第四讲 乘除法器设计

- **常用的机器编码格式**
- **定点乘法器的原理及实现**
  - 原码一位乘法设计
  - 补码一位乘法设计
  - 阵列乘法器设计
- **定点除法器原理及实现**
  - 原码除法器设计
  - 补码除法器设计
  - 阵列除法器设计

4.1 常用机器数的编码格式

- 原码表示法
- 反码表示法
- 补码表示法

4.1常用机器数的编码格式

- **4.1.1 原码表示法**

➢ 原码表示法是一种最简单的机器数表示法, 其最高位为符号位, 符号位为0时表示该数为正, 符号位为1时表示该数为负, 数值部分与真值相同。

➢ **原码形式为$X_S . X_1X_2\cdots X_n$, 其中X_S表示符号位。**

纯小数原码的定义为:

$$[X]_{原} = \begin{cases} X & 0 \le X < 1 \\ 1 - X = 1 + |X| & -1 < X \le 0 \end{cases}$$

纯整数原码的定义为:

$$[X]_{原} = \begin{cases} X, & 0 \leqslant X < 2^n \\ 2^n - X = 2^n + |X|, & -2^n < X \leqslant 0 \end{cases}$$

4.1常用机器数的编码格式

- 原码的优点是直观易懂，机器数和真值间的转换很容易，用原码实现乘、除运算的规则简单。
- 缺点是加、减运算规则较复杂。

4.1常用机器数的编码格式

- **4.1.2 反码表示法**

➢ **反码是对一个数的各位求反。**

- 正数的反码和原码的形式相同；
- 负数的反码是符号位为1，数值部分等于其各位的绝对值求反。

4.1常用机器数的编码格式

- **4.1.3 补码表示法**

➢ 补码的符号位表示方法与原码相同(即正数为0, 负数为1), 其数值部分的表示与数的正负有关:

- 正数: 数值部分与真值形式相同;
- 负数: 将真值的数值部分按位取反, 且在最低位加1。

4.1常用机器数的编码格式

- 补码形式为$X_S . X_1X_2\cdots X_n$，其中X_S表示符号位。

纯小数补码的定义为：

$$[X]_{补}=\begin{cases} X, & 0\leqslant X<1 \\ 2+X=2-|X|, & -1<X\leqslant 0 \end{cases}$$

纯整数补码的定义为：

$$[X]_{补}=\begin{cases} X, & 0\leqslant X<2^n \\ 2^{n+1}+X=2^{n+1}-|X|, & -2^n<X\leqslant 0 \end{cases}$$

- 在补码表示中，真值0的表示形式是唯一的：

$$[+0]_{补}=[-0]_{补}=00000$$

4.2 定点乘法器的原理及实现

- 乘法运算是计算机中一种重要的基本运算,实现方法包括以下几种。

(1) 用软件实现乘法运算。

(2) 在加法器基础上增加一些硬件实现乘法运算。

(3) 设置专用硬件乘法器实现乘法运算。使用该方法会使计算机结构复杂，成本增加，但能使运算速度大大提高。

4.2 定点乘法器的原理及实现

- 乘法运算是计算机中一种重要的基本运算,实现方法包括以下几种。

(1) 用软件实现乘法运算。

(2) 在加法器基础上增加一些硬件实现乘法运算。

(3) 设置专用硬件乘法器实现乘法运算。使用该方法会使计算机结构复杂，成本增加，但能使运算速度大大提高。

4.2 定点乘法器的原理及实现

- 原码一位乘法设计
- 原码二位乘法设计
- 补码一位乘法设计
- 阵列乘法器设计

设计方法

- 模块功能与原理分析
- 模块结构与电路模型
- VHDL语言设计实现
- FPGA验证

4.2.1 原码一位乘法原理及实现

- 原码一位乘法的法则是：

  ①乘积的符号为被乘数的符号位与乘数的符号位相异或；

  ②乘积的绝对值为被乘数的绝对值与乘数的绝对值之积。即

$$[X]_{原} \times [Y]_{原} = (X0 \oplus Y0)(|X| \times |Y|)$$

4.2.1 原码一位乘法原理及实现

- 手工乘法运算

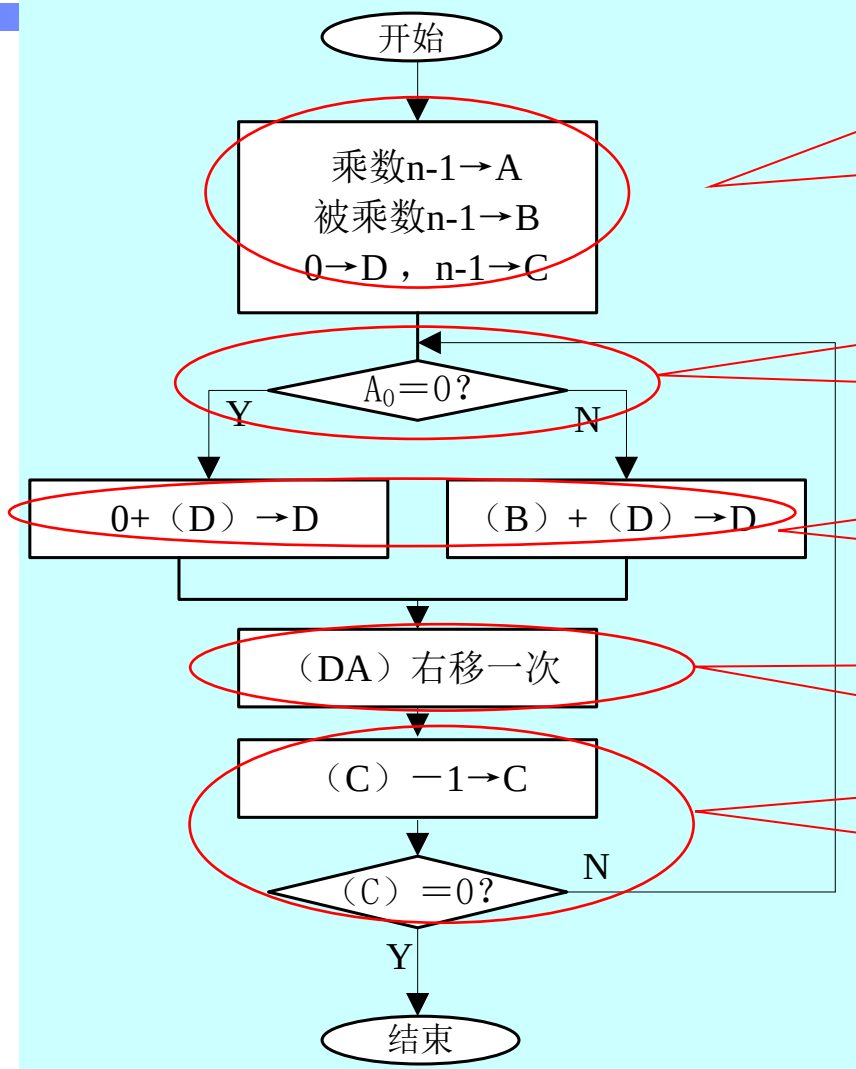
例：若$[X]_{原}=0.1101$，$[Y]_{原}=1.1011$，求两者之积。

解：乘积的符号为$0 \oplus 1 = 1$

手算过程如下：

$$
\begin{array}{r}
1101 \\
\times\ 1011 \\
\hline
1101 \\
1101 \\
0000 \\
1101 \\
\hline
.\ 10001111
\end{array}
$$

4.2.1 原码一位乘法原理及实现



绝对值乘法流程图

A,B,C,D都是寄存器

二选一选择器

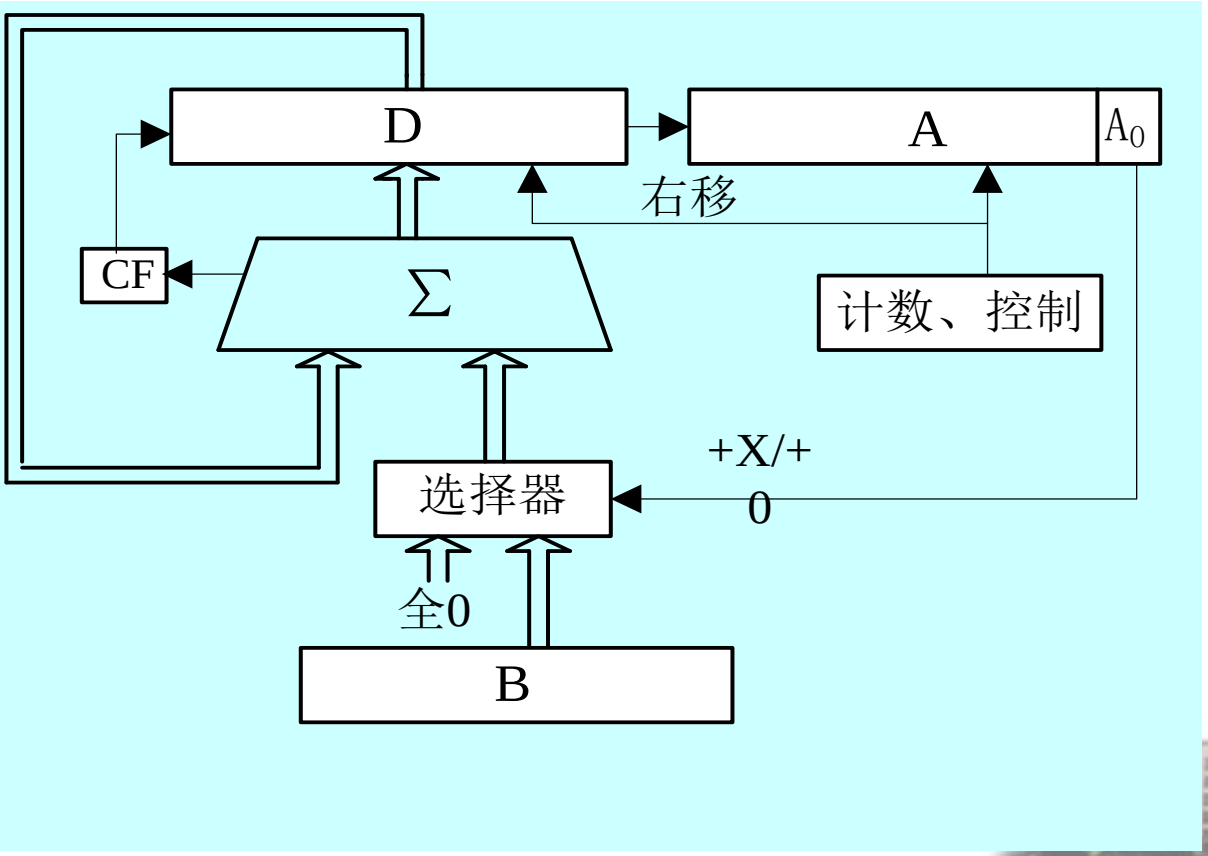
一位全加器

移位寄存器

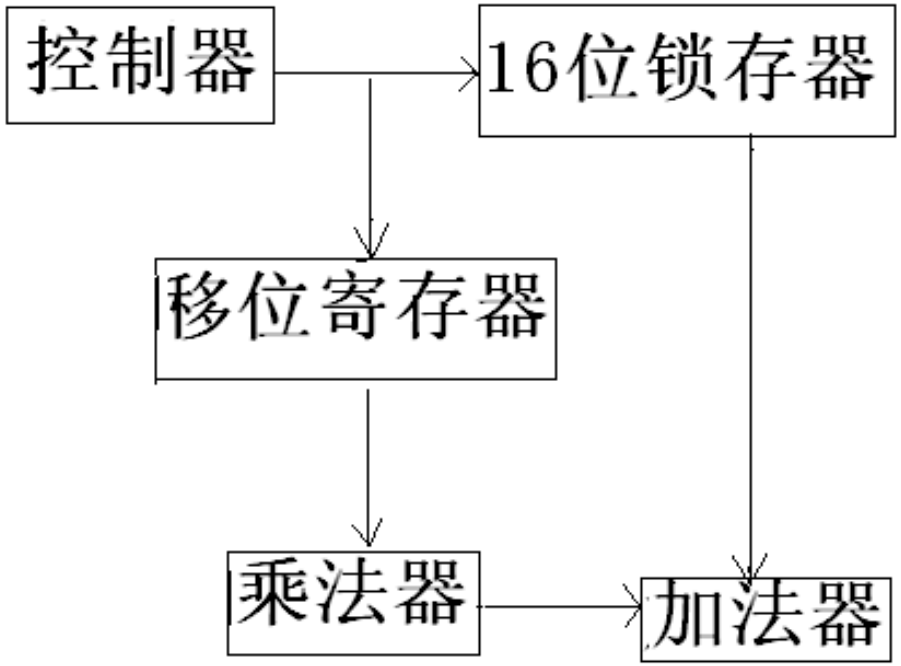
计数器

4.2.1 原码一位乘法原理及实现

- 原码一位乘法器框图



原码一位乘法器功能模块



(1)控制器设计

- **控制器功能：控制移位寄存器和16位寄存器。**
- **端口定义：**

```
PORT (CLK, START : IN STD_LOGIC;
      CLKOUT,RSTALL,DONE: OUT STD_LOGIC );
```

(1)控制器设计

- **输入端口**
- CLK：乘法时钟信号
- START：乘法器启动信号。信号的上跳沿及其高电平有两个功能，即16位寄存器清零和被乘数A[7..0]向移位寄存器加载；低电平则作为乘法使能信号。
- **输出端口**
- CLKOUT:时钟控制端
- RSTALL：清零端口
- DONE：乘法完成标志位

(1)控制器设计

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY ARICTL IS
    PORT (CLK, START : IN STD_LOGIC;
        CLKOUT,RSTALL,DONE: OUT STD_LOGIC );
END ARICTL;
ARCHITECTURE behav OF ARICTL IS
    SIGNAL CNT4B : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
    PROCESS(CLK, START)
    BEGIN
        RSTALL <= START;
        IF START = '1' THEN  CNT4B <= "0000";
        ELSIF CLK'EVENT AND CLK ='1' THEN
          IF CNT4B < 8 THEN CNT4B <= CNT4B + 1; END IF;
        END IF;
    END PROCESS;
    PROCESS(CLK, CNT4B, START)
    BEGIN
        IF START = '0' THEN
            IF CNT4B < 8 THEN  CLKOUT <= CLK;
            ELSE CLKOUT <= '0'; DONE<='1'; END IF;
        ELSE  CLKOUT <= CLK;  DONE<='0';END IF;
    END PROCESS;
END behav;
```

计数器

端口输出控制信号

(2)16位锁存器设计

- 16位锁存器功能：存储部分积及部分积移位
- 端口定义

```
PORT (
        CLK : IN STD_LOGIC;
        CLR : IN STD_LOGIC;
        D : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
        Q : OUT STD_LOGIC_VECTOR(15
    DOWNTO 0)
     );
```

(2)16位锁存器设计

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY REG16B IS                                          -- 16位锁存器
    PORT (
        CLK : IN STD_LOGIC;
        CLR : IN STD_LOGIC;
        D : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
        Q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
    );
END REG16B;
ARCHITECTURE behav OF REG16B IS
    SIGNAL R16S : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
    PROCESS(CLK, CLR)
    BEGIN
     IF CLR = '1' THEN                                    -- 清零信号
     R16S <= "0000000000000000";-- 时钟到来时，锁存输入值，并右移低8位
        ELSIF CLK'EVENT AND CLK = '1' THEN
            R16S(6 DOWNTO 0)  <= R16S(7 DOWNTO 1); -- 右移低8位
            R16S(15 DOWNTO 7) <= D;                -- 将输入锁到高8位
        END IF;
    END PROCESS;
    Q <= R16S;
END behav;
```

移位及锁存功能

(3)移位寄存器

- 移位寄存器功能是右移一位操作。
- 端口定义

```
PORT (
        CLK : IN STD_LOGIC;
        LOAD : IN STD_LOGIC;
    DIN : IN STD_LOGIC_VECTOR(7 DOWNTO
0);
    QB : OUT STD_LOGIC
 );
```

(3)移位寄存器设计

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY SREG8B IS                                          -- 8位右移寄存器
    PORT (  CLK : IN STD_LOGIC;    LOAD : IN STD_LOGIC;
             DIN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
              QB : OUT STD_LOGIC  );
END SREG8B;
ARCHITECTURE behav OF SREG8B IS
    SIGNAL REG8 : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    PROCESS (CLK, LOAD)
    BEGIN
        IF CLK'EVENT AND CLK = '1' THEN
            IF LOAD = '1' THEN                            -- 装载新数据
                REG8 <= DIN;
               ELSE
                                                          -- 数据右移
                REG8(6 DOWNTO 0) <= REG8(7 DOWNTO 1);
            END IF;
        END IF;
    END PROCESS;
    QB <= REG8(0);                                        -- 输出最低位
END behav;
```

移位功能

(4)1位乘法器设计

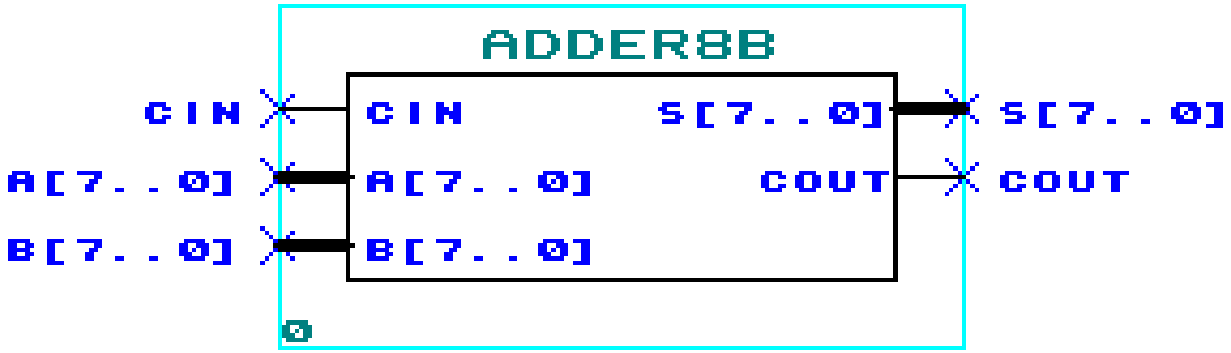
- 1位乘法器功能：当前数据位与另外一个操作数进行与运算。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ANDARITH IS                                    -- 选通与门模块
    PORT ( ABIN : IN STD_LOGIC;
           DIN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          DOUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) );
END ANDARITH;
ARCHITECTURE behav OF ANDARITH IS
BEGIN
    PROCESS(ABIN, DIN)
    BEGIN
        FOR I IN 0 TO 7 LOOP                          -- 循环，完成8位与1位运算
            DOUT(I) <= DIN(I) AND ABIN;
        END LOOP;
    END PROCESS;
END behav;
```

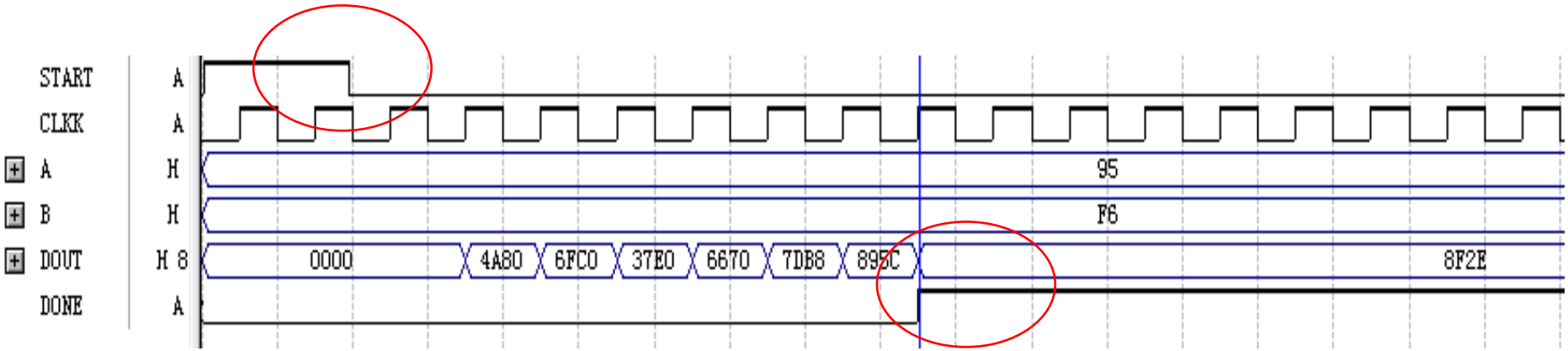
1位乘法运算

(5)加法器设计

- 8位并行加法器设计



(6)仿真结果


START
CLKK
A
B
DOUT
DONE
A
A
H
H
H 8
A
95
F6
0000
4A80
6FC0
37E0
6670
7DB8
895C
8F2E

4.2.2 原码二位乘法器设计

- **原码二位乘法**

Y_{i+1}	Y_i	C	操　作
0	0	0	+0，　右移2次，C＝0
0	0	1	+｜X｜，右移2次，C＝0
0	1	0	+｜X｜，右移2次，C＝0
0	1	1	+2｜X｜，右移2次，C＝0
1	0	0	+2｜X｜，右移2次，C＝0
1	0	1	－｜X｜，右移2次，C＝1
1	1	0	－｜X｜，右移2次，C＝1
1	1	1	+0，　右移2次，C＝1

原码二位乘法的法则表

4.2.2 原码二位乘法器设计

- 例：设X = +0.100111，
  Y = -0.100111，利用原码求积。

符号位	D	A	操作
0 0 0 1 1 1	0 0 0 0 0 0 0 1 1 0 0 1	1 0 0 1 1 1	C=0 —X
1 1 1 1 1 1 0 0 1	0 1 1 0 0 1 1 1 0 1 1 0 0 0 1 1 1 0	0 1 1 0 0 1	C=1 →右移二次 C=1, +2X
0 0 1 0 0 0 0 0 1	0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 1 0	0 0 0 1 1 0	C=0 →右移二次 C=0, +2X
0 0 1 0 0 0	0 1 1 1 1 1 0 1 0 1 1 1	1 1 0 0 0 1	C=0 →右移二次

乘积之符号＝0 ⊕ 1＝1

$[X \cdot Y]_{原}$ = 1.0 1 0 1 1 1 1 1 0 0 0 1

4.2.2 原码二位乘法器设计

$Y_{i+1}=Y_i=C$

Y_{i+1}	Y_i	C	操　作
0	0	0	+0，　右移2次，C=0
0	0	1	+|X|，右移2次，C=0
0	1	0	+|X|，右移2次，C=0
0	1	1	+2|X|，右移2次，C=0
1	0	0	+2|X|，右移2次，C=0
1	0	1	−|X|，右移2次，C=1
1	1	0	−|X|，右移2次，C=1
1	1	1	+0，　右移2次，C=1

4.2.2 原码二位乘法器设计

$Y_{i+1}=0$ && $Y_i \oplus C=1$

Y_{i+1}	Y_i	C	操　作
0	0	0	+0，　右移2次，C＝0
0	0	1	+｜X｜，右移2次，C＝0
0	1	0	+｜X｜，右移2次，C＝0
0	1	1	+2｜X｜，右移2次，C＝0
1	0	0	+2｜X｜，右移2次，C＝0
1	0	1	－｜X｜，右移2次，C＝1
1	1	0	－｜X｜，右移2次，C＝1
1	1	1	+0，　右移2次，C＝1

4.2.2 原码二位乘法器设计

$Y_{i+1} \oplus Y_i = 1$ && $Y_i = C$

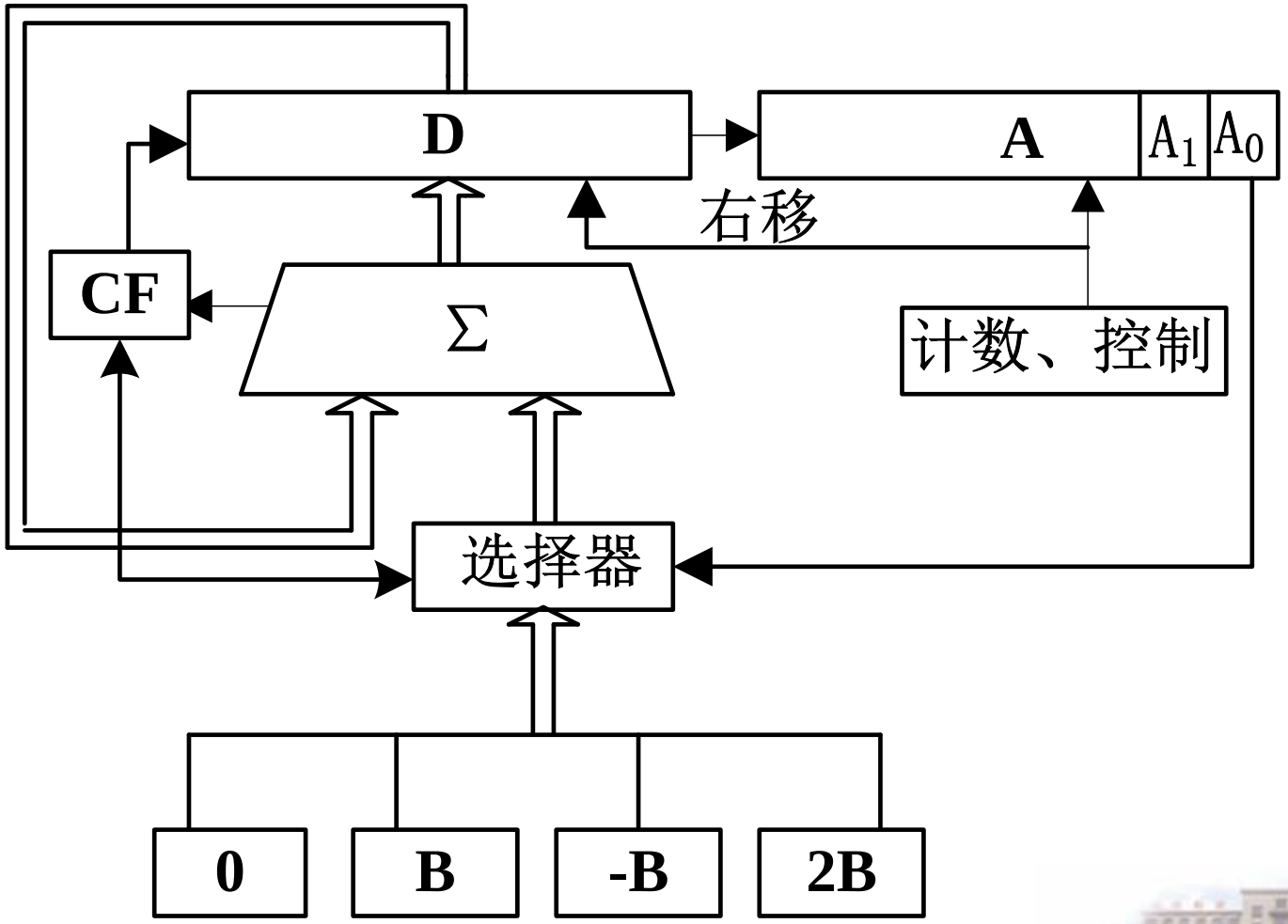
Y_{i+1}	Y_i	C	操　作
0	0	0	+0，　右移2次，C＝0
0	0	1	+｜X｜，右移2次，C＝0
0	1	0	+｜X｜，右移2次，C＝0
0	1	1	+2｜X｜，右移2次，C＝0
1	0	0	+2｜X｜，右移2次，C＝0
1	0	1	－｜X｜，右移2次，C＝1
1	1	0	－｜X｜，右移2次，C＝1
1	1	1	+0，　右移2次，C＝1

4.2.2 原码二位乘法器设计

Y_{i+1}	Y_i	C	操　作
0	0	0	+0，　右移2次，C＝0
0	0	1	+｜X｜，右移2次，C＝0
0	1	0	+｜X｜，右移2次，C＝0
0	1	1	+2｜X｜，右移2次，C＝0
1	0	0	+2｜X｜，右移2次，C＝0
1	0	1	－｜X｜，右移2次，C＝1
1	1	0	－｜X｜，右移2次，C＝1
1	1	1	+0，　右移2次，C＝1

$Y_{i+1}=1$ && $Y_i \oplus C=1$

4.2.2 原码二位乘法器设计



4.2.3 补码一位乘法运算

- **布斯（Booth）法**

➢ 假定被乘数X和乘数Y均为用补码表示的纯小数，其中X0、Y0是它们的符号位：

$$[X]_{补} = X0 . X_{-1}X_{-2}\cdots X_{-(n-1)}$$

$$[Y]_{补} = Y0 . Y_{-1}Y_{-2}\cdots Y_{-(n-1)}$$

➢ 布斯法补码一位乘法的算法公式为：

$$[X\cdot Y]_{补} = [X]_{补}\left[(Y_{-1} - Y_0)\, 2^{0} + (Y_{-2} - Y_{-1})\, 2^{-1} + (Y_{-3} - Y_{-2})\, 2^{-2} + \cdots\cdots + (Y_{-(n-1)} - Y_{-(n-2)})\, 2^{-(n-2)} + (0 - Y_{-(n-1)})\, 2^{-(n-1)} \right]$$

4.2.3 布斯补码一位乘法运算

◆乘数的相邻两位的操作规律

Y_i Y_{i-1}	$Y_{i-1}-Y_i$	操　　作
0　0	0	+0，右移一次
0　1	1	$+[X]_{补}$，右移一次
1　0	−1	$+[-X]_{补}$，右移一次
1　1	0	+0，右移一次

4.2.3 布斯补码一位乘法运算

- 例：已知X = 0.1010, Y = - 0.1101。利用布斯法补码一位乘法求积。

解：首先将两数用补码表示：[X]补 = 00.1010，[Y]补 = 11.0011，而[- X]补 = 11.0110。

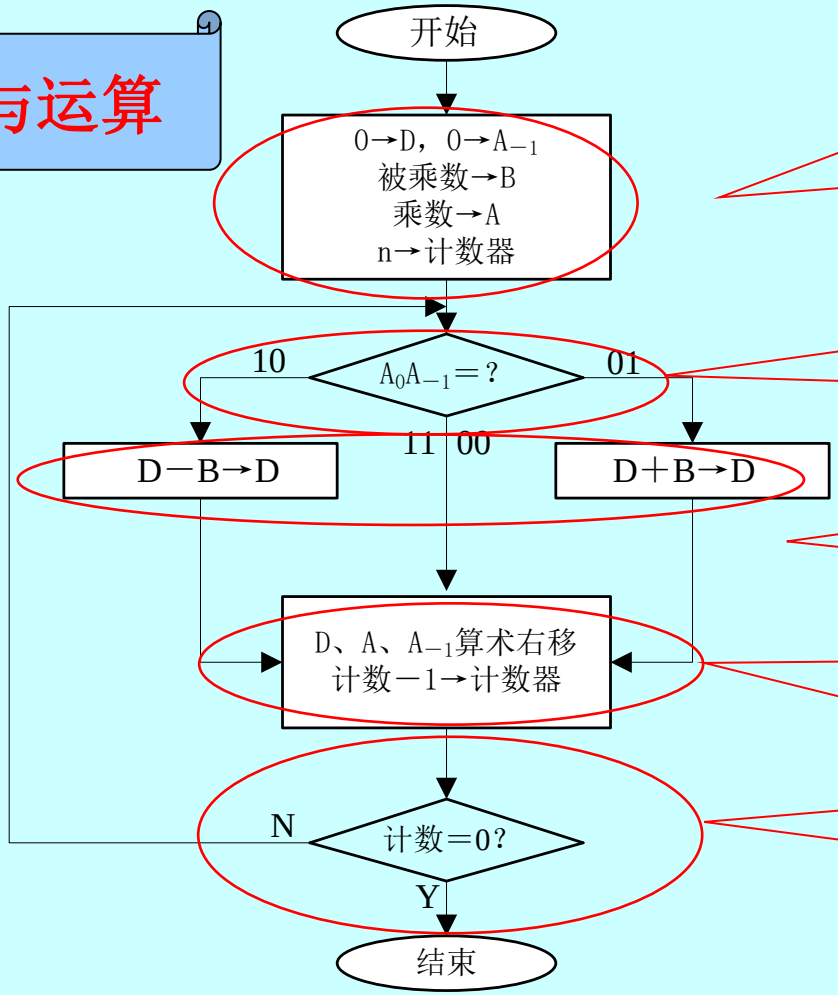
4.2.3 布斯补码一位乘法运算

符号	D	A	A_{-1}	操作
0 0	0 0 0 0	1 0 0 1 1	0	
1 1	0 1 1 0			+[—X]补
1 1	0 1 1 0			
1 1	1 0 1 1	0 1 0 0 1	1	右移一位
0 0	0 0 0 0			+0
1 1	1 0 1 1			
1 1	1 1 0 1	1 0 1 0 0	1	右移一位
0 0	1 0 1 0			+[X]补
0 0	0 1 1 1			
0 0	0 0 1 1	1 1 0 1 0	0	右移一位
0 0	0 0 0 0			+0
0 0	0 0 1 1			
0 0	0 0 0 1	1 1 1 0 1	0	右移一位
1 1	0 1 1 0			+[—X]补
1 1	0 1 1 1			
1 1	1 0 1 1	1 1 1 1 0		右移一位

[X • Y]补 = 1.01111110

4.2.3 布斯补码一位乘法运算

符号位参与运算



布斯补码一位乘法流程图

A,B,C,D都是寄存器

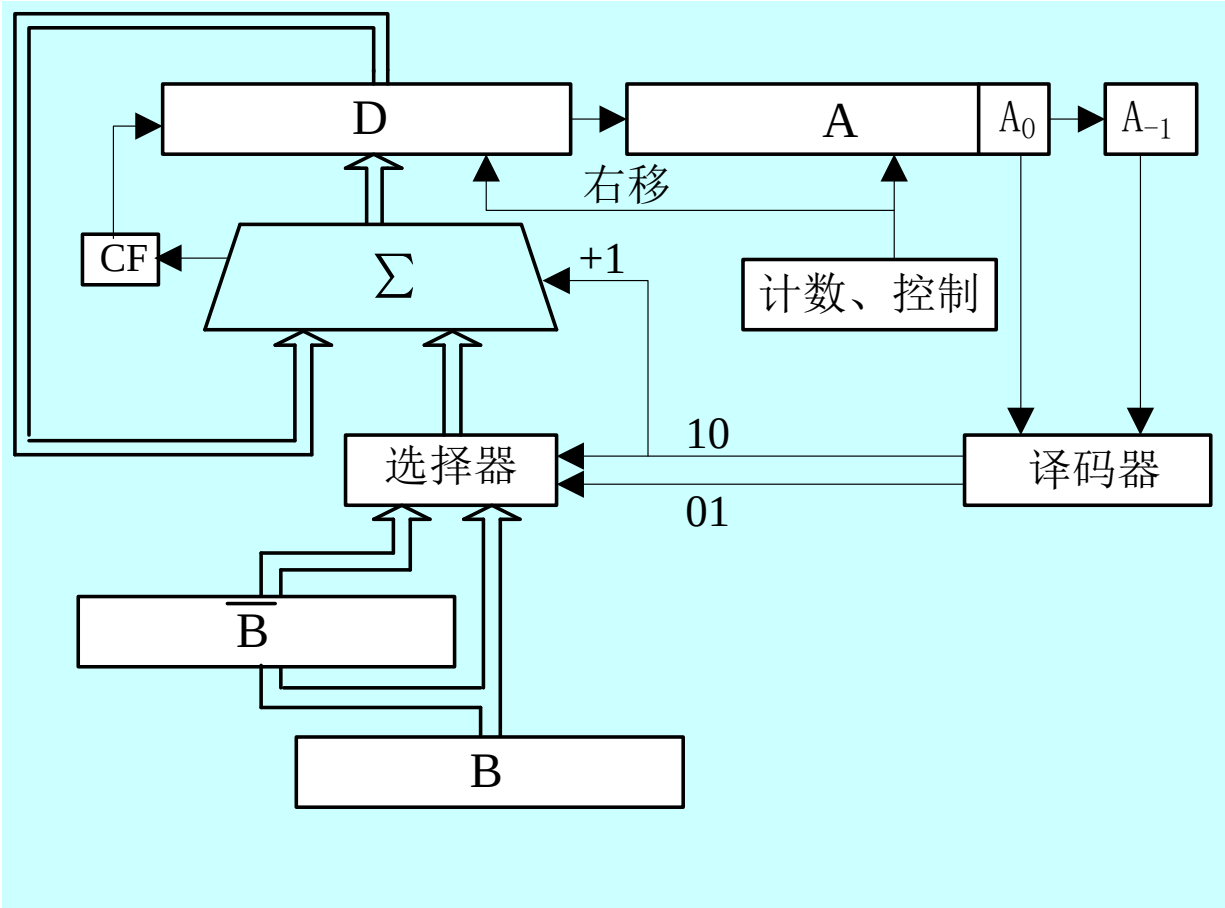
四选一选择器

加法器

移位寄存器

计数器

4.2.3 布斯补码一位乘法运算

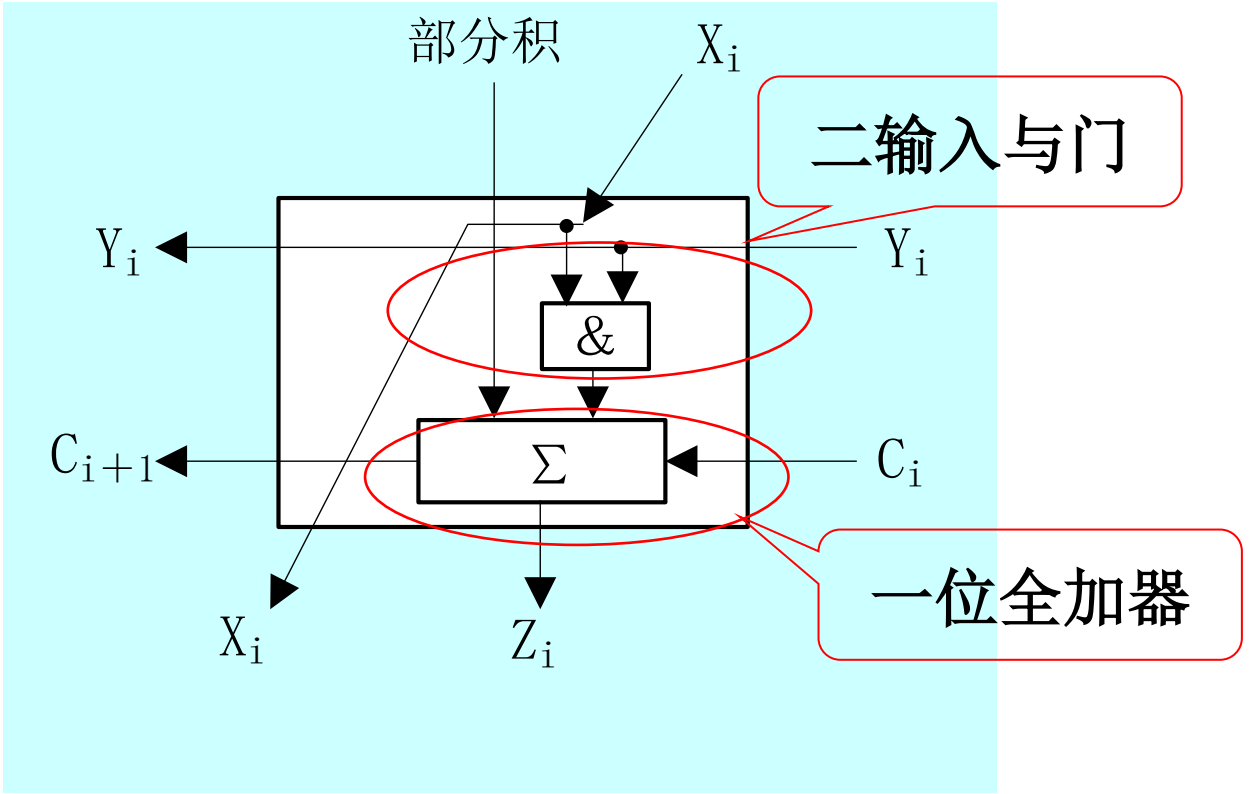


4.2.4 阵列乘法器设计

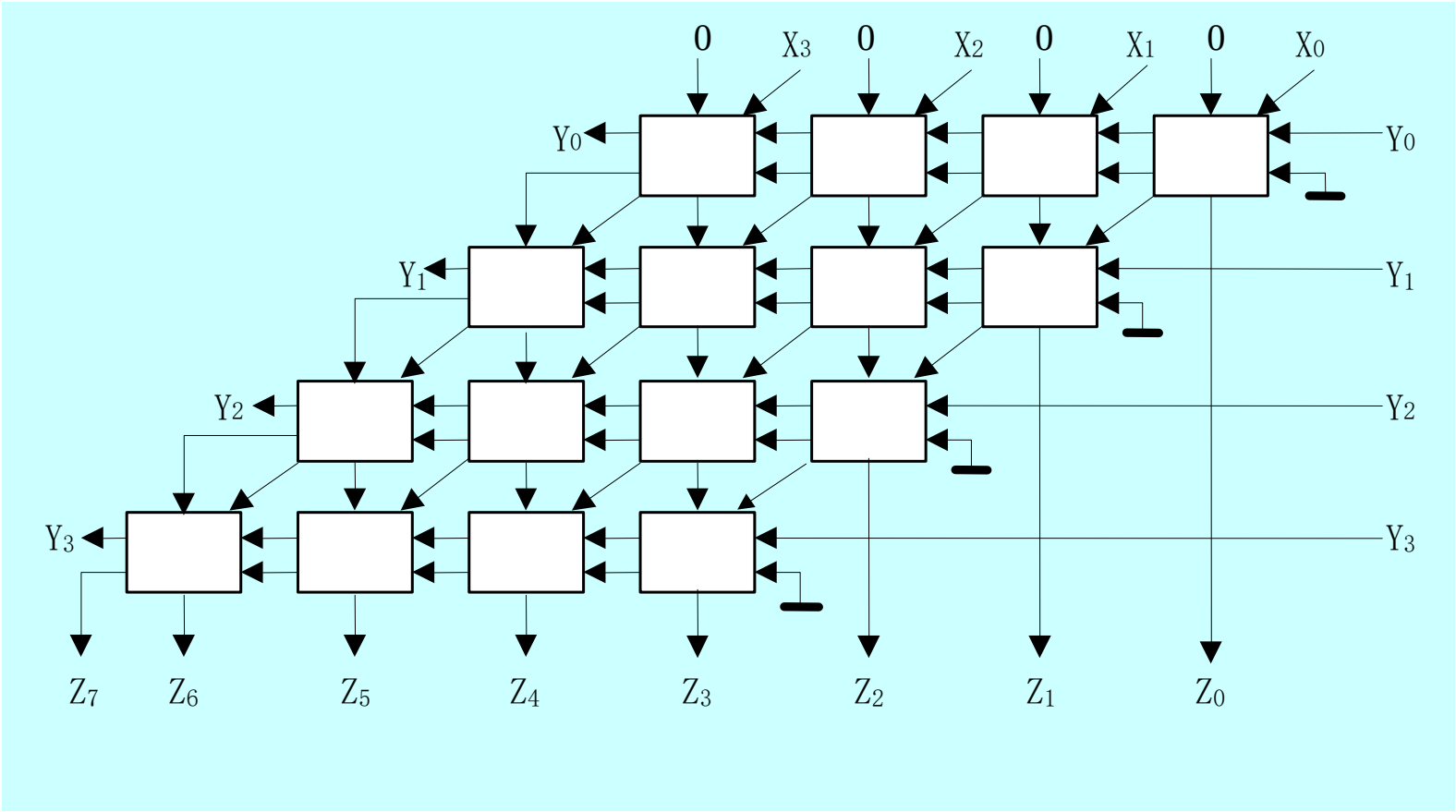
设$X=X_3X_2X_1X_0$，$Y=Y_3Y_2Y_1Y_0$，计算$X\cdot Y=?$

			X_3	X_2	X_1	X_0
		$\times$	Y_3	Y_2	Y_1	Y_0
			X_3Y_0	X_2Y_0	X_1Y_0	X_0Y_0
		X_3Y_1	X_2Y_1	X_1Y_1	X_0Y_1	
	X_3Y_2	X_2Y_2	X_1Y_2	X_0Y_2		
X_3Y3	X_2Y_3	X_1Y_3	X_0Y_3			
Z_6	Z_5	Z_4	Z_3	Z_2	Z_1	Z_0

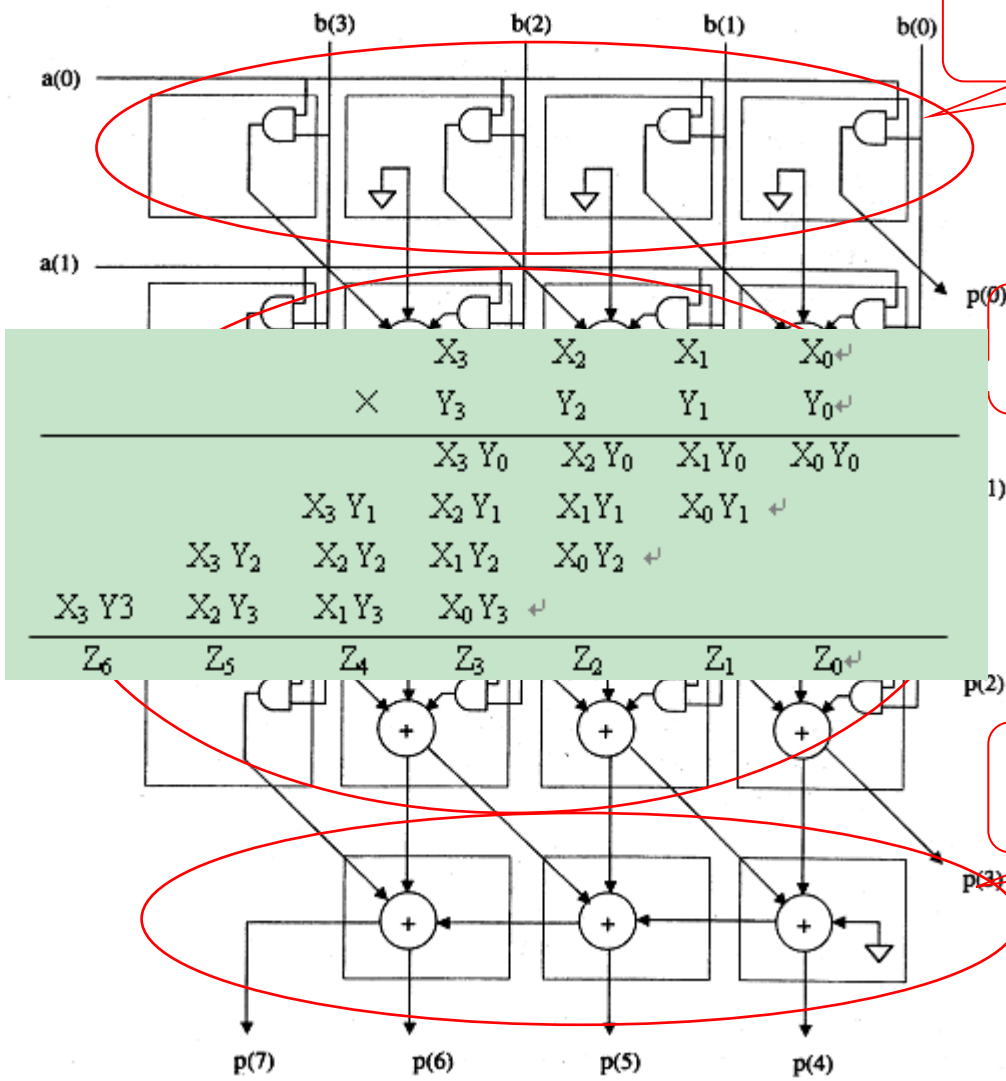
(1)基本乘加单元



(2)定点无符号数阵列乘法器



(2)定点无符号数阵列乘法器



$$
\begin{array}{cccccccc}
 & & & X_3 & X_2 & X_1 & X_0 \\
 & & \times & Y_3 & Y_2 & Y_1 & Y_0 \\
\hline
 & & & X_3Y_0 & X_2Y_0 & X_1Y_0 & X_0Y_0 \\
 & & X_3Y_1 & X_2Y_1 & X_1Y_1 & X_0Y_1 & \\
 & X_3Y_2 & X_2Y_2 & X_1Y_2 & X_0Y_2 & & \\
X_3Y3 & X_2Y_3 & X_1Y_3 & X_0Y_3 & & & \\
\hline
Z_6 & Z_5 & Z_4 & Z_3 & Z_2 & Z_1 & Z_0
\end{array}
$$

(3)TOP_ROW功能实现

```
1  --------top_row.vhd(component): -----------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE work.my_components.all;
5  ------------------------------------------------
6  ENTITY top_row IS
7     PORT (a: IN STD_LOGIC;
8           b: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
9           sout, cout: OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
10          p: OUT STD_LOGIC);
11 END top_row;
12 ------------------------------------------------
13 ARCHITECTURE structural OF top_row IS
14 BEGIN
15    U1: COMPONENT and_2 PORT MAP(a, b(3), sout(2));
16    U2: COMPONENT and_2 PORT MAP(a, b(2), sout(1));
17    U3: COMPONENT and_2 PORT MAP(a, b(1), sout(0));
18    U4: COMPONENT and_2 PORT MAP(a, b(0), p);
19    cout(2) <= '0'; cout(1) <= '0'; cout(0) <= '0';
20 END structural;
```

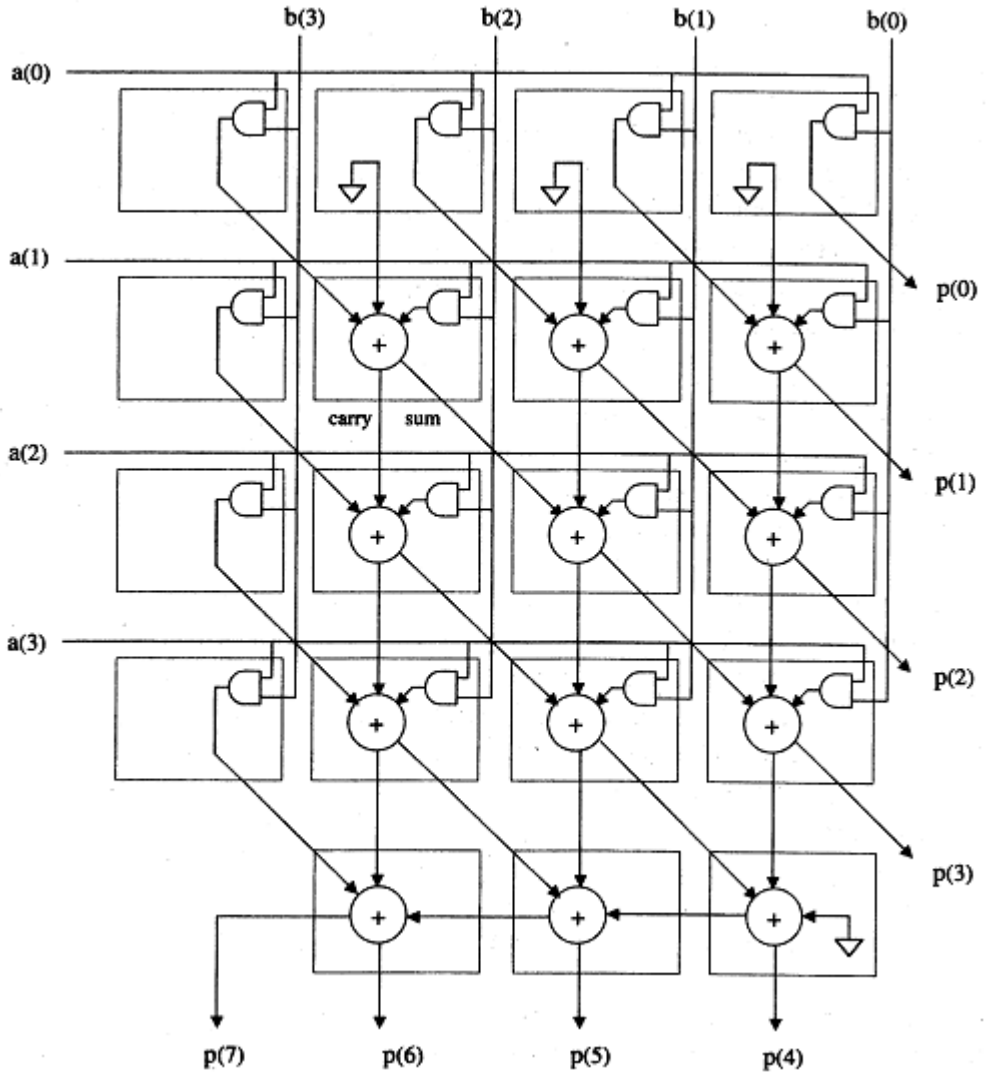
(4)MID_ROW功能实现

```
1  ---------mid_row.vhd(component): ---------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE work.my_components.all;
5  ------------------------------------------------
6  ENTITY mid_row IS
7     PORT (a: IN STD_LOGIC;
8           b: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
9           sin, cin: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
10          sout, cout: OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
11          p: OUT STD_LOGIC);
12 END mid_row;
13 ------------------------------------------------
14 ARCHITECTURE structural OF mid_row IS
15    SIGNAL and_out: STD_LOGIC_VECTOR(2 DOWNTO 0);
16 BEGIN
17    U1: COMPONENT and_2 PORT MAP(a, b(3), sout(2));
18    U2: COMPONENT and_2 PORT MAP(a, b(2), and_out(2));
19    U3: COMPONENT and_2 PORT MAP(a, b(1), and_out(1));
20    U4: COMPONENT and_2 PORT MAP(a, b(0), and_out(0));
21    U5: COMPONENT fau PORT MAP(sin(2), cin(2), and_out(2)
22       sout(1), cout(2));
23    U6: COMPONENT fau PORT MAP(sin(1), cin(1), and_out(1)
24       sout(0), cout(1));
25    U7: COMPONENT fau PORT MAP(sin(0), cin(0), and_out(0)
26       p, cout(0));
27 END structural;
```

(5)LOW_ROW功能实现

```
1  ---------lower_row.vhd(component): --------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE work.my_components.all;
5  ------------------------------------------------
6  ENTITY lower_row IS
7     PORT (sin, cin: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
8           p: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
9  END lower_row;
10 ------------------------------------------------
11 ARCHITECTURE structural OF lower_row IS
12    SIGNAL local: STD_LOGIC_VECTOR(2 DOWNTO 0);
13 BEGIN
14    local(0) <= '0';
15    U1: COMPONENT fau PORT MAP(sin(0), cin(0), local(0),
16       p(0), local(1));
17    U2: COMPONENT fau PORT MAP(sin(1), cin(1), local(1),
18       p(1), local(2));
19    U3: COMPONENT fau PORT MAP(sin(2), cin(2), local(2),
20       p(2), p(3));
21 END structural;
```

(6)阵列乘法器结构描述实现



4.3 定点数除法运算

- 定点数除法分为原码除法和补码除法两类。
- 除法实现方法

①双操作数加法器将除法分为若干次“加减与移位”的循环，由时序控制部分实现；

②采用迭代除法，将除法转换为乘法处理，可以利用快速乘法器实现除法器；

③阵列除法器，一次求得商与余数，实现快速除法的基本途径。

4.3.1 原码除法运算

- 原码除法的法则应包括：

①除数≠0；定点纯小数时，｜被除数｜ < ｜除数｜；定点纯整数时，｜被除数｜ > ｜除数｜。

②与原码乘法类似的是原码除法商的符号和商的值也是分别处理的，商的符号等于被除数的符号与除数的符号相异或。

③商的值等于被除数的绝对值除以除数的绝对值。

④将商的符号与商的值拼接在一起就得到原码除法的商。

4.3.2 定点除法器的原理及实现

◆恢复余数法：先减后判，如果减后发现不够减，则上商0，并加上除数，即恢复到减操作之前的余数（第一步的余数即被除数）。

◆其缺点是即增加了一些不必要的操作，又使操作步数随着不够减情况发生的次数而变。

4.3.2 定点除法器的原理及实现

- 不恢复余数除法（加减交替除法）
  - 先减后判，如果发现不够减，则上商0，并将下一步的减除数操作改为加除数操作。
  - 这样可使操作步数固定，只与所需商的位数有关，而与是否够减无关，因此能减少运算时间。

4.3.3 原码加减交替除法器

- **原码加减交替除法器的运算法则：**

1.除法运算前，应满足条件：$X^* < Y^*$,且$Y^* \neq 0$,否则，按溢出或非法除数处理；

2.符号位不参与运算，单独处理：$q_f = x_f \oplus y_f$；

3.部分余数采用单符号位或双符号位；

4.每步部分余数运算规则：

①若余数R≥0, 则商上1，左移一次，减除数；

②若余数R < 0, 则商上0，左移一次，加除数。

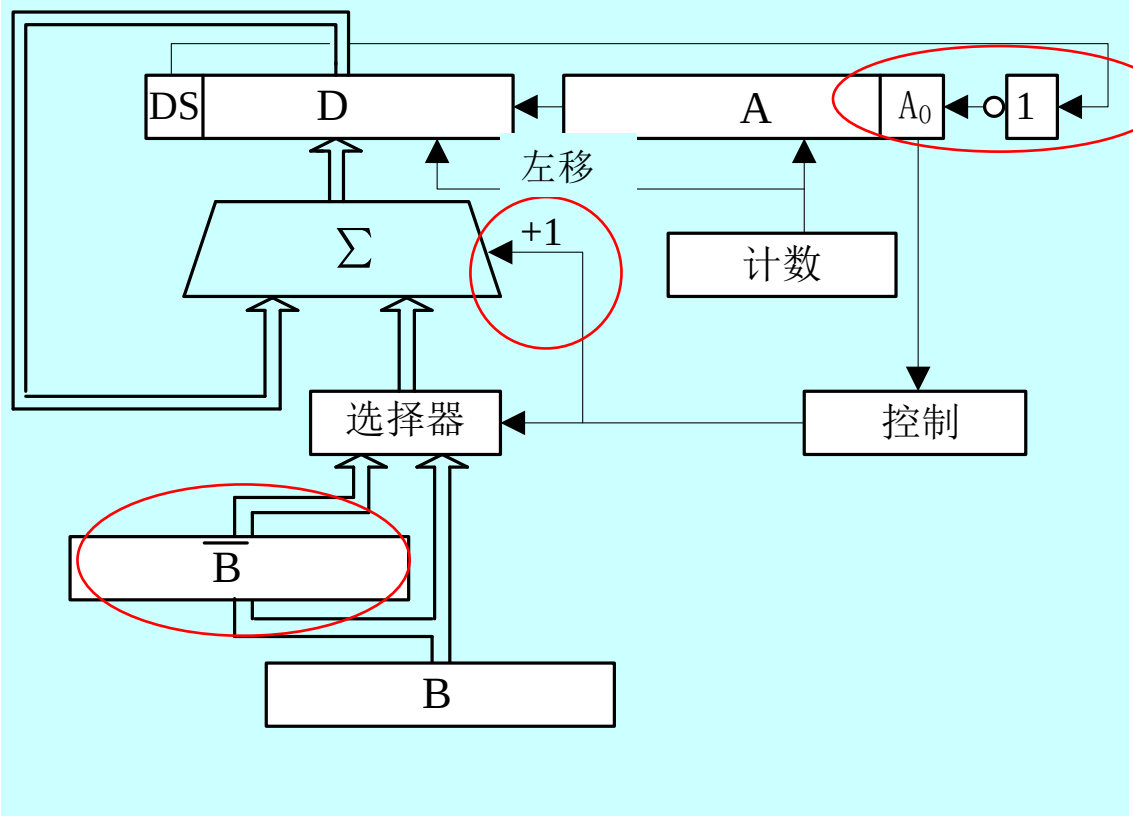
4.3.3 原码加减交替除法器

- 例:若X = －0.10001011，Y = 0.1110 试利用原码加减交替除法器求商及余数。

  解：写出［X］原 = 1.10001011，［Y］原 = 0.1110。 商符 = 1⊕0 = 1；

符号	被除数（余数）	商	操作
0 0	1 0 0 0 1 0 1 1	0	
0 1	0 0 0 1 0 1 1 0		左移一位
1 1	0 0 1 0		－|Y|
0 0	0 0 1 1 0 1 1 0	1	R≥0，商为1
0 0	0 1 1 0 1 1 0 1		左移一位
1 1	0 0 1 0		－|Y|
1 1	1 0 0 0 1 1 0 1	0	R<0，商为0
1 1	0 0 0 1 1 0 1 0		左移一位
0 0	1 1 1 0		+|Y|
1 1	1 1 1 1 1 0 1 0	0	R<0，商为0
1 1	1 1 1 1 0 1 0 0		左移一位
0 0	1 1 1 0		+|Y|
0 0	1 1 0 1 0 1 0 0	1	R≥0，商为1

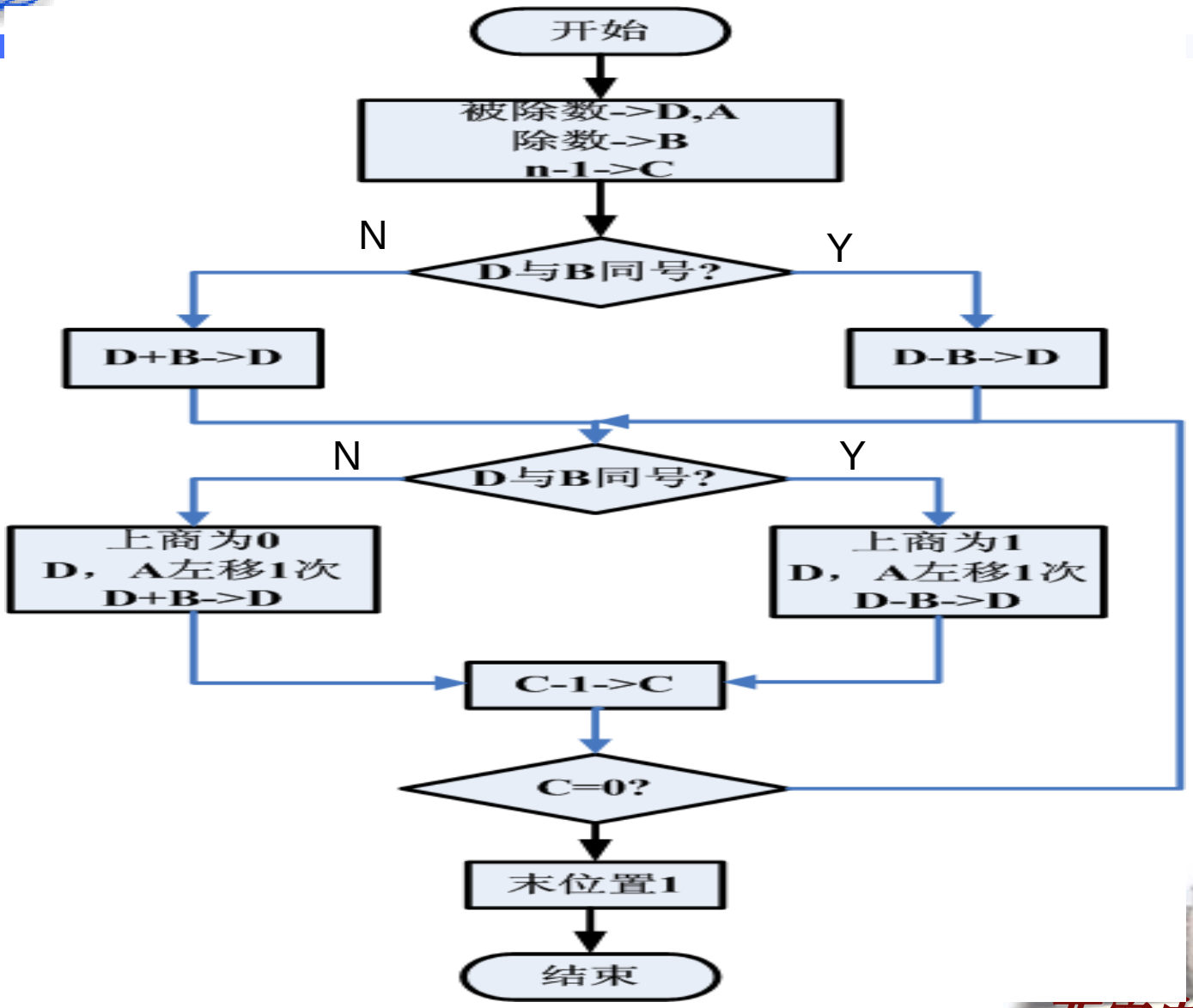
4.3.3 原码加减交替除法器



4.3.4 补码除法运算

- ①符号判断。被除数和除数同号，被除数减除数；若异号则加除数。
- ②余数与除数同号，上商为1，余数左移1位，下次用余数减除数操作求商。若异号，上商为0，余数左移1位，下次用余数加除数操作求商。
- ③重复②直至除尽或达到精度要求。
- ④商修正。在除不尽时，最低位恒置1修正。

4.3.4 补码除法运算


开始
被除数->D,A
除数->B
n-1->C
N
D与B同号?
Y
D+B->D
D-B->D
N
D与B同号?
Y
上商为0
D，A左移1次
D+B->D
上商为1
D，A左移1次
D-B->D
C-1->C
C=0?
末位置1
结束

4.3.4 补码除法运算

- 例：若X = -0.10001011，Y = 0.1110 试利用补码除法求商及余数。

解：写出

[X] 补 = 1.01110101

，

[Y] 补 = 0.1110。

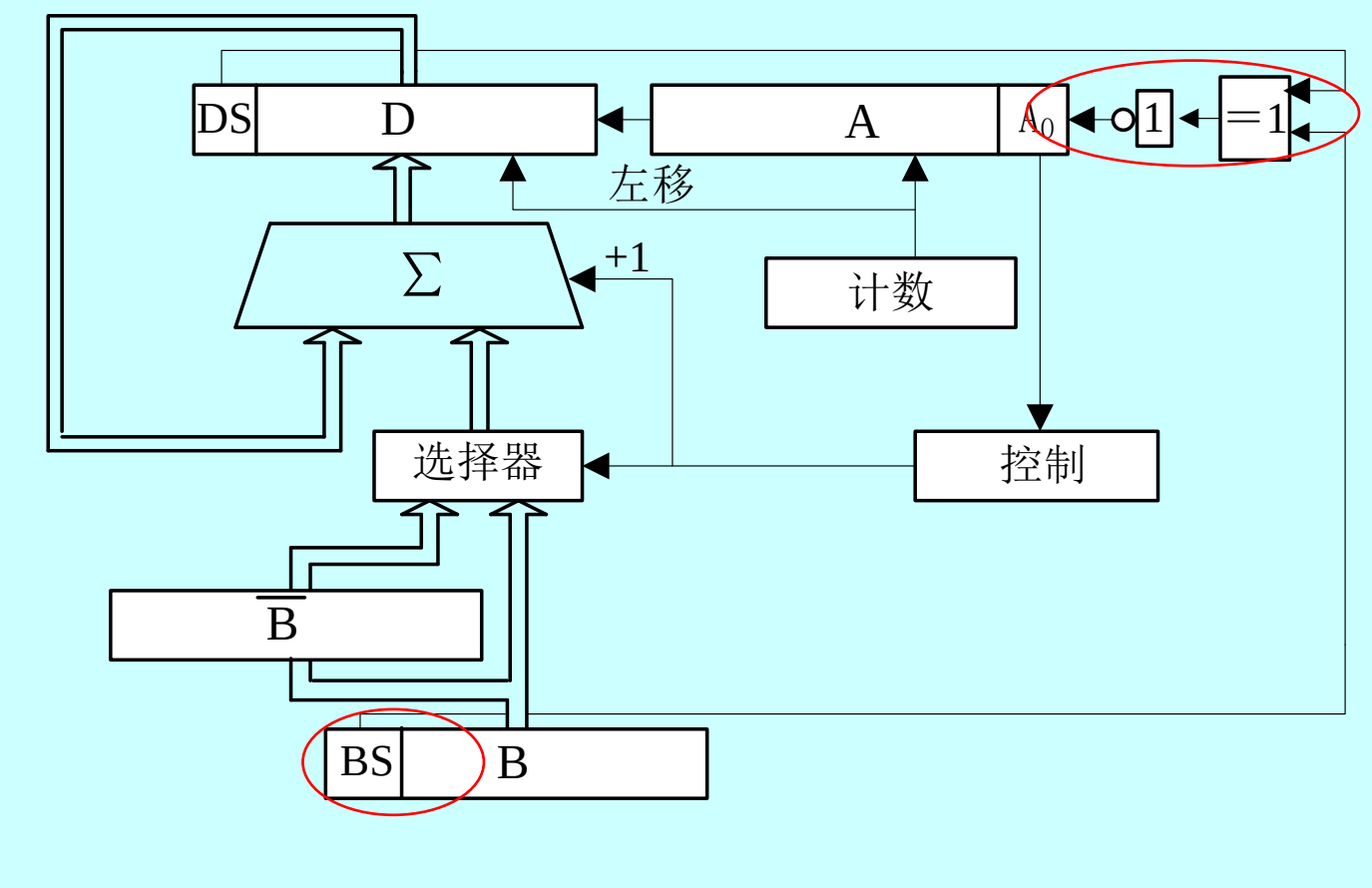
[- Y] 补 = 1.0010

符号	被除数（余数）	商	操作
1 1	0 1 1 1 0 1 0 1		X,Y 异号
0 0	1 1 1 0		＋〔Y〕补
0 0	0 1 0 1	1	R与Y同号,上商1
0 0	1 0 1 0 1 0 1 1		左移一位
1 1	0 0 1 0		＋〔－Y〕补
1 1	1 1 0 0	0	R与Y异号,上商0
1 1	1 0 0 1 0 1 1 0		左移一位
0 0	1 1 1 0		＋〔Y〕补
0 0	0 1 1 1	1	R与Y同号,上商1
0 0	1 1 1 0 1 1 0 1		左移一位
1 1	0 0 1 0		＋〔－Y〕补
0 0	0 0 0 0	1	R与Y同号,上商1
0 0	0 0 0 1 1 0 1 1		左移一位
1 1	0 0 1 0		＋〔－Y〕补
1 1	0 0 1 1	0	R与Y异号,上商0
1 1	0 0 1 1 1 0 1 1		

〔商〕补 = **1.01101**；余数为：〔余数〕补 = **1.0011**×2^{-4}

4.3.4 补码除法运算

- 补码除法器框图



4.3.4 补码除法设计

- 端口定义

```
port( oper_a,oper_b:  in std_logic_vector(7 downto 0);--被除数，除数，最高位为符号位
        done: out std_logic;--完成除法操作标志
        clk,rst:in std_logic;--时钟信号/复位信号
        Q,R:  out std_logic_vector(7 downto 0)--商Q最高位为符号位，余数R
    );
```

4.3.4 补码除法设计

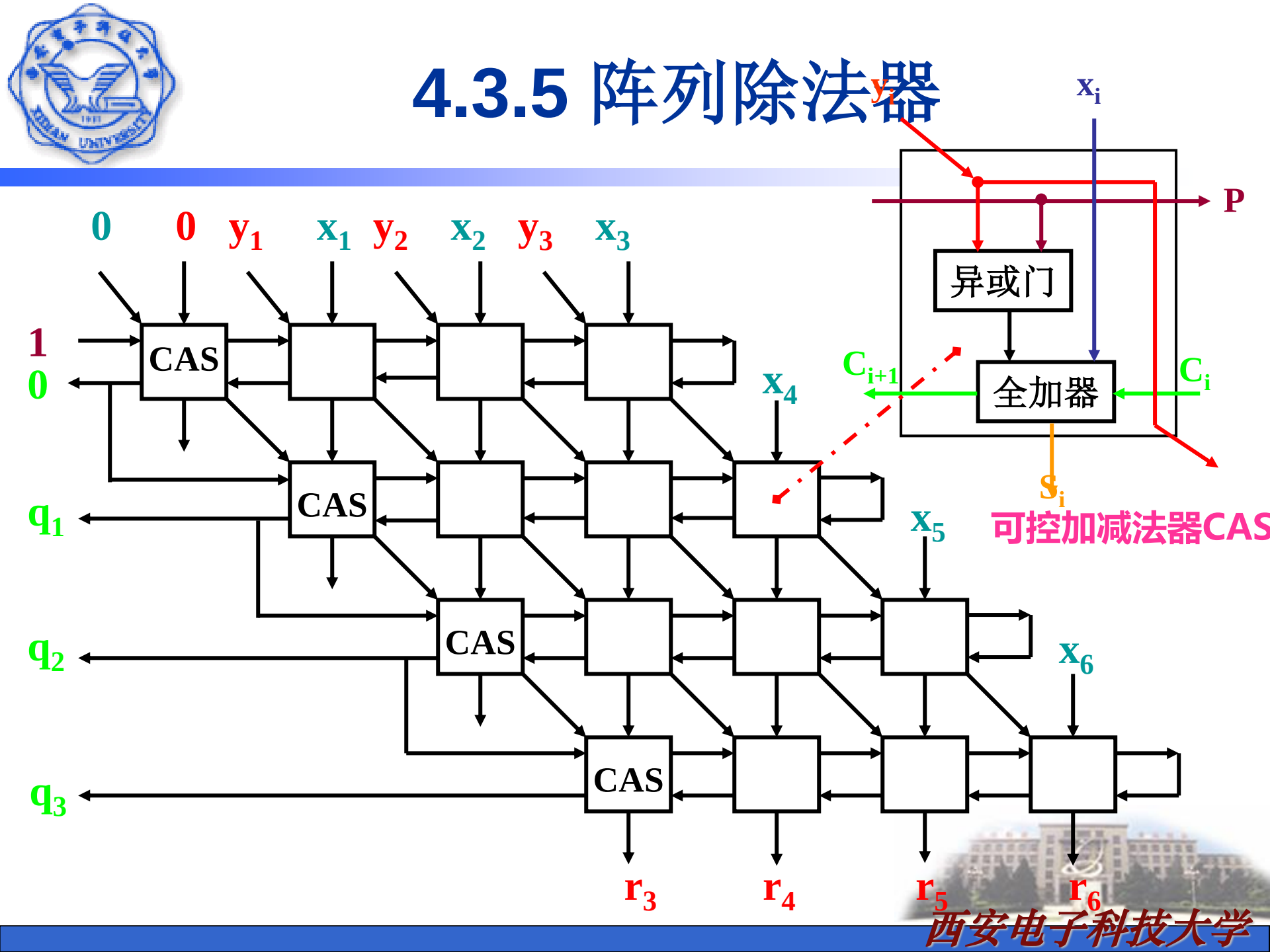
```
architecture Behavioral of div8_c is
constant  itera: integer :=7; --累加移位次数
begin
        process(clk,rst)
                variable Q_temp:std_logic_vector(7 downto 0);--商
                variable count :integer range 0 to 7;--累加和移位计数器
                variable R_temp,B:std_logic_vector(8 downto 0);--余数和除数，双符号位，运算开始时余数即为被除数
      begin
                        if(rst='1') then
                                Q_temp(7 downto 0):="00000000";
                                R_temp(8 downto 0):=oper_a(7)&oper_a(7 downto 0);
                                B(8 downto 0):=oper_b(7)&oper_b(7 downto 0);
                                count:=itera;
                                done<='0';
                        elsif(clk='1' and clk'event) then
                                if(count>0) then
                                        if(R_temp(8)=B(8))then Q_temp(0):='1';
                                                else Q_temp(0):='0';
                                        end if;
                                        Q_temp(7 downto 0):=Q_temp(6 downto 0)&'0';
                                        R_temp(8 downto 0):=R_temp(7 downto 0)&'0';
                                        if(R_temp(8)=B(8))then R_temp:=R_temp-B;
                                                else R_temp:=R_temp+B;
                                        end if;
                                        count:=count-1;
                                end if;
                        elsif(count=0) then
                                        Q_temp(7 downto 0):=(not Q_temp(7)) &Q_temp(6 downto 1)&'1';
                                        done<='1';
                                        count:=itera;
                        end if;
                        Q<=Q_temp;
                        R<=R_temp(7 downto 0);
end process;
end Behavioral;
```

复位赋初值

判断余数与除数符号是否相同

运算结束

4.3.5 阵列除法器

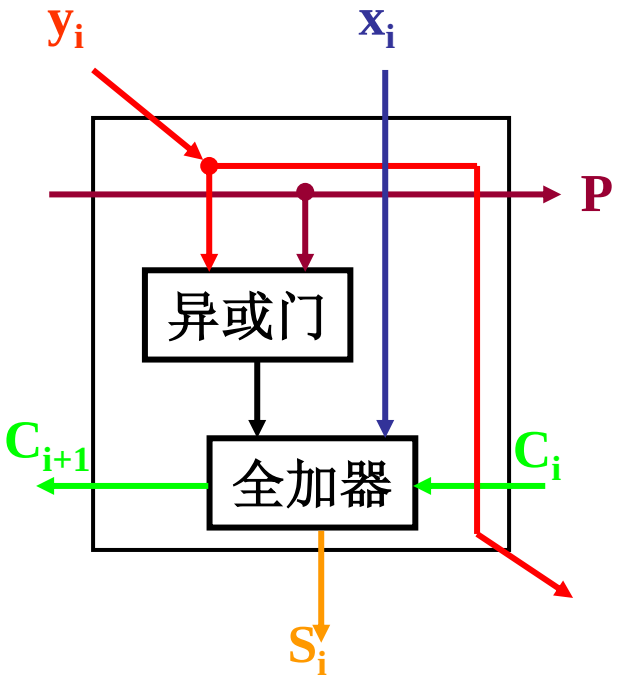

0
0
y1
x1
y2
x2
y3
x3
1
0
CAS
CAS
CAS
CAS
q1
q2
q3
x4
x5
x6
r3
r4
r5
r6
yi
xi
P
异或门
全加器
Ci+1
Ci
Si
可控加减法器CAS

4.3.5 阵列除法器

（1）可控加/减法单元
(CAS —Controllable Adder Subtracter)

当P = 0 时，做加法；

当P = 1 时，做减法，变+Y*为+[-Y*]补 。



4.3.5 阵列除法器

(2) 阵列除法算法流程

设 被除数 $X = 0.x_1\ x_2\ x_3\ x_4\ x_5\ x_6$

除 数 $Y = 0.y_1\ y_2\ y_3$

则 商 $Q = 0.q_1\ q_2\ q_3$

余数 $R = 0.00r_3\ r_4\ r_5\ r_6$

4.3.6 阵列除法过程

第一步：试减，即P=1，实现 $X+[-Y]_{补}$ 。

因为X*<Y*, 所以一定不够减，则最高位进位$C_{i+1}=0$，

可*利用此进位输出产生商和下一步的P*。

第二步：P=0，做 $X+Y$ 。

当最高位进位$C_{i+1}=1$时，表示够减, 则 $q_1=1$, $P=1$;

当最高位进位$C_{i+1}=0$时，表示不够减, 则 $q_1=0$, $P=0$。

第三步和第四步：P=0时,做 $X+Y$;P=1时,做 $X+[-Y]_{补}$ 。

上商和P值产生的规则与第二步相同。

4.3.7 阵列除法器设计

- 根据原理说明采用VHDL进行功能设计。
- 参考阵列乘法器设计方法进行设计。