# 第五讲

# 前两讲内容简要回顾

- **逻辑门电路：数字电路中的基本逻辑单元电路**
  - 由晶体管和MOS管（晶体二极管、晶体三极管、NMOS、PMOS）构建门电路（与、或、非、与非、或非等）
- **布尔代数：分析与设计数字系统的重要理论工具**
  - **逻辑代数基本概念**
    - 逻辑常量/变量，典型逻辑运算
  - **逻辑代数的运算法则**
    - 公理5、定律9、定理3、基本公式4及其推论1
  - **逻辑函数的表达式**
    - 标准表达式：最小项表达式、最大项表达式
    - 标准表达式可由真值表直接得出
  - **逻辑函数的简化法**
    - 利用对偶规则，可将“或与”表达式转化成“与或”表达式来化简
    - 常用简化法：合并乘积项法<互补律,消变量>、吸收项法<吸收律/包含律，减与项>和配项法<互补律，加与项>

# 第二部分：组合逻辑

一. 逻辑门电路

二. 布尔代数

三. Verilog HDL介绍

1. Verilog HDL概述
2. Verilog HDL的词法
3. Verilog HDL常用语句
4. 不同抽象级别的Verilog HDL模型

四. 基本组合逻辑部件设计

1. 组合逻辑电路设计概述
2. 运算单元电路
3. 编码器/译码器
4. 多路选择器

# 组合逻辑电路的结构和特点

- 数字电路分类：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。
  - 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。
  - 常用的组合逻辑电路：算术逻辑运算电路、 编码器/译码器、数据选择器、数值比较器、奇偶校验器等

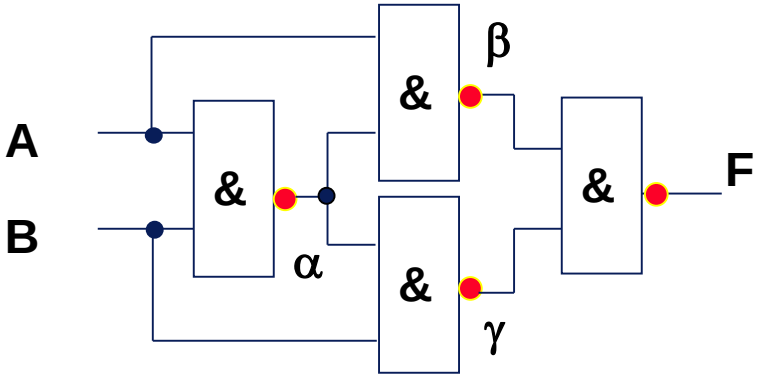

- 特点
  - 由逻辑门电路组成
  - 输出不能再直接反馈到输入（不能有环路），没有存储电路
  - 当时的输出仅由当时的输入决定——速度快

# 组合逻辑电路的分析方法

## 组合逻辑电路的不同表示方法：

逻辑图 → 表达式 → 真值表 → 电路功能

【例】分析下图电路




| A | B | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

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

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

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

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

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

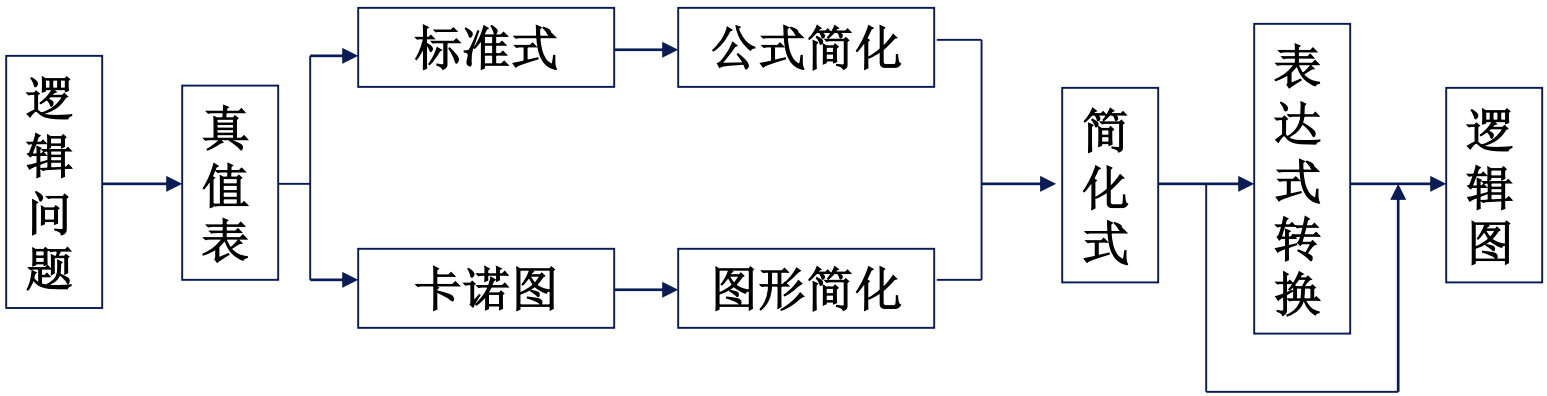
电路功能：异或电路

# 组合逻辑电路的设计方法

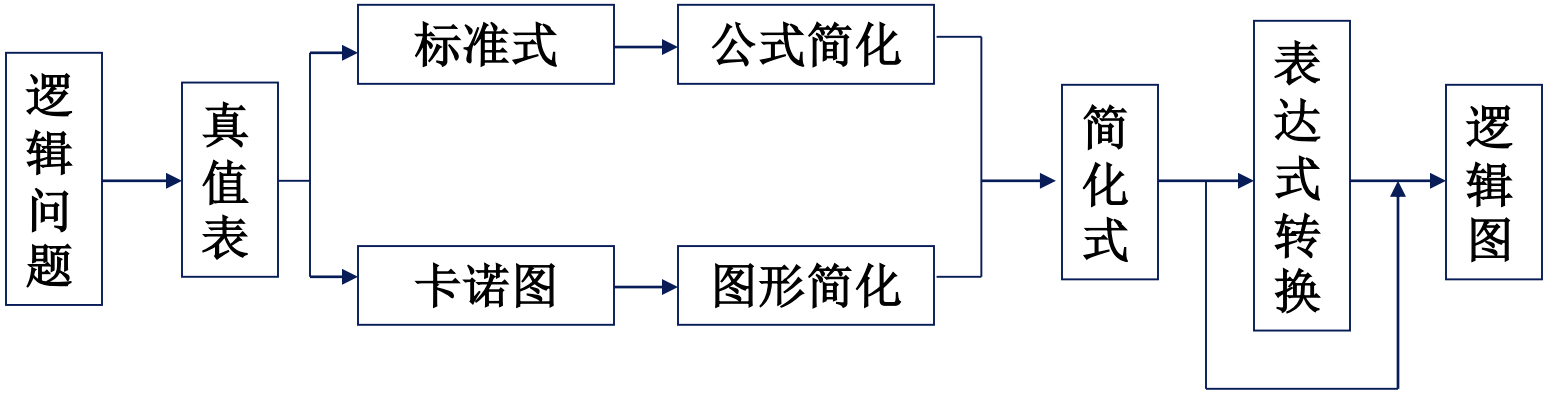
❖ **组合逻辑电路的设计** —— 根据给定的功能要求，采用某种设计方法，得到满足功能要求且最简单的组合逻辑电路

❖**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，分析因果关系，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，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，画出逻辑电路图



# 组合逻辑电路的手工设计方法

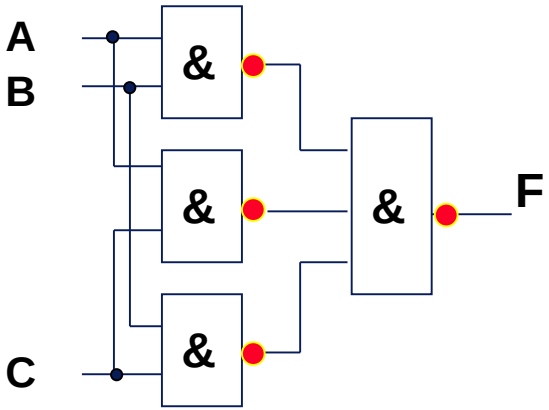



三人表决器

| A | B | C | 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 |

$$F = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$
$$= AB + AC + BC = \overline{\overline{AB} \cdot \overline{AC} \cdot \overline{BC}}$$

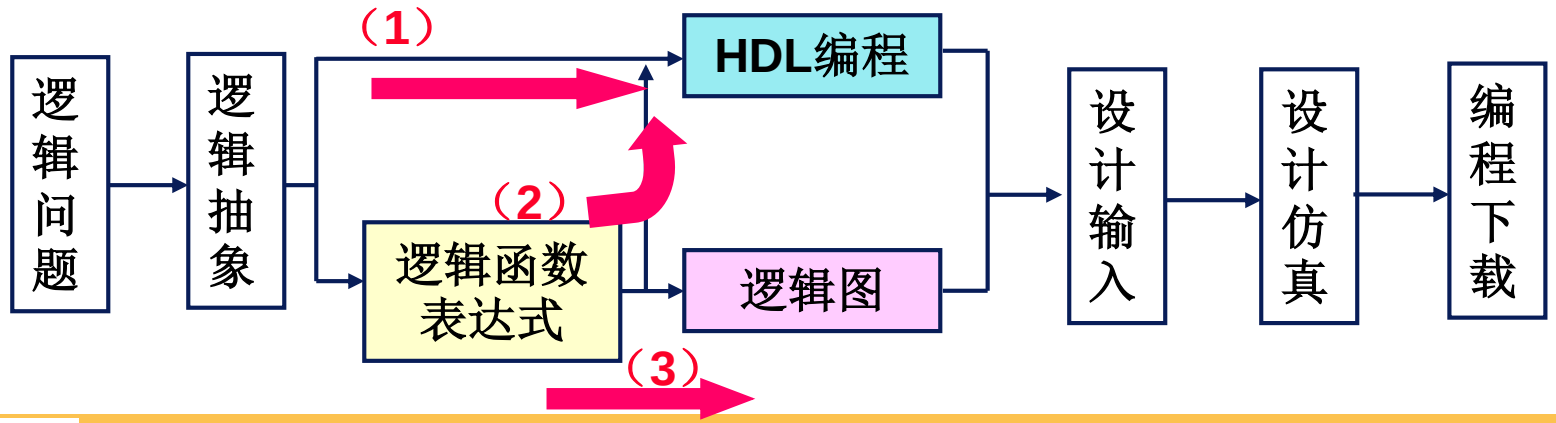
全部用与非门实现



# 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
  - 逻辑抽象→HDL编程（系统级描述，如用case语句或if-else语句）
  - 逻辑抽象→写出逻辑函数表达式→HDL编程（算法级描述，assign语句）
  - 逻辑抽象→写出逻辑函数表达式→绘逻辑图（适于简单电路）



# 第二部分：组合逻辑

一. 逻辑门电路

二. 布尔代数

三. Verilog HDL介绍

1. Verilog HDL概述
2. Verilog HDL的词法
3. Verilog HDL常用语句
4. 不同抽象级别的Verilog HDL模型

四. 基本组合逻辑部件设计

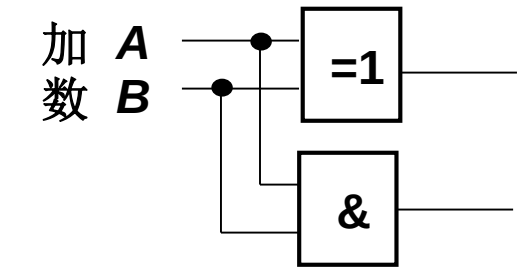
1. 组合逻辑电路设计概述
2. 运算单元电路
3. 编码器/译码器
4. 多路选择器

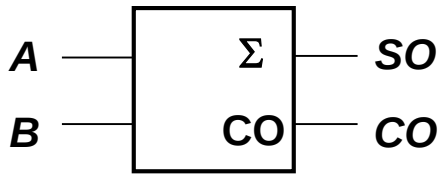
# 运算单元电路 —— 1位半加器

- 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。




$$SO = A \oplus B$$

$$CO = AB$$

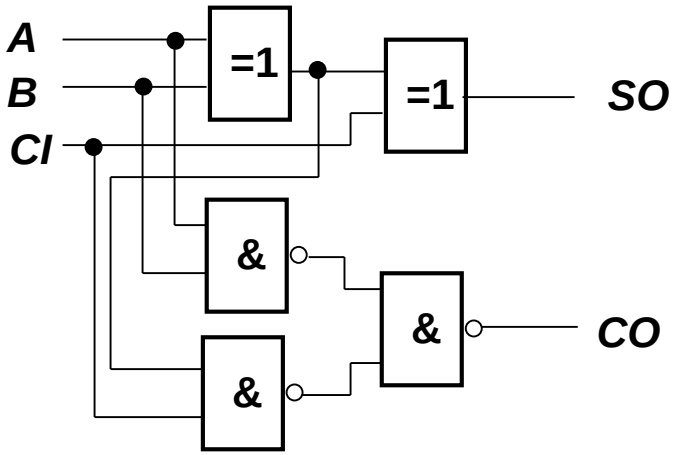
真值表

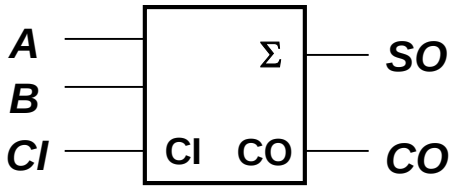
| A | B | SO | CO |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

# 运算单元电路 —— 1位全加器

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




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

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

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 |

# 运算单元电路 —— 1位全加器的HDL设计 —— 算法级

❖ 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）

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

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

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
```

# 运算单元电路 —— 1位全加器HDL设计 —— 系统级

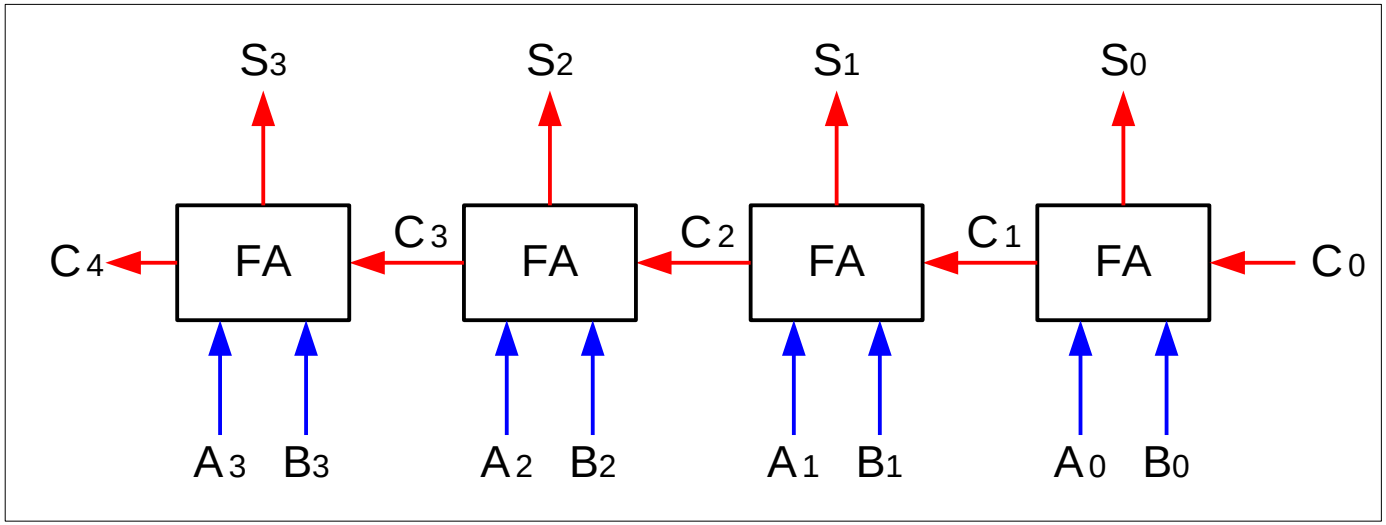
❖方法二：采用行为描述方式的系统级抽象，根据逻辑功能定义直接描述，程序更简洁！

```
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 “{ }”，将进位与和拼接在一起成为一个 2 位数 ！

# 运算单元电路 —— 多位加法器 —— 串行进位

❖并行加法器——串行进位




$$C_1 = A_0B_0 + C_0(A_0 \oplus B_0)$$
$$C_2 = A_1B_1 + C_1(A_1 \oplus B_1)$$
$$C_3 = A_2B_2 + C_2(A_2 \oplus B_2)$$
$$C_4 = A_3B_3 + C_3(A_3 \oplus B_3)$$

串行进位的特点：

1. 进位串行传递
2. 进位延时较长

## ❖ 并行加法器——并行进位（或先行进位）

$$C_1 = A_0B_0 + C_0(A_0 \oplus B_0)$$

$$C_2 = A_1B_1 + C_1(A_1 \oplus B_1)$$

$$C_3 = A_2B_2 + C_2(A_2 \oplus B_2)$$

$$C_4 = A_3B_3 + C_3(A_3 \oplus B_3)$$

令 $G_i = A_iB_i$，$P_i = A_i \oplus B_i$

$$C_1 = G_0 + P_0C_0$$

$$C_2 = G_1 + C_1P_1 = G_1 + G_0P_1 + P_0P_1C_0$$

$$C_3 = G_2 + C_2P_2 = G_2 + G_1P_2 + G_0P_1P_2 + P_0P_1P_2C_0$$

$$C_4 = G_3 + C_3P_3 = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3 + P_0P_1P_2P_3C_0$$

## ❖ 并行进位链

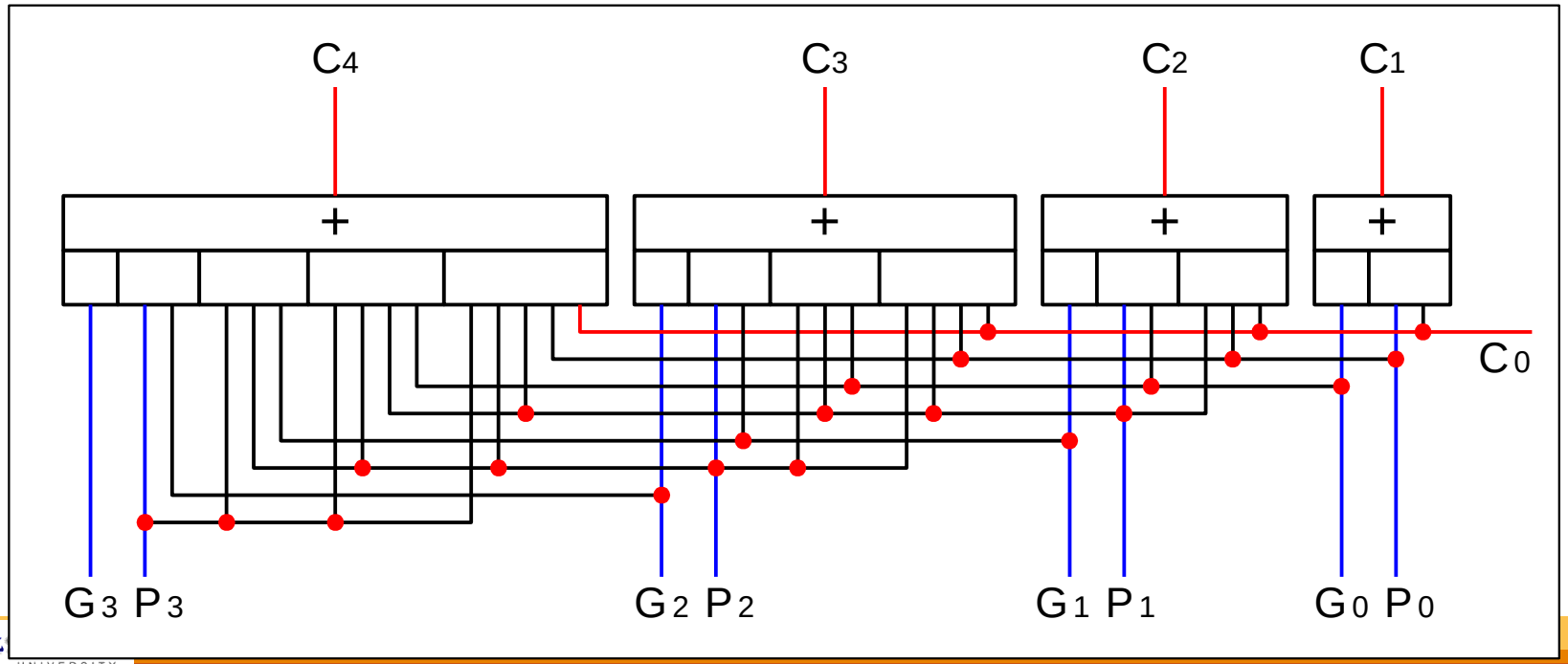
令 $G_i = A_i B_i$，$P_i = A_i \oplus B_i$

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + C_1 P_1 = G_1 + G_0 P_1 + P_0 P_1 C_0$$

$$C_3 = G_2 + C_2 P_2 = G_2 + G_1 P_2 + G_0 P_1 P_2 + P_0 P_1 P_2 C_0$$

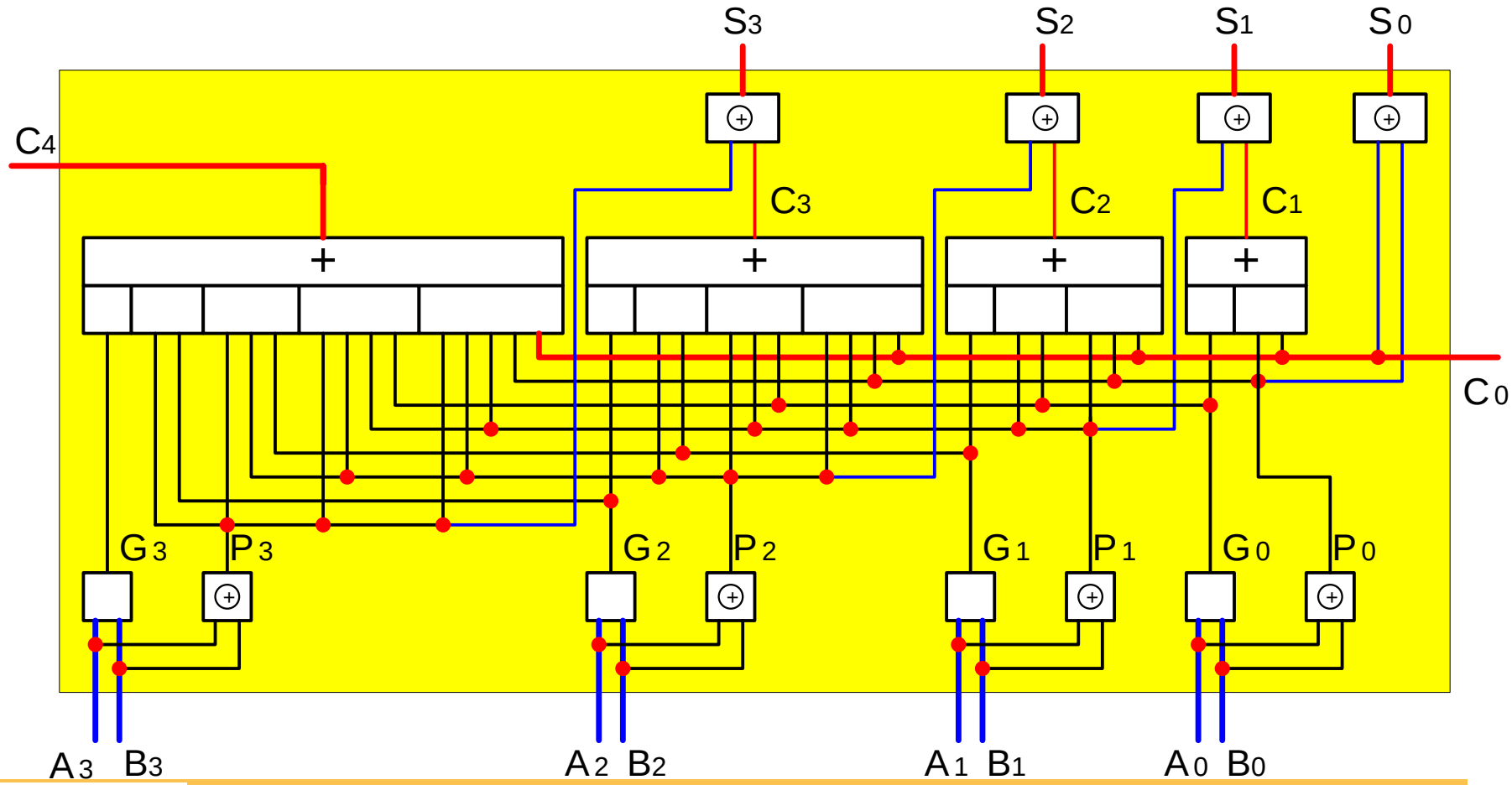
$$C_4 = G_3 + C_3 P_3 = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + P_0 P_1 P_2 P_3 C_0$$



**并行进位的特点：**

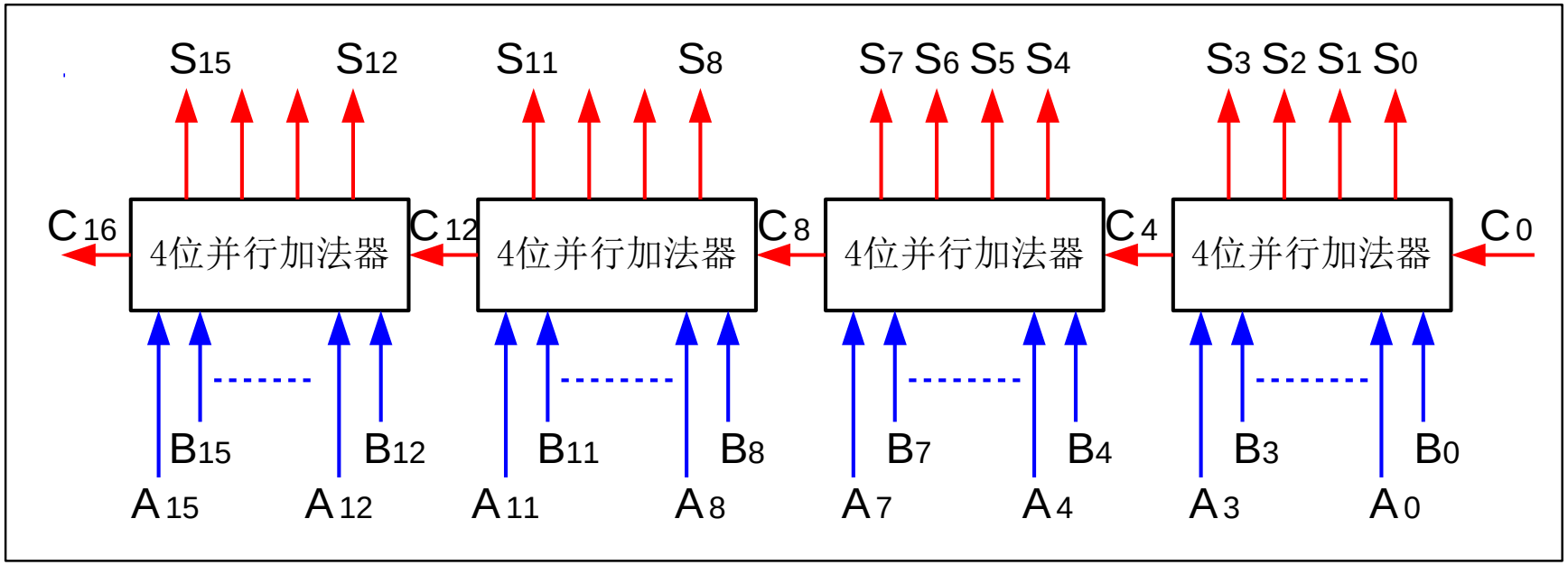
1. 同时产生进位
2. 加法延时缩短
3. 实现相对复杂

## ❖ 并行进位加法器



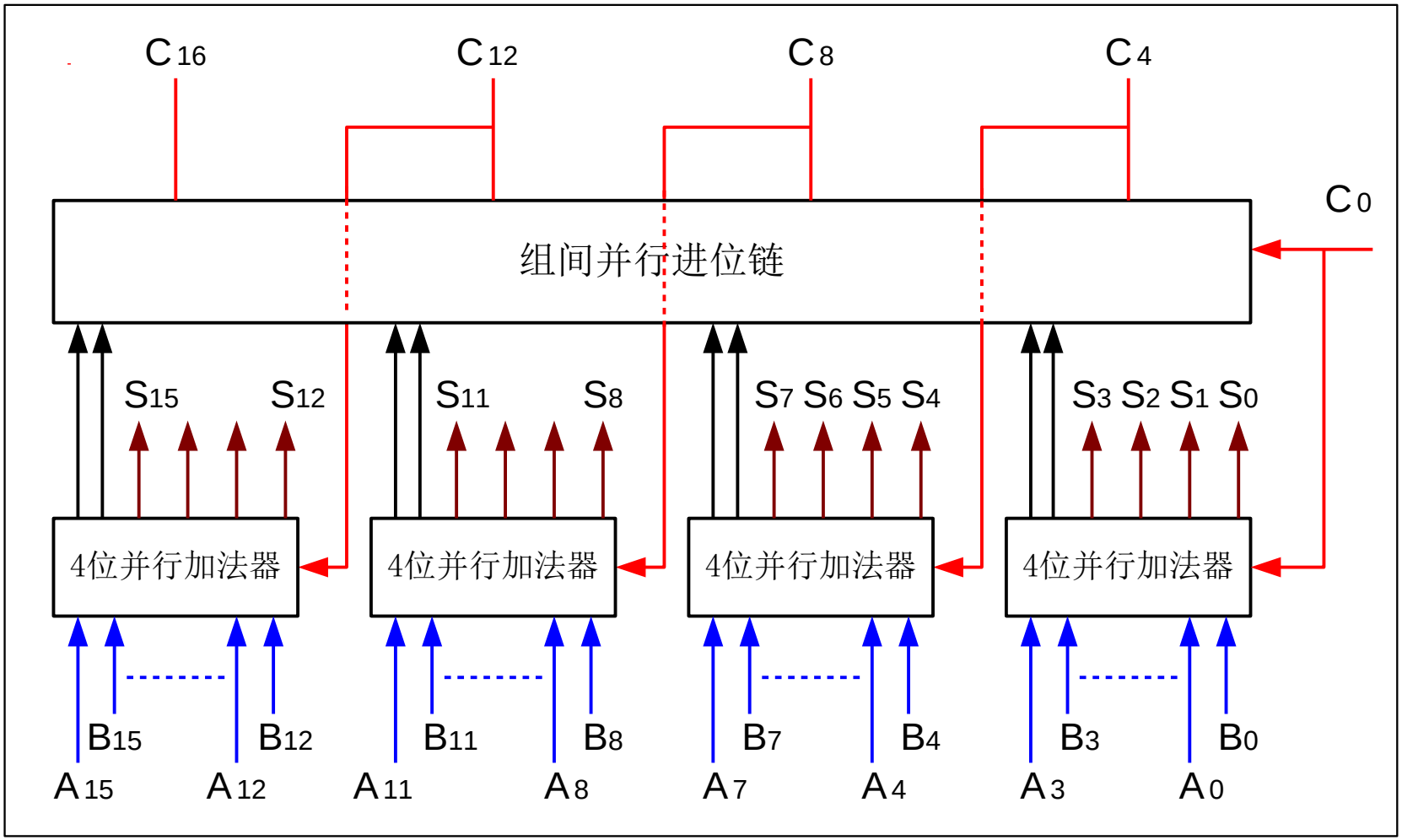
# 运算单元电路 —— 多位加法器 —— 并行进位

- 分组并行进位加法器（组内并行，组间传递）



# 运算单元电路 —— 多位加法器 —— 并行进位

❖分组并行进位加法器（组内并行，组间并行）



# 运算单元电路 —— 多位加法器的HDL设计 —— 系统级

- 用Verilog HDL行为描述方式，很容易编写出任意位数的加法器
- 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电路

- 原则（以定点整数为例说明）

$$[A+B]_{补}=[A]_{补}+[B]_{补}$$

$$[A-B]_{补}=[A]_{补}+[-B]_{补}$$

- $[X]_{补}$与$[-X]_{补}$

若$[x]_{补}=x_0x_1x_2\ldots x_{n-1}$

则$[-x]_{补}=\overline{x}_0\overline{x}_1\overline{x}_2\ldots\overline{x}_{n-1}+1$

所以有

$$[A-B]_{补}=[A]_{补}+\overline{[B]_{补}}+1$$

**加减法可共用同一套加法器电路**

# 运算单元电路 —— 阵列乘法器

## ❖ 基本思路

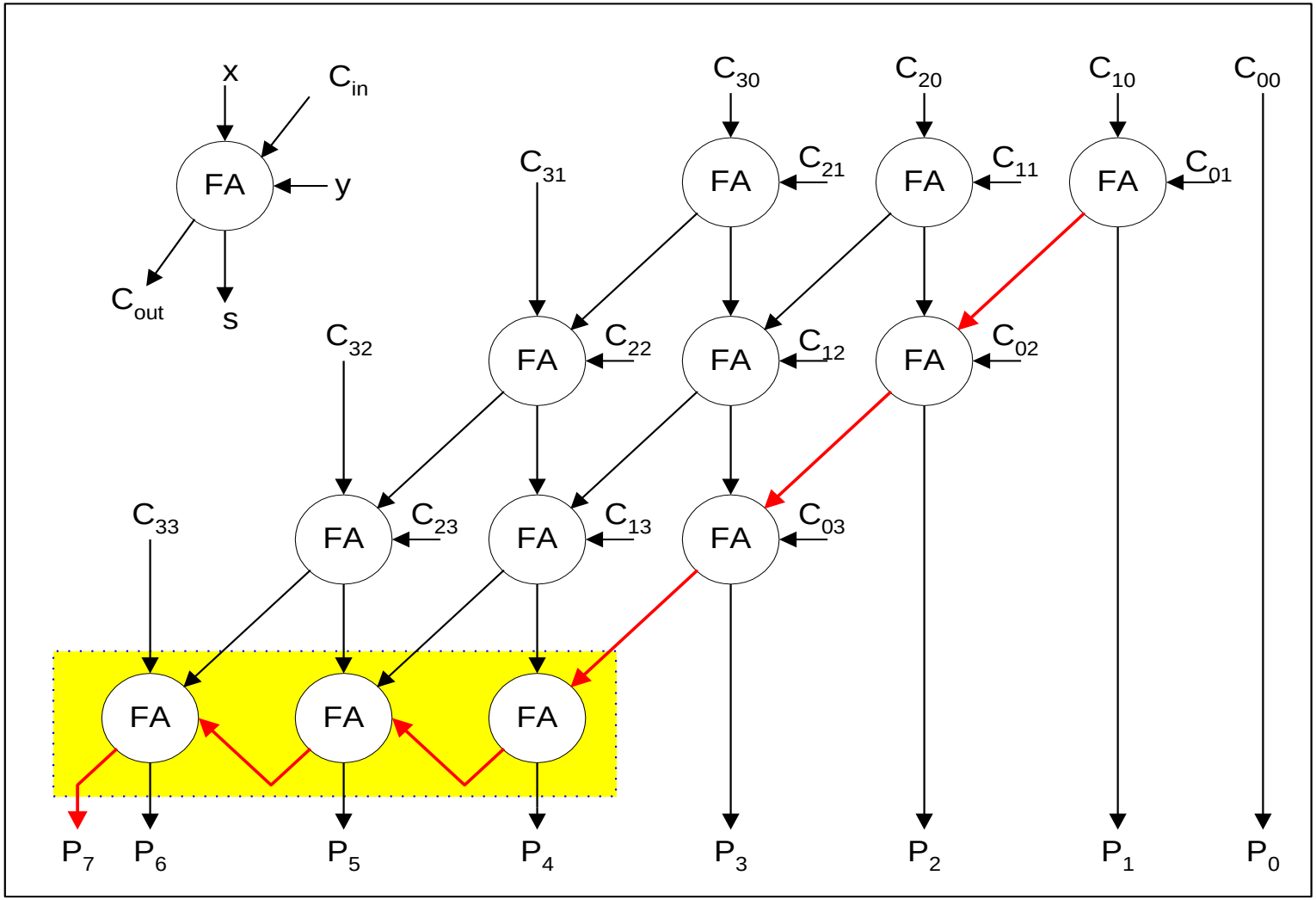
- ➢ 利用若干全加器，完全由硬件直接计算乘法结果
- ➢ 以 4 位无符号数为例

$$
\begin{array}{cccccccc}
 & & & & A_3 & A_2 & A_1 & A_0 \\
\times & & & & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & & & C_{30} & C_{20} & C_{10} & C_{00} \\
 & & & C_{31} & C_{21} & C_{11} & C_{01} & \\
 & & C_{32} & C_{22} & C_{12} & C_{02} & & \\
+ & C_{33} & C_{23} & C_{13} & C_{03} & & & \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}
$$

其中 $C_{ij} = A_i B_j$

# 运算单元电路 —— 阵列乘法器

## ❖实现电路



# 运算单元电路 —— 阵列乘法器

- 总结：
  - 对于n位的阵列乘法，需全加器n(n-1)个
  - 最长路径 2(n-1)个全加器延时
  - 最后的串性进位可采用先行进位加法器

# 运算单元电路 —— 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。

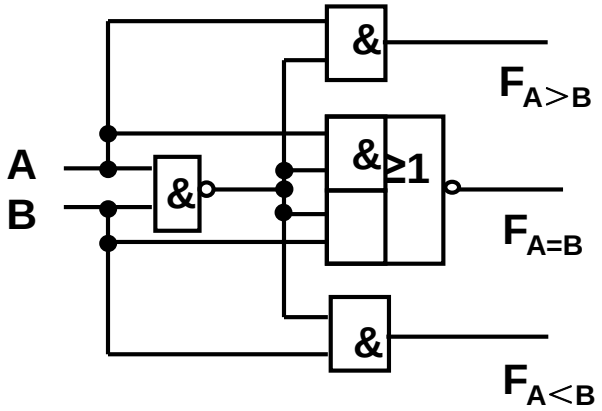
## 1、1位数值比较器

用来比较两个一位二进制数大小的电路。

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

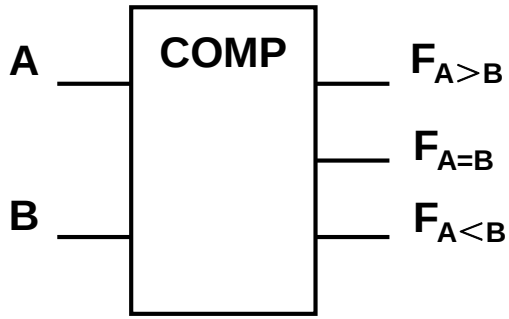
$$F_{A<B} = B\overline{AB} = B(\overline{A}+\overline{B}) = \overline{A}B$$

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




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 |

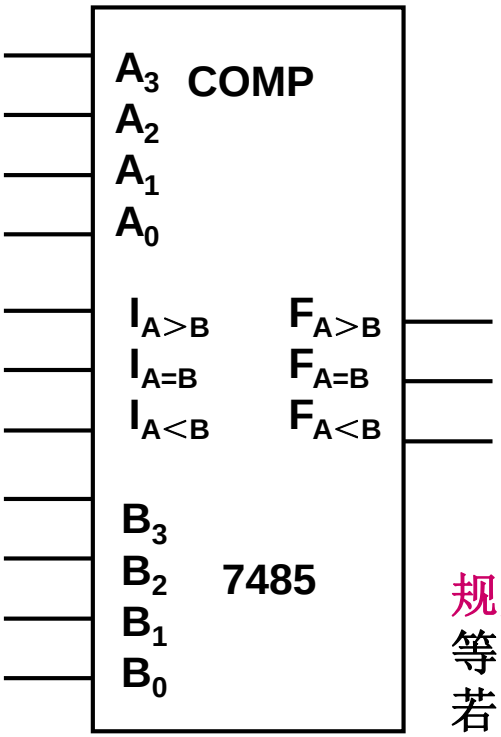


# 运算单元电路 —— 4位数值比较器（7485）

## 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。

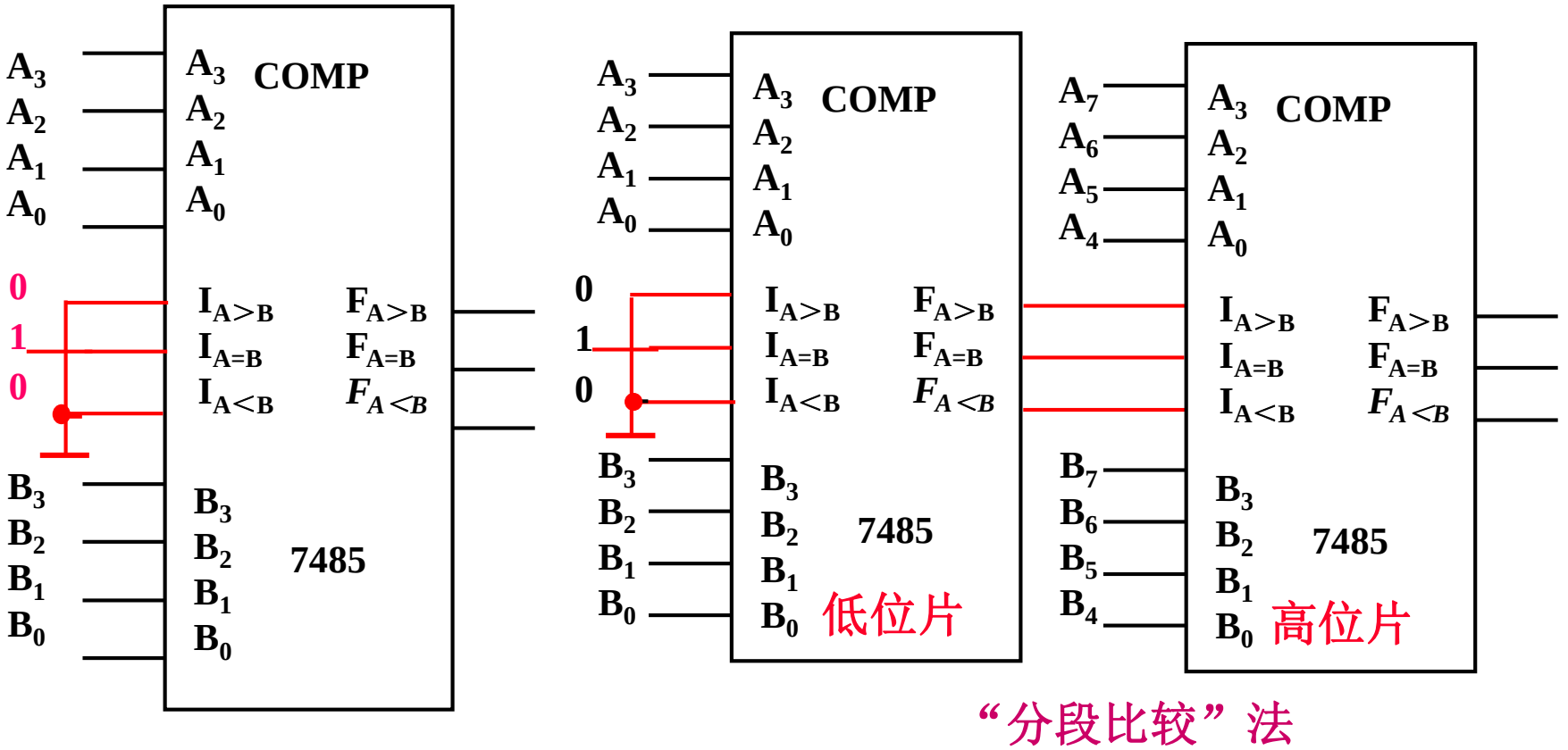
若$A_3>B_3$ 则A>B；
若$A_3<B_3$ 则A<B；
若$A_3=B_3$ 则再比较低位

# 运算单元电路 —— 7485的使用与扩展方法

## （3）使用与扩展方法

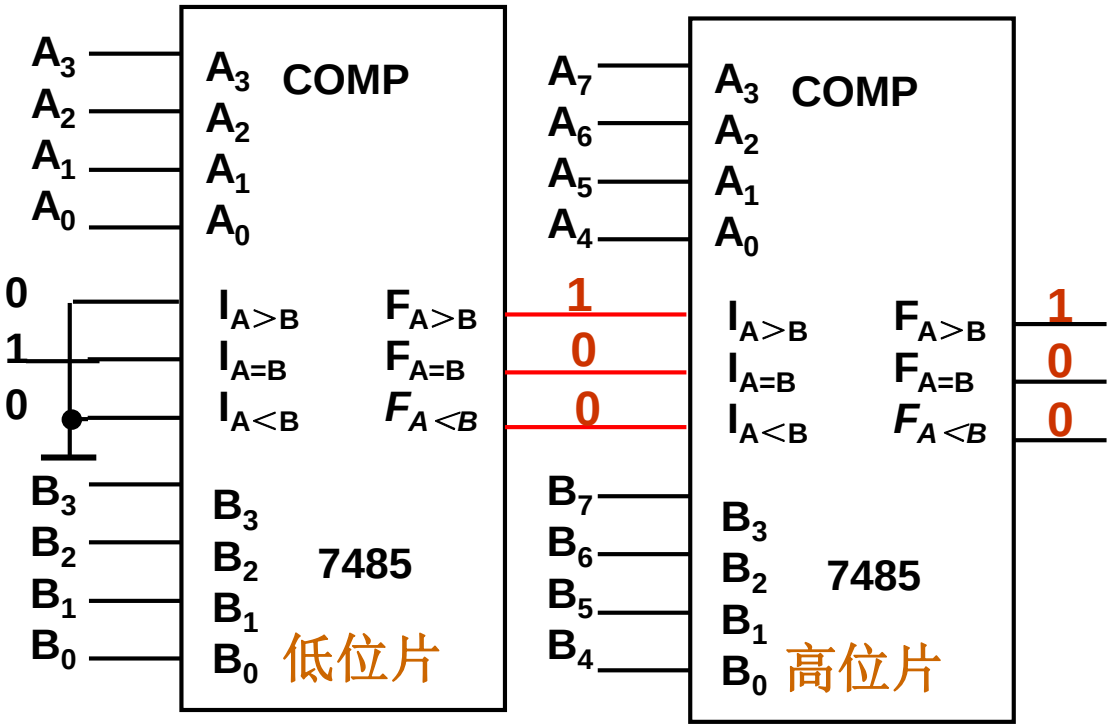
①单片使用——4位数值比较器

② 2片扩展——8位数值比较器




“分段比较”法

# 运算单元电路 ——2片扩展——“分段比较”法

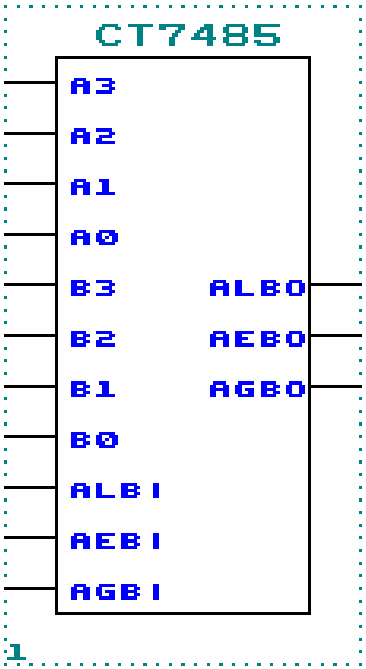



- 低位片和高位片并行工作，每片的比较仍是由高位到低位逐位进行
  - 若高4位数不相等，则由两个高4位数A7~A4与B7~B4的大小决定A和B的大小。

- 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。

# 运算单元电路 —— 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。



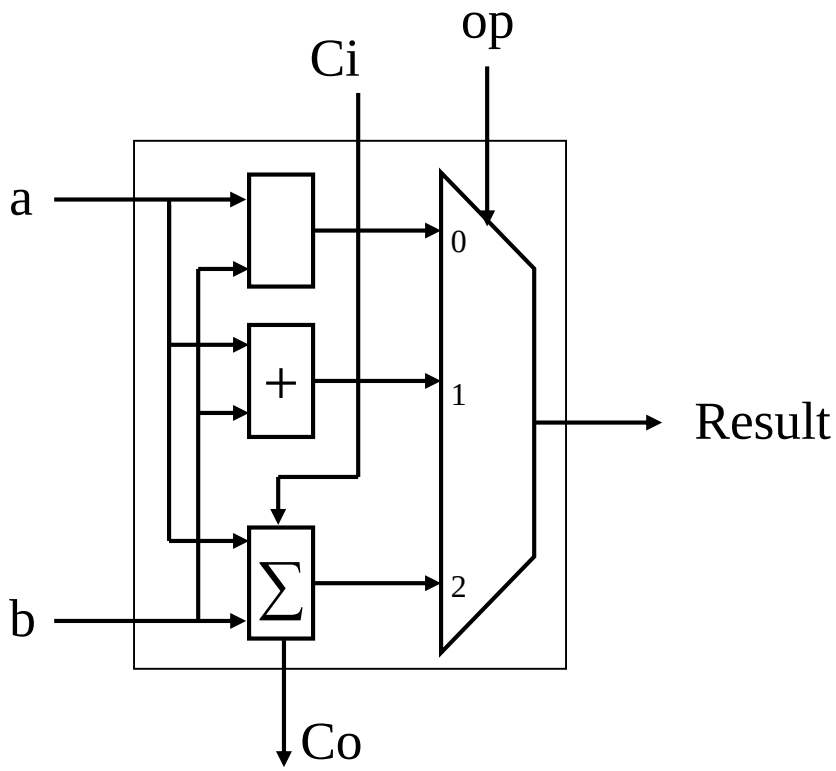
# 运算单元电路 —— 7485的Verilog HDL源程序

```
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
```

# 运算单元电路 —— 1位ALU —— 与或加功能

算术逻辑运算单元ALU：

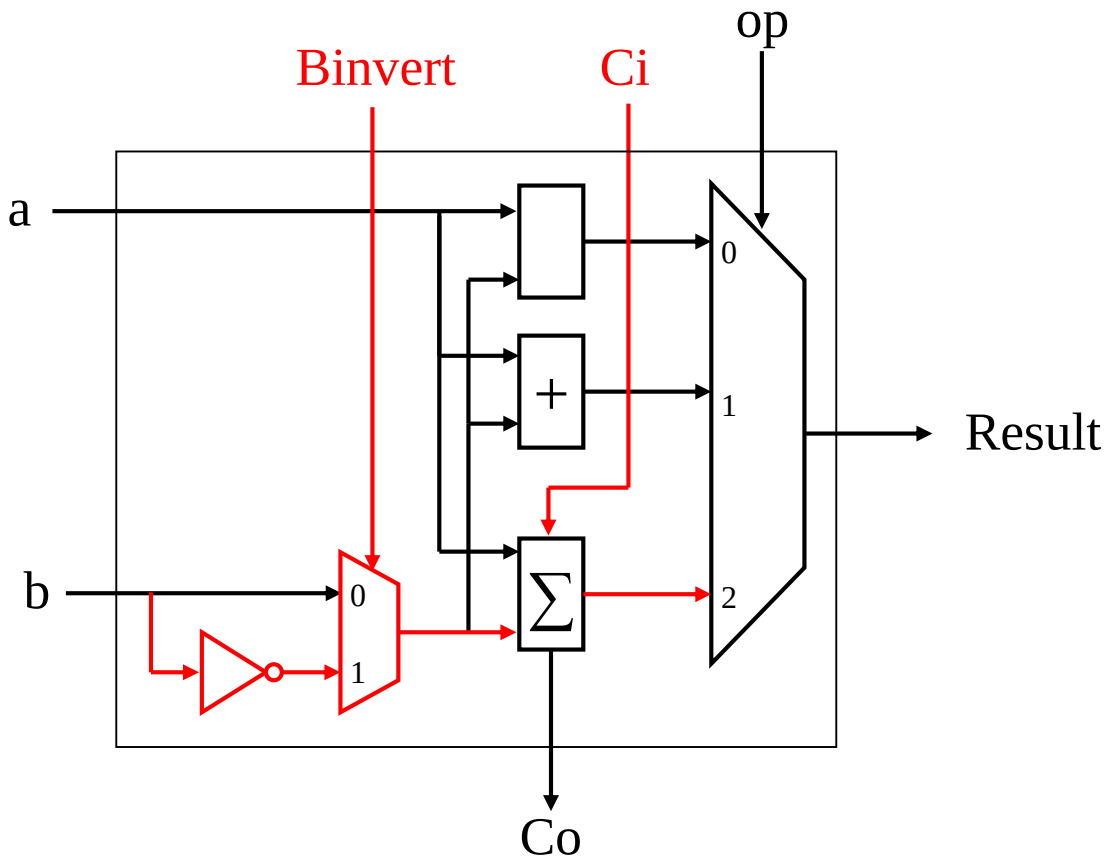
- 与、或功能
- 加法功能



# 运算单元电路 —— 1位ALU —— 与或加减功能

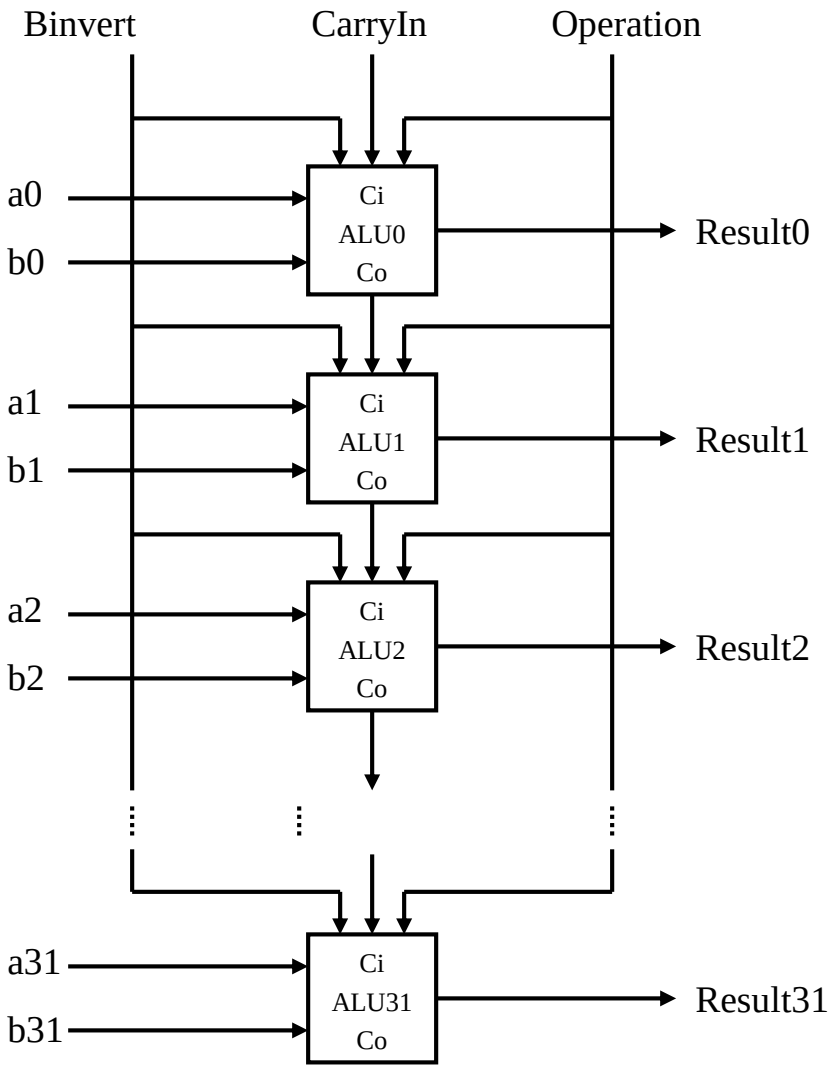
- 与、或功能
- 加法功能
- 减法（Binvert=1,且$C_i$=1）

$$[A-B]_{补}=[A]_{补}+\overline{[B]_{补}}+1$$



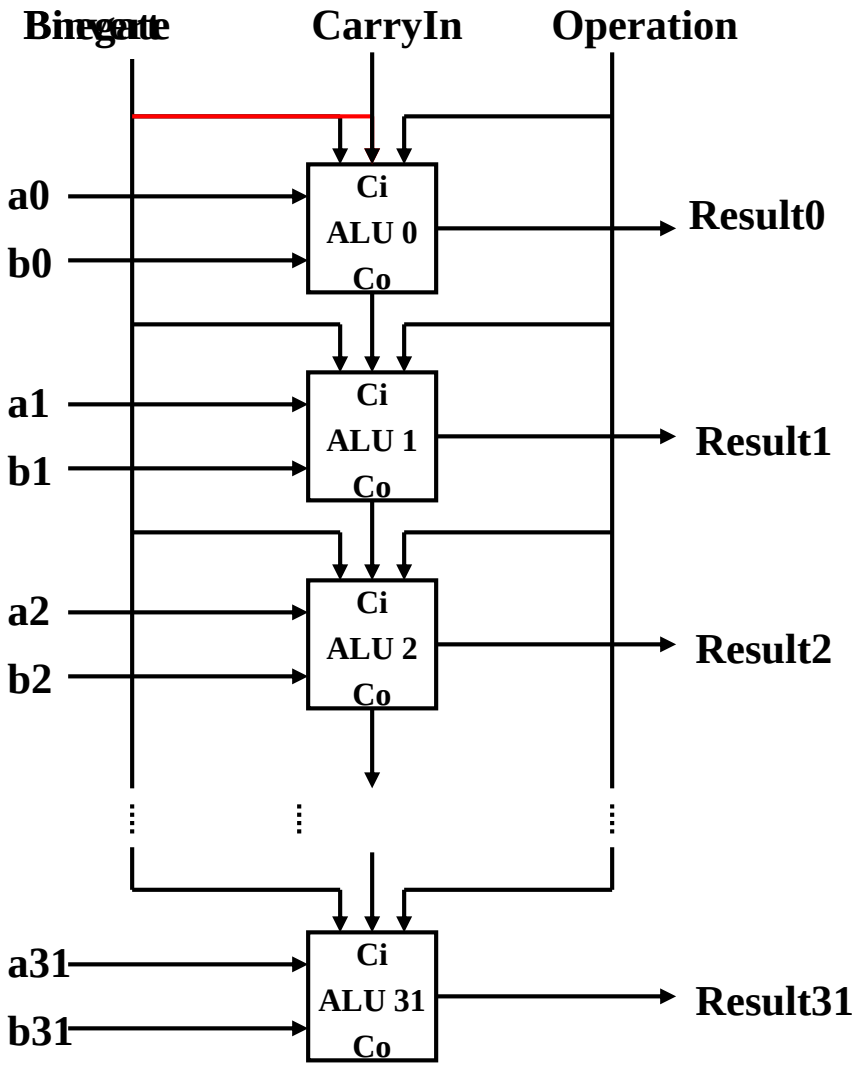
# 运算单元电路 —— 32位ALU —— 与或加减功能

- 与、或
- 加法
- 减法



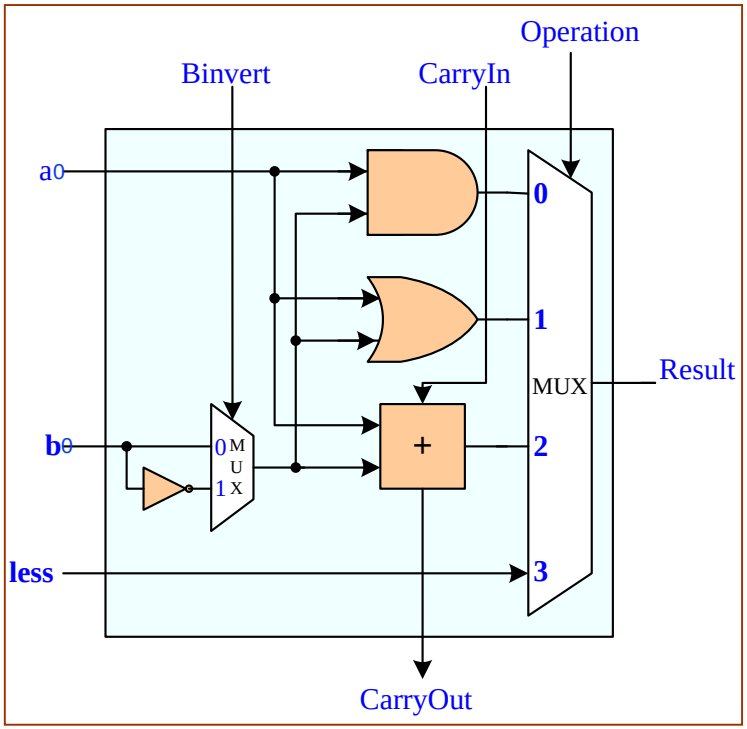
# 运算单元电路 —— 32位ALU —— 简化控制

- 加法
  - Binvert=0,CarryIn=0
- 减法
  - Binvert=1,CarryIn=1
- 引入Bnegate

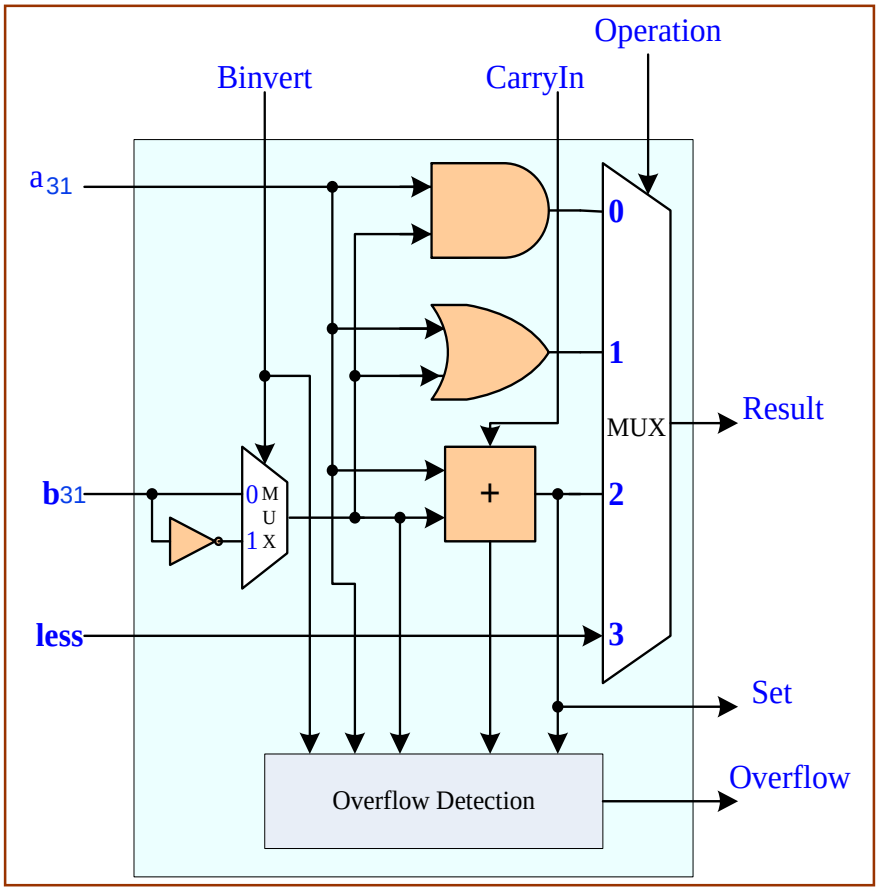


# 运算单元电路 —— 1位ALU（带比较功能）

- 逻辑：与、或
- 算术
  - 加：Binvert=0,CarryIn=0
  - 减：Binvert=1,CarryIn=1
- 比较：Less（小于）



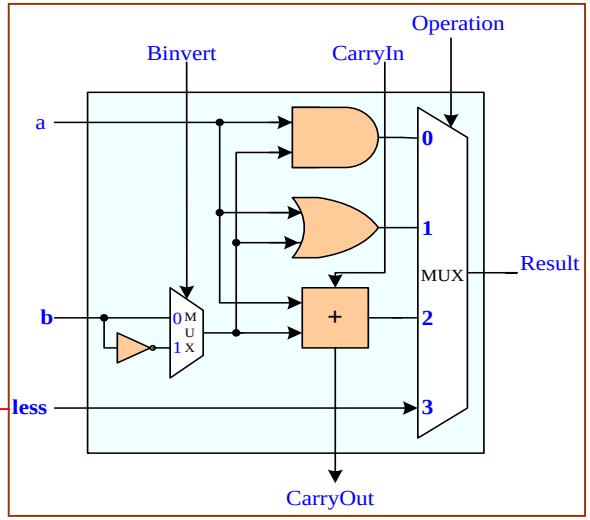

1位ALU（带比较功能）



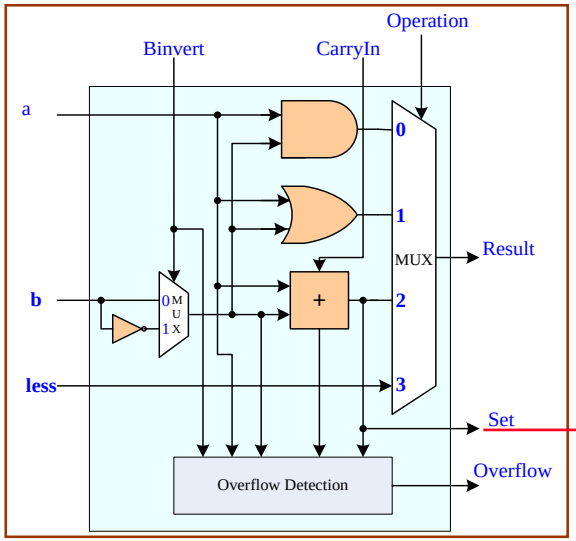

最高位（MSB）

a < b: (a-b) < 0, Set = 1
a >= b: (a-b) >= 0, Set = 0

# 运算单元电路 —— 32位ALU（带比较功能）




**ALU 0**



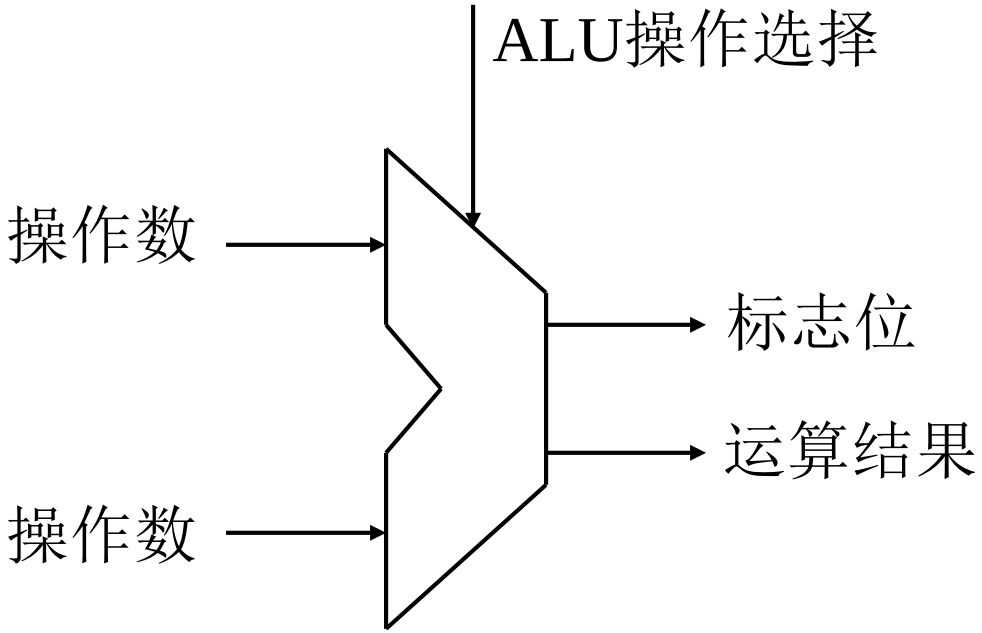

**ALU 31**

0

**a < b: (a-b) < 0, Set = 1**

**a >= b: (a-b) >= 0, Set = 0**

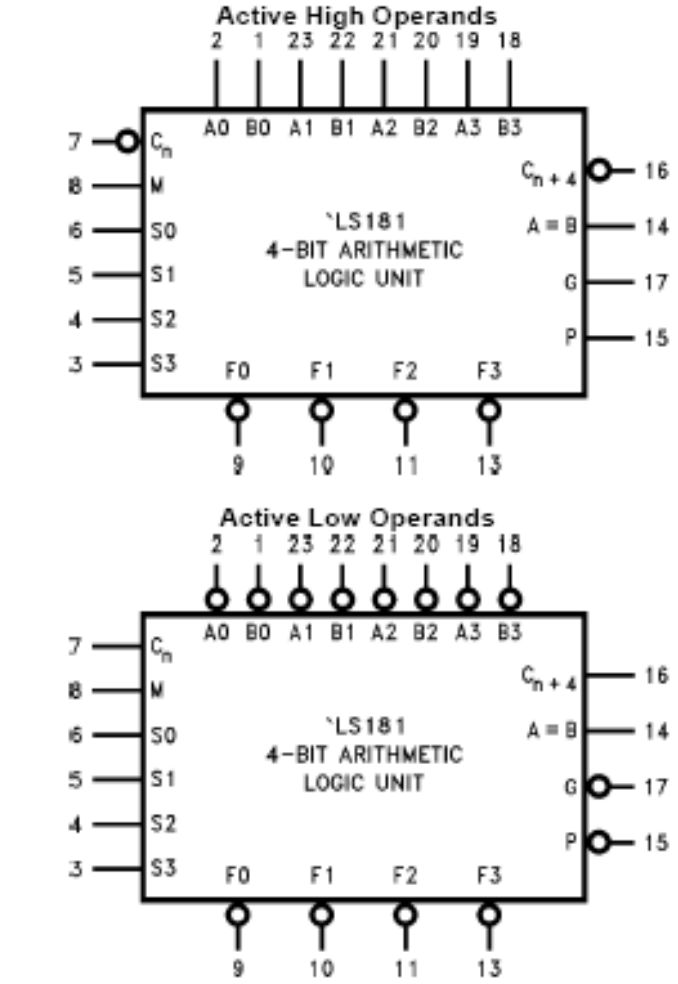
# 运算单元电路 —— ALU抽象表示



# 运算单元电路 —— ALU芯片—DM74LS181N

- 4位ALU
- 提供16种算术逻辑运算
- 两种工作模式：

  正逻辑和负逻辑



# 运算单元电路 —— ALU芯片—DM74LS181N

## Function Table

**Function Table**

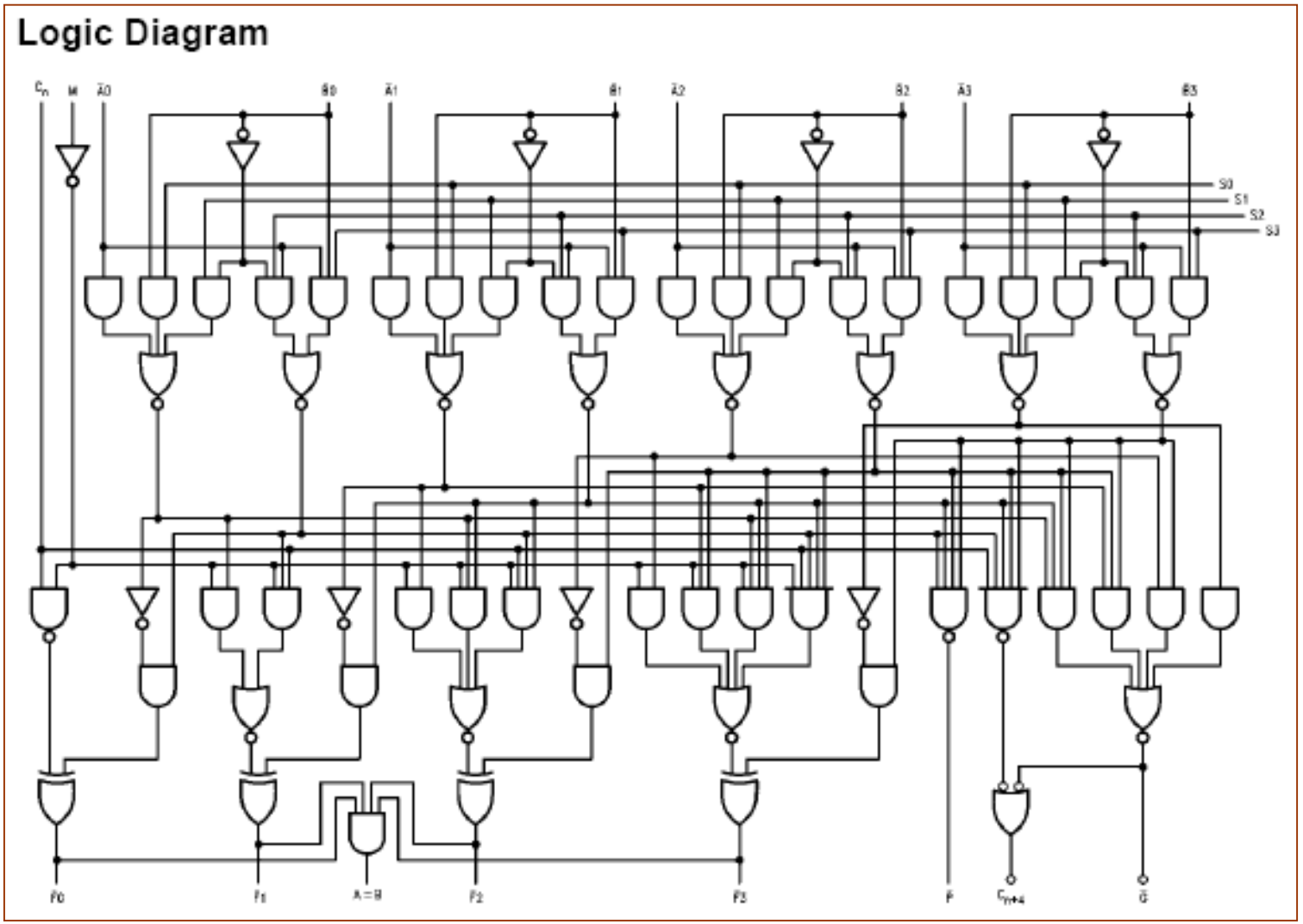
| Mode Select Inputs | | | | Active LOW Operands & $F_n$ Outputs | | Active HIGH Operands & $F_n$ Outputs | |
|---|---|---|---|---|---|---|---|
| | | | | Logic | Arithmetic (Note 2) | Logic | Arithmetic (Note 2) |
| S3 | S2 | S1 | S0 | (M = H) | (M = L) ($C_n$ = L) | (M = H) | (M = L) ($C_n$ = H) |
| L | L | L | L | $\overline{A}$ | A minus 1 | $\overline{A}$ | A |
| L | L | L | H | $\overline{AB}$ | AB minus 1 | $\overline{A}+\overline{B}$ | A + B |
| L | L | H | L | $\overline{A}+B$ | $A\overline{B}$ minus 1 | $\overline{A}B$ | $A+\overline{B}$ |
| L | L | H | H | Logic 1 | minus 1 | Logic 0 | minus 1 |
| L | H | L | L | $\overline{A}+\overline{B}$ | A plus $(A+\overline{B})$ | $\overline{AB}$ | A plus $A\overline{B}$ |
| L | H | L | H | $\overline{B}$ | AB plus $(A+\overline{B})$ | $\overline{B}$ | $(A+B)$ plus $A\overline{B}$ |
| L | H | H | L | $\overline{A}\oplus\overline{B}$ | A minus B minus 1 | $A\oplus B$ | A minus B minus 1 |
| L | H | H | H | $A+\overline{B}$ | $A+\overline{B}$ | $A\overline{B}$ | AB minus 1 |
| H | L | L | L | $\overline{A}B$ | A plus (A + B) | $\overline{A}+B$ | A plus AB |
| H | L | L | H | $A\oplus B$ | A plus B | $\overline{A}\oplus\overline{B}$ | A plus B |
| H | L | H | L | B | $A\overline{B}$ plus (A + B) | B | $(A+\overline{B})$ plus AB |
| H | L | H | H | A + B | A + B | AB | AB minus 1 |
| H | H | L | L | Logic 0 | A plus A (Note 1) | Logic 1 | A plus A (Note 1) |
| H | H | L | H | $A\overline{B}$ | AB plus A | $A+\overline{B}$ | $(A+B)$ plus A |
| H | H | H | L | AB | $A\overline{B}$ minus A | A + B | $(A+\overline{B})$ plus A |
| H | H | H | H | A | A | A | A minus 1 |

**Note 1:** Each bit is shifted to the next most significant position.

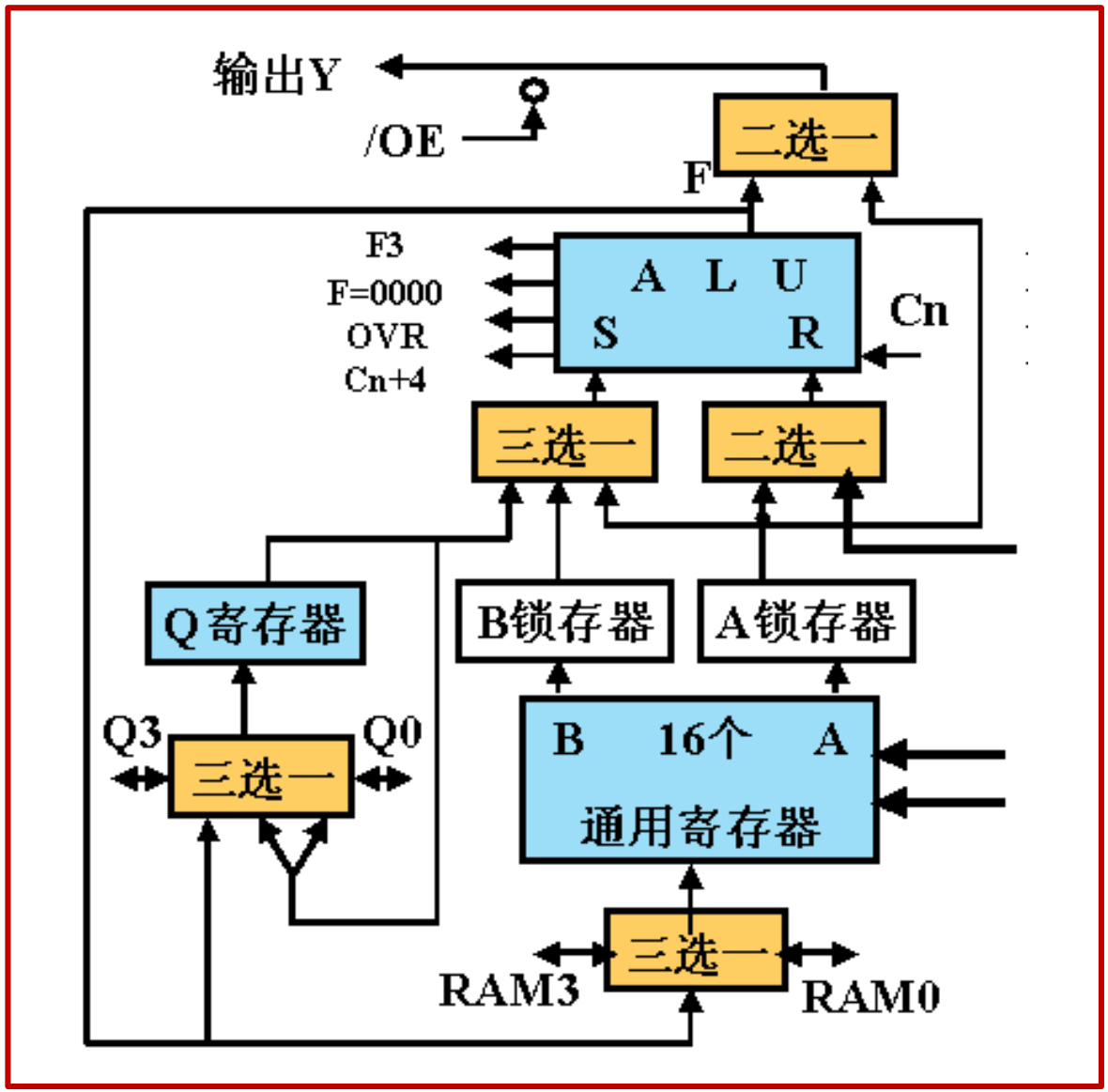
**Note 2:** Arithmetic operations expressed in 2s complement notation.

# 运算单元电路 —— ALU芯片—DM74LS181N

## LogicDiagram



# 运算单元电路 —— ALU芯片— AM2901（4位运算器器件）




## 控制信号

$I_2 I_1 I_0$ 选数据源

$I_5 I_4 I_3$ 选操作功能

$I_8 I_7 I_6$ 选结果安排

# 运算单元电路 —— ALU芯片— AM2901（4位运算器器件）

| 编码 | | | 数据来源 | |
|---|---|---|---|---|
| I2 | I1 | I0 | R | S |
| L | L | L | A | Q |
| L | L | H | A | B |
| L | H | L | 0 | Q |
| L | H | H | 0 | B |
| H | L | L | 0 | A |
| H | L | H | D | A |
| H | H | H | D | Q |
| H | H | H | D | 0 |

| 编码 | | | 运算 |
|---|---|---|---|
| I5 | I4 | I3 | 功能 |
| L | L | L | R + S |
| L | L | H | S − R |
| L | H | L | R − S |
| L | H | H | R ∨ S |
| H | L | L | R ∧ S |
| H | L | H | $\overline{R} \wedge S$ |
| H | H | L | R ∀ S |
| H | H | H | $\overline{R \forall S}$ |

| 编 码 | | | 结 果 处 理 | | |
|---|---|---|---|---|---|
| I8 | I7 | I6 | 通用寄存器组 | Q 寄存器 | Y 输出 |
| L | L | L | | F→Q | F |
| L | L | H | | | F |
| L | H | L | F→B | | A |
| L | H | H | F→B | | F |
| H | L | L | F/2→B | Q/2→Q | F |
| H | L | H | F/2→B | | F |
| H | H | L | 2F→B | 2Q→Q | F |
| H | H | H | 2F→B | | F |

# 第六讲

# 第二部分：组合逻辑

一. 逻辑门电路

二. 布尔代数

三. Verilog HDL介绍

1. Verilog HDL概述
2. Verilog HDL的词法
3. Verilog HDL常用语句
4. 不同抽象级别的Verilog HDL模型

四. 基本组合逻辑部件设计

1. 组合逻辑电路设计概述
2. 运算单元电路
3. 编码器/译码器
4. 多路选择器

# 编码器

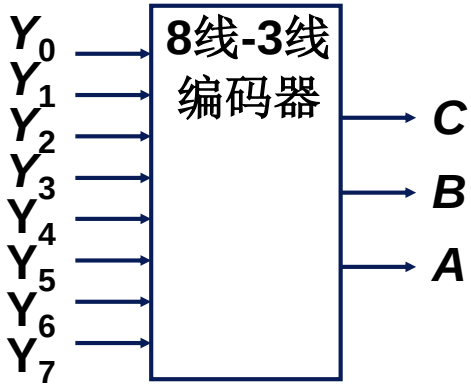
- **编码的含义：**为了区分一系列不同的事物，将其中的每个事物用一组二值（0或1）代码表示；或者说，用二进制代码来表示特定信息。
- **编码：**将加在电路若干输入端中的某一个输入端的信号，变换成相应的一组二进制代码输出的过程。
- **编码器（Encoder）：**实现编码功能的数字电路。
- **编码器的作用：**将某一时刻仅一个输入有效的多个输入的变量情况，用较少的输出状态组合来表达，或者说将输入的每一个高、低电平信号编成一组对应的二进制代码，以便于后续的识别和处理。
- **编码器的类型：**二进制编码器、BCD码编码器、优先编码器

# 二进制编码器

❖**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。

- **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
- 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
- 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。
- 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

# 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。

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

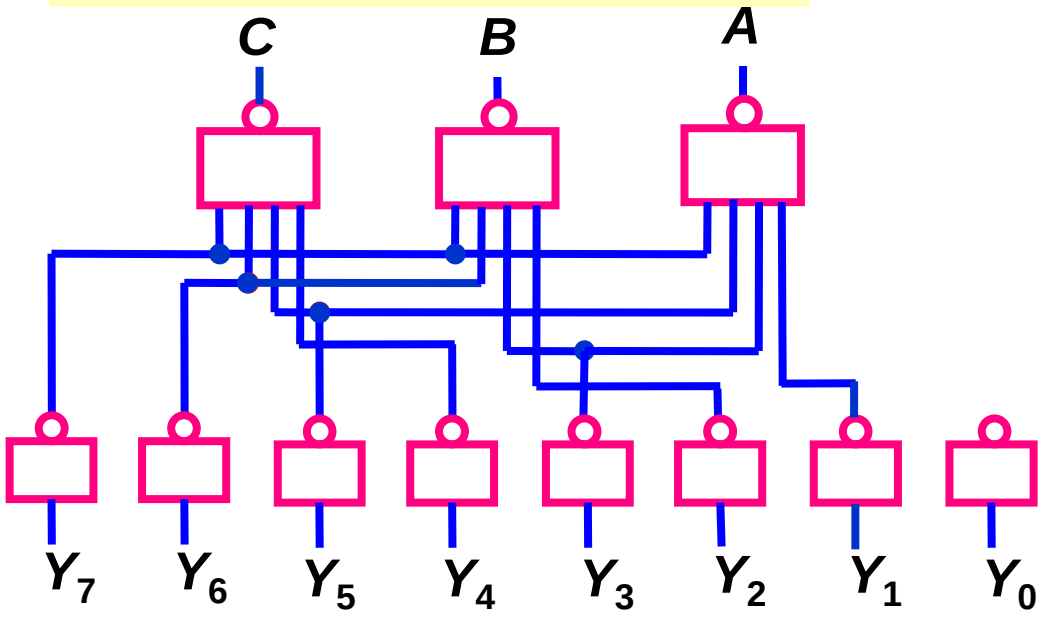
$$B = Y_2 + Y_3 + Y_6 + Y_7 = \overline{\overline{Y_2} \cdot \overline{Y_3} \cdot \overline{Y_6} \cdot \overline{Y_7}}$$

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

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 |

输出的二进制编码，等于对应有效输入的编号



# 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
```

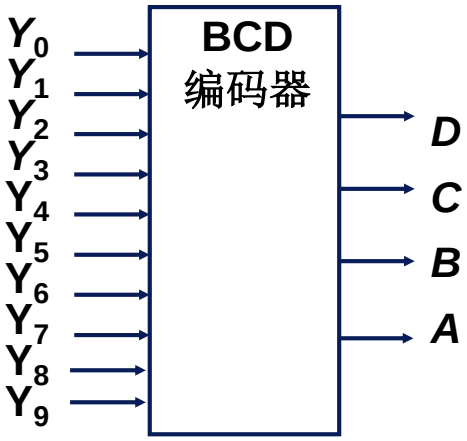
还可以写为？

❖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'bxxx;
   endcase
```

# 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 |

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

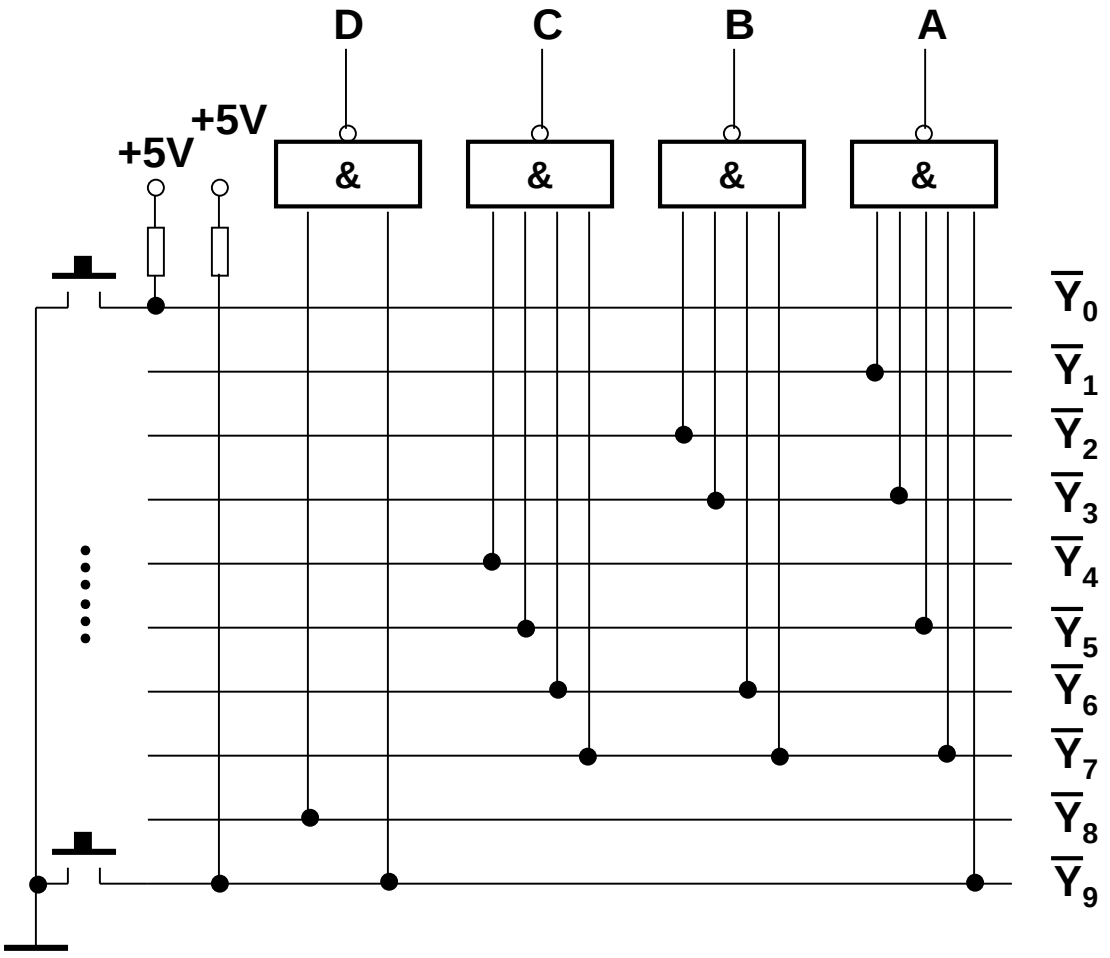
$$C = Y_4 + Y_5 + Y_6 + Y_7 = \overline{\overline{Y_4} \cdot \overline{Y_5} \cdot \overline{Y_6} \cdot \overline{Y_7}}$$

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

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

# 8421BCD编码器的逻辑图

根据逻辑表达式可以直接画出逻辑图




$$D = \overline{\overline{Y}_8 \cdot \overline{Y}_9}$$

$$C = \overline{\overline{Y}_4 \cdot \overline{Y}_5 \cdot \overline{Y}_6 \cdot \overline{Y}_7}$$

$$B = \overline{\overline{Y}_2 \cdot \overline{Y}_3 \cdot \overline{Y}_6 \cdot \overline{Y}_7}$$

$$A = \overline{\overline{Y}_1 \cdot \overline{Y}_3 \cdot \overline{Y}_5 \cdot \overline{Y}_7 \cdot \overline{Y}_9}$$

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=9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
```

将输入最高位写在最左边；标明位宽；用下划线分隔多位数字

# 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

# 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 $\overline{I}_0 \sim \overline{I}_9$，$\overline{I}_9$的优先权最高，$\overline{I}_0$最低。

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

| $\overline{I}_9$ | $\overline{I}_8$ | $\overline{I}_7$ | $\overline{I}_6$ | $\overline{I}_5$ | $\overline{I}_4$ | $\overline{I}_3$ | $\overline{I}_2$ | $\overline{I}_1$ | $\overline{I}_0$ | $\overline{Y}_3$ | $\overline{Y}_2$ | $\overline{Y}_1$ | $\overline{Y}_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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

# 第二部分：组合逻辑

一. 逻辑门电路

二. 布尔代数

三. Verilog HDL介绍

1. Verilog HDL概述
2. Verilog HDL的词法
3. Verilog HDL常用语句
4. 不同抽象级别的Verilog HDL模型

四. 基本组合逻辑部件设计

1. 组合逻辑电路设计概述
2. 运算单元电路
3. 编码器/译码器
4. 多路选择器

# 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）**：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**：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**：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

# 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

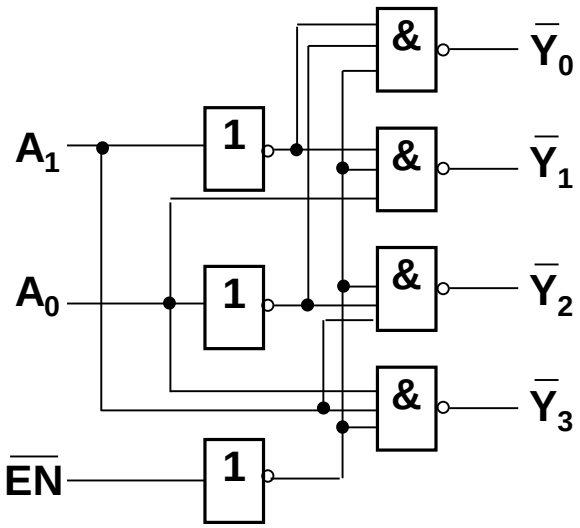
与编码器对应，任何时刻只允许有一个输入为有效电平

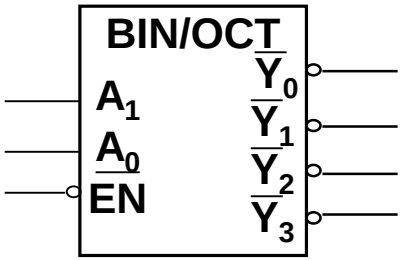
② 高电平输出有效时，每个输出都是对应的输入变量最小项；低电平输出有效时，每个输出都是对应的输入变量最小项的反

二进制译码器也称为最小项译码器。

# 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。




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

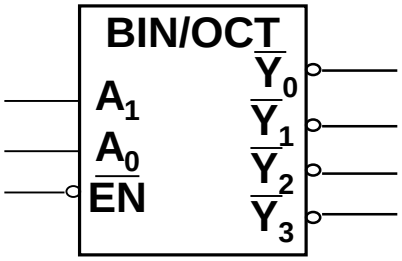
$$\overline{Y}_2 = \overline{A_1 \overline{A}_0} = \overline{m}_2 \qquad \overline{Y}_3 = \overline{A_1 A_0} = \overline{m}_3$$

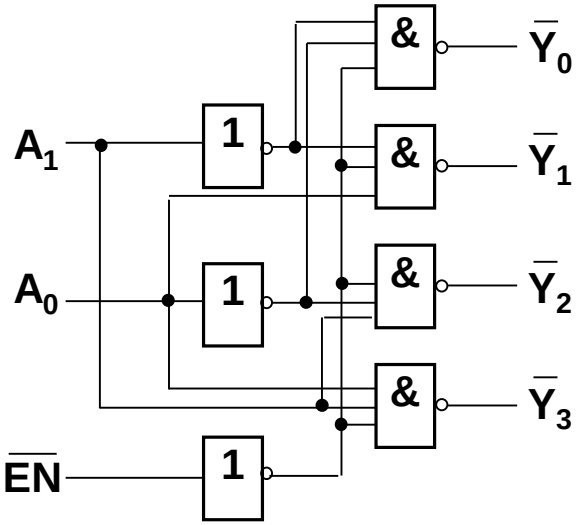
- 当低电平输出有效时，每个输出都是对应的输入变量最小项的反

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




③ 画出逻辑图

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 |

② 根据真值表推导出逻辑表达式（最大项推导法）

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

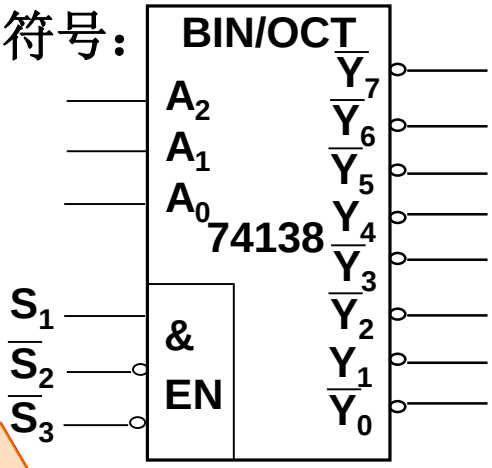
$$\overline{Y}_1 = \overline{\overline{A}_1 A_0} = \overline{m}_1$$

$$\overline{Y}_2 = \overline{A_1 \overline{A_0}} = \overline{m}_2$$

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

# 3线-8线译码器（74138）

逻辑符号：




$S_1$、$\overline{S_2}$、$\overline{S_3}$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

功能表

| $S_1\overline{S_2}\overline{S_3}$ | $A_2$ $A_1$ $A_0$ | $\overline{Y_7}\overline{Y_6}\overline{Y_5}\overline{Y_4}\overline{Y_3}\overline{Y_2}\overline{Y_1}\overline{Y_0}$ |
|---|---|---|
| ≠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 |

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

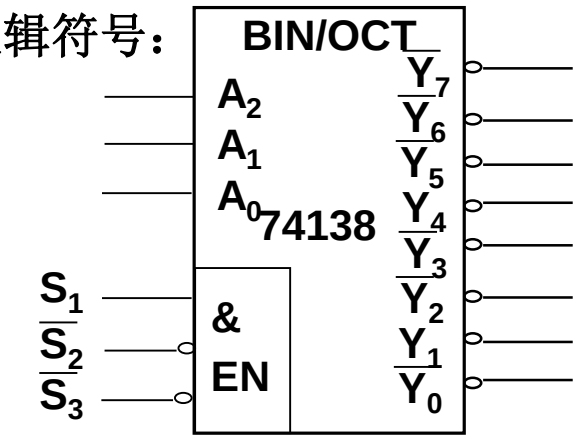
$$\overline{Y_2} = \overline{A_2 A_1 A_0} = \overline{m_2}; \overline{Y_3} = \overline{A_2 A_1 A_0} = \overline{m_3}$$

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

$$\overline{Y_6} = \overline{A_2 A_1 A_0} = \overline{m_6}; \overline{Y_7} = \overline{A_2 A_1 A_0} = \overline{m_7}$$

# 74138的手工设计方法

逻辑符号：




① 真值表

| $S_1\overline{S_2}\overline{S_3}$ | $A_2$ $A_1$ $A_0$ | $\overline{Y_7}\,\overline{Y_6}\,\overline{Y_5}\,\overline{Y_4}\,\overline{Y_3}\,\overline{Y_2}\,\overline{Y_1}\,\overline{Y_0}$ |
|---|---|---|
| ≠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）

③ 画出逻辑图

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

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

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

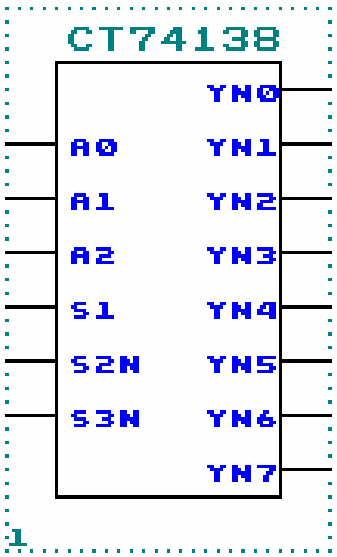
$$\overline{Y_6}=\overline{A_2A_1\overline{A_0}}=\overline{m_6};\overline{Y_7}=\overline{A_2A_1A_0}=\overline{m_7}$$

# 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
```

# 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DCBA出现伪码1010~1111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DCBA＝1010~1111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

# BCD译码器（74xx42）

- 输出$Y_9$~$Y_0$为低电平有效
- 当输入DCBA为无用码（伪码）1010~1111时，译码器输出$Y_9$~$Y_0$均为“1”，不会出现低电平，不会产生错误译码

最高位

| | D C B A | Y9 Y8 Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0 |
|---|---|---|
| 0 | 0 0 0 0 | 1 1 1 1 1 1 1 1 1 0 |
| 1 | 0 0 0 1 | 1 1 1 1 1 1 1 1 0 1 |
| 2 | 0 0 1 0 | 1 1 1 1 1 1 1 0 1 1 |
| 3 | 0 0 1 1 | 1 1 1 1 1 1 0 1 1 1 |
| 4 | 0 1 0 0 | 1 1 1 1 1 0 1 1 1 1 |
| 5 | 0 1 0 1 | 1 1 1 1 0 1 1 1 1 1 |
| 6 | 0 1 1 0 | 1 1 1 0 1 1 1 1 1 1 |
| 7 | 0 1 1 1 | 1 1 0 1 1 1 1 1 1 1 |
| 8 | 1 0 0 0 | 1 0 1 1 1 1 1 1 1 1 |
| 9 | 1 0 0 1 | 0 1 1 1 1 1 1 1 1 1 |
| 不用 | 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 |

# 74xx42的Verilog HDL源程序（1/2）

手工设计：根据真值表推导出逻辑表达式（最大项推导法），画出逻辑图

HDL设计：也可以直接用HDL来描述功能（case语句或assign语句）

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

```
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）

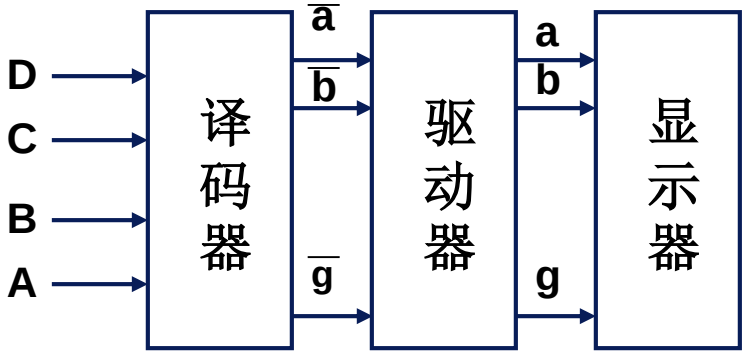
```
always
   begin
      case ({D,C,B,A})
         'b0000 : Y_SIGNAL = 'b1111111110;
         'b0001 : Y_SIGNAL = 'b1111111101;
         'b0010 : Y_SIGNAL = 'b1111111011;
         'b0011 : Y_SIGNAL = 'b1111110111;
         'b0100 : Y_SIGNAL = 'b1111101111;
         'b0101 : Y_SIGNAL = 'b1111011111;
         '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
```

# 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）

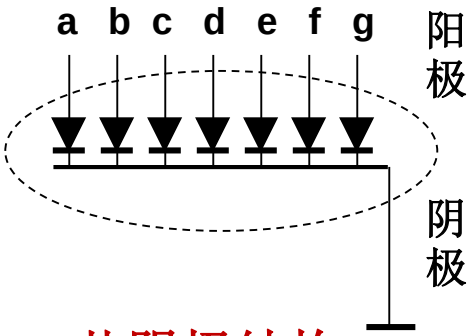



① 显示器件

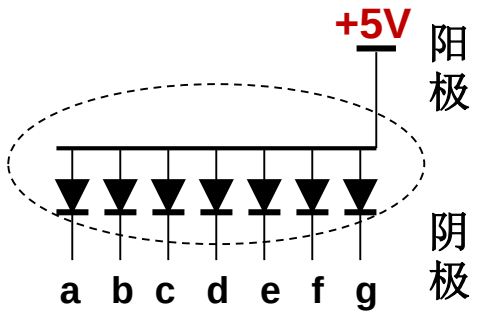
- 辉光数码管、7段荧光数码管、液晶显示器
- 目前广泛使用的显示数字的器件是7段数码显示器（由7段可发光的线段拼合而成），包括半导体数码显示器和液晶显示器两种。
- 数码显示器人们通常又称为数码管。半导体7段数码管实际上是由7个发光二极管（LED）构成的，因此又称为LED数码管。

# 半导体7段数码管


半导体7段数码管




共阴极结构



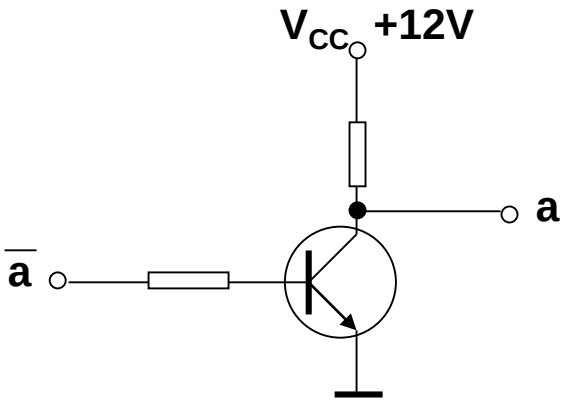

共阳极结构

- 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。

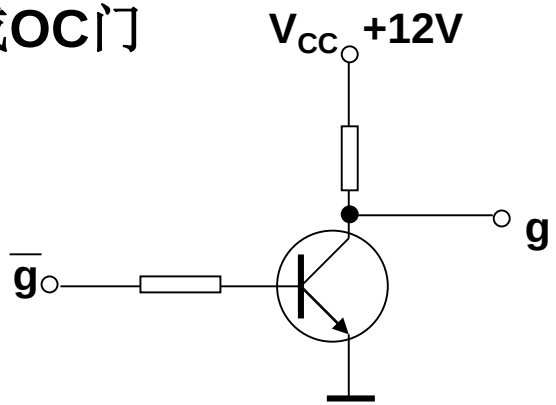
# 驱动电路和译码器

## ② 驱动电路

使译码器输出反相，并提供大的驱动电流。

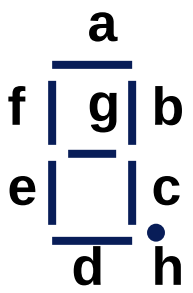



反相器
或OC门




## 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 | |



# 第二部分：组合逻辑

一. 逻辑门电路

二. 布尔代数

三. Verilog HDL介绍

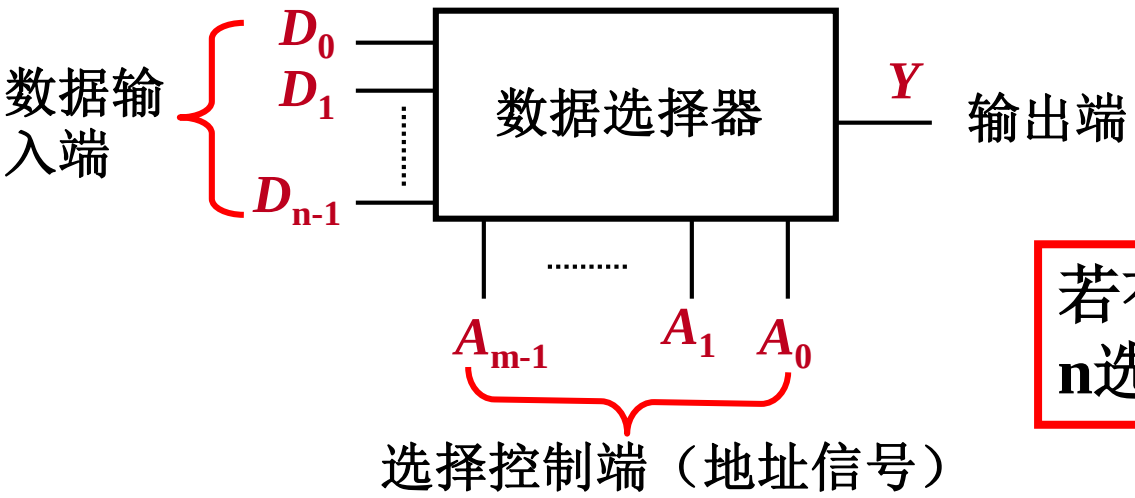
1. Verilog HDL概述
2. Verilog HDL的词法
3. Verilog HDL常用语句
4. 不同抽象级别的Verilog HDL模型

四. 基本组合逻辑部件设计

1. 组合逻辑电路设计概述
2. 运算单元电路
3. 编码器/译码器
4. 多路选择器

# 数据选择器

- 从一组输入数据选出其中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**多路选择器**（**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。

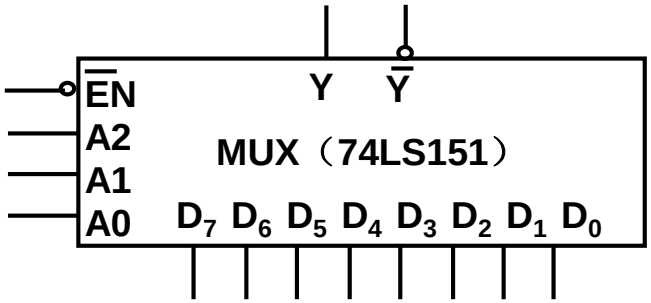


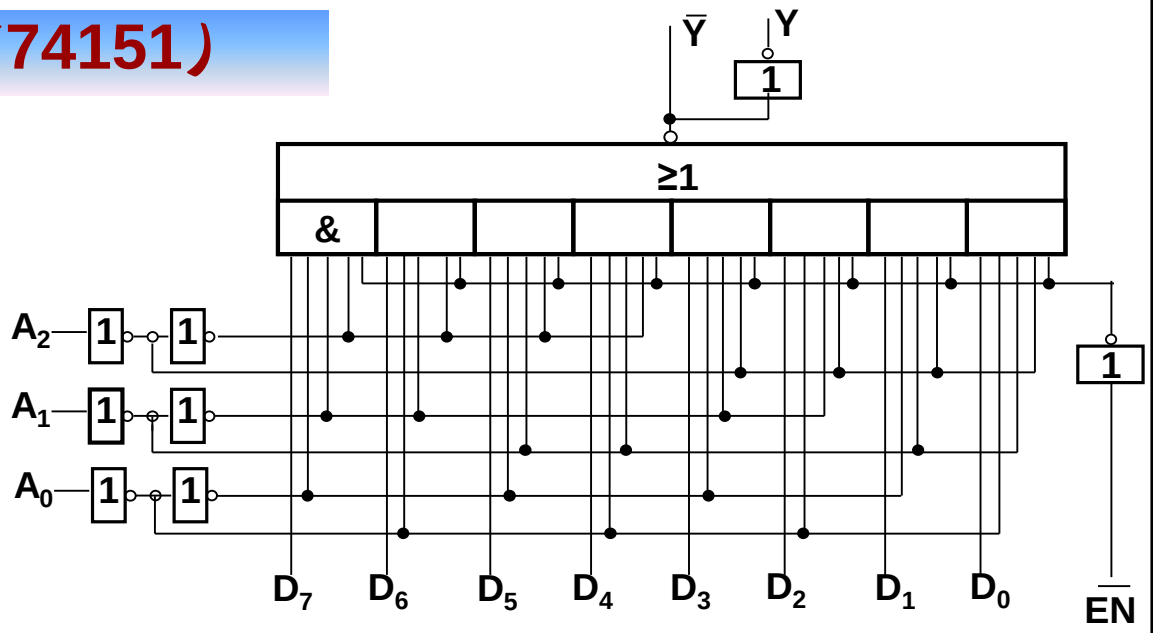

若有n个输入，则称为n选1数据选择器。

# 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




$$Y = \overline{A_2}\,\overline{A_1}\,\overline{A_0}D_0 + \overline{A_2}\,\overline{A_1}A_0D_1 + \overline{A_2}A_1\overline{A_0}D_2 + \overline{A_2}A_1A_0D_3 + A_2\overline{A_1}\,\overline{A_0}D_4 + A_2\overline{A_1}A_0D_5 + A_2A_1\overline{A_0}D_6 + A_2A_1A_0D_7 \quad (\overline{EN} = 0)$$

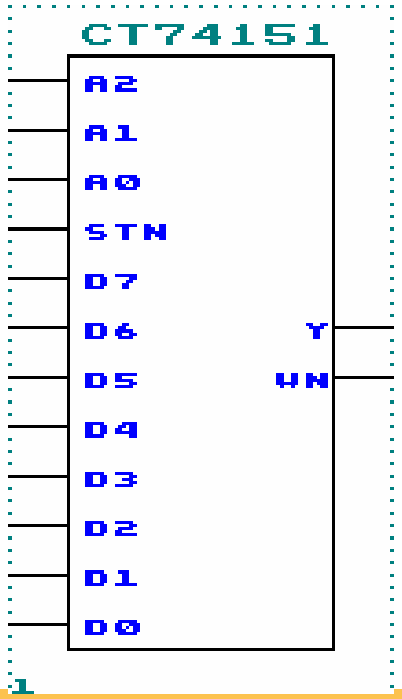
$$= m_0D_0 + m_1D_1 + m_2D_2 + m_3D_3 + m_4D_4 + m_5D_5 + m_6D_6 + m_7D_7$$

# 8选1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：



# 74151的Verilog HDL源程序

## ❖HDL设计

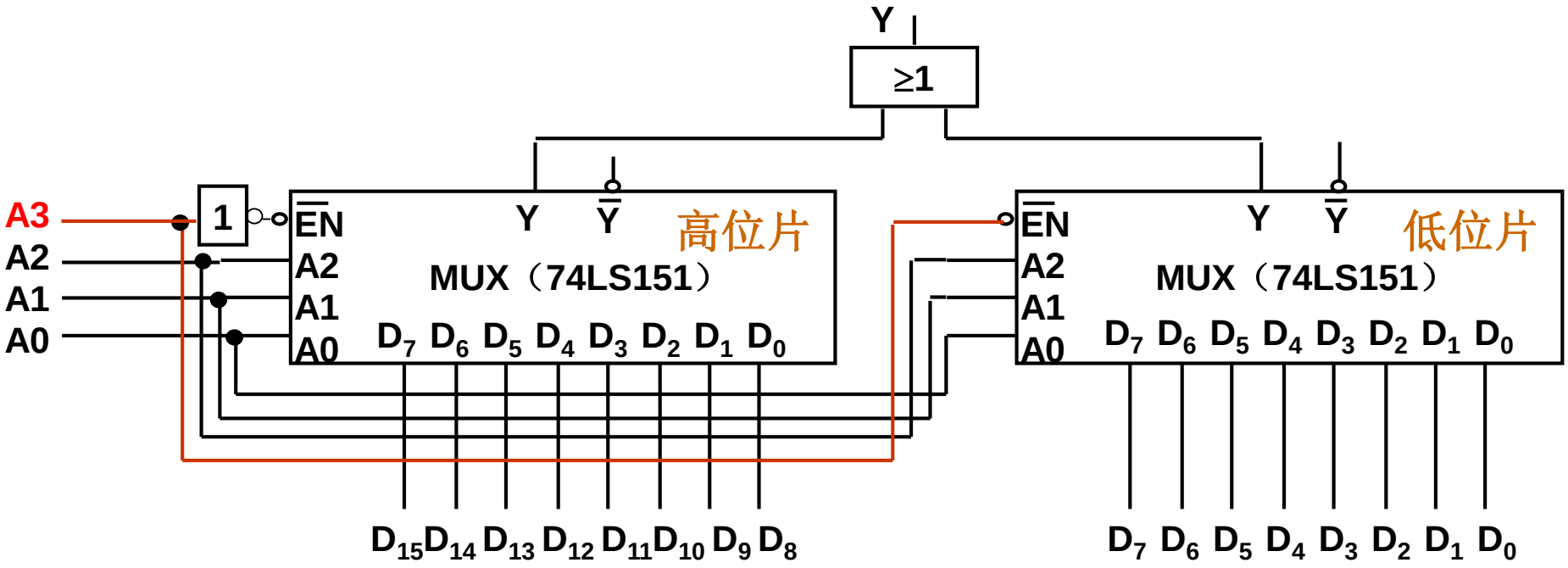
采用if-else语句和case语句描述

```
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
```

# 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。

# 数据选择器的功能（1/2）

❖功能①

$A_2A_1A_0$为控制端——

8选1数据选择器（8路开关）

在/EN=0时

$$Y=\overline{A_2}\,\overline{A_1}\,\overline{A_0}D_0+\overline{A_2}\,\overline{A_1}A_0D_1+\overline{A_2}A_1\overline{A_0}D_2+\overline{A_2}A_1A_0D_3+A_2\overline{A_1}\,\overline{A_0}D_4+A_2\overline{A_1}A_0D_5+A_2A_1\overline{A_0}D_6+A_2A_1A_0D_7\ (\overline{EN}=0)$$

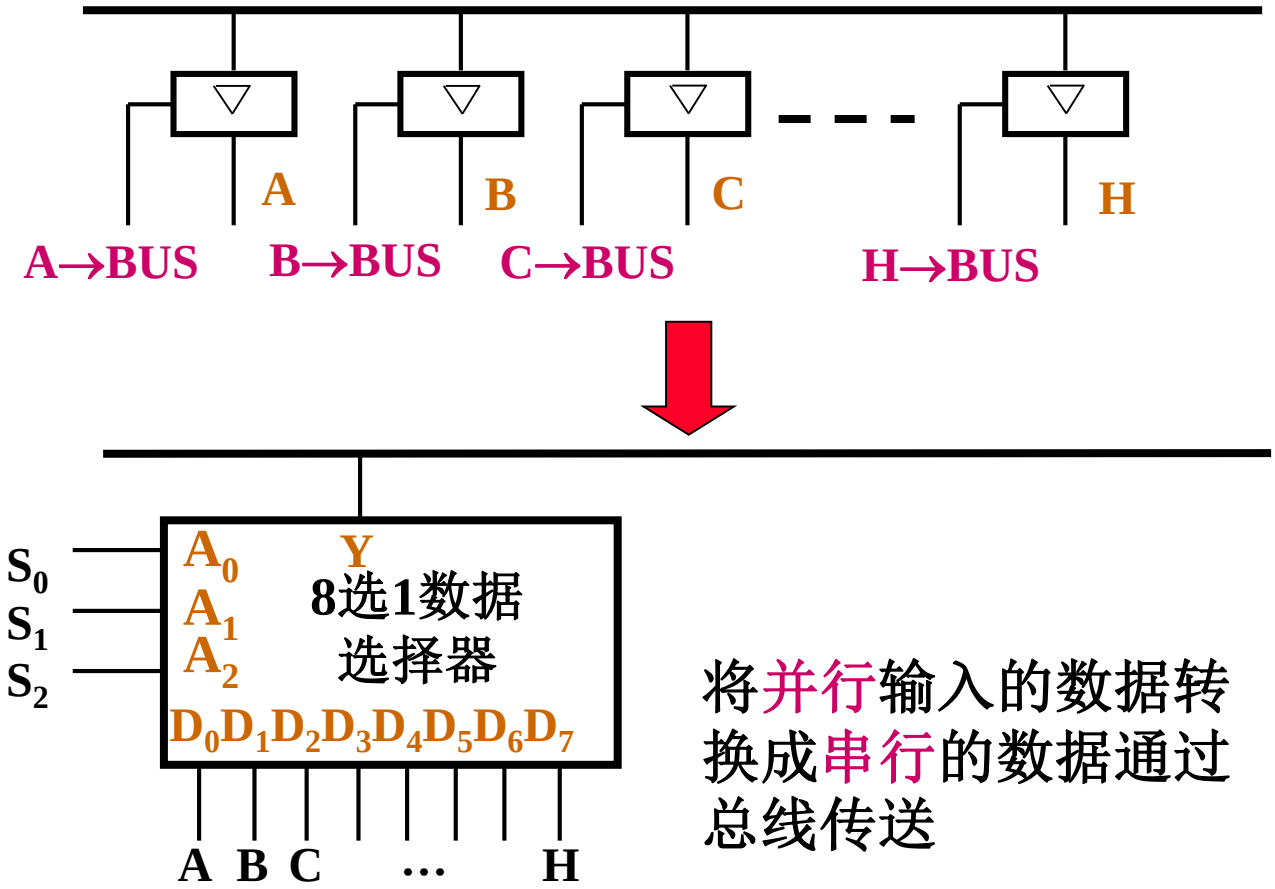
$$=m_0D_0+m_1D_1+m_2D_2+m_3D_3+m_4D_4+m_5D_5+m_6D_6+m_7D_7$$

功能表（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 | $D_7$ |

# 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 （并串转换）




| $S_2S_1S_0$ | 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 |

将并行输入的数据转换成串行的数据通过总线传送

# 数据选择器的功能（2/2）

## 功能 ②： $D_7 \sim D_0$为控制端——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。

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

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

# 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。

## 利用数据选择器实现逻辑函数

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

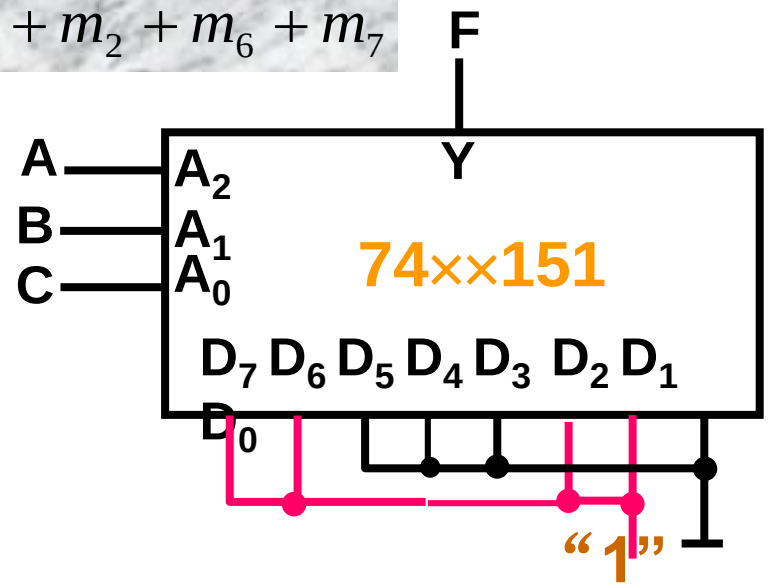
解：使用8选1数据选择器74151

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：

$D_0=0$，$D_1=1$，$D_2=1$，$D_3=0$，

$D_4=0$，$D_5=0$，$D_6=1$，$D_7=1$