数字逻辑应急手册

牟正强

——组合逻辑电路的分析和设计方法

分析：

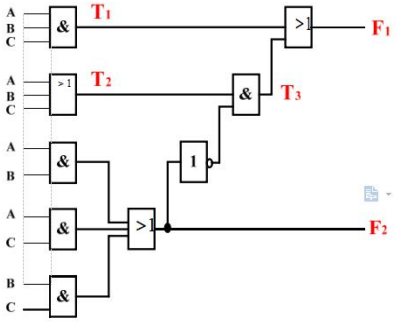
(1)分别用符号标注各级门的输出端。

(2)从输入端到输出端逐级写出输出变量对输入变量的逻辑表达式，最后得到输入变量表示的输出函数表达式。需要时用卡诺图或公式化简法化简逻辑函数成最简形式。

(3)列真值表。

(4)根据真值表或函数表达式确定电路的逻辑功能。有时功能难以用简练的语言描述，此时列真值表即可。

例分析图示电路的逻辑功能。



真值表

输入			输出	
A	B	C	F_2	F_1
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

设计：

① 分析事件的因果关系，确定输入和输出变量；

② 定义逻辑状态的含意；

③ 根据因果关系列出真值表；

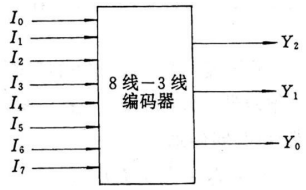
最简：器件最少，器件种类最少， 而且器件之间的连线也最少。

——例题：试用与非门设计一个将 8421-BCD 码转换为余 3 码的码制转换电路。

——编码器(Encoder)

一、普通编码器(Common Encoder)： 任何时刻只允许输入一个编码信号，否则将发生混乱。

3 位二进制普通编码器示例：



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

$$\begin{cases} Y_2 = I_4 + I_5 + I_6 + I_7 \\ Y_1 = I_2 + I_3 + I_6 + I_7 \\ Y_0 = I_1 + I_3 + I_5 + I_7 \end{cases}$$

或门实现

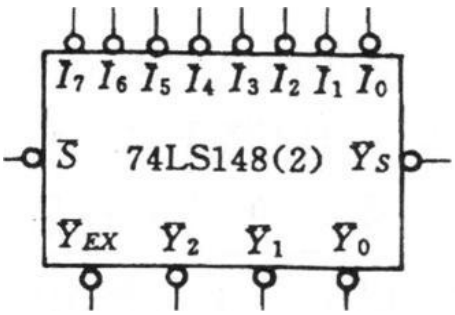
$$\begin{cases} Y_2 = \overline{\overline{I_4}\cdot\overline{I_5}\cdot\overline{I_6}\cdot\overline{I_7}} \\ Y_1 = \overline{\overline{I_2}\cdot\overline{I_3}\cdot\overline{I_6}\cdot\overline{I_7}} \\ Y_0 = \overline{I_1}\cdot\overline{I_3}\cdot\overline{I_5}\cdot\overline{I_7} \end{cases}$$

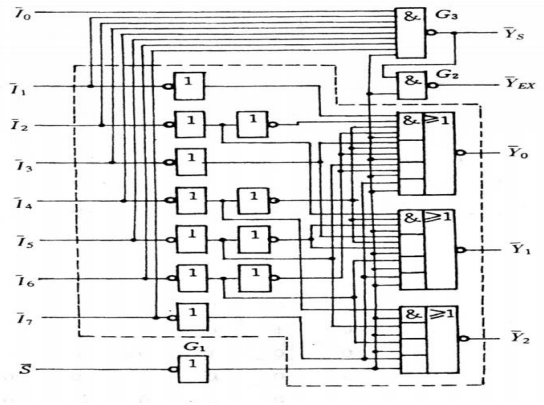
与非门实现

思考 1：如何用与非门实现 8421-BCD 码普通编码器？

二、优先编码器(Priority Encoder)： 允许同时输入两个以上编码信号。不过在设计 优先编码器时已经将所有的输入信号按优先顺序排了队，当几个输入信号同时出现时， 只对其中优先权最高的一个进行编码。

8 线-3 线优先编码器 74LS148





输入									输出				
$\overline{S}$	$\overline{I}_0$	$\overline{I}_1$	$\overline{I}_2$	$\overline{I}_3$	$\overline{I}_4$	$\overline{I}_5$	$\overline{I}_6$	$\overline{I}_7$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$	$\overline{Y}_S$	$\overline{Y_{EX}}$
1	×	×	×	×	×	×	×	×	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	0	1
0	×	×	×	×	×	×	×	0	0	0	0	1	0
0	×	×	×	×	×	×	0	1	0	0	1	1	0
0	×	×	×	×	×	0	1	1	0	1	0	1	0
0	×	×	×	×	0	1	1	1	0	1	1	1	0
0	×	×	×	0	1	1	1	1	1	0	0	1	0
0	×	×	0	1	1	1	1	1	1	0	1	1	0
0	×	0	1	1	1	1	1	1	1	1	0	1	0
0	0	1	1	1	1	1	1	1	1	1	1	1	0

$$\begin{cases}\overline{Y_2}=\overline{(I_4+I_5+I_6+I_7)\cdot S}\\ \overline{Y_1}=\overline{(I_2\overline{I_4}\,\overline{I_5}+I_3\overline{I_4}\,\overline{I_5}+I_6+I_7)\cdot S}\\ \overline{Y_0}=\overline{(I_1I_2\overline{I_4}\,\overline{I_6}+I_3\overline{I_4}\,\overline{I_6}+I_5\overline{I_6}+I_7)\cdot S}\end{cases}$$

时，$\overline{Y}_S$ 才为低电平。$\overline{Y}_S$ 为低电平表示“电路工作，但无编码输入”。

$\overline{Y}_S$ 为选通输出端，其表达式为：

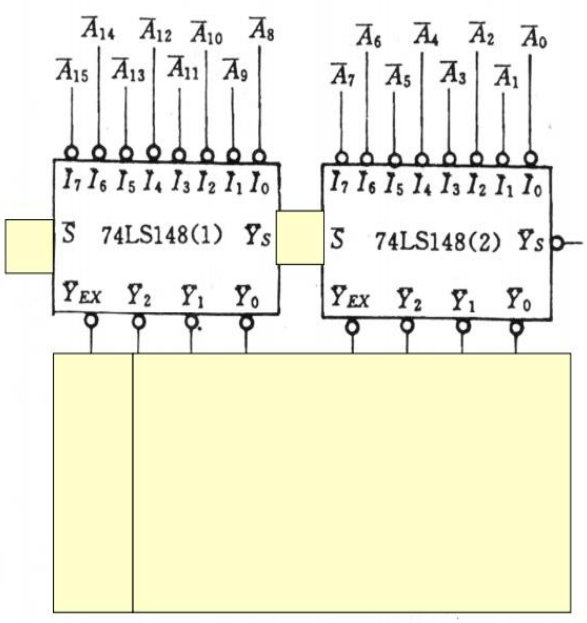
$$\overline{Y_S}=\overline{\overline{I_0}\,\overline{I_1}\,\overline{I_2}\,\overline{I_3}\,\overline{I_4}\,\overline{I_5}\,\overline{I_6}\,\overline{I_7}\cdot S}$$

(3) $\overline{Y_{EX}}$为扩展端，用于扩展编码功能，其表达式为：

$$\overline{Y_{EX}}=\overline{(I_0+I_1+I_2+I_3+I_4+I_5+I_6+I_7)\cdot S}$$

$\overline{S}=0$），Y_{EX} 即为低电平。所以，Y_{EX} 低电平输出信号表示“电路工作，且有编码输入”。

例、试用两片 74LS148 接成 16 线-4 线优先编码器，将 16 个低电平输入信号 $A_0 \sim A_{15}$ 编为‘0000—1111’ 16 个4位二进制代码，其中 A_{15} 的优先权最高，A_0 的优先 权最低。

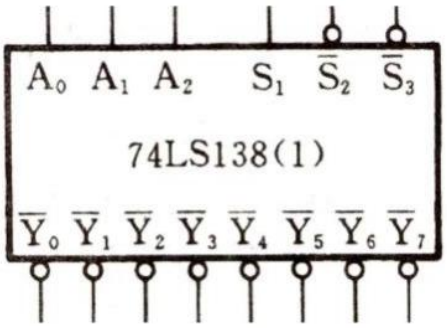


思考 2：如何用一片 74LS148 实现 8421-BCD 码优先编码器？

——译码器(Decoder)： 每个输入的二进制代码对应的输出为高、 低电平信号。

一、二进制译码器(最小项译码器)

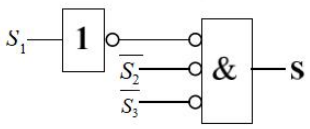
三极管集成门译码器电路



输入					输出							
S_1	$\overline{S}_2+\overline{S}_3$	A_2	A_1	A_0	$\overline{Y}_0$	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	$\overline{Y}_4$	$\overline{Y}_5$	$\overline{Y}_6$	$\overline{Y}_7$
0	×	×	×	×	1	1	1	1	1	1	1	1
×	1	×	×	×	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

$$\begin{cases}\overline{Y}_0=\overline{\overline{A}_2\overline{A}_1\overline{A}_0}=\overline{m_0}\\ \overline{Y}_1=\overline{\overline{A}_2\overline{A}_1A_0}=\overline{m_1}\\ \overline{Y}_2=\overline{\overline{A}_2A_1\overline{A}_0}=\overline{m_2}\\ \overline{Y}_3=\overline{\overline{A}_2A_1A_0}=\overline{m_3}\end{cases}$$

$$\begin{cases}\overline{Y}_4=\overline{A_2\overline{A}_1\overline{A}_0}=\overline{m_4}\\ \overline{Y}_5=\overline{A_2\overline{A}_1A_0}=\overline{m_5}\\ \overline{Y}_6=\overline{A_2A_1\overline{A}_0}=\overline{m_6}\\ \overline{Y}_7=\overline{A_2A_1A_0}=\overline{m_7}\end{cases}$$



二、二—十进制译码器

将输入的 BCD 码的 10 个代码译成 10 个高、低电平输出信号。它属于码制变换译码器中的一种。

4 线－10 线译码器 74LS42 是二－十进制译码器的 一个典型例子，它将所输入的 8421－BCD 码二进制代 码译成十进制代码 0~9。

74LS42 功能表

序号	输入				输出									
	A_3	A_2	A_1	A_0	$\overline{Y}_0$	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	$\overline{Y}_4$	$\overline{Y}_5$	$\overline{Y}_6$	$\overline{Y}_7$	$\overline{Y}_8$	$\overline{Y}_9$
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	1	1	1	1	1
2	0	0	1	0	1	1	0	1	1	1	1	1	1	1
3	0	0	1	1	1	1	1	0	1	1	1	1	1	1
4	0	1	0	0	1	1	1	1	0	1	1	1	1	1
5	0	1	0	1	1	1	1	1	1	0	1	1	1	1
6	0	1	1	0	1	1	1	1	1	1	0	1	1	1
7	0	1	1	1	1	1	1	1	1	1	1	0	1	1
8	1	0	0	0	1	1	1	1	1	1	1	1	0	1
9	1	0	0	1	1	1	1	1	1	1	1	1	1	0
伪码	1	0	1	0	1	1	1	1	1	1	1	1	1	1
	1	0	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	0	0	1	1	1	1	1	1	1	1	1	1
	1	1	0	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	0	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1

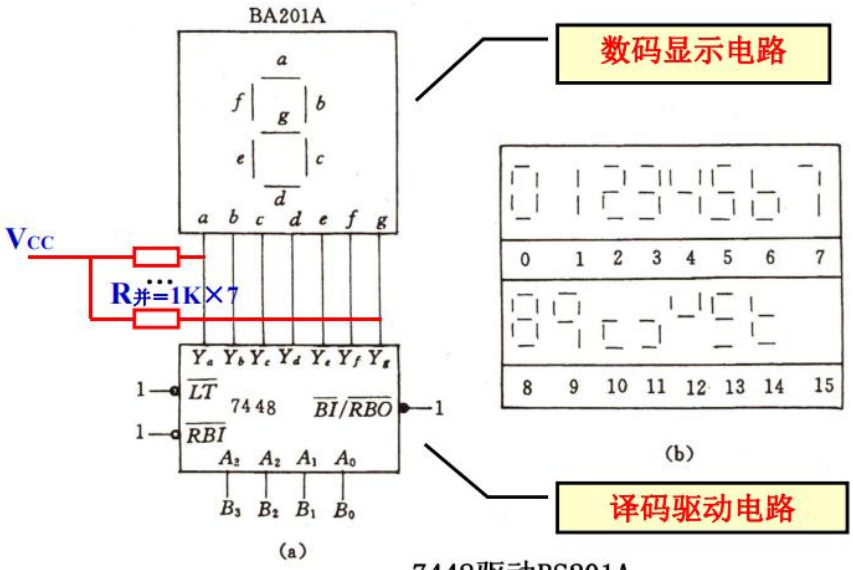
$$\begin{cases} \overline{Y_0} = \overline{\overline{A_3}\,\overline{A_2}\,\overline{A_1}\,\overline{A_0}} \\ \overline{Y_1} = \overline{\overline{A_3}\,\overline{A_2}\,\overline{A_1}A_0} \\ \overline{Y_2} = \overline{\overline{A_3}\,\overline{A_2}A_1\overline{A_0}} \\ \overline{Y_3} = \overline{\overline{A_3}\,\overline{A_2}A_1A_0} \\ \overline{Y_4} = \overline{\overline{A_3}A_2\overline{A_1}\,\overline{A_0}} \end{cases} \qquad \begin{cases} \overline{Y_5} = \overline{\overline{A_3}A_2\overline{A_1}A_0} \\ \overline{Y_6} = \overline{\overline{A_3}A_2A_1\overline{A_0}} \\ \overline{Y_7} = \overline{\overline{A_3}A_2A_1A_0} \\ \overline{Y_8} = \overline{A_3\overline{A_2}\,\overline{A_1}\,\overline{A_0}} \\ \overline{Y_9} = \overline{A_3\overline{A_2}\,\overline{A_1}A_0} \end{cases}$$

思考：如何实现 5421、2421、余 3 码等 BCD 码的译码转换？

三、显示译码器

将数字(0~9)、文字、符号(A~F)等的二进制代码翻译并显示出来的电路叫显示译码器。它包括译码 驱动电路和数码显示器两部分。

例：BCD七段字符译码显示电路



7448驱动BS201A

（a）接线图 （b）七段显示字型

- 74LS48 除了有实现 7 段显示译码器基本功能的输入(DCBA)和输出(Ya~Yg)端外，还引入了灯测试输入端(LT)和动态灭零输入端(RBI)，以及既有 输入功能又有输出功能的消隐输入/动态灭零输出(BI/RBO)端
- 逻辑功能：
- 1 ）7 段译码功能 (LT=1 ，RBI=1)在灯测试输入端(LT)和动态灭零输入端(RBI)都接无效电平时，输入 DCBA 经 7448 译码，输出高电平有效的
- 7 段字符显示器的驱动信号，显示相应字符。除 DCBA= 0000 外， RBI 也可以接 低电平，见表中 1~16 行。
- 2)消隐功能 (BI=0)此时 BI/RBO 端作为输入端，该端输入低电平信号时，表倒数第 3 行，无论 LT 和 RBI 输入什么电平信号，不管输入 DCBA 是什么状态，输出全为“0”， 7 段显示器熄灭。该功能主要用于多显示器的动态显示。
- 3)灯测试功能 (LT = 0)此时 BI/RBO 端作为输出端，端输入低电平信号时，表最后一行，与及 DCBA 输入无关，输出全为“1”，显示器 7 个字段都点亮。 该功能用于 7 段显示器测试，判别是否有损坏的字段。
- 4)动态灭零功能 (LT=1 ，RBI=1)此时 BI/RBO 端也作为输出端， LT 端输入高 电平信号， RBI 端输入低电平信号，若此时 DCBA=0000，表 1 倒数第 2 行，输 出全为“0”，显示器熄灭，不显示这个零。 DCBA≠0，则对显示无影响。该功能主要用于多个 7 段显示器同时显示时熄灭高位的零。

74ls48功能表—七段译码驱动器功能表													
十进数或功能	输入			BI/RBO	输出							备注
	LT	RBI	D C B A		a	b	c	d	e	f	g	
0	H	H	0 0 0 0	H	1	1	1	1	1	1	0	1
1	H	x	0 0 0 1	H	0	1	1	0	0	0	0	
2	H	x	0 0 1 0	H	1	1	0	1	1	0	1	
3	H	x	0 0 1 1	H	1	1	1	1	0	0	1	
4	H	x	0 1 0 0	H	0	1	1	0	0	1	1	
5	H	x	0 1 0 1	H	1	0	1	1	0	1	1	
6	H	x	0 1 1 0	H	0	0	1	1	1	1	1	
7	H	x	0 1 1 1	H	1	1	1	0	0	0	0	
8	H	x	1 0 0 0	H	1	1	1	1	1	1	1	
9	H	x	1 0 0 1	H	1	1	1	0	0	1	1	
10	H	x	1 0 1 0	H	0	0	0	1	1	0	1	
11	H	x	1 0 1 1	H	0	0	1	1	0	0	1	
12	H	x	1 1 0 0	H	0	1	0	0	0	1	1	
13	H	x	1 1 0 1	H	1	0	0	1	0	1	1	
14	H	x	1 1 1 0	H	0	0	0	1	1	1	1	
15	H	x	1 1 1 1	H	0	0	0	0	0	0	0	
BI	x	x	x x x x	L	0	0	0	0	0	0	0	2
RBI	H	L	0 0 0 0	L	0	0	0	0	0	0	0	3
LT	L	x	x x x x	H	1	1	1	1	1	1	1	4

四、译码器的应用

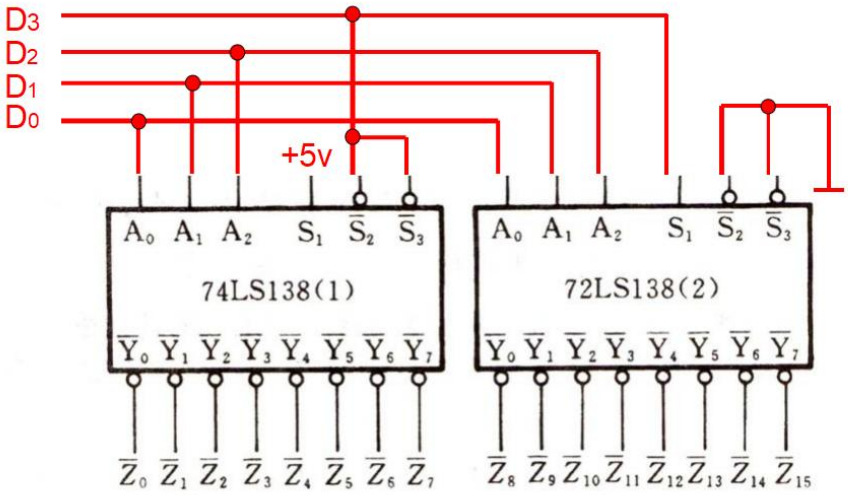
(1)在存储器中的应用

用作地址译码器或指令译码器，译码器输入地 址码，输出为存储单元地址。如n 位地址线可寻址 2^n 个单元。

(2)扩展应用

在需进行大容量译码时，可将芯片进行扩展。

例、 试用两片74LS138组成4线－16线译码器，将输入的4位二进制代码 $D_3D_2D_1D_0$ 译成16个独立的低电平信号 $Z_0 \sim Z_{15}$ 。

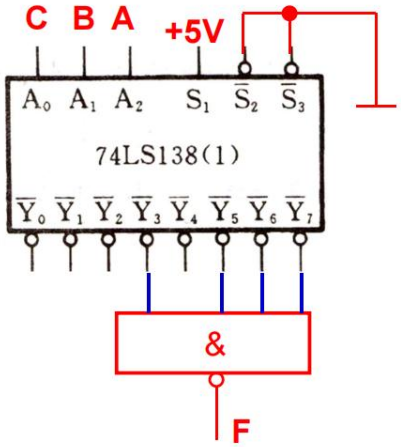


用两片74LS138接成的4线－16线译码器

思考：如何用 74LS138 实现 5 线－32 线译码器？

（3）实现逻辑函数

例、用74LS138实现函数F(A,B,C)=AB+AC+BC



注：实现多变量译码输入的逻辑函数时，可以先扩展再按上述方法实现。

例、试用74LS138设计一个多输出的组合逻辑电路。

输出的逻辑函数为

$$\begin{cases} Z_1 = A\overline{C} + \overline{A}BC + A\overline{B}C \\ Z_2 = BC + A\overline{B}C \\ Z_3 = \overline{A}B + A\overline{B}C \\ Z_4 = \overline{A}B\overline{C} + \overline{B}C + ABC \end{cases}$$

思考：如何用74LS138实现组合逻辑函数

$$F(A,B,C,D) = ABC + A\overline{B}\,\overline{D} + \overline{B}\,\overline{C}D$$

(4) 有些二进制译码器还可作数据分配器使用。

——数据分配器（ Demultiplexer）

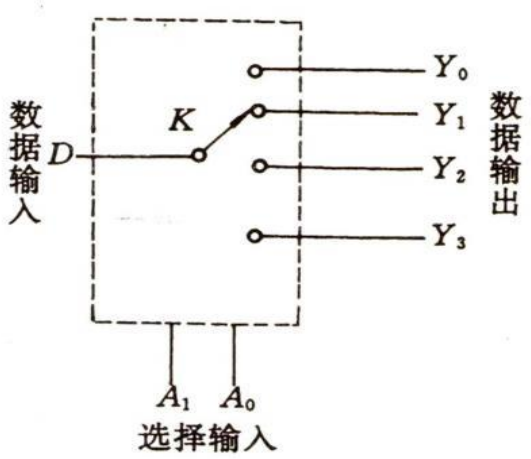
数据传输过程中，有时需要将数据分配到不同的数据通道上，能够完成这种功能的电路称为数据分配器，亦称多路分配器、多路调节器，简称 DEMUX，其电路为单输入、多输出形式。

1 、DEMUX 的应用

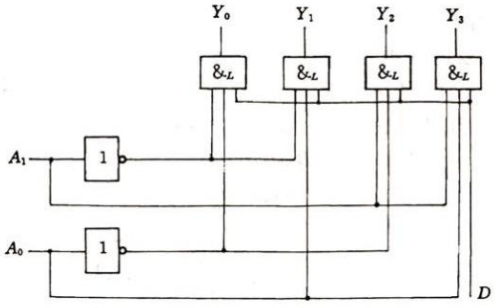
基本用途：有选择的将一个数据送到多路输出中的一路。

2、数据分配器的逻辑功能

DEMUX 的功能如同多位开关一样，将输入 D 送到选择输入指定的通道上（如图所示）。



图所示为一个四路数据分配器的逻辑图， D 为 被传输的数据， AO ，A1 是选择输入端， YO ~ Y3 为数据输出端。



数据分配器示意图

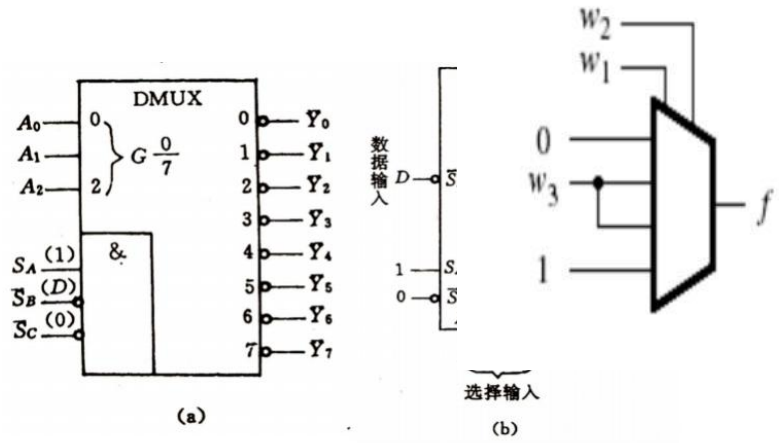
数据分配真值表

A_1	A_0	Y_0	Y_1	Y_2	Y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

3 、1 路－8 路 DEMUX74138 （应用 (4) ）

74138 不仅可以作 3 线－8 线译码器，而且还可用

作 1 路－8 路数据分配器(如图所示)



74138用作1路－8路数据分配器的逻辑符号

(a)国际逻辑符号　　(b)惯用逻辑符号

——数据选择器(Multiplexer)

MUX 的功能正好与 DEMUX 相反，为多输入、 单输出形式。

目前，常用的 MUX 有二选一、四选一、八选 一和十六选一等多种类型。

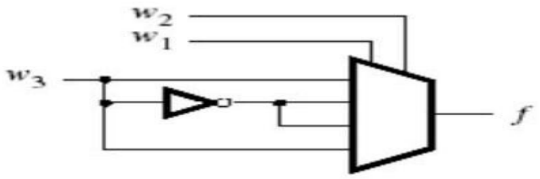
利用 4 选 1 多路器来实现 3 输入表决器

w_1	w_2	w_3	f
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

w_1	w_2	f
0	0	0
0	1	w_3
1	0	w_3
1	1	1

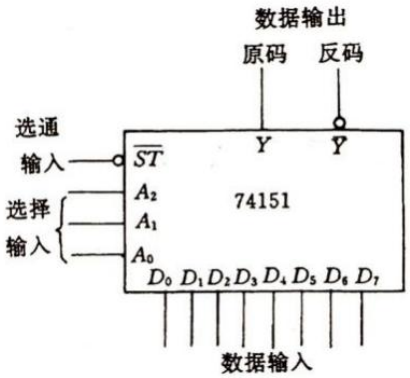
用 4 选 1 多路器来实现 3 位输入的异或逻辑

w_1	w_2	w_3	f	
0	0	0	0	w_3
0	0	1	1	
0	1	0	1	$\overline{w}_3$
0	1	1	0	
1	0	0	1	$\overline{w}_3$
1	0	1	0	
1	1	0	0	w_3
1	1	1	1	



2、八选一数据选择器 74151

八选一 MUX 需要 3 个选择输入端， 8 个数据输入端， 并有互补的原码和反码两种输出形式。



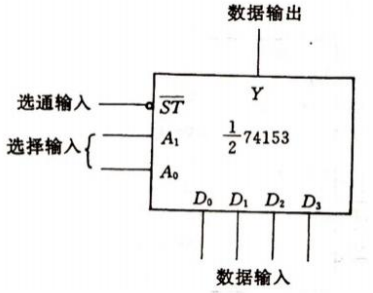
74151惯用逻辑符号

74151真值表

输入				输出	
$\overline{ST}$	A_2	A_1	A_0	Y	$\overline{Y}$
1	×	×	×	0	1
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	$\overline{D}_7$

3、双四选一数据选择器 74153

74153 包含两个完全相同的 4 选一 MUX，两个 MUX 有公共的地址输入端，而数据输入和输出端各自独立。通过给定不同的地址代码(A1A0),即可从 4 个输入数据中选出所需要的一个，并送至输出端 Y。

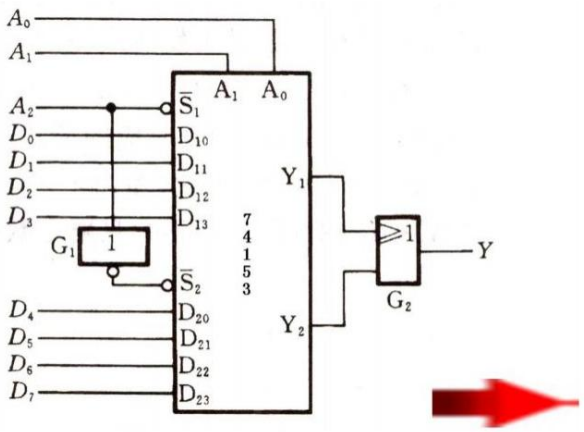


74153惯用逻辑符号

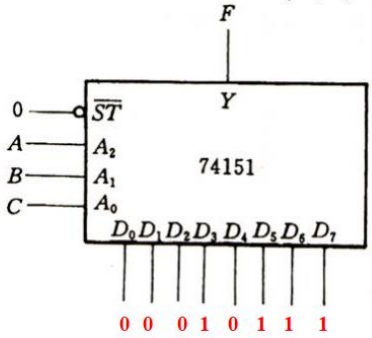
74153真值表

输入			输出
$\overline{ST}$	A_1	A_0	Y
1	×	×	0
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3

例： **试用双四选一MUX74LS153组成一个8选一MUX。**



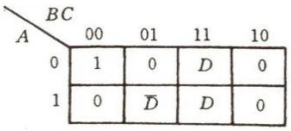
例、 试用74151实现逻辑函数F(A,B,C)=AB+AC+BC



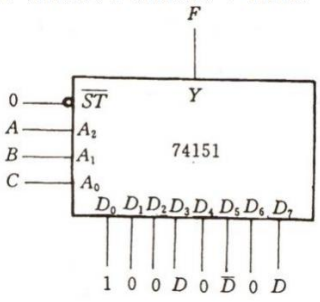
用74151实现逻辑函数

例、 试用一片74151实现逻辑函数

$$F(A,B,C,D) = ABCD + \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}BC\overline{D} + \overline{A}\overline{B}\overline{C}$$



(a) 引入变量的卡诺图



(b) 逻辑图

用74151实现逻辑函数

思考： 如何用一片74151实现逻辑函数

$$F(A,B,C,D,E) = \overline{A}\overline{B}C\overline{D}\overline{E} + A\overline{B}CD\overline{E} + \overline{A}\overline{B}CDE + \overline{A}BCDE + \overline{A}BCDE$$

一、一位数值比较器

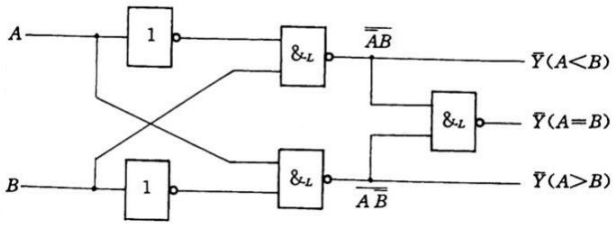
两个1位二进制数A,B相比的情况有以下几种：

①A>B（即A=1,B=0），则 $\overline{AB}=1$，所以可用 $\overline{AB}$ 作为A>B的输出信号 $Y_{(A>B)}$ 。

②同理可用 AB 作为A<B的输出信号 $Y_{(A<B)}$ 。

③同理可用A⊙B 作为A=B的输出信号 $Y_{(A=B)}$ 。

于是， 1位数值比较器的电路图可如下设计：

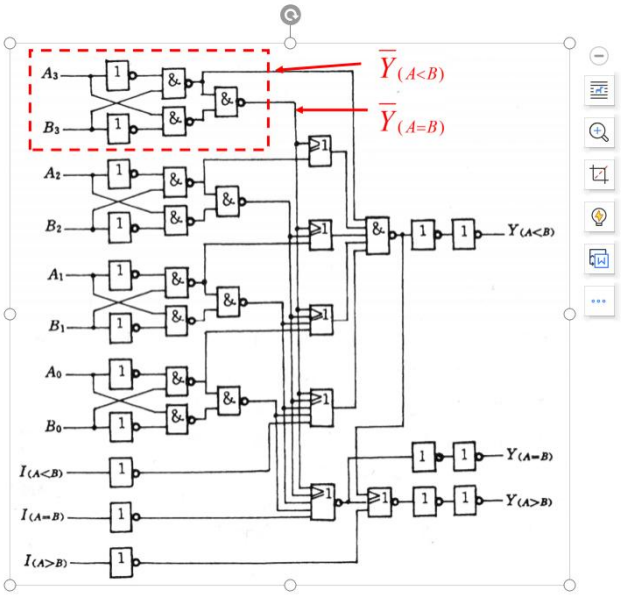


1位数值比较器逻辑图

二、多位数值比较器

在比较两个多位数的大小时，必须自高而低的逐位比较，而且只有在高位相等时，才需比较较低位。

下图示出了4位比较器CC14585的逻辑图。



——加法器(Adder)

一、1 位加法器

1、半加器(Half Adder)

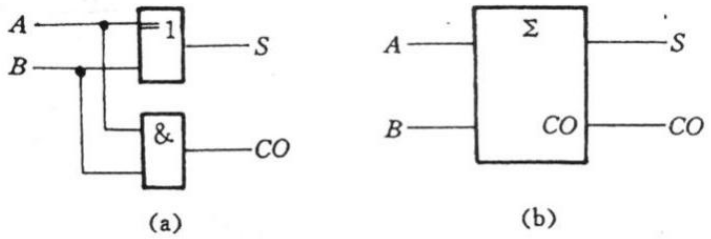
若不考虑有来自低位的进位将两个 1 位二进制数相加， 称为半加。实现半加运算的电路叫做半加器。半加器的真值表、逻辑表达式、电路图和惯用符号如下所示：

半加器的真值表

输入		输出	
A	B	S	CO
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

半加器的逻辑表达式

$$\begin{cases} S = A\overline{B} + \overline{A}B = A \text{由} B \\ CO = AB \end{cases}$$



半加器的电路图和惯用逻辑符号

2、全加器(Full Adder)

全加器的真值表

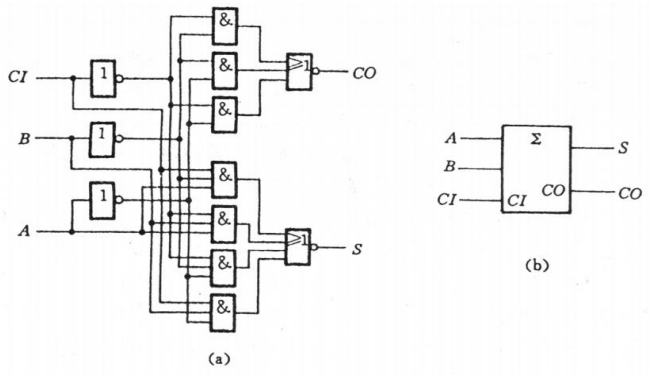
输入			输出	
CI	A	B	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

全加器的逻辑表达式

$$\begin{cases} S = \overline{\overline{A\overline{B}} \cdot \overline{CI} + A\overline{B} \cdot CI + \overline{A}B \cdot CI + AB \cdot \overline{CI}} \\ CO = \overline{\overline{A\overline{B}} + \overline{B \cdot CI} + \overline{A} \cdot \overline{CI}} \end{cases}$$

或

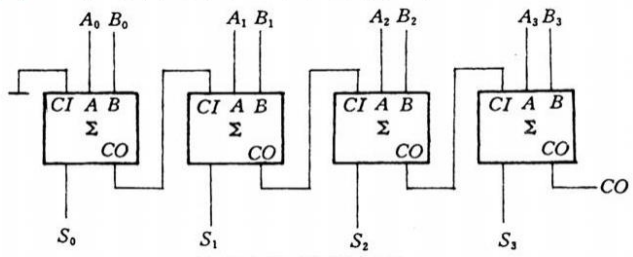
$$\begin{cases} S = A \oplus B \oplus CI \\ CO = AB + CI(A + B) \end{cases}$$



二、多位加法器

1、串行进位加法器

原理：依次将低位全加器的进位输出端 CO 接到高位全加器的进位输入端 CI 即可构成多位串行加法器。

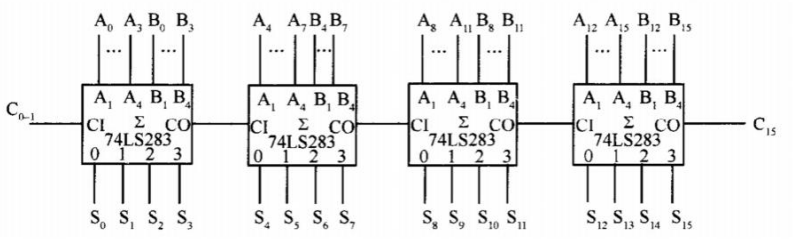


4位串行加法器电路

用举例：多人表决电路。

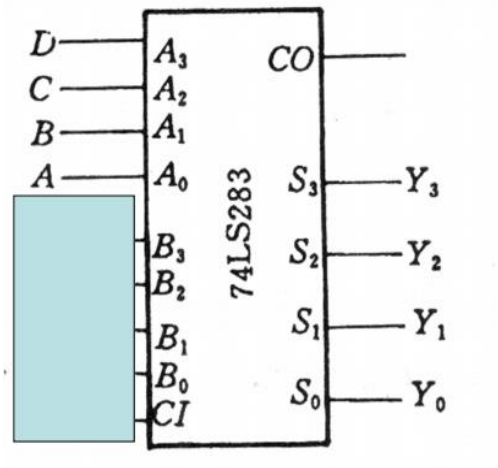
串行进位加法器的优点：电路结构比较简单； 缺点：运算速度慢。

例： 4片74283级联成16位二进制加法电路的电路为：



三、用加法器设计组合逻辑电路

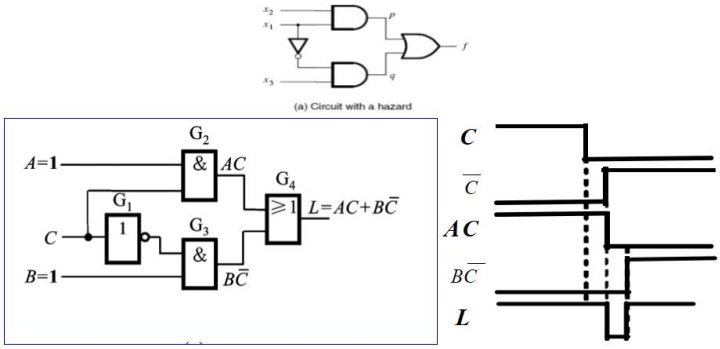
对“变量＋变量”或“变量＋常量”类型的逻辑函 数用加法器设计起来非常简单。



——产生的竞争冒险的原因

不考虑门的延时时间

考虑门的延时时间，当A=0　B=1



竞争：当一个逻辑门的两个输入端的信号同时向相反方向变化，而变化的时间有差异的现象。
冒险:两个输入端的信号取值的变化方向是相反时，如门电路输出端的逻辑表达式简化成两个互补信号相乘或者相加，由竞争而可能产生输出干扰脉冲的现象。

成因：当两个输入信号同时向相反的逻辑电平跳变时(一个从 1 变为 0，一个从 0 变为 1)，由于存在时刻上的差异，使两个信号在 t 的极短时间内同时为高电平或低电平，从而产生尖峰脉冲，不符合门电路稳态下的 逻辑功能，产生内部噪声。
竞争：门电路两个输入信号同时向相反的逻辑电平 跳变(一个从 1 变为 0，一个从 0 变为 1)的现象叫竞争。有竞争不一定产生尖峰脉冲。由于竞争而在电路输出端可能产生尖峰脉冲的现象 叫做竞争－冒险。

二、检查竞争－冒险现象的方法

1、可通过逻辑函数式判断组合逻辑电路中是否有竞 争－冒险存在。 只要输出端的逻辑函数在一定条件下能化 简成 Y = A + A 或 Y = A . A 的形式，则可判定存在竞争 －冒险(此方法适用于任何瞬间只可能有一个输入变量改 变状态的情况)。
2 、用计算机辅助分析，运行数字电路的模拟程序。
3 、用实验检查。

三、消除竞争－冒险现象的方法

(一)接入滤波电容

尖峰脉冲一般都很窄(几十 ns 以内)，只要在输出 端并接一个很小的滤波电容 Cf (TTL 电路中通常为几 十~几百皮法)，就足以将尖峰脉冲的幅度削弱至门电 路的阈值电压以下。
优点：简单易行。
缺点：增加了输出电压波形的上升和下降时间，使波形变坏。

(二)引入选通脉冲

优点：简单，不需增加电路元件。
缺点：正常的输出信号也将变成脉冲信号，宽度与选通脉冲相同，且此选通脉冲必须与输入信号同步。

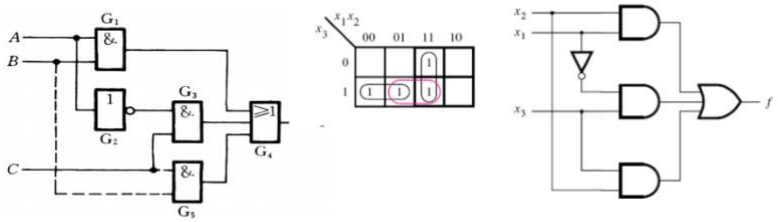
(三)修改逻辑设计

有时可用增加冗余项的方法消除竞争－冒险现象。
例：将 Y = AB + AC 化成 Y = AB + AC+ BC 可使电路功能不变，而消去 B=C=1 时的竞争－冒险现象。

$$f = x_1x_2 + \overline{x}_1x_3 \qquad f = x_1x_2 + \overline{x}_1x_3 + x_2x_3$$

优点：运用得当可收到令人满意的结果。
缺点：有利条件并不是任何时候都存在，其适用范围是有限的。

例：将 $Y = AB + \overline{A}C$ 化成 $Y = AB + \overline{A}C + BC$ ，可使电路功能不变，而消去B=C=1时的竞争－冒险现象。



修改逻辑设计消除竞争－冒险现象的示意图

Verilog组合逻辑电路与 时序逻辑电路

8－3编码器是将2的n次方个分离的信息以n个二进制代码来表示。

```
module bianma8_3(i, y);
input[7:0] i;
output[2:0] y;
reg[2:0] y;
always @ (i)
    begin
    case(i[7:0])
    8’ b00000001: y[2:0] = 3’ b000;
    8’ b00000010: y[2:0] = 3’ b001;
    8’ b00000100: y[2:0] = 3’ b010;

    8’ b00001000: y[2:0] = 3’ b011;
    8’ b00010000: y[2:0] = 3’ b100;
    8’ b00100000: y[2:0] = 3’ b101;

    8’ b01000000: y[2:0] = 3’ b110;
    8’ b10000000: y[2:0] = 3’ b111;
    default: y[2:0] = 3’ b000;
    endcase
    end
endmodule
```

3 － 8译码器是将n个二进制选择线,最多译码成2的n次方个分离的信息以来表示。

```
module decoder3_8
    (y, a, g1, g2, g3);
 output[7:0] y;
 input[2:0] a;
 input g1, g2, g3;
 reg[7:0] y;
always @ (a, y, g1, g2, g3)
 begin
  if(g1 == 0) y = 8’b1111_1111;
  else if(g2 == 1) y = 8’b1111_1111;
  else if(g3 == 1) y = 8’b1111_1111;
  else
    case(a[2:0])
      3’b000: y[7:0] = 8’b1111_1110;
      3’b001: y[7:0] = 8’b1111_1101;
      3’b010: y[7:0] = 8’b1111_1011;
      3’b011: y[7:0] = 8’b1111_0111;
      3’b100: y[7:0] = 8’b1110_1111;
      3’b101: y[7:0] = 8’b1101_1111;
      3’b110: y[7:0] = 8’b1011_1111;
      3’b111: y[7:0] = 8’b0111_1111;
      default: y[7:0] = 8’b1111_1111;
    endcase
 end
endmodule
```

n

```
module decoder3_8(y, a, g1, g2, g3);
 output[2:0] y;
 input[2:0] a;
 input g1, g2, g3;
 reg[2:0] y;
always @ (a, g1, g2, g3)
 begin
  if(g1 ==0) y = 8’b1111_1111;
  else if(g2 ==1) y = 8’b1111_1111;
  else if(g3 ==1) y = 8’b1111_1111;
  else
    begin
        y = 8’b0000_0001<<a;
        y = ~y;
    end
end
endmodule
```

四选一数据选择器：对四个数据源进行选择使用两位地址码A1A0产生地址信号来选择输出。

case语句实现

```
module mux41(y, g, d0, d1, d2, d3, a);
  output y;
  input[1:0] a;
  input g;
  input d0, d1, d2, d3;
  reg y;
always @ (d0, d1, d2, d3, a, g)
 begin
   if(g ==0) y = 0;
    else begin
         case(a[1:0])
           2'b00: y = d0;
           2'b01: y = d1;
           2'b10: y = d2;
           2'b11: y = d3;
    end
end
endmodule
```

门元件实现

```
module mux41(y, g, d0, d1, d2, d3, a);
  output[2:0] y;
  input[1:0] a;
  input g;
  input d0, d1, d2, d3;
  reg[2:0] y;
  wire nota1, nota2, x1, x2, x3, x4;
  not  (nota1, a[1]),
       (nota2, a[2]);
  and (x1, d0, nota1, nota[0]);
       (x2, d1, nota1, a[0]);
       (x3, d2, a[1], nota[0]);
       (x4, d3, a[1], a[0]);
  or  (y, x1, x2, x3, x4);
endmodule
```

数据流方式实现

```
module mux4_1a(y, g, d0, d1, d2, d3, a);
  output y;
  input[1:0] a;
  input g;
  input d0, d1, d2, d3;
  reg y;
assign y =
((d0&~a[1]&~a[0])|(d1&~a[1]&a[0])

|(d2&a[1]&~a[0])|(d3&a[1]&a[0]))&g;
endmodule
```

条件运算符描述实现

```
module mux4_1a(y, g, d0, d1, d2, d3, a);
  output y;
  input[1:0] a;
  input g;
  input d0, d1, d2, d3;
  reg y;
assign y =
g?(a[1]?(a[0]?d3:d2) :(a[0]?d1:d0)):0;
endmodule
```

数据分配器实现的功能与数据选择器相反。数据分配器是将一个数据源根的数据根据需要送到不同的通道上，实现数据分配功能的逻辑电路成为数据分配器。

```
module dmux (y0, y1, y2, y3, din, sel);
   output y0, y1, y2, y3;
   input[1:0] sel;
   input din;
   reg y0, y1, y2, y3;
  always @  (din, sel)
    begin
     y0 = 0; y1 = 0; y2 = 0; y3 = 0;
     case(sel[1:0])
     2'b00: y0 = din;
     2'b01: y1 = din;
     2'b10: y2 = din;
     2'b11: y3 = din;
     default:;
     endcase
   end
endmodule
```

```
module comparator (y1, y2, y3, a, b);
   output y1, y2, y3;
   input[3:0] a, b;
   reg y0, y1, y2, y3;
  always @  (a, b)
    begin
     if(a > b) begin
          y1 = 1; y2 = 0; y3 = 0;
            end
     else if(a == b) begin
          y1 = 0; y2 = 1; y3 = 0;
            end
     else if(a < b) begin
          y1 = 0; y2 = 0; y3 = 1;
            end
   end
endmodule
```

数值比较器

```
//4位全加器的行为描述
module add4 (cin, sum, cout, a, b);
  output[3:0] sum;
  output cout;
  input[3:0] a, b;
  input cin;
  reg cout;
  reg[3:0] sum;
 always @ (*)
  begin
   {cout, sum} = a + b + cin;
  end
endmodule
```

```
//混合方式
module add1 (cin, sum, cout, a, b);
  output sum, cout;
  input a, b;
  reg cout, m1, m2, m3;
  wire s1;
 xor  (s1, a, b);
always @ (a, b, cin)
  begin
    m1 = a & b;
    m2 = a & cin;
    m3 = cin & b;
    cout = (m1 | m2) | m3;
  end
 assign sum = s1 ^ cin;
endmodule
```

超前进位加法器

```
module fulladd4(sum, c_out, a, b, cin);
 output [3:0] sum;
 output c_out;
 input [3:0] a, b;
 input cin;
 wire p0, g0, p1, g1, p2, g2, p3, g3;
 wire c4, c3, c2, c1;
 assign p0 = a[0] ^ b[0];
        p1 = a[1] ^ b[1];
        p2 = a[2] ^ b[2];
        p3 = a[3] ^ b[3]
 assign g0 = a[0] & b[0];
        g1 = a[0] & b[1];
        g2 = a[0] & b[2];
        g3 = a[0] & b[3];
```

```
 assign c1 = g0 | (p0 & cin),
        c2 = g1 | (p1 & c1) ,
        c3 = g2 | (p2 & c2),
        c4 = g3 | (p3 & c2);

 assign sum[0] = p0 ^ cin;
        sum[1] = p1 ^ c1;
        sum[2] = p2 ^ c2;
        sum[3] = p3 ^ c3;

 assign cout = c4;
endmodule
```

```
//行为描述，4位全减器
module sub4 (cin, dout, cout, a, b);
 output[3:0] dout;
 output cout;
  input[3:0] a, b;
  input cin;
  reg[3;0] dout,
  reg cout;
 always @ (a, b)
  begin
    {cout, dout} = a – b – cin;
  end
endmodule
```

- 分为两类：

(1) 连续赋值语句——assign语句，用于对wire型变量赋值，是描述组合逻辑最常用的方法之一。

[例] assign c=a&b; //a、b、c均为wire型变量

(2) 过程赋值语句——用于对reg型变量赋值，有两种方式：

- 非阻塞(non-blocking)赋值方式：
  赋值符号为<=，如b <= a；
- 阻塞(blocking)赋值方式：
  赋值符号为=，如b = a；

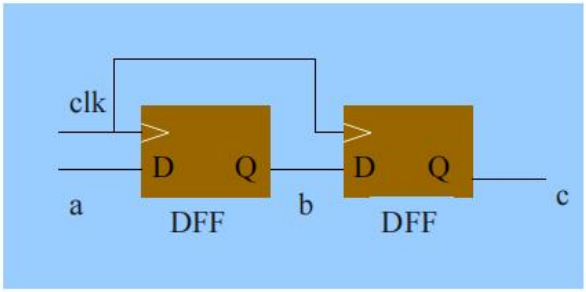
23

1. 非阻塞赋值方式

注：c的值比b的值落后一个时钟周期！

```
always @(posedge clk)
  begin
    b <= a ;
    c <= b;
  end
```

非阻塞赋值在块结束时才完成赋值操作！



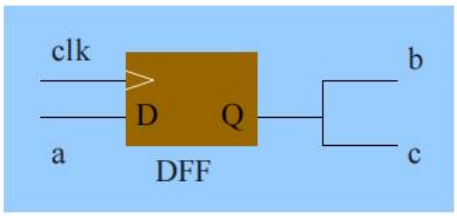
非阻塞的意思是每条赋值语句的结果直到 always块的结尾才能看到。

always 块中所有非阻塞赋值语句在求值时所用的值全部都是进入 always 时，各个变量已具有的值。

2. 阻塞赋值方式

```
always @(posedge clk)
  begin
    b = a ;
    c = b;
  end
```

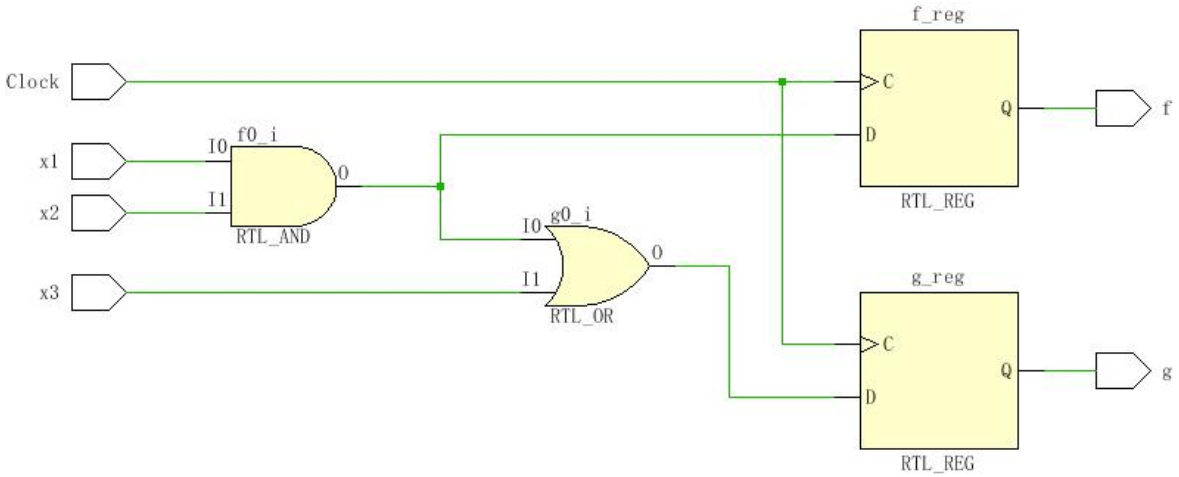
阻塞赋值在该语句结束时就完成赋值操作！



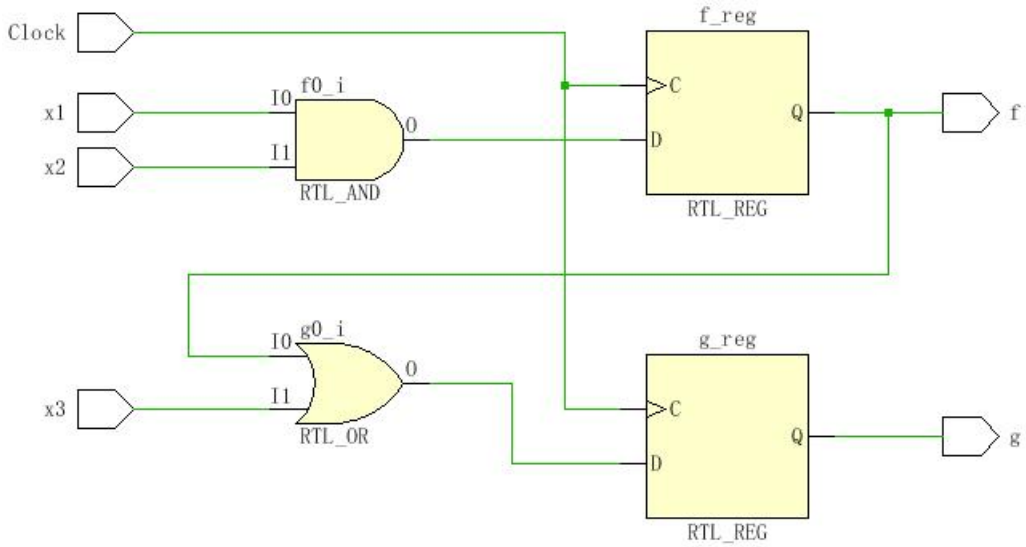
注：在一个块语句中，如果有多条阻塞赋值语句，在前面的赋值语句没有完成之前，后面的语句就不能被执行，就像被阻塞了一样，因此称为阻塞赋值方式。

这里c的值与b的值一样 ！

```
87 module example7_5 (x1, x2, x3, Clock, f, g);
88     input x1, x2, x3, Clock;
89     output reg f, g;
90     always @(posedge Clock)
91     begin
92         f = x1 & x2;
93         g = f | x3;
94     end
95 endmodule
```




```
87 module example7_5 (x1, x2, x3, Clock, f, g);
88     input x1, x2, x3, Clock;
89     output reg f, g;
90     always @(posedge Clock)
91     begin
92         f <= x1 & x2;
93         g <= f | x3;
94     end
95 endmodule
```




```
module cnt16 (cout, q, clk, clr, load, en, d);
  output[3:0] q; //输出
  output cout; //进位信号
   input clk, clr, load, en;
   input[3:0] d;
   reg[3:0] q;
   reg cout;
  always @  (posedge clk)  begin
    if (clr) begin q <= 0; end
    else if (load) begin q <= d; end
          else if (en) begin
                    q <= q + 1;
                    if(q == 4'b1111) begin cout <= 1; end
                    else begin  cout <= 0; end
                end
                else begin q <= q; end
    end
endmodule
```

同步4位计数器，同步清零，同步置数。

```
module cnt24 (ten, one, cout, clk, clr);
  output[3:0] ten, one; //输出
  output cout; //进位信号
   input clk, clr;
   reg[3;0] ten, one;
   reg cout;
  always @  (posedge clk) begin
    if (clr) begin ten <= 0; one <= 0; end
    else begin
            if({ten, one}) == 8'b0010_0011) //24十进制
               begin ten <= 0; one <= 0; cout <= 1; end
             else if(one==4'b1001)
                   begin  one <= 0; ten<=ten+1;
                          cout <= 0; end
                    else begin
                          one <= one + 1; cout <=0; end
          end
end
endmodule
```

同步24进制计数器，同步清零。

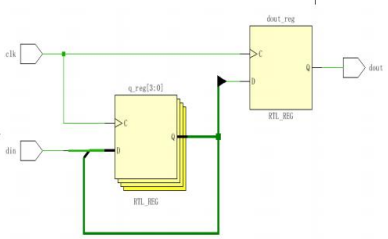
```
module count60(qout, cout,
  data, load, cin, reset, clk);
 output [7:0] qout;
 output cout;
 input [7:0] data;
 input load, cin, clk, reset;
 reg [7:0] qout;
 always @ (posedge clk)
 begin
      if (reset) qout <= 0;
      else if (load) qout <= data;
           else if (cin) begin
                 if(qout[3:0] == 9) begin
                    qout[3:0] <= 0;
                     if(qout[7:4] == 5) qout[7:4] <= 0;
                     else  qout[7:4] <= qout[7:4]+1;
                  end
                else   qout[3:0] <= qout[3:0]+1;
          end
     end
 assign cout = ((qout == 8'h59)&cin)?1:0;
endmodule
```

异步4位2进制计数器

```
module yb_cnt4
        (q, clk, rst);
  output[3:0] q;
  input clk, rst;
  reg[3:0] q;
  reg[3:0] qn;
always @ (posedge clk)
 begin if(!rst) begin q[0] = 0; qn[0] = 1; end
        else begin  q[0] = ~q[0]; qn[0] =~q[0]; end
 end
 always @ (posedge qn[0])
 begin if(!rst) begin q[1] = 0; qn[1] = 1; end
        else begin q[1] = ~q[1]; qn[1] =~q[1]
 end
 always @ (posedge qn[1])
 begin if(!rst) begin q[2] = 0; qn[2] = 1; end
        else begin q[2] = ~q[2]; qn[2] =~q[2]; end
 end
 always @ (posedge qn[2])
 begin if(!rst) begin q[3] = 0; qn[3] = 1; end
        else begin q[3] = ~q[3]; qn[3] =~q[3]; end
 end
endmodule
```

移位寄存器

```
module siso4 (dout, clk, din);//串入串出
  output dout;//
  input clk;
  input din;
  reg dout;
  reg[3:0] q;
 always @ (posedge clk)
   begin
     q[0] <= din;
     q[3:1] <= q[2:0];
      dout <= q[3];
   end
endmodule
```



```
module sipo (dout, din,clr,clk);//串入并出
  output[4:0] dout;
  input clk, din,clr;
  reg[4:0] dout;  //五位
 always @ (posedge clk )
   begin
     if(clr) begin
       dout <= 0;
     end
     else begin
          dout <= {dout, din};
     end
endmodule
```

分频系数不是2 的整数次幂

```
module div6(div6, clk);
  output div6;
  input clk;
  reg div6;
  reg[2:0] cnt;
 always @ (posedge clk)
    begin
      if (cnt == 3'b010) begin
           div6 <= ~ div6;
            cnt <= 0;
          end
      else begin
             cnt <= cnt + 1;
             end
   end
endmodule
```