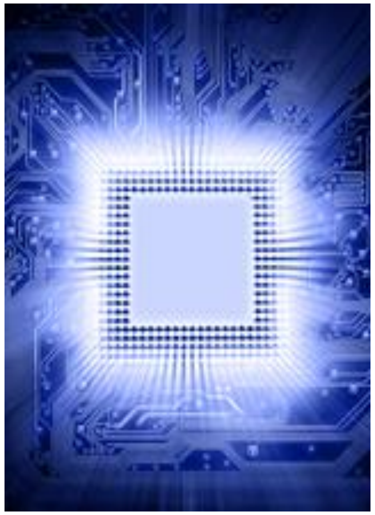
# 第六章 组合逻辑设计实践

武港山

南京大学人工智能学院



# 主要内容

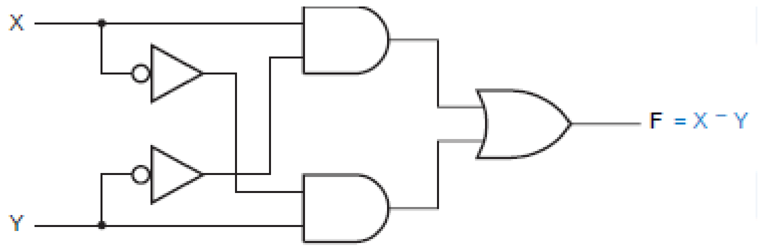
- 设计文档编制标准
- 电路的定时
- 组合PLD的内部结构
  - 译码器
  - 编码器
  - 三态器件
  - 多路复用器
  - 校验电路
  - 比较器
  - 加法器、减法器和ALU
  - 组合乘法器

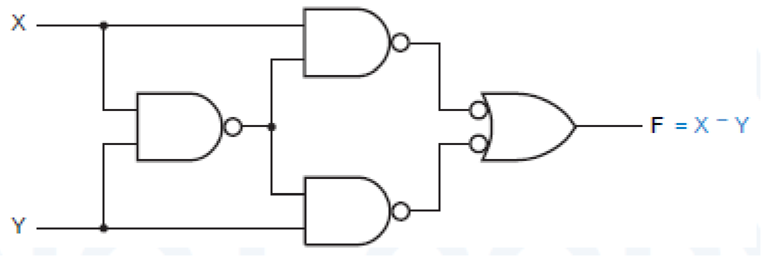
# 3.6 校验电路

- 异或门和异或非门
  - 函数： $X \oplus Y = X' \cdot Y + X \cdot Y'$
  - 真值表：

| $X$ | $Y$ | $X \oplus Y$ (XOR) | $(X \oplus Y)'$ (XNOR) |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

  - 电路实现：与或门，三级与非门





# 3.6 校验电路

- 异或门和异或非门
  - 符号：
    - 根据“圈到圈”设计原则，选用合适功能表达的符号。

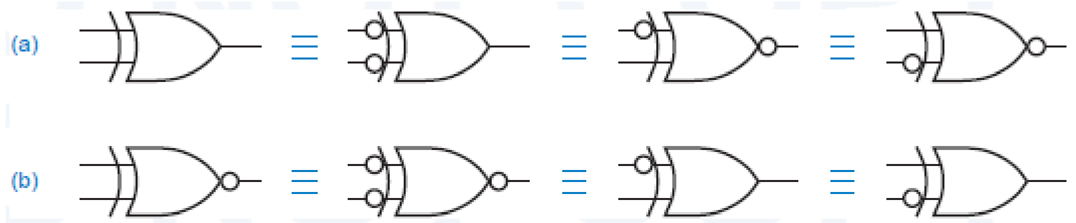
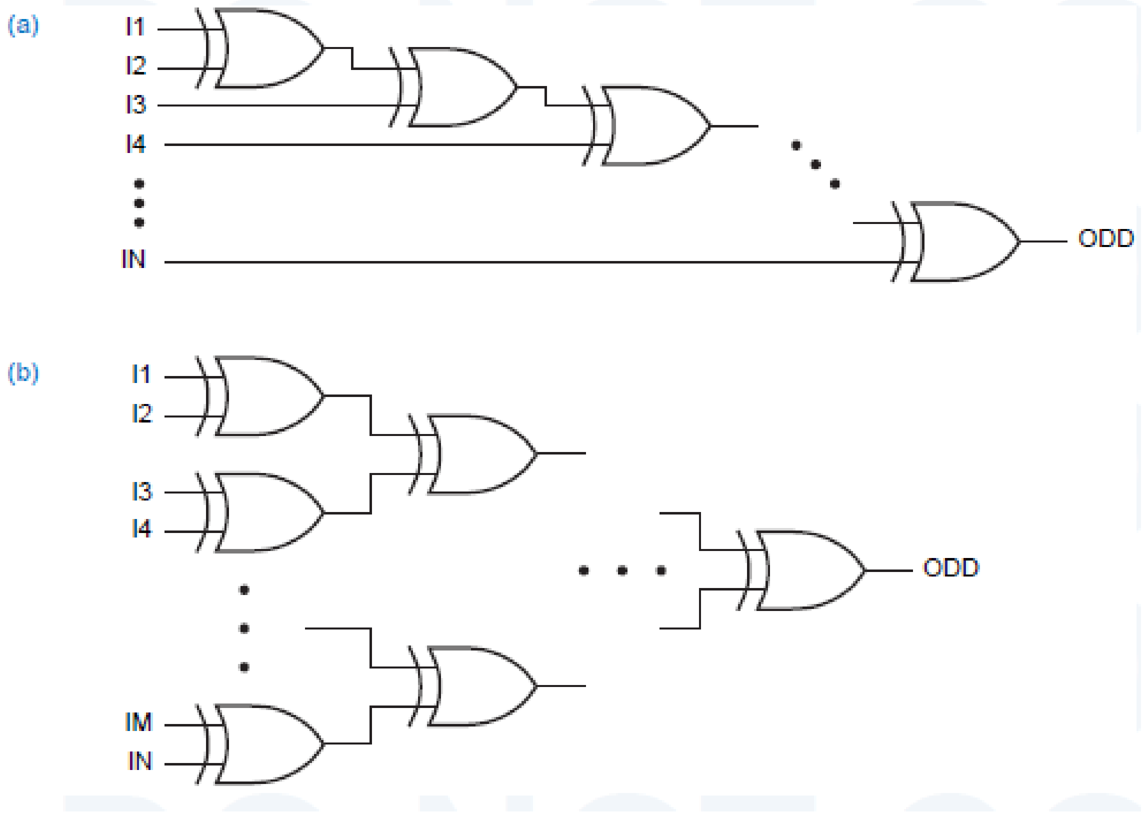


Figure 5-71 Equivalent symbols for (a) XOR gates; (b) XNOR gates.

  - 许多PLD和CPLD中除了提供输出极性控制的异或门外，还提供乘积项的异或处理配置。
  - FPGA中可以直接使用原语设计。
    - 计数器中特别有用。

## 3.6 校验电路

- 奇偶校验电路
  - 检测信息源中1的个数。
    - 奇校验
    - 偶校验
  - 基本方法:
    - 奇校验



## 3.6 校验电路

- 奇偶校验器件74x280
  - 9个输入
  - 2个输出

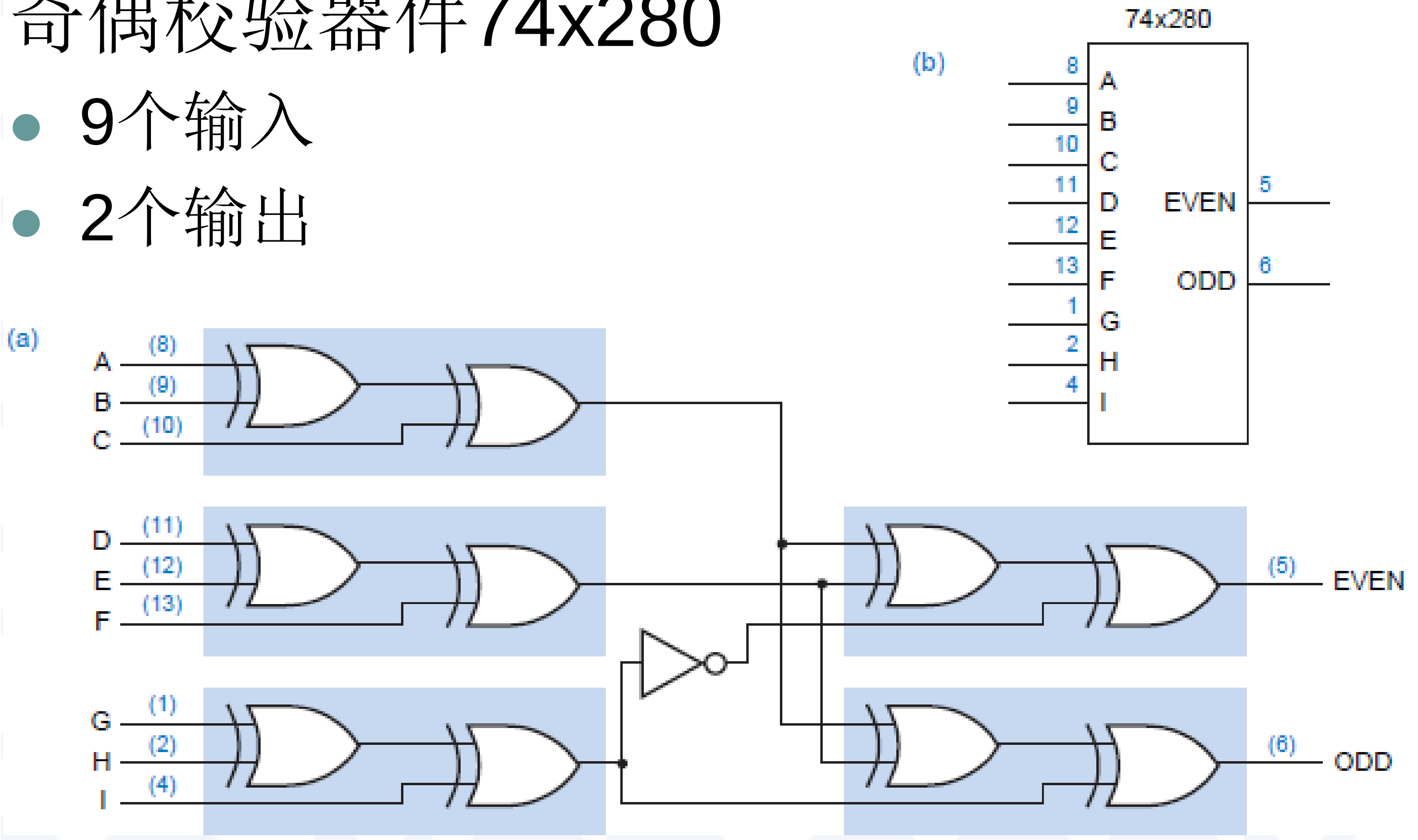
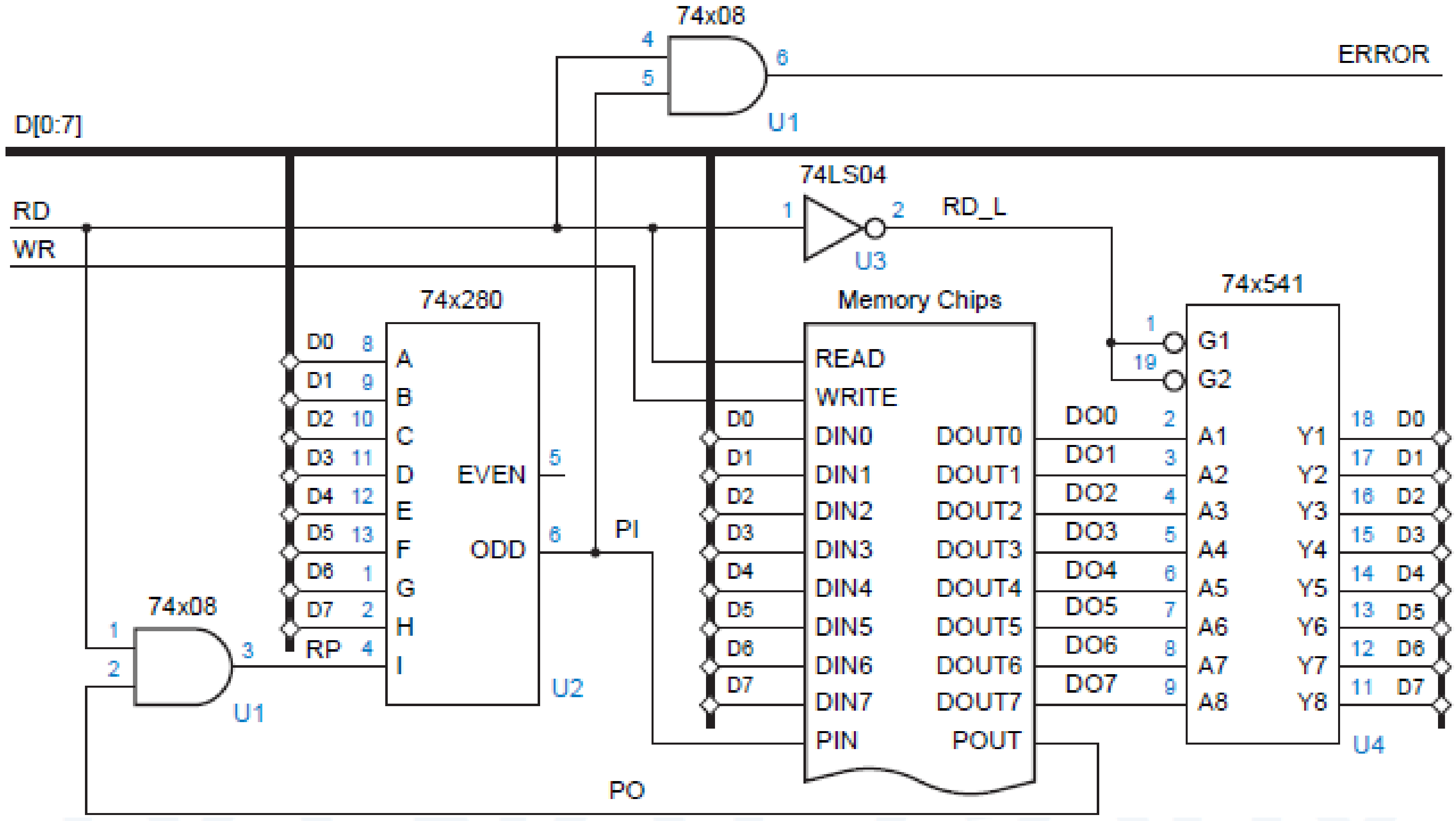


**Figure 5-74** The 74x280 9-bit odd/even parity generator: (a) logic diagram, including pin numbers for a standard 16-pin dual in-line package; (b) traditional logic symbol.

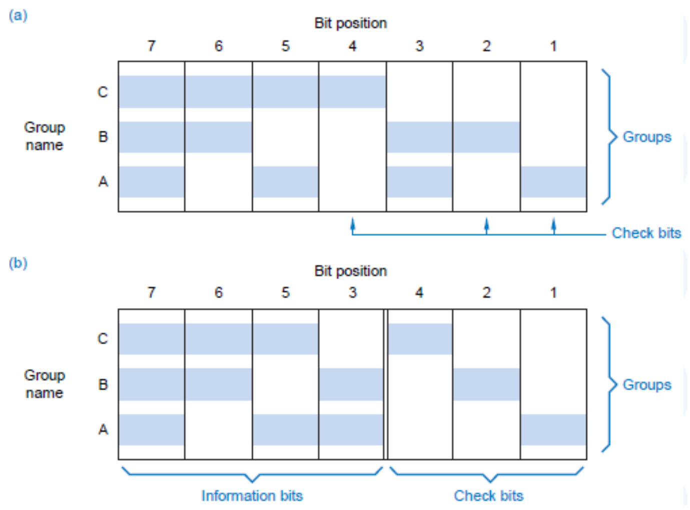
# 3.6 校验电路

- 奇偶校验应用
  - 检测数据存储过程中是否存在错误。



# 3.6 校验电路

- 奇偶校验应用
  - 7位海明码纠错电路。



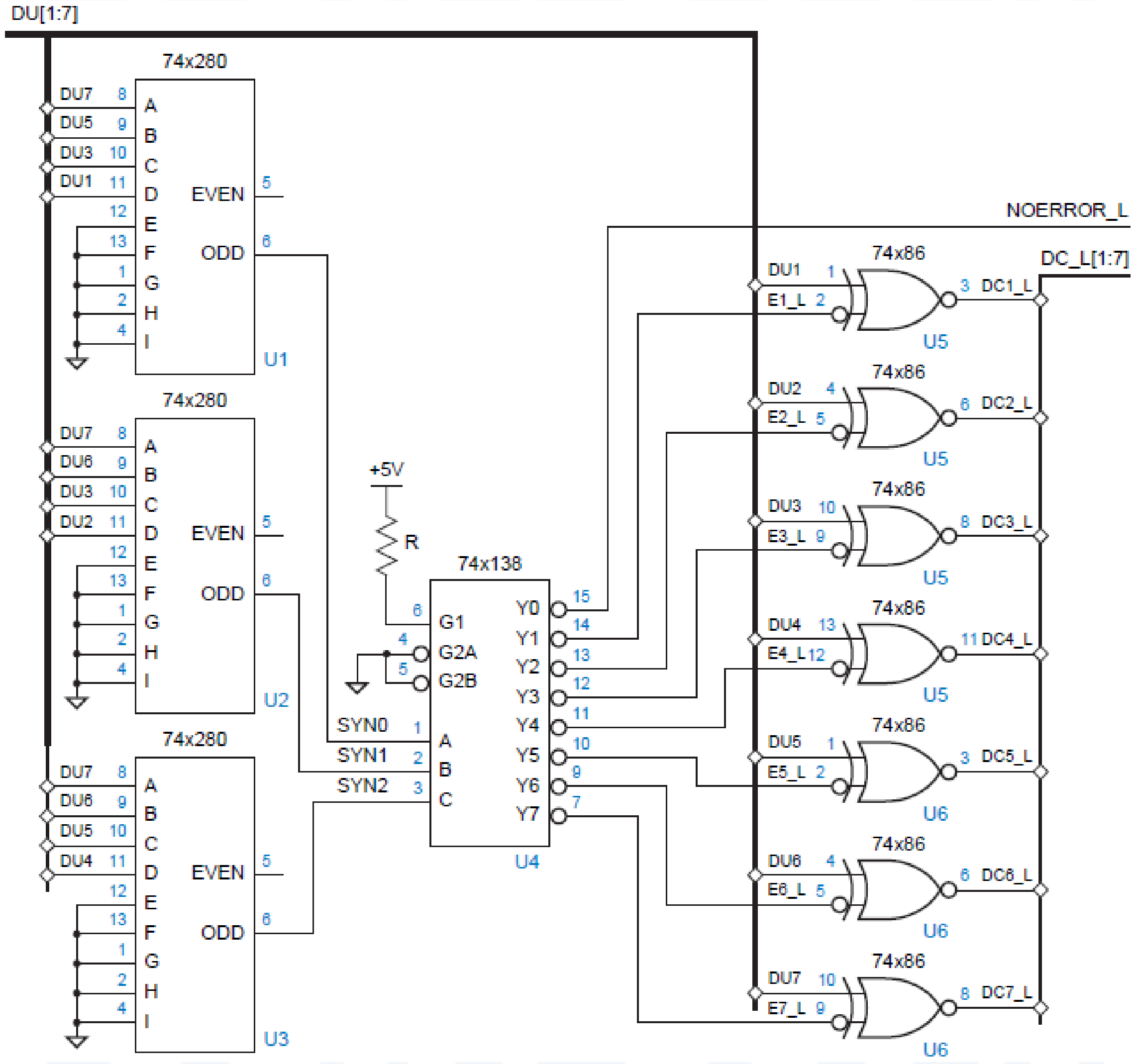


**Figure 5-76** Error-correcting circuit for a 7-bit Hamming code.

## 3.6 校验电路

- Verilog实现
  - 异或有专门的操作符

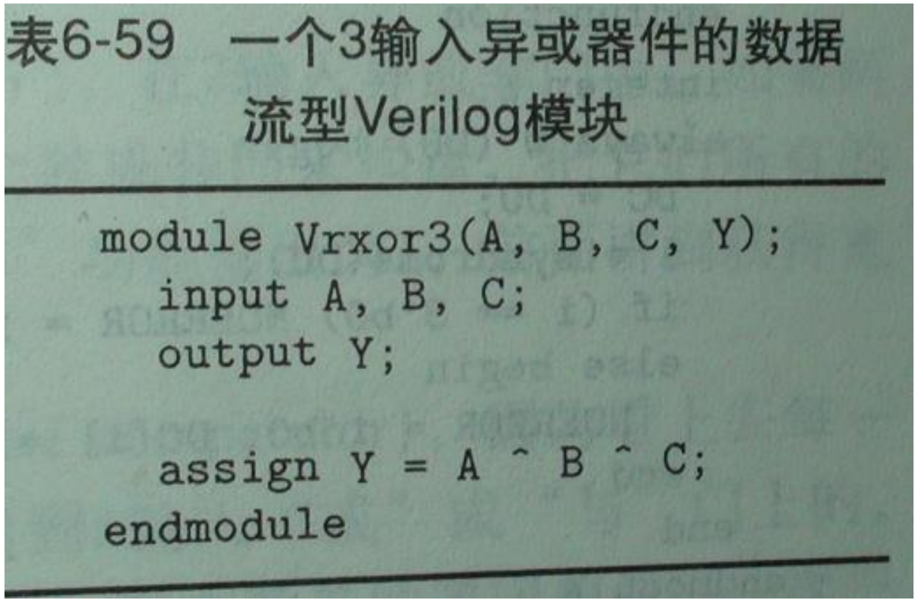


表6-59 一个3输入异或器件的数据流型Verilog模块

```
module Vrxor3(A, B, C, Y);
  input A, B, C;
  output Y;

  assign Y = A ^ B ^ C;
endmodule
```

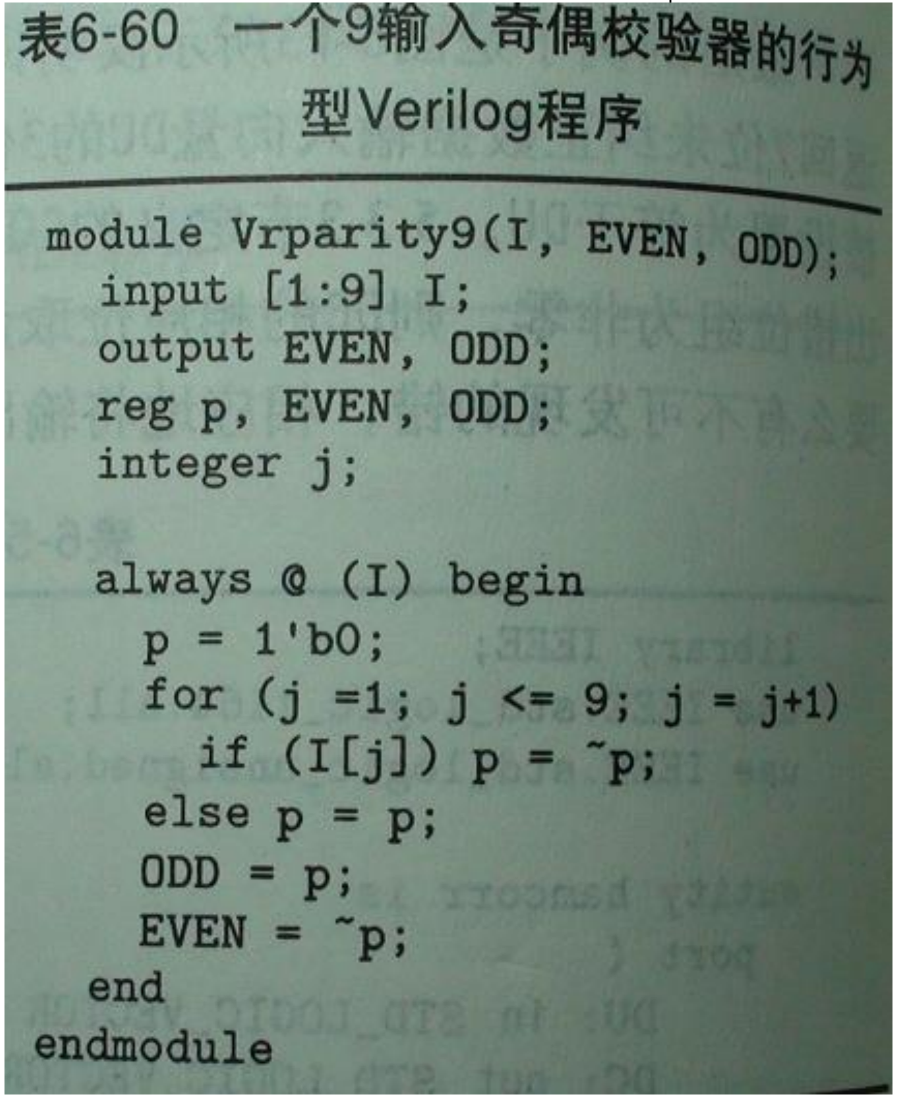


表6-60 一个9输入奇偶校验器的行为型Verilog程序

```
module Vrparity9(I, EVEN, ODD);
  input [1:9] I;
  output EVEN, ODD;
  reg p, EVEN, ODD;
  integer j;

  always @ (I) begin
    p = 1'b0;
    for (j =1; j <= 9; j = j+1)
      if (I[j]) p = ~p;
    else p = p;
    ODD = p;
    EVEN = ~p;
  end
endmodule
```

## 3.6 校验电路

- Verilog实现

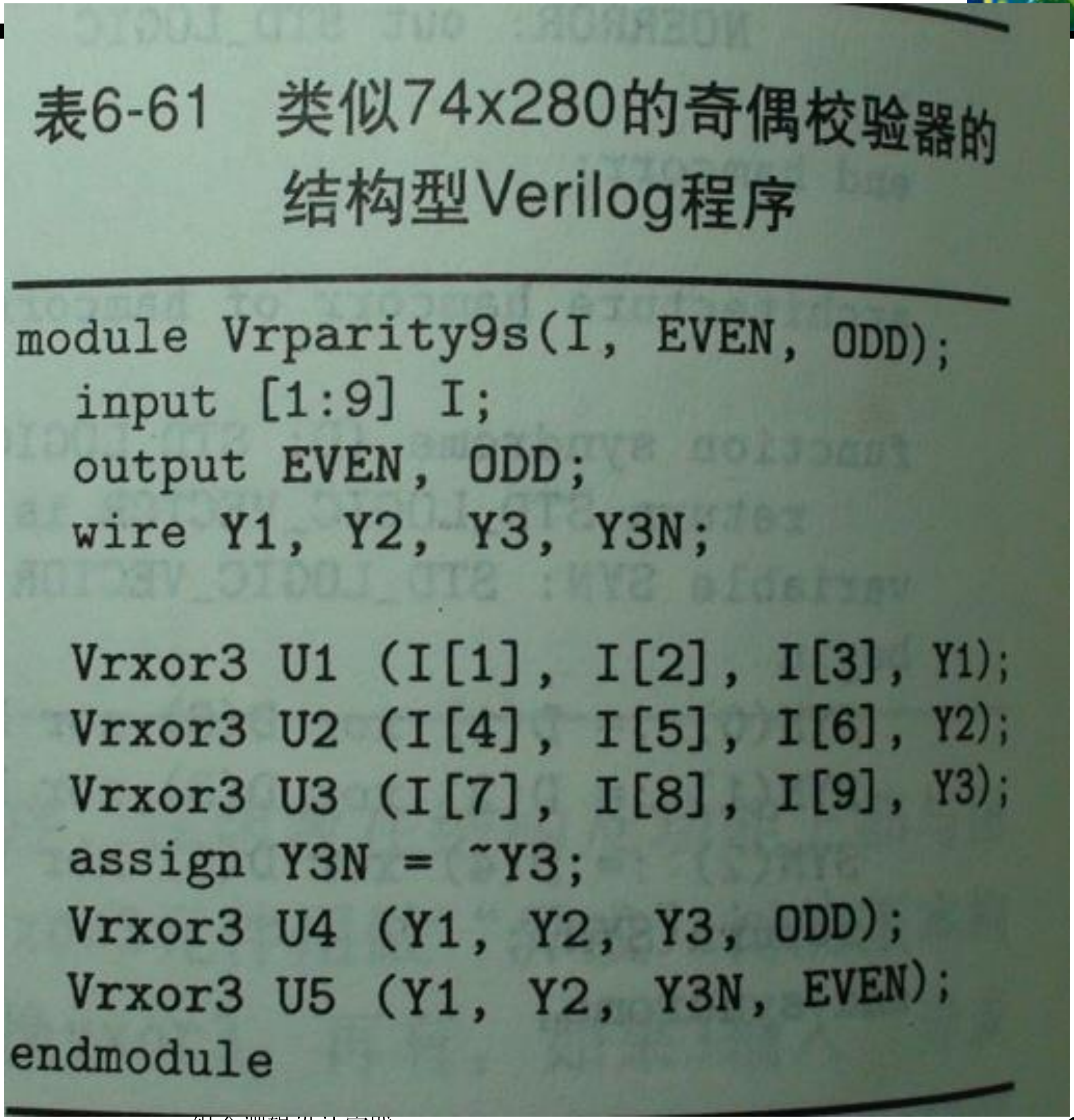


表6-61 类似74x280的奇偶校验器的结构型Verilog程序

```
module Vrparity9s(I, EVEN, ODD);
  input [1:9] I;
  output EVEN, ODD;
  wire Y1, Y2, Y3, Y3N;

  Vrxor3 U1 (I[1], I[2], I[3], Y1);
  Vrxor3 U2 (I[4], I[5], I[6], Y2);
  Vrxor3 U3 (I[7], I[8], I[9], Y3);
  assign Y3N = ~Y3;
  Vrxor3 U4 (Y1, Y2, Y3, ODD);
  Vrxor3 U5 (Y1, Y2, Y3N, EVEN);
endmodule
```

# 3.6 校验电路

●

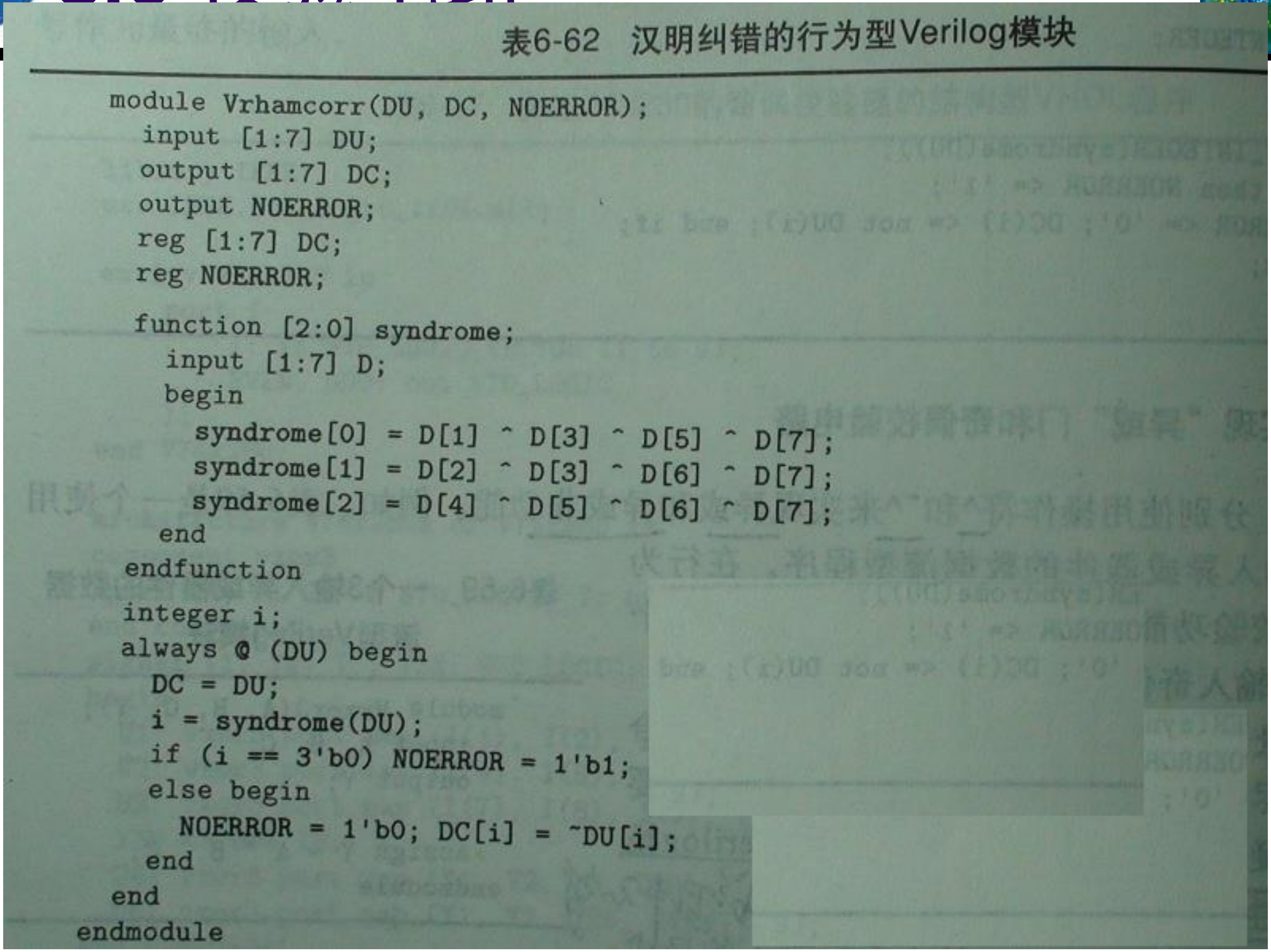
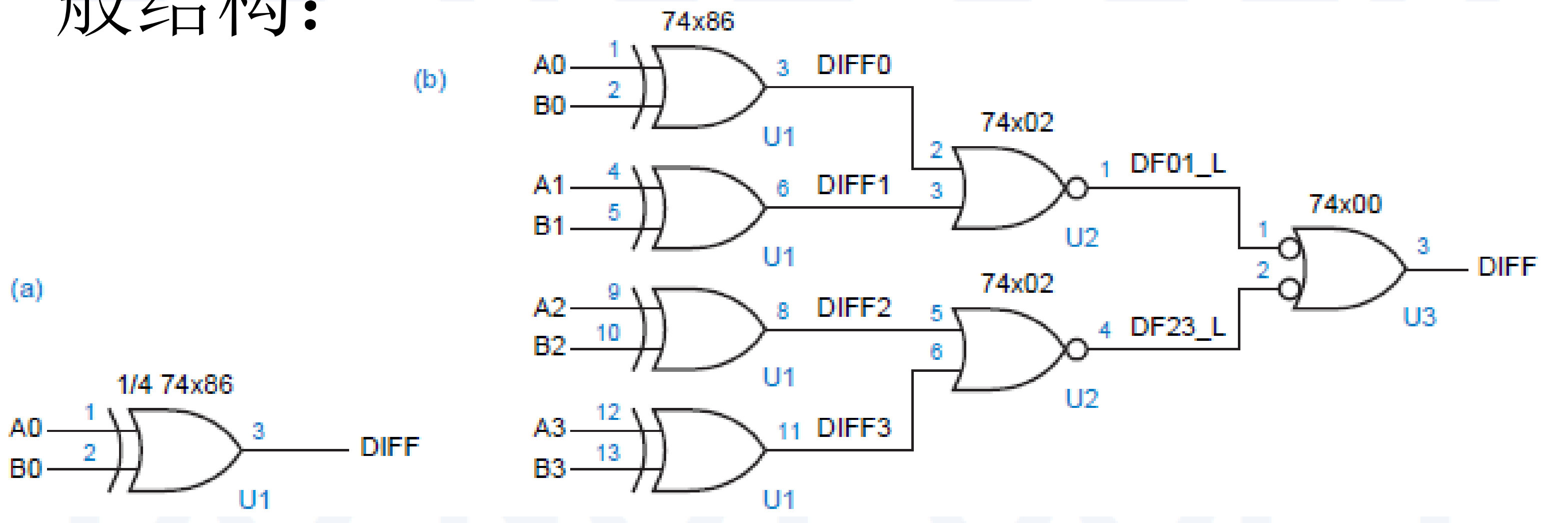


表6-62　汉明纠错的行为型Verilog模块

```
module Vrhamcorr(DU, DC, NOERROR);
  input [1:7] DU;
  output [1:7] DC;
  output NOERROR;
  reg [1:7] DC;
  reg NOERROR;

  function [2:0] syndrome;
    input [1:7] D;
    begin
      syndrome[0] = D[1] ^ D[3] ^ D[5] ^ D[7];
      syndrome[1] = D[2] ^ D[3] ^ D[6] ^ D[7];
      syndrome[2] = D[4] ^ D[5] ^ D[6] ^ D[7];
    end
  endfunction

  integer i;
  always @ (DU) begin
    DC = DU;
    i = syndrome(DU);
    if (i == 3'b0) NOERROR = 1'b1;
    else begin
      NOERROR = 1'b0; DC[i] = ~DU[i];
    end
  end
endmodule
```

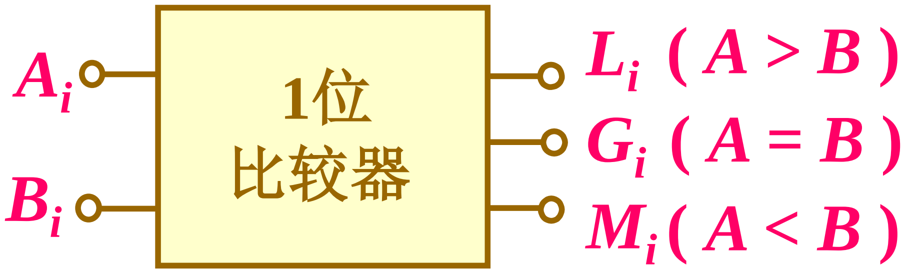
# 3.7 比较器

- 两种类型：
  - 比较器
  - 大小比较器
- 一般结构：



# 数据比较器

## 1 位数值比较器



真值表

| $A_i$ | $B_i$ | $L_i$ | $G_i$ | $M_i$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |

函数式

$$L_i = A_i\overline{B_i} \qquad G_i = \overline{A_i}\,\overline{B_i} + A_iB_i$$

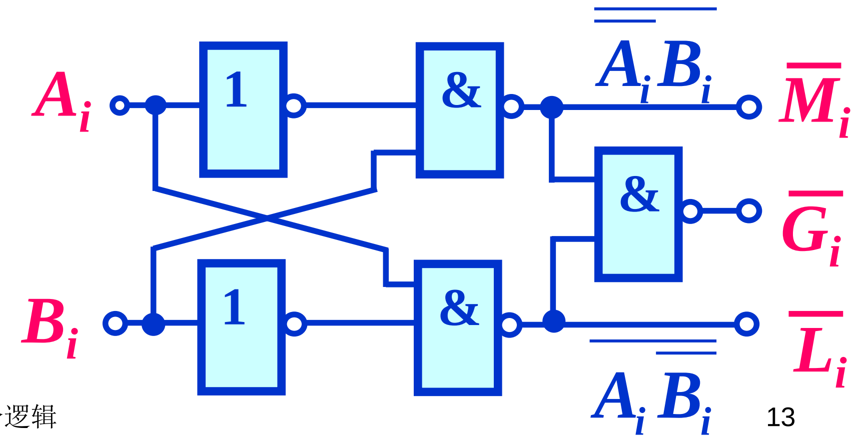
$$M_i = \overline{A_i}B_i \qquad = A_i \odot B_i$$

$$\overline{L_i} = \overline{A_i\overline{B_i}} \qquad \overline{G_i} = \overline{\overline{A_i\overline{B_i}} \cdot \overline{\overline{A_i}B_i}}$$

$$\overline{M_i} = \overline{\overline{A_i}B_i}$$

逻辑图

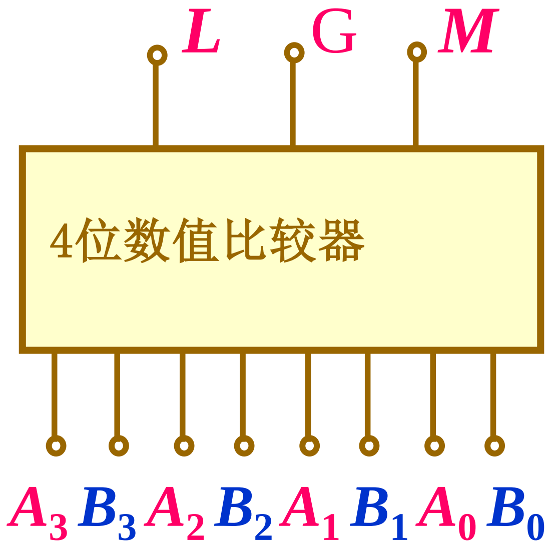
— 用与非门和非门实现



# 数据比较器

## 4 位数值比较器

$A = A_3A_2A_1A_0 \quad B = B_3B_2B_1B_0$



真值表

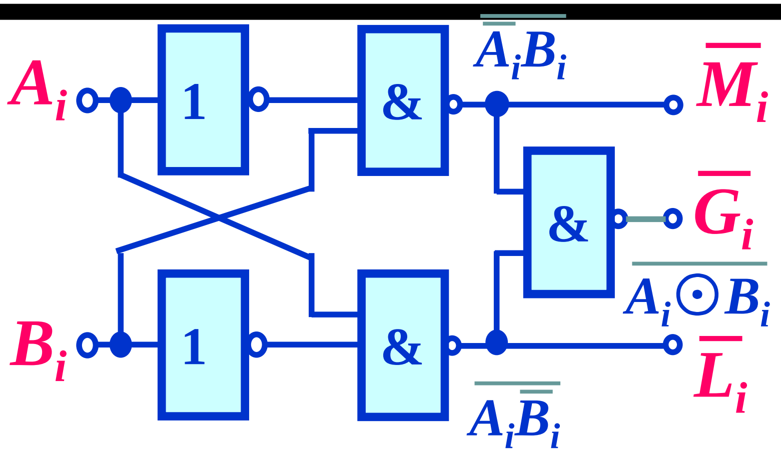
| 比较输入 | | | | 输出 | | |
|---|---|---|---|---|---|---|
| $A_3$ $B_3$ | $A_2$ $B_2$ | $A_1$ $B_1$ | $A_0$ $B_0$ | $L$ | $G$ | $M$ |
| > | × | × | × | 1 | 0 | 0 |
| = | > | × | × | 1 | 0 | 0 |
| = | = | > | × | 1 | 0 | 0 |
| = | = | = | > | 1 | 0 | 0 |
| = | = | = | = | 0 | 1 | 0 |
| < | × | × | × | 0 | 0 | 1 |
| = | < | × | × | 0 | 0 | 1 |
| = | = | < | × | 0 | 0 | 1 |
| = | = | = | < | 0 | 0 | 1 |

$A > B \quad L = 1$

$A = B \quad G = 1$

$A < B \quad M = 1$

## 1 位数值比较器

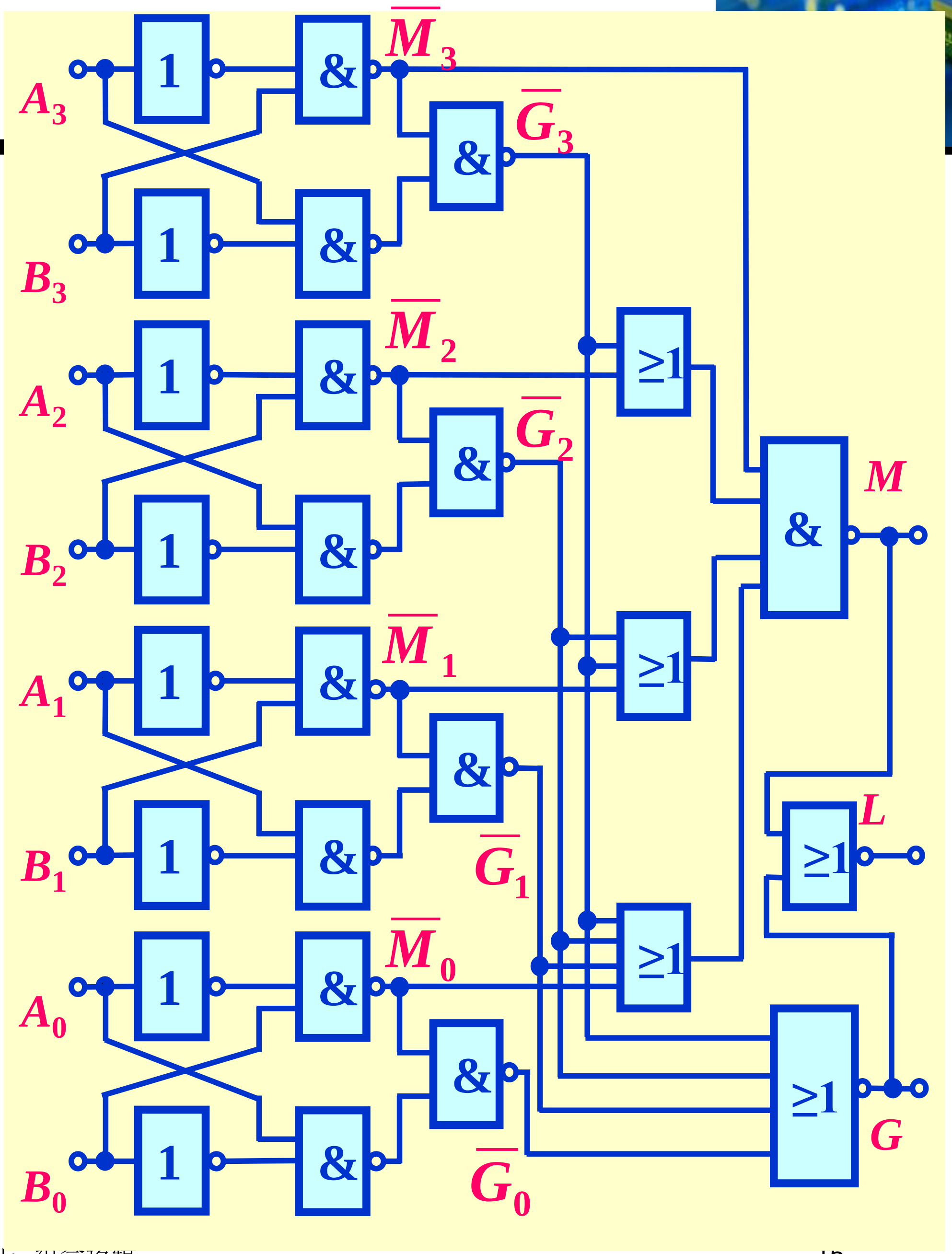


## 4 位数值比较器

$$M = \bar{A}_3B_3 + (A_3 \odot B_3)\,\bar{A}_2B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\,\bar{A}_1\,B_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\,\bar{A}_0B_0$$

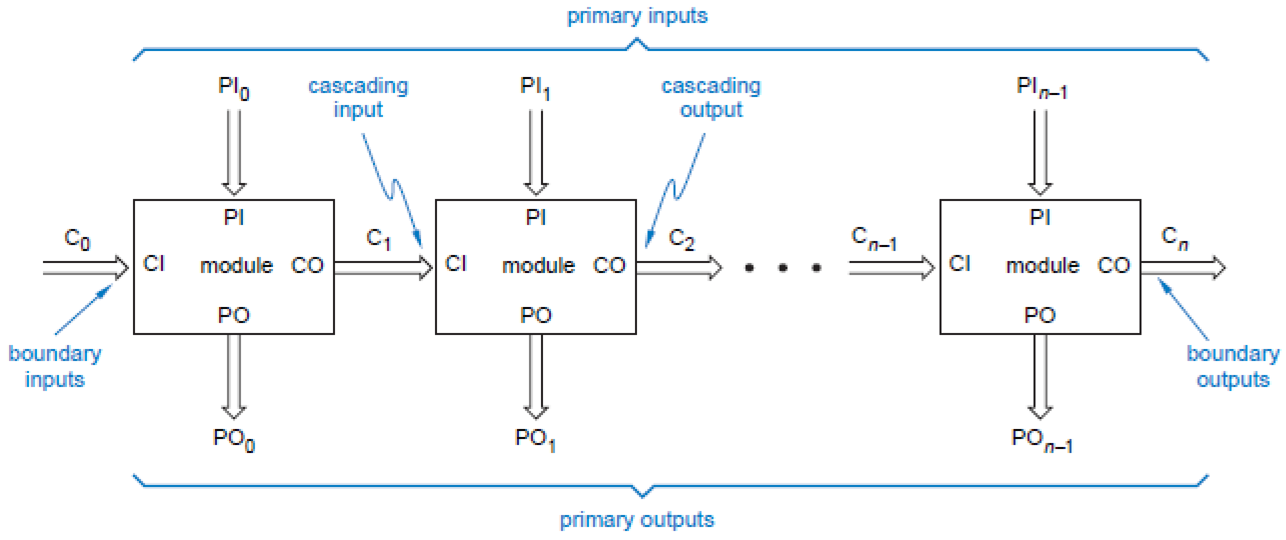
$$G = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

$$L = \overline{M+G}$$



# 3.7 比较器

- 迭代电路
  - 形态：



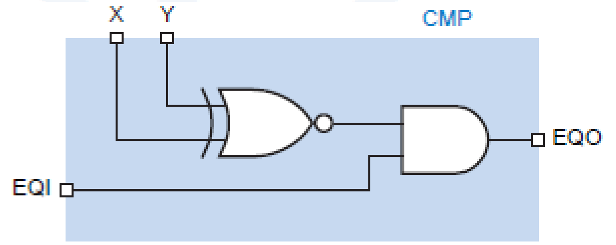
  - 特点：
    - 每个模块都有主输入、主输出、级联输入、级联输出。
    - 最左边有级联输入叫边界输入；最右边的级联输出叫边界输出。

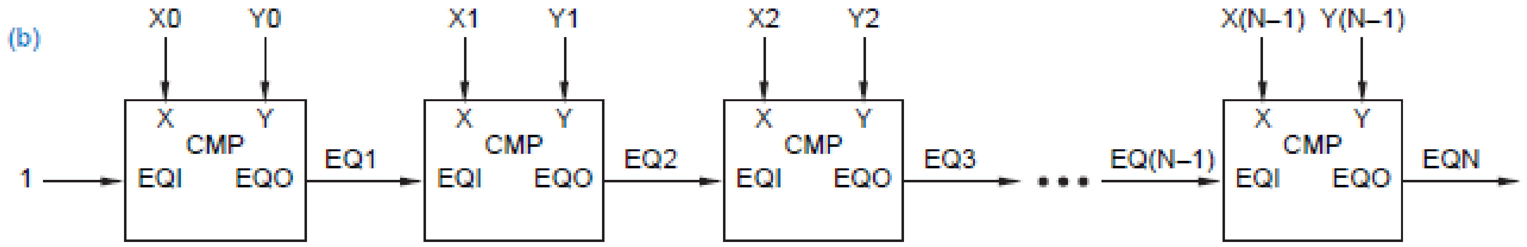
# 3.7 比较器

- 迭代电路
  - 算法：

    1. Set $C_0$ to its initial value and set $i$ to 0.
    2. Use $C_i$ and $PI_i$ to determine the values of $PO_i$ and $C_{i+1}$.
    3. Increment $i$.
    4. If $i < n$, go to step 2.

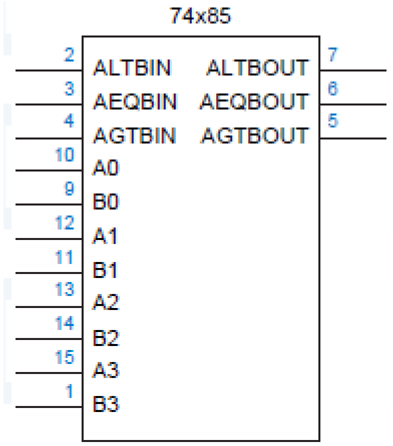
  - 迭代比较电路





# 3.7 比较器

- 标准MSI大小比较器
  - 74x85
    - 规格：
      - 4位比较器
      - 大于、小于、等于信号级联
    - 逻辑：



$$\text{AGTBOUT} = (A > B) + (A = B) \cdot \text{AGTBIN}$$

$$\text{AEQBOUT} = (A = B) \cdot \text{AEQBIN}$$

$$\text{ALTBOUT} = (A < B) + (A = B) \cdot \text{ALTBIN}$$

$$\begin{aligned}(A > B) = {} & A3 \cdot B3' + \\ & (A3 \oplus B3)' \cdot A2 \cdot B2' + \\ & (A3 \oplus B3)' \cdot (A2 \oplus B2)' \cdot A1 \cdot B1' + \\ & (A3 \oplus B3)' \cdot (A2 \oplus B2)' \cdot (A1 \oplus B1)' \cdot A0 \cdot B0'\end{aligned}$$

## 3.7 比较器

- 标准MSI大小比较器
  - 74x85
    - 应用：12位比较器

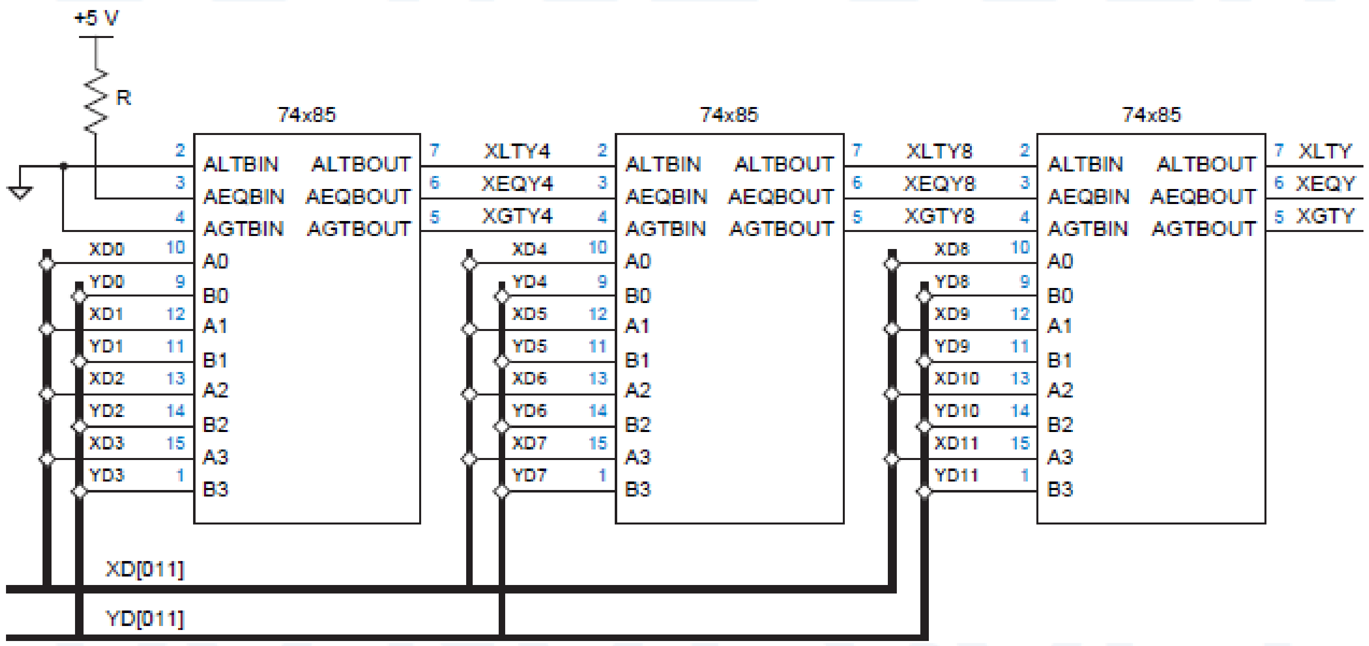
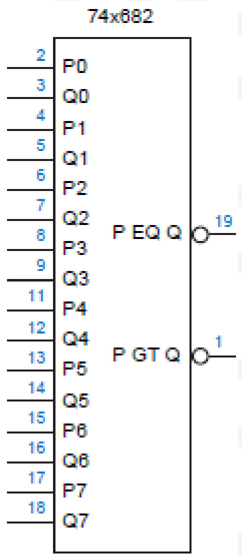


Figure 5-81 A 12-bit comparator using 74x85s.

## 3.7 比较器

- 标准MSI大小比较
  - 74x682
    - 规格：
      - 8位比较器
      - 大于、等于输出
      - 无级联



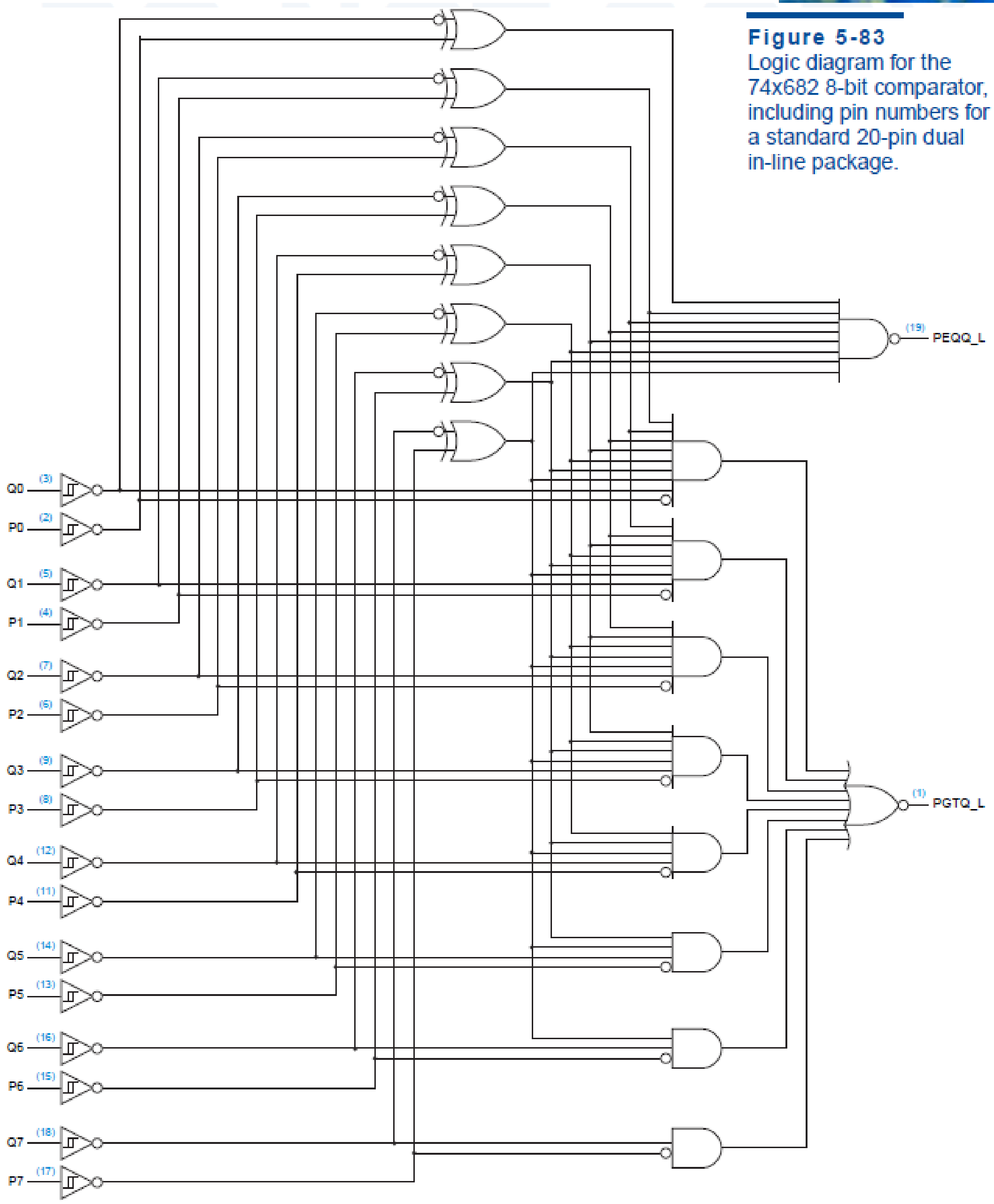
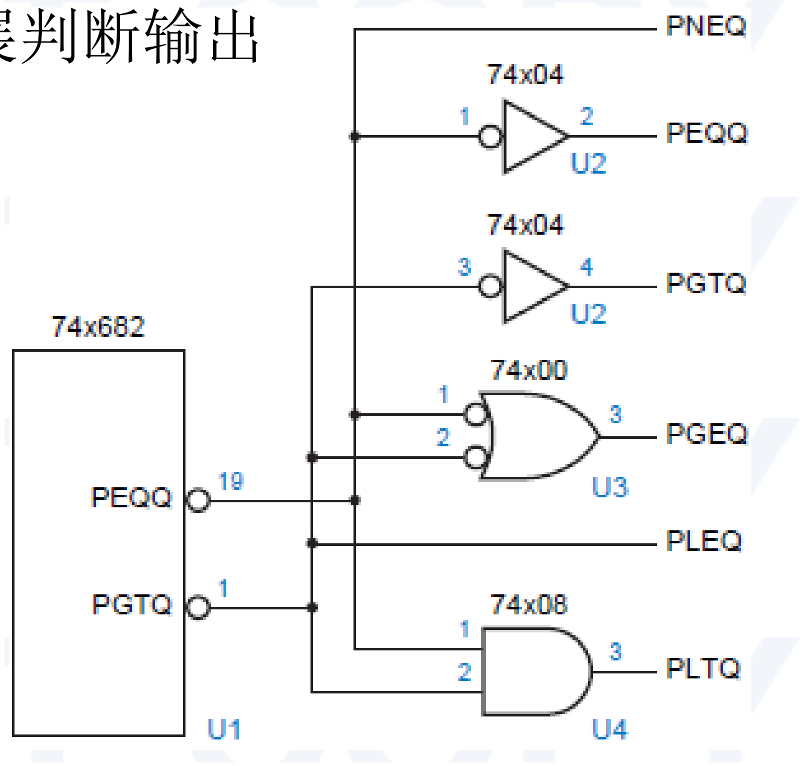


**Figure 5-83**
Logic diagram for the 74x682 8-bit comparator, including pin numbers for a standard 20-pin dual in-line package.

# 3.7 比较器

- 标准MSI大小比较器
  - 74x682
    - 应用：扩展判断输出



# 3.7 比较器

- Verilog实现比较器
  - 有相应操作符：

> >= < <= == !=

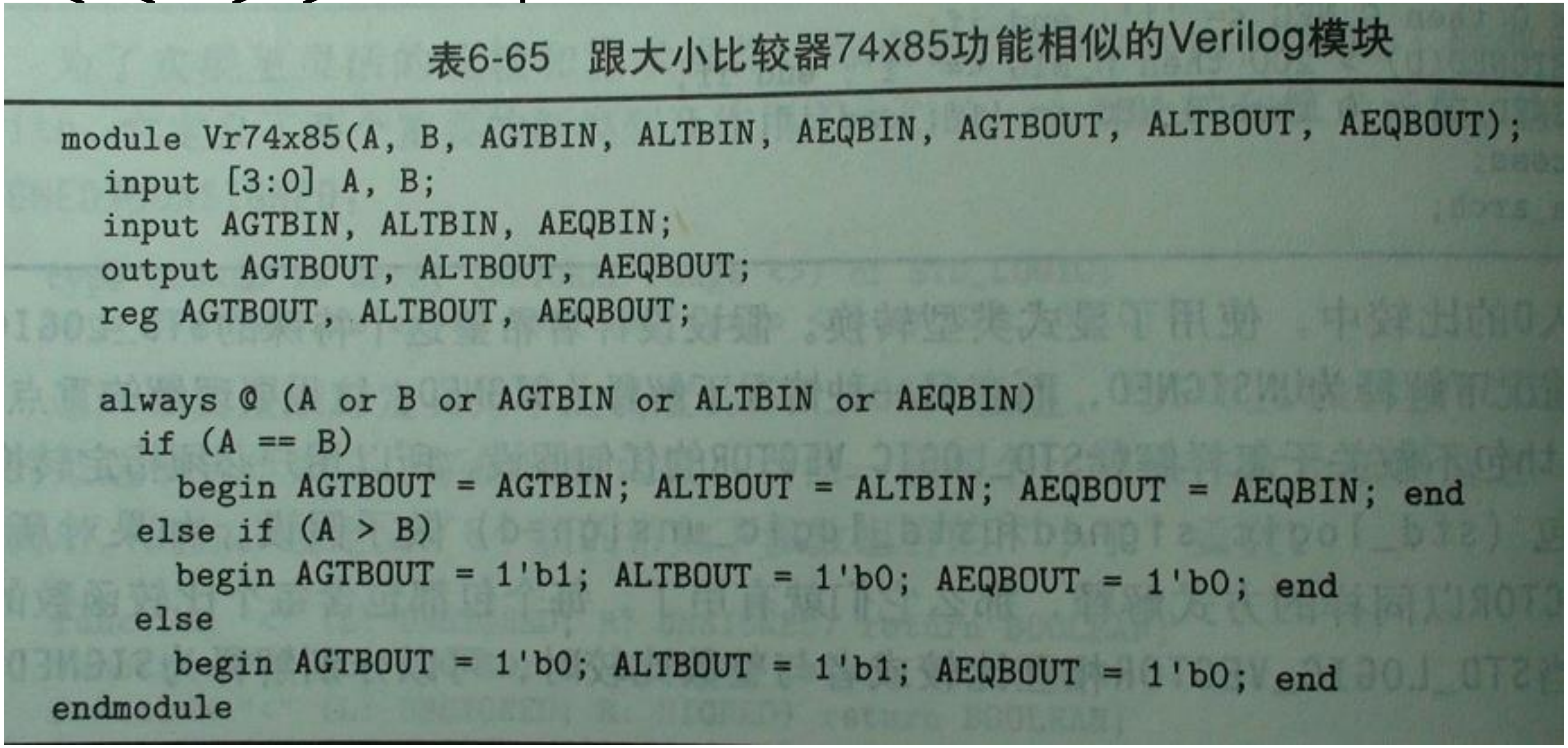


表6-65　跟大小比较器74x85功能相似的Verilog模块

```
module Vr74x85(A, B, AGTBIN, ALTBIN, AEQBIN, AGTBOUT, ALTBOUT, AEQBOUT);
  input [3:0] A, B;
  input AGTBIN, ALTBIN, AEQBIN;
  output AGTBOUT, ALTBOUT, AEQBOUT;
  reg AGTBOUT, ALTBOUT, AEQBOUT;

  always @ (A or B or AGTBIN or ALTBIN or AEQBIN)
    if (A == B)
      begin AGTBOUT = AGTBIN; ALTBOUT = ALTBIN; AEQBOUT = AEQBIN; end
    else if (A > B)
      begin AGTBOUT = 1'b1; ALTBOUT = 1'b0; AEQBOUT = 1'b0; end
    else
      begin AGTBOUT = 1'b0; ALTBOUT = 1'b1; AEQBOUT = 1'b0; end
endmodule
```

# 3.7 比较器

- Verilog实现比较器

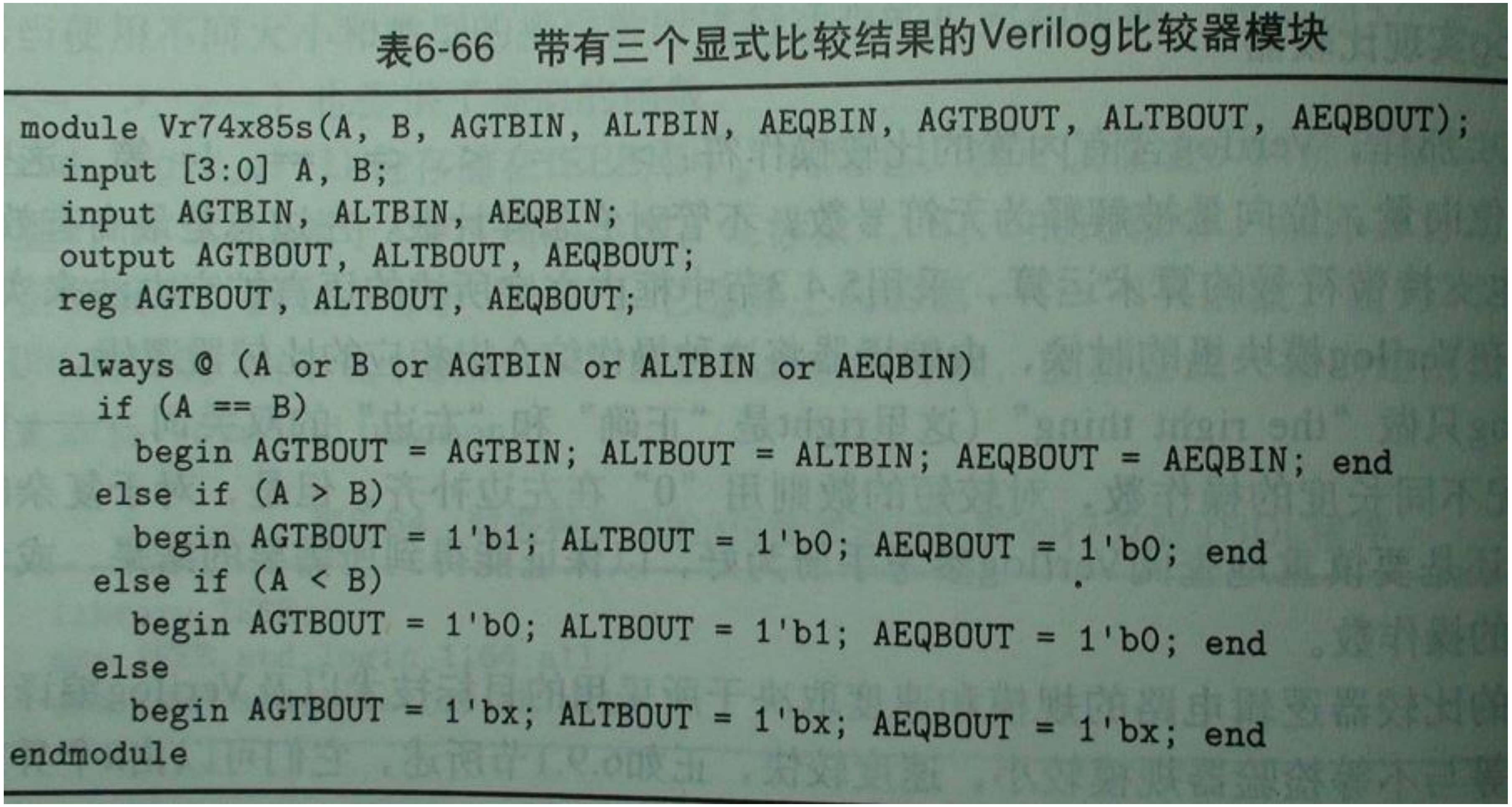


表6-66　带有三个显式比较结果的Verilog比较器模块

```
module Vr74x85s(A, B, AGTBIN, ALTBIN, AEQBIN, AGTBOUT, ALTBOUT, AEQBOUT);
  input [3:0] A, B;
  input AGTBIN, ALTBIN, AEQBIN;
  output AGTBOUT, ALTBOUT, AEQBOUT;
  reg AGTBOUT, ALTBOUT, AEQBOUT;

  always @ (A or B or AGTBIN or ALTBIN or AEQBIN)
    if (A == B)
      begin AGTBOUT = AGTBIN; ALTBOUT = ALTBIN; AEQBOUT = AEQBIN; end
    else if (A > B)
      begin AGTBOUT = 1'b1; ALTBOUT = 1'b0; AEQBOUT = 1'b0; end
    else if (A < B)
      begin AGTBOUT = 1'b0; ALTBOUT = 1'b1; AEQBOUT = 1'b0; end
    else
      begin AGTBOUT = 1'bx; ALTBOUT = 1'bx; AEQBOUT = 1'bx; end
endmodule
```

# 3.7 比较器

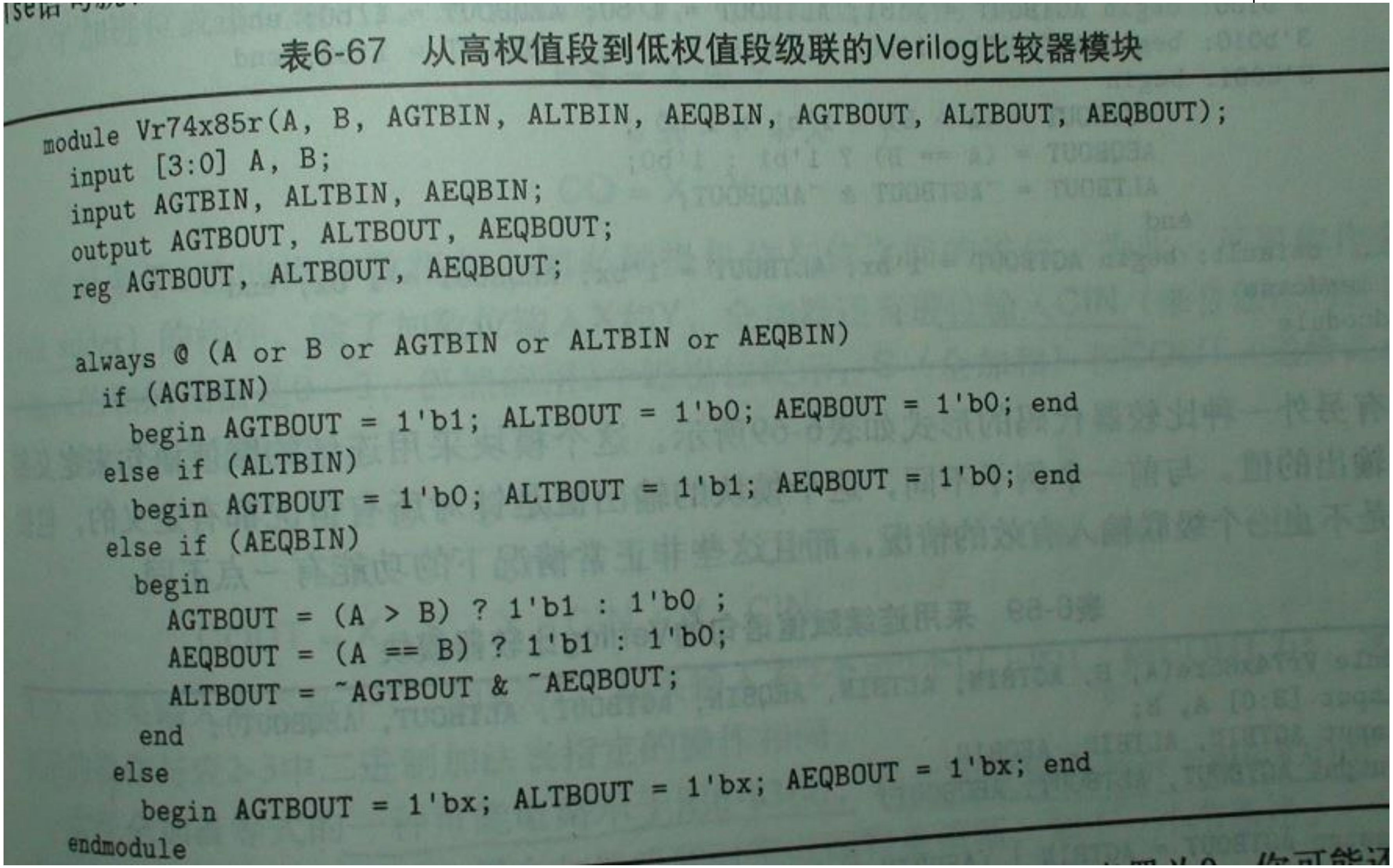


表6-67 从高权值段到低权值段级联的Verilog比较器模块

```
module Vr74x85r(A, B, AGTBIN, ALTBIN, AEQBIN, AGTBOUT, ALTBOUT, AEQBOUT);
  input [3:0] A, B;
  input AGTBIN, ALTBIN, AEQBIN;
  output AGTBOUT, ALTBOUT, AEQBOUT;
  reg AGTBOUT, ALTBOUT, AEQBOUT;

  always @ (A or B or AGTBIN or ALTBIN or AEQBIN)
    if (AGTBIN)
      begin AGTBOUT = 1'b1; ALTBOUT = 1'b0; AEQBOUT = 1'b0; end
    else if (ALTBIN)
      begin AGTBOUT = 1'b0; ALTBOUT = 1'b1; AEQBOUT = 1'b0; end
    else if (AEQBIN)
      begin
        AGTBOUT = (A > B) ? 1'b1 : 1'b0 ;
        AEQBOUT = (A == B) ? 1'b1 : 1'b0;
        ALTBOUT = ~AGTBOUT & ~AEQBOUT;
      end
    else
      begin AGTBOUT = 1'bx; ALTBOUT = 1'bx; AEQBOUT = 1'bx; end
endmodule
```

# 3.7 比较器

- Verilog实现比较器

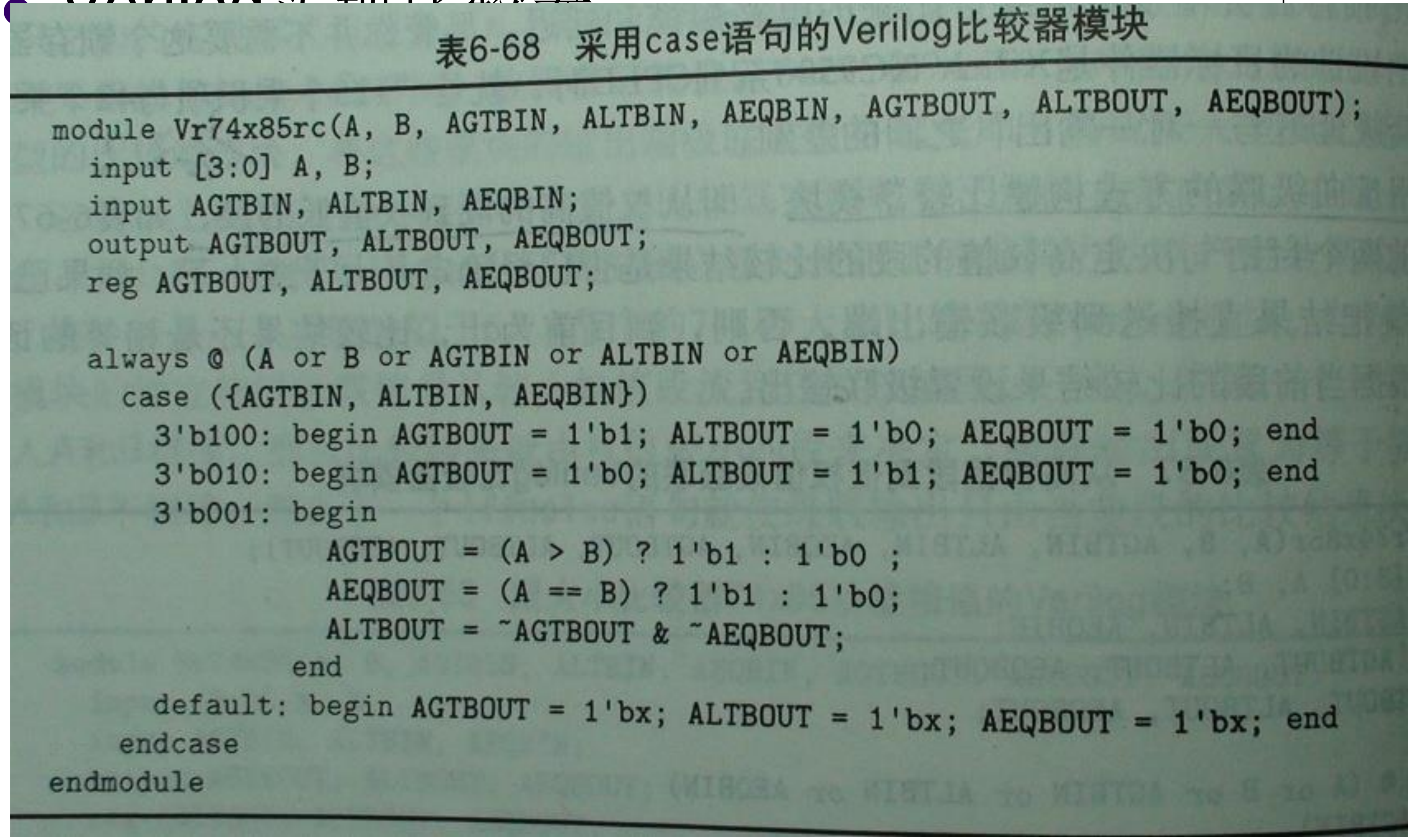


表6-68　采用case语句的Verilog比较器模块

```
module Vr74x85rc(A, B, AGTBIN, ALTBIN, AEQBIN, AGTBOUT, ALTBOUT, AEQBOUT);
  input [3:0] A, B;
  input AGTBIN, ALTBIN, AEQBIN;
  output AGTBOUT, ALTBOUT, AEQBOUT;
  reg AGTBOUT, ALTBOUT, AEQBOUT;

  always @ (A or B or AGTBIN or ALTBIN or AEQBIN)
    case ({AGTBIN, ALTBIN, AEQBIN})
      3'b100: begin AGTBOUT = 1'b1; ALTBOUT = 1'b0; AEQBOUT = 1'b0; end
      3'b010: begin AGTBOUT = 1'b0; ALTBOUT = 1'b1; AEQBOUT = 1'b0; end
      3'b001: begin
                AGTBOUT = (A > B) ? 1'b1 : 1'b0 ;
                AEQBOUT = (A == B) ? 1'b1 : 1'b0;
                ALTBOUT = ~AGTBOUT & ~AEQBOUT;
              end
      default: begin AGTBOUT = 1'bx; ALTBOUT = 1'bx; AEQBOUT = 1'bx; end
    endcase
endmodule
```

## 3.7 比较器

- Verilog实现比较器
  - 有相应操作符：

>, >=, <, <=, ==, !=

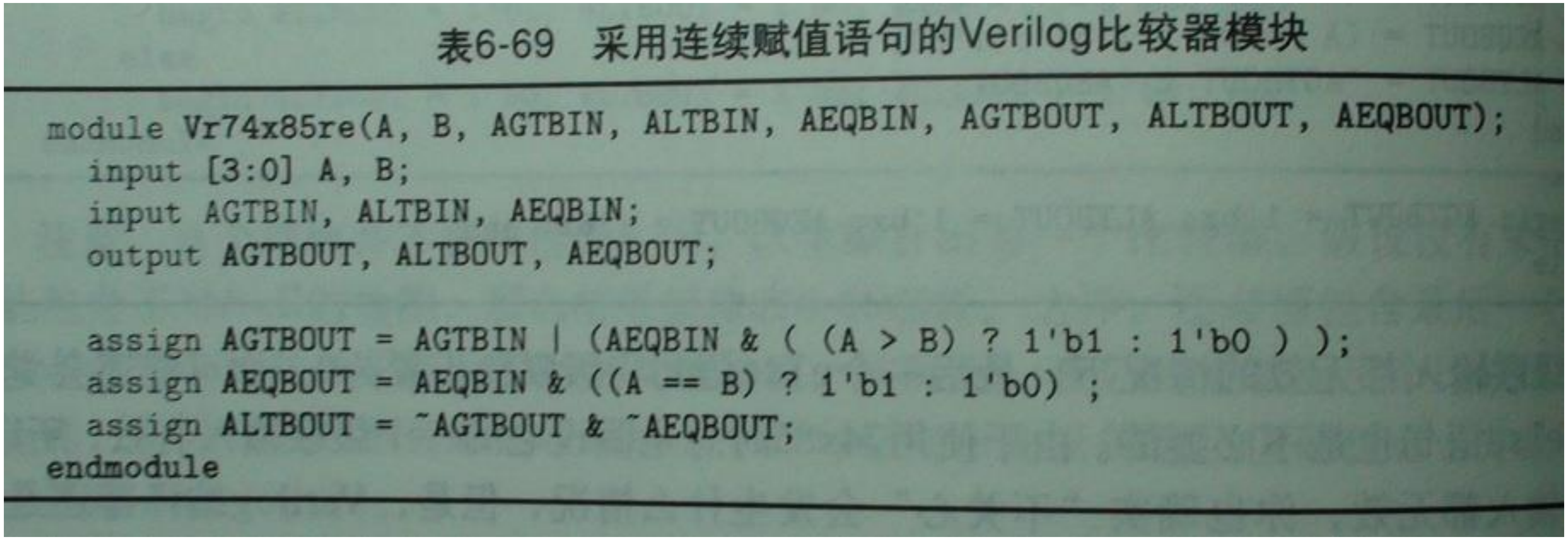


表6-69　采用连续赋值语句的Verilog比较器模块

```
module Vr74x85re(A, B, AGTBIN, ALTBIN, AEQBIN, AGTBOUT, ALTBOUT, AEQBOUT);
  input [3:0] A, B;
  input AGTBIN, ALTBIN, AEQBIN;
  output AGTBOUT, ALTBOUT, AEQBOUT;

  assign AGTBOUT = AGTBIN | (AEQBIN & ( (A > B) ? 1'b1 : 1'b0 ) );
  assign AEQBOUT = AEQBIN & ((A == B) ? 1'b1 : 1'b0) ;
  assign ALTBOUT = ~AGTBOUT & ~AEQBOUT;
endmodule
```

## 3.8 加法器、减法器和ALU

- 加法器：
  - 根据相应的数制规定，完成两个数加法运算的电路。
- 减法器：
  - 根据相应的数制规定，完成两个数减法运算的电路。
  - 可以直接设计。
  - 可以通过减数变补，进行加法运算。
- ALU
  - 可以根据操作码完成加法、减法等运算功能的电路。

## 3.8 加法器、减法器和ALU

- 半加器：执行两个1位二进制数加法运算的电路
  - 函数：

$$\begin{aligned} HS &= X \oplus Y \\ &= X \cdot Y' + X' \cdot Y \\ CO &= X \cdot Y \end{aligned}$$

- 全加器：考虑低位进位的1位二进制数加法电路
  - 函数：

$$\begin{aligned} S &= X \oplus Y \oplus CIN \\ &= X \cdot Y' \cdot CIN' + X' \cdot Y \cdot CIN' + X' \cdot Y' \cdot CIN + X \cdot Y \cdot CIN \\ COUT &= X \cdot Y + X \cdot CIN + Y \cdot CIN \end{aligned}$$

## 3.8 加法器、减法器和ALU

- 半加器：执行两个1位二进制数加法运算的电路
  - 函数：

$$\begin{aligned} HS &= X \oplus Y \\ &= X \cdot Y' + X' \cdot Y \\ CO &= X \cdot Y \end{aligned}$$

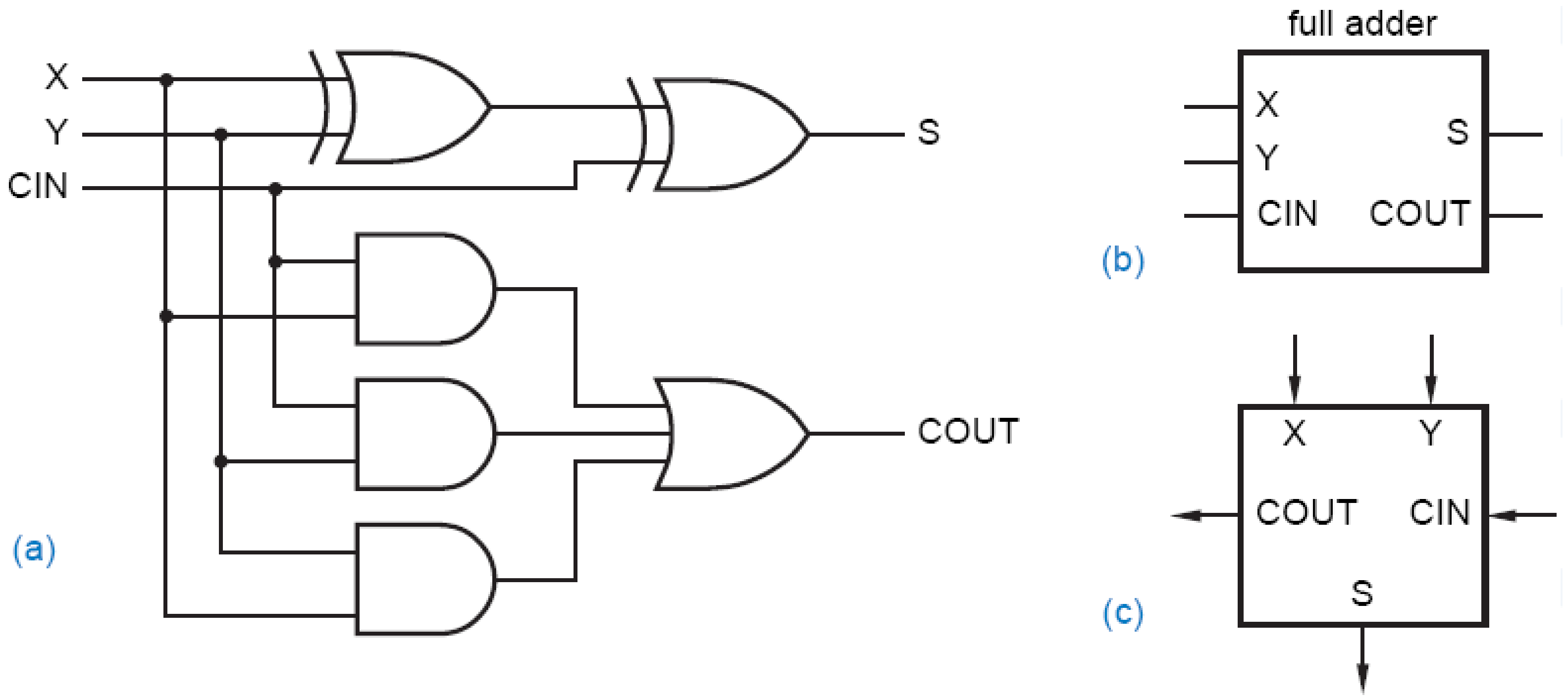
- 全加器：考虑低位进位的1位二进制数加法电路
  - 函数：

$$\begin{aligned} S &= X \oplus Y \oplus CIN \\ &= X \cdot Y' \cdot CIN' + X' \cdot Y \cdot CIN' + X' \cdot Y' \cdot CIN + X \cdot Y \cdot CIN \\ COUT &= X \cdot Y + X \cdot CIN + Y \cdot CIN \end{aligned}$$

## 3.8 加法器、减法器和ALU

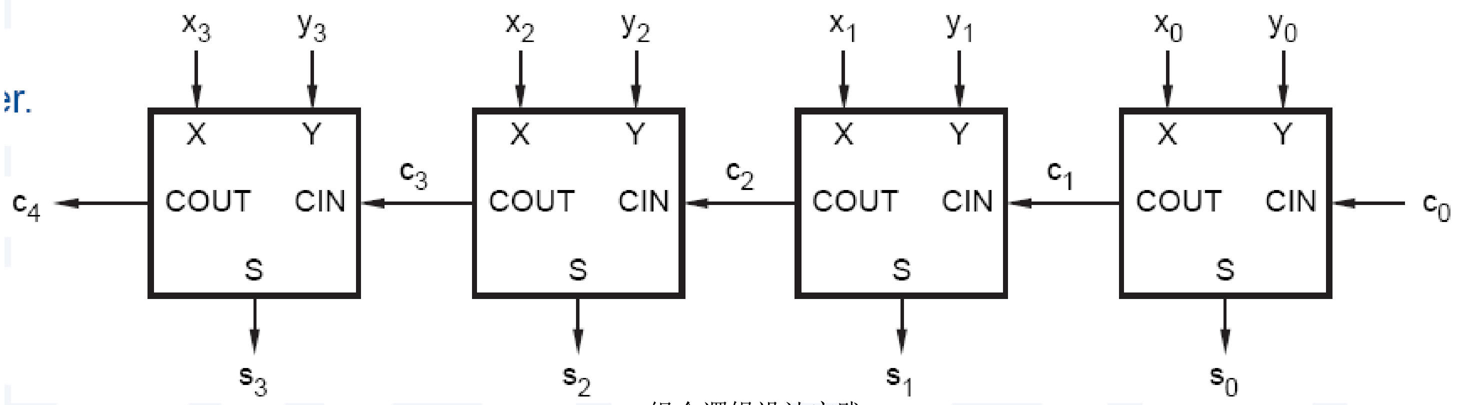
- 全加器：
  - 电路和符号

Full adder: (a) gate-level circuit diagram; (b) logic symbol; (c) alternate logic symbol suitable for cascading.



# 3.8 加法器、减法器和ALU

- 串行进位加器：行波进位加法器 ripper adder
  - 规格：
    - 两个二进制字，每个n位，相加。
  - 方法：
    - n个全加器的级联，属于迭代电路。
  - 延迟：　$t_{\mathrm{ADD}} = t_{\mathrm{XYCout}} + (n-2)\cdot t_{\mathrm{CinCout}} + t_{\mathrm{CinS}}$
  - 特点：简单、速度慢



## 3.8 加法器、减法器和ALU

- 减法器
  - 1位全减器：
    - X被减数；Y减数；BIN借位输入。
  - 函数：

$$D = X \oplus Y \oplus BIN$$
$$BOUT = X' \cdot Y + X' \cdot BIN + Y \cdot BIN$$

  - 减法转换成加法：

$$BOUT = X' \cdot Y + X' \cdot BIN + Y \cdot BIN$$
$$\begin{aligned} BOUT' &= (X + Y') \cdot (X + BIN') \cdot (Y' + BIN') \\ &= X \cdot Y' + X \cdot BIN' + Y' \cdot BIN' \end{aligned}$$
$$\begin{aligned} D &= X \oplus Y \oplus BIN \\ &= X \oplus Y' \oplus BIN' \end{aligned}$$

## 3.8 加法器、减法器和ALU

- 减法器
  - 减法转换成加法：

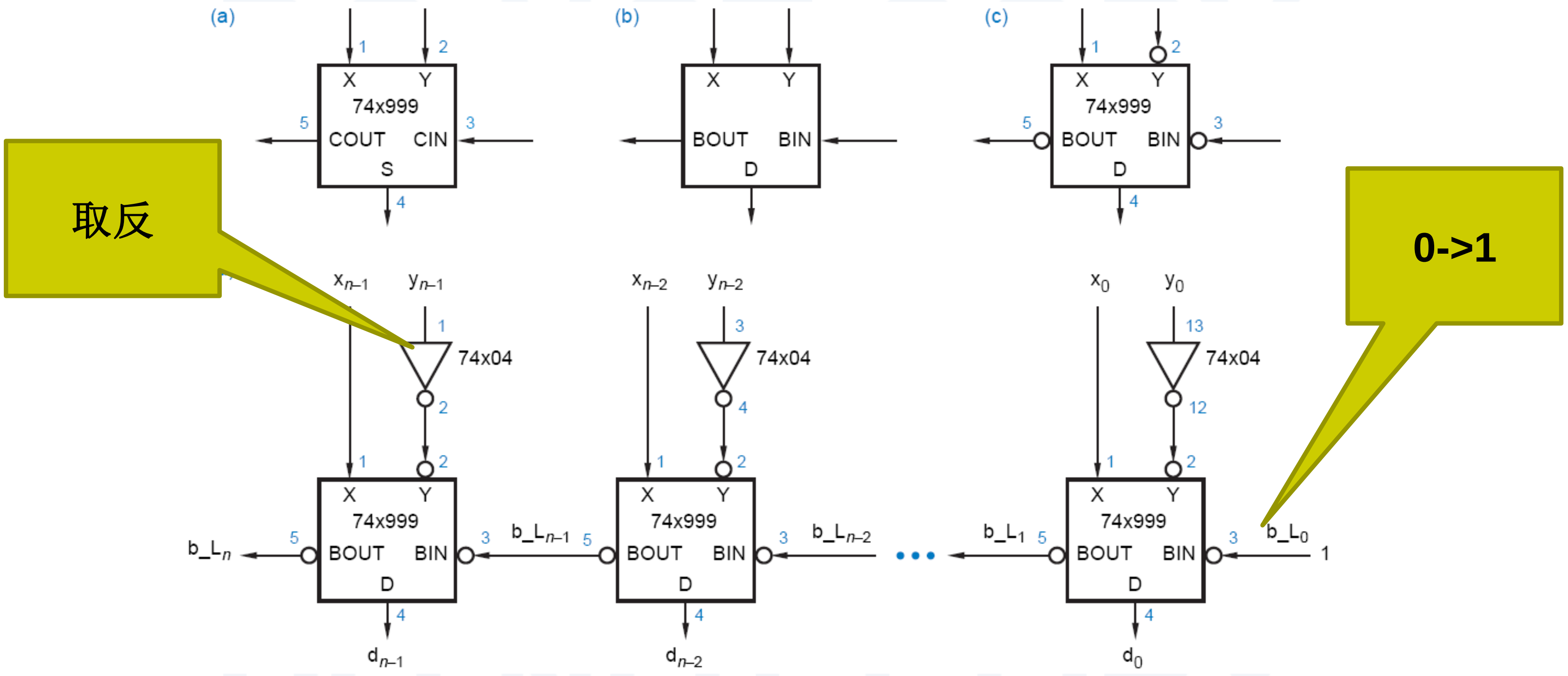


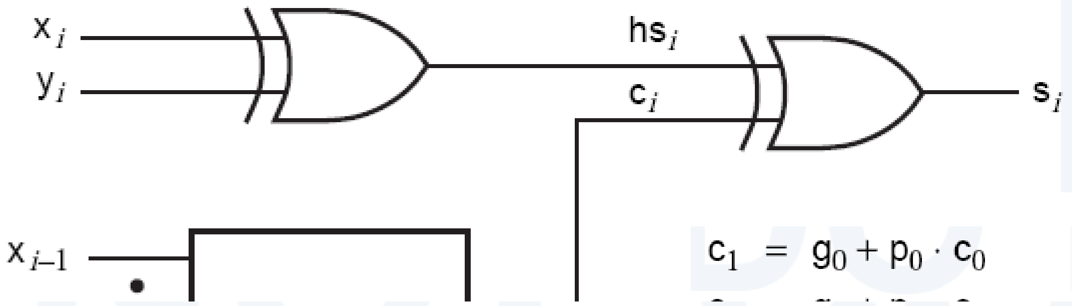
**Figure 5-87** Designing subtractors using adders: (a) full adder; (b) full subtractor; (c) interpreting the device in (a) as a full subtractor; (d) ripple subtractor.

## 3.8 加法器、减法器和ALU

- 先行进位加法器（超前进位加法器）
  - 想法： $s_i = x_i \oplus y_i \oplus c_i$

$$g_i = x_i \cdot y_i$$
$$p_i = x_i + y_i$$

$$c_{i+1} = g_i + p_i \cdot c_i$$



$$c_1 = g_0 + p_0 \cdot c_0$$

- For a particular combination of inputs $x_i$ and $y_i$, adder stage $i$ is said to *generate* a carry if it produces a carry-out of 1 ($c_{i+1} = 1$) independent of the inputs on $x_0 - x_{i-1}$, $y_0 - y_{i-1}$, and $c_0$.
- For a particular combination of inputs $x_i$ and $y_i$, adder stage $i$ is said to *propagate* carries if it produces a carry-out of 1 ($c_{i+1} = 1$) in the presence of an input combination of $x_0 - x_{i-1}$, $y_0 - y_{i-1}$, and $c_0$ that causes a carry-in of 1 ($c_i = 1$).

$\cdot c_0)$

$$= g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_0$$

# 3.8 加法器、减法器和ALU

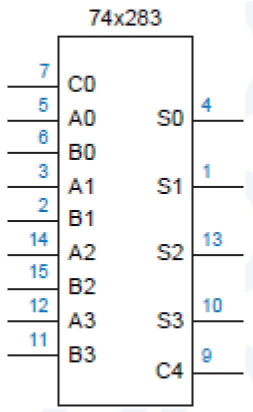
- MSI加法器
  - 74x283：
    - 4位先行进位加法器
    - 分析:
    - 符号

$$
\begin{aligned}
hs_i &= x_i \oplus y_i \\
&= x_i \cdot y_i' + x_i' \cdot y_i \\
&= x_i \cdot y_i' + x_i \cdot x_i' + x_i' \cdot y_i + y_i \cdot \\
&= (x_i + y_i) \cdot (x_i' + y_i') \\
&= (x_i + y_i) \cdot (x_i \cdot y_i)' \\
&= p_i \cdot g_i'
\end{aligned}
$$

$$
\begin{aligned}
c_{i+1} &= p_i \cdot g_i + p_i \cdot c_i \\
&= p_i \cdot (g_i + c_i)
\end{aligned}
$$

g为1时，p肯定为1，所以，g和pg等效。方便分解。

$$
\begin{aligned}
c_1 &= p_0 \cdot (g_0 + c_0) \\
c_2 &= p_1 \cdot (g_1 + c_1) \\
&= p_1 \cdot (g_1 + p_0 \cdot (g_0 + c_0)) \\
&= p_1 \cdot (g_1 + p_0) \cdot (g_1 + g_0 + c_0) \\
c_3 &= p_2 \cdot (g_2 + c_2) \\
&= p_2 \cdot (g_2 + p_1 \cdot (g_1 + p_0) \cdot (g_1 + g_0 + c_0)) \\
&= p_2 \cdot (g_2 + p_1) \cdot (g_2 + g_1 + p_0) \cdot (g_2 + g_1 + g_0 + c_0) \\
c_4 &= p_3 \cdot (g_3 + c_3) \\
&= p_3 \cdot (g_3 + p_2 \cdot (g_2 + p_1) \cdot (g_2 + g_1 + p_0) \cdot (g_2 + g_1 + g_0 + c_0)) \\
&= p_3 \cdot (g_3 + p_2) \cdot (g_3 + g_2 + p_1) \cdot (g_3 + g_2 + g_1 + p_0) \cdot (g_3 + g_2 + g_1 + g_0 + c_0)
\end{aligned}
$$



# 3.8 加法器、减法器和ALU

- MSI加法器
  - 74x283：
    - 电路设计

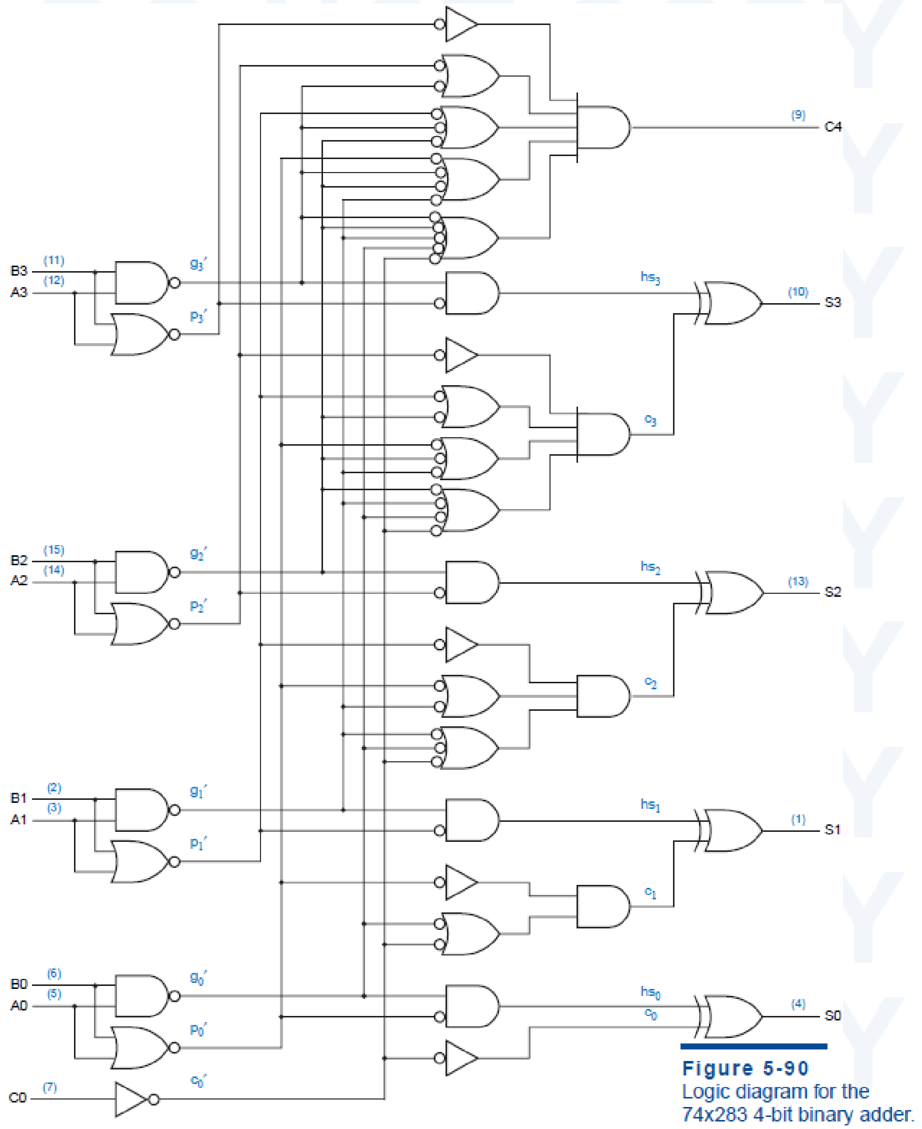
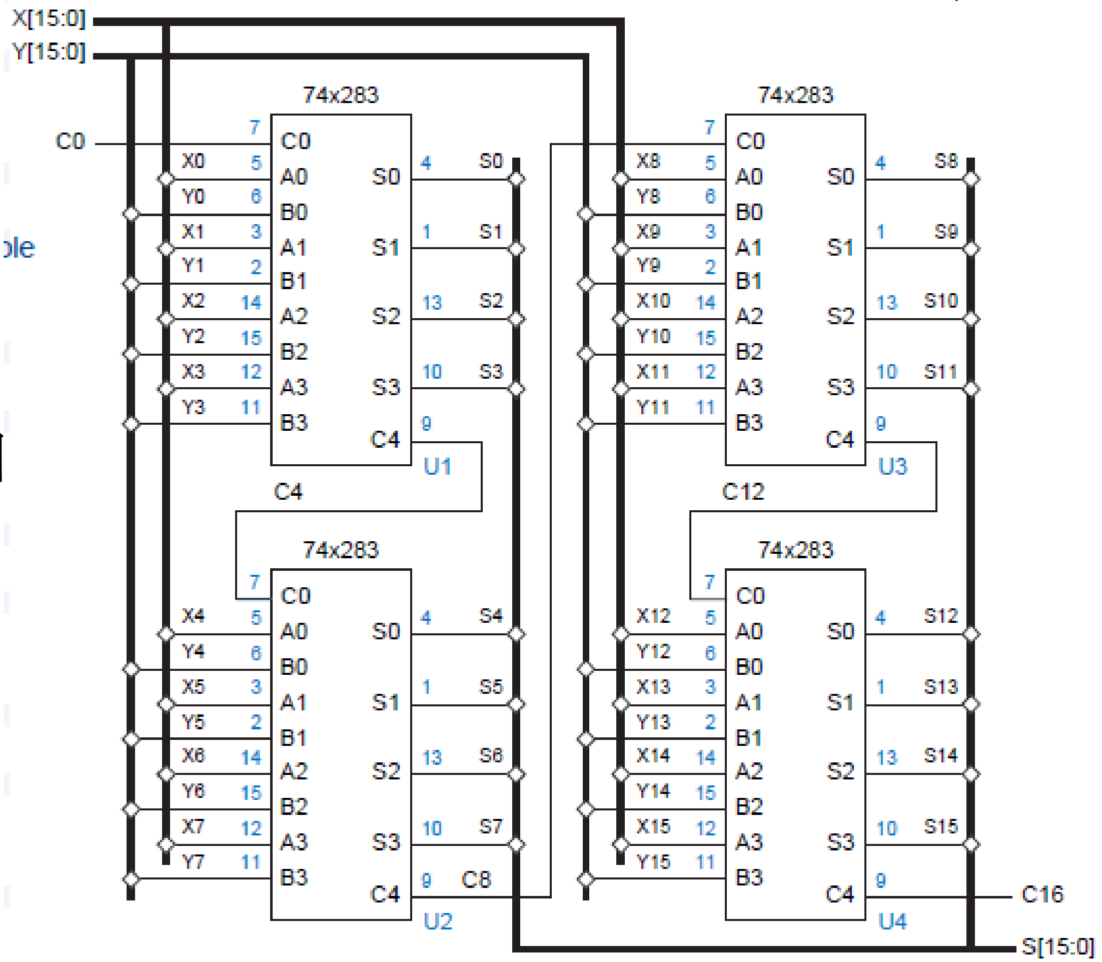


**Figure 5-90**
Logic diagram for the 74x283 4-bit binary adder.

# 3.8 加法器、减法器和ALU

- MSI加法器
  - 74x283：
    - 应用：’283的进位信号c0到c4的延迟很短相当于2个反相门，可以级联扩展计算位数，**组间串行进位加法器**（group-ripple adder）。

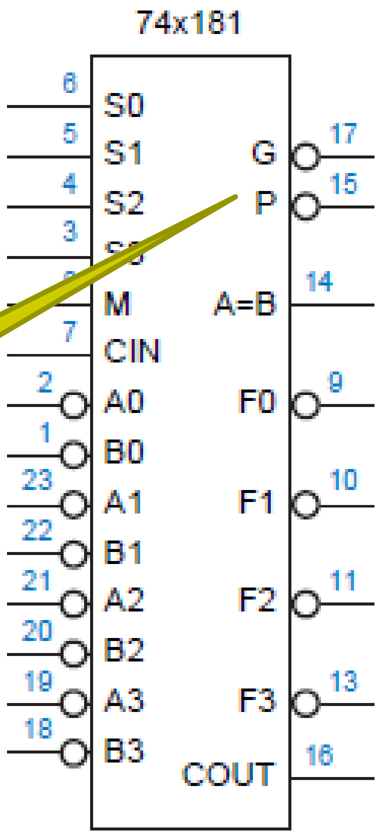


# 3.8 ALU

- MSI ALU
  - 功能：能够对2个b位的操作数进行若干不同的算术和逻辑操作，要执行的操作由一组功能选择输入来指定。
    - 典型：4位，3-5功能输入，32种操作
  - 74x181 4位ALU
    - 逻辑符号：

组间先行进位，后面介绍。

Logic symbol for the 74x181 4-bit ALU.



# 3.8 ALU

- MSI ALU
  - 74x181 4位ALU
    - 功能列表
    - 算术运算时Cin和Cout可以进行级联。

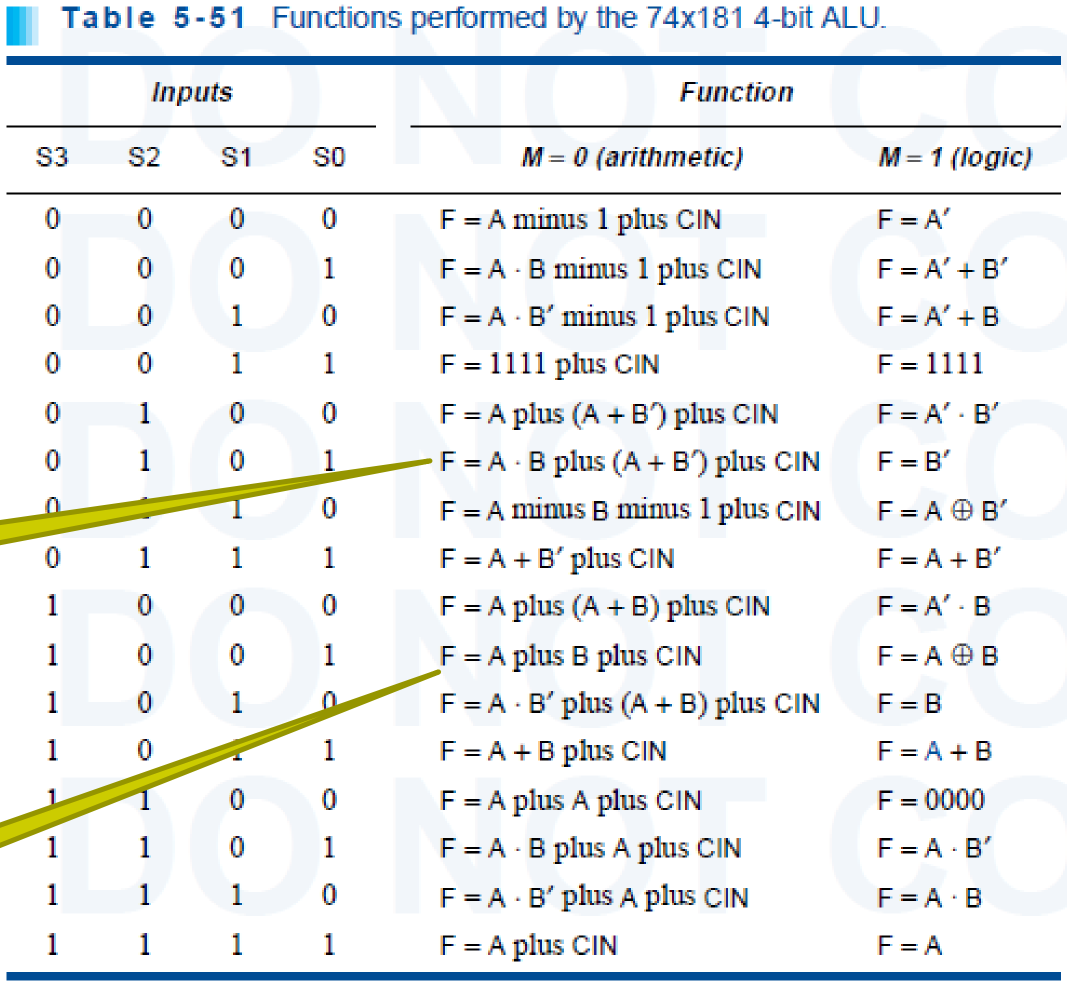


Table 5-51 Functions performed by the 74x181 4-bit ALU.

| Inputs | | | | Function | |
|---|---|---|---|---|---|
| S3 | S2 | S1 | S0 | M = 0 (arithmetic) | M = 1 (logic) |
| 0 | 0 | 0 | 0 | F = A minus 1 plus CIN | F = A′ |
| 0 | 0 | 0 | 1 | F = A · B minus 1 plus CIN | F = A′ + B′ |
| 0 | 0 | 1 | 0 | F = A · B′ minus 1 plus CIN | F = A′ + B |
| 0 | 0 | 1 | 1 | F = 1111 plus CIN | F = 1111 |
| 0 | 1 | 0 | 0 | F = A plus (A + B′) plus CIN | F = A′ · B′ |
| 0 | 1 | 0 | 1 | F = A · B plus (A + B′) plus CIN | F = B′ |
| 0 | 1 | 1 | 0 | F = A minus B minus 1 plus CIN | F = A ⊕ B′ |
| 0 | 1 | 1 | 1 | F = A + B′ plus CIN | F = A + B′ |
| 1 | 0 | 0 | 0 | F = A plus (A + B) plus CIN | F = A′ · B |
| 1 | 0 | 0 | 1 | F = A plus B plus CIN | F = A ⊕ B |
| 1 | 0 | 1 | 0 | F = A · B′ plus (A + B) plus CIN | F = B |
| 1 | 0 | 1 | 1 | F = A + B plus CIN | F = A + B |
| 1 | 1 | 0 | 0 | F = A plus A plus CIN | F = 0000 |
| 1 | 1 | 0 | 1 | F = A · B plus A plus CIN | F = A · B′ |
| 1 | 1 | 1 | 0 | F = A · B′ plus A plus CIN | F = A · B |
| 1 | 1 | 1 | 1 | F = A plus CIN | F = A |

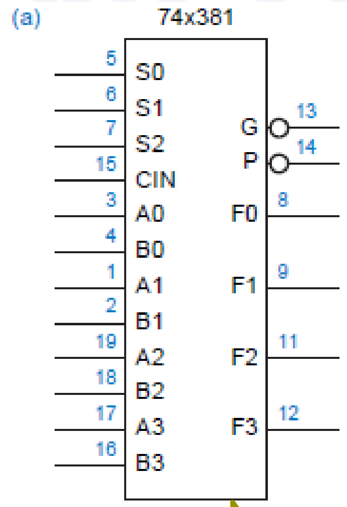
从来没用

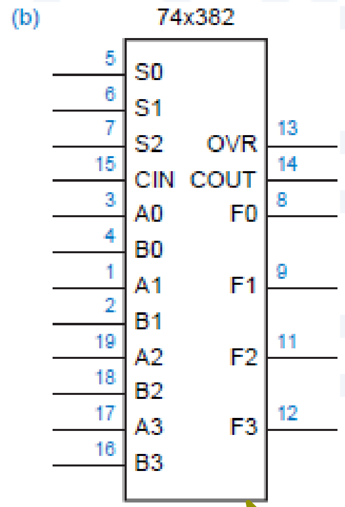
补码加法

低电平有效，高电平有效时，功能表要重定制

# 3.8 ALU

- MSI ALU  另外两个器件
  - 74x381和74x382





| *Inputs* | | | |
|---|---|---|---|
| S2 | S1 | S0 | *Function* |
| 0 | 0 | 0 | F = 0000 |
| 0 | 0 | 1 | F = B minus A minus 1 plus CIN |
| 0 | 1 | 0 | F = A minus B minus 1 plus CIN |
| 0 | 1 | 1 | F = A plus B plus CIN |
| 1 | 0 | 0 | F = A ⊕ B |
| 1 | 0 | 1 | F = A + B |
| 1 | 1 | 0 | F = A · B |
| 1 | 1 | 1 | F = 1111 |

组间先行进位

串行进位，溢出

# 3.8 ALU

- MSI ALU
  - 组间先行进位：group-carry lookahead
    - 不同于组间串行进位，先行进位信号直接由输入的操作数决定，没有组间串行的进位信号。
    - 函数：

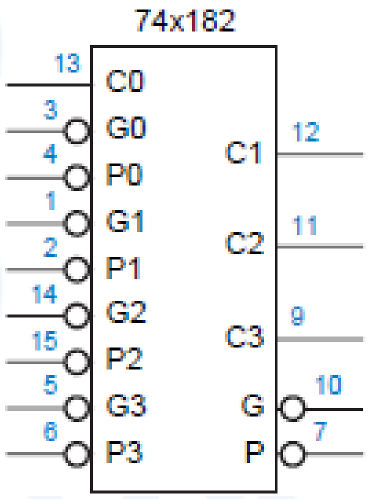
$$G\_L = (g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0)'$$

$$P\_L = (p_3 \cdot p_2 \cdot p_1 \cdot p_0)'$$

  - 74x182：4组先行进位计算器件

$$c_{i+1} = g_i + p_i \cdot c_i$$
$$= (g_i + p_i) \cdot (g_i + c_i)$$

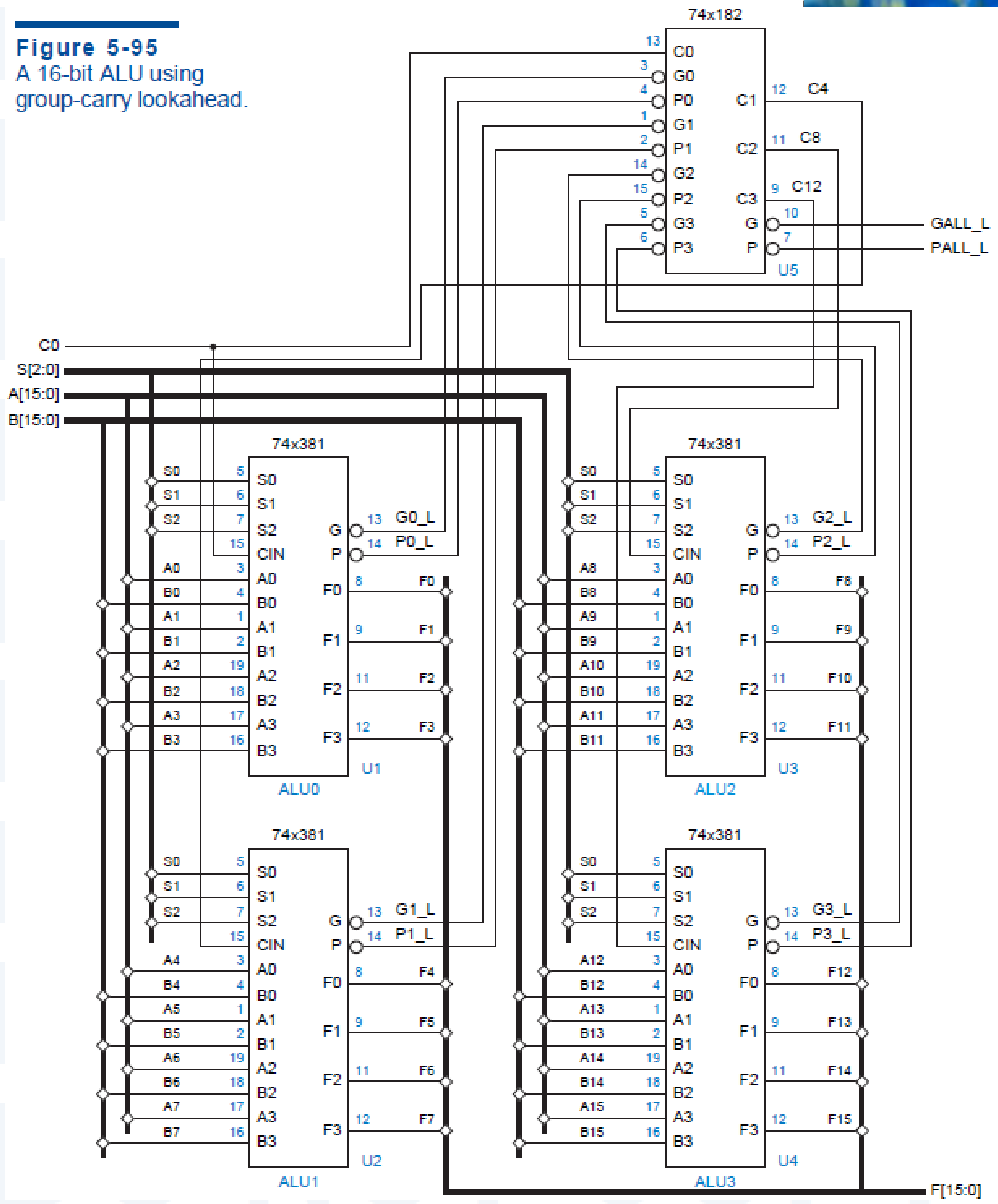
$$C1 = (G0+P0) \cdot (G0+C0)$$
$$C2 = (G1+P1) \cdot (G1+G0+P0) \cdot (G1+G0+C0)$$
$$C3 = (G2+P2) \cdot (G2+G1+P1) \cdot (G2+G1+G0+P0) \cdot (G2+G1+G0+C0)$$



# 3.8 ALU

- MSI ALU
  - 74x182：4组先行进位计算器件应用：
    - 16位组间进位ALU
    - 182增加了一级延迟。
    - 更多位可以扩展"超级组"，每个"超级组"有自己的'182

**Figure 5-95**
A 16-bit ALU using group-carry lookahead.



## 3.8 加法器、减法器和ALU

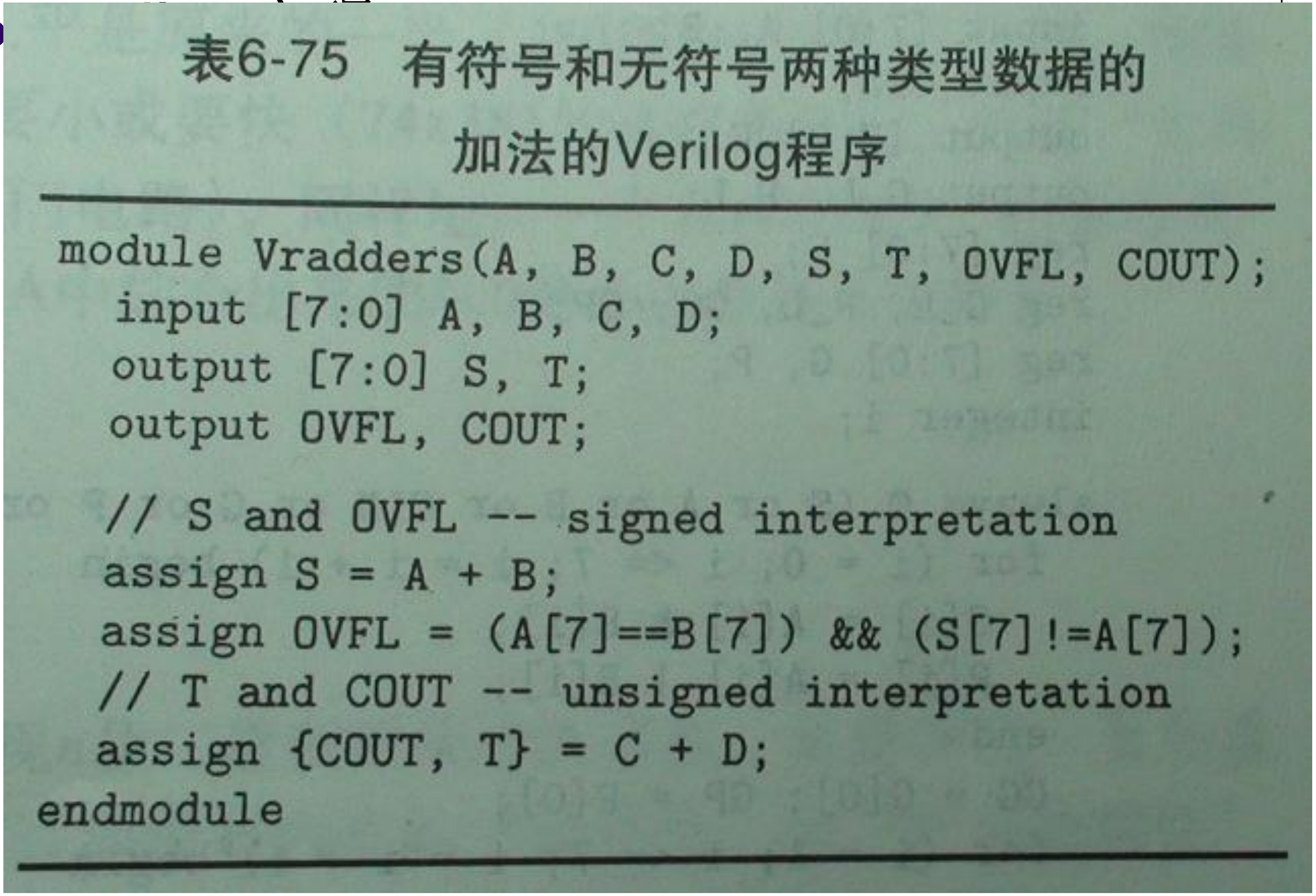


表6-75 有符号和无符号两种类型数据的加法的Verilog程序

```
module Vradders(A, B, C, D, S, T, OVFL, COUT);
  input [7:0] A, B, C, D;
  output [7:0] S, T;
  output OVFL, COUT;

  // S and OVFL -- signed interpretation
  assign S = A + B;
  assign OVFL = (A[7]==B[7]) && (S[7]!=A[7]);
  // T and COUT -- unsigned interpretation
  assign {COUT, T} = C + D;
endmodule
```

# 3.8 加法器、减法器和ALU

- Verilog实现
  - 有用于位向量的内部加法和减法操作符。

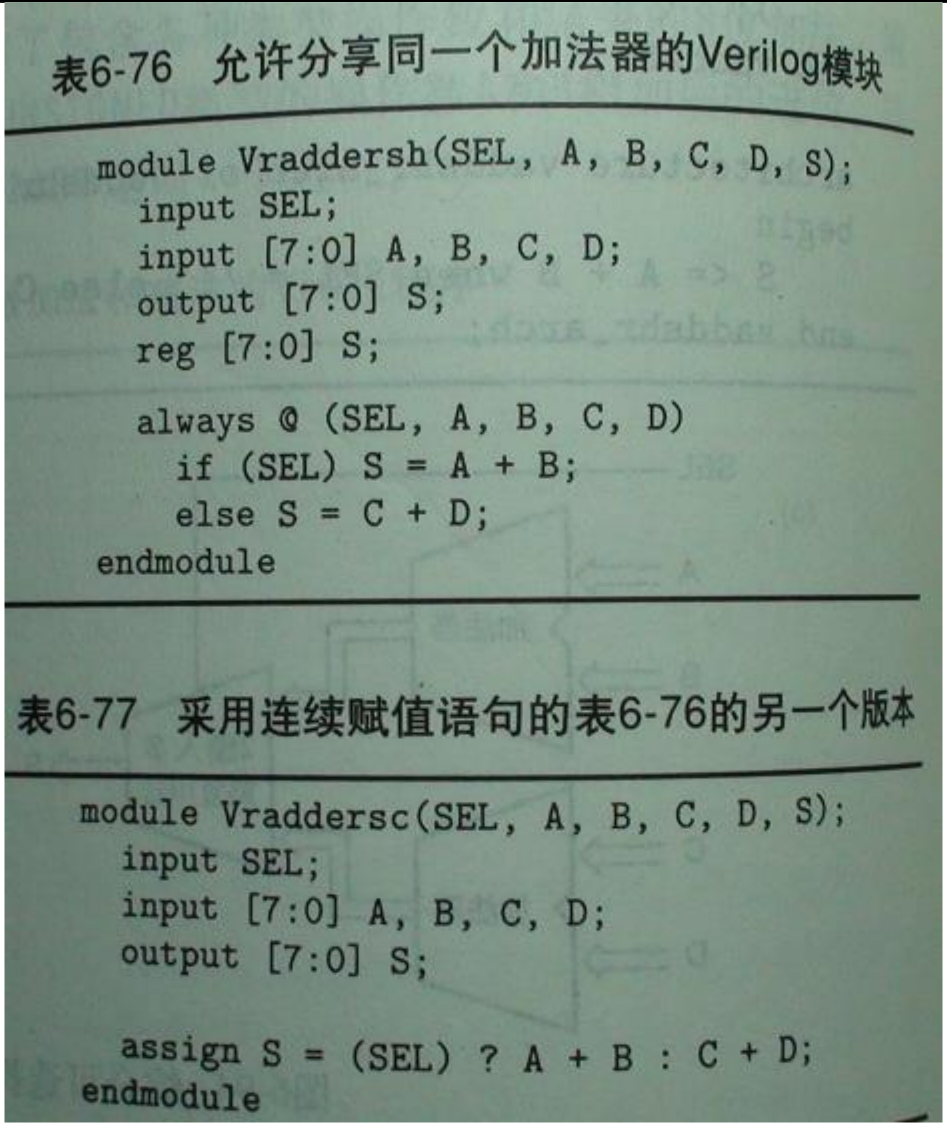


表6-76 允许分享同一个加法器的Verilog模块

```
module Vraddersh(SEL, A, B, C, D, S);
  input SEL;
  input [7:0] A, B, C, D;
  output [7:0] S;
  reg [7:0] S;

  always @ (SEL, A, B, C, D)
    if (SEL) S = A + B;
    else S = C + D;
endmodule
```

表6-77 采用连续赋值语句的表6-76的另一个版本

```
module Vraddersc(SEL, A, B, C, D, S);
  input SEL;
  input [7:0] A, B, C, D;
  output [7:0] S;

  assign S = (SEL) ? A + B : C + D;
endmodule
```

# 3.8 加法器、

- Verilog实现
  - 74x381的实现。

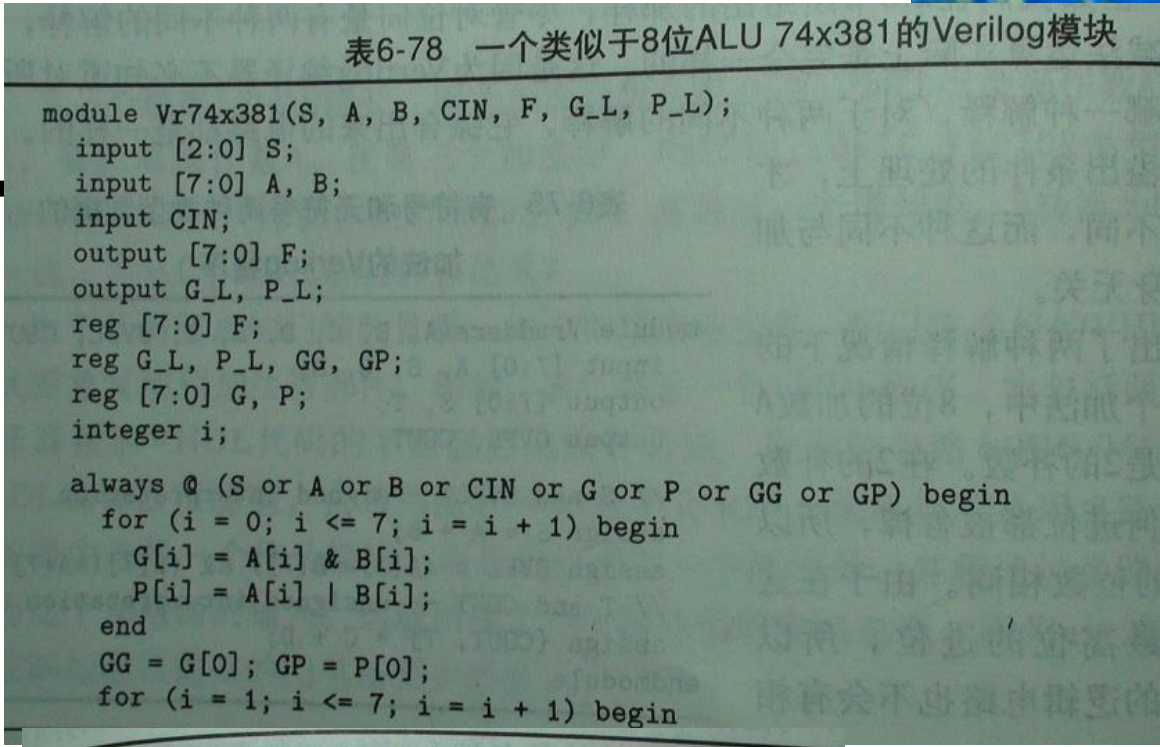
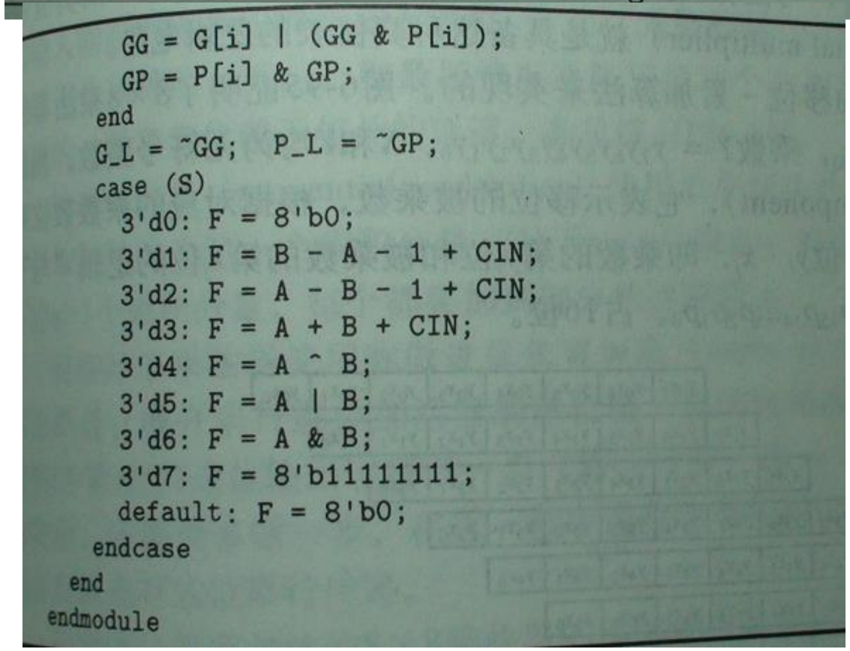


表6-78　一个类似于8位ALU 74x381的Verilog模块

```
module Vr74x381(S, A, B, CIN, F, G_L, P_L);
  input [2:0] S;
  input [7:0] A, B;
  input CIN;
  output [7:0] F;
  output G_L, P_L;
  reg [7:0] F;
  reg G_L, P_L, GG, GP;
  reg [7:0] G, P;
  integer i;

  always @ (S or A or B or CIN or G or P or GG or GP) begin
    for (i = 0; i <= 7; i = i + 1) begin
      G[i] = A[i] & B[i];
      P[i] = A[i] | B[i];
    end
    GG = G[0]; GP = P[0];
    for (i = 1; i <= 7; i = i + 1) begin
```



```
      GG = G[i] | (GG & P[i]);
      GP = P[i] & GP;
    end
    G_L = ~GG;  P_L = ~GP;
    case (S)
      3'd0: F = 8'b0;
      3'd1: F = B - A - 1 + CIN;
      3'd2: F = A - B - 1 + CIN;
      3'd3: F = A + B + CIN;
      3'd4: F = A ^ B;
      3'd5: F = A | B;
      3'd6: F = A & B;
      3'd7: F = 8'b11111111;
      default: F = 8'b0;
    endcase
  end
endmodule
```

# 3.9 组合乘法器

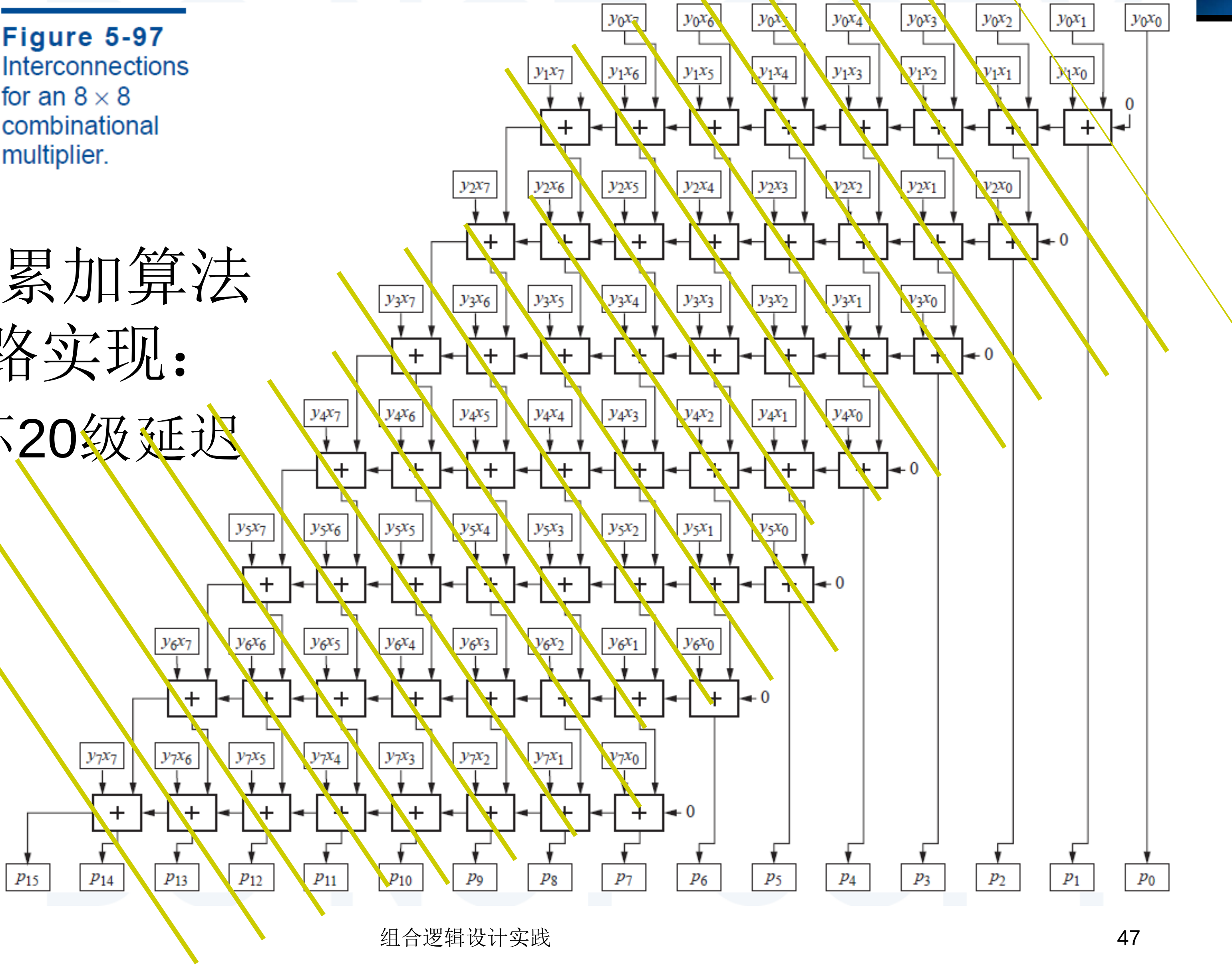
- 移位-累加算法：

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | $y_0x_7$ | $y_0x_6$ | $y_0x_5$ | $y_0x_4$ | $y_0x_3$ | $y_0x_2$ | $y_0x_1$ | $y_0x_0$ |
| | | | | | | | | | $y_1x_7$ | $y_1x_6$ | $y_1x_5$ | $y_1x_4$ | $y_1x_3$ | $y_1x_2$ | $y_1x_1$ | $y_1x_0$ | |
| | | | | | | | | $y_2x_7$ | $y_2x_6$ | $y_2x_5$ | $y_2x_4$ | $y_2x_3$ | $y_2x_2$ | $y_2x_1$ | $y_2x_0$ | | |
| | | | | | | | $y_3x_7$ | $y_3x_6$ | $y_3x_5$ | $y_3x_4$ | $y_3x_3$ | $y_3x_2$ | $y_3x_1$ | $y_3x_0$ | | | |
| | | | | | | $y_4x_7$ | $y_4x_6$ | $y_4x_5$ | $y_4x_4$ | $y_4x_3$ | $y_4x_2$ | $y_4x_1$ | $y_4x_0$ | | | | |
| | | | | | $y_5x_7$ | $y_5x_6$ | $y_5x_5$ | $y_5x_4$ | $y_5x_3$ | $y_5x_2$ | $y_5x_1$ | $y_5x_0$ | | | | | |
| | | | | $y_6x_7$ | $y_6x_6$ | $y_6x_5$ | $y_6x_4$ | $y_6x_3$ | $y_6x_2$ | $y_6x_1$ | $y_6x_0$ | | | | | | |
| + | | | $y_7x_7$ | $y_7x_6$ | $y_7x_5$ | $y_7x_4$ | $y_7x_3$ | $y_7x_2$ | $y_7x_1$ | $y_7x_0$ | | | | | | | |
| | | $p_{15}$ | $p_{14}$ | $p_{13}$ | $p_{12}$ | $p_{11}$ | $p_{10}$ | $p_9$ | $p_8$ | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |

# 3.9 组合乘法器

**Figure 5-97** Interconnections for an 8 × 8 combinational multiplier.

- 移位-累加算法的电路实现：
  - 最坏20级延迟


$y_0x_7$ $y_0x_6$ $y_0x_5$ $y_0x_4$ $y_0x_3$ $y_0x_2$ $y_0x_1$ $y_0x_0$
$y_1x_7$ … $y_1x_0$
$y_7x_7$ … $y_7x_0$
+
0
$p_{15}$ $p_{14}$ $p_{13}$ $p_{12}$ $p_{11}$ $p_{10}$ $p_9$ $p_8$ $p_7$ $p_6$ $p_5$ $p_4$ $p_3$ $p_2$ $p_1$ $p_0$

# 3.9 组合乘法器

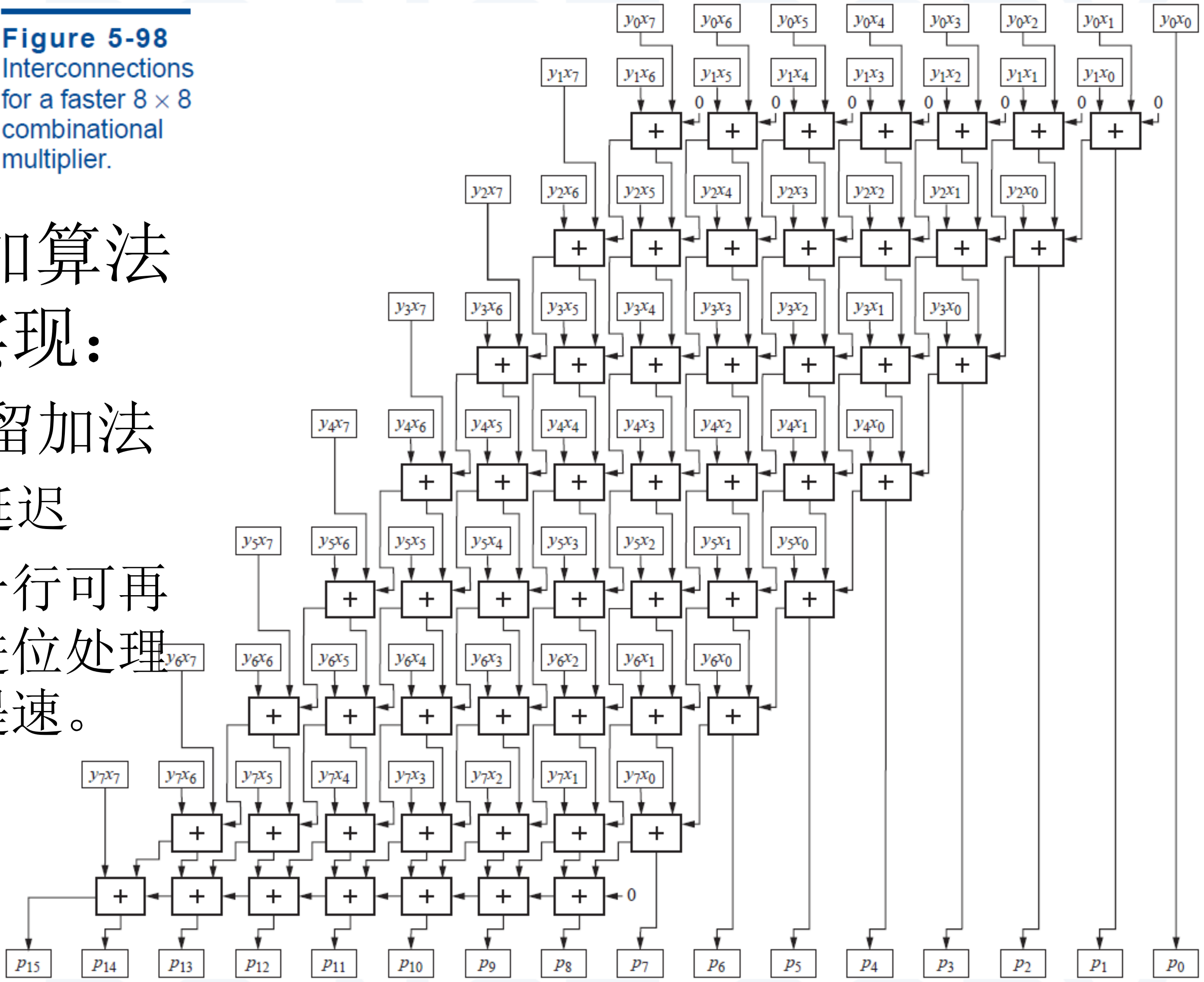


**Figure 5-98** Interconnections for a faster 8 × 8 combinational multiplier.

- 移位-累加算法的电路实现：
  - 进位保留加法
    - 14级延迟
    - 最后一行可再先行进位处理进行提速。

## 3.9 组合乘法器

- Verilog实现
  - 有乘法符号*，用于两个无符号的位向量的乘法。宽度加倍。

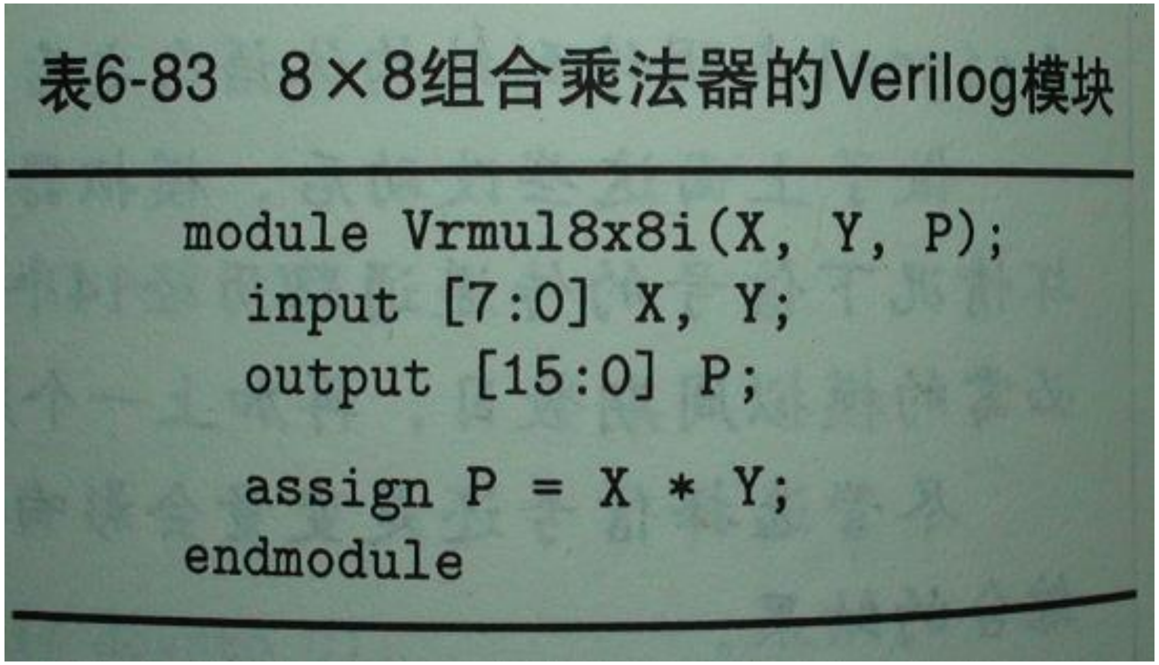


表6-83 8×8组合乘法器的Verilog模块

```
module Vrmul8x8i(X, Y, P);
  input [7:0] X, Y;
  output [15:0] P;

  assign P = X * Y;
endmodule
```

# 3.9 组合乘法器

- Verilog实现

间的关系。

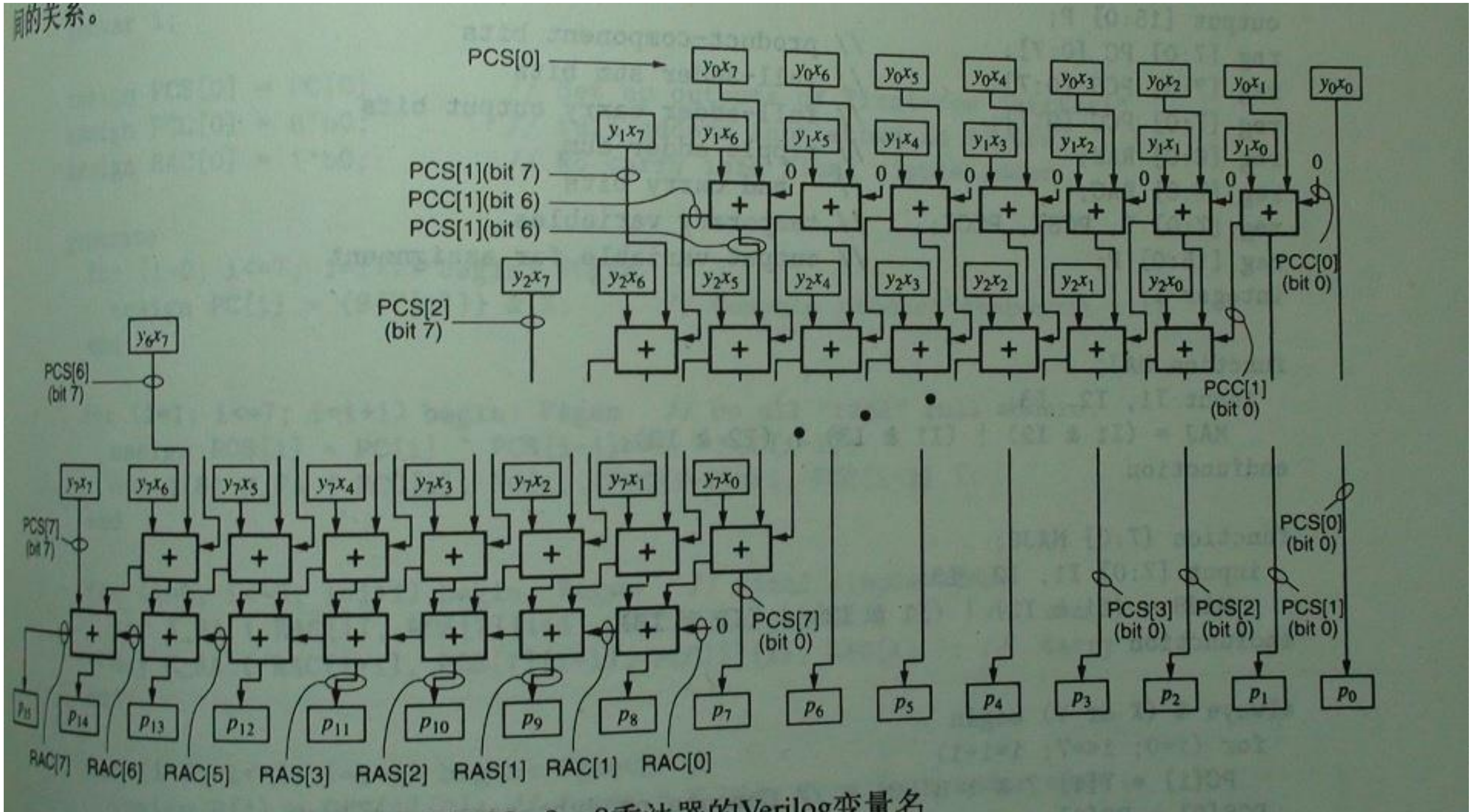


图6-99　8×8乘法器的Verilog变量名

# 3.9 组合乘法器

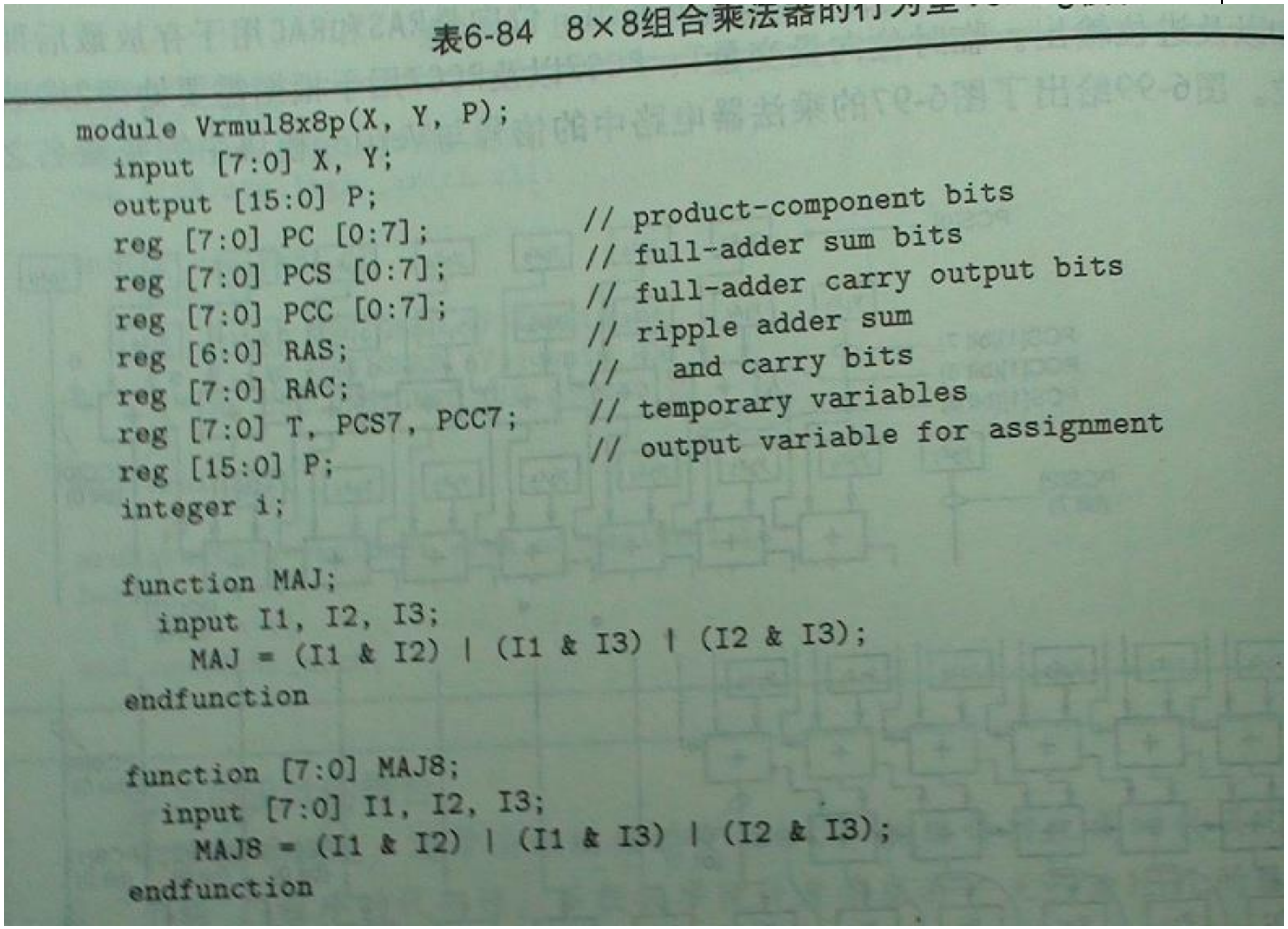
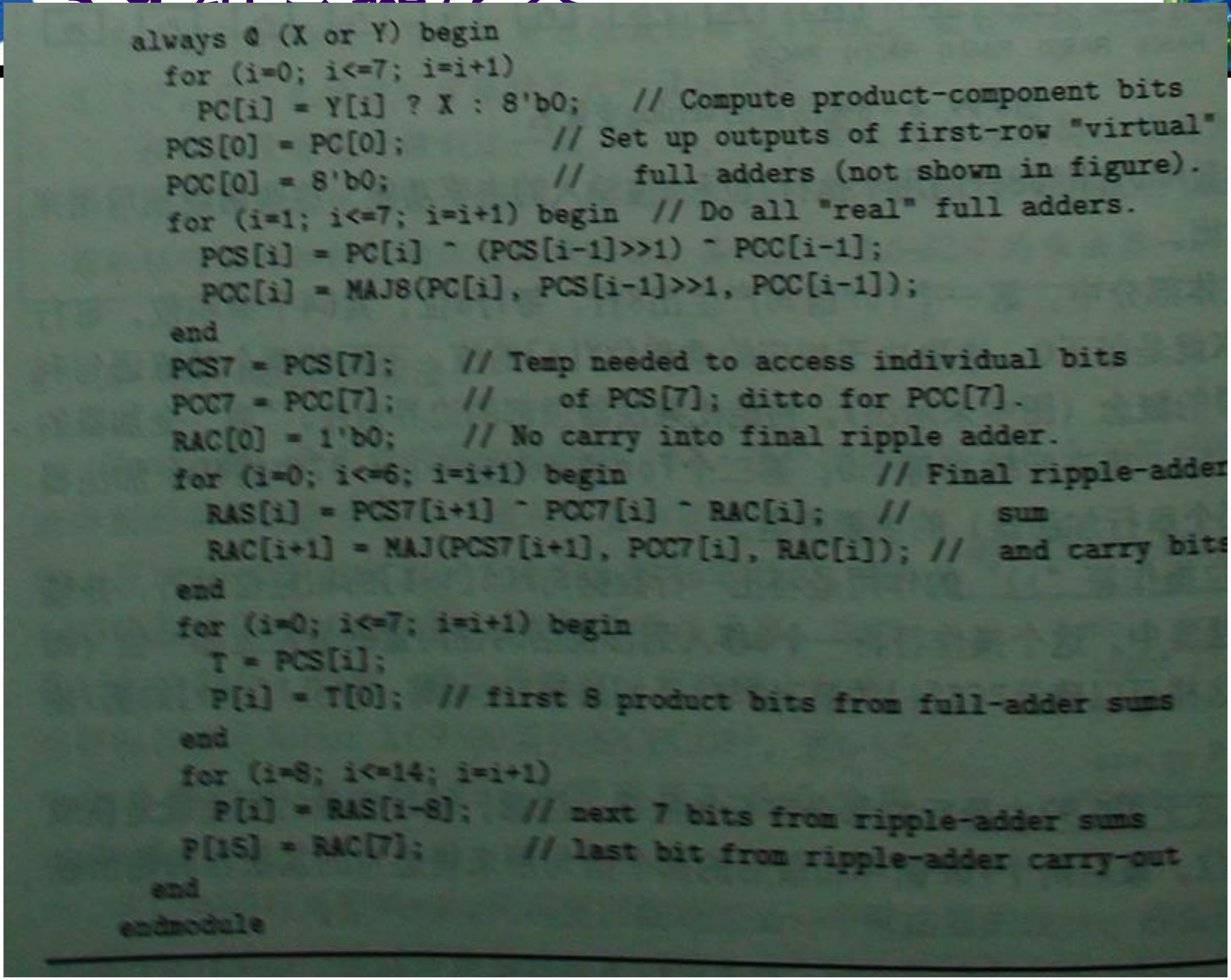
- 

表6-84 8×8组合乘法器的行为型

```
module Vrmul8x8p(X, Y, P);
  input [7:0] X, Y;
  output [15:0] P;
  reg [7:0] PC [0:7];          // product-component bits
  reg [7:0] PCS [0:7];         // full-adder sum bits
  reg [7:0] PCC [0:7];         // full-adder carry output bits
  reg [6:0] RAS;               // ripple adder sum
  reg [7:0] RAC;               //   and carry bits
  reg [7:0] T, PCS7, PCC7;     // temporary variables
  reg [15:0] P;                // output variable for assignment
  integer i;

  function MAJ;
    input I1, I2, I3;
      MAJ = (I1 & I2) | (I1 & I3) | (I2 & I3);
  endfunction

  function [7:0] MAJ8;
    input [7:0] I1, I2, I3;
      MAJ8 = (I1 & I2) | (I1 & I3) | (I2 & I3);
  endfunction
```

# 3.9 组合乘法器

●



```
always @ (X or Y) begin
  for (i=0; i<=7; i=i+1)
    PC[i] = Y[i] ? X : 8'b0;    // Compute product-component bits
  PCS[0] = PC[0];               // Set up outputs of first-row "virtual"
  PCC[0] = 8'b0;                //   full adders (not shown in figure).
  for (i=1; i<=7; i=i+1) begin  // Do all "real" full adders.
    PCS[i] = PC[i] ^ (PCS[i-1]>>1) ^ PCC[i-1];
    PCC[i] = MAJ8(PC[i], PCS[i-1]>>1, PCC[i-1]);
  end
  PCS7 = PCS[7];     // Temp needed to access individual bits
  PCC7 = PCC[7];     //     of PCS[7]; ditto for PCC[7].
  RAC[0] = 1'b0;     // No carry into final ripple adder.
  for (i=0; i<=6; i=i+1) begin                  // Final ripple-adder
    RAS[i] = PCS7[i+1] ^ PCC7[i] ^ RAC[i];      //       sum
    RAC[i+1] = MAJ(PCS7[i+1], PCC7[i], RAC[i]); //   and carry bits
  end
  for (i=0; i<=7; i=i+1) begin
    T = PCS[i];
    P[i] = T[0];  // first 8 product bits from full-adder sums
  end
  for (i=8; i<=14; i=i+1)
    P[i] = RAS[i-8];     // next 7 bits from ripple-adder sums
  P[15] = RAC[7];        // last bit from ripple-adder carry-out
end
endmodule
```

## 3.9 组合

- Verilog实现
  - 详细的设计。

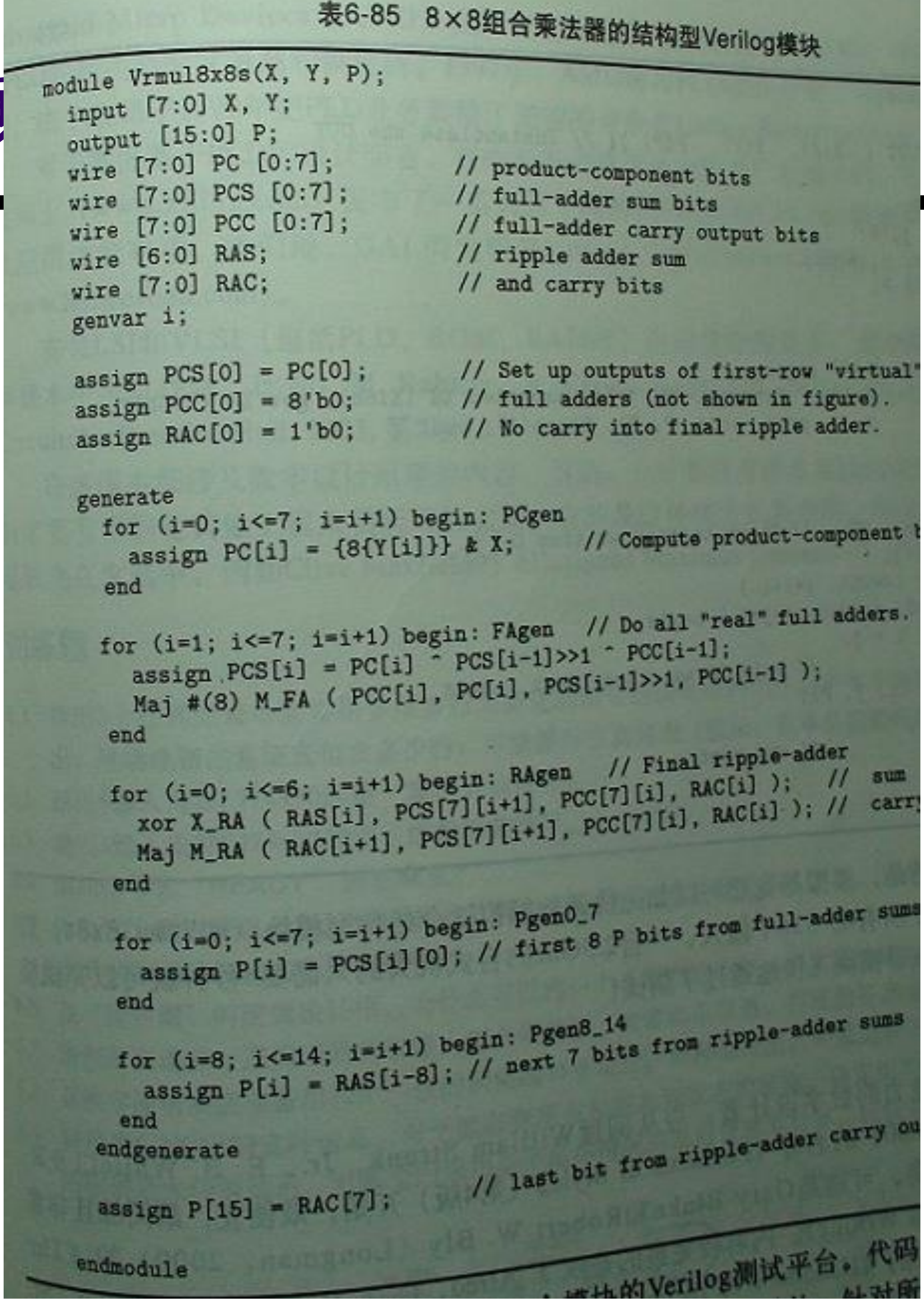


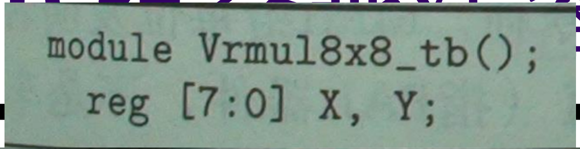
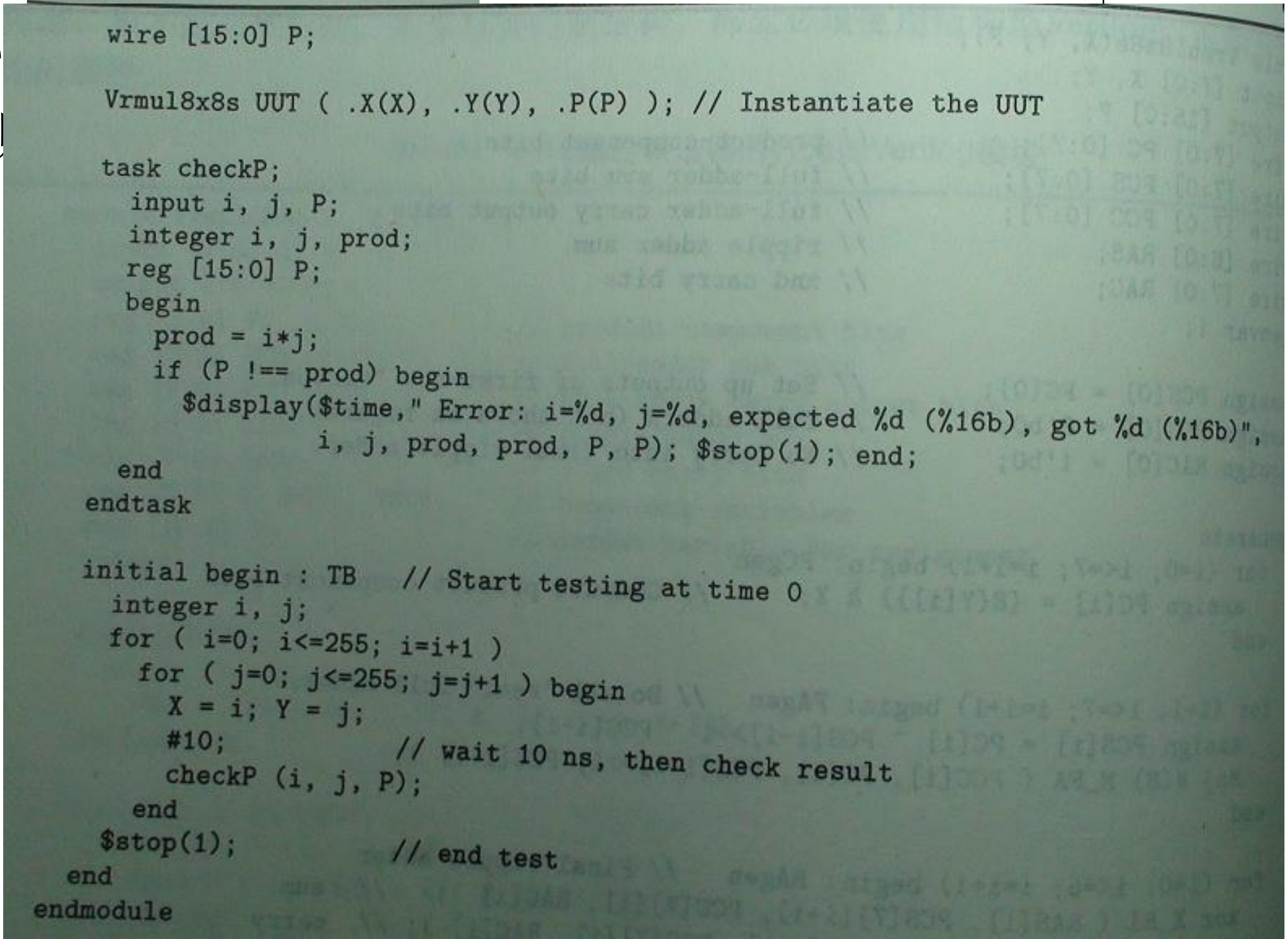
表6-85 8×8组合乘法器的结构型Verilog模块

```
module Vrmul8x8s(X, Y, P);
  input [7:0] X, Y;
  output [15:0] P;
  wire [7:0] PC [0:7];              // product-component bits
  wire [7:0] PCS [0:7];             // full-adder sum bits
  wire [7:0] PCC [0:7];             // full-adder carry output bits
  wire [6:0] RAS;                   // ripple adder sum
  wire [7:0] RAC;                   // and carry bits
  genvar i;

  assign PCS[0] = PC[0];            // Set up outputs of first-row "virtual"
  assign PCC[0] = 8'b0;             // full adders (not shown in figure).
  assign RAC[0] = 1'b0;             // No carry into final ripple adder.

  generate
    for (i=0; i<=7; i=i+1) begin: PCgen
      assign PC[i] = {8{Y[i]}} & X;      // Compute product-component
    end

    for (i=1; i<=7; i=i+1) begin: FAgen  // Do all "real" full adders.
      assign PCS[i] = PC[i] ^ PCS[i-1]>>1 ^ PCC[i-1];
      Maj #(8) M_FA ( PCC[i], PC[i], PCS[i-1]>>1, PCC[i-1] );
    end

    for (i=0; i<=6; i=i+1) begin: RAgen   // Final ripple-adder
      xor X_RA ( RAS[i], PCS[7][i+1], PCC[7][i], RAC[i] );   // sum
      Maj M_RA ( RAC[i+1], PCS[7][i+1], PCC[7][i], RAC[i] ); // carr
    end

    for (i=0; i<=7; i=i+1) begin: Pgen0_7
      assign P[i] = PCS[i][0]; // first 8 P bits from full-adder sums
    end

    for (i=8; i<=14; i=i+1) begin: Pgen8_14
      assign P[i] = RAS[i-8]; // next 7 bits from ripple-adder sums
    end
  endgenerate

  assign P[15] = RAC[7];          // last bit from ripple-adder carry ou
endmodule
```

的Verilog测试平台。代码

# 3.9 组合乘法器

- Ve
  - 测

```
module Vrmul8x8_tb();
  reg [7:0] X, Y;
    wire [15:0] P;

    Vrmul8x8s UUT ( .X(X), .Y(Y), .P(P) ); // Instantiate the UUT

    task checkP;
      input i, j, P;
      integer i, j, prod;
      reg [15:0] P;
      begin
        prod = i*j;
        if (P !== prod) begin
          $display($time," Error: i=%d, j=%d, expected %d (%16b), got %d (%16b)",
                   i, j, prod, prod, P, P); $stop(1); end;
      end
    endtask

    initial begin : TB   // Start testing at time 0
      integer i, j;
      for ( i=0; i<=255; i=i+1 )
        for ( j=0; j<=255; j=j+1 ) begin
          X = i; Y = j;
          #10;              // wait 10 ns, then check result
          checkP (i, j, P);
        end
      $stop(1);             // end test
    end
endmodule
```

## 本章小结

- 要点：
  - 逻辑设计文档的内容及相关标准
    - 每种文档的功效：　知道
    - 有效电平　会用
    - 圈到圈逻辑设计　会用
  - 电路定时
    - 定时图的画法　会读
    - 定时的计算方法　会算
  - 组合PLD
    - 类型、结构　知道
    - 设计（PLA、PAL、GAL）　会用

# 本章小结

- 要点：续
  - 中规模标准器件设计
    - 功能：　　知道
    - 设计：　　读懂
    - 标准芯片：　　会用
      - 级联
      - 扩展等
- 分清楚了解、掌握、会用的含义。
- Verilog程序至少要能读得懂。