数字系统设计

华东理工大学电子与通信工程系
主讲:木昌洪
Email：changhongmu@ecust.edu.cn

第5章 组合逻辑电路

本章介绍组合逻辑电路的特点、组合逻辑电路的分析方法和设计方法（基于Verilog HDL）；加法器、编码器、译码器、数据选择器、数值比较器、奇偶校验器等常用组合逻辑电路的电路结构、工作原理和使用方法。

- 5.1 概述
- 5.2 常用组合逻辑电路及其设计方法

本 章 重 点

- 组合逻辑电路的分析方法和设计方法；
- 常用组合逻辑电路的电路结构和逻辑功能；
- 编码器、译码器、数据选择器的应用；
- 基于Verilog HDL的组合逻辑电路设计方法。

内容概要

5.1.1 组合逻辑电路的结构和特点

5.1.2 组合逻辑电路的分析方法

5.1.3 组合逻辑电路的设计方法

5.1.1 组合逻辑电路的结构和特点

- 按照逻辑功能的不同特点，数字电路分为两大类：
  - 组合逻辑电路和时序逻辑电路

- **组合逻辑电路是将逻辑门以一定的方式组合在一起，使其具有一定逻辑功能的数字电路。**
- 它是一种无记忆电路——任一时刻的输出信号仅取决于该时刻的输入信号，而与信号作用前电路原来所处的状态无关。

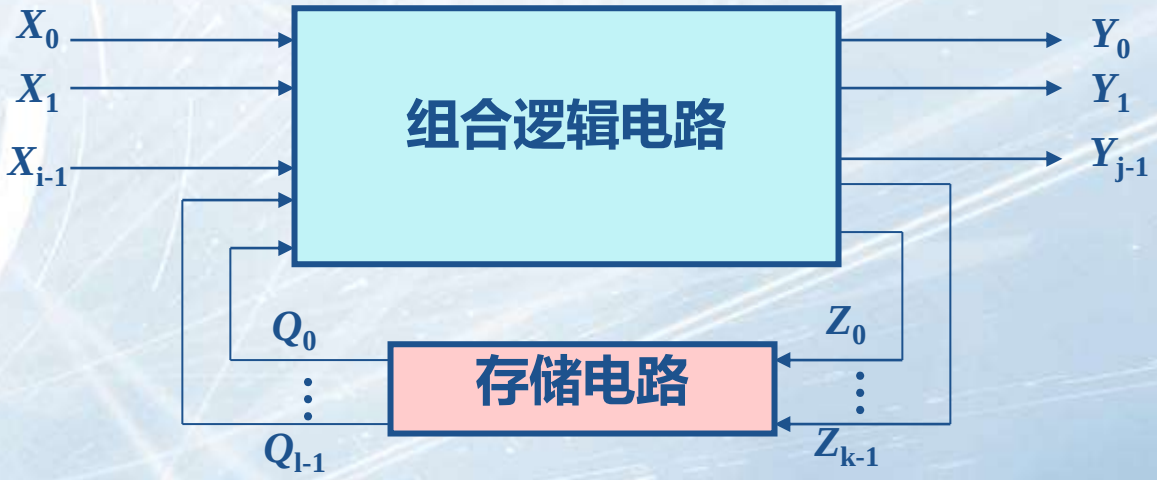
- 组合逻辑电路的特点
  - 由逻辑门电路组成
  - 没有反馈电路和存储电路
  - 当时的输出仅由当时的输入决定——速度快

组合逻辑电路的表述方法

- 组合逻辑电路可以用逻辑函数表达式、真值表、卡诺图、逻辑图及波形图分析和表述
  - 逻辑函数表达式
    - 一般为与或式，但形式不唯一，通过变换可实现用不同门电路组成逻辑图；一定程度上可以直接用于自动设计（如HDL）的描述
  - 真值表
    - 直观反映变量取值与函数值之间的关系，具有唯一性，有利于自动设计（如HDL）的描述
  - 卡诺图
    - 过去化简逻辑函数的主要工具，现在几乎已不使用
  - 逻辑图
    - 直观表示变量之间的逻辑关系，一个逻辑函数表达式可以用不同的逻辑图实现；一般只适于简单电路的描述
  - 波形图
    - 直观表示输入与输出信号的波形，通过分析波形可以得到真值表

时序逻辑电路

- 如果某逻辑电路任一时刻的输出信号不仅取决于当时的输入信号，而且还取决于电路原来的状态，则称为时序逻辑电路。
- 时序逻辑电路的特点
  - 由组合逻辑电路和存储电路两部分组成。
  - 具有“记忆”功能——任一时刻的输出信号不仅取决于该时刻的输入信号，而且还取决于电路原来的状态，即还与以前的输入有关。



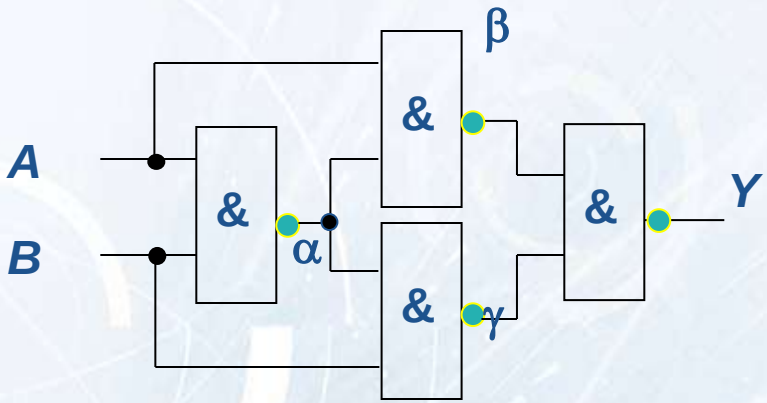
5.1.2 组合逻辑电路的分析方法

- 组合逻辑电路的分析——根据给定的组合逻辑电路，通过分析确定其逻辑功能

逻辑图 → 逻辑表达式 → 真值表 → 电路功能

化简

【例5.1】分析下图电路



A B	Y
0 0	0
0 1	1
1 0	1
1 1	0

$$\alpha = \overline{AB}$$

$$\beta = \overline{\alpha A} = \overline{\overline{AB}A}$$

$$\gamma = \overline{\alpha B} = \overline{\overline{AB}B}$$

$$Y = \overline{\beta\gamma} = \overline{\overline{AB}A} \cdot \overline{\overline{AB}B} = \overline{AB}A + \overline{AB}B$$

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

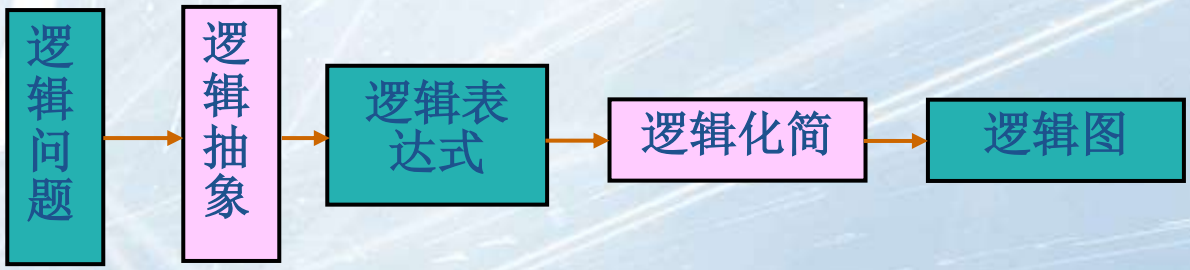
电路功能：异或电路

5.1.3 组合逻辑电路的设计方法

- **组合逻辑电路的设计——根据给定的功能要求，采用某种设计方法，得到满足功能要求、且最简单的组合逻辑电路。**

组合逻辑电路的手工设计方法

- 逻辑抽象——确定输入、输出变量，列出真值表
- 写出逻辑函数表达式——根据真值表写出逻辑函数的标准表达式
- 逻辑化简——用公式化简法或卡诺图化简法化简为最简逻辑函数表达式
- 绘逻辑图——根据最简逻辑函数表达式画出原理图



组合逻辑电路的自动设计方法

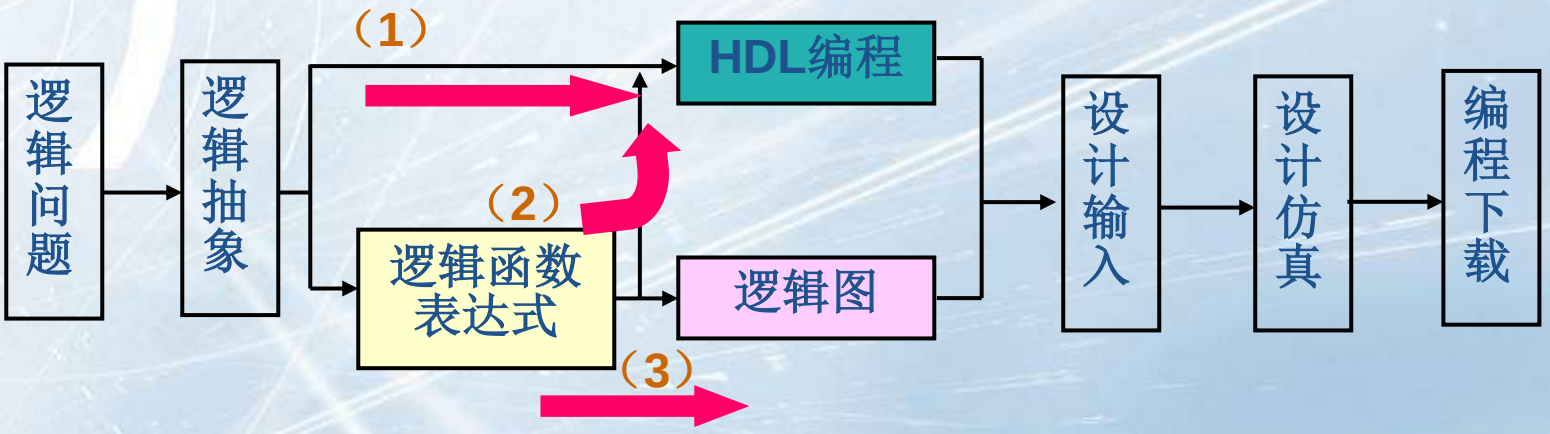
- 基于HDL和EDA工具的组合逻辑电路的设计方法
  - 逻辑抽象——确定输入、输出变量，列出真值表（复杂系统也可不写出真值表，而直接用HDL的系统级描述方式）
  - HDL编程——如用case语句、if-else语句，assign语句
  - 写出逻辑表达式——根据真值表写出逻辑函数的标准表达式

- **有3种途径**

（1）逻辑抽象→HDL编程（系统级描述，如用case语句或if-else语句）

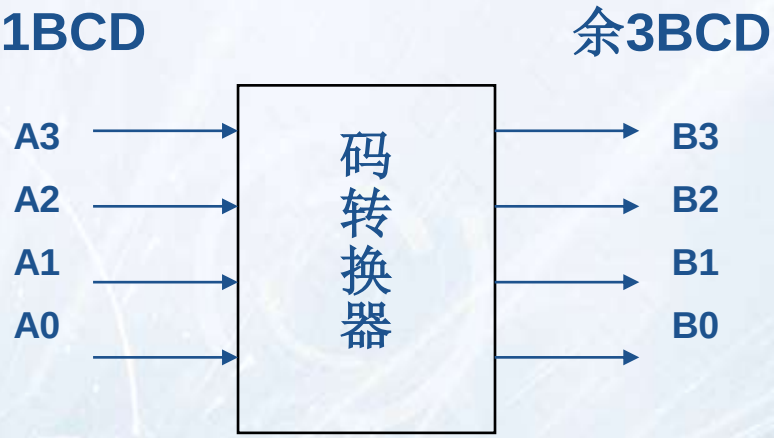
（2）逻辑抽象→写出逻辑函数表达式→HDL编程（算法级描述，assign语句）

（3）逻辑抽象→写出逻辑函数表达式→绘逻辑图（适于简单电路）



组合逻辑电路的设计方法举例

【例5.2 】 8421BCD码转换为余3BCD码的码转换器的设计



- 分析
  - 余3BCD码由每个8421BCD码加上3得到，直接列出真值表
  - 1010~1111不会在输入端出现，作为约束项（输入变量取值组合不允许出现或不会出现，或者出现与否对输出没有影响，这些取值组合代表的最小项称为约束项）处理，对应输出用x表示

A3 A2 A1 A0	B3 B2 B1 B0
0000	0011
0001	0100
0010	0101
0011	0110
0100	0111
0101	1000
0110	1001
0111	1010
1000	1011
1001	1100
1010	xxxx
1011	xxxx
1100	xxxx
1101	xxxx
1110	xxxx
1111	xxxx

HDL编程

（1）case语句描述——系统级抽象

根据输入与输出间的真值表

```
module bcd8421(A,B);
    input[3:0]      A;
    output[3:0]     B;
    reg[3:0]        B;
    always @(A)
    begin
        case (A)
            0 : B = 3;    1 : B = 4;
            2 : B = 5;    3 : B = 6;
            4 : B = 7;    5 : B = 8;
            6 : B = 9;    7 : B = 10;
            8 : B = 11;  9 : B = 12;
           default : B = 4'hx;
        endcase
   end
endmodule
```

（2）if语句描述——系统级抽象

根据电路的逻辑功能定义

```
module bcd8421_1(A,B);
    input[3:0]      A;
    output[3:0]     B;
    reg[3:0]        B;
    always @(A)
        begin
            if (A <= 9)    B = A + 3;
            else           B =4'hx;
        end
endmodule
```

5.2 常用组合逻辑电路及其设计方法

内容概要

5.2.1 算术运算电路

5.2.2 编码器

5.2.3 译码器

5.2.4 数据选择器

5.2.5 数值比较器

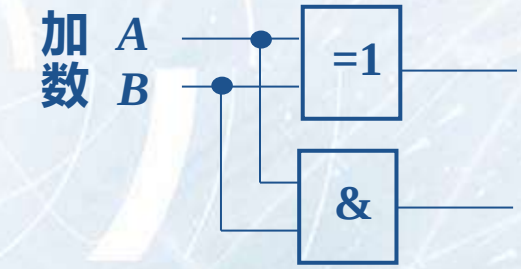
5.2.6 奇偶校验器

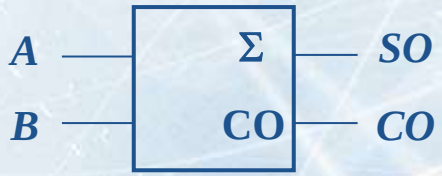
5.2.1 算术运算电路

- 常用的组合逻辑电路有算术运算电路、编码器、译码器、数据选择器、数值比较器、奇偶校验器等
- 算术运算电路是能完成二进制数算术运算的器件
- 半加器和全加器是算术运算电路的基本单元电路

1、半加器

- **半加器——能对两个1位二进制数进行相加求和，并向高位进位的逻辑电路。**
- 特点：不考虑来自低位的进位。





$$SO = A \oplus B$$
$$CO = AB$$

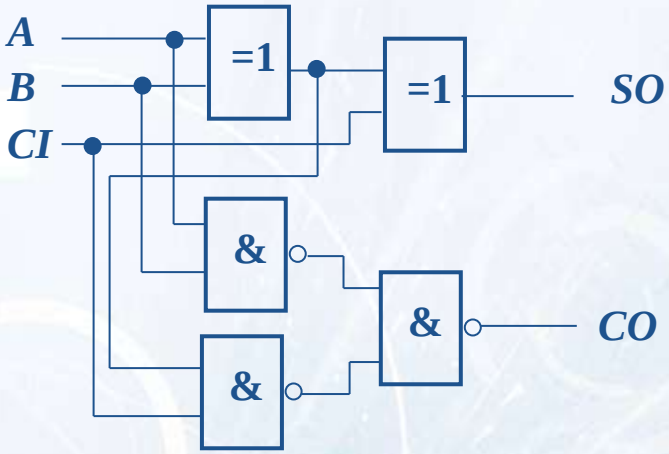
真值表

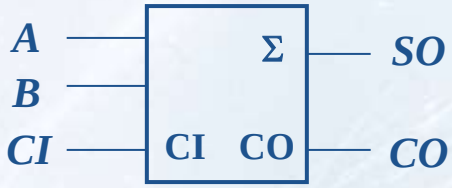
A	B	SO	CO
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2、全加器

1位全加器——能对两个1位二进制数进行相加并考虑低位来的进位、求得和并向高位进位的逻辑电路称为全加器。

特点：考虑来自低位的进位的加法运算电路





$$SO = A \oplus B \oplus CI$$
$$= (A \oplus B)\cdot \overline{CI} + \overline{A \oplus B}\cdot CI$$
$$= (A\overline{B} + \overline{A}B)\overline{CI} + (\overline{A}\overline{B} + AB)CI$$
$$= A\overline{B}\,\overline{CI} + \overline{A}B\overline{CI} + \overline{A}\,\overline{B}CI + ABCI$$

$$CO = \overline{\overline{(A \oplus B)CI}\cdot\overline{AB}}$$
$$= (A\overline{B} + \overline{A}B)CI + AB$$
$$= A\overline{B}CI + \overline{A}BCI + AB$$

真值表

A B CI	SO	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

1位全加器的HDL设计

- 方法一：根据全加器的功能列出1位全加器的真值表，由真值表推出输出的逻辑表达式，然后用assign语句建模（算法级描述）

$$SO = \overline{A}\,\overline{B}CI + \overline{A}B\overline{C}I + A\overline{B}\,\overline{C}I + ABCI$$

$$CO = \overline{A}BCI + A\overline{B}CI + AB\overline{C}I + ABCI$$

真值表

A B CI	SO	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

1位全加器Verilog HDL源程序（assign建模）

```
module adder_1(A,B,CI,SO,CO);
    input   A,B,CI;
    output  SO,CO;
    assign  SO = (!A&&!B&&CI)||(!A&&B&&!CI)||
                 (A&&!B&&!CI)||(A&&B&&CI);
    assign  CO =(!A&&B&&CI)||(A&&!B&&CI)||
                 (A&&B&&!CI)||(A&&B&&CI);
endmodule
```

1位全加器Verilog HDL源程序（行为描述）

方法二：采用行为描述方式的系统级抽象根据逻辑功能定义直接描述，程序更简洁！

```
module adder_2(A,B,CI,SO,CO);
    input       A,B,CI;
    output      SO,CO;
    assign      {CO,SO} = A+B+CI;
endmodule
```

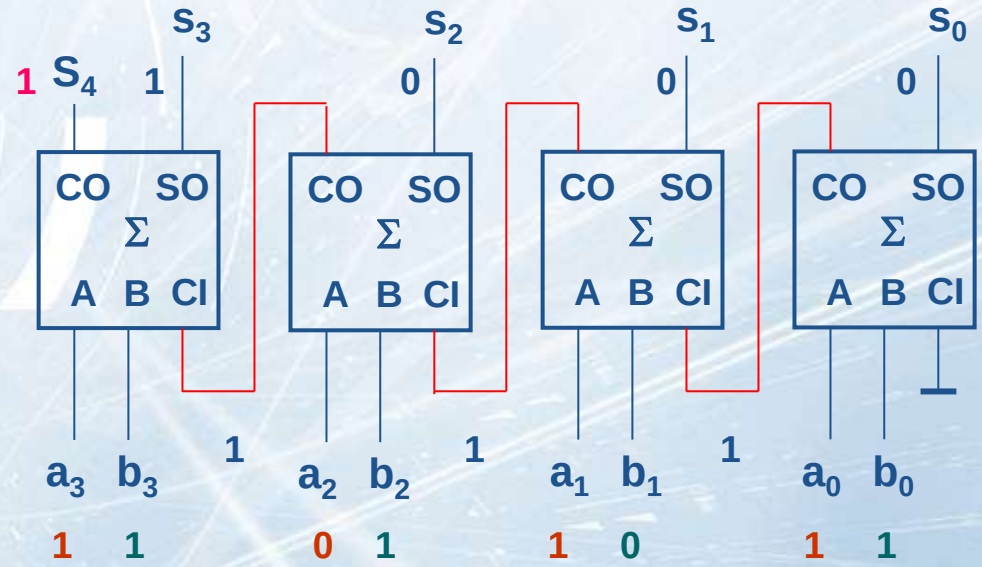
- 这里用位拼接运算符 “{ }”将进位与算术和拼接在一起成为一个2位数

3、多位加法器

- 能实现多位二进制数相加的电路称为**多位加法器（Adder）**。
- 由多个1位全加器可以扩展成多位加法器。
- 按进位方式不同，多位加法器分为串行进位加法器和并行进位加法器。

（1）串行进位加法器

- 串行进位加法器是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*，加法从低位开始，高位全加必须等低位进位来到后才能进行，因此完成加法的时间较长。
- 优点：电路比较简单，连接方便；缺点：运算速度不高。



4位串行进位加法器

$A = a_3a_2a_1a_0(=1011)$

$B = b_3b_2b_1b_0(=1101)$

$S = A + B = (s_4)\ s_3s_2s_1s_0$

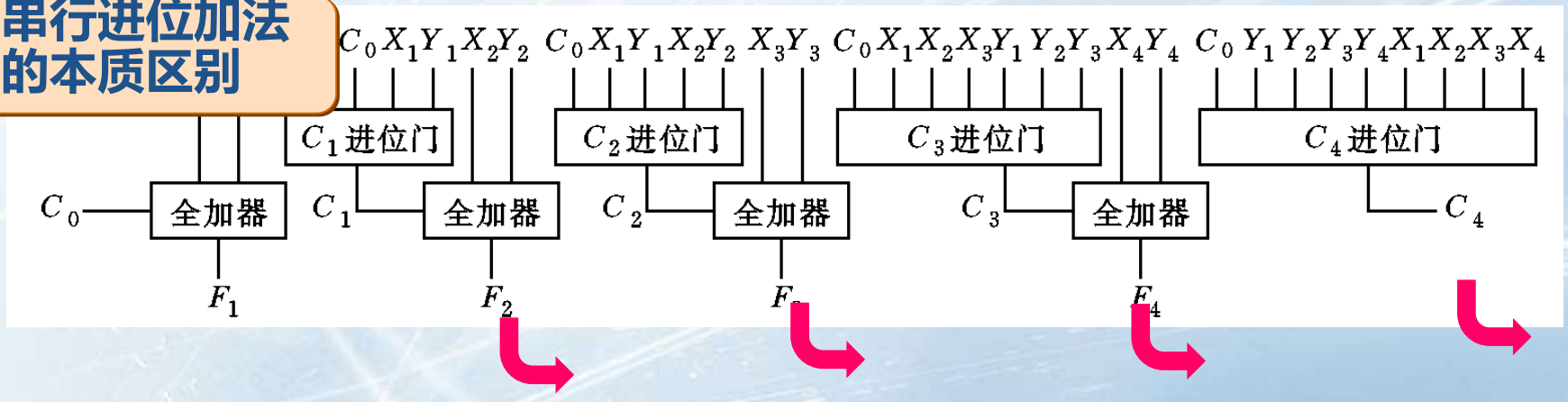
$=1\ 1000$

（2）并行进位加法器（超前进位加法器）

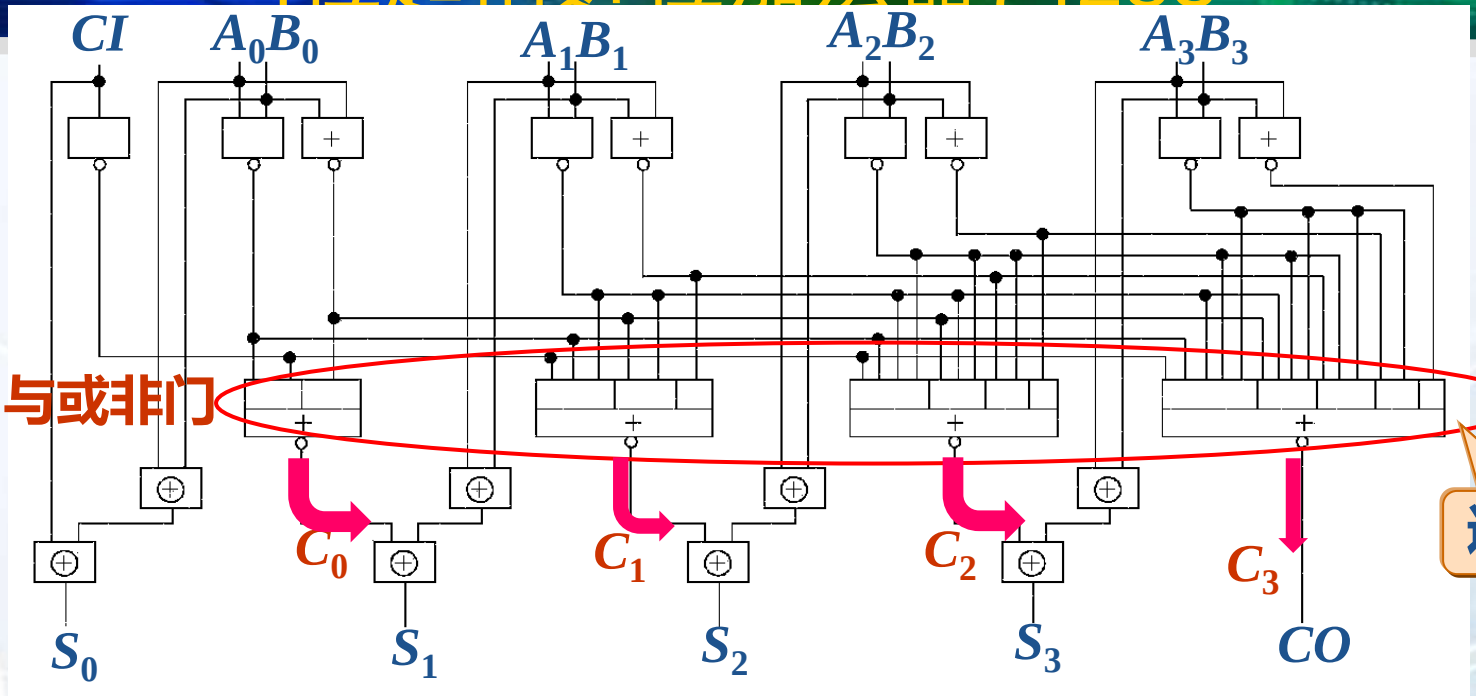
（2）并行进位加法器

- 串行进位加法器运算速度不高，且延迟级数与位数成正比。
- 考虑设置专用的进位形成电路同时产生各位的进位Cn，进位输入由专门的“进位门”综合所有低位的加数、被加数及最低位进位来提供，这样构成的加法器称为并行进位加法器，又称超前进位或快速进位加法器。
- **各位进位由所有低位的加数、被加数及最低位进位C_0来决定，而与前一级加法器的进位输出无关，多位加法器的加法运算可以同时进行，因此完成加法的时间较快——提高了运算速度。**

与串行进位加法器的本质区别

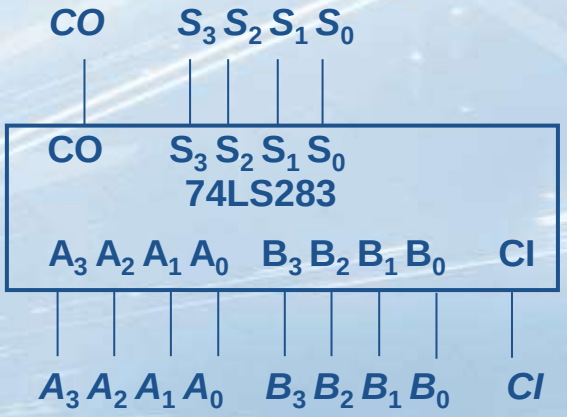


4位超前进位加法器74283



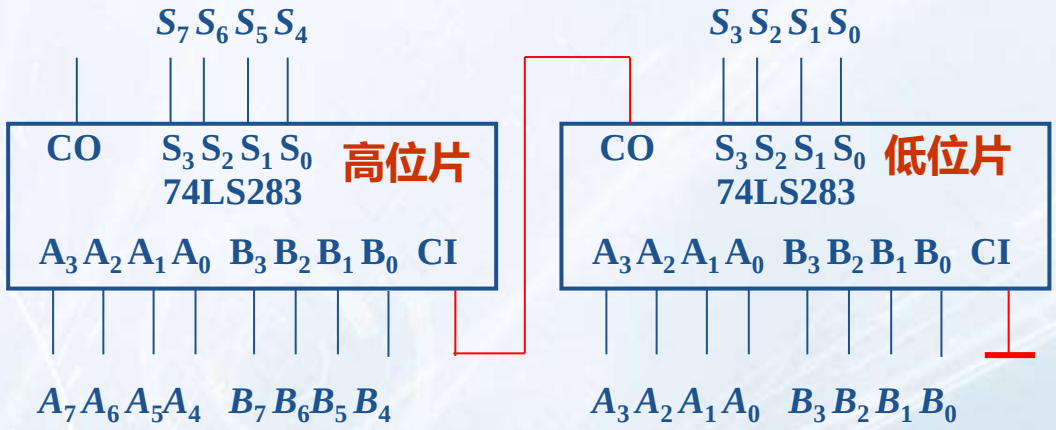
$$S_0 = A_0 \oplus B_0 \oplus CI$$

$$C_0 = \overline{\overline{A_0} + \overline{B_0} + \overline{A_0} \cdot \overline{B_0} \cdot \overline{CI}}$$



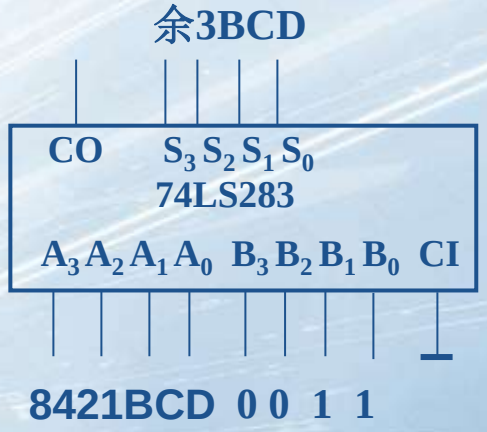
74283 扩展方法及主要用途

① 扩展——用2片74283构成8位加法器
片内超前进位、片间串行进位



② 用74283构成8421BCD码到余3BCD码的转换电路

余3BCD码= 8421BCD码 + 3



多位加法器的设计

- 用Verilog HDL行为描述方式很容易编写出任意位数的加法器电路。
- 8位加法器的Verilog HDL源程序adder_8.v:

```
module adder_8(a,b,cin,sum,cout);
    parameter    width=8;
    input [width-1:0]      a,b;
    input                  cin;
    output [width-1:0]     sum;
    output                 cout;
    assign          {cout,sum} = a+b+cin;
endmodule
```

- **这里用parameter常量width表示加法器的位数，通过修改width，可以方便地实现不同位宽的加法器。**

5.2.2 编码器

- 为了区分一系列不同的事物，将其中的每个事物用一组二值（0或1）代码表示；或者说，用二进制代码来表示特定信息——编码的含义。
- 将加在电路若干输入端中的某一个输入端的信号变换成相应的一组二进制代码输出的过程叫做编码。
- 实现编码功能的数字电路称为编码器（Encoder）。
- 编码器的作用是将某一时刻仅一个输入有效的多个输入的变量情况用较少的输出状态组合来表达，或者说将输入的每一个高、低电平信号编成一组对应的二进制代码，以便于后续的识别和处理。
- 通常有二进制编码器、BCD码编码器及优先编码器。

编码器的应用举例

【例5.3】一个7层高的大楼，每层有一个火警报警传感器，如有火警希望在控制中心的数码显示屏上显示出火警的楼层数，假设不会在两层上同时出现火警。

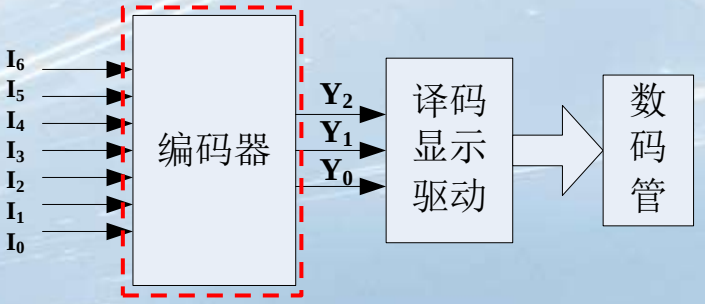
- 分析：译码显示驱动器电路其输入是8421BCD码，而现在大楼每层有一个传感器，相当于输入有7个信号，这里无法直接与译码显示电路相连，须在两者之间加上一个转换电路——编码器，将其7种状态转换为4位（或更少位）的二进制输出。

编码表

输入	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

由于有7个传感器，并且同一时刻仅一个传感器有效，故输入共7种状态，可以用3位二进制数据来描述，假设输入用I_0、I_1、I_2、I_3、I_4、I_5、I_6表示，输出为Y_2、Y_1、Y_0。

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



1、二进制编码器

- 用n位二进制代码对$M=2^n$个信号进行编码的电路叫二进制编码器。
  - 特点：任意一时刻只能对一个信号进行编码，即任何时刻只允许一个输入信号有效（低电平或高电平），而其余信号为无效电平，否则输出将发生混乱。
  - n位二进制符号可以表示2^n种信息，称为2^n线-n线编码器。
  - 常用的有8线-3线编码器
- 如果在输入等于1时对输入信号进行编码，则称这类编码器为高电平输入有效，此时输出等于对应有效输入的编号的二进制编码；如果在输入等于0时对输入信号进行编码，则称这类编码器为低电平输入有效。

8线-3线编码器（高电平输入有效）

- 如果任何时刻输出编码仅对应一个有效输入信号，则对某位输出，在编码表中挑出所有为“1”的值，将其对应的输入信号**相或**，得到输出表达式。

$$C = Y_4 + Y_5 + Y_6 + Y_7 = \overline{\overline{Y_4} \cdot \overline{Y_5} \cdot \overline{Y_6} \cdot \overline{Y_7}}$$

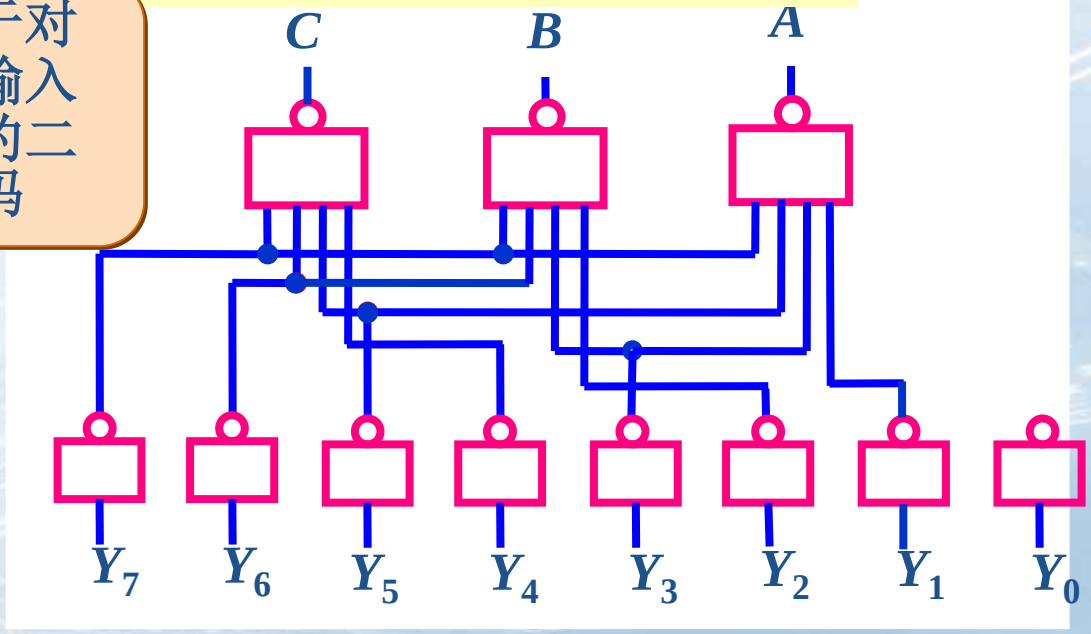
$$B = Y_2 + Y_3 + Y_6 + Y_7 = \overline{\overline{Y_2} \cdot \overline{Y_3} \cdot \overline{Y_6} \cdot \overline{Y_7}}$$

$$A = Y_1 + Y_3 + Y_5 + Y_7 = \overline{\overline{Y_1} \cdot \overline{Y_3} \cdot \overline{Y_5} \cdot \overline{Y_7}}$$

编码表

输入	C B A
Y_0	0 0 0
Y_1	0 0 1
Y_2	0 1 0
Y_3	0 1 1
Y_4	1 0 0
Y_5	1 0 1
Y_6	1 1 0
Y_7	1 1 1

输出等于对应有效输入的编号的二进制编码



8线-3线编码器的真值表

真值表（高电平输入有效）

Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	C	B	A
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	1	1	x	x	x
0	0	0	0	0	1	1	1	x	x	x
约束项……								……		
1	1	1	1	1	1	1	1	x	x	x

- 利用最小项推导法写出各输出的逻辑函数表达式

$$C=\overline{Y}_7\overline{Y}_6\overline{Y}_5Y_4\overline{Y}_3\overline{Y}_2\overline{Y}_1\overline{Y}_0+\overline{Y}_7\overline{Y}_6Y_5\overline{Y}_4\overline{Y}_3\overline{Y}_2\overline{Y}_1\overline{Y}_0$$
$$+\overline{Y}_7Y_6\overline{Y}_5\overline{Y}_4\overline{Y}_3\overline{Y}_2\overline{Y}_1\overline{Y}_0+Y_7\overline{Y}_6\overline{Y}_5\overline{Y}_4\overline{Y}_3\overline{Y}_2\overline{Y}_1\overline{Y}_0$$

- 如果任何时刻Y_7~Y_0中仅有一个输入取值为1，即输入变量取值的组合仅有表中的前8种状态，则输入变量为其他取值下输出等于1的那些最小项均为约束项。利用这些约束项化简上式，得到：

$$C=Y_4+Y_5+Y_6+Y_7$$

8线-3线编码器的HDL设计

- 方法一：根据8线-3线编码器的功能列出真值表，由真值表推出输出的逻辑表达式，然后用assign语句建模（算法级描述）

```
module encoder8_3(Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7, C,B,A);
    input   Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7;
    output  C,B,A;
    assign   C =! (!Y4&&!Y5&&!Y6&&!Y7);
    assign   B =! (!Y2&&!Y3&&!Y6&&!Y7);
    assign   A =! (!Y1&&!Y3&&!Y5&&!Y7);
endmodule
```

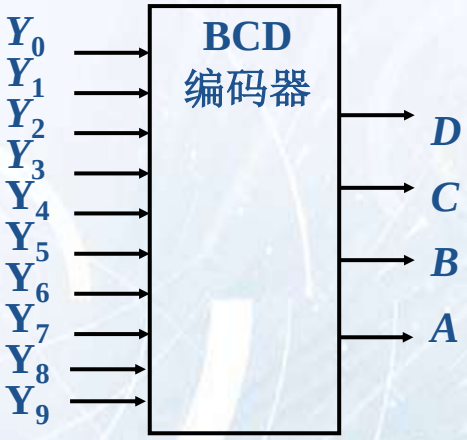
还可以写为?

- 方法二：根据逻辑功能定义，采用case语句直接描述，设计过程更简单！

```
reg     C,B,A;
always
  case ({Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7})
        'b10000000 : {C,B,A} = 0;
        'b01000000 : {C,B,A} = 1;
        ……
        'b00000001 : {C,B,A} = 7;
         default : {C,B,A} = 3'bx;
   endcase
```

2、BCD码编码器

- BCD码编码器就是用二进制码表示十进制数的编码器，也称为二-十进制编码器，或称为10线-4线编码器。
- 用4位二进制代码对十进制数的10个数码进行编码。
- BCD有多种编码方式：8421BCD、2421BCD或余3BCD。
- 通常用8421BCD来表示十进制数，构成8421BCD编码器。



高电平输入有效

- **10个输入端，分别接代表十进制数0~9的10个按键**

编码表

输入	D C B A
Y_0	0 0 0 0
Y_1	0 0 0 1
Y_2	0 0 1 0
Y_3	0 0 1 1
Y_4	0 1 0 0
Y_5	0 1 0 1
Y_6	0 1 1 0
Y_7	0 1 1 1
Y_8	1 0 0 0
Y_9	1 0 0 1

$$D = Y_8 + Y_9 = \overline{\overline{Y_8} \cdot \overline{Y_9}}$$

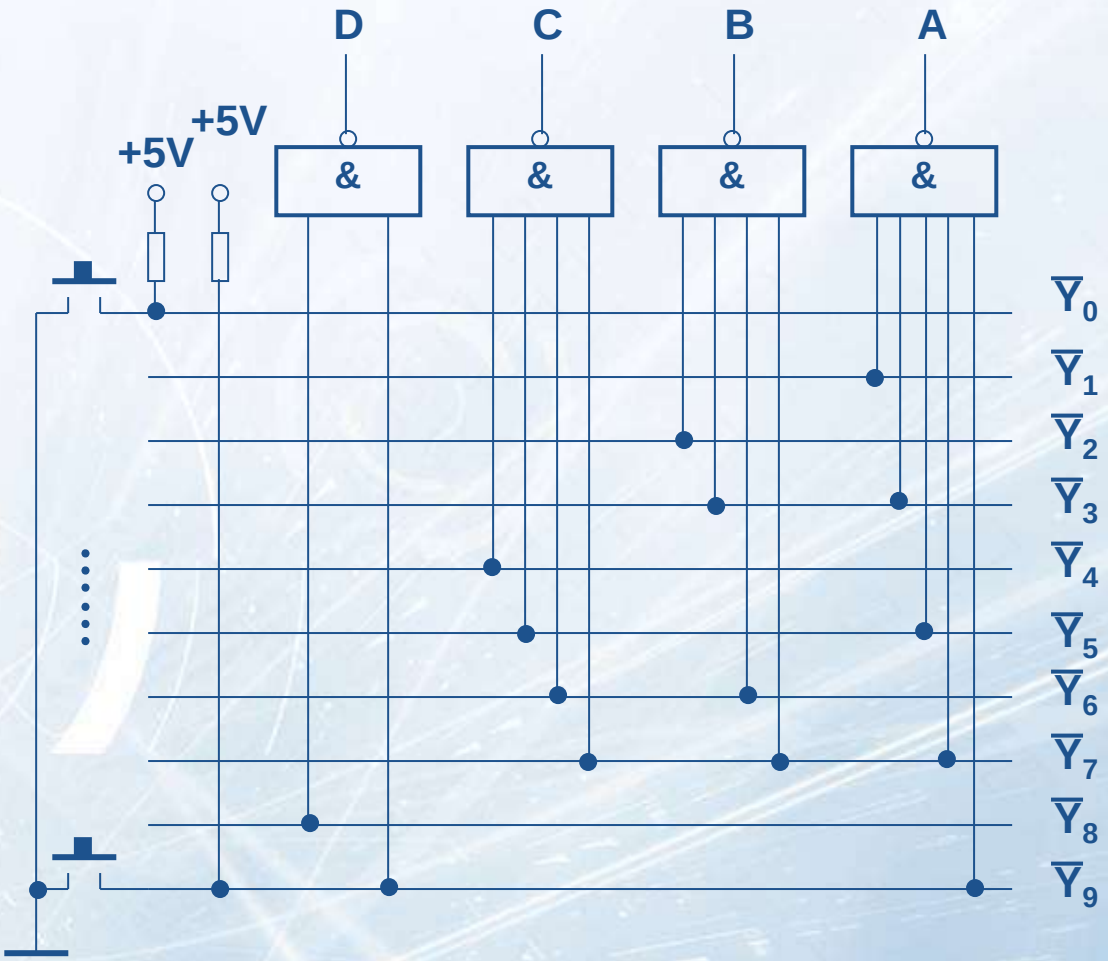
$$C = Y_4 + Y_5 + Y_6 + Y_7 = \overline{\overline{Y_4} \cdot \overline{Y_5} \cdot \overline{Y_6} \cdot \overline{Y_7}}$$

$$B = Y_2 + Y_3 + Y_6 + Y_7 = \overline{\overline{Y_2} \cdot \overline{Y_3} \cdot \overline{Y_6} \cdot \overline{Y_7}}$$

$$A = Y_1 + Y_3 + Y_5 + Y_7 + Y_9 = \overline{\overline{Y_1} \cdot \overline{Y_3} \cdot \overline{Y_5} \cdot \overline{Y_7} \cdot \overline{Y_9}}$$

8421BCD编码器的逻辑图

根据逻辑表达式可以直接画出逻辑图



低电平输入有效

8421BCD编码器的Verilog HDL源程序

- **根据逻辑功能定义，直接采用case 语句描述——设计过程最简单！**
- 假设**高电平输入有效**
- **Y0=1时，DCBA=0000；Y1=1时，DCBA=0001；……**

  Y9=1时，DCBA=1001。

```
module bcd8421_3(Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,Y9,D,C,B,A);
    input          Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,Y9;
    output         D,C,B,A;
    reg            D,C,B,A;
    always
      begin
        case ({Y9,Y8,Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0})
          10'b00_0000_0001: {D,C,B,A} = 0;
          10'b00_0000_0010 : {D,C,B,A} = 1;
          10'b00_0000_0100 : {D,C,B,A} = 2;
          10'b00_0000_1000 : {D,C,B,A} = 3;
          10'b00_0001_0000 : {D,C,B,A} = 4;
          10'b00_0010_0000 : {D,C,B,A} = 5;
          10'b00_0100_0000 : {D,C,B,A} = 6;
          10'b00_1000_0000 : {D,C,B,A} = 7;
          10'b01_0000_0000 : {D,C,B,A} = 8;
          10'b10_0000_0000 : {D,C,B,A} = 9;
           default : {D,C,B,A} = 4'bxxxx;
        endcase
      end
endmodule
```

3、优先编码器

- 二进制编码器要求任何时刻只允许有一个输入信号有效，否则输出将发生混乱——当同时有多个输入信号有效时不能使用二进制编码器！
- 优先编码器可以避免这种情况发生。优先编码器事先对所有输入信号进行优先级别排序，允许两位以上的输入信号同时有效；但任何时刻只对优先级最高的输入信号编码，对优先级别低的输入信号则不响应，从而保证编码器可靠工作。

- **如有两个或两个以上的输入有效时，只对优先级最高的输入信号进行编码的编码器称为优先编码器。**
  - 优点：当有两个或两个以上的输入有效时，输出不会发生混乱。
  - 广泛应用于计算机的优先中断系统、键盘编码系统中。
- **74LS148——8线-3线优先编码器，8个输入信号，低电平有效；3个输出端，反码输出**
- **74LS147——10线-4线优先编码器，10个输入信号，低电平有效；4个输出端，反码输出**

计算机键盘按键编码原理

- 1984年，IBM推出了IBM AT键盘接口标准，该标准定义了84-101键。1987年，IBM又推出了PS/2键盘接口标准，采用6脚mini-DIN连接器使键盘与计算机相连。
- 键盘包含一个由若干键组成的矩阵。所有的键都被键盘中的一个板上处理器监控，称作键盘编码器（encoder），其功能是对每一个键按照一定的规则进行编码；监控是哪个或哪几个键被按下或释放，并把相应的编码数据送到主机。encoder大部分时间在“扫描”，监视着键矩阵。一旦发现有键被按下，或释放，或被按住不放，encoder就会向计算机发送一个数据包（称为扫描码）。
- 有两种不同的扫描码：通码（make code）和断码（break code）。当一个键被按下或者按住不放时就发送通码，当一个键被释放时就发送断码。每个按键被分配了唯一的通码和断码。这样，主机通过查找唯一的扫描码就可以确定是哪个按键被按下或释放。
- 通码和断码的集合称为扫描码集。共有3种标准的扫描码集。所有现代的键盘都支持并默认使用第二套扫描码集。

部分按键的扫描码（十六进制）

按 键	通 码	断 码
1/!	16	F0_16
2/@	1E	F0_1E
3/#	26	F0_26
4/$	25	F0_25
5/%	2E	F0_2E
6/^	36	F0_36
7/&	3D	F0_3D
8/*	3E	F0_3E
9/(	46	F0_46
Shift（左）	12	F0_12
Shift（右）	59	F0_59
↑	E0_75	E0_F0_75
→	E0_74	E0_F0_74
←	E0_6B	E0_F0_6B
↓	E0_72	E0_F0_72
Ctrl（左）	14	F0_14
Ctrl（右）	E0_14	E0_F0_14

- 通码和断码的发送序列举例：当需要在编辑器中敲入“！”时，先按下【Shift】键，再按下【1/!】键，然后释放【1/!】键，再释放【Shift】键，则发送到计算机的字节序列为“12h,16h,F0h,16h,F0h,12h”。
- 计算机根据编码识别按下的键是“！”，再进行后续处理

优先编码器 (74147)的设计

10线－4线优先编码器CT74147的输入信号为$\bar{I}_0 \sim \bar{I}_9$，$\bar{I}_9$的优先权最高，$\bar{I}_0$最低。

4线输出信号为$\bar{Y}_3 \sim \bar{Y}_0$，当$\bar{I}_9=0$（有效）时，$\bar{Y}_3 \sim \bar{Y}_0=0110$（“9”的BCD码的反码），依此类推。

$\bar{I}_9$	$\bar{I}_8$	$\bar{I}_7$	$\bar{I}_6$	$\bar{I}_5$	$\bar{I}_4$	$\bar{I}_3$	$\bar{I}_2$	$\bar{I}_1$	$\bar{I}_0$	$\bar{Y}_3$	$\bar{Y}_2$	$\bar{Y}_1$	$\bar{Y}_0$
0	x	x	x	x	x	x	x	x	x	0	1	1	0
1	0	x	x	x	x	x	x	x	x	0	1	1	1
1	1	0	x	x	x	x	x	x	x	1	0	0	0
1	1	1	0	x	x	x	x	x	x	1	0	0	1
1	1	1	1	0	x	x	x	x	x	1	0	1	0
1	1	1	1	1	0	x	x	x	x	1	0	1	1
1	1	1	1	1	1	0	x	x	x	1	1	0	0
1	1	1	1	1	1	1	0	x	x	1	1	0	1
1	1	1	1	1	1	1	1	0	x	1	1	1	0
1	1	1	1	1	1	1	1	1	0	1	1	1	1

低电平输入有效

输出等于优先级最高的输入信号对应编号的反码

优先编码器(74147)的Verilog HDL源程序

- 利用if_else语句的分支具有先后顺序的特点，用if_else语句可方便地实现优先编码器。

```
module
 CT74147(IN0,IN1,IN2,IN3,IN4,IN5,
IN6,IN7,IN8,IN9,YN0,YN1,YN2,YN3);
    input     IN0,IN1,IN2,IN3, IN4,
              IN5,IN6,IN7,IN8,IN9;
   output    YN0,YN1,YN2,YN3;
   reg        YN0,YN1,YN2,YN3;
   reg[3:0]   Y; //中间变量
   always @(IN0 or IN1 or IN2 or
            IN3 or IN4 or IN5 or IN6
            or IN7 or IN8 or IN9)
     begin
      if (IN9 == 1'b0)        Y = 4'b0110;
      else if (IN8 == 1'b0)   Y = 4'b0111;
      else if (IN7 == 1'b0)   Y = 4'b1000;
      else if (IN6 == 1'b0)   Y = 4'b1001;
      else if (IN5 == 1'b0)   Y = 4'b1010;
      else if (IN4 == 1'b0)   Y = 4'b1011;
      else if (IN3 == 1'b0)   Y = 4'b1100;
      else if (IN2 == 1'b0)   Y = 4'b1101;
      else if (IN1 == 1'b0)   Y = 4'b1110;
      else if (IN0 == 1'b0)   Y = 4'b1111;
      YN0 = Y[0];
      YN1 = Y[1];
      YN2 = Y[2];
      YN3 = Y[3];
   end
endmodule
```

电平有效型输入在参数表中可以忽略，也可写为always

优先编码器的应用实例—医院病室呼叫控制电路

【例5.4 】某医院有一、二、三、四号病室4间，每室设有呼叫按钮，同时在护士值班室内对应地装有一号、二号、三号、四号4个指示灯。

现要求当一号病室的按钮按下时，无论其他病室的按钮是否按下，只有一号灯亮。当一号病室的按钮没有按下而二号病室的按钮按下时，无论三、四号病室的按钮是否按下，只有二号灯亮。当一、二号病室的按钮都未按下而三号病室的按钮按下时，无论四号病室的按钮是否按下，只有三号灯亮。只有在一、二、三号病室的按钮均未按下而按下四号病室的按钮时，四号灯才亮。

试用优先编码器74HC148和门电路设计满足上述控制要求的逻辑电路，给出控制4个指示灯状态的高、低电平信号。

设计思路

解： 74xx148——8线-3线优先编码器，8个输入信号，低电平有效；3个输出端，反码输出。

输入信号低电平有效

若以 $\overline{A_1}$、$\overline{A_2}$、$\overline{A_3}$、$\overline{A_4}$的低电平分别表示一、二、三、四号病室按下按钮时给出的信号，则将它们接到74HC148的 $\overline{I_3}$ 、 $\overline{I_2}$ 、 $\overline{I_1}$ 、 $\overline{I_0}$ 输入端以后，便在74HC148的输出端 $\overline{Y_2}$ 、$\overline{Y_1}$ 、$\overline{Y_0}$ 得到了对应的输出编码。

若以 Z_1、Z_2 、Z_3 、Z_4 分别表示一、二、三、四号灯的点亮信号（输出信号高电平有效），还需将74HC148输出的代码译成 Z_1 、Z_2、 Z_3、Z_4 对应的输出高电平信号。可列出电路的逻辑真值表。

真值表和输出的逻辑函数表达式

真值表

$\overline{A_1}$	$\overline{A_2}$	$\overline{A_3}$	$\overline{A_4}$	$\overline{Y_2}$	$\overline{Y_1}$	$\overline{Y_0}$	Z_1	Z_2	Z_3	Z_4
0	x	x	x	1	0	0	1	0	0	0
1	0	x	x	1	0	1	0	1	0	0
1	1	0	x	1	1	0	0	0	1	0
1	1	1	0	1	1	1	0	0	0	1

低电平输入有效

输出高电平有效

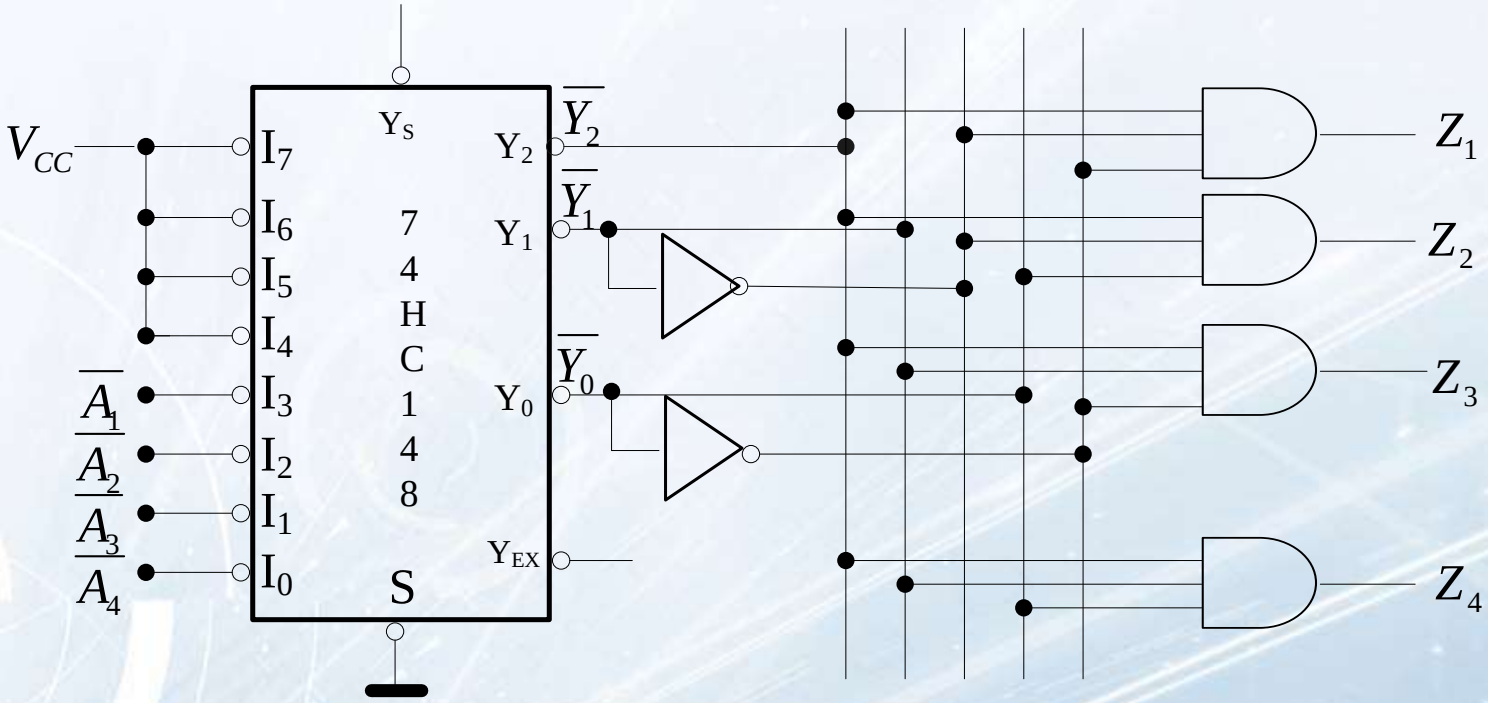
当/A_1有效时，即/I_3 有效，其输入编号为3，则输出/Y_2、/Y_1、/Y_0为3的反码=100。

输出的逻辑函数表达式

$$Z_1 = \overline{Y_2}Y_1Y_0 \qquad Z_2 = \overline{Y_2}Y_1\overline{Y_0}$$

$$Z_3 = \overline{Y_2}\,\overline{Y_1}Y_0 \qquad Z_4 = \overline{Y_2}\,\overline{Y_1}\,\overline{Y_0}$$

电路连接图



思考：如果本例采用HDL实现，应该怎样描述？

5.2.3 译码器

- 将二进制代码所表示的信息翻译成对应输出的高低电平信号的过程称为译码，译码是编码的反操作。实现译码功能的电路称为译码器（Decoder）。
- 常用的译码器有变量译码器、码制变换译码器和显示译码器。
  - 变量译码器（二进制译码器）是用来表示输入变量状态全部组合的译码器。n个输入代码有2^n个状态，因此n位二进制译码器有n个输入端和2^n个输出端，一般称为n线-2^n线译码器。常用的有双2线-4线译码器74××139，3线-8线译码器74××138，4线-16线译码器74××154等。
  - 码制变换译码器是将输入的某个进制代码转换成对应的其他码制输出的译码器。如二-十进制码（8421码）至十进制码译码器（简称BCD译码器）、余3码至十进制码译码器、余3循环码至十进制码译码器等。
  - 显示译码器是将输入代码转换成驱动7段数码显示器各段的电平信号的译码器。常用的有74××47（低电平输出有效）、74××49（高电平输出有效）、74××48（高电平输出有效）等。

1、二进制译码器

- 二进制译码器将每个输入二进制代码译成对应的一根输出线上的高电平（或低电平）信号。因此称为n线-2^n线译码器。
- 二进制编码器将输入的每一个高（或低）电平信号编成一个对应的二进制代码。为避免输出混乱，任何时刻只允许一个输入信号有效。
- 二进制译码器功能与二进制编码器正好相反，它是将具有特定含义的不同二进制代码辨别出来，并转换成相应的电平信号。

特点

① 任何时刻最多只允许1个输出有效

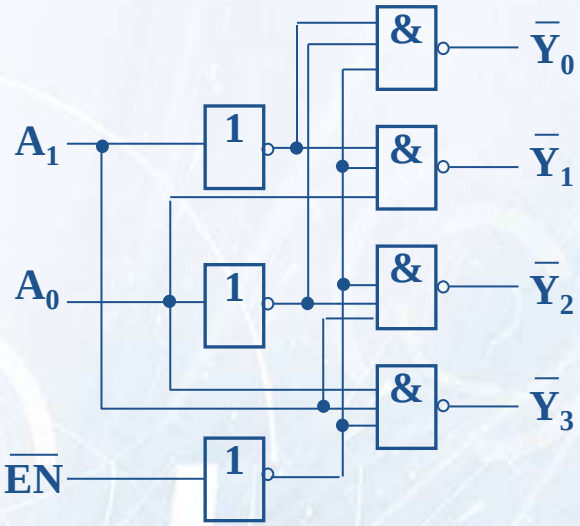
——与编码器对应，任何时刻只允许有一个输入为有效电平

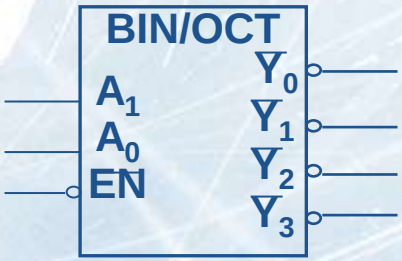
② 高电平输出有效时，每个输出都是对应的输入最小项；低电平输出有效时，每个输出都是对应的输入最小项的反

——二进制译码器也称为最小项译码器。

（1）2线-4线译码器（74139）

- 低电平输出有效
- $\overline{EN}$ 为使能控制端（选通信号），为0时，译码器处于工作状态；为1时，处于禁止工作状态。





$$\overline{Y}_0 = \overline{\overline{A}_1 \overline{A}_0} = \overline{m}_0 \qquad \overline{Y}_1 = \overline{\overline{A}_1 A_0} = \overline{m}_1$$

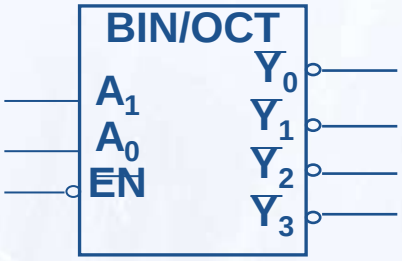
$$\overline{Y}_2 = \overline{A_1 \overline{A}_0} = \overline{m}_2 \qquad \overline{Y}_3 = \overline{A_1 A_0} = \overline{m}_3$$

❖ 当低电平输出有效时，每个输出都是对应的输入变量最小项的反

功能表

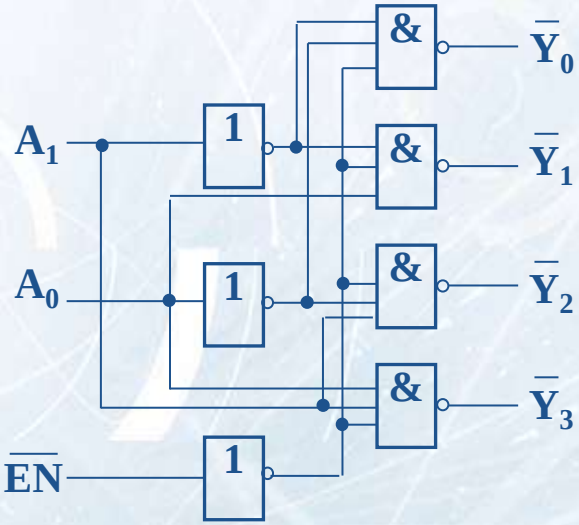
$\overline{EN}$	A_1	A_0	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
1	X	X	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

2线-4线译码器（74139）的手工设计方法



① 真值表

$\overline{EN}$	A_1	A_0	$\overline{Y_3}$	$\overline{Y_2}$	$\overline{Y_1}$	$\overline{Y_0}$
1	X	X	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



③ 画出逻辑图

② 根据真值表推导出逻辑表达式（最大项推导法）

$$\overline{Y}_0 = A_1 + A_0 = \overline{\overline{A_1 + A_0}} = \overline{\overline{A}_1 \overline{A}_0} = \overline{m}_0$$

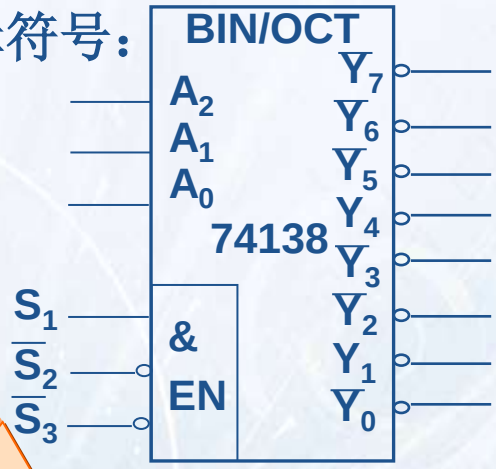
$$\overline{Y}_1 = \overline{\overline{A}_1 A_0} = \overline{m}_1$$

$$\overline{Y}_2 = \overline{A_1 \overline{A}_0} = \overline{m}_2$$

$$\overline{Y}_3 = \overline{A_1 A_0} = \overline{m}_3$$

（2）3线-8线译码器（74138）

逻辑符号：



功能表

$S_1\overline{S_2}\overline{S_3}$	A_2 A_1 A_0	Y_7 Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 Y_0
≠100	X X X	1 1 1 1 1 1 1 1
=100	0 0 0	1 1 1 1 1 1 1 0
=100	0 0 1	1 1 1 1 1 1 0 1
=100	0 1 0	1 1 1 1 1 0 1 1
=100	0 1 1	1 1 1 1 0 1 1 1
=100	1 0 0	1 1 1 0 1 1 1 1
=100	1 0 1	1 1 0 1 1 1 1 1
=100	1 1 0	1 0 1 1 1 1 1 1
=100	1 1 1	0 1 1 1 1 1 1 1

S_1、/S_2、/S_3为3个使能输入端，只有当它们分别为1、0、0时，译码器才正常译码；否则禁止工作

$$\overline{Y_0}=\overline{\overline{A_2}\,\overline{A_1}\,\overline{A_0}}=\overline{m_0};\ \overline{Y_1}=\overline{\overline{A_2}\,\overline{A_1}A_0}=\overline{m_1}$$

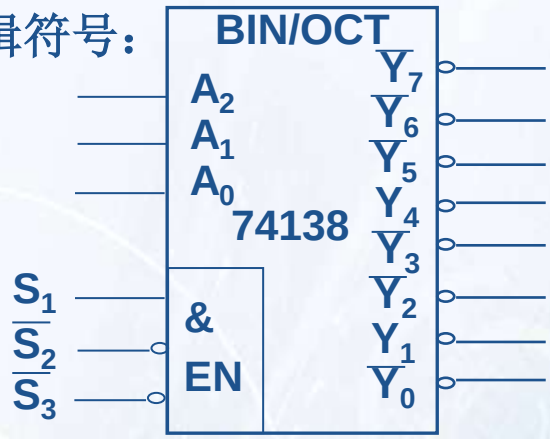
$$\overline{Y_2}=\overline{\overline{A_2}A_1\overline{A_0}}=\overline{m_2};\ \overline{Y_3}=\overline{\overline{A_2}A_1A_0}=\overline{m_3}$$

$$\overline{Y_4}=\overline{A_2\overline{A_1}\,\overline{A_0}}=\overline{m_4};\ \overline{Y_5}=\overline{A_2\overline{A_1}A_0}=\overline{m_5}$$

$$\overline{Y_6}=\overline{A_2A_1\overline{A_0}}=\overline{m_6};\ \overline{Y_7}=\overline{A_2A_1A_0}=\overline{m_7}$$

74138的手工设计方法

逻辑符号：



① 真值表

$S_1\overline{S_2}\overline{S_3}$	A_2 A_1 A_0	Y_7 Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 Y_0
≠100	X X X	1 1 1 1 1 1 1 1
=100	0 0 0	1 1 1 1 1 1 1 0
=100	0 0 1	1 1 1 1 1 1 0 1
=100	0 1 0	1 1 1 1 1 0 1 1
=100	0 1 1	1 1 1 1 0 1 1 1
=100	1 0 0	1 1 1 0 1 1 1 1
=100	1 0 1	1 1 0 1 1 1 1 1
=100	1 1 0	1 0 1 1 1 1 1 1
=100	1 1 1	0 1 1 1 1 1 1 1

② 根据真值表推导出逻辑表达式（最大项推导法）

$$\overline{Y_0} = \overline{\overline{A_2}\,\overline{A_1}\,\overline{A_0}} = \overline{m_0}; \overline{Y_1} = \overline{\overline{A_2}\,\overline{A_1}A_0} = \overline{m_1}$$

$$\overline{Y_2} = \overline{\overline{A_2}A_1\overline{A_0}} = \overline{m_2}; \overline{Y_3} = \overline{\overline{A_2}A_1A_0} = \overline{m_3}$$

$$\overline{Y_4} = \overline{A_2\overline{A_1}\,\overline{A_0}} = \overline{m_4}; \overline{Y_5} = \overline{A_2\overline{A_1}A_0} = \overline{m_5}$$

$$\overline{Y_6} = \overline{A_2A_1\overline{A_0}} = \overline{m_6}; \overline{Y_7} = \overline{A_2A_1A_0} = \overline{m_7}$$

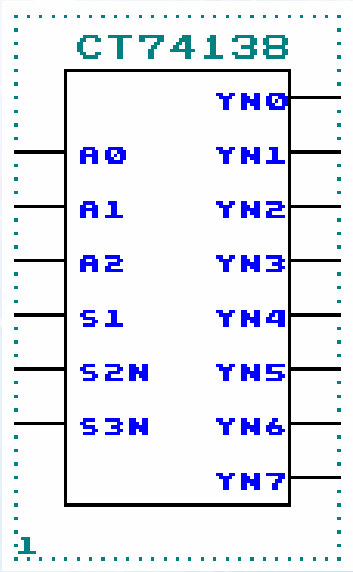
③ 画出逻辑图

74138的Verilog HDL设计方法（1/2）

HDL设计—— 采用if-else语句和case语句描述

分析： 分两种情况

- **当输入S1、/S2、/S3为100时，译码器工作；当S1、/S2、/S3不等于100时，译码器禁止工作——适合用if-else语句描述。**
- **在译码器工作时，根据输入A2~A0的取值不同，某一个输出信号为有效电平——适合用case语句描述。**




```
module CT74138(A0,A1,A2,S1,S2N,S3N,YN0,
               YN1,YN2,YN3,YN4,YN5,YN6,YN7);
    input       A0,A1,A2,S1,S2N,S3N;
    output      YN0,YN1,YN2,YN3,YN4,YN5, YN6,YN7;
    reg         YN0,YN1,YN2,YN3,YN4,YN5, YN6,YN7;
    reg[7:0]    Y_SIGNAL;
```

中间变量，便于对多个信号一次赋值

74138的Verilog HDL设计方法（2/2）

使能信号均有效时译码器处于工作状态

```
always
   begin
     if (S1 && !S2N && !S3N==1)
       begin
          case ({A2,A1,A0})
             3 'b000 : Y_SIGNAL =8 'b11111110;
             3 'b001 : Y_SIGNAL =8 'b11111101;
             3 'b010 : Y_SIGNAL =8 'b11111011;
             3 'b011 : Y_SIGNAL =8 'b11110111;
             3 'b100 : Y_SIGNAL =8 'b11101111;
             3 'b101 : Y_SIGNAL = 8'b11011111;
             3 'b110 : Y_SIGNAL =8 'b10111111;
             3 'b111 : Y_SIGNAL =8 'b01111111;
             default : Y_SIGNAL =8'b11111111;
          endcase
       end
```

使能信号无效时译码器处于禁止状态

```
     else Y_SIGNAL = 8'b11111111;
     YN0 = Y_SIGNAL[0];
     YN1 = Y_SIGNAL[1];
     YN2 = Y_SIGNAL[2];
     YN3 = Y_SIGNAL[3];
     YN4 = Y_SIGNAL[4];
     YN5 = Y_SIGNAL[5];
     YN6 = Y_SIGNAL[6];
     YN7 = Y_SIGNAL[7];
   end
endmodule
```

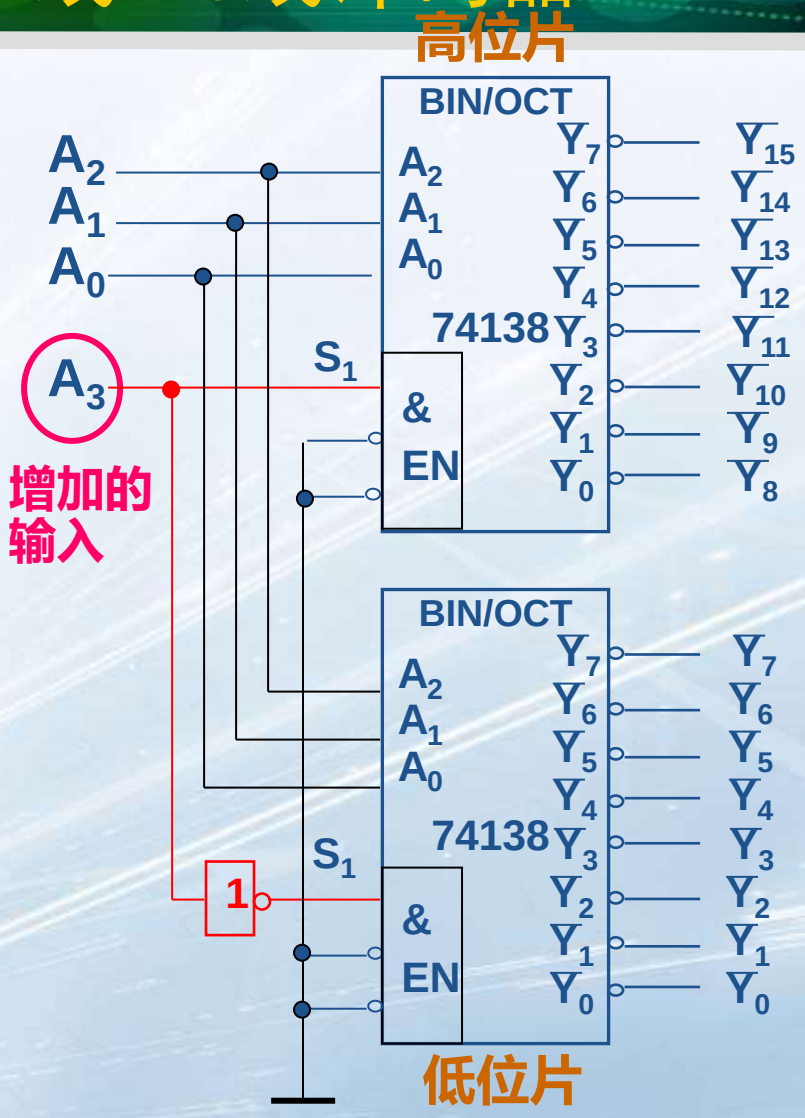
用2片74138扩展为4线-16线译码器

使能端的功能

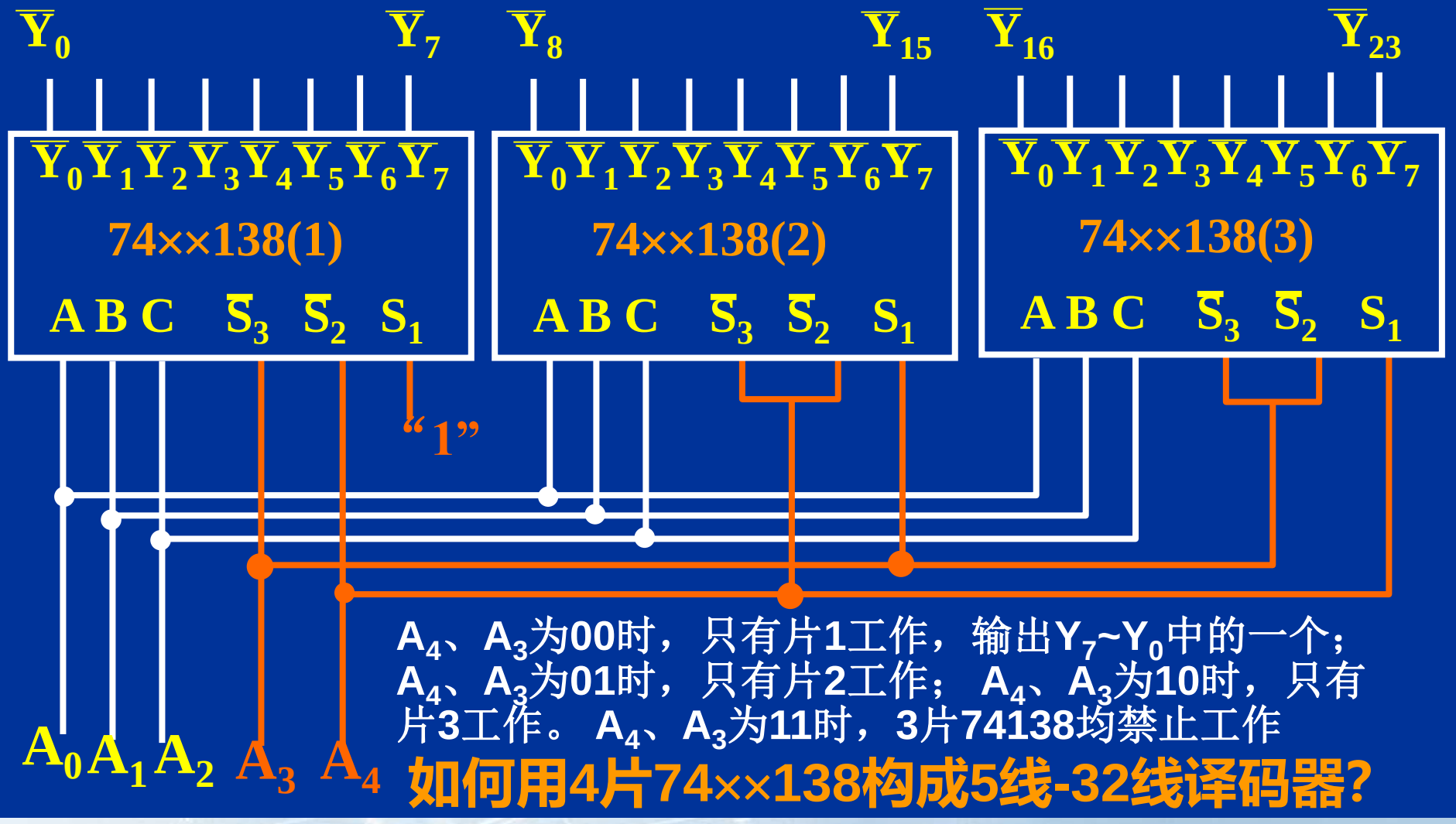
- 在集成电路中增加使能控制(Enable)端EN，是电路设计中常用的技术，使得集成电路更加灵活、可靠。
- 灵活：用于扩展；可靠：用于选通

译码器扩展方法

- 合理利用使能端S_1、$\overline{S_2}$、$\overline{S_3}$
- A_3=0时，低位片工作，高位片禁止，在A_2~A_0作用下，低位片的$\overline{Y_7}$~$\overline{Y_0}$作输出，对应输出$\overline{Y_7}$~$\overline{Y_0}$；
- A_3=1时，低位片禁止，高位片工作，在A_2~A_0作用下，高位片的$\overline{Y_7}$~$\overline{Y_0}$作输出，对应输出$\overline{Y_{15}}$~$\overline{Y_8}$。



用3片74138扩展为5线-24线译码器



A_4、A_3为00时，只有片1工作，输出Y_7~Y_0中的一个；A_4、A_3为01时，只有片2工作；A_4、A_3为10时，只有片3工作。A_4、A_3为11时，3片74138均禁止工作

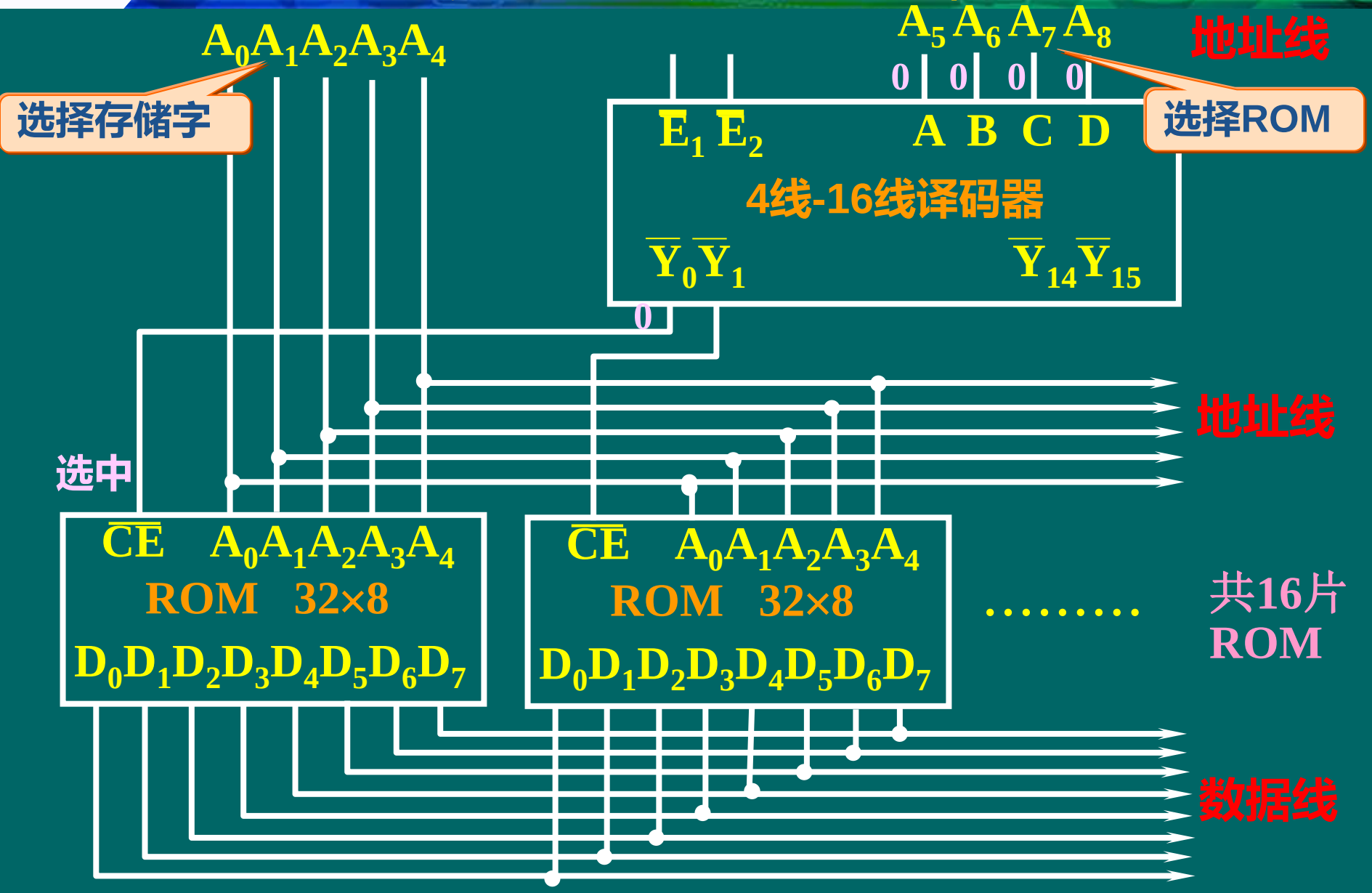
如何用4片74××138构成5线-32线译码器?

译码器应用之一：实现存储器系统的地址译码

【例5.5】假定由16片ROM （Read Only Memory，只读存储器）（32×8， 32个存储单元，位宽为8位）组成一个存储器系统，每片ROM有一个片选（/CE）端，当其为0时该片ROM被选中。利用译码器实现对每个存储单元的访问。

- 分析：
  - 16片ROM有16个片选端，如果不使用译码器，则计算机需要16根片选信号线。译码器可以将较少位数的二进制数据翻译成较多位数的信号，故可以使用4线-16线译码器，产生16个片选信号，接到每个ROM的片选端。
  - 由于每片ROM有32个存储单元，共16片ROM，则共有16×32=2^9个存储单元，可以用9根地址线来寻址。 由地址线A8~A5来选择ROM； 由地址线A4~A0 来选择被选中的ROM中32个存储单元中的某一个 。

存储器系统的地址译码电路



存储器系统的地址译码原理

- 每片32×8的三态输出的半导体只读存储器（ROM）只能存储32个字，每个字为8位，由地址A_4~A_0对32个字中的一个进行选择。当片选信号/CE=0，允许选中的字从D_7~D_0输出到数据线上；当/CE=1， ROM被禁止，输出D_7~D_0为高阻态。
- 本存储器系统共使用存储器16片，总容量为32×16=512个字，用地址码A_8~A_0对512（ = 2^9）个存储字进行选择。16片存储器的相应输出端D_7~D_0并联在一起，作为存储器系统的输出。
- 用一片4线-16线译码器的输出（低电平输出有效）控制存储器的片选端/CE，由高位地址码A_8~A_5从16片ROM中选出一片，再由低位地址码A_4~A_0从被选片中选择某一个存储字。A_4~A_0同时加至各存储器的地址输入端。

译码器应用之二：实现组合逻辑函数

- 二进制译码器每个输出都是对应输入的最小项（或最小项的非）
- 任意一个逻辑函数都可变换为最小项之和的标准形式，因此利用二进制译码器和门电路，可以实现单输出或多输出的任意组合逻辑函数

【例5.6】利用二进制译码器74138和门电路实现组合逻辑函数

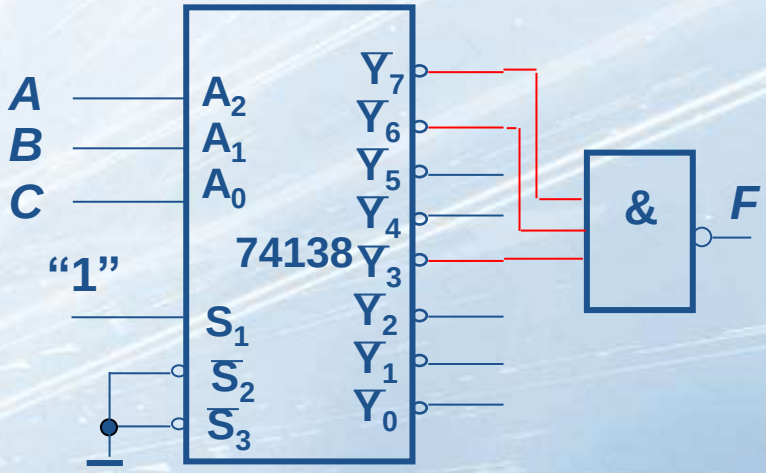
F（A,B,C）=AB+BC

解：

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

$$=AB\overline{C}+ABC+\overline{A}BC=\sum m(3,6,7)=\overline{\overline{m_3}\cdot\overline{m_6}\cdot\overline{m_7}}=\overline{\overline{Y_3}\cdot\overline{Y_6}\cdot\overline{Y_7}}$$

由上式画出逻辑图：



2、码制变换译码器

- 码制变换译码器是将输入的二进制代码转换成对应的其他码制输出的译码器。
  - 二-十进制译码器（BCD译码器，4-10线译码器）：将输入的4位8421码翻译成0~9十个十进制数的电路。用于驱动十进制数字显示管、指示灯、继电器
    - 完全译码的BCD译码器：当输入ABCD出现伪码0101~1111时，译码器输出Y_0~Y_9均为“1”，如74xx42（输出为低电平有效）
    - 不完全译码的BCD译码器：当ABCD = 0101~1111时，Y_0~Y_9均为任意值
  - 余3码至十进制码译码器
  - 余3循环码至十进制码译码器

完全译码的BCD译码器（74xx42）

- 输出Y_0~Y_9为低电平有效
- 当输入为无用码（伪码） 0101~1111时，译码器输出Y_0~Y_9均为“1”，不会出现低电平，不会产生错误译码

最高位

	A	B	C	D	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
1	1	0	0	0	1	0	1	1	1	1	1	1	1	1
2	0	1	0	0	1	1	0	1	1	1	1	1	1	1
3	1	1	0	0	1	1	1	0	1	1	1	1	1	1
4	0	0	1	0	1	1	1	1	0	1	1	1	1	1
5	1	0	1	0	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	1	1	1	0	1	1	1	1	1	1	1	0	1	1
8	0	0	0	1	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
不用	0	1	0	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
	0	0	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
	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	1	1	1

设计：根据真值表推导出逻辑表达式（最大项推导法），画出逻辑图

$$Y_0=A+B+C+D$$
$$=\overline{\bar{A}\bar{B}\bar{C}\bar{D}}$$
$$Y_1=\overline{\bar{A}B\bar{C}\bar{D}}$$
$$\vdots$$
$$Y_9=\overline{\bar{A}B\bar{C}D}$$

也可以直接用HDL来描述功能（case语句或assign语句）

74xx42的Verilog HDL源程序（1/2）

```
module CT7442(D,C,B,A, YN0, YN1,YN2, YN3,YN4,
        YN5,YN6,YN7,YN8,YN9);
    input       D,C,B,A;
    output      YN0,YN1,YN2,YN3,YN4,YN5, YN6,YN7,YN8,YN9;
    reg         YN0,YN1,YN2,YN3,YN4,YN5, YN6,YN7,YN8,YN9;
    reg[9:0]    Y_SIGNAL;
```

中间变量，便于对多个信号一次赋值

74xx42的Verilog HDL源程序（2/2）

```
always
   begin
      case ({D,C,B,A})
         'b0000 : Y_SIGNAL = 'b1111111110;
         'b0001 : Y_SIGNAL = 'b1111111101;
         'b0010 : Y_SIGNAL = 'b1111111011;
         'b0011 : Y_SIGNAL = 'b1111110111;
         'b0100 : Y_SIGNAL = 'b1111101111;
         'b0101 : Y_SIGNAL = 'b1111011111;
         'b0110 : Y_SIGNAL = 'b1110111111;
         'b0111 : Y_SIGNAL = 'b1101111111;
         'b1000 : Y_SIGNAL = 'b1011111111;
         'b1001 : Y_SIGNAL = 'b0111111111;
         default :Y_SIGNAL = 'b1111111111;
      endcase
```

```
      YN0 = Y_SIGNAL[0];
      YN1 = Y_SIGNAL[1];
      YN2 = Y_SIGNAL[2];
      YN3 = Y_SIGNAL[3];
      YN4 = Y_SIGNAL[4];
      YN5 = Y_SIGNAL[5];
      YN6 = Y_SIGNAL[6];
      YN7 = Y_SIGNAL[7];
      YN8 = Y_SIGNAL[8];
      YN9 = Y_SIGNAL[9];
   end
endmodule
```

不完全译码的BCD译码器

- 输出Y_0~Y_9为低电平有效
- $ABCD$=0101~1111时，Y_0~Y_9均为任意值，很可能出现两个以上的输出为“0”，这是不允许的！

	A	B	C	D	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
1	1	0	0	0	1	0	1	1	1	1	1	1	1	1
2	0	1	0	0	1	1	0	1	1	1	1	1	1	1
3	1	1	0	0	1	1	1	0	1	1	1	1	1	1
4	0	0	1	0	1	1	1	1	0	1	1	1	1	1
5	1	0	1	0	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	1	1	1	0	1	1	1	1	1	1	1	0	1	1
8	0	0	0	1	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
不用	0	1	0	1	×	×	×	×	×	×	×	×	×	×
	1	1	0	1	×	×	×	×	×	×	×	×	×	×
	0	0	1	1	×	×	×	×	×	×	×	×	×	×
	1	0	1	1	×	×	×	×	×	×	×	×	×	×
	0	1	1	1	×	×	×	×	×	×	×	×	×	×
	1	1	1	1	×	×	×	×	×	×	×	×	×	×

根据真值表推导出逻辑表达式

$$Y_0=A+B+C+D$$
$$=\overline{\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}}$$
$$Y_1=\overline{\overline{A}\,\overline{B}\,\overline{C}\,D}$$
$$\vdots$$
$$Y_8=\overline{\overline{A}\,D}$$
$$Y_9=\overline{A\,D}$$

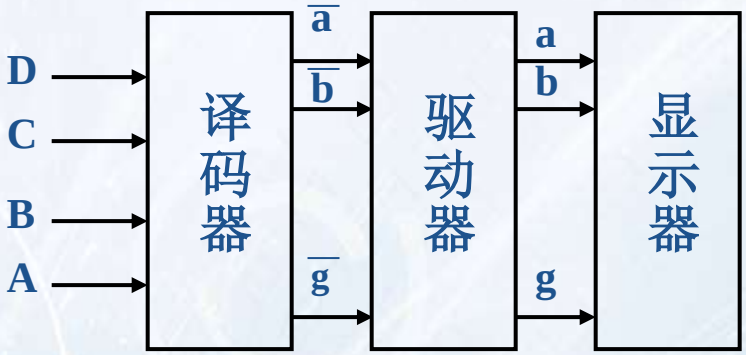
也可以直接用HDL来描述功能（case语句或assign语句）

```
default :Y_SIGNAL = 'bxxxxxxxxxx;
```

3、显示译码器

显示译码器用于驱动数码显示器，是一种将二进制代码表示的数字、文字、符号用人们习惯的形式直观显示出来的电路。

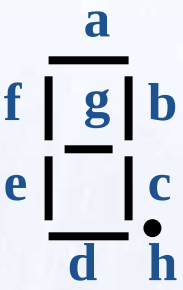
（1）电路结构（8421BCD译码显示电路）



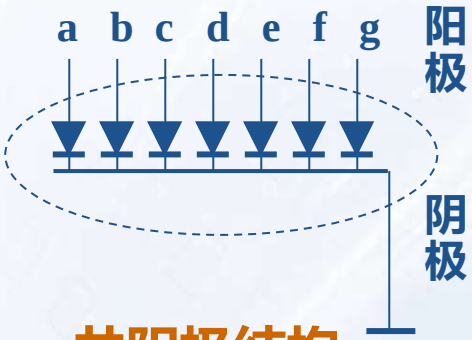
① 显示器件

- 辉光数码管、7段荧光数码管、液晶显示器
- 目前广泛使用的显示数字的器件是7段数码显示器（由7段可发光的线段拼合而成），包括半导体数码显示器和液晶显示器两种。
- 数码显示器人们通常又称为数码管。半导体7段数码管实际上是由7个发光二极管（LED）构成的，因此又称为LED数码管。

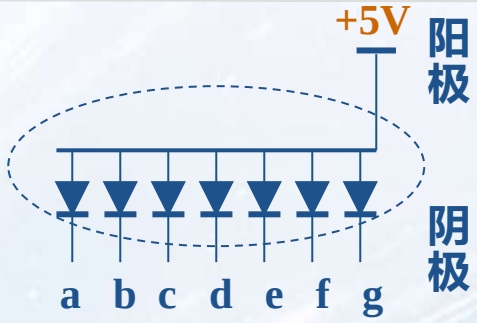
半导体7段数码管



半导体7段数码管



共阴极结构



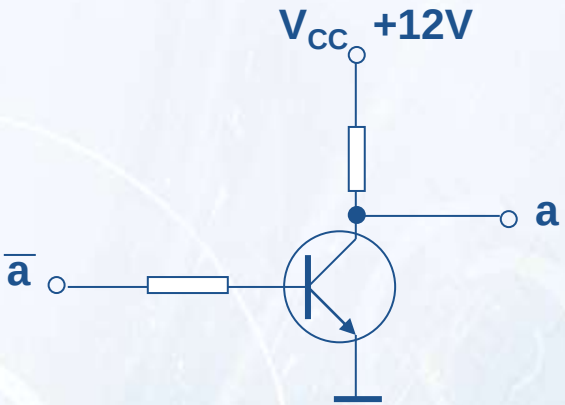
共阳极结构

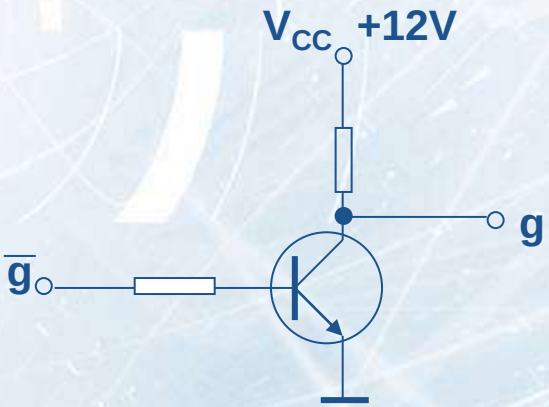
- 若使用**共阴极**LED数码管，则显示译码器的输出应为**高**电平输出有效；若使用**共阳极**LED数码管，则译码器应为**低**电平输出有效。

驱动电路和译码器

② 驱动电路

反相器（或OC门）





③ 译码器（共阳器件）

D C B A	$\overline{a}$ $\overline{b}$ $\overline{c}$ $\overline{d}$ $\overline{e}$ $\overline{f}$ $\overline{g}$	显示数字
0 0 0 0	0 0 0 0 0 0 1	0
0 0 0 1	1 0 0 1 1 1 1	1
0 0 1 0	0 0 1 0 0 1 0	2
0 0 1 1	0 0 0 0 1 1 0	3
0 1 0 0	1 0 0 1 1 0 0	4
0 1 0 1	0 1 0 0 1 0 0	5
0 1 1 0	1 1 0 0 0 0 0	6
0 1 1 1	0 0 0 1 1 1 1	7
1 0 0 0	0 0 0 0 0 0 0	8
1 0 0 1	0 0 0 1 1 0 0	9
1 0 1 0	X X X X X X X	
1 0 1 1	X X X X X X X	
1 1 0 0	X X X X X X X	
1 1 0 1	X X X X X X X	
1 1 1 0	X X X X X X X	
1 1 1 1	X X X X X X X	

集成7段显示译码器（7448）

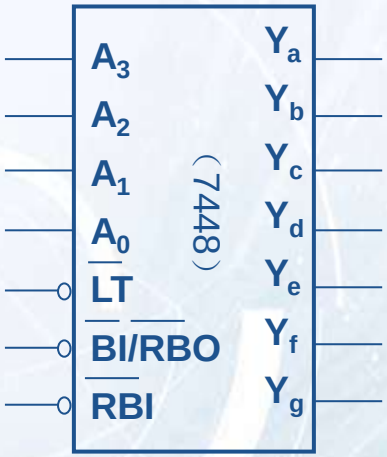
常用显示译码器

74××47(低电平输出有效)

74××49(高电平输出有效)

74××48(高电平输出有效)

7448逻辑符号



灭零输入

消隐输入

7448功能表

数字	输入 LT	输入 RBI	输入 A_3	A_2	A_1	A_0	输出 BI/RBO	输出 Y_a	Y_b	Y_c	Y_d	Y_e	Y_f	Y_g
0	1	1	0	0	0	0	1	1	1	1	1	1	1	0
1	1	X	0	0	0	1	1	0	1	1	0	0	0	0
2	1	X	0	0	1	0	1	1	1	0	1	1	0	1
3	1	X	0	0	1	1	1	1	1	1	1	0	0	1
4	1	X	0	1	0	0	1	0	1	1	0	0	1	1
5	1	X	0	1	0	1	1	1	0	1	1	0	1	1
6	1	X	0	1	1	0	1	0	0	1	1	1	1	1
7	1	X	0	1	1	1	1	1	1	1	0	0	0	0
8	1	X	1	0	0	0	1	1	1	1	1	1	1	1
9	1	X	1	0	0	1	1	1	1	1	0	0	1	1
消隐	X	X	X	X	X	X	0	0	0	0	0	0	0	0
脉冲消隐	1	0	0	0	0	0	0	0	0	0	0	0	0	0
灯测试	0	X	X	X	X	X	1	1	1	1	1	1	1	1

灭灯

灭零

7段显示译码器（7448）的功能

功能

- 7段译码
- 其他功能：灯测试、灭灯、

$\overline{LT}$—灯测试(低电平有效)， $\overline{LT}$ =0时无论A_3~A_0为什么，
Y_a~Y_g全为1，7段全部点亮，检查数码管各段能否正常发光

$\overline{BI}$ / $\overline{RBO}$—灭灯输入/灭零输出(低电平有效)，$\overline{BI}$=0时无论
A_3~A_0为什么， Y_a~Y_g全为0，7段全部熄灭

$\overline{RBI}$—灭零输入(低电平有效)，当$\overline{RBI}$=0，$A_3A_2A_1A_0$=0000
时灯灭； $\overline{RBI}$=1，$A_3A_2A_1A_0$=0000时，显示0。

- 灭零输出/RBO与灭零输入/RBI配合使用，可实现多位数码显示的灭零控制。

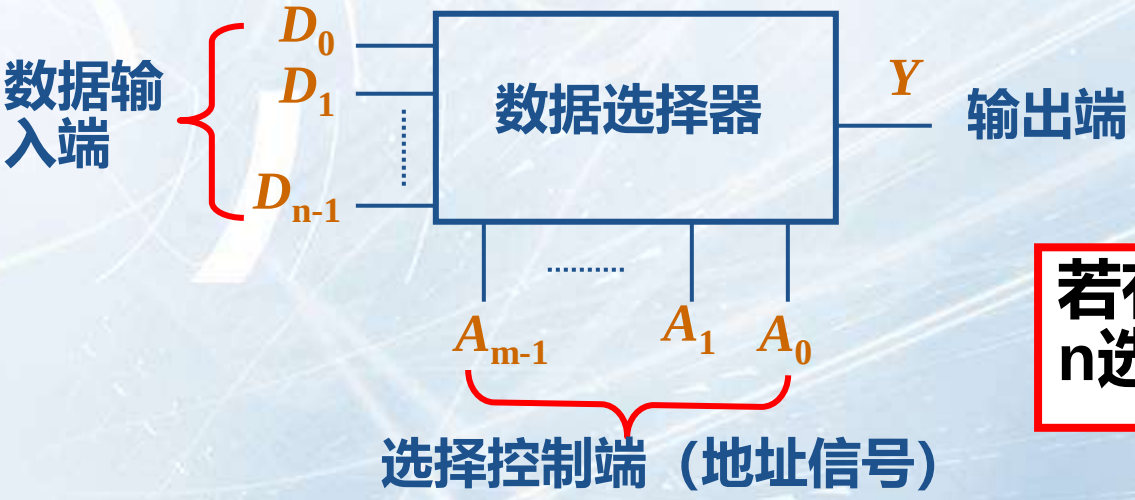
有灭零控制的7位数码显示系统

将$\overline{BI}$/$\overline{RBO}$和$\overline{RBI}$配合使用，可以实现多位数显示时的“无效0消隐”功能。

- 将整数部分的高位/RBO与低位的/RBI相连，小数部分的低位/RBO与高位的/RBI相连，就可以把整数部分不代表数值的高位0和小数部分不代表数值的低位0灭掉。

5.2.4 数据选择器

- 从一组输入数据选出其中需要的一个数据作为输出的过程叫做数据选择，具有数据选择功能的电路称为数据选择器（Data Selector）。
- 数据选择器又称多路开关（Multiplexer，多路器），它是以“与或非”门或以“与或”门为主体的组合逻辑电路。它在选择控制信号的作用下，能从多路平行输入数据中任选一路数据作为输出。
- 常用的集成数据选择器有四2选1（74××157）、双4选1（74××153）、8选1（74××151）及16选1（74××150）数据选择器等。



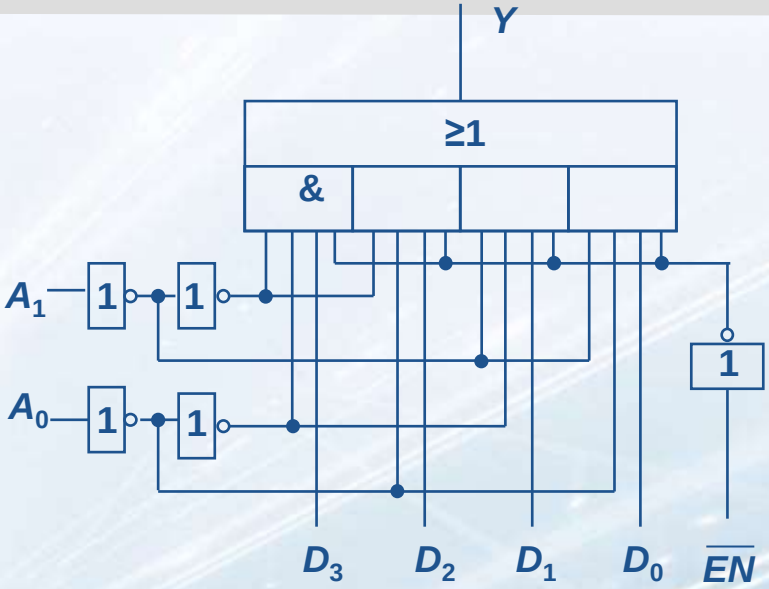
若有n个输入，则称为n选1数据选择器。

1、4选1数据选择器（74153）

- 数据选择器不仅可以用作数据选择器，还可以用作多功能运算电路。

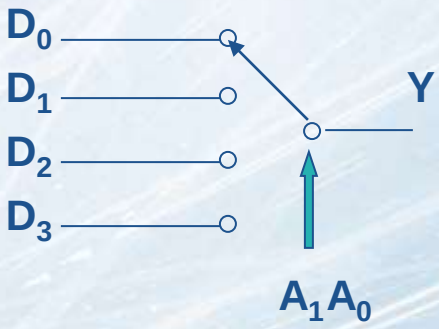
$$Y=\overline{A_1}\,\overline{A_0}D_0+\overline{A_1}A_0D_1+A_1\overline{A_0}D_2+A_1A_0D_3$$
$$(\overline{EN}=0)$$

功能（1）：A_1A_0为控制端——数据选择器（多路开关）

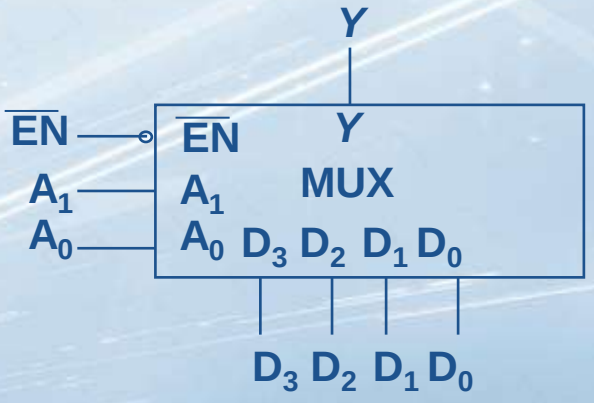


功能表

$\overline{EN}$	A_1	A_0	Y
1	X	X	0
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3



相当于单刀多掷开关



4选1数据选择器的功能（2）

功能（2）：$D_3D_2D_1D_0$为控制端——多功能运算电路（共2^4=16种）

- 通过D_3~D_0取不同的值，从输入变量A_1、A_0的各个最小项中选取某几个最小项输出，实现不同的运算电路

$$Y = D_0\overline{A_1}\,\overline{A_0} + D_1\overline{A_1}A_0 + D_2A_1\overline{A_0} + D_3A_1A_0$$

$D_3D_2D_1D_0$	Y	$D_3D_2D_1D_0$	Y
0000	0	1000	
0001	$\overline{A_1}\,\overline{A_0}$	1001	$\overline{A_1}\,\overline{A_0} + A_1A_0 = \overline{A_1 \oplus A_0}$
0010		1010	
0011		1011	
0100		1100	
0101		1101	
0110	$\overline{A_1}A_0 + A_1\overline{A_0} = A_1 \oplus A_0$	1110	
		1111	1

同或

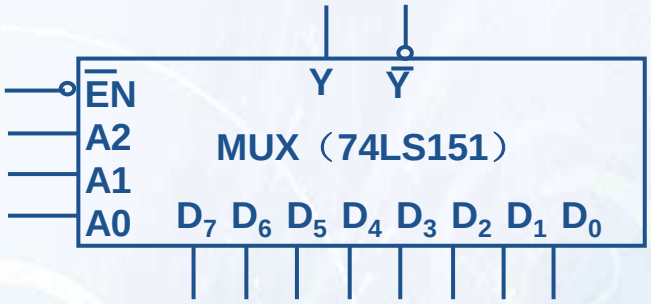
异或

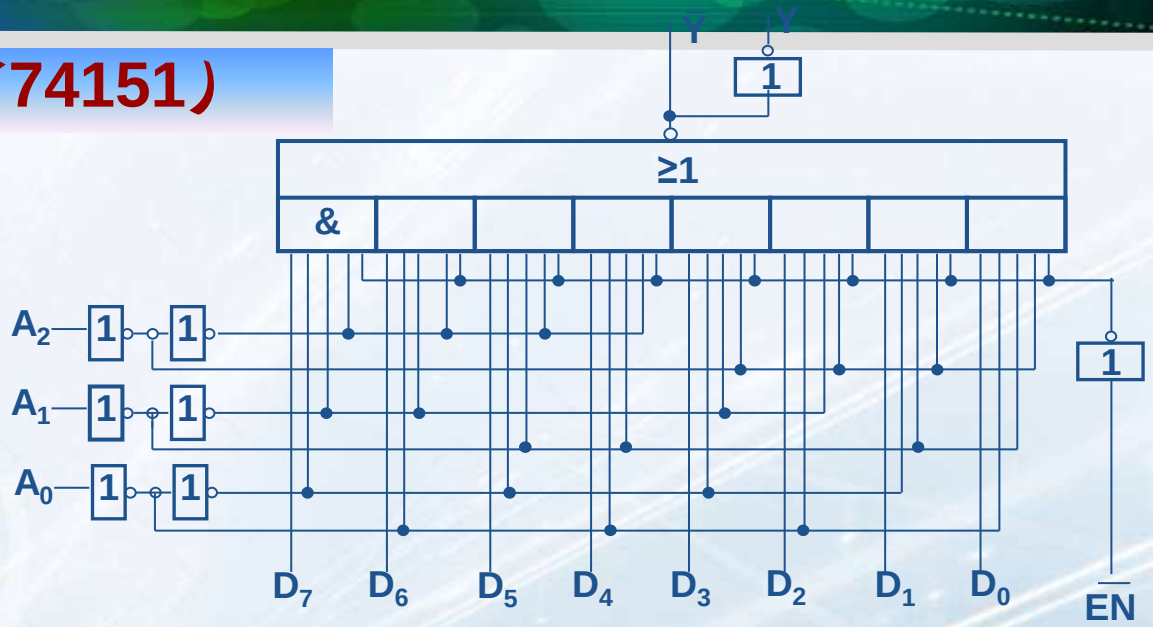
输出为由输入变量A_1、A_0构成的各种最小项的表达式——传统数字电路设计中，可利用数据选择器的运算功能实现任意组合逻辑电路。

2、8选1数据选择器（74151）

2、8选1数据选择器（74151）

❖ 逻辑图与逻辑符号





$$Y = \overline{A_2}\,\overline{A_1}\,\overline{A_0}D_0 + \overline{A_2}\,\overline{A_1}A_0D_1 + \overline{A_2}A_1\overline{A_0}D_2 + \overline{A_2}A_1A_0D_3 + A_2\overline{A_1}\,\overline{A_0}D_4 + A_2\overline{A_1}A_0D_5 + A_2A_1\overline{A_0}D_6 + A_2A_1A_0D_7 \quad (\overline{EN} = 0)$$
$$= m_0D_0 + m_1D_1 + m_2D_2 + m_3D_3 + m_4D_4 + m_5D_5 + m_6D_6 + m_7D_7$$

❖ 功能

① $A_2A_1A_0$控制——

8选1数据选择器（8路开关）

在/EN=0时

8选1数据选择器的功能（2）

功能表（$\overline{EN}$=0）

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

功能 ② D_7 ～ D_0控制——多功能运算电路

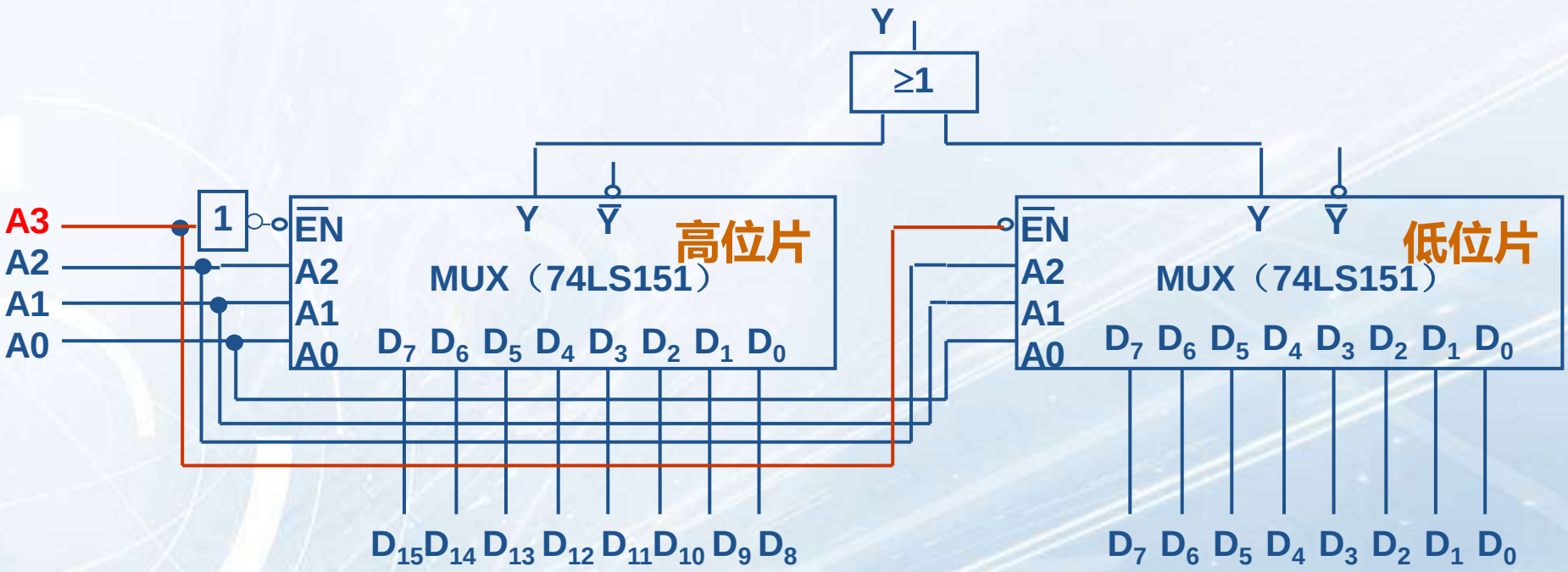
- 通过D_7~D_0取不同的值，从输入变量A_2、A_1、A_0的各个最小项中选取某几个最小项输出，实现不同的运算电路
- 有2^8=256种功能——包含3变量的各种最小项表达式——可实现任意组合逻辑电路的设计。

$$Y = D_0m_0 + D_1m_1 + D_2m_2 + D_3m_3 + D_4m_4 + D_5m_5 + D_6m_6 + D_7m_7$$

当D_7 ～ D_0为0000_0000时，Y=0；
当D_7 ～ D_0为1111_1111时，Y=1；
当D_7 ～ D_0为0000_0001时，$Y=m_0$；
当D_7 ～ D_0为1010_0101时，$Y=m_7+m_5+m_2+m_0$。

数据选择器（74151）的扩展

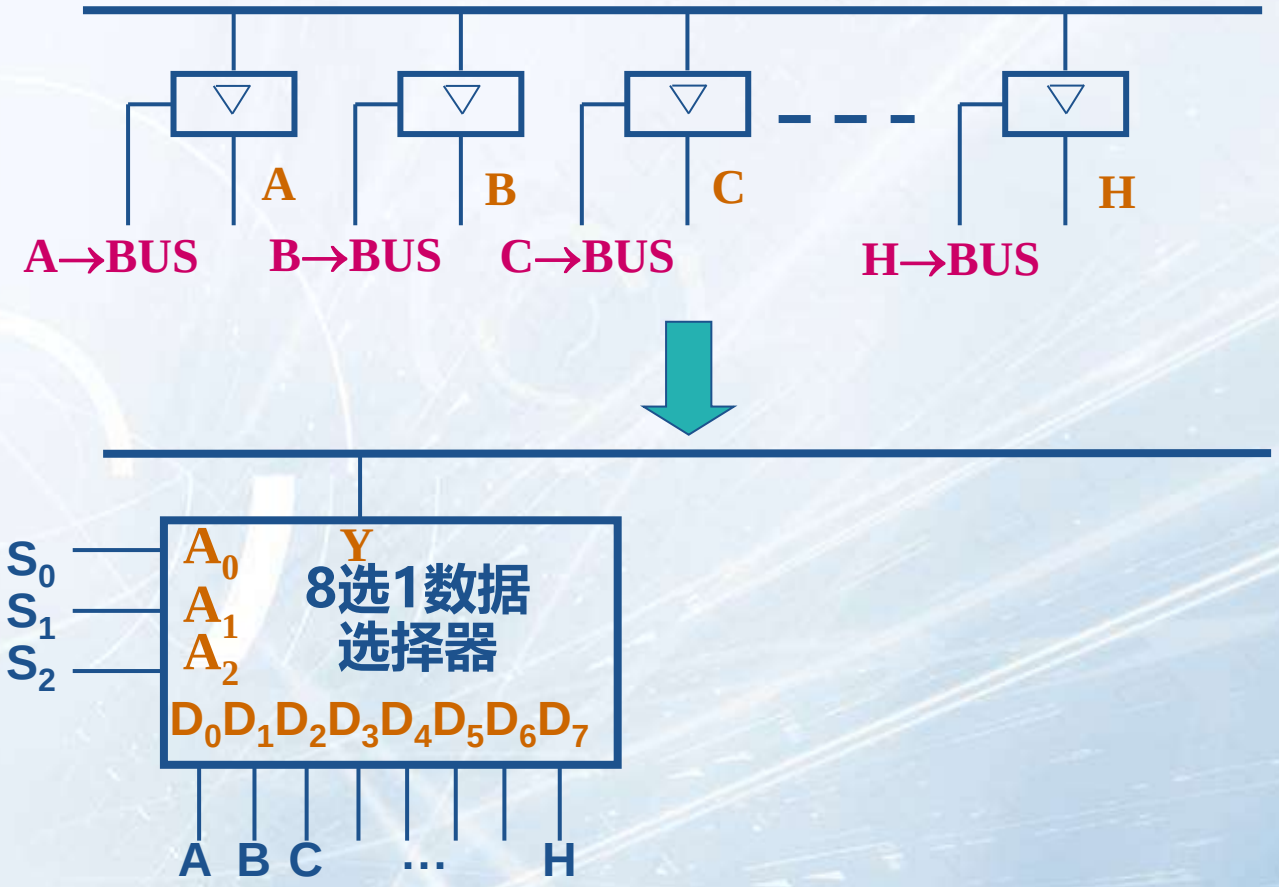
应用使能端将2片74151（8选1）扩展为16选1数据选择器



- A_3=0时，低位片工作，从输入D_7~D_0中选择1个输出；
- A_3=1时，高位片工作，从输入D_{15}~D_8中选择1个输出。

3、数据选择器的应用（1/2）

- 数据选择器除选择功能外，还有其他的用途
  - 代替三态门，实现总线发送控制



$S_2S_1S_0$	Y
0 0 0	A
0 0 1	B
0 1 0	C
0 1 1	D
1 0 0	E
1 0 1	F
1 1 0	G
1 1 1	H

数据选择器的应用（2/2）

- 函数发生器——用数据选择器实现任意组合逻辑函数

  数据选择器可以看成是用N个控制端D_{N-1}~D_0选择2^N个最小项的某几个组成“与-或”表达式。选择某些控制信号D_i为“1”，就是选中这些最小项组成逻辑函数。

- 如果给定一个组合逻辑函数，如何用数据选择器实现？
  - 利用互补律，将组合逻辑函数变换为最小项之和的标准形式；
  - 根据该逻辑函数有几个输入变量，确定数据选择器的地址输入为几位，将逻辑函数的输入变量作为数据选择器的地址输入；
  - 写出数据选择器的输出表达式；
  - 比较逻辑函数的标准形式与数据选择器的输出表达式，推导出数据选择器的D_i哪些为1，哪些为0；
  - 画出电路图。

【例5.7 】利用数据选择器实现逻辑函数

【例5.7 】利用数据选择器实现逻辑函数

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

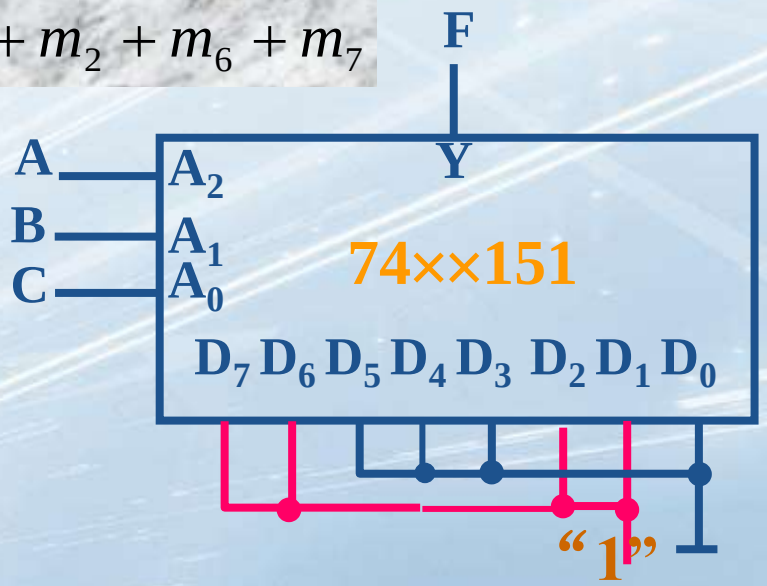
解：使用8选1数据选择器74151

将逻辑函数的输入变量作为数据选择器的地址输入。

首先将组合逻辑函数变换为最小项之和的标准形式：

$$F=\overline{A}\,\overline{B}C+\overline{A}B\overline{C}+AB(\overline{C}+C)=m_1+m_2+m_6+m_7$$

74151输出 $Y=D_0m_0+D_1m_1+D_2m_2+D_3m_3+D_4m_4+D_5m_5+D_6m_6+D_7m_7$



比较F和Y，得：

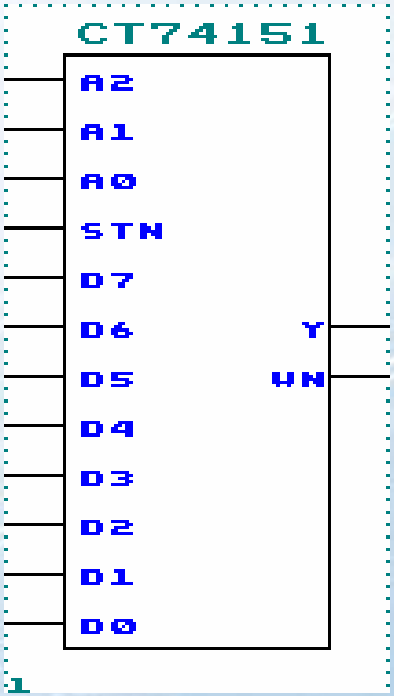
D_0=0，D_1=1，D_2=1，D_3=0，

D_4=0，D_5=0，D_6=1，D_7=1

4、数据选择器（74151）的HDL设计

- 信号定义
  - D7～D0：8位数据输入端
  - A2～A0：地址输入端
  - STN：使能控制端（低电平有效）
  - Y：同相数据输出端
  - WN：反相数据输出端，即WN是Y的反相输出

74151的元件符号：



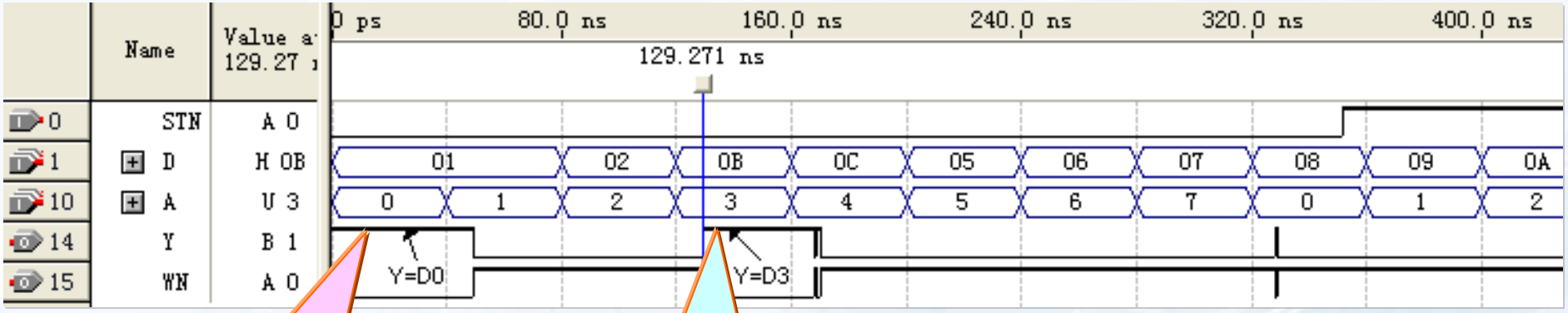
注意：教材P126有错，A3~A0应为A2~A0。

74151的Verilog HDL源程序

- HDL设计
  采用if-else语句和case语句描述

```
module CT74151(A2,A1,A0,STN,D7,
      D6,D5,D4,D3,D2,D1,D0,Y,WN);
   input      A2,A1,A0,STN;
   input      D7,D6,D5,D4,D3,
              D2,D1,D0;
   output     Y,WN;
   reg        Y,WN;
   always
     begin
        if (STN == 0)
          begin
            case ({A2,A1,A0})
              3 'b000 : Y = D0;
              3 'b001 : Y = D1;
              3'b010 : Y = D2;
              3'b011 : Y = D3;
              3'b100 : Y = D4;
              3 'b101 : Y = D5;
              3 'b110 : Y = D6;
              3 'b111 : Y = D7;
            endcase
          end
        else Y = 1'b0;
        WN = ~Y;
      end
endmodule
```

CT74151.v的仿真波形



A2 A1 A0=000 时 Y = D0

A2 A1 A0=011 时 Y = D3

- 为便于编辑输入信号，可以将一组同名的节点如D7,D5,D4,D3,D2,D1,D0表示为一个总线信号
  - 在波形编辑器中将信号按D7,D5,D4,D3,D2,D1,D0的顺序调整位置(D7 在最上面）；
  - 选中这些信号，单击右键，选中“Grouping>Group”命令，在“Group”窗口中为这组信号命名(如D），设置进制。
- 也可以用“Grouping>Ungroup”命令将总线信号展开为单个的节点

用数据选择器设计组合逻辑电路的方法

（1）逻辑抽象

- 确定输入、输出变量；
- 定义逻辑状态的含义；
- 列出真值表。

（2）写出逻辑函数表达式

- 根据真值表写出逻辑函数的标准表达式

（3）选定数据选择器器件

- 若函数有M个输入变量，选择的数据选择器有n位地址输入，则应取M≤n+1，以M=n+1时器件的利用最充分——可以少用一个地址输入
- 例如有4个输入变量，可以选择具有3位地址输入的数据选择器（8选1数据选择器），3个输入变量接数据选择器的3位地址输入端，1个输入变量接数据输入端

（4）确定输入变量与地址输入端和数据输入端的对应关系

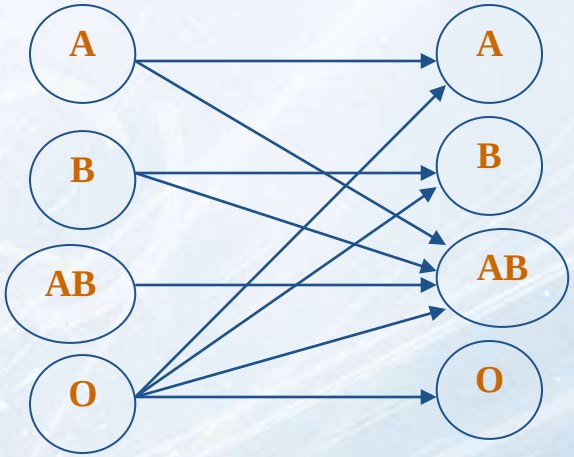
- 将逻辑函数式化为最小项之和的形式，并与数据选择器输出的逻辑函数式对照比较，确定输入变量与地址输入端和数据输入端的对应关系

（5）画出逻辑电路图

- 根据（4）进行连线，数据选择器的输出端即所设计的逻辑函数

数据选择器的应用实例

【例5.8】人的血型有A、B、AB、O 等4种。输血时输血者的血型与受血者的血型必须符合下图中用箭头指示的授受关系。试用数据选择器设计一个逻辑电路，判断输血者与受血者的血型是否符合上述规定。

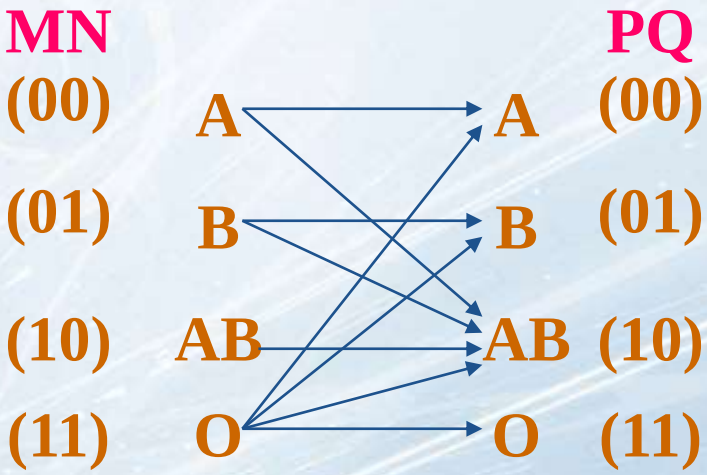


- **提示：可以用两个逻辑变量的4种取值表示输血者的血型；用另外两个逻辑变量的4种取值表示受血者的血型。**

设计思路

解：

- 确定输入、输出变量，定义逻辑状态的含义
- 输入变量：以*MN*的4种状态组合表示输血者的4种血型，并以*PQ*的4种状态组合表示受血者的4种血型。
- 输出变量：用*Z*表示判断结果，*Z*=0表示符合题目要求， *Z*=1表示不符合要求。



输入信号状态定义

真值表和输出的逻辑函数表达式

列出表示Z与 M、N、P、Q之间逻辑关系的真值表

M	N	P	Q	Z	M	N	P	Q	Z
0	0	0	0	0	1	0	0	0	1
0	0	0	1	1	1	0	0	1	1
0	0	1	0	0	1	0	1	0	0
0	0	1	1	1	1	0	1	1	1
0	1	0	0	1	1	1	0	0	0
0	1	0	1	0	1	1	0	1	0
0	1	1	0	0	1	1	1	0	0
0	1	1	1	1	1	1	1	1	0

$$Z=\overline{M}\,\overline{N}\,\overline{P}Q+\overline{M}\,\overline{N}PQ+\overline{M}N\,\overline{P}\,\overline{Q}+\overline{M}NPQ+M\,\overline{N}\,\overline{P}\,\overline{Q}+M\,\overline{N}\,\overline{P}Q+M\,\overline{N}PQ$$

逻辑函数与数据选择器的输出对照比较

- 取8选1数据选择器74xx151实现上式的逻辑函数
- 已知8选1数据选择器的输出为

$$Y=\overline{A_2}\,\overline{A_1}\,\overline{A_0}\bullet D_0+\overline{A_2}\,\overline{A_1}A_0\bullet D_1+\overline{A_2}A_1\overline{A_0}\bullet D_2+\overline{A_2}A_1A_0\bullet D_3+$$
$$A_2\overline{A_1}\,\overline{A_0}\bullet D_4+A_2\overline{A_1}A_0\bullet D_5+A_2A_1\overline{A_0}\bullet D_6+A_2A_1A_0\bullet D_7$$

- 将*Z*变换成与*Y*对应的形式

$$Z=\overline{M}\,\overline{N}\,\overline{P}\bullet Q+\overline{M}\,\overline{N}P\bullet Q+\overline{M}N\overline{P}\bullet\overline{Q}+\overline{M}NP\bullet Q+M\overline{N}\,\overline{P}\bullet 1+$$
$$M\overline{N}P\bullet Q+MN\overline{P}\bullet 0+MNP\bullet 0$$

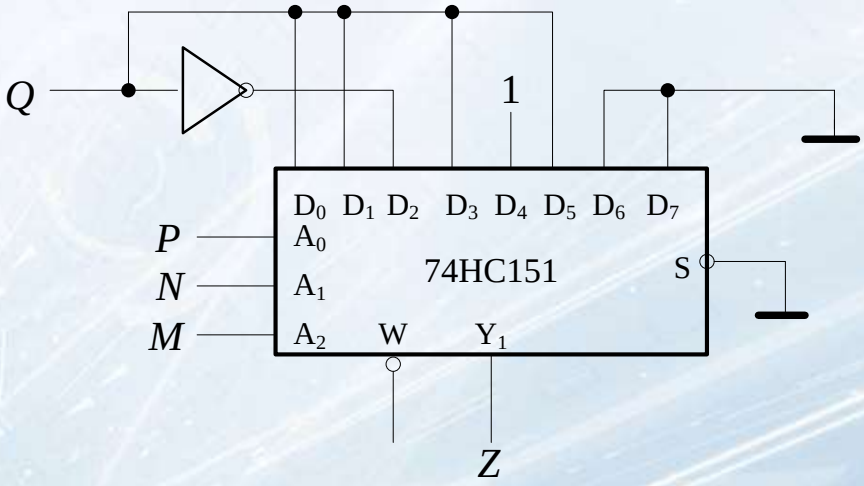
- **3个输入变量*M*、*N*、*P*接数据选择器的3位地址输入A2、A1、A0，1个输入变量Q接数据输入。**

电路连接图

令数据选择器的输入为

$$A_2 = M、\ A_1 = N、\ A_0 = P、\ D_0 = D_1 = D_3 = D_5 = Q$$

$$、\ D_2 = \overline{Q}、\ D_4 = 1、\ D_6 = D_7 = 0$$



思考：如果本例采用HDL实现，应该怎样描述？哪种方法更简单？

5.2.5 数值比较器

- 数值比较器是一种关系运算电路，它可以对两个二进制数或二-十进制编码的数进行比较，得出大于、小于和相等的结果。
- 分为“等值”比较器和“量值”比较器，“等值”比较器只检验两个数是否相等；“量值”比较器不但检验两个数是否相等，而且还要检验两个数中哪个为大。

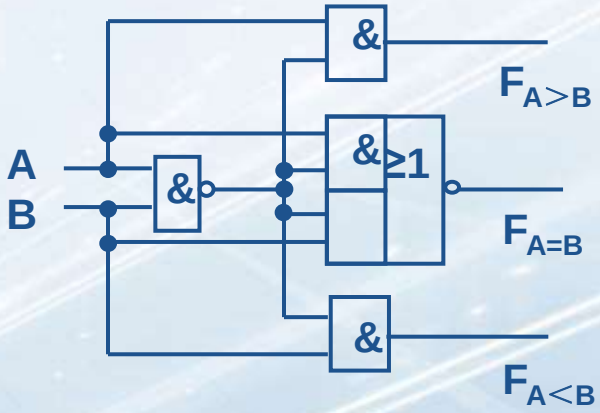
1、1位数值比较器

用来比较两个一位二进制数大小的电路。

$$F_{A>B} = A\overline{AB} = A(\overline{A}+\overline{B}) = A\overline{B}$$

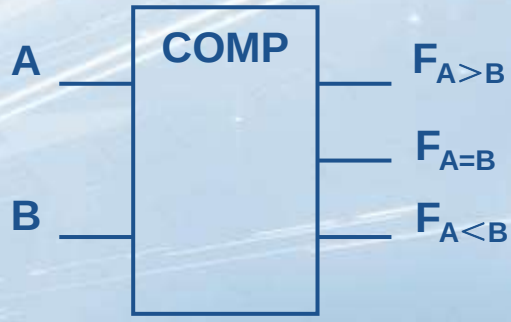
$$F_{A<B} = B\overline{AB} = B(\overline{A}+\overline{B}) = \overline{A}B$$

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



真值表

A	B	$F_{A>B}$	$F_{A=B}$	$F_{A<B}$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

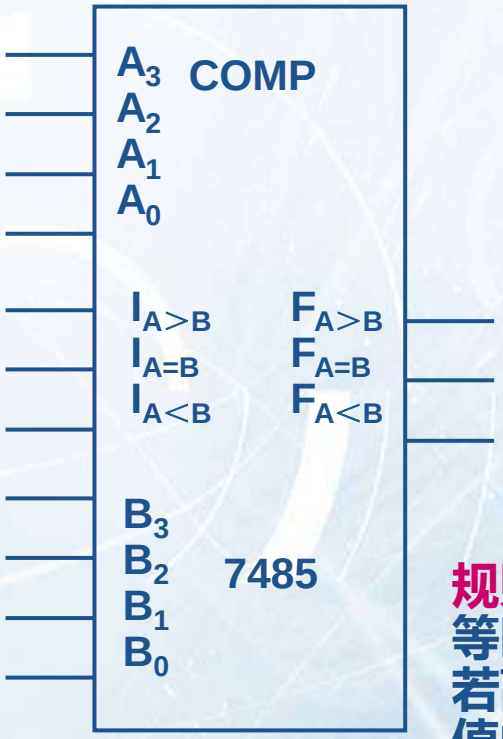


4位数值比较器（7485）

2、4位数值比较器（7485）

用来比较两个4位二进制数大小的电路。

（1）逻辑符号



（2）功能表

A3 B3	A2 B2	A1 B1	A0 B0	$I_{A>B}$ $I_{A=B}$ $I_{A<B}$	$F_{A>B}$ $F_{A=B}$ $F_{A<B}$
A3>B3	X	X	X	X X X	1 0 0
A3<B3	X	X	X	X X X	0 0 1
A3=B3	A2>B2	X	X	X X X	1 0 0
A3=B3	A2<B2	X	X	X X X	0 0 1
A3=B3	A2=B2	A1>B1	X	X X X	1 0 0
A3=B3	A2=B2	A1<B1	X	X X X	0 0 1
A3=B3	A2=B2	A1=B1	A0>B0	X X X	1 0 0
A3=B3	A2=B2	A1=B1	A0<B0	X X X	0 0 1
A3=B3	**A2=B2**	**A1=B1**	**A0=B0**	**a b c**	**a b c**

级联输入端，用于芯片的扩展

规则：从高位开始比较，高位不等时，数值的大小由高位决定；若高位相等，则再比较低位，数值的大小由低位比较结果决定。

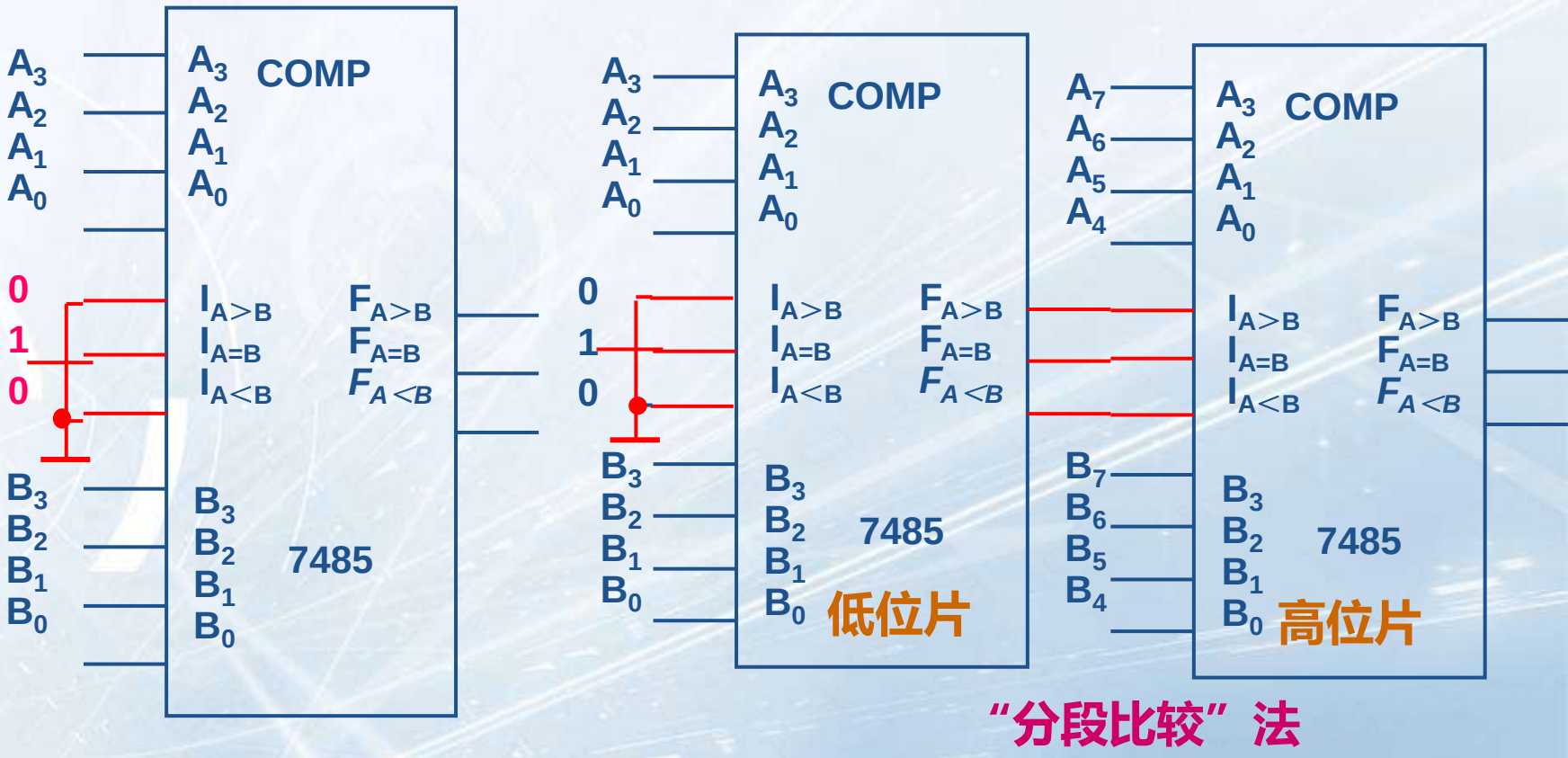
若$A_3>B_3$ 则A>B；
若$A_3<B_3$ 则A<B；
若$A_3=B_3$ 则再比较低位

7485的使用与扩展方法

（3）使用与扩展方法

①单片使用——4位数值比较器

② 2片扩展——8位数值比较器



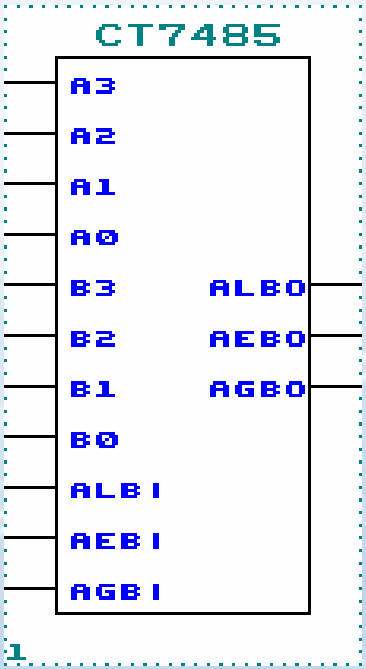
2片扩展——“分段比较”法

- 低位片和高位片并行工作，每片的比较仍是由高位到低位逐位进行
  - 若高4位数不相等，则由两个高4位数A7~A4与B7~B4的大小决定A和B的大小。

- 若高4位分别相等，则由两个低4位数A3~A0与B3~B0的大小决定A和B的大小：若A3~A0>B3~B0，则低位片的输出$F_{A>B}$、$F_{A=B}$、$F_{A<B}$为100，即高位片的级联输入$I_{A>B}$、$I_{A=B}$、$I_{A<B}$为100，由功能表的最后一行可以得出，高位片的输出$F_{A>B}$、$F_{A=B}$、$F_{A<B}$也为100，即A>B；同理，若A3~A0<B3~B0，则可推出A<B；若A3~A0=B3~B0，则可推出A=B。

数值比较器（7485）的HDL设计

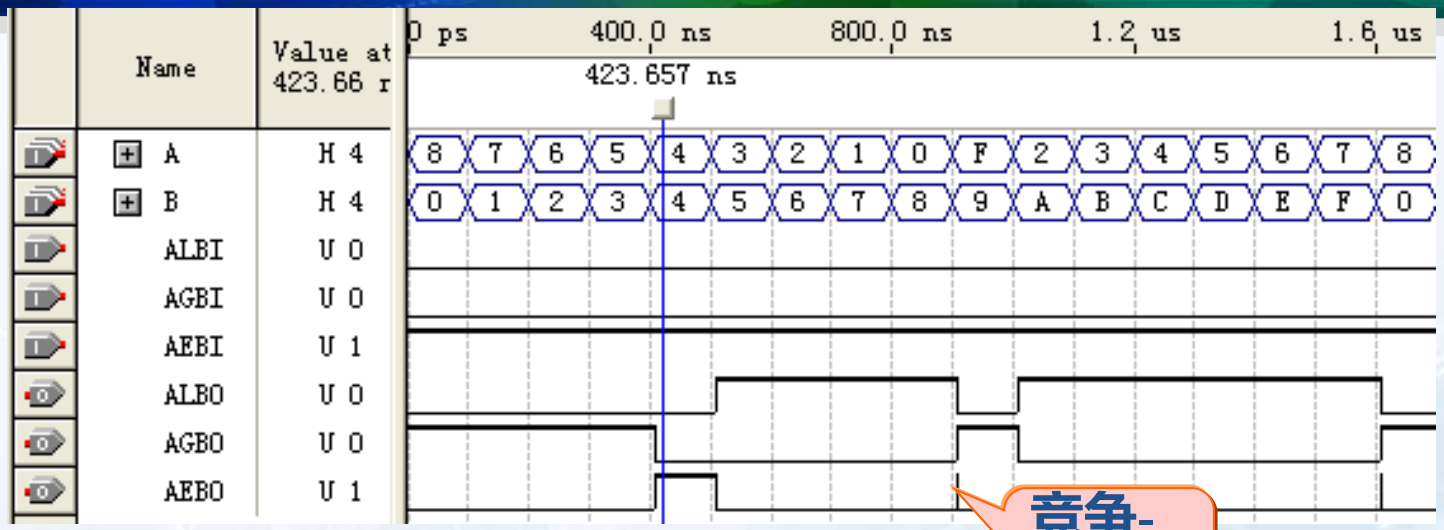
- 可以方便地用HDL设计多位数值比较器，而不必用扩展的方法
- 采用if-else语句
- 信号定义
  - A3～A0和B3～B0：两个4位二进制数输入信号；
  - ALBI（即$I_{A<B}$）：A小于B输入信号；
  - AEBI（即$I_{A=B}$）：A等于B输入信号；
  - AGBI（即$I_{A>B}$）：A大于B输入信号；
  - ALBO（即$F_{A<B}$）：A小于B输出信号；
  - AEBO（即$F_{A=B}$）：A等于B输出信号；
  - AGBO（即$F_{A>B}$）：A大于B输出信号。



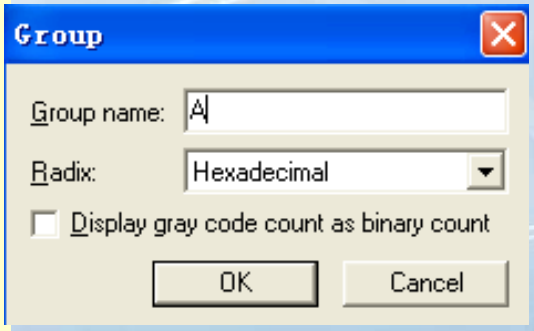
7485的Verilog HDL源程序

```
module  CT7485(A3,A2,A1,A0,B3,B2,B1,B0,ALBI,AEBI,
          AGBI,ALBO,AEBO,AGBO);
    input          A3,A2,A1,A0,B3,B2,B1,B0,ALBI,AEBI,AGBI;
    output         ALBO,AEBO,AGBO;
    reg            ALBO,AEBO,AGBO;
    wire[3:0]      A_SIGNAL,B_SIGNAL;
    assign         A_SIGNAL = {A3,A2,A1,A0}; //拼接成4位wire型向量
    assign         B_SIGNAL = {B3,B2,B1,B0}; //拼接成4位wire型向量
    always
       begin
          if (A_SIGNAL > B_SIGNAL)
            begin  ALBO = 0; AEBO = 0; AGBO = 1;end
          else if (A_SIGNAL < B_SIGNAL)
            begin  ALBO = 1; AEBO = 0; AGBO = 0;end
          else // if(A_SIGNAL == B_SIGNAL)可省略
            begin  ALBO = ALBI; AEBO = AEBI; AGBO = AGBI;end
       end
endmodule
```

CT7485.V的时序仿真波形



- 为便于编辑输入波形，对于成组的信号（如A3、A2、A1、A0）可以先将其组合为一个总线信号，再赋值
  - 先在波形编辑器中将信号A3、A2、A1、A0按从上至下的顺序排列（不能弄反！）；
  - 再选中这4个信号，单击右键，在快捷菜单中选择Group；
  - 在Group窗口中键入Group name为“A”，选择Radix为“Hexadecimal”，单击OK。

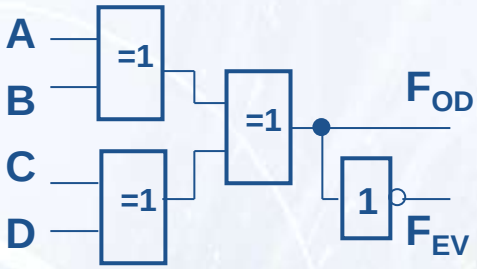


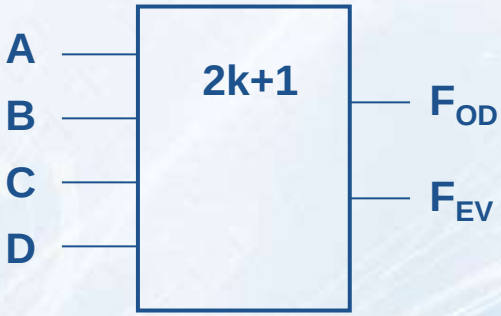
5.2.6 奇偶校验器

- 在数据传输过程中由于信道的干扰，或者在数据记录过程中由于外界的干扰，可能导致传输来的数据或记录的数据与原始数据不完全相同，即数据中的某一位或某几位出现了差错。
- 通过检测原始数据和接收数据中包含“1”的个数是奇数还是偶数，可以初步判断接收到的数据是否有错——如果原始数据包含奇数个“1”，而接收数据包含偶数个“1”，则一定有错！
- **奇偶校验就是检测数据中包含“1”的个数是奇数还是偶数。**
- **奇偶校验器是采用“奇偶校验”方法来检查数据传输后和数码记录中是否存在错误的一种逻辑电路。**
- 广泛用于计算机的内存储器以及磁盘和磁带之类的外部设备中；在通信中也常用到。

4位奇偶校验器

1、4位奇偶校验器





判奇输出端 $F_{OD} = A \oplus B \oplus C \oplus D$

判偶输出端 $F_{EV} = \overline{A \oplus B \oplus C \oplus D}$

真值表

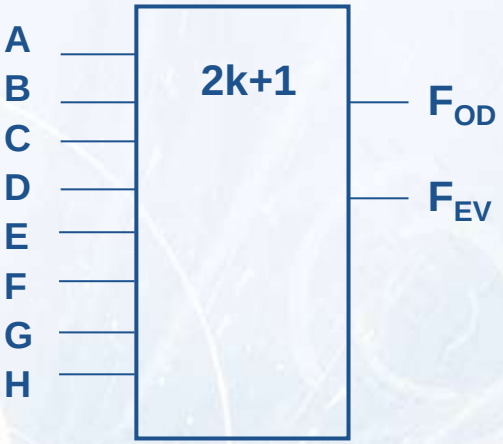
A B C D	F_{OD}	F_{EV}
0 0 0 0	0	1
0 0 0 1	1	0
0 0 1 0	1	0
0 0 1 1	0	1
0 1 0 0	1	0
0 1 0 1	0	1
0 1 1 0	0	1
0 1 1 1	1	0
1 0 0 0	1	0
1 0 0 1	0	1
1 0 1 0	0	1
1 0 1 1	1	0
1 1 0 0	0	1
1 1 0 1	1	0
1 1 1 0	1	0
1 1 1 1	0	1

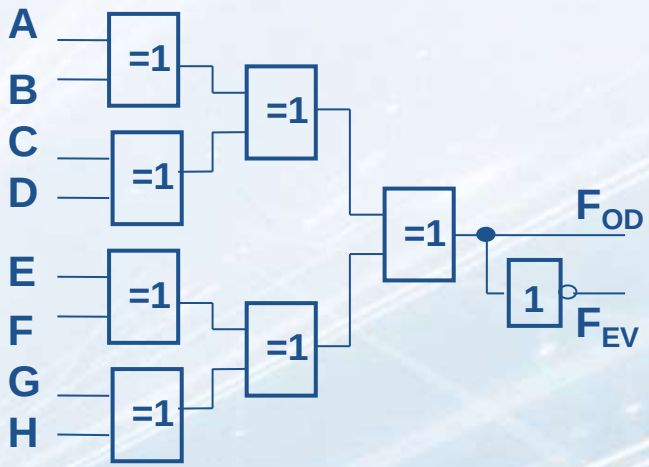
“模2加”运算

- 奇偶校验器一般由**异或门**构成，异或运算也称为“**模2加**”运算——只考虑两个二进制数相加后的算术和，而不考虑它们的进位。
- 当两个（1位）二进制数“模2加”时，若和为1，表示两个数中有奇数个“1”；若和为0，表示有偶数个“1”。
- 同理，当n个（1位）二进制数“模2加”时，若**和为1**，表示n个数中有**奇数个“1”**；若和为0，表示有偶数个“1”。
- 如果数据的位数较多，则可用塔状级联的异或门构成奇偶校验器。

8位奇偶校验器

2、8位奇偶校验器





$$F_{OD} = A \oplus B \oplus C \oplus D \oplus E \oplus F \oplus G \oplus H$$

$$F_{EV} = \overline{A \oplus B \oplus C \oplus D \oplus E \oplus F \oplus G \oplus H}$$

集成8位奇偶校验器/产生器（74180）

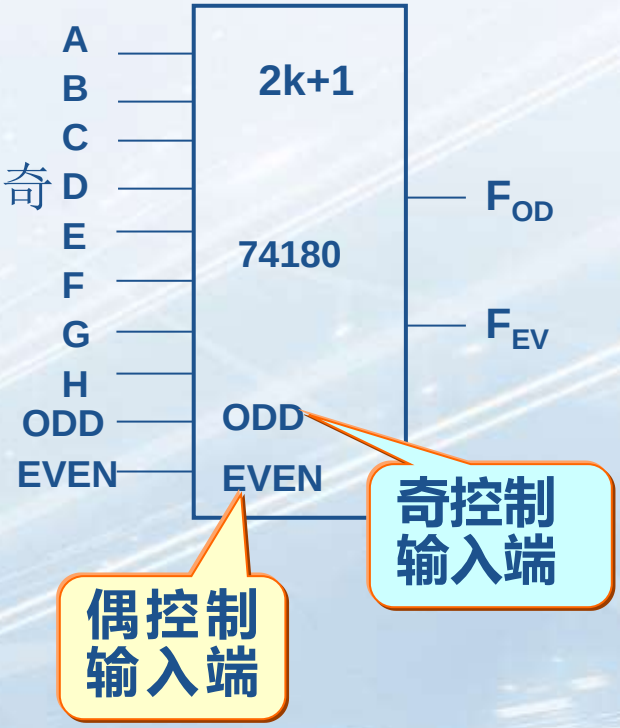
3、集成8位奇偶校验器/产生器（74180）

- 奇偶校验器还有奇偶产生的功能，通常称为奇偶校验器/产生器。
- 常用的集成奇偶校验器/产生器有74xx180、74xx280等。它们还有偶控制输入端EVEN 和奇控制输入端ODD

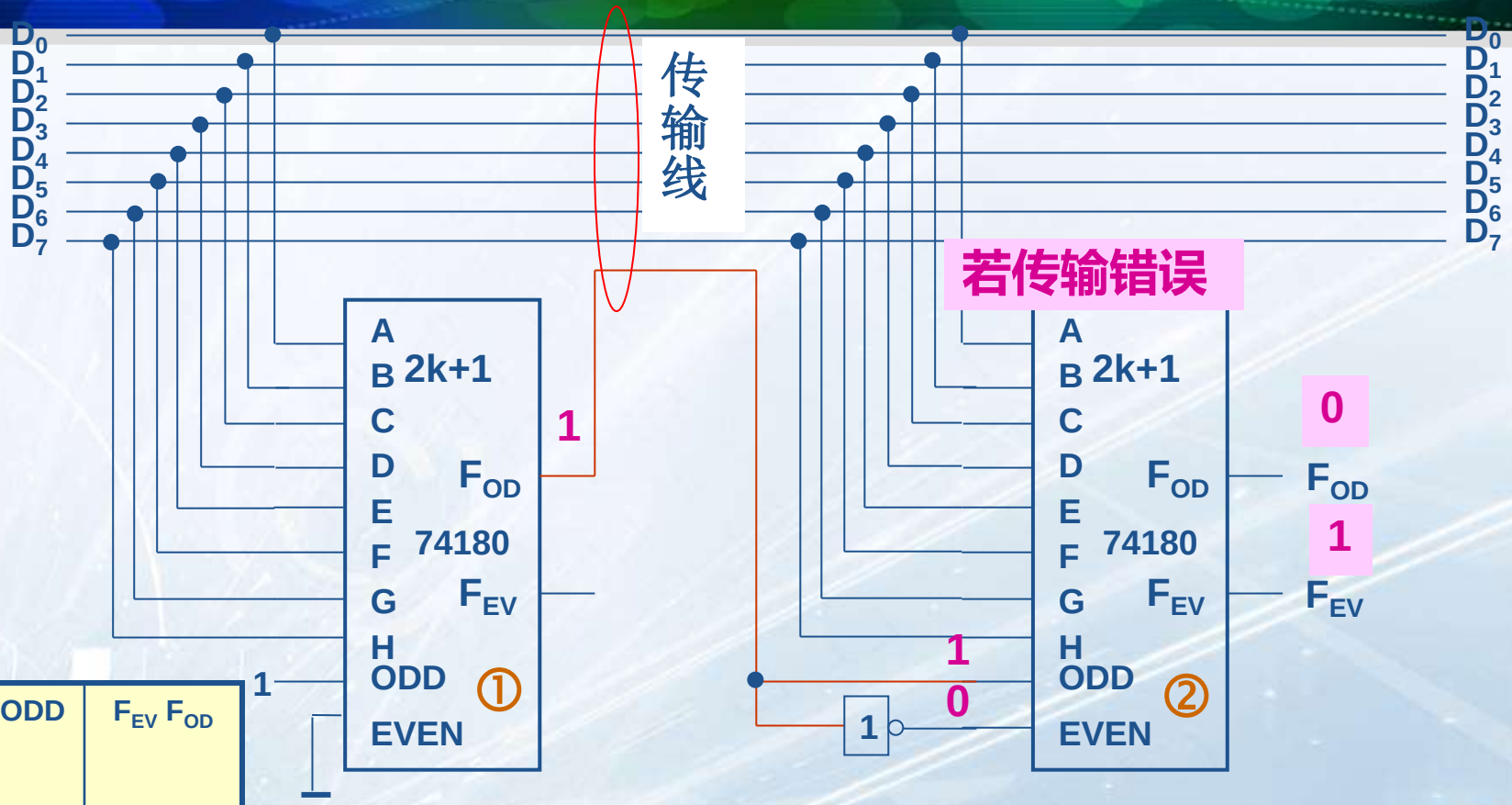
74xx180功能表

A~H中1的个数	EVEN	ODD	F_{EV} F_{OD}
偶数	1	0	1 0
奇数	1	0	0 1
偶数	0	1	0 1
奇数	0	1	1 0
X	1	1	0 0
X	0	0	1 1

EVEN和ODD同时为1或0没有意义



奇偶校验系统



A~H中1的个数	EVEN	ODD	F_{EV} F_{OD}
偶数	1	0	1 0
奇数	1	0	0 1
偶数	0	1	0 1
奇数	0	1	1 0
X	1	1	0 0
X	0	0	1 1

无论D0~D7中1的个数为偶数还是奇数，加上片①的F_{OD}后，组成9位数据中1的个数一定是奇数

若输出F_{OD}=1，F_{EV}=0，表示数据传输正确；若F_{OD}=0，F_{EV}=1，表示数据传输有差错

奇偶校验系统的工作原理

- 若D_0~D_7中1的个数为偶数，则片①的F_{OD}=1；若为奇数，则片①的F_{OD}=0
- 无论D_0~D_7中1的个数为偶数还是奇数，加上片①的F_{OD}后，组成9位数据中1的个数一定是**奇数**——片①称为**奇产生器**
- 若原数据D_0~D_7中有**偶数个1**，则片①的F_{OD}=1，则片② 的ODD=1， EVEN=0，传输无误时，片② 的F_{OD}=1 ，F_{EV}=0，表示数据传输正确；若原数据D_0~D_7中有**奇数个1**，则片①的F_{OD}=0，则片② 的ODD=0， EVEN=1，在传输无误时，片② 的F_{OD}=1 ，F_{EV}=0，表示数据传输正确。
- 若传输过程中有一个数据位发生差错，则9位数据中1的个数由奇数变为偶数，片② 的F_{OD}=0 ，F_{EV}=1，表示数据传输有差错。

片①的
ODD=1，
EVEN=0

A~H中1的个数	EVEN	ODD	F_{EV} F_{OD}
偶数	1	0	1 0
奇数	1	0	0 1
偶数	0	1	0 1
奇数	0	1	1 0
X	1	1	0 0
X	0	0	1 1

奇偶校验器（74180）的HDL设计

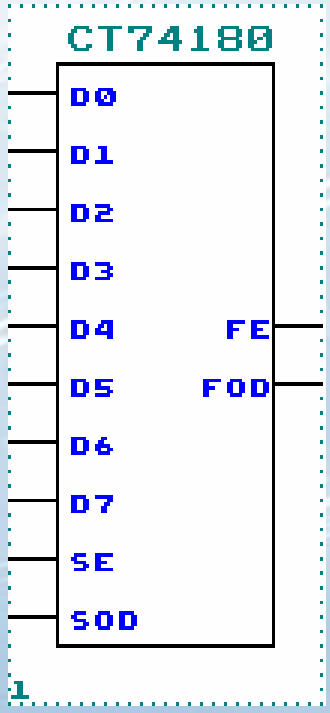
❖信号定义

- D0～D7 （即A～H ）：8位数据输入端；
- SE（即EVEN）和SOD（即ODD）：两个控制信号输入端；
- FE（即F_{EV}）：偶校验输出端，FOD（即F_{OD}）：奇校验输出端。

真值表

D0～D7中1的个数	SE	SOD	FE FOD
偶数	1	0	1 0
奇数	1	0	0 1
偶数	0	1	0 1
奇数	0	1	1 0
X	1	1	0 0
X	0	0	1 1

CT74180的元件符号



分析： 输入SE和SOD有4种取值组合，在不同的取值组合下，输出FE和FOD取不同的值——适合用case语句描述；
当SE、SOD为10和01时，根据D0~D7中1的个数为偶数或奇数， FE和FOD又取不同的值——适合用if语句描述。

74180的Verilog HDL源程序（1/2）

根据真值表，采用case语句和if语句直接描述其功能：

```
module CT74180(D0,D1, D2, D3,D4,D5, D6,D7,
               SE,SOD,FE,FOD);        // D0～D7 对应A～H
    input     D0,D1,D2,D3,D4,
              D5,D6,D7,SE,SOD;
    output    FE,FOD;
    reg       FE,FOD;
    reg       FE_SIGNAL;
    wire[7:0]  A_SIGNAL;
    assign     A_SIGNAL = {D0,D1,D2,D3,D4,D5,D6,D7};
```

关键：怎样描述一组输入数据中为1的个数是奇数或偶数？

74180的Verilog HDL源程序（2/2）

```
always @(A_SIGNAL or SE or SOD)
  begin
    FE_SIGNAL = ^A_SIGNAL; //异或(缩减运算)
    case ({SE,SOD})
      2'b00 : begin FE=1'b1; FOD=1'b1;end
      2‘b01 : if (FE_SIGNAL==1’b0) //有偶数个“1”时
                 begin FE=1'b0; FOD=1'b1;end
              else   begin FE=1‘b1; FOD=1’b0;end //有奇数个“1”时
      2‘b10 : if (FE_SIGNAL==1’b0) //有偶数个“1”时
                 begin FE=1'b1; FOD=1'b0;end
              else   begin FE=1'b0; FOD=1'b1;end //有奇数个“1”时
      2'b11 : begin FE=1'b0; FOD=1'b0;end
    endcase
  end
endmodule
```

n个数异或时，若结果为1，表示其中有奇数个“1”；若结果为0，表示有偶数个“1”。

偶数	0	1	0 1
奇数	0	1	1 0
X	1	1	0 0
X	0	0	1 1

100

本章小结（1/8）

1、组合逻辑电路与时序逻辑电路的区别

- 没有反馈电路和存储电路
- 是一种无记忆电路——任一时刻的输出信号仅取决于该时刻的输入信号，而与信号作用前电路原来所处的状态无关。

2、组合逻辑电路的表述方法

⑩ **逻辑函数表达式、真值表、卡诺图、逻辑图及波形图**

3、组合逻辑电路的分析方法

- **根据给定的逻辑电路，通过分析确定其逻辑功能。**
  - 根据逻辑图写出逻辑函数表达式；利用公式法进行化简，得到最简表达式；写出真值表；根据真值表说明电路的逻辑功能。

本章小结（2/8）

4、组合逻辑电路的设计方法

- 根据给定的设计要求设计出相应的组合逻辑电路。
- 组合逻辑电路的手工设计方法（了解即可）
  - 逻辑抽象（列出真值表）；写出逻辑函数表达式；逻辑化简；绘逻辑图
- 组合逻辑电路的自动设计方法

（1）逻辑抽象→HDL编程（系统级描述，根据逻辑功能定义直接用case语句或if-else语句描述；或者写出真值表，然后用case语句描述。）

常用，熟练掌握！

（2）逻辑抽象→写出逻辑函数表达式→HDL编程（算法级描述，assign语句）

（3）逻辑抽象→写出逻辑函数表达式→绘逻辑图（适于简单电路）

本章小结（3/8）

5、常用的组合逻辑电路

- 有算术运算电路、 编码器、译码器、数据选择器、数值比较器、奇偶校验器等
- 过去考虑到常用组合逻辑电路的特点，为便于使用，一般将它们制成标准化的中规模集成器件。
- 多数集成组合逻辑电路都设置了控制端（使能端），用来控制电路的工作状态（工作或禁止）；作为输出信号的选通信号，实现器件的扩展。
- Verilog HDL具有多种建模方式（系统级、算法级、RTL级、门级）和功能描述语句（case语句、if语句、assign语句），用来设计组合逻辑电路非常方便、快捷。
- 当组合逻辑电路的输出信号作为后续电路的边沿触发信号时，若存在竞争-冒险（输出有毛刺），则一定要消除！

本章小结（4/8）

（1） 算术运算电路

- 能完成二进制数算术运算的器件
- 基本单元电路：半加器和全加器
- 多位加法器
  - 串行进位加法器、并行进位加法器

（2） 编码器（Encoder）

- **将加在电路若干输入端中的某一个输入端的信号变换成相应的一组二进制代码输出的过程叫做编码。**
- **实现编码功能的数字电路称为编码器。**
  - **二进制编码器：用n位二进制代码对$M=2^n$个信号进行编码的电路（如8线-3线编码器）。任何时刻只允许一个输入信号有效**
  - **BCD码编码器：用二进制码表示十进制数的编码器（如8421BCD编码器）。**
  - **优先编码器：如有两个或两个以上的输入有效时，只对优先级最高的输入信号进行编码的编码器。**

本章小结（5/8）

（3） 译码器（Decoder）

- 将二进制代码所表示的信息翻译成对应输出的高低电平信号的过程称为译码，实现译码功能的电路称为译码器。
  - 变量译码器（二进制译码器）：表示输入变量状态全部组合的译码器。一般称为n线-2^n线译码器。常用的有双2线-4线译码器，3线-8线译码器，4线-16线译码器。任何时刻最多只允许1个输出有效
  - 码制变换译码器：将输入的二进制代码转换成对应的其他码制输出的译码器。如BCD译码器。
  - 显示译码器：将输入代码转换成驱动7段数码显示器各段的电平信号的译码器。
- 二进制译码器应用：实现存储器系统的地址译码，实现组合逻辑函数
- 如果给定一个组合逻辑函数，如何用利用二进制译码器和门电路实现？
- 有灭零控制的多位数码显示系统

本章小结（6/8）

（4）数据选择器（Data Selector）

- 从一组输入数据选出其中需要的一个数据作为输出的过程叫做**数据选择**，具有数据选择功能的电路称为**数据选择器**。
- 数据选择器又称多路开关（Multiplexer，多路器），常用的有四2选1、双4选1、8选1及16选1数据选择器等 。
- 功能
  - 地址信号作为控制信号——实现数据选择
  - 数据输入信号作为控制信号——多功能运算电路，实现任意组合逻辑函数
  - 代替三态门，实现总线发送控制
- 如果给定一个组合逻辑函数，如何用数据选择器实现？
- 对于一个实际应用题目，如何用数据选择器设计组合逻辑电路？

本章小结（7/8）

（5）数值比较器

- **数值比较器是一种关系运算电路，它可以对两个二进制数或二-十进制编码的数进行比较，得出大于、小于和相等的结果。**
- 分为“等值”比较器和“量值”比较器
- 常用的有1位数值比较器、4位数值比较器

本章小结（8/8）

（6） 奇偶校验器

- 奇偶校验就是检测数据中包含“1”的个数是奇数还是偶数。
- 奇偶校验器是采用“奇偶校验”方法来检查数据传输后和数码记录中是否存在错误的一种电路。
- 常用有4位奇偶校验器、8位奇偶校验器。
- 异或运算也称为“模2加”运算——只考虑两个二进制数相加后的和，而不考虑它们的进位。
  - 当n个二进制数“模2加”时，若和为1，表示n个数中有奇数个“1”；若和为0，表示有偶数个“1”。
- 奇偶校验系统
  - 若输出F_{OD}=1，F_{EV}=0，表示数据传输正确；若F_{OD}=0，F_{EV}=1，表示数据传输有差错

记住各种常用组合逻辑电路的分析方法和HDL设计方法！

教材第5章勘误表

页码	错 误	更 正
98	Y=αγ	Y= βγ
111	图4.28中前5个CT7448的$\overline{RBI}$和$\overline{RBO}$颠倒了	将原“$\overline{RBI}$”改成“$\overline{RBO}$”，原“$\overline{RBO}$”改成“$\overline{RBI}$“
115	表4.14中后4行有错	改为与本课件P84中功能表一致
121	default: {D,C,B,A}=4’bx;	“4’bx”改为“4’bxxxx”
122	……YN3~YN0是4线数据输入端	“输入”改为“输出”
126	……若A_3~A_0=0000时，输出$\overline{W}$=D_0; 若A_3~A_0=0001时，输出$\overline{W}$=D_1……	“A_3~A_0=0000”改为“A_2~A_0=000”， “A_3~A_0=0001”改为“A_2~A_0=001”
127	else Y=1’b1; ……当使能控制STN=0时，电路被禁止，输出Y=1（无效）……	“1’b1” 改为“1’b0 ” “STN=0”改为“STN=1” “Y=1”改为“Y=0”
130	2'b00 : begin FE=1'b0; FOD=1'b0;end 2'b11 : begin FE=1'b1; FOD=1'b1;end	2'b00 : begin FE=1'b1; FOD=1'b1;end 2'b11 : begin FE=1'b0; FOD=1'b0;end