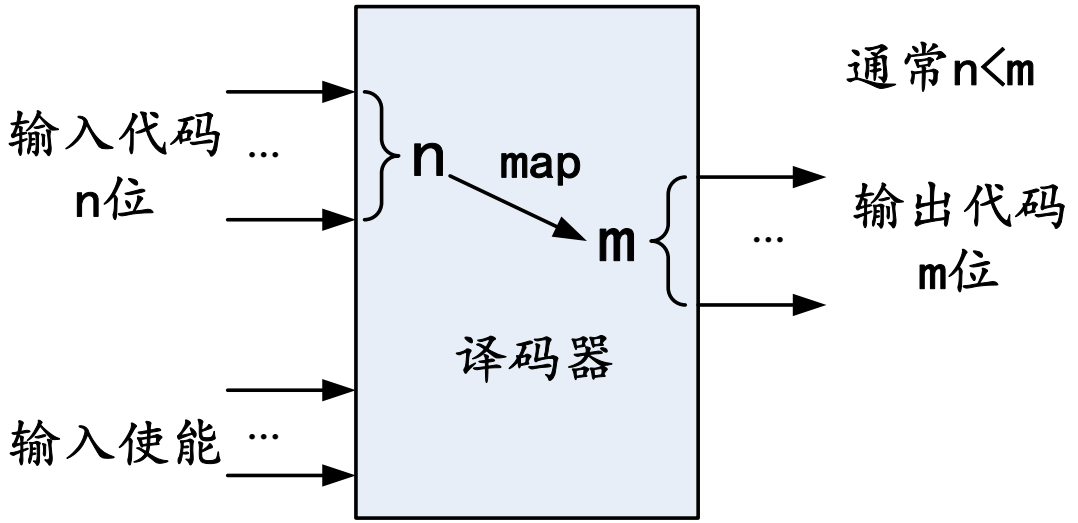
常用中规模组合逻辑器件及其应用

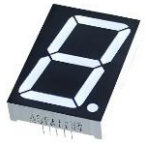
- 译码器
- 数据选择器

3.3 译码器

译码器（Decoder）是一种具有“翻译”功能的多输入、多输出组合逻辑电路，将输入二进制代码的各种状态按其原意翻译成对应的编码输出。



- **变量译码器**：一种较少输入变为较多输出的器件，分为二进制译码器和二-十进制译码器两类。
- **显示译码器**：用来驱动显示器件，以显示数字、字符或图形的器件。



二进制译码器：

n个输入端(即n位二进制码)

2^n个输出线

常见的有：

2-4译码器

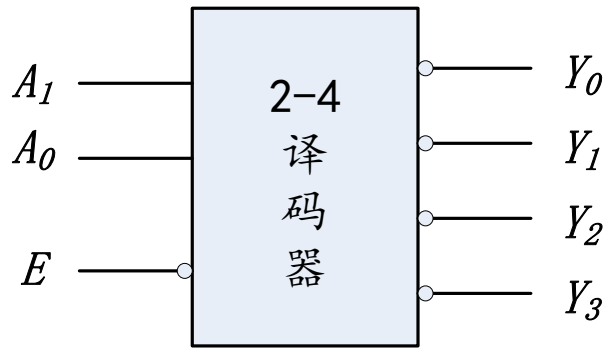
3-8译码器

4-16译码器

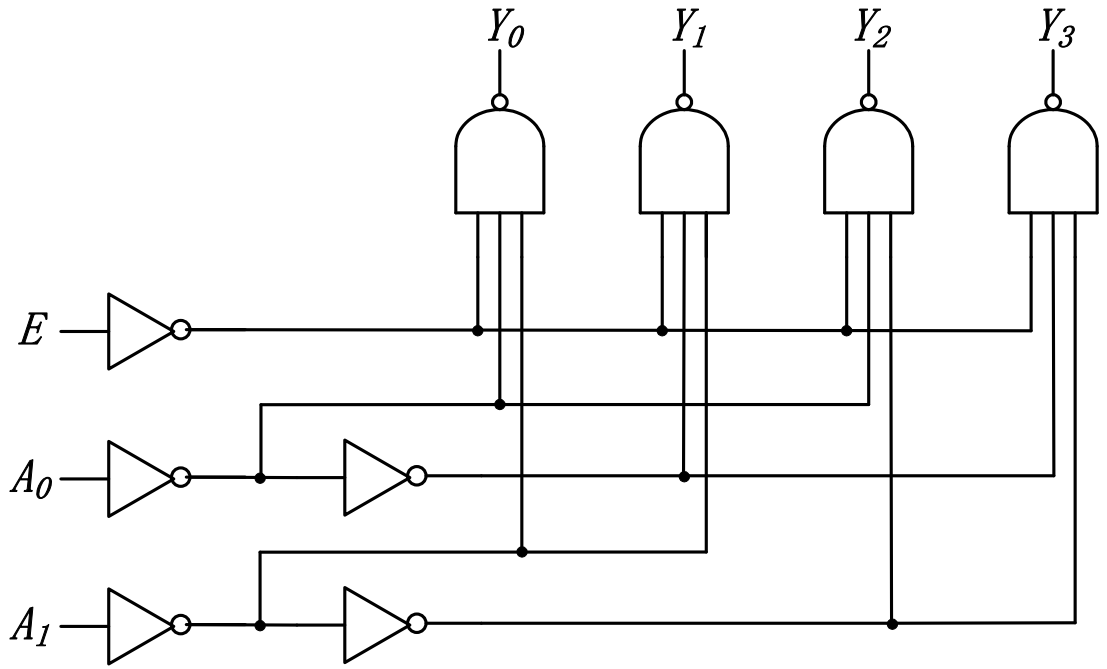
2-4译码器

2-4译码器功能表

E	A1	A0	Y0	Y1	Y2	Y3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0



2-4译码器符号图



2-4译码器逻辑电路

E	A1	A0	Y0	Y1	Y2	Y3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

E=0时

A1	A0	Y0	Y1	Y2	Y3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

当E=0时，2-4译码器的输出函数分别为：

$$Y_0 = \overline{\overline{A_1}\,\overline{A_0}} \qquad Y_1 = \overline{\overline{A_1}A_0}$$

$$Y_2 = \overline{A_1\overline{A_0}} \qquad Y_3 = \overline{A_1 A_0}$$

$$\Rightarrow \quad Y_i = \overline{m_i}$$

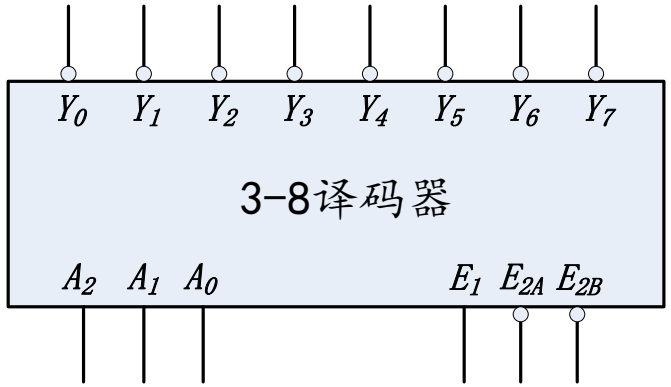
考虑上E，得到：

E	Yi
0	$\overline{m_i}$
1	1

$$Y_i = \overline{E}\,\overline{m_i} + E = E + \overline{m_i} = \overline{\overline{E}m_i}$$

$$Y_i = \overline{\overline{E}m_i}\,(i = 0,1,2,3)$$

3-8译码器



E1	E_{2A}+E_{2B}	A_2	A_1	A_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	X	X	X	X	1	1	1	1	1	1	1	1
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
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

推导其输出端的表达式：

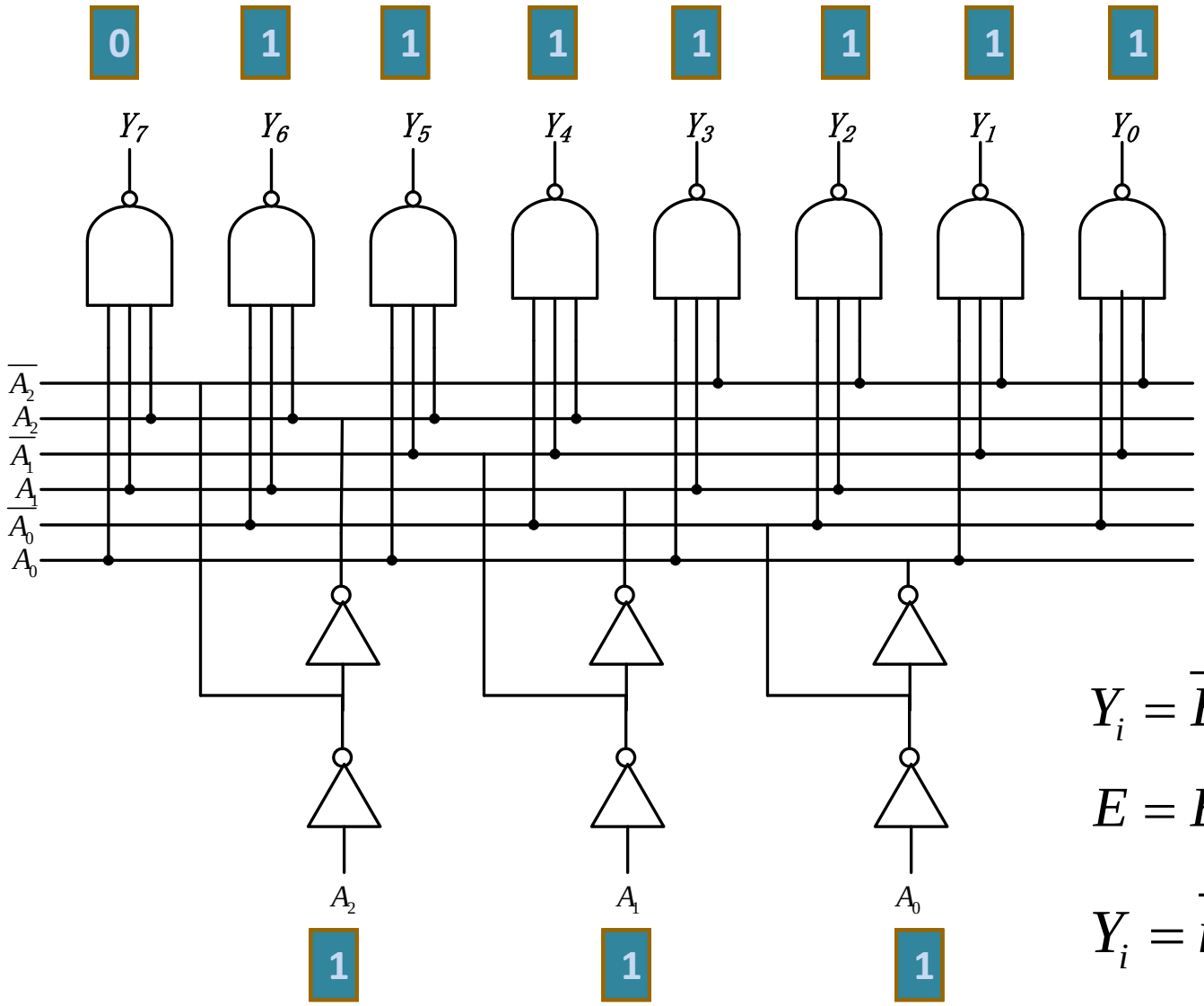
$$\overline{Y_i} = E_1\overline{E_{2A} + E_{2B}}mi$$

$$Y_i = \overline{Em_i}$$

$$E = E_1\overline{E_{2A} + E_{2B}} = E_1\overline{E_{2A}}\,\overline{E_{2B}}$$

3-8 译码器逻辑电路

工作原理



输出低电平有效

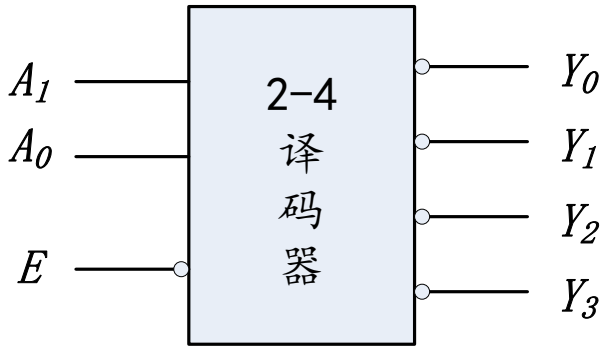
$$Y_i = \overline{Em_i} \qquad (i = 0 \sim 7)$$

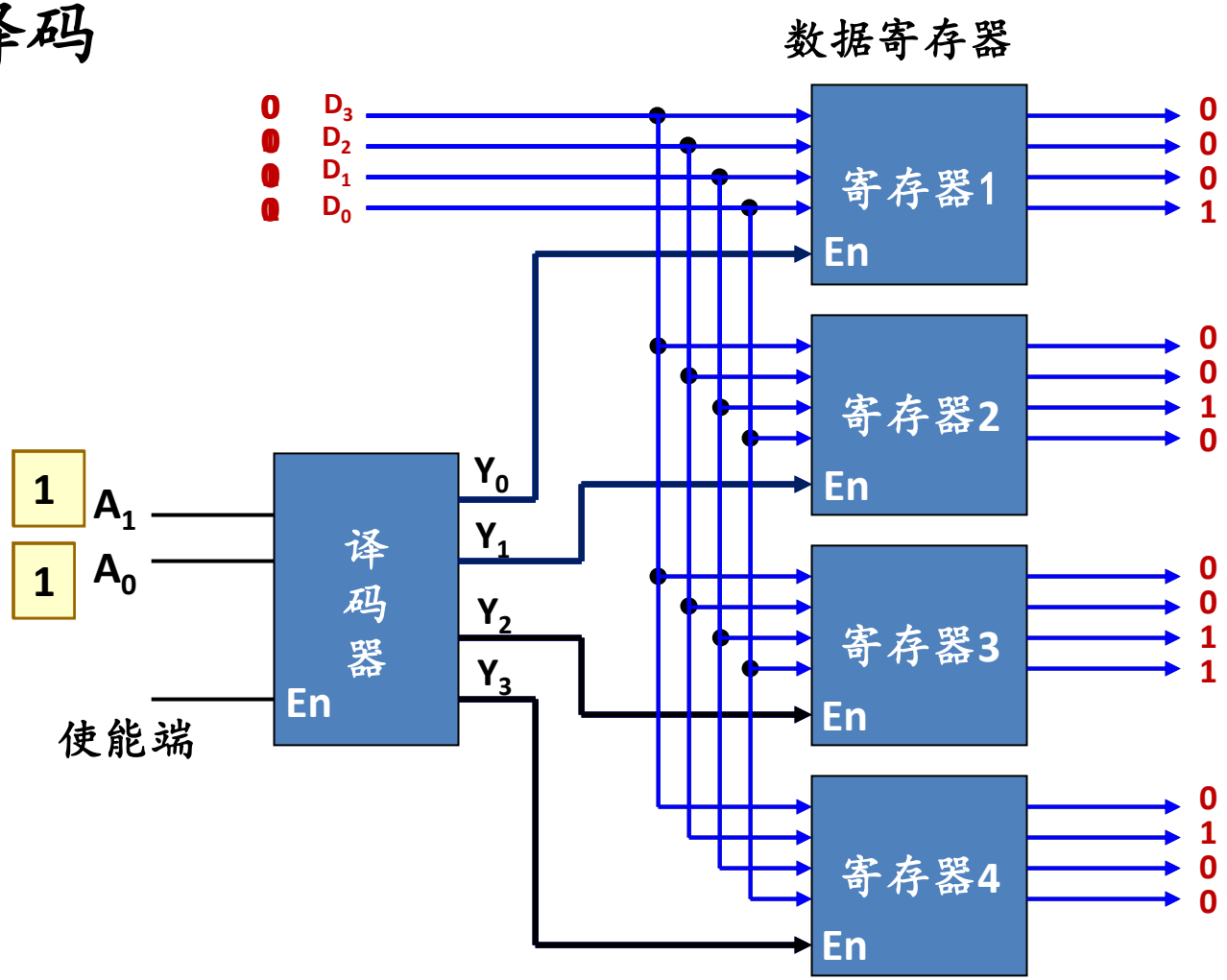
$$E = E_1 \cdot \overline{E_{2A} + E_{2B}} = E_1 \cdot \overline{E_{2A}} \cdot \overline{E_{2B}}$$

$Y_i = \overline{m_i}$ （当E有效时）

3.4.1 典型应用之一：

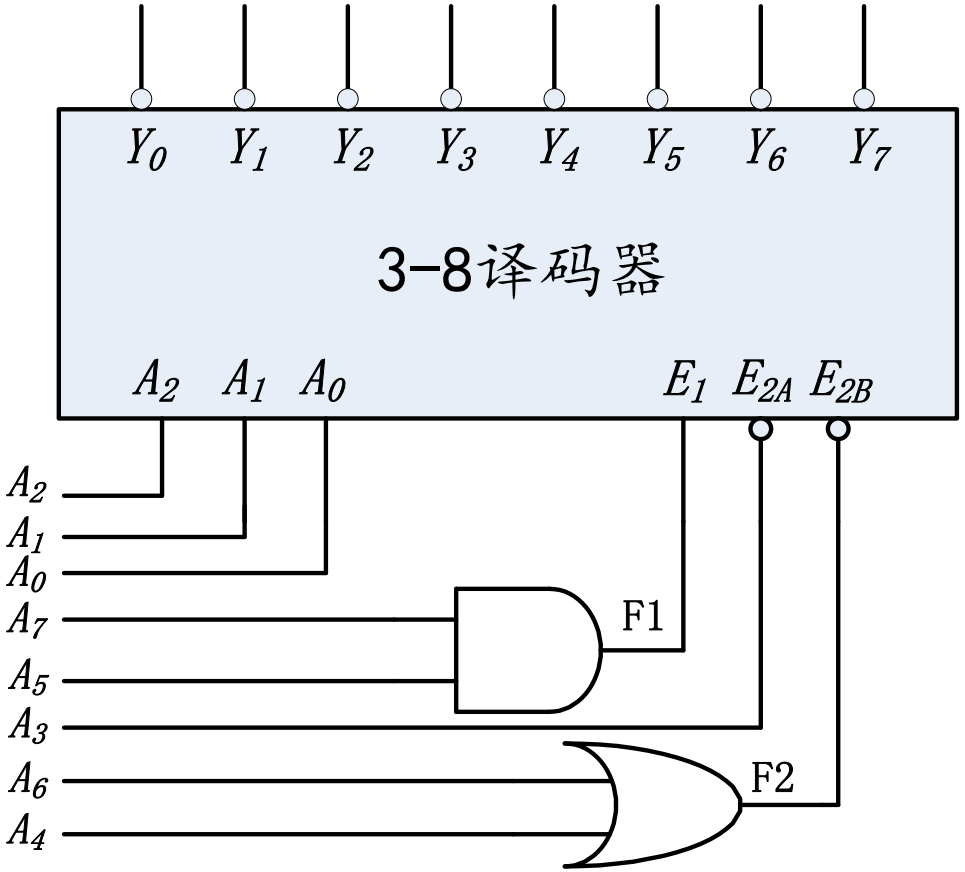
实现存储系统的地址译码





典型应用之一：实现存储系统的地址译码

例1：分析下图所示Y0-Y7的译码地址。



$F1=1 \rightarrow A_5A_7=1 \rightarrow A_5=1 \quad A_7=1$

$F2=0 \rightarrow A_4+A_6=0 \rightarrow A_4=0 \quad A_6=0$

$A_3=0$

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
1	0	1	0	0	x	x	x

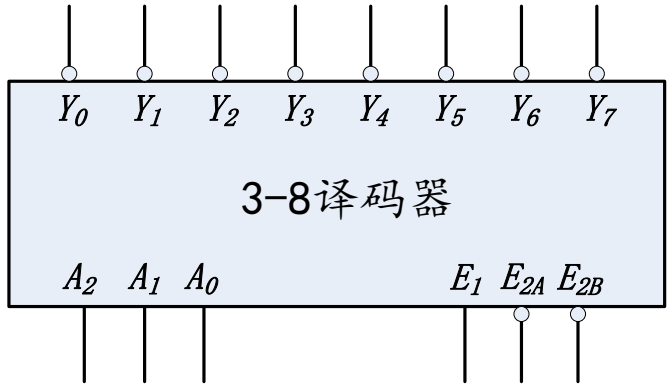
地址为：$(10100000)_2$-$(10100111)_2$

（A0H - A7H）

思考题：请用74138设计一地址译码电路，实现2E0-2E7的地址译码。

提示：

2E0H-2E7H，对应的地址线为：



A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
1	0	1	1	1	0	0	0	0	0
1	0	1	1	1	0	0	0	0	1
1	0	1	1	1	0	0	0	1	0
1	0	1	1	1	0	0	0	1	1
1	0	1	1	1	0	0	1	0	0
1	0	1	1	1	0	0	1	0	1
1	0	1	1	1	0	0	1	1	0
1	0	1	1	1	0	0	1	1	1

3.4.2 典型应用之二：实现组合逻辑函数

【例 2】 试用3-8译码器实现函数：

$$F_1 = \sum m(0,4,7)$$

$$F_2 = \sum m(1,2,3,5,6,7)$$

分析：因为当译码器的使能端有效时，每个输出 $Y_i = \overline{m}_i = M_i$ ，因此只要将函数的输入变量加至译码器的地址输入端，并在输出端辅以少量的门电路，便可以实现逻辑函数。

$$F_1 = \sum m(0,4,7)$$

$$F_2 = \sum m(1,2,3,5,6,7)$$

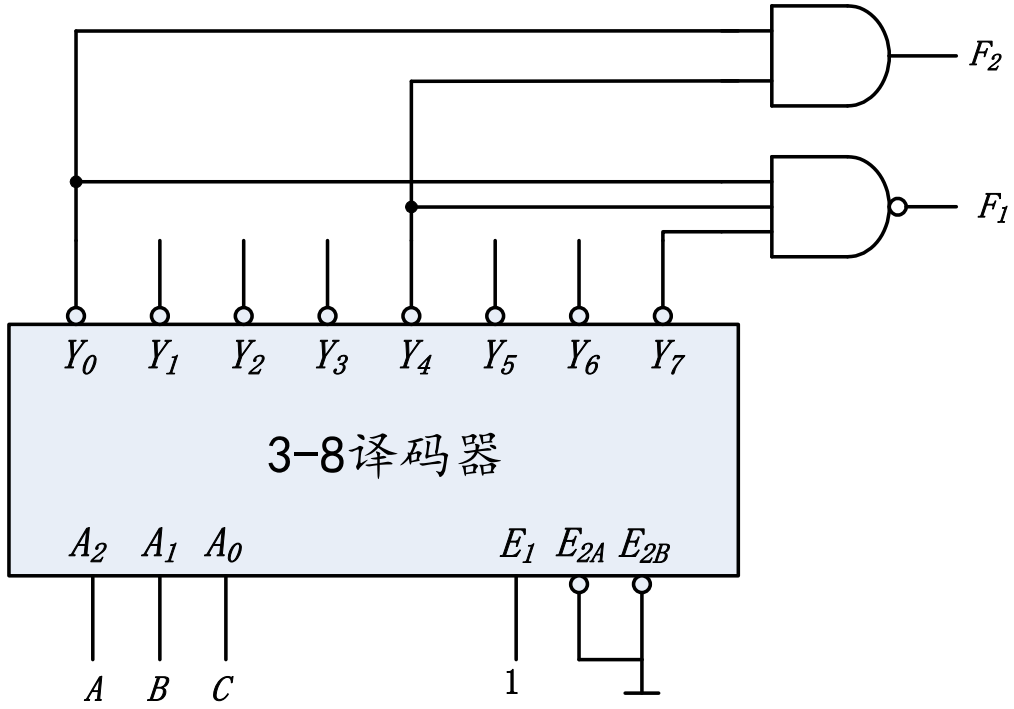
$$F_1 = m_0 + m_4 + m_7 = \overline{\overline{m}_0 \cdot \overline{m}_4 \cdot \overline{m}_7} = \overline{Y_0 \cdot Y_4 \cdot Y_7}$$

$$F_2 = m_1 + m_2 + m_3 + m_5 + m_6 + m_7$$

$$= M_0 \cdot M_4 = Y_0 \cdot Y_4$$

A B / C	00	01	11	10
0	0	1	1	0
1	1	1	1	1

F_2



思考题1：

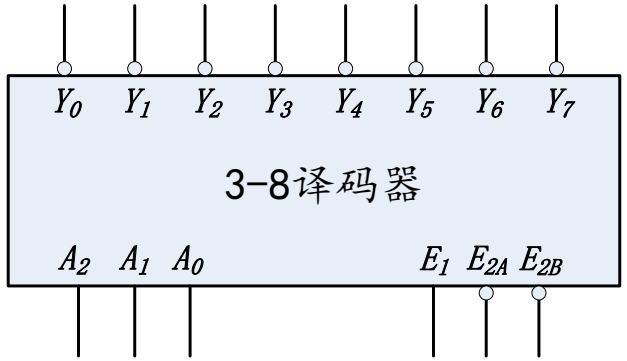
用74138和门电路实现下面的逻辑函数。

$$f_1(a,b,c)=\sum m(1,2,4,5)$$

$$f_2(a,b,c)=\prod M(2,3,6,7)$$

思考题2：

用74138和门电路设计一个一位二进制全加器。



一位二进制全加器真值表

A_i	B_i	C_i	C_{i+1}	S_i
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

3.4.3 二进制译码器的扩展

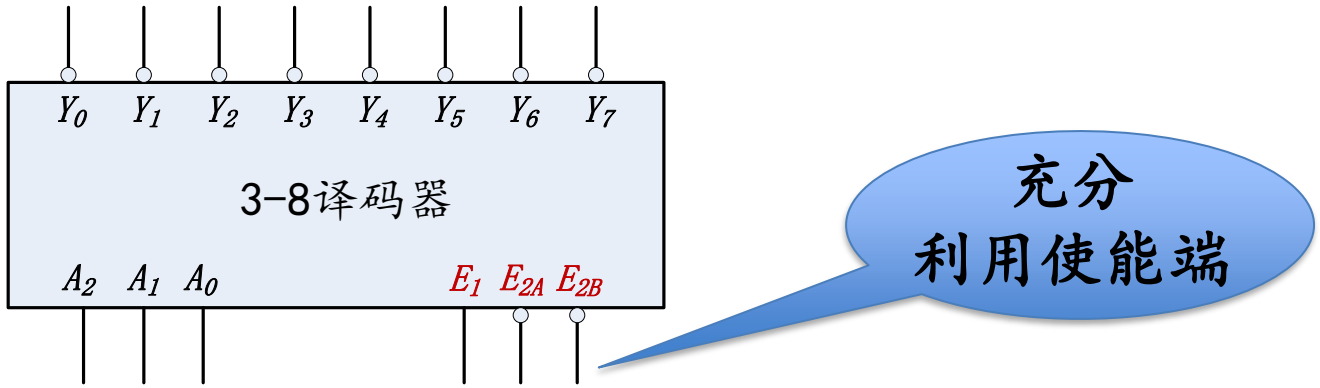
通常用译码器的使能端来实现二进制译码器的扩展

例1. 用3-8译码器实现4-16译码器。

分析：

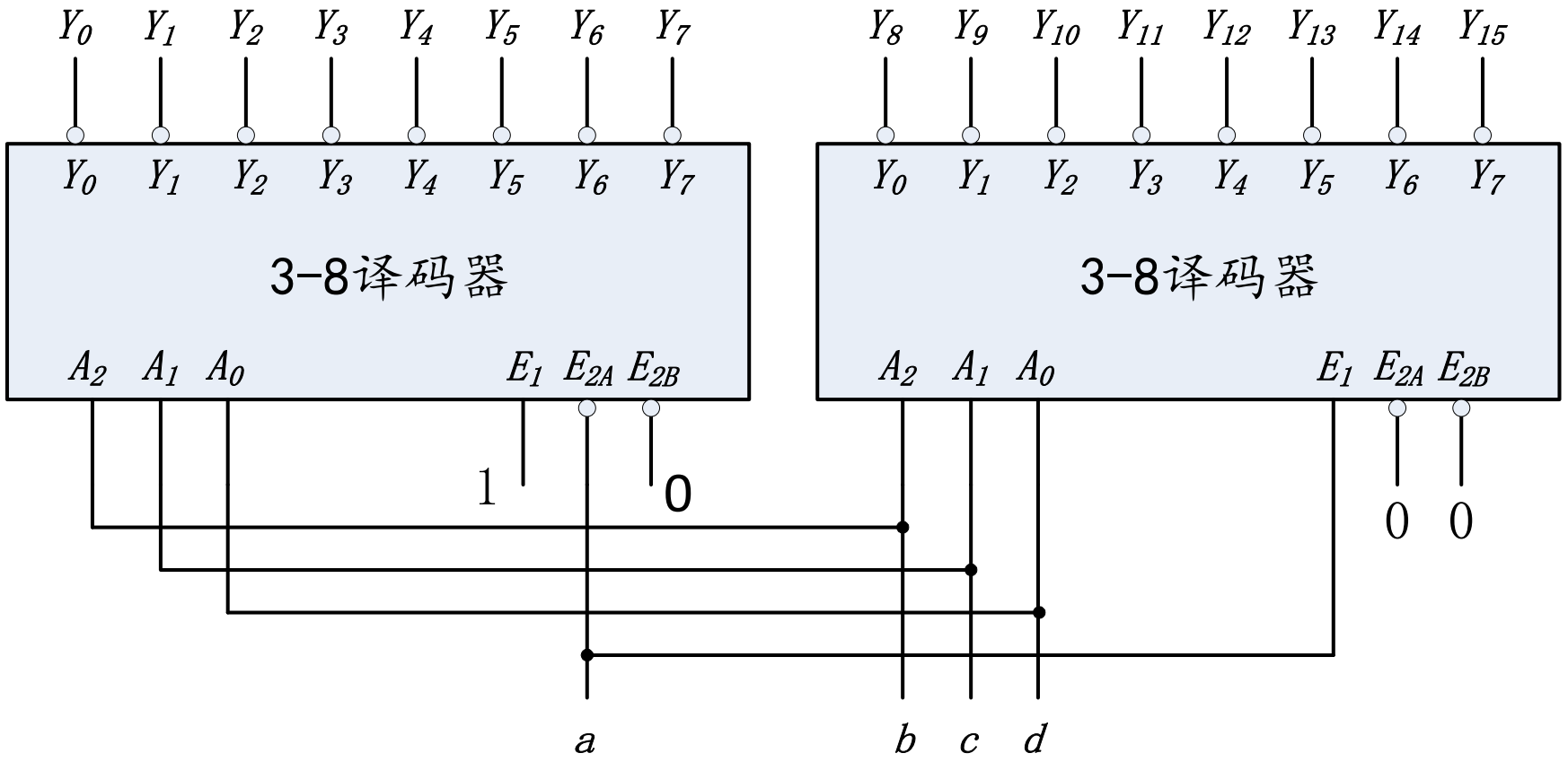
3-8译码器：3个输入端，8个输出端

4-16译码器：4个输入端，16个输出端



a	b	c	d	Y
0	0	0	0	Y_0
0	0	0	1	Y_1
0	0	1	0	Y_2
0	0	1	1	Y_3
0	1	0	0	Y_4
0	1	0	1	Y_5
0	1	1	0	Y_6
0	1	1	1	Y_7
1	0	0	0	Y_8
1	0	0	1	Y_9
1	0	1	0	Y_{10}
1	0	1	1	Y_{11}
1	1	0	0	Y_{12}
1	1	0	1	Y_{13}
1	1	1	0	Y_{14}
1	1	1	1	Y_{15}

二进制译码器的扩展

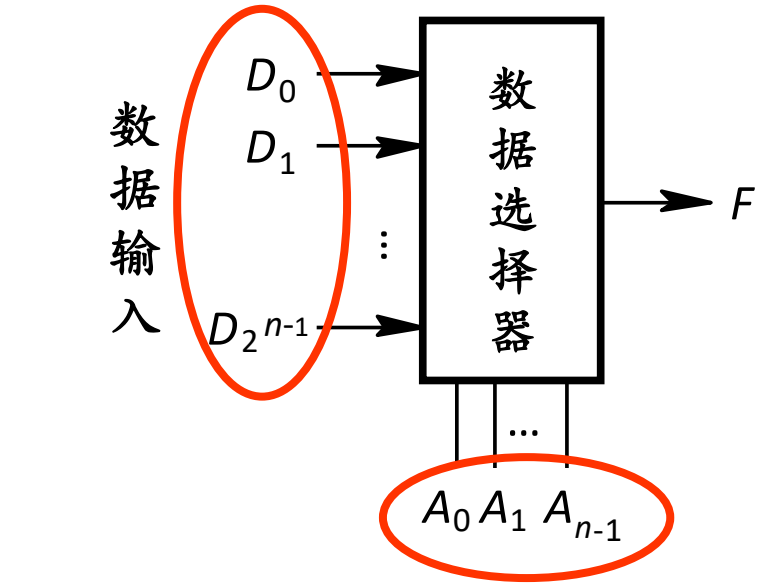


3.5 数据选择器功能

（Multiplexer，简称MUX）

数据选择器又称多路选择器。它有*n*位地址输入、2^n位数据输入、1位输出。

每次在地址输入的控制下，从多路输入数据中选择一路输出，其功能类似于一个单刀多掷开关。



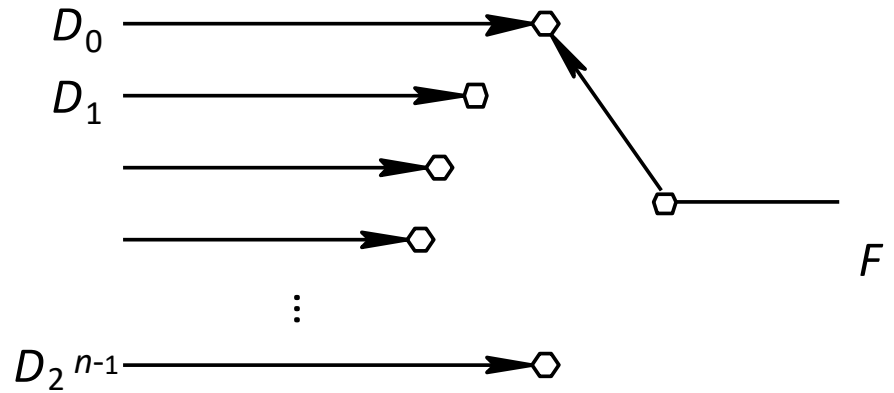
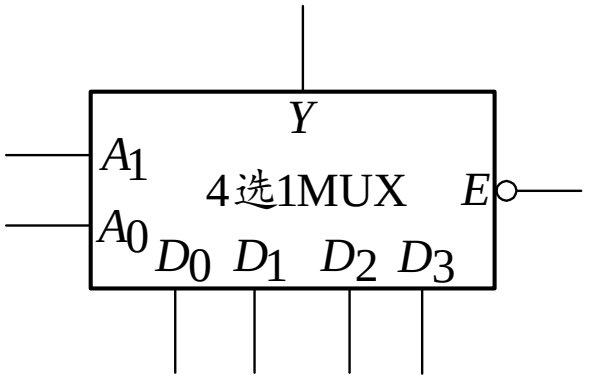
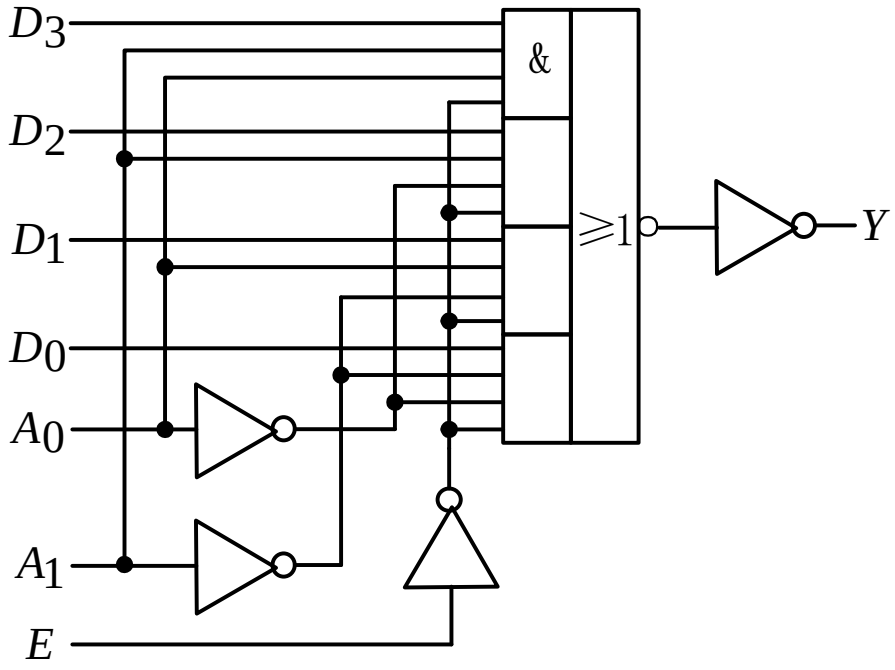


图 1 数据选择器框图及等效开关

4选1数据选择器



(b) 逻辑符号



(a) 逻辑图

***D0~D3*是数据输入端，也称为数据通道；**

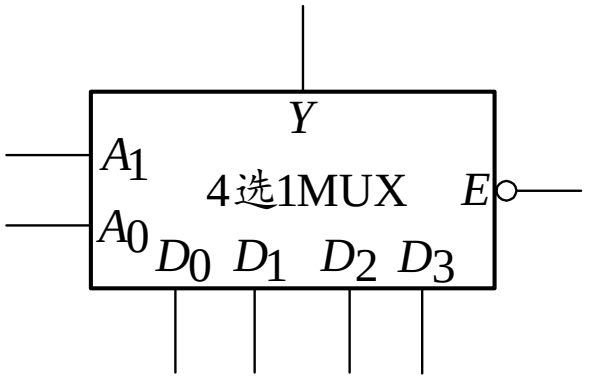
A_1、A_0是地址输入端，或称选择输入端；

***Y*是输出端；**

***E*是使能端，低电平有效。**

常用的数据选择器有：

2选1、4选1、8选1、16选1等。



(*b*) 逻辑符号

4选1MUX功能表

E	A_1	A_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	×	×	0

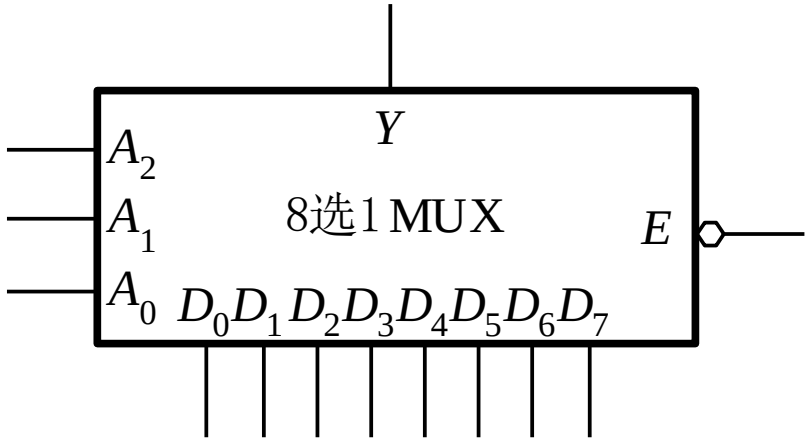
当E=0时，其逻辑功能的表达式：

$$Y = \overline{A}_1\overline{A}_0D_0 + \overline{A}_1A_0D_1 + A_1\overline{A}_0D_2 + A_1A_0D_3$$

$$= \sum_{i=0}^{3} m_iD_i$$

$$Y = (\overline{A}_1\overline{A}_0 \quad \overline{A}_1A_0 \quad A_1\overline{A}_0 \quad A_1A_0)\begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \end{bmatrix} = (A_1A_0)_m(D_0D_1D_2D_3)^T$$

8选1数据选择器



(a)符号图

$$Y=\sum_{i=0}^{7} m_i D_i$$

$$=(A_2A_1A_0)_m(D_0D_1D_2D_3D_4D_5D_6D_7)^T$$

(c)表达式

(b)功能表

E	A_2	A_1	A_0	Y
1	×	×	×	0
0	0	0	0	D_0
0	0	0	1	D_1
0	0	1	0	D_2
0	0	1	1	D_3
0	1	0	0	D_4
0	1	0	1	D_5
0	1	1	0	D_6
0	1	1	1	D_7

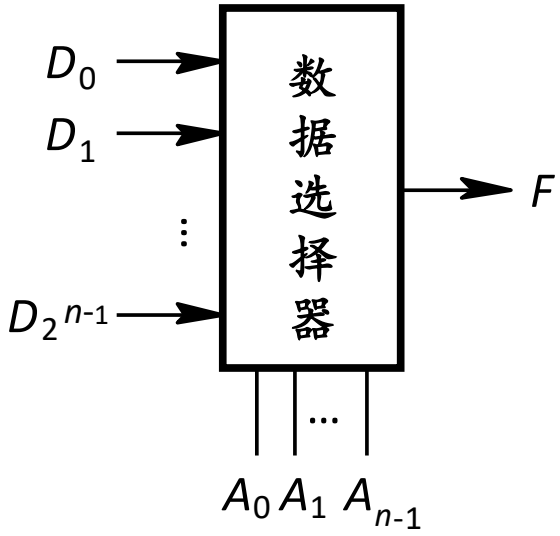
数据选择器的典型应用：

① 作数据选择，以实现多路信号分时传送。

② 实现组合逻辑函数。

③ 在数据传输时实现并—串转换。

④ 产生序列信号。



3.6 数据选择器应用之：实现逻辑函数

对于*n*个地址输入的MUX，其表达式为：

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

其中mi是由地址变量An-1、…、A1、A0组成的地址最小项。

任何一个具有*l*个输入变量的逻辑函数都可以用最小项之和来表示：

$$F=\sum_{i=0}^{2^l-1} m_i (1, or 0)$$

这里的mi是由函数的输入变量A、B、C、…组成的最小项。

1) $l \le n$的情况

(l为函数的输入变量数，n为选用的MUX的地址输入端数)

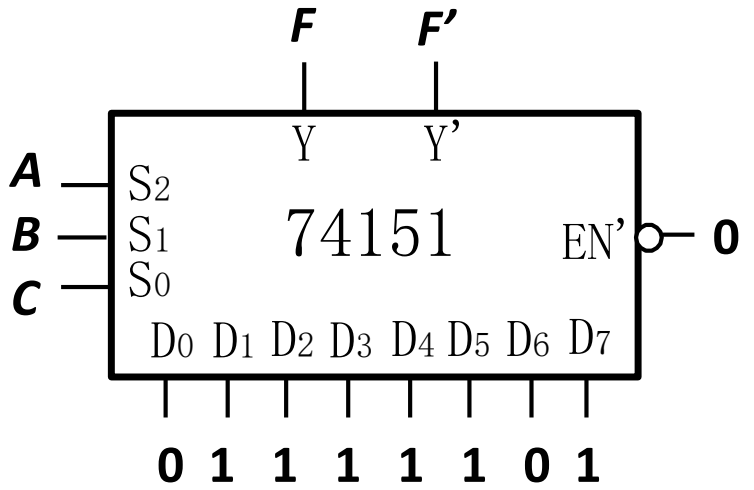
当$l=n$时，只要将函数的输入变量A、B、C、…依次接到MUX的地址输入端，根据函数F所需要的最小项，确定MUX中D_i的值(0或1)即可；

当$l<n$时，将MUX的高位地址输入端不用(接0或1)，其余同上。

$$Y=\sum_{i=0}^{2^n-1} m_i D_i \qquad F=\sum_{i=0}^{2^l-1} m_i (1, or 0)$$

例1. 用8选1 实现下面的组合逻辑函数.

$$F=\sum m(1,2,3,4,5,7)$$



从F可以看出, 该逻辑函数包含3 个变量 (设为A、B、C),

一个 8选1 数据选择器有 3 个地址输入端。

$$Y=\sum_{i=0}^{7} m_i D_i=(A_2A_1A_0)(D_0D_1D_2D_3D_4D_5D_6D_7)^T$$

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

$$F=\sum m(1,2,3,4,5,7)=m_1+m_2+m_3+m_4+m_5+m_7$$

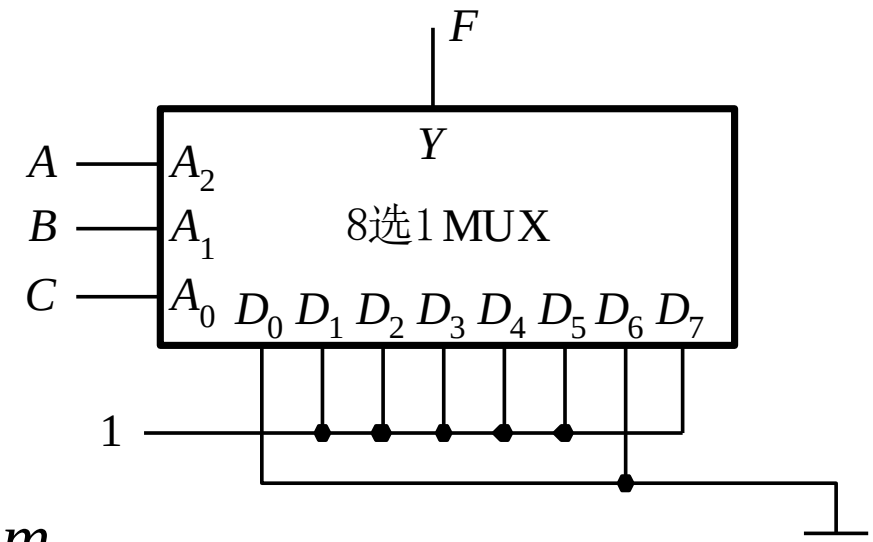
$$=m_0\cdot 0+m_1\cdot 1+m_2\cdot 1+m_3\cdot 1+m_4\cdot 1+m_5\cdot 1+m_6\cdot 0+m_7\cdot 1$$

例2. 试用8选1MUX实现逻辑函数：

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

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

F



$$F = \sum m(1,2,3,4,5,7) = m_1 + m_2 + m_3 + m_4 + m_5 + m_7$$

$$= m_0 \cdot 0 + m_1 \cdot 1 + m_2 \cdot 1 + m_3 \cdot 1 + m_4 \cdot 1 + m_5 \cdot 1 + m_6 \cdot 0 + m_7 \cdot 1$$

注意:因为函数F中各最小项的标号是按A、B、C的权为4、2、1写的，因此A、B、C必须依次加到A_2、A_1、A_0端。

数据选择器应用之：实现逻辑函数

当l＞n的情况

当逻辑函数的变量数l大于MUX的地址输入端数n时，不能采用前面所述的简单方法。

如果从l个输入变量中选择n个直接作为MUX的地址输入，那么，多余的$(l-n)$个变量就要反映到MUX的数据输入D_i端，即D_i是多余输入变量的函数，简称余函数。

因此设计的关键是如何求出函数$D_i=f(l-n)$。

例. 用4选1的MUX实现逻辑函数：$F=\sum m(1,2,3,4,5,7)$

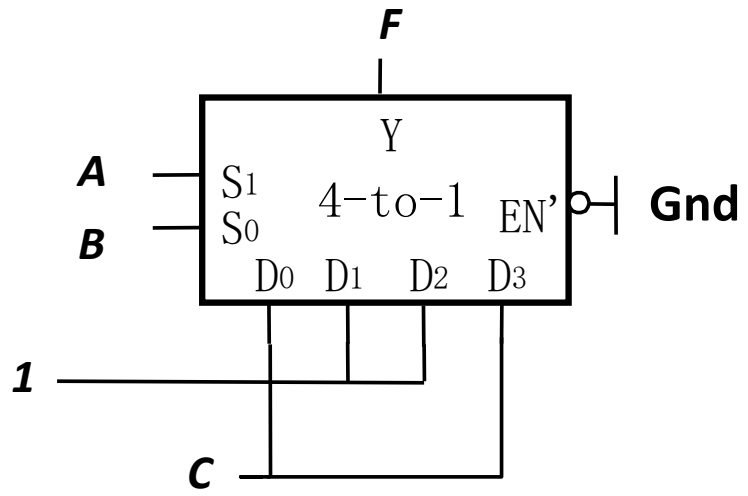
变量数 l > 数据选择器地址输入端数n

$$Y=\sum_{i=0}^{3} m_i D_i=(A_1A_0)(D_0D_1D_2D_3)^T=m_0\cdot D_0+m_1\cdot D_1+m_2\cdot D_2+m_3\cdot D_3$$

$$F=\sum m(1,2,3,4,5,7)$$
$$=\bar{A}\bar{B}C+\bar{A}B\bar{C}+\bar{A}BC+A\bar{B}\bar{C}+A\bar{B}C+ABC$$

必须选择2个变量作为数据选择器的地址端

$$F=\bar{A}\bar{B}C+\bar{A}B\bar{C}+\bar{A}BC+A\bar{B}\bar{C}+A\bar{B}C+ABC$$
$$=m_0C+m_1\bar{C}+m_1C+m_2\bar{C}+m_2C+m_3C$$
$$=m_0C+m_1\cdot 1+m_2\cdot 1+m_3C$$



降维K图法求余子式D_i

n变量的逻辑函数，可以用n维(即n变量)K图表示，也可以用(n-1)、(n-2)、…维K图表示，这种(n-1)、(n-2)、…维K图称为降维K图。

降维K图法求余子式D_i的求解步骤：

① 画出函数F的K图

② 选择地址输入

③ 在F的K图上确定余函数D_i的范围(子K图)

④ 求余函数D_i

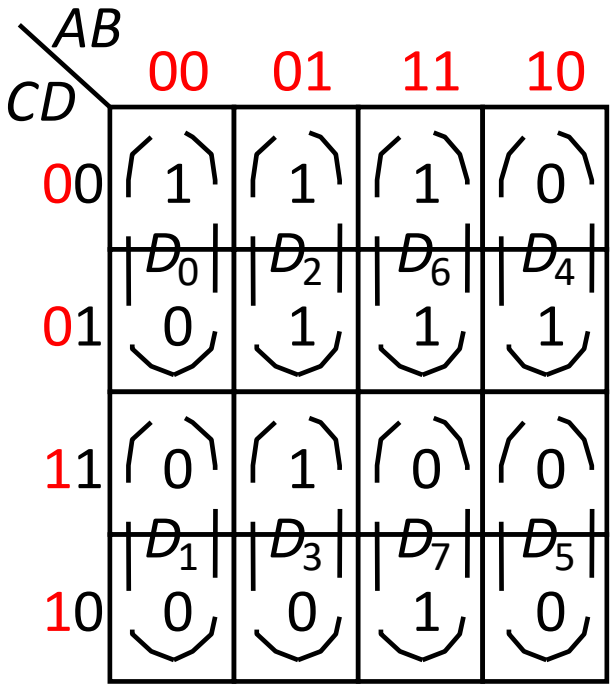
⑤ 画出逻辑图

例. 试用8选1MUX实现逻辑函数：$F(A,B,C,D)=\sum m(0,4,5,7,9,12,13,14)$

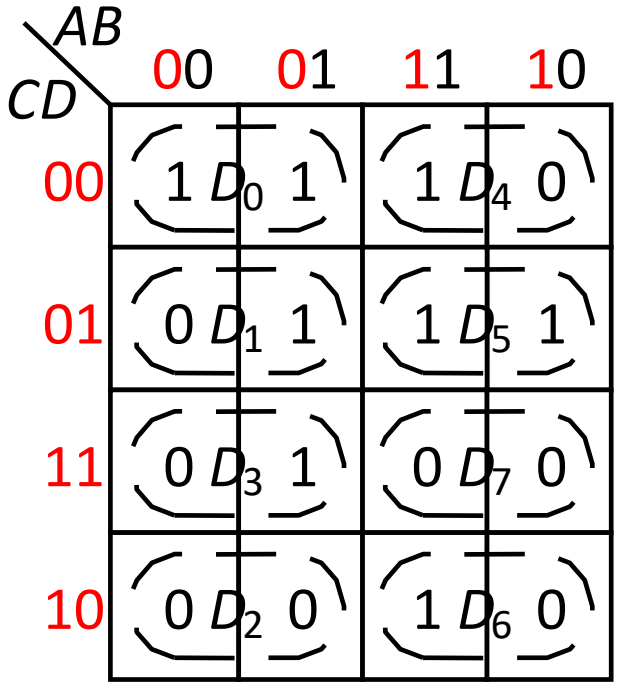
① 画出F的四变量K图

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

② 选择地址变量,确定余函数Di

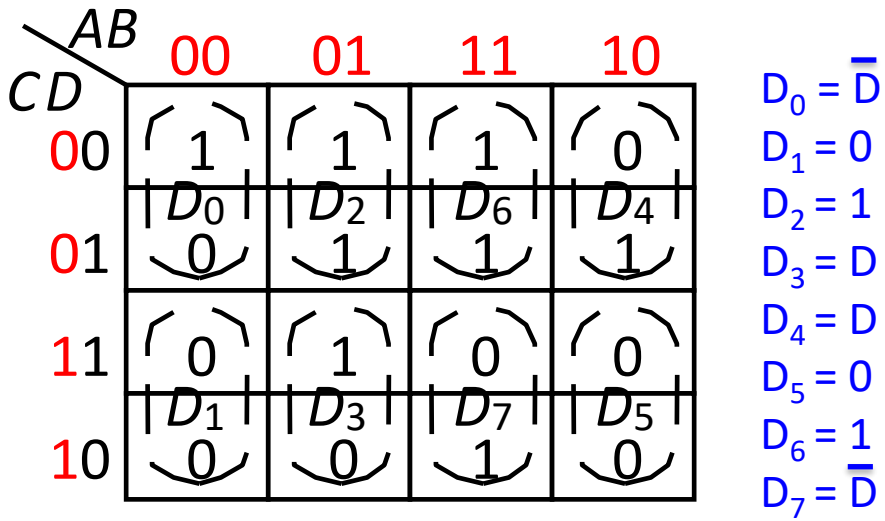


选择地址（$A_2A_1A_0$=ABC）



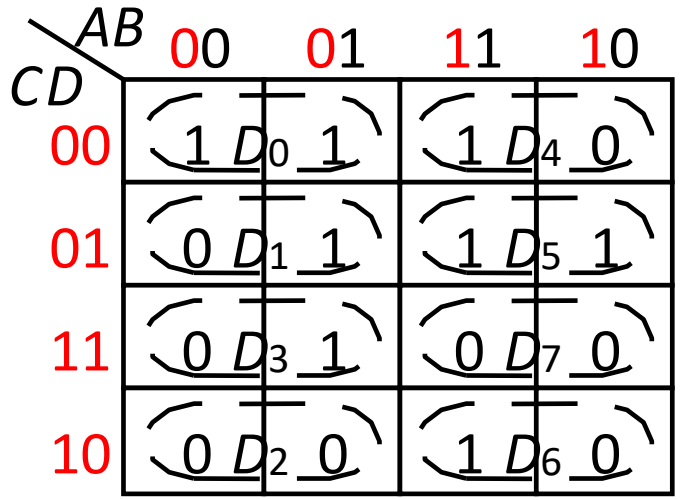
选择地址（$A_2A_1A_0$=ACD）

分别确定D_0、D_1、…、D_7。



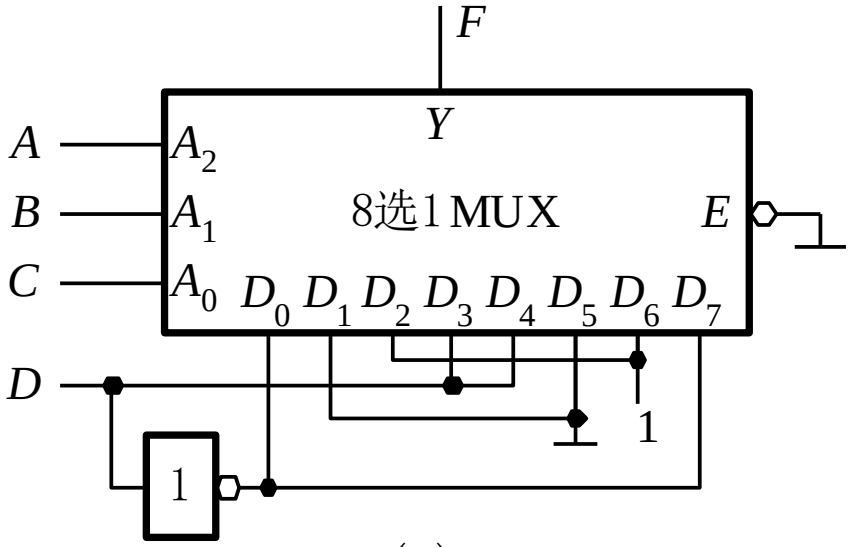
$D_0 = \overline{D}$
$D_1 = 0$
$D_2 = 1$
$D_3 = D$
$D_4 = D$
$D_5 = 0$
$D_6 = 1$
$D_7 = \overline{D}$

选择地址（$A_2A_1A_0$=ABC）

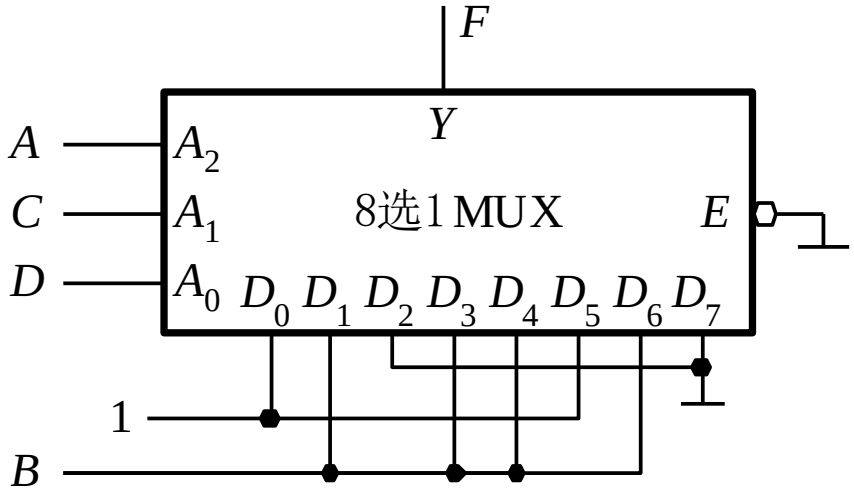


$D_0 = 1$
$D_1 = B$
$D_2 = 0$
$D_3 = B$
$D_4 = B$
$D_5 = 1$
$D_6 = B$
$D_7 = 0$

选择地址（$A_2A_1A_0$=ACD）



(a)

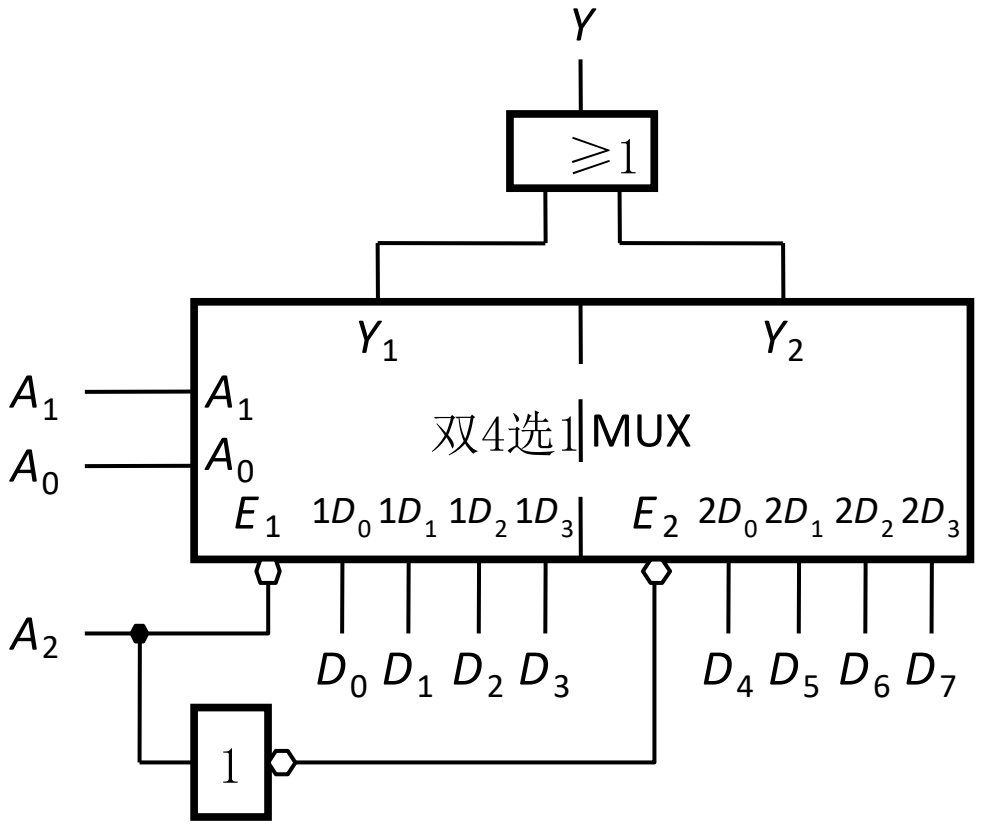


(b)

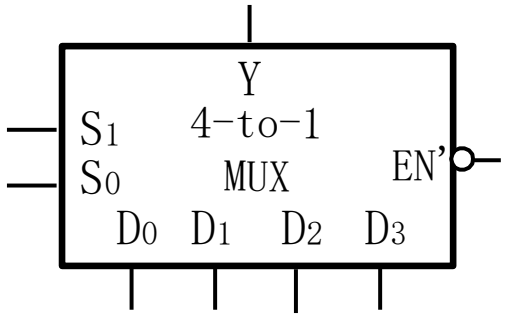
3.7 数据选择器的扩展

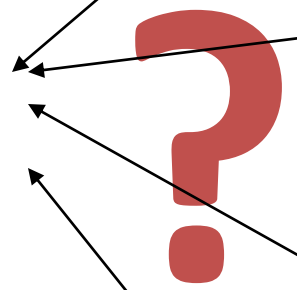
例.试用两个4选1 MUX扩展为8选1 MUX。

A_2	A_1	A_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



用4-to-1MUX实现16-to-1MUX





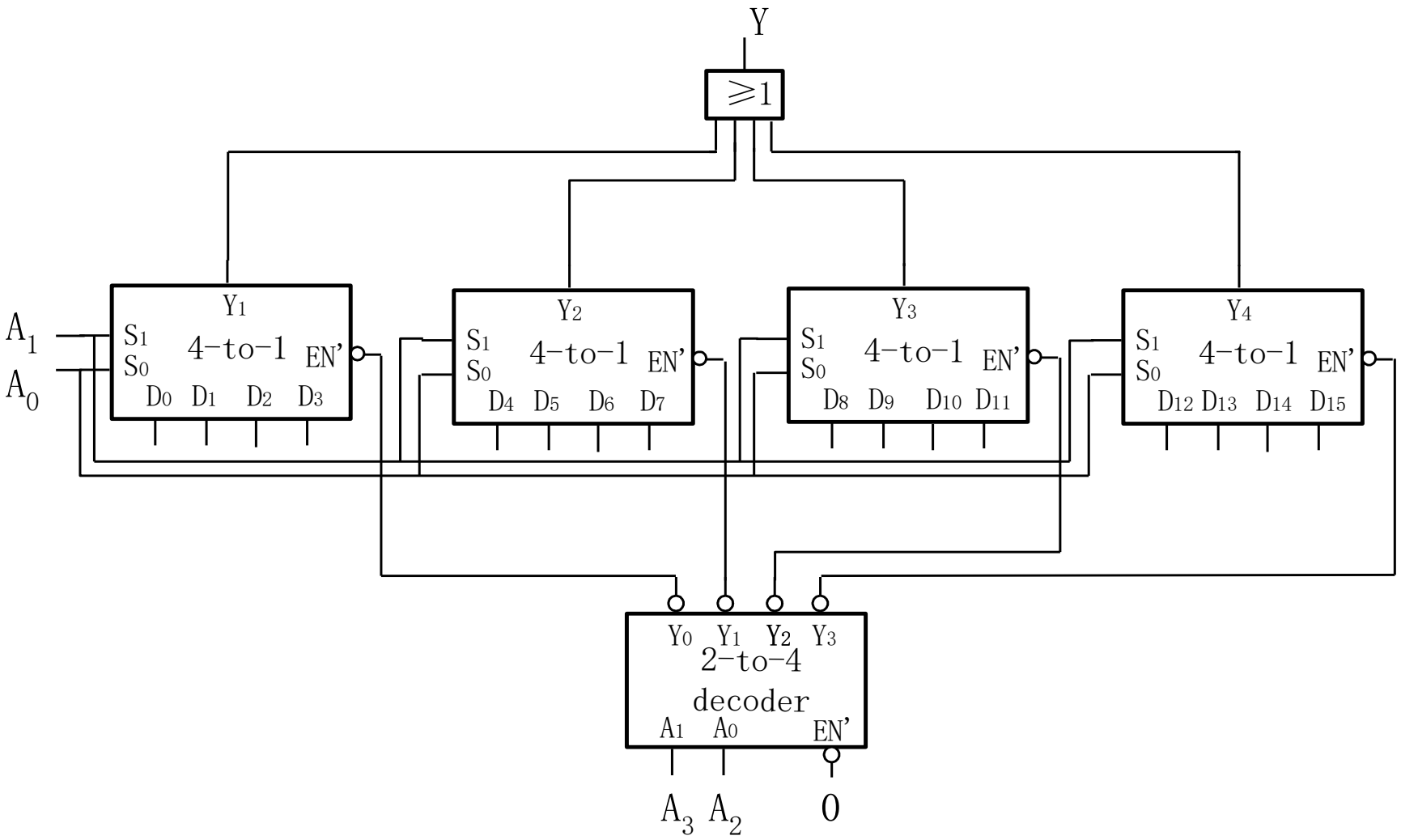
A_3	A_2	A_1	A_0	Y
0	0	0	0	D_0
0	0	0	1	D_1
0	0	1	0	D_2
0	0	1	1	D_3
0	1	0	0	D_4
0	1	0	1	D_5
0	1	1	0	D_6
0	1	1	1	D_7
1	0	0	0	D_8
1	0	0	1	D_9
1	0	1	0	D_{10}
1	0	1	1	D_{11}
1	1	0	0	D_{12}
1	1	0	1	D_{13}
1	1	1	0	D_{14}
1	1	1	1	D_{15}

实现方案：

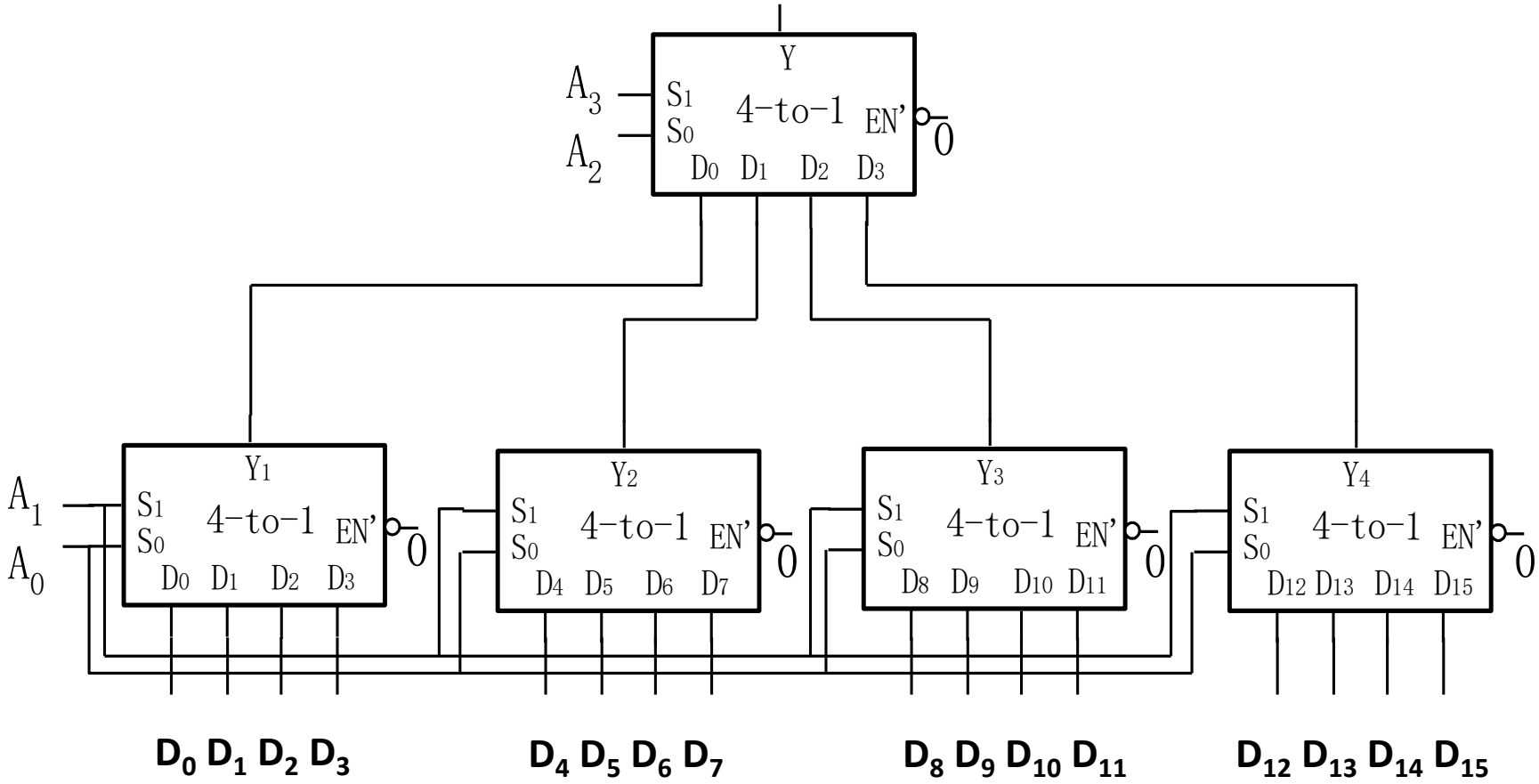
1.用A_3 和A_2 控制4-to-1的 使能

2.用A_3 和A_2 控制4-to-1的 输出

方案一：用A_3和A_2控制4-to-1的 使能实现16-to-1



方案二：用A_3和A_2控制4-to-1的输出实现16-to-1



3.8 组合逻辑电路中的竞争与冒险

一、竞争、冒险

由于信号通过导线和逻辑门将产生时间延迟，因此信号经过不同路径（不同数目的门、不同长度导线的传输）到达电路中同一逻辑门时，不同输入端的时间有差异。

竞争：信号经不同路径到达某一点时，所用的时间不同，这个时间差称之为竞争

冒险：由于竞争使得电路产生了暂时错误输出称之为冒险。由于竞争，在输出端将发生瞬时错误，表现为输出端产生了错误的尖峰脉冲（或称毛刺）。

说明： (1) 一般来说，时延对数字系统是有害的，它会降低系统的工作速度，还会产生竞争冒险现象。

(2) 毛刺的存在不仅可能导致负载电路误动作，还会增大电路的功耗。

(3) 组合电路中的竞争是普遍现象，但不是所有的竞争都会产生冒险。

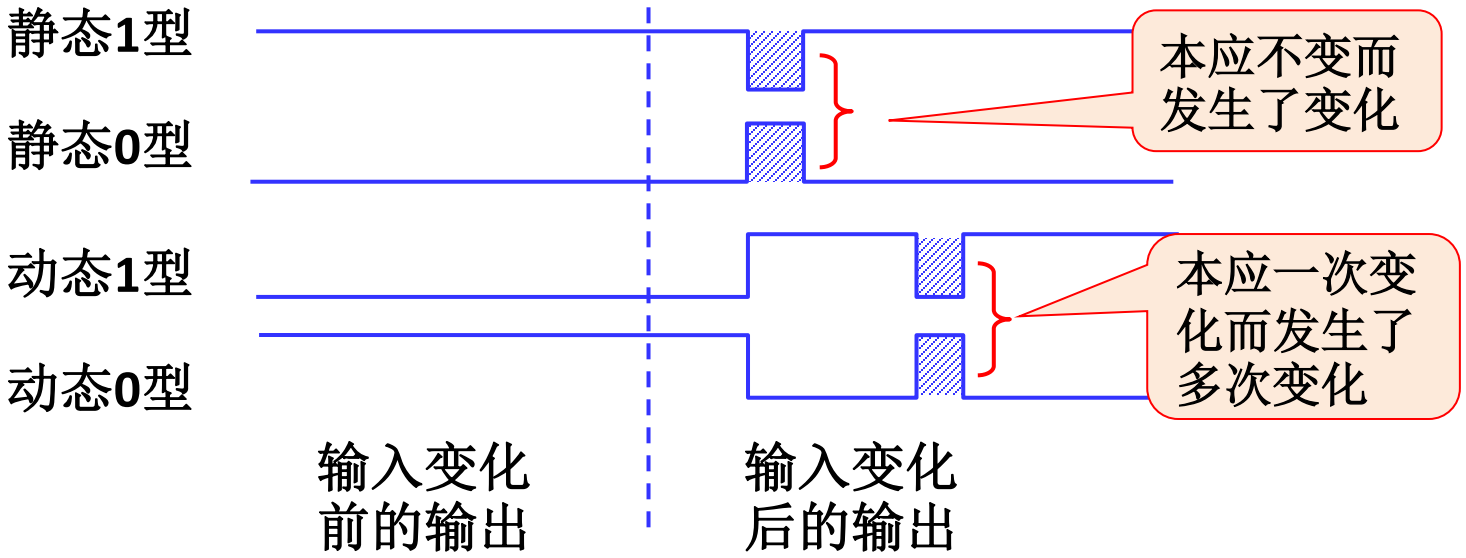
3.8 组合逻辑电路中的竞争与冒险

冒险的分类：

逻辑冒险～一个变量的变化。
功能冒险～多个变量的变化。

逻辑冒险
- 静态冒险～本应不变而发生了变化。
- 动态冒险～本应一次变化而发生了多次变化。

输出错误
- 偏0型冒险～产生高电平错误。
- 偏1型冒险～产生低电平错误。



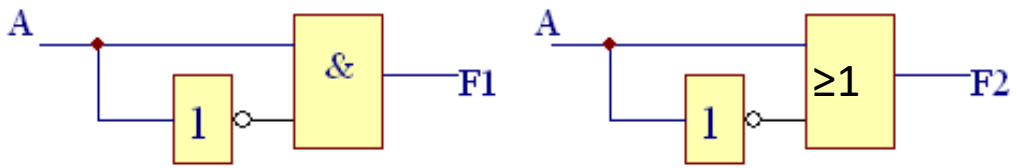
3.8 组合逻辑电路中的竞争与冒险

- 冒险产生的原因：

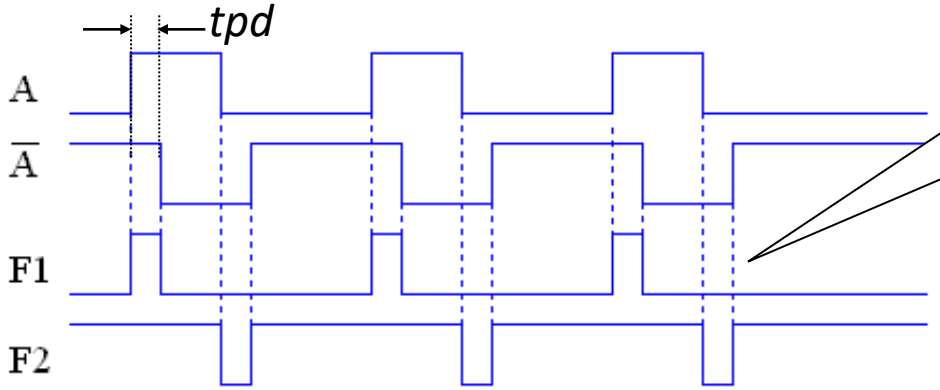
  信号路径不同

  器件延时不同

- 静态逻辑冒险举例



$F1=A\cdot\overline{A}=0$　　$F2=A+\overline{A}=1$



一般情况：

$A\cdot\overline{A}$ 产生偏0型冒险

$A+\overline{A}$ 产生偏1型冒险

3.8 组合逻辑电路中的竞争与冒险

一、冒险的判别

1. 代数法

判断任意一个逻辑电路的输出在其他变量取某个固定的逻辑值时，是否出现以下现象：

$F = A + \bar{A}$　　称为偏1型冒险

$F = A \cdot \bar{A}$　　称为偏0型冒险

2. 卡诺图法

如果卡诺图中有两个卡诺圈相切，且相切处未被其他卡诺圈包围，则存在冒险。

3.8 组合逻辑电路中的竞争与冒险

一、冒险的判别

例： 试判断电路$F=\overline{A}\,\overline{C}+\overline{A}B+AC$是否可能产生冒险。

解： 变量A和C具备竞争的条件, 应分别进行检查。

检查C:

$AB=00 \quad F=\overline{C}$

$AB=01 \quad F=1$

$AB=10 \quad F=C$

$AB=11 \quad F=C$

C的变化不会产生冒险

检查A:

$BC=00 \quad F=\overline{A}$

$BC=01 \quad F=A$

$BC=10 \quad F=\overline{A}$

$BC=11 \quad F=A+\overline{A}$

当B=C=1时, A的变化可能使电路产生冒险.

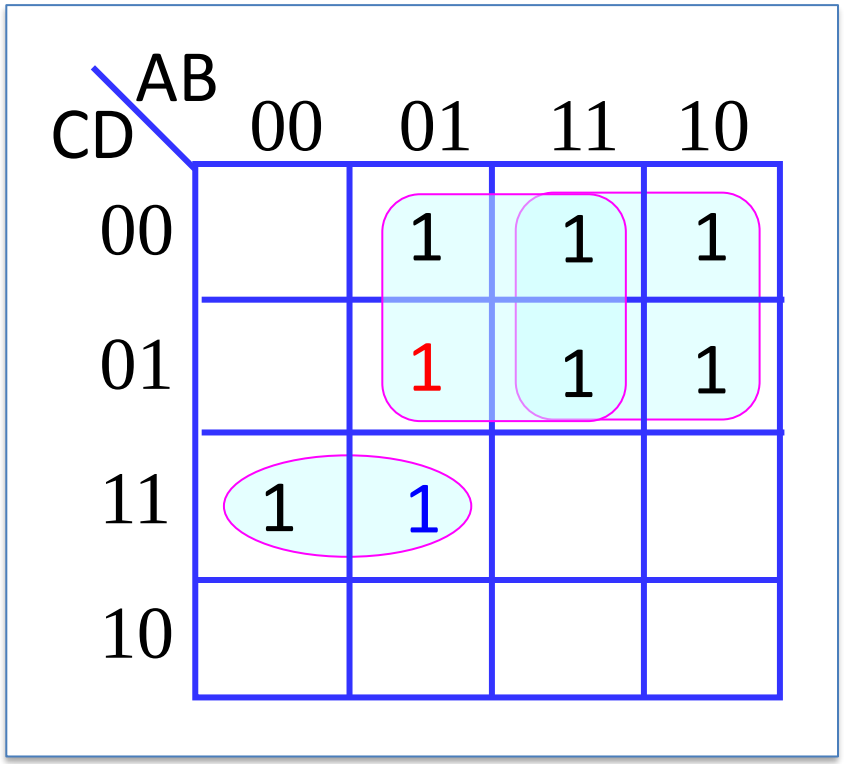
3.8 组合逻辑电路中的竞争与冒险

一、冒险的判别

当描述电路的逻辑函数为“**与或**”式时，可采用卡诺图来判断是否存在冒险。

其方法是观察是否存在“**相切**”的卡诺圈，若存在则可能产生冒险。

例：在电路$F = B\overline{C} + A\overline{C} + \overline{A}CD$的卡诺图中，相邻最小项$\overline{A}B\overline{C}D$与$\overline{A}BCD$不被同一卡诺圈所包含，因此当$A=0$，$B=D=1$时(此时$F=C+\overline{C}$)，电路可能由于$C$的变化而产生险象。



3.8 组合逻辑电路中的竞争与冒险

一、冒险的消除

1. 冗余项

在保证逻辑功能不变的前提下，只要在卡诺图中两卡诺圈相切处加一个卡诺圈使其相交，即增加一个冗余项，就可消除冒险。

2. 滤波法

在输出端并接一个很小的滤波电容或在本级输出端与下级输入端之间串接一个RC滤波电路来消除其影响。

3. 选通法

在电路上增加一个选通信号，当输入信号变化时，输出端与电路断开，当输入稳定后，选通信号工作，使输出端与电路接通。

3.8 组合逻辑电路中的竞争与冒险

一、冒险的消除

1. 增加冗余项消除冒险

在保证逻辑功能不变的前提下，只要在卡诺图中两卡诺圈相切处加一个卡诺圈使其相交，即增加一个冗余项，就可消除冒险。

$F = AB + \overline{A}C$

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

相切点，需增加卡诺圈

$F = AB + \overline{A}C + BC$

$F = A\overline{C} + \overline{A}BD + \overline{A}C\,\overline{D}$

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

相切点，需增加卡诺圈

$F = A\overline{C} + \overline{A}BD + B\overline{C}D + \overline{A}C\overline{D} + \overline{A}BC$

3.8 组合逻辑电路中的竞争与冒险

一、冒险的消除

1. 增加冗余项消除冒险

在表达式中"加"上多余的"与项"或"乘"上多余的"或项"，使原函数不可能在某种条件下再出现 $X+\overline{X}$ 或 $X\cdot\overline{X}$ 的形式，从而消除可能产生

(1) 利用定理：$AB+\overline{A}C=AB+\overline{A}C+BC$ 给原函数增加冗余项。

例：电路如图所示，用增加冗余项的方法消除电路中的险象。

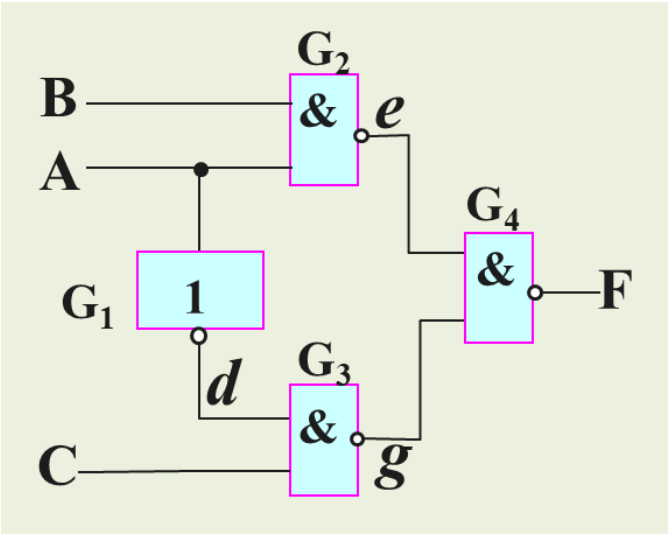
解：原电路对应的函数表达式为

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

根据定理，增加冗余项BC：

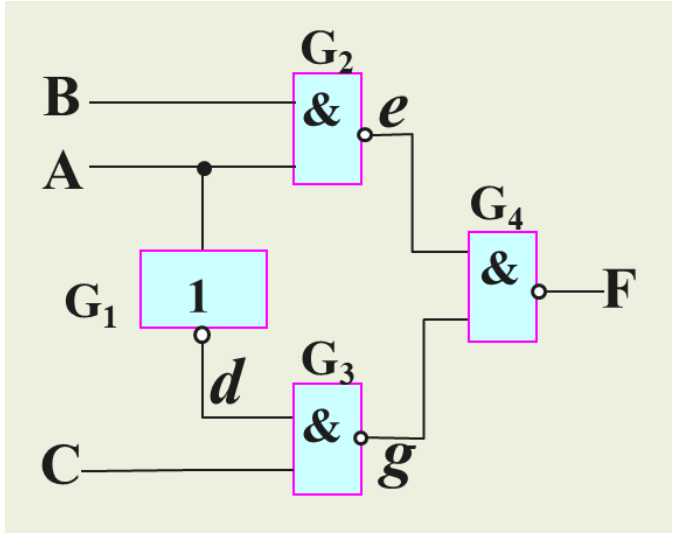
$$F=AB+\overline{A}C+BC$$

$$=\overline{\overline{AB}\cdot\overline{\overline{A}C}\cdot\overline{BC}}$$



3.8 组合逻辑电路中的竞争与冒险

一、冒险的消除



例：电路如图所示，用增加冗余项的方法消除电路中的险象。

解：原电路对应的函数表达式为

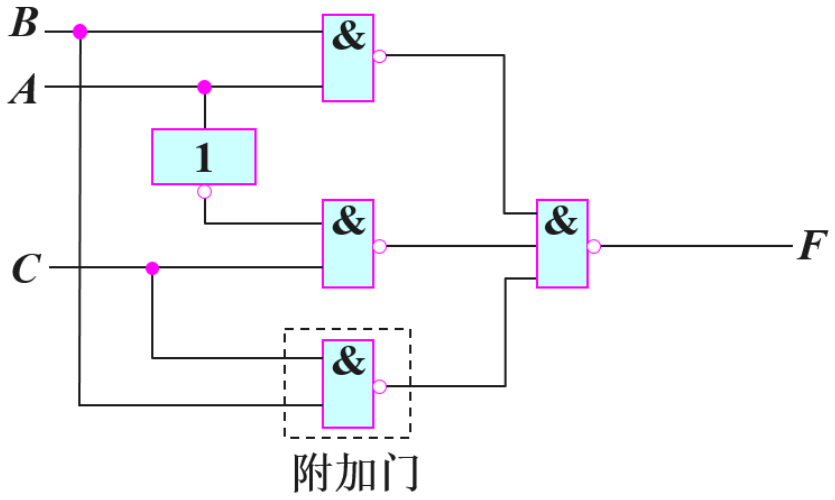
$$F=\overline{\overline{AB}\cdot\overline{\overline{A}C}}=AB+\overline{A}C$$

根据定理，增加冗余项BC：

$$F=AB+\overline{A}C+BC$$

$$=\overline{\overline{AB}\cdot\overline{\overline{A}C}\cdot\overline{BC}}$$

电路改进如右图所示：

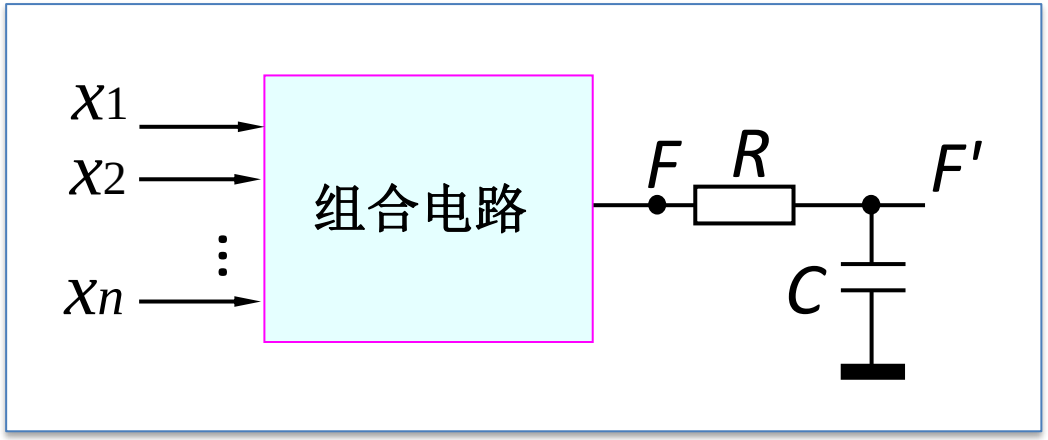


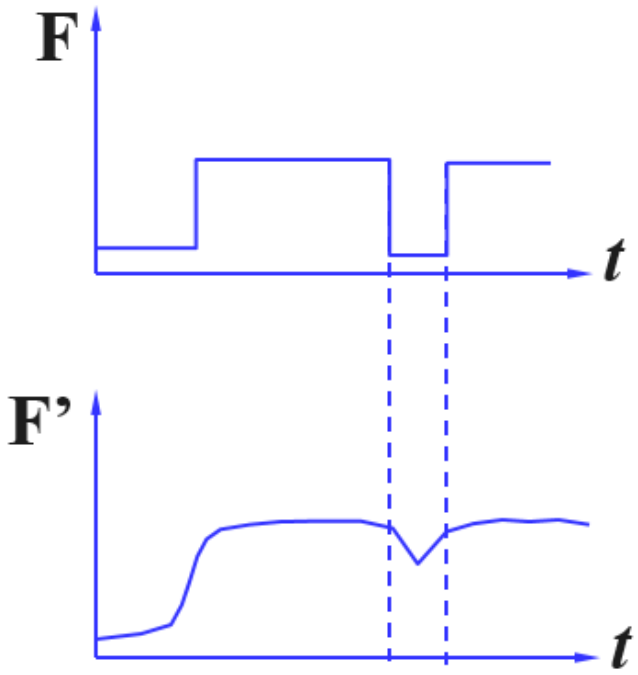
当$B=C=1$时，函数由$F=A+\overline{A}$变成了$F=1$

3.8 组合逻辑电路中的竞争与冒险

一、冒险的消除

2. <u>滤波法</u>：在输出端接一个很小的滤波电容或在本级输出端与下级输入端之间串接一个RC积分电路来消除其影响。





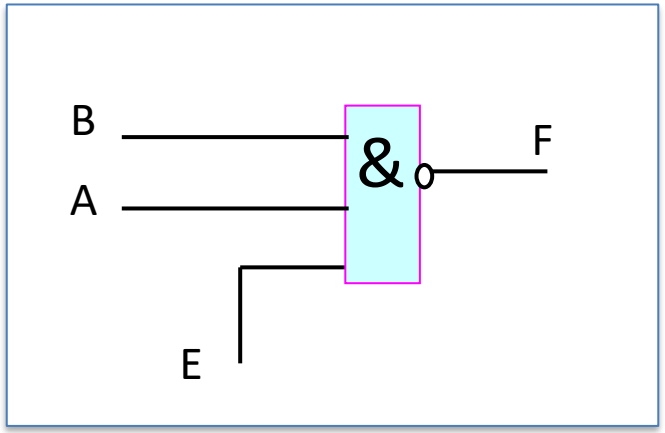
使用此方法时要适当选择时间常数($\tau=RC$)，要求τ足够大，以便“削平”尖脉冲；但又不能太大，以免使正常的输出发生畸变。

3.8 组合逻辑电路中的竞争与冒险

一、冒险的消除

3. 选通法

在电路上增加一个选通信号，当输入信号变化时，输出端与电路断开，当输入稳定后，选通信号工作，使输出端与电路接通。



(1) 先使E＝0，关闭与非门

(2) 等A、B信号都来到后，

让E＝1,得到可靠的$F=AB$