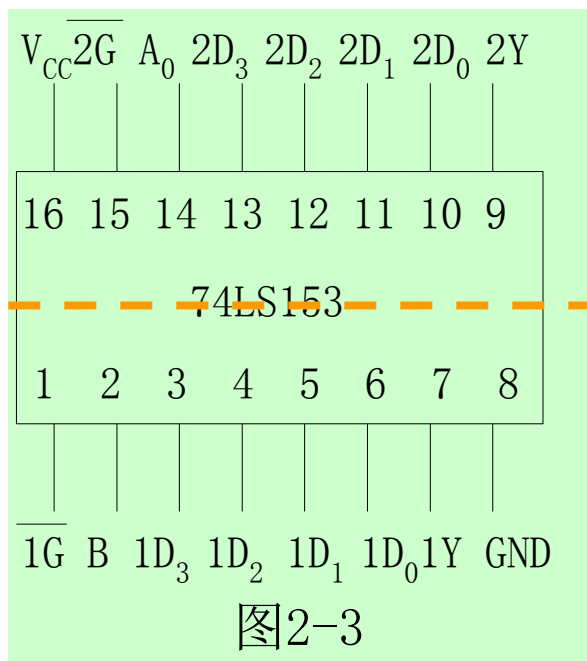
提示：本题有三个变量A、B、C，表示三台设备。“1”表示故障；“0”表示正常。

有两个逻辑函数L1、L2，分别表示红灯和黄灯。“1”——灯亮，表示报警；“0”——灯不亮，表示不报警。

要求采用MSI组合电路设计方法，用双四选一数据选择器（74LS153）实现，并验证其逻辑功能。

双四选一数据选择器（74LS153）的管脚图如图2-3所示。逻辑功能表如表2-1所示。

74LS153功能表



选择输入端		数据输入				选通输入	输出
B	A	D ₀	D ₁	D ₂	D ₃	\overline{G}	Y
X	X	X	X	X	X	1	Z（高阻态）
0	0	0	X	X	X	0	0
0	0	1	X	X	X	0	1
0	1	X	0	X	X	0	0
0	1	X	1	X	X	0	1
1	0	X	X	0	X	0	0
1	0	X	X	1	X	0	1
1	1	X	X	X	0	0	0
1	1	X	X	X	1	0	1

根据74LS153功能表列出输出逻辑表达式

$$Y = \overline{B}\overline{A}D_0 + \overline{B}AD_1 + B\overline{A}D_2 + BAD_3$$

$$= m_0D_0 + m_1D_1 + m_2D_2 + m_3D_3$$

根据74LS153功能表列出输出逻辑表达式

$$Y = \overline{B}\overline{A}D_0 + \overline{B}AD_1 + B\overline{A}D_2 + BAD_3$$

$$= m_0D_0 + m_1D_1 + m_2D_2 + m_3D_3$$

根据真值表表列出逻辑表达式

$$L1 = \overline{B}AC + B\overline{A}C + B\overline{A}\overline{C} + BAC$$

$$= \overline{B}\overline{A}0 + \overline{B}AC + B\overline{A}C + (BAC + B\overline{A}\overline{C})$$

$$= m_00 + m_1D_1 + m_2D_2 + m_31$$

令B、A分别从选择输入端B、A输入；

数据输入端 $1D_0=0$, $1D_1=1D_2=C$, $1D_3=1$

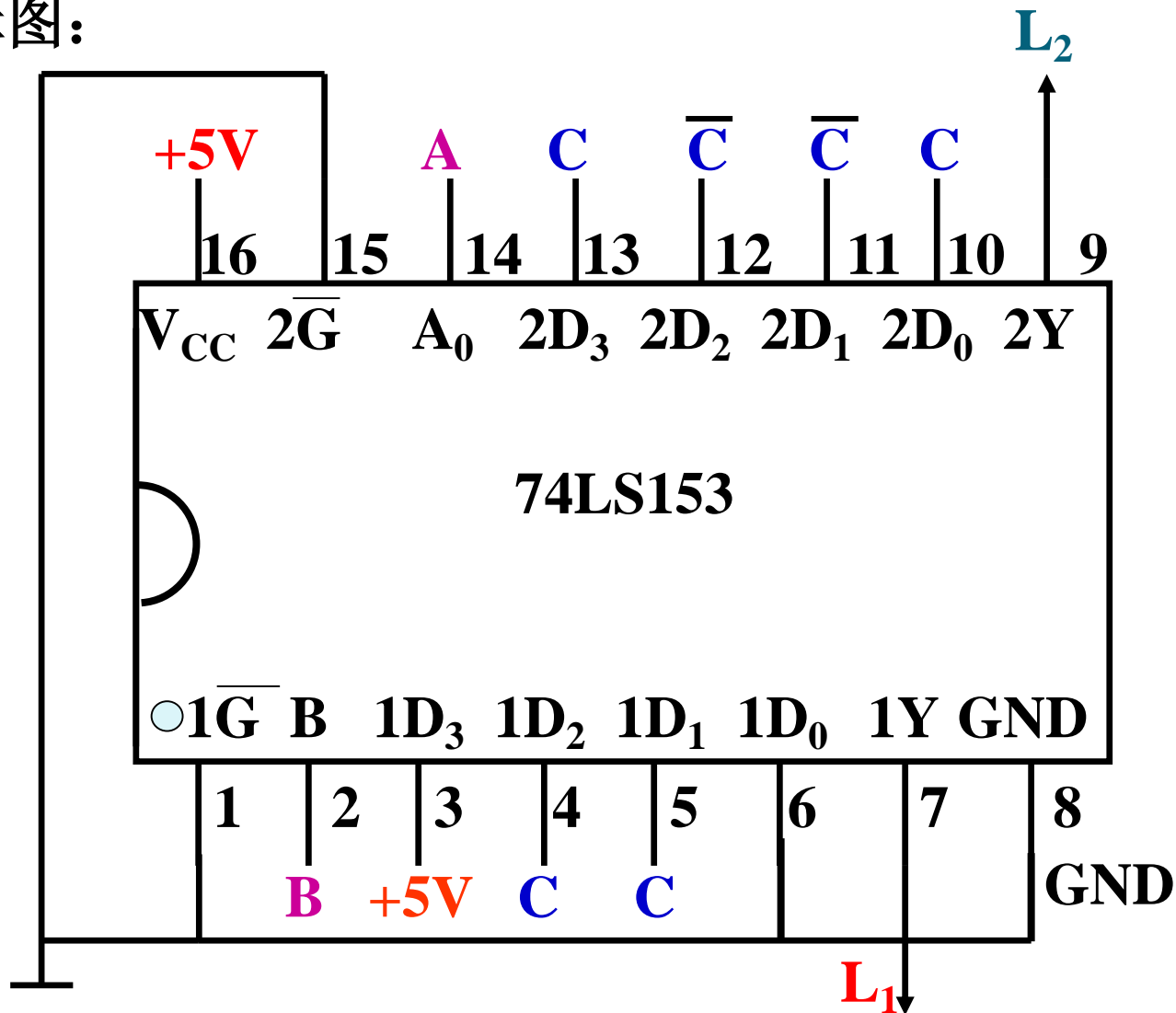
$$L_2 = \overline{B}\overline{A}C + \overline{B}\overline{A}\overline{C} + B\overline{A}\overline{C} + BAC$$

$$= m_0D_0 + m_1D_1 + m_2D_2 + m_3D_3$$

B	A	C	L ₁	L ₂
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

数据输入端 $2D_0=2D_3=C$, $2D_1=2D_2=\overline{C}$

(3) 逻辑图:



注意: “0”接地; “1”接 $+5V$;

用译码器实现组合逻辑函数

译码器的逻辑函数表达式是 $\overline{Y}_i = \overline{m_i}$
任意组合逻辑函数的表达式可以写成

$$\begin{aligned} F &= m_0 + m_1 + \cdots + m_{2^n-1} \\ &= \overline{\overline{m_0} \cdot \overline{m_1} \cdots \overline{m_{2^n-1}}} \\ &= \overline{Y_0 \cdot Y_1 \cdots Y_{2^n-1}} \end{aligned}$$

一个二进制译码器的输出包含了全部输入变量的最小项，常称为完全译码器或变量译码器。用 n 变量译码器加上输出门，就能获得任何形式的输入变量不大于 n 的组合逻辑函数，且很容易实现多输出函数

用译码器实现组合逻辑函数

例1 用译码器实现一组多输出逻辑函数

$$\begin{cases} F_1 = \overline{A}\overline{B} + \overline{B}C + AC \\ F_2 = \overline{A}\overline{B} + B\overline{C} + ABC \\ F_3 = \overline{A}C + BC + A\overline{C} \end{cases}$$

解 对于三变量多输出逻辑函数可以选用 3线—8线译码器实现。当使能控制端控制条件成立时：

$$\overline{Y_0} = \overline{m_0} = \overline{\overline{A_2}\overline{A_1}\overline{A_0}}$$

$$\overline{Y_1} = \overline{m_1} = \overline{\overline{A_2}\overline{A_1}A_0}$$

$$\overline{Y_2} = \overline{m_2} = \overline{\overline{A_2}A_1\overline{A_0}}$$

$$\overline{Y_3} = \overline{m_3} = \overline{\overline{A_2}A_1A_0}$$

$$\overline{Y_4} = \overline{m_4} = \overline{A_2\overline{A_1}\overline{A_0}}$$

$$\overline{Y_5} = \overline{m_5} = \overline{A_2\overline{A_1}A_0}$$

$$\overline{Y_6} = \overline{m_6} = \overline{A_2A_1\overline{A_0}}$$

$$\overline{Y_7} = \overline{m_7} = \overline{A_2A_1A_0}$$

4.2.1 采用MSI实现组合逻辑函数

写出多输出逻辑函数的最小项表达式，并做适当的变换

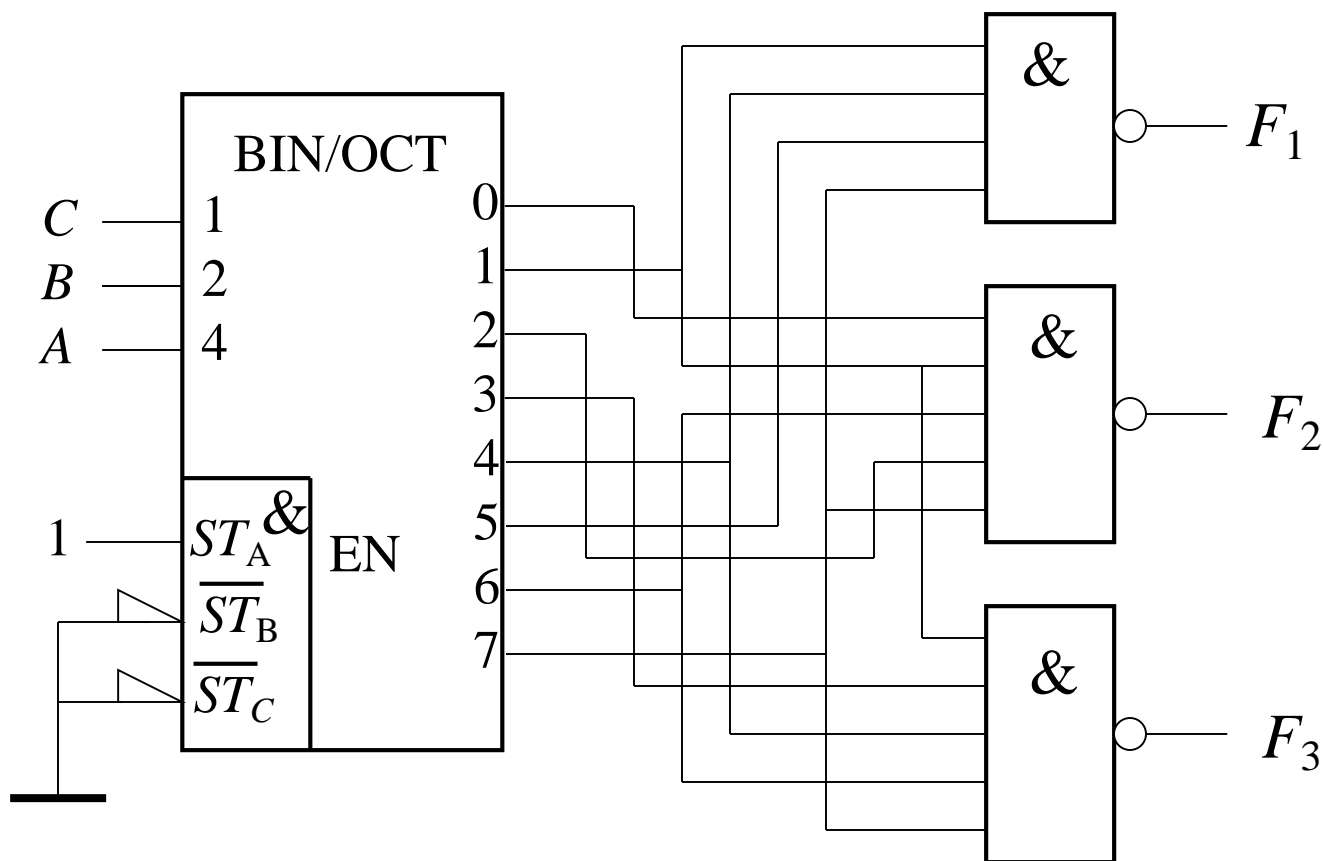
$$\begin{aligned} F_1 &= \overline{A}\overline{B} + \overline{B}C + AC = m_1 + m_4 + m_5 + m_7 \\ &= \overline{\overline{m_1} \cdot \overline{m_4} \cdot \overline{m_5} \cdot \overline{m_7}} = \overline{\overline{Y_1} \cdot \overline{Y_4} \cdot \overline{Y_5} \cdot \overline{Y_7}} \end{aligned}$$

$$\begin{aligned} F_2 &= \overline{A}\overline{B} + \overline{B}C + ABC = m_0 + m_1 + m_2 + m_6 + m_7 \\ &= \overline{\overline{m_0} \cdot \overline{m_1} \cdot \overline{m_2} \cdot \overline{m_6} \cdot \overline{m_7}} = \overline{\overline{Y_0} \cdot \overline{Y_1} \cdot \overline{Y_2} \cdot \overline{Y_6} \cdot \overline{Y_7}} \end{aligned}$$

$$\begin{aligned} F_3 &= \overline{A}C + \overline{B}C + A\overline{C} = m_1 + m_3 + m_4 + m_6 + m_7 \\ &= \overline{\overline{m_1} \cdot \overline{m_3} \cdot \overline{m_4} \cdot \overline{m_6} \cdot \overline{m_7}} = \overline{\overline{Y_1} \cdot \overline{Y_3} \cdot \overline{Y_4} \cdot \overline{Y_6} \cdot \overline{Y_7}} \end{aligned}$$

将输入变量A、B、C分别加到译码器的地址输入端 A_2 、 A_1 、 A_0 ，用与非门作为 F_2 、 F_1 、 F_0 的输出门，即可得到用3线—8线译码器实现 F_2 、 F_1 、 F_0 的逻辑电路

4.2.1 采用MSI实现组合逻辑函数



用加法器设计组合电路

➤ 基本原理:

➤ 若能生成函数可变换成

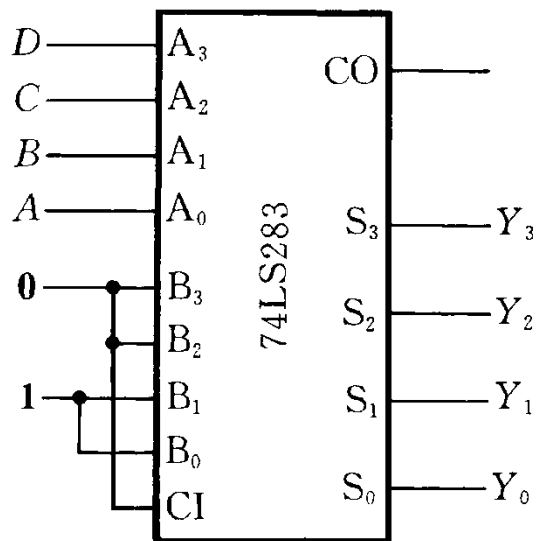
输入变量与输入变量相加

➤ 若能生成函数可变换成

输入变量与常量相加

例：将BCD的8421码转换为余3码

$$Y_3Y_2Y_1Y_0 = DCBA + 0011$$



输 入				输 出			
D	C	B	A	Y ₃	Y ₂	Y ₁	Y ₀
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

4.3 组合逻辑电路的冒险现象

此前讨论的组合逻辑电路设计都是在理想情况下进行的。实际上集成门和门与门之间的连线都有一定延迟时间，输入信号变化需要过渡时间，多个信号发生变化时也有先后和快慢的差异。受上述诸多因素的影响，在理想情况下设计的组合逻辑电路，便可能在输入信号发生变化的瞬间，在输出端出现一些不正确的尖峰信号，这些尖峰信号（毛刺信号）的出现称为冒险现象。

4.3.1 静态逻辑冒险

在组合逻辑电路中，如果输入信号变化前、后的稳定输出相同，而只在转换瞬间有冒险，称为静态冒险

如果输入信号变化前、后稳态输出为1,转换瞬间出现0的毛刺(序列为1—0—1),称这种静态冒险为静态0冒险

如果输入信号变化前、后稳态输出为0,转换瞬间出现1的毛刺(序列为0—1—0),称这种静态冒险为静态1冒险

在组合逻辑电路中也还可能发生动态冒险但本节只讨论静态冒险

4.3.2 如何判断是否存在逻辑冒险

发生静态逻辑冒险有两种情况

☆当有输入变量 A 和 \bar{A} 通过不同途径传输到输出端时，那么当输入变量 A 发生变化时，输出端可能产生静态逻辑冒险

表现为逻辑函数的表达式可以简化变换成

$$F=A+\bar{A} \quad \text{或} \quad F=A \cdot \bar{A}$$

的形式即可判断可能发生静态冒险

例如 $F=cd+\bar{b}\bar{d}+a\bar{c}$

$$\text{当 } abc=111 \text{ 时 } F=d+\bar{d}$$

$$abd=111 \text{ 时 } F=c+\bar{c}$$

都有可能发生静态冒险

4.3.2 如何判断是否存在逻辑冒险

☆当有两个或两个以上输入变量发生变化时输出端有可能出现静态冒险

静态逻辑冒险可以根据逻辑函数表达式来判断。

当 p ($p \geq 2$)个输入变量发生变化时，如果由不变的 $(n-p)$ 个变量组成的乘积项不是该逻辑函数表达式中的乘积项或多余项，则该 p 个变量发生变化时就有可能发生静态逻辑冒险

静态逻辑冒险也可以用逻辑函数卡诺图来判断。

当 p ($p \geq 2$)个输入变量发生变化时，如果由不变的 $(n-p)$ 个变量组成的乘积项所包含的2个方格中既有1又有0，则该 p 个变量发生变化时就有可能发生静态逻辑冒险

4.3.2 如何判断是否存在逻辑冒险

例如 $F = cd + b\bar{d} + a\bar{c}$

当 $abcd$ 由0100变化成1101时， a 、 d 发生了变化，由不变变量 b 、 \bar{c} 组成的乘积项 $b\bar{c}$ 既不是 F 的乘积项也不是多余项，可能产生静态逻辑冒险

在卡诺图中

当 $abcd$ 由0100变化成1101时，由不变变量 b 、 \bar{c} 组成的乘积项包含的4个方格中既有1又有0，可能产生静态逻辑冒险
变量 a 、 d 变化的先后次序(红、蓝两组箭头表示的途径)影响输出是否可能产生静态逻辑冒险

$cd \backslash ab$	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	1	1	1	1
10	0	1	1	0

4.3.2 如何判断是否存在逻辑冒险

当 $abcd$ 由0111变化成1110时，虽然由不变变量 b 、 c 组成的乘积项包含的4个方格中全部为1，不会产生静态逻辑冒险。但是在 $b=c=1$ 的特定条件下存在 $F=d+\bar{d}$ 的情况，因此也可能产生静态逻辑冒险

$cd \backslash ab$	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	1	1	1	1
10	0	1	1	0

当 $abcd$ 由1001变化成1011时，由于 $a=d=1$ 存在 $F=\bar{c}+c$ 的情况，因此 c 变量发生变化时，也可能产生静态逻辑冒险

4.3.3 如何避免逻辑冒险

- ❖修改逻辑设计 利用增加多余项来消除冒险，此法适用范围有限
- ❖引入取样脉冲 由于冒险现象仅仅发生在输入信号变化的瞬间，而稳定状态是没有冒险的，采用取样脉冲，错开输入信号发生转换的瞬间，正确反映组合逻辑电路稳定时的输出值可以有效地避免各种冒险
- ❖输出端加接滤波电容器 在输出端加接适当容量的滤波电容器可以在一定程度上滤除冒险产生的毛刺，使其降低到门电路可以容忍的电平