# 第五章　运算方法与运算器（一）

秦磊华 计算机学院

# 本章主要内容

## 基于补码数据表示研究运算方法和设计运算器(简)

5.1 定点加法运算

5.2 加法运算器设计

5.3 定点减法运算

5.4 定点减法运算器设计

## 1.补码加法运算方法

$$[X+Y]_{补} = [X]_{补} + [Y]_{补}$$

1) 理解意义

2) 公式证明思路及可用的方法

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

MOD 2 $(2^{n+1})$

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

# 3.1 定点数加/减运算

$[X+Y]_{补} = [X]_{补} + [Y]_{补}$

1) $X > 0, Y > 0$ (可直接理解)

2) $X > 0, Y < 0$

3) $X < 0, Y > 0$ (2/3证明相同)

4) $X < 0, Y < 0$

# 3.1 定点数加/减运算



**1) X >0 ,Y >0**

因为： $x + y > 0$

所以： $x + y = [x + y]_{补} \pmod{2^{n+1}}$

华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

2) X >0 ,Y <0

所以: $[x]_{补}=x$  $[y]_{补}=2+y$

$[x]_{补}+[y]_{补}=x+2+y$

$=2+(x+y)$

当 $x+y<0$ 时

$2+(x+y)=[x+y]_{补}$ (mod 2)

当 $x+y>0$ 时

$2+(x+y)>2$ 模舍去

$[x]_{补}+[y]_{补}=2+(x+y)=x+y$ (mod 2)

$=[x+y]_{补}$ (mod 2)

3) X <0 ,Y> 0

同左

4) $X < 0, Y < 0$

$[x]_{补} = 2 + x \quad [y]_{补} = 2 + y$

$[x]_{补} + [y]_{补} = (2 + x) + (2 + y)$

$= 4 + x + y$

$= 2 + (2 + x + y) \mod 2$

$-2 \leqslant x + y < 0$

故 $0 \leqslant 2 + x + y < 2$

故 $2 + (2 + x + y) \mod 2 = (2 + x + y)$

$= [x + y]_{补} \mod 2$

# 3.1 定点数加/减运算

例1 已知 X = 0.10101, Y = 0.01000， 求 X+Y

解： $[X]_{补} = 0.10101$, $[Y]_{补} = 0.01000$

$$
\begin{array}{r}
0\ .1\ 0\ 1\ 0\ 1 \\
+\ 0\ .0\ 1\ 0\ 0\ 0 \\
\boxed{0\ 0\ \ 0\ 0\ 0\ 0} \\
\hline
0\ .1\ 1\ 1\ 0\ 1
\end{array}
$$

X+Y = 0.11101

华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

例2 已知 X = -10111, Y =-1000， 求X+Y

解： $[X]_{补} = 101001$, $[Y]_{补} = 11000$

```
     1 0 1 0 0 1
  +  1 1 1 0 0 0
   0 1 1 0 0 0
  ----------------
   1 1 0 0 0 0 1
```

X+Y = -11111

## 2.补码减法运算方法

$[X-Y]_{补} = [X]_{补} - [Y]_{补} = [X]_{补} + [-Y]_{补}$

1) 理解意义

2) 公式证明思路及可用的方法

$[X-Y]_{补} = [X+(-Y)]_{补}$

$= [X]_{补} + [-Y]_{补}$ !

$[Y-Y]_{补} = [Y]_{补} + [-Y]_{补}$

$\Rightarrow [Y]_{补} + [-Y]_{补} = 0$

$\Rightarrow [Y]_{补} = -[-Y]_{补}$

## 2. 补码减法运算方法

$$[X-Y]_{补} = [X]_{补} - [Y]_{补} = [X]_{补} + [-Y]_{补}$$

例3 已知 $[X]_{补} = 101001$， 求 $[-X]_{补}$

由 $[X]_{补} = 101001$

$X = -10111$

$-X = 10111$

$[-X]_{补} = 010111$

$[X]_{补} = 101001$

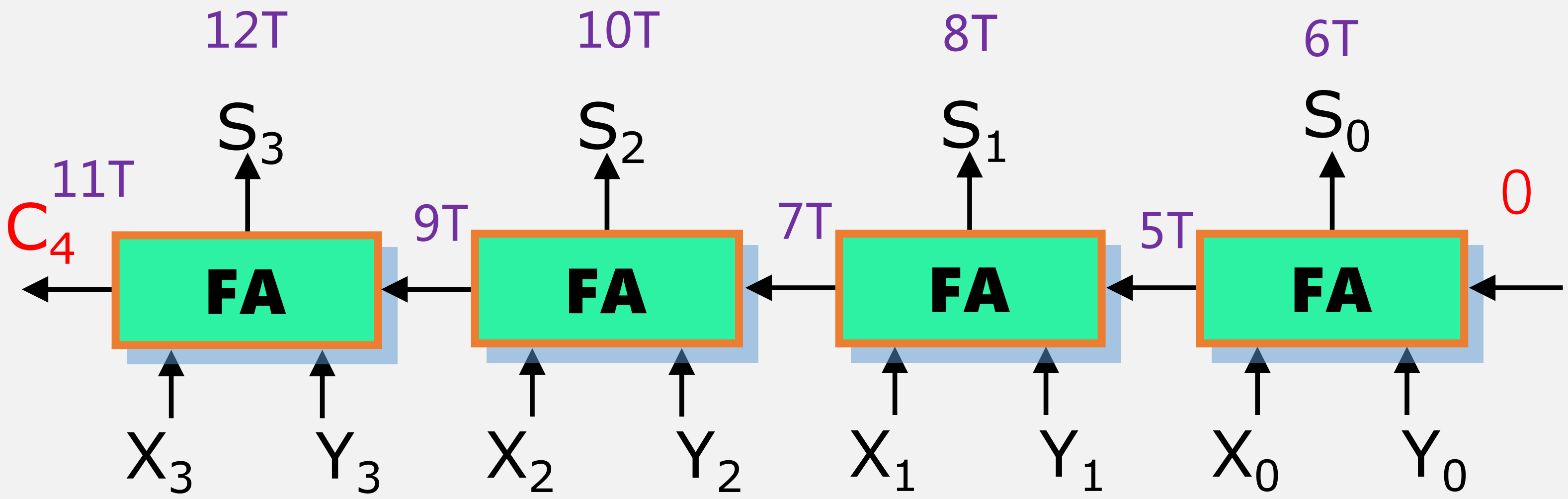
例4 已知 X = -10111, Y =-1000， 求X - Y

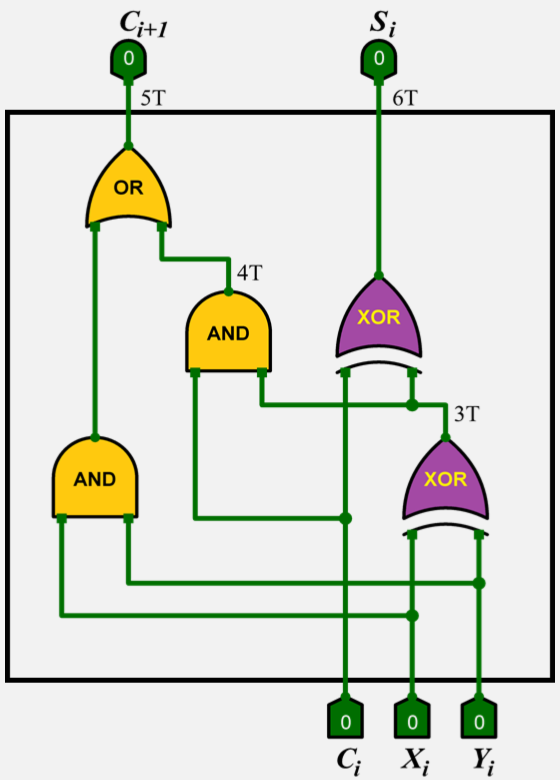
解： $[X]_{补} = 101001$， $[Y]_{补} = 11000$  $[-Y]_{补} = 01000$

```
     1 0 1 0 0 1
  +  0 0 1 0 0 0
   0 0 1 0 0 0
  ---------------
     1 1 0 0 0 1
```

X-Y = -1111

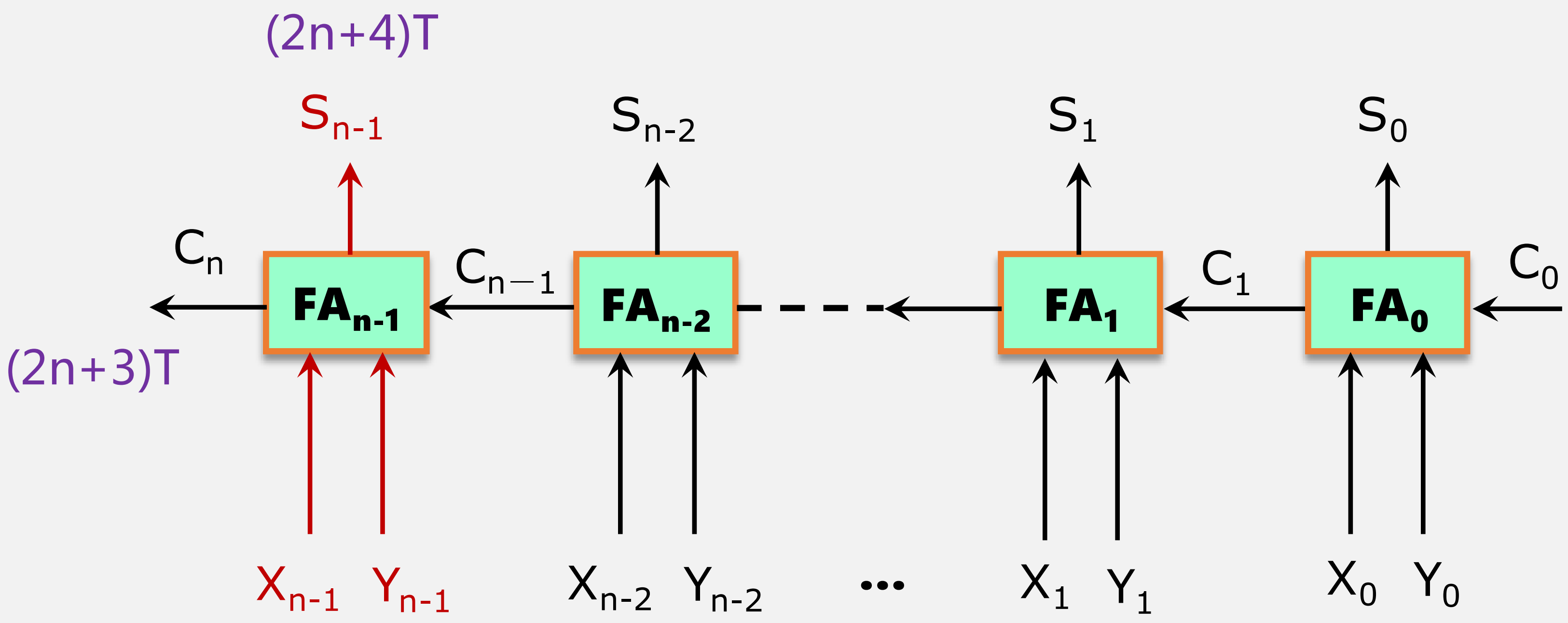
3. 基本补码加法运算器设计



华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

## 3. 基本补码加法运算器设计

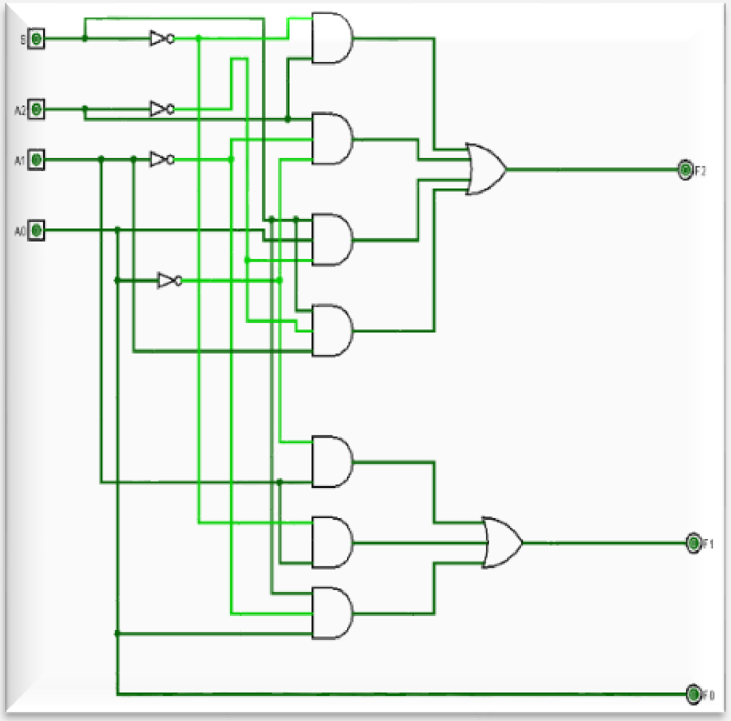


# 3.1 定点数加/减运算

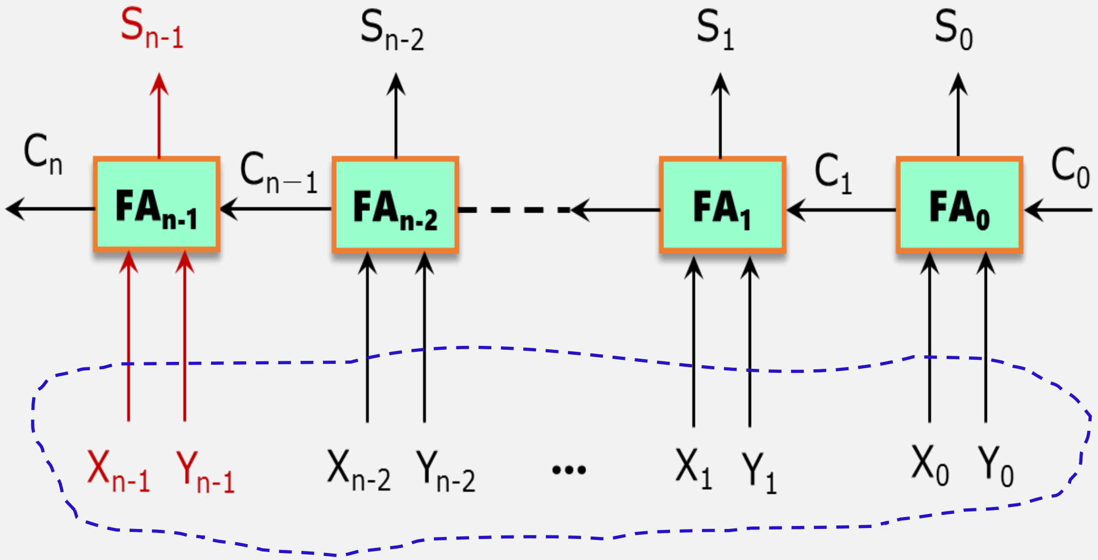
## 3. 基本补码减法运算器设计

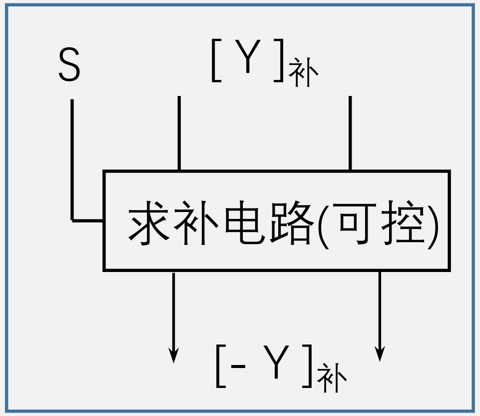
$$[X+Y]_{补} = [X]_{补} + [Y]_{补}$$

$$[X-Y]_{补} = [X]_{补} - [Y]_{补} = [X]_{补} + [-Y]_{补}$$



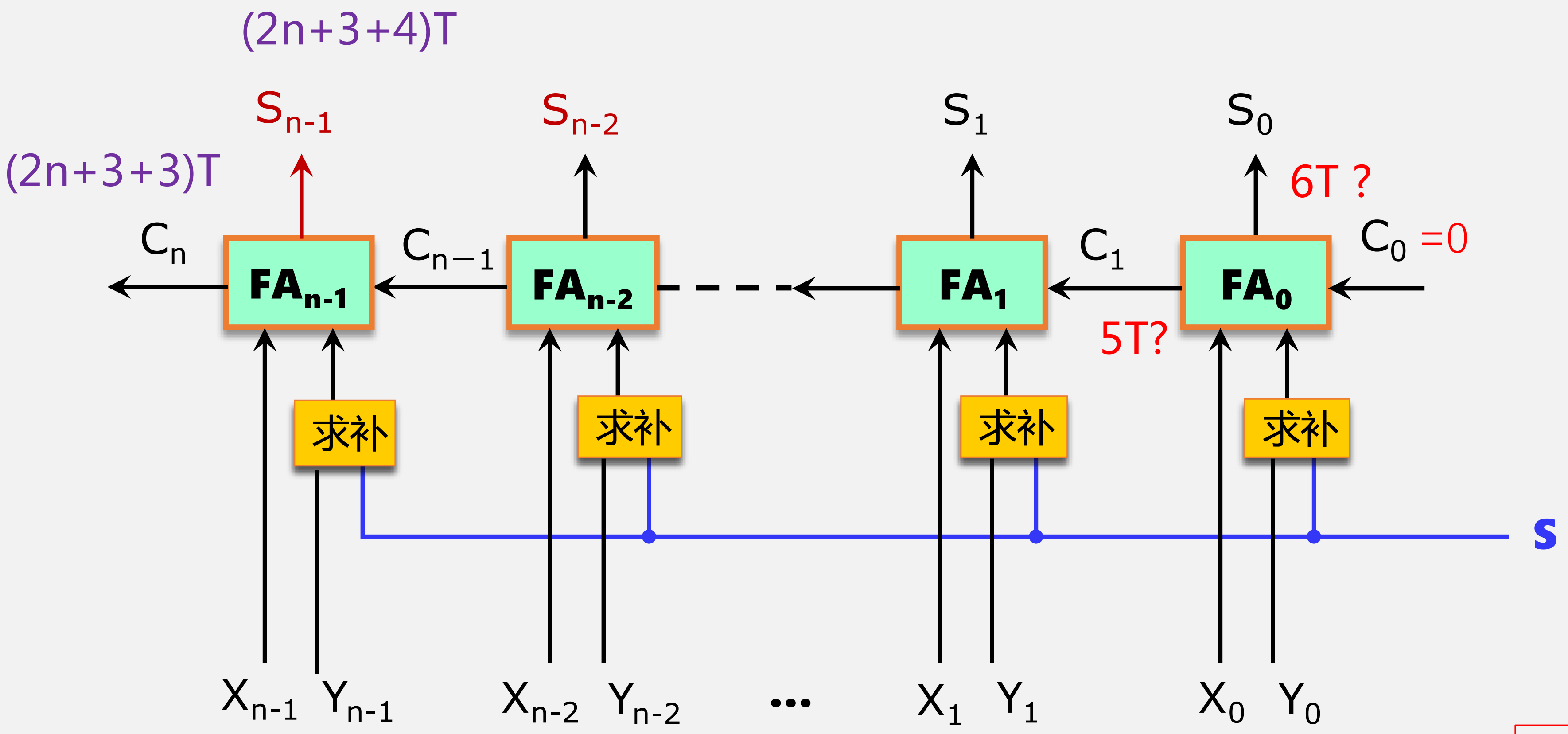
3T



华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

3. 基本补码减法运算器设计




S = 0 : 加法

S = 1 : 减法

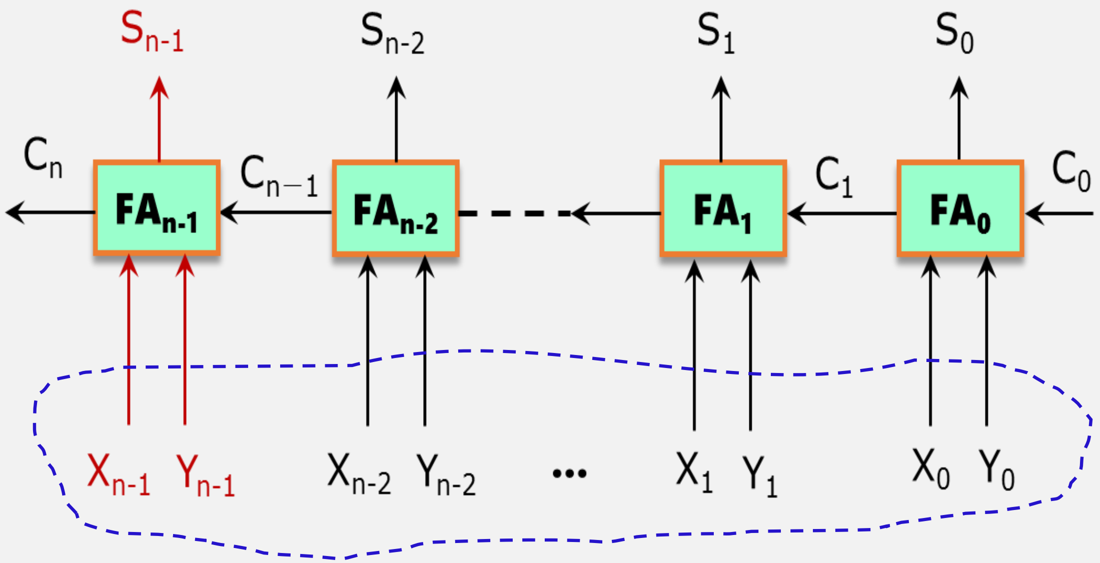
减法不满足交换律

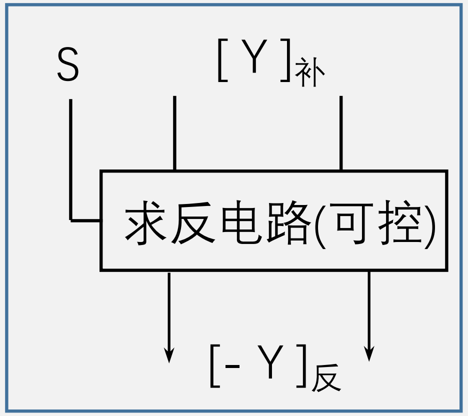
# 3.1 定点数加/减运算

## 3. 基本补码减法运算器设计

$$[X+Y]_{补} = [X]_{补} + [Y]_{补}$$

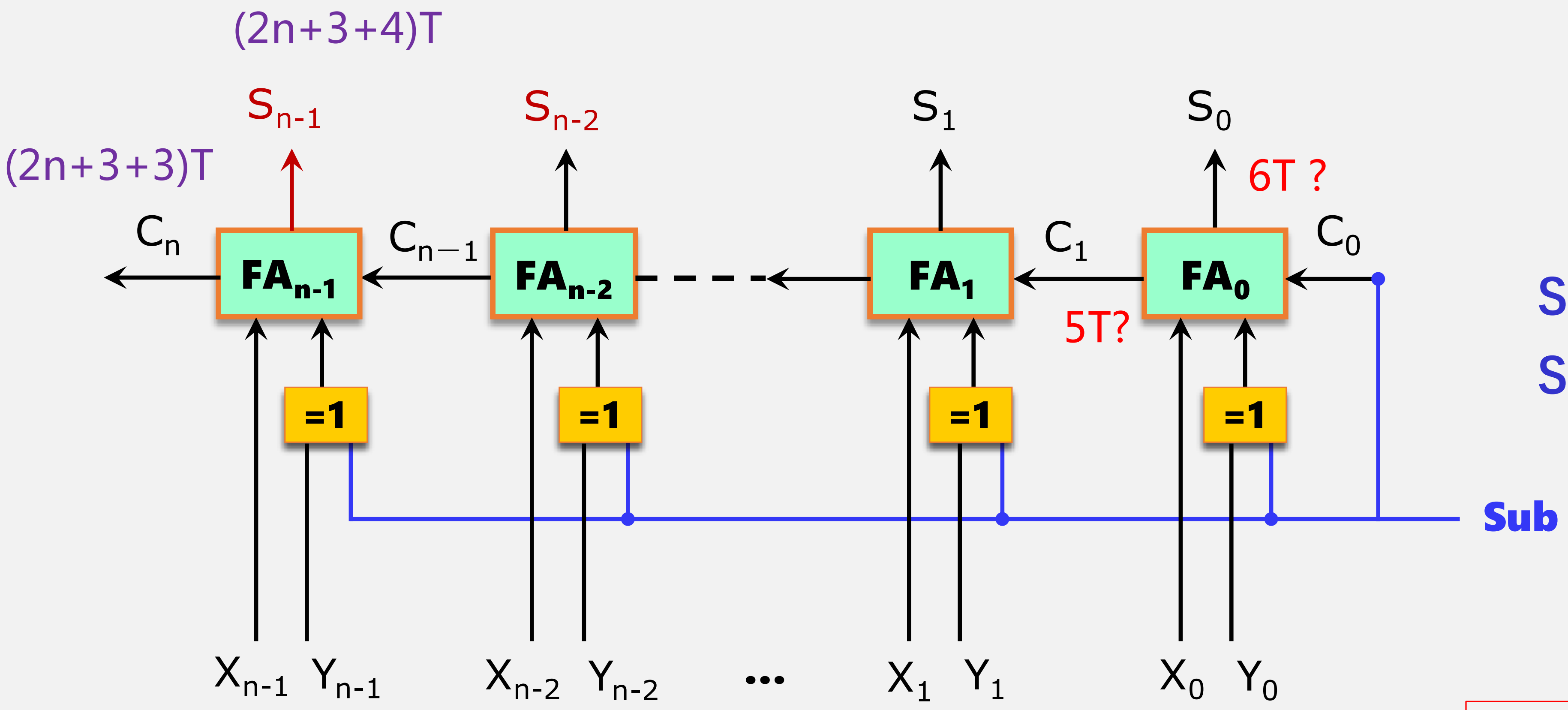
$$[X-Y]_{补} = [X]_{补} - [Y]_{补} = [X]_{补} + [-Y]_{补}$$




$$\begin{array}{r} 1101101 \\ \oplus\ 1111111 \\ \hline 0010010 \end{array} \qquad \begin{array}{r} 1101101 \\ \oplus\ 0000000 \\ \hline 1101101 \end{array}$$

## 3. 基本补码减法运算器设计




Sub = 0 : 加法

Sub = 1 : 减法

减法不满足交换律

## 4. 补码加减法运算溢出

例5 已知 X = -11011 , Y = - 1 0 0 0 求 X + Y

解： $[X]_{补} = 100101$, $[Y]_{补} = 11000$

$$
\begin{array}{r}
1\ 0\ 0\ 1\ 0\ 1 \\
+\ 1\ 1\ 1\ 0\ 0\ 0 \\
\hline
\boxed{1}\ 0\ 1\ 1\ 1\ 0\ 1
\end{array}
$$

X+Y = +11101

华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

## 4.补码加减法运算溢出

例6 已知 X = +10101 , Y = + 10000 求 X + Y

解： $[X]_{补} = 010101$, $[Y]_{补} = 01000$

```
   0 1 0 1 0 1
 + 0 1 1 0 0 0
 -------------
   1 0 1 1 0 1
```

X+Y = -10011

## 4. 补码加减法运算溢出

$$
\begin{array}{r}
0\;1\;0\;1\;0\;1 \\
+\;0\;1\;1\;0\;0\;0 \\
\hline
1\;0\;1\;1\;0\;1
\end{array}
\qquad
\begin{array}{r}
1\;0\;0\;1\;0\;1 \\
+\;1\;1\;1\;0\;0\;0 \\
\hline
\boxed{1}\;0\;1\;1\;1\;0\;1
\end{array}
$$

**溢出：** 运算结果超出对应数据类型和机器字长的数据表示范围

## 5.补码加减法运算溢出检测

◆ 根据概念，溢出只发生在同号数相加时，包括$[X]_{补}$与$[Y]_{补}$, $[X]_{补}$与$[-Y]_{补}$同号

1) 方法1：对操作数和运算结果的符号位进行检测

当结果的符号与操作数的符号不同时表明发生了溢出

```
  0 1 0 1 0 1
+ 0 1 1 0 0 0
-------------
  1 0 1 1 0 1
```

```
  1 0 0 1 0 1
+ 1 1 1 0 0 0
-------------
1 0 1 1 1 0 1
```

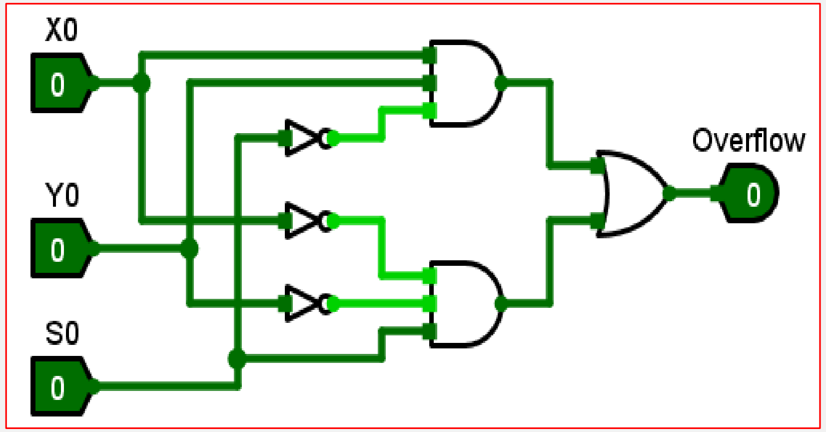
# 3.1 定点数加/减运算

## 5.补码加减法运算溢出检测

设$X_n$，$Y_n$ 为参加运算数的符号位， $S_n$ 为结果符号位

$$\text{Overflow} = X_nY_n\overline{S_n} + \overline{X_n}\,\overline{Y_n}S_n$$

当Overflow=1时，结果溢出，根据逻辑表达式，得到相应电路



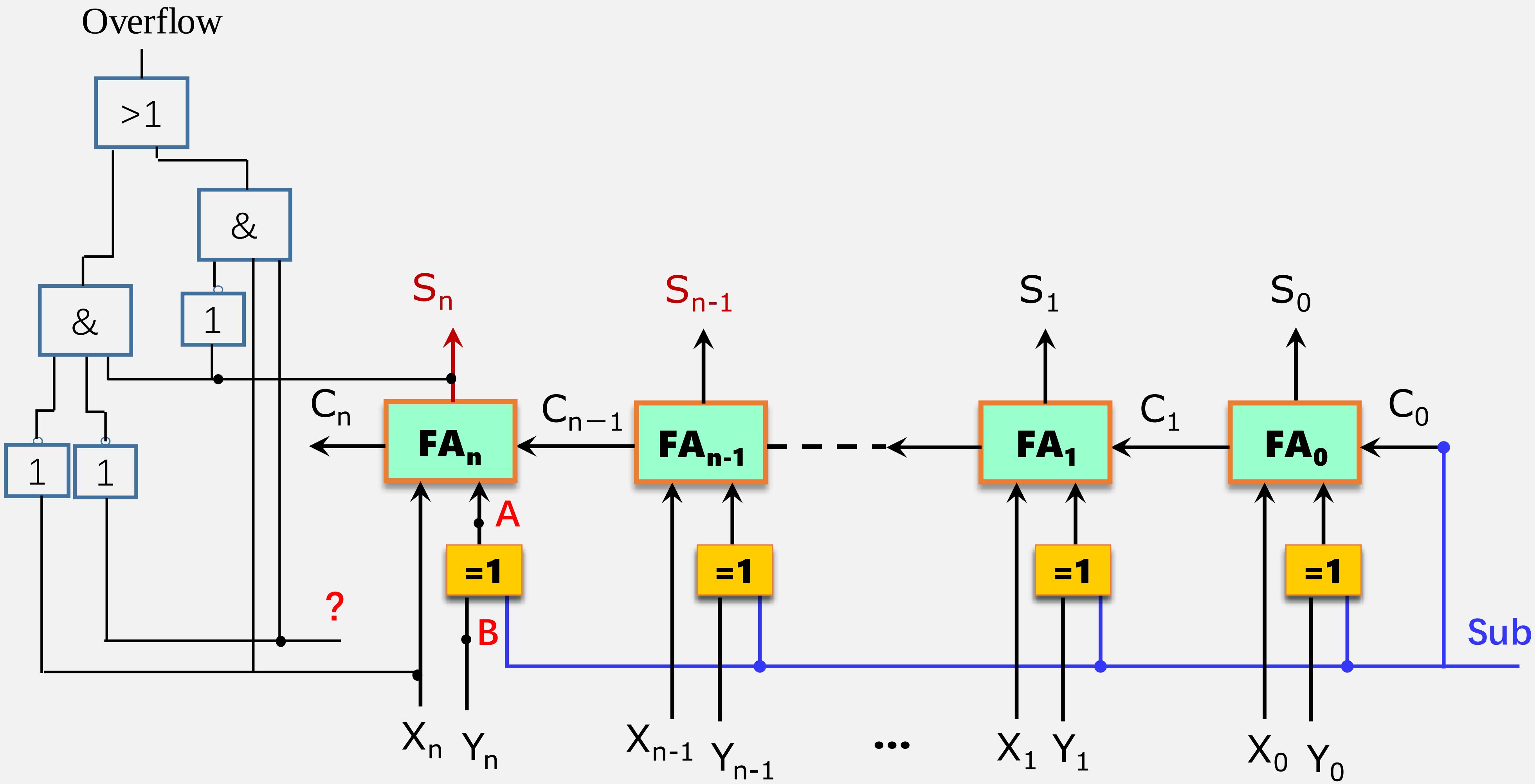

代价分析:

6个逻辑门，3级时延

华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

## 5·补码加减法运算溢出检测

$$Overflow = X_nY_n\overline{S_n} + \overline{X_n}\,\overline{Y_n}S_n$$



# 3.1 定点数加/减运算

## 5.补码加减法运算溢出检测

2) 方法2：通过运算中最高位数据的进位与符号位的进位是否同步来检测

假**符号位进位**为Cf，**最高数据位进位**为Cn

$C_f=0\quad C_n=0$

```
    0 1 0 1 0 1
+   0 0 1 0 0 0
  0 0
  -------------
    0 1 1 1 0 1
```

$C_f=1\quad C_n=1$

```
    1 1 0 1 0 1
+   1 1 1 0 0 0
  1 1
  -------------
  1 1 0 1 1 0 1
```

## 5.补码加减法运算溢出检测

2) 方法2：通过运算中最高位数据的进位与符号位的进位是否同步来检测

设**符号位进位**为Cf，**最高数据位进位**为Cn

$C_f=0 \quad C_n=1$

```
    0 1 0 1 0 1
+   0 1 1 0 0 0
  0 1
  -------------
    1 0 1 1 0 1
```

$C_f=1 \quad C_n=0$

```
     1 0 0 1 0 1
+    1 1 1 0 0 0
   1 0
  --------------
   1 0 1 1 1 0 1
```

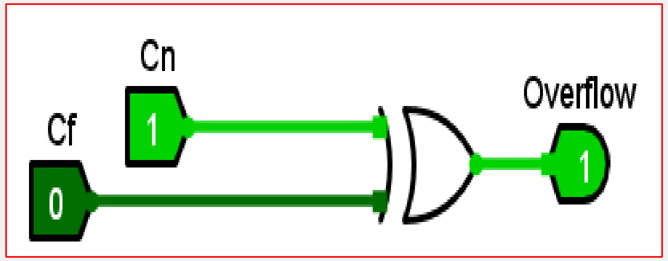
$$Overflow = C_f \oplus C_n$$

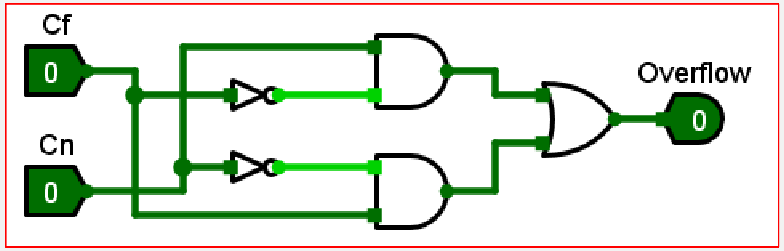
## 5·补码加减法运算溢出检测

2) 方法2：通过运算中最高位数据的进位与符号位的进位是否同步来检测

设**符号位进位**为Cf，**最高数据位进位**为Cn

$$Overflow = C_f \oplus C_n$$




代价分析:

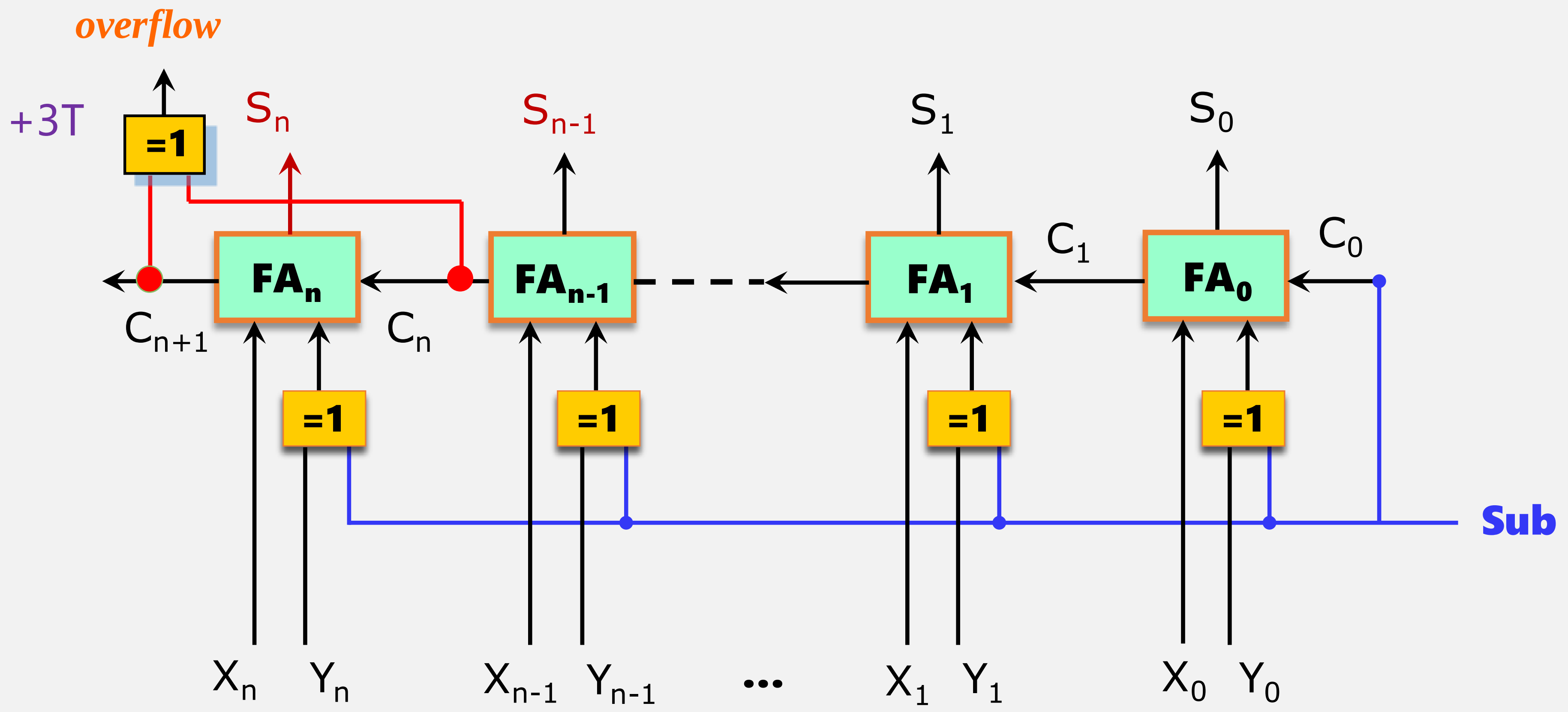
1个异或逻辑门，3T时延

代价分析:

5个逻辑门，3T时延

## 5.补码加减法运算溢出检测

方法2： $Overflow = C_f \oplus C_n$



## 5.补码加减法运算溢出检测

3) 方法3: 用变型补码的双符号位，设双符号位为 $f_1$ $f_2$, 其中$f_1$为最高符号位

```
    0 0 1 0 1 0 1
+   0 0 0 1 0 0 0
  ---------------
    0 0 1 1 1 0 1
```

$f_1=0$ $f_2=0$

```
      1 1 1 0 1 0 1
+     1 1 1 1 0 0 0
  -----------------
  [1] 1 1 0 1 1 0 1
```

$f_1=1$ $f_2=1$

## 5.补码加减法运算溢出检测

3) 方法3： 用变型补码的双符号位，设双符号位为 $f_1$ $f_2$, 其中$f_1$为最高符号位

```
  0 0 1 0 1 0 1
+ 0 0 1 1 0 0 0
---------------
  0 1 0 1 1 0 1
```

$f_1$=0 $f_2$ =1 正溢出(上溢)

```
  1 1 0 0 1 0 1
+ 1 1 1 1 0 0 0
---------------
1 1 0 1 1 1 0 1
```

$f_1$=1 $f_2$ = 0 负溢出(下溢)

$$Overflow = f_1 \oplus f_2$$

第一符号位是结果的正确符号位

华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

## 5·补码加减法运算溢出检测

3) 方法3： 用变型补码的双符号位，设双符号位为 $f_1$ $f_2$, 其中$f_1$为最高符号位

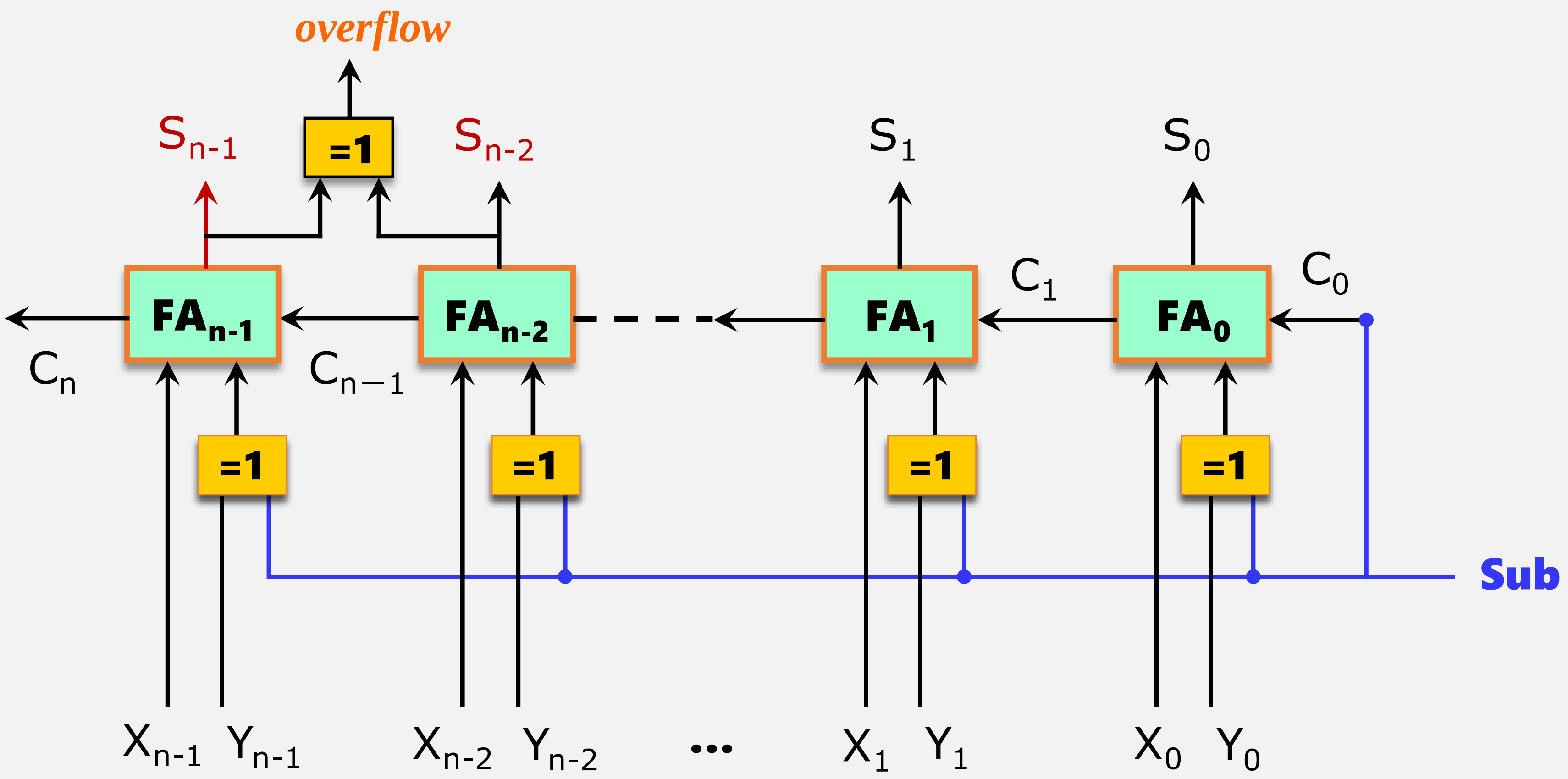
$$Overflow = f_1 \oplus f_2$$

溢出检测电路和成本分析同方法2。

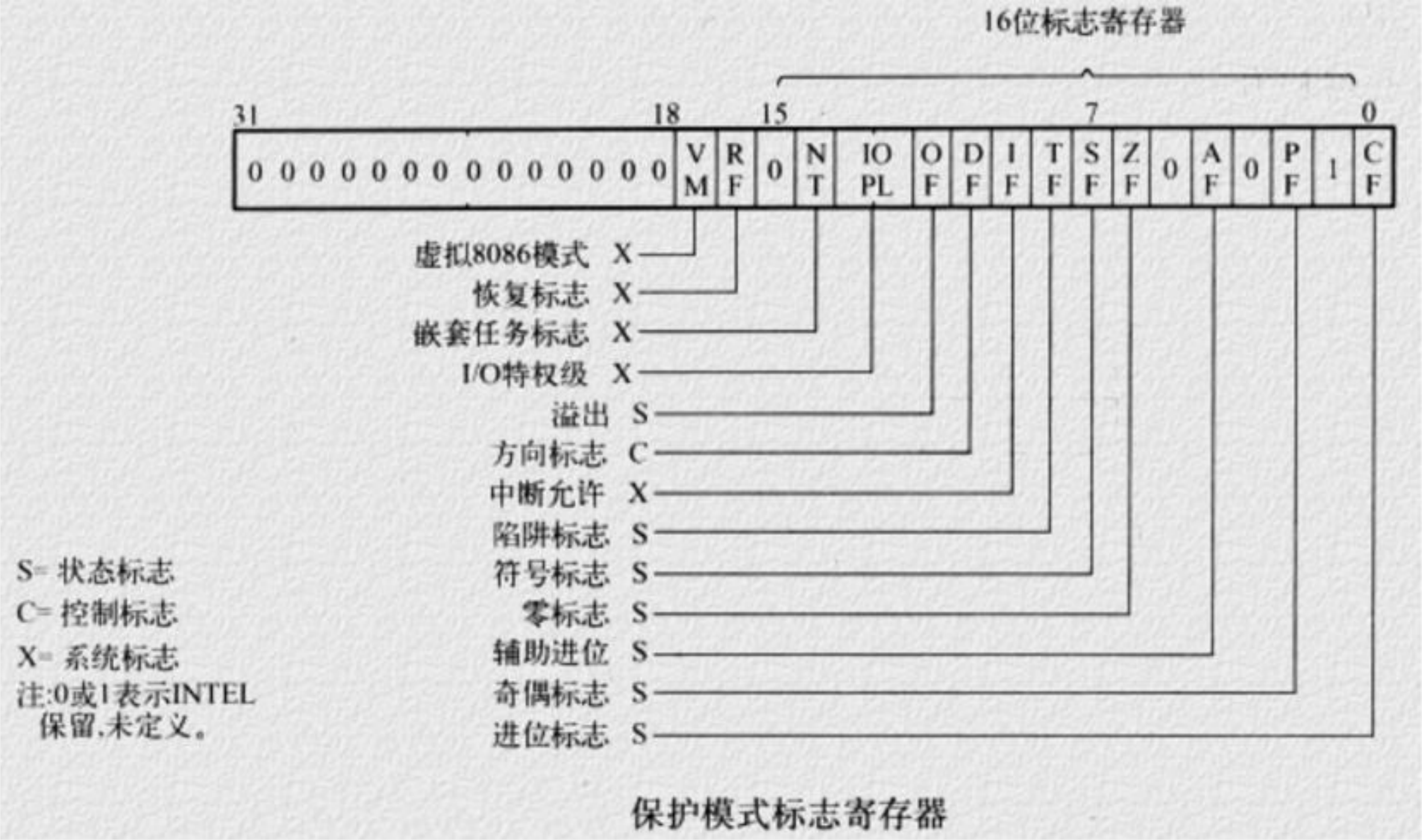
与方法2不同的是，需要增加了一位符号位，会影响数据表示范围吗？

## 5.补码加减法运算溢出检测

方法3：$Overflow = f_1 \oplus f_2$



## 5.补码加减法运算溢出检测




MOVE AX, 0XFFFFF

………

ADD AX, BX

JO LOOP

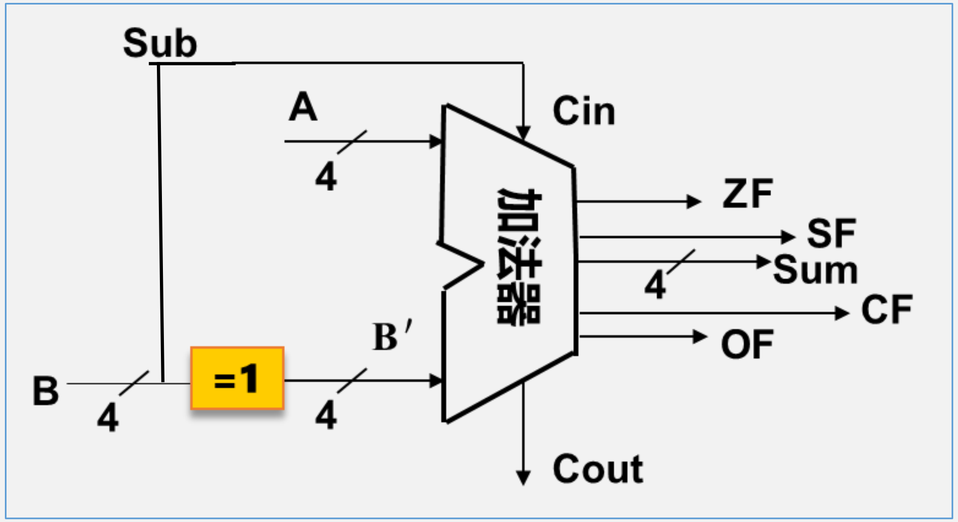
……

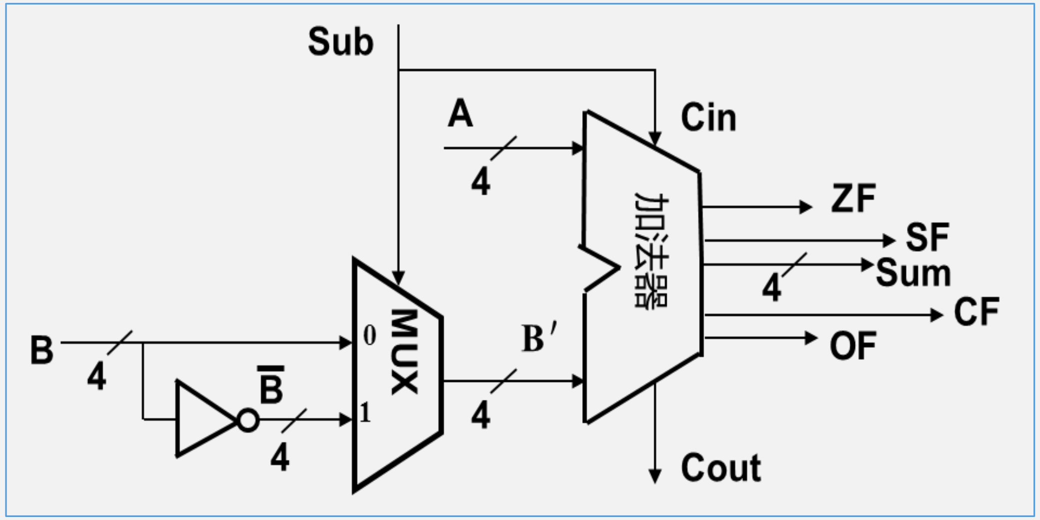
LOOP:

………

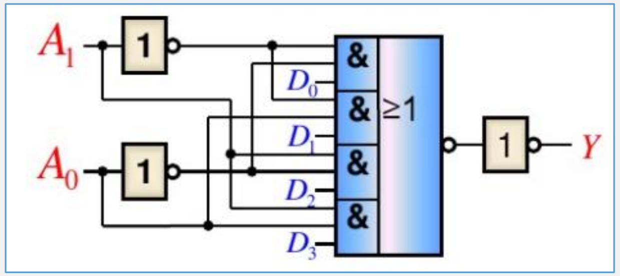
华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

## 5. 补码加减法运算溢出检测




比较两种方案哪种更好？为什么？



## 5.补码加减法运算溢出检测

问题：若CPU中未设计上述溢出检测电路或未设置标志寄存器，如何检测溢出？

```
int tadd_ok(int x,int y)
 {
    int sum= x+y;
    int neg_over= x<0&&y<0&&sum >=0;
    int pos_over= x>=0&&y>=0&&sum<0;
    return !neg_over&&!pos_over;
}
```

- 体会软硬溢出检测的逻辑等效性
- 体会软硬溢出检测的差异性
- 对你未来的程序设计有何启示？
- 这就是软硬协同的系统观

## 6. 无符号数加减法运算溢出检测

| | | |
|---|---|---|
| | 0 0 0 0 0 1 0 1 | 3 |
| + | 0 0 0 0 1 0 0 0 | 8 |
| | 0 0 0 0 1 1 0 1 | 11 |

加法未溢出 Cout=0

| | | |
|---|---|---|
| | 1 1 1 1 1 1 1 1 | 255 |
| + | 1 1 1 1 1 1 1 0 | 254 |
| | $\boxed{1}$ 1 1 1 1 1 1 0 1 | 253 |

加法溢出 Cout=1 和数小于加数

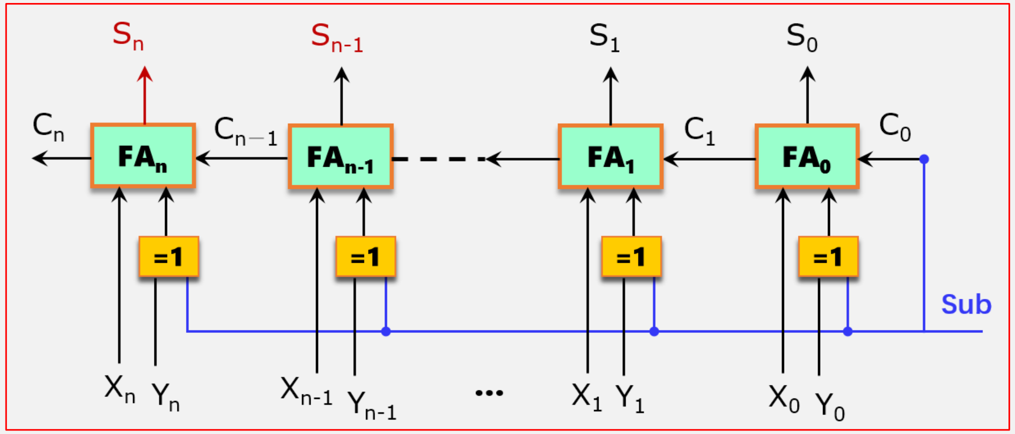
## 6. 无符号数加减法运算溢出检测

```
    0 0 0 0 0 1 0 1      5
 +  1 1 1 1 0 1 1 0    -10
 -----------------------
  0 1 1 1 1 1 0 1 1    127
```

减法溢出 Cout=0 差大于被减数

```
    0 0 0 0 1 0 1 0     10
 +  1 1 1 1 1 0 1 1     -5
 -----------------------
  1 0 0 0 0 0 1 0 1      5
```

减法未溢出 Cout=1




无符号溢出检测 = $SUB \oplus C_{out}$

## 6. 无符号数加减法运算溢出检测

$$
\begin{array}{r}
0\,1\,0\,0\,0\,0\,0\,1 \\
+\;0\,1\,0\,0\,0\,0\,1\,0 \\
\hline
0\;1\,0\,0\,0\,0\,0\,1\,1
\end{array}
$$

- 无符号数加， 65+ 66 = 131， 正常
- 有符号数加， 65+ 66= -125， 溢出
- 无符号数减， 65- 190 = 131， 溢出
- 有符号数减， 65-(-66) =-125, 溢出

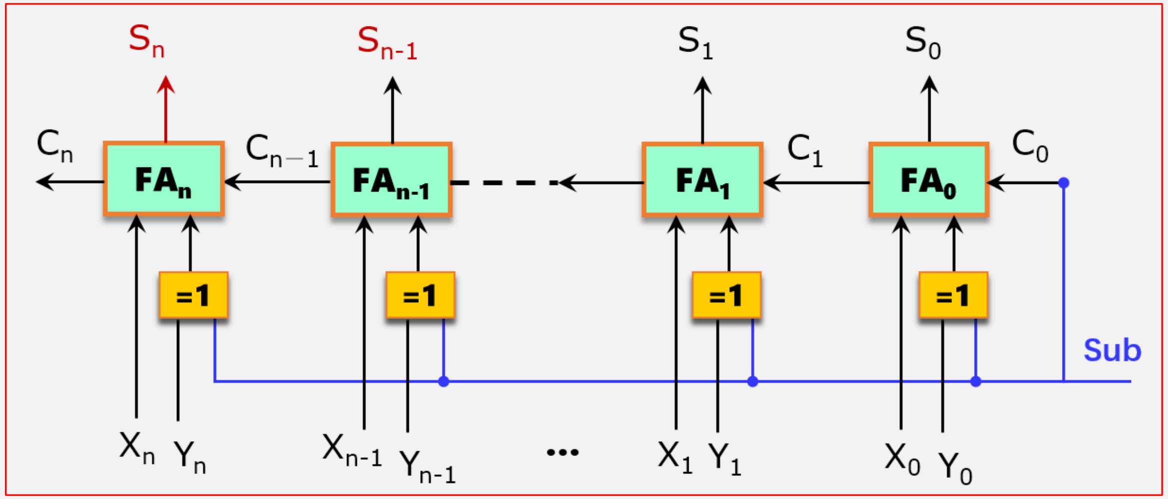
无符号溢出检测 = SUB ⊕ $C_{out}$

## 7. 快速加法器设计

$$S_i = X_i \oplus Y_i \oplus C_i$$

$$C_{i+1} = X_i Y_i + (X_i \oplus Y_i) C_i$$

$S_n$时延: (2n+4+3)T




$G_i = X_i Y_i$ 进位生成函数 Generate

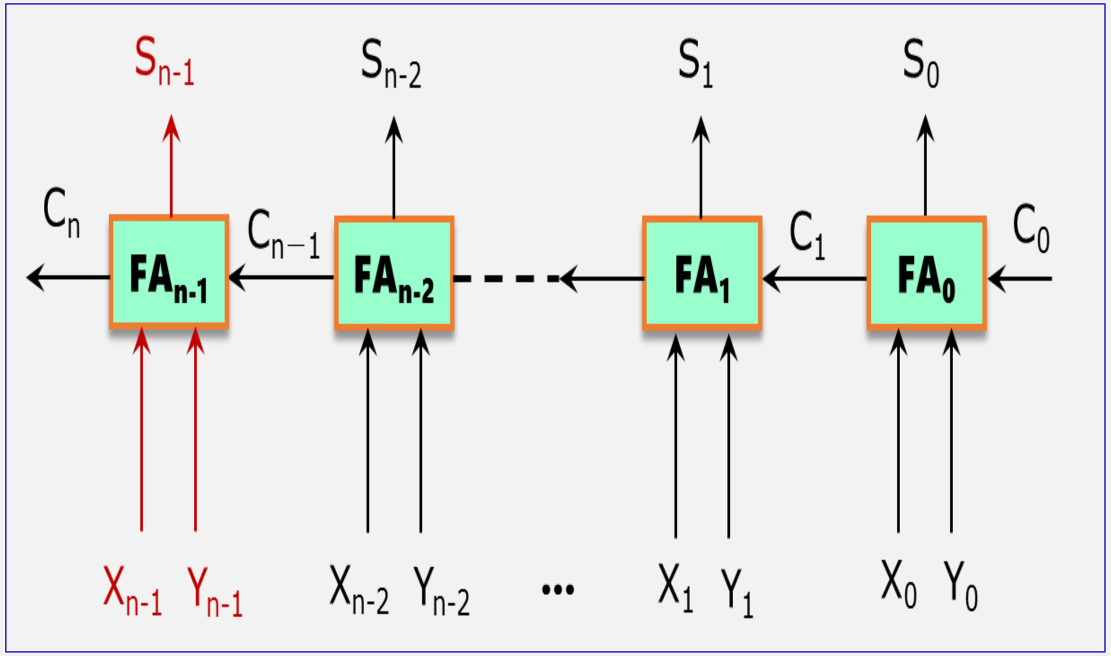
$P_i = X_i \oplus Y_i$ 进位传递函数 Propagate

$$C_{i+1} = G_i + P_i C_i$$

## 7. 快速加法器设计

$$G_i = X_i Y_i$$

$$P_i = X_i \oplus Y_i$$




$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1$$

$$C_3 = G_2 + P_2 C_2$$

……

$$C_n = G_{n-1} + P_{n-1} C_{n-1}$$

$$C_{n+1} = G_n + P_n C_n$$

高位**依赖**于低位进位,形成了串行进位链

华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

7. 快速加法器设计

$$G_i = X_i Y_i$$

$$P_i = X_i \oplus Y_i$$

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1$$

$$= G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2$$

$$= G_2 + P_2(G_1 + P_1 G_0 + P_1 P_0 C_0)$$

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

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

# 3.1 定点数加/减运算

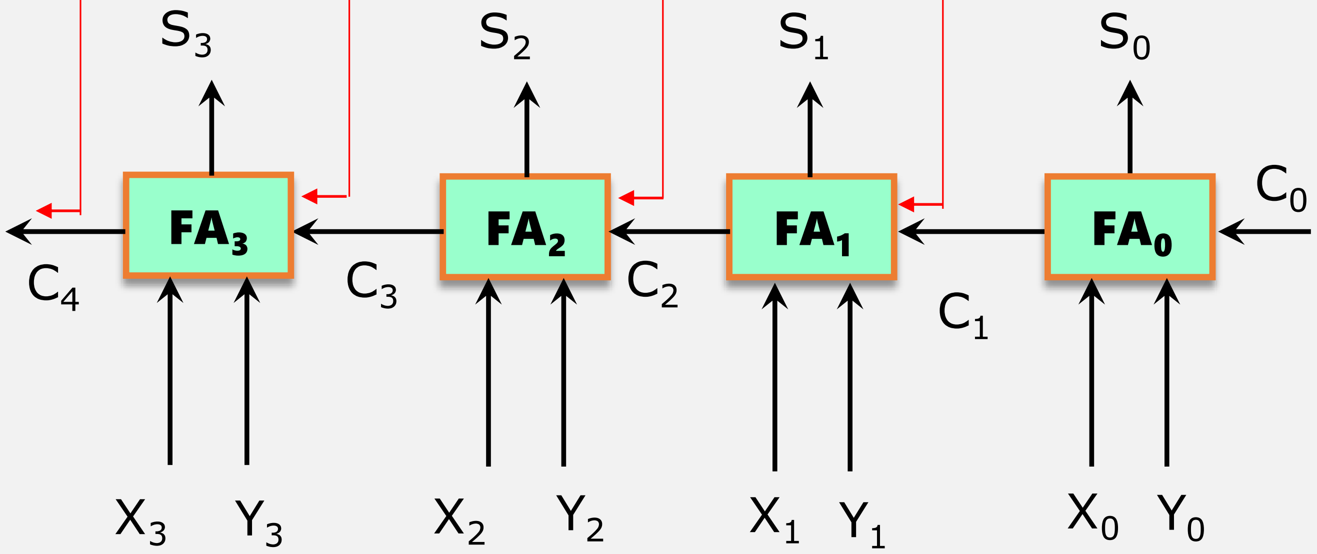
## 7. 快速加法器设计

$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

$C_2 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$

$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$

$C_1 = G_0 + P_0C_0$



华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

## 7. 快速加法器设计

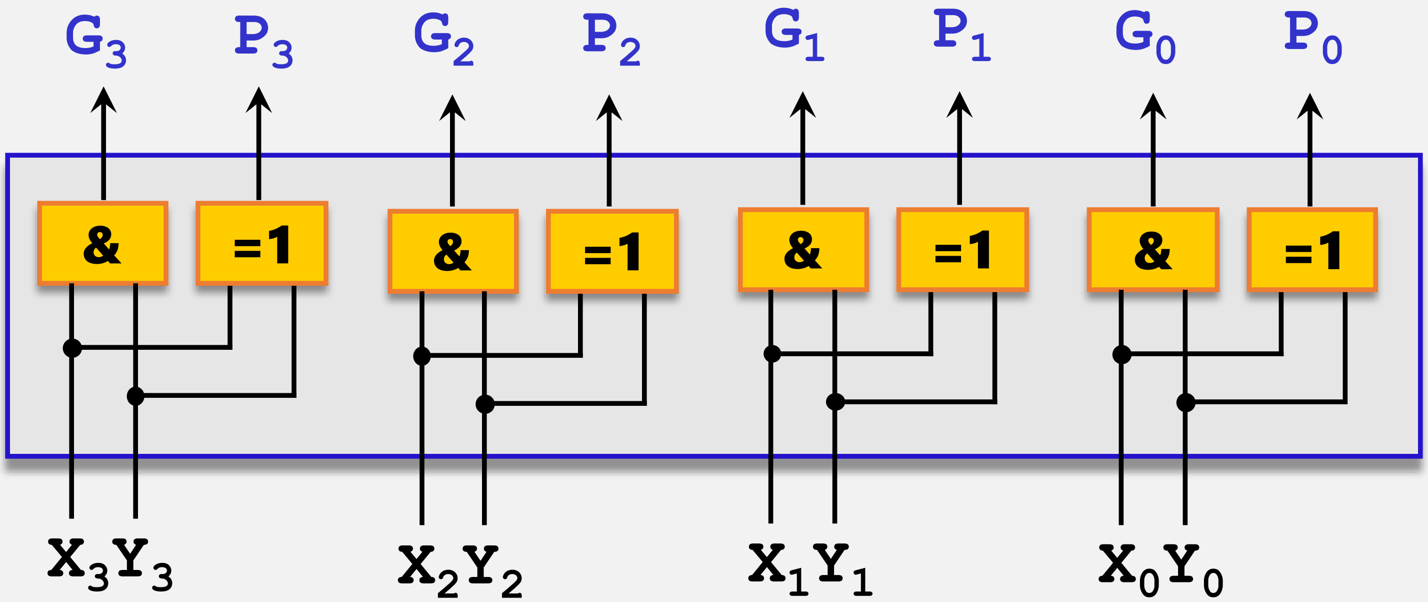
$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

$$C_{16} = G_{15} + P_{14}G_{13} + \cdots + \underbrace{P_{15}P_{14}\ldots P_1P_0C_0}_{\text{17个输入端的与门}}$$

扇入系数：逻辑门的输入端数量

扇出系数：逻辑门直接连接的输出端数量

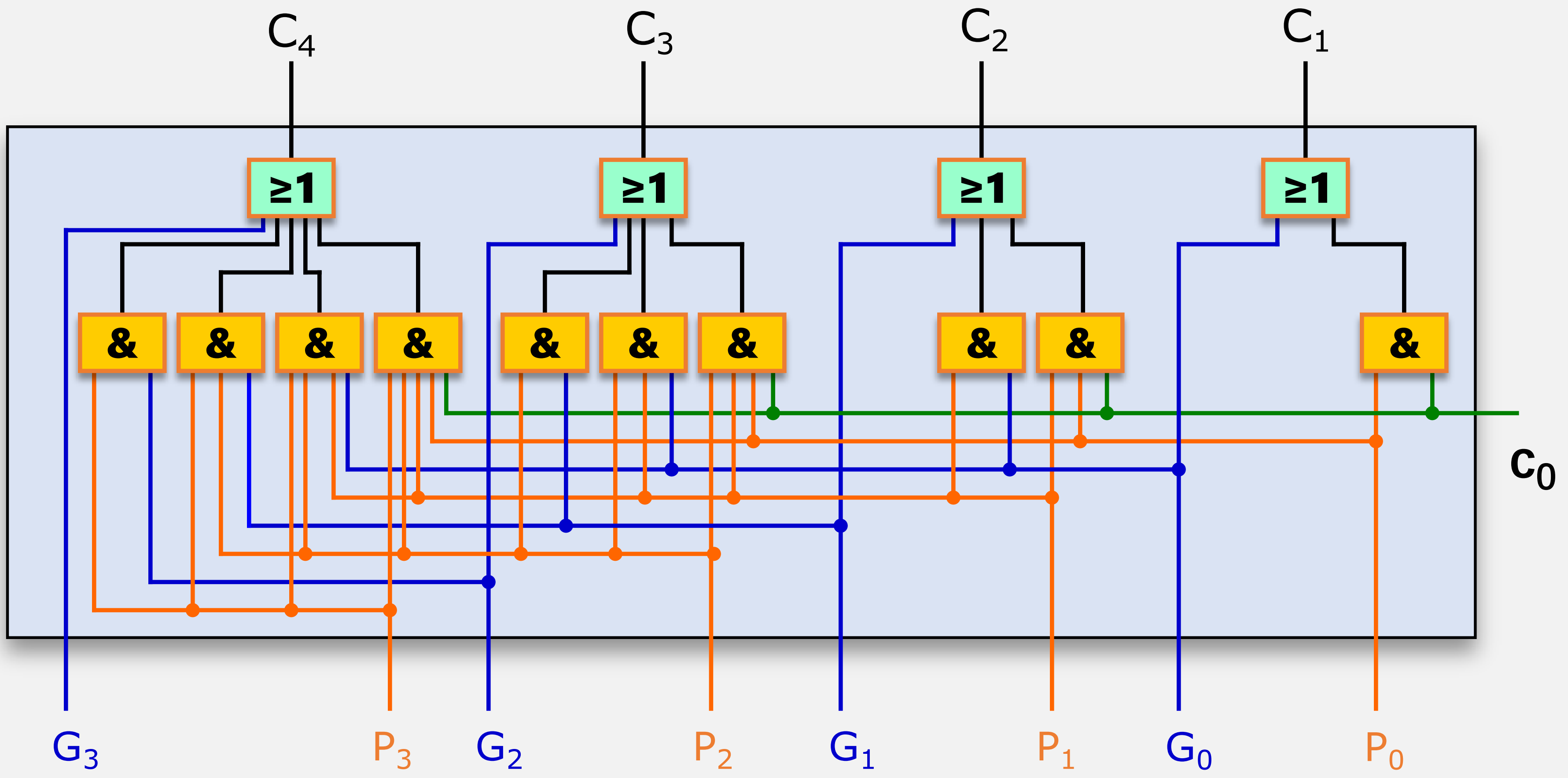
## 7. 快速加法器设计




$G_i = X_iY_i$

$P_i = X_i \oplus Y_i$

## 7. 快速加法器设计



## 7. 快速加法器设计

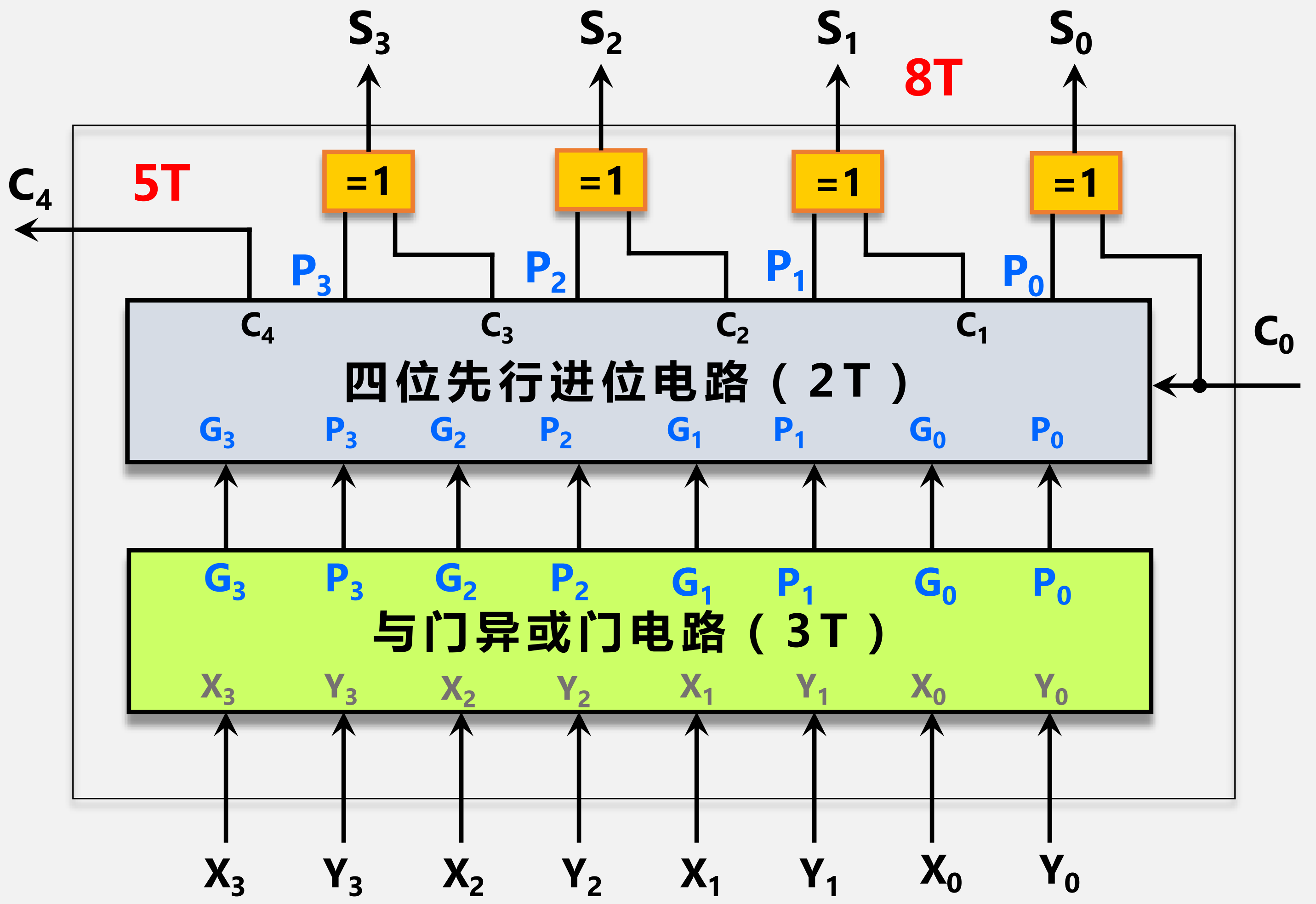
$$G_i = X_iY_i$$

$$P_i = X_i \oplus Y_i$$

$$S_3 = X_3 \oplus Y_3 \oplus C_3 = P_3 \oplus C_3$$
$$S_2 = X_2 \oplus Y_2 \oplus C_2 = P_2 \oplus C_2$$
$$S_1 = X_1 \oplus Y_1 \oplus C_1 = P_1 \oplus C_1$$
$$S_0 = X_0 \oplus Y_0 \oplus C_0 = P_0 \oplus C_0$$

华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

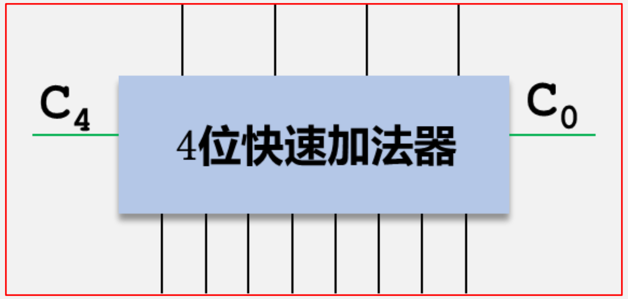
7. 快速加法器设计



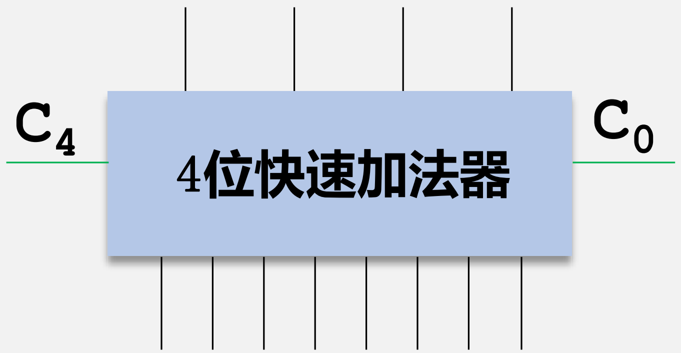

$$S_3 = X_3 \oplus Y_3 \oplus C_3 = P_3 \oplus C_3$$

$$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

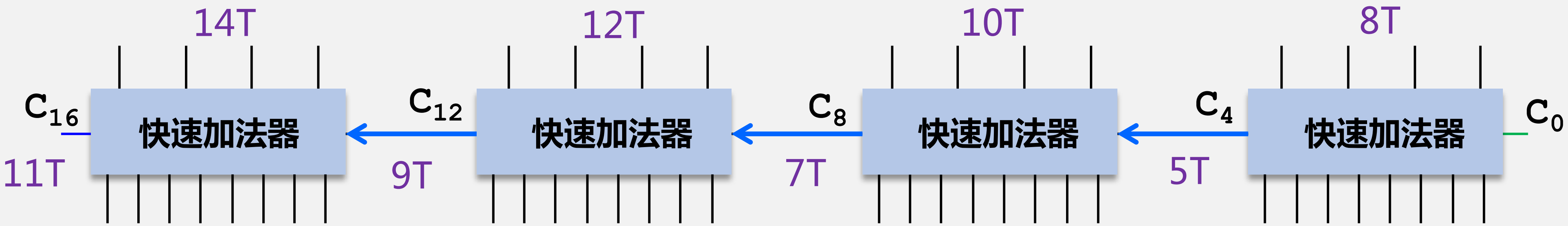
$$P_i = X_i \oplus Y_i \qquad G_i = X_iY_i$$



## 7. 快速加法器设计




- 组内先行进位
- 组间？ 串行进位
- 可否组间并行？



华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

## 7. 快速加法器设计

$$C_4 = G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0 + P_3P_2P_1P_0\ C_0$$

$G^* = G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0$ 成组进位生成函数

$P^* = P_3P_2P_1P_0$ 成组进位传递函数

$C_4 = G^* + P^*C_0$

$C_1 = G_0 + P_0C_0$

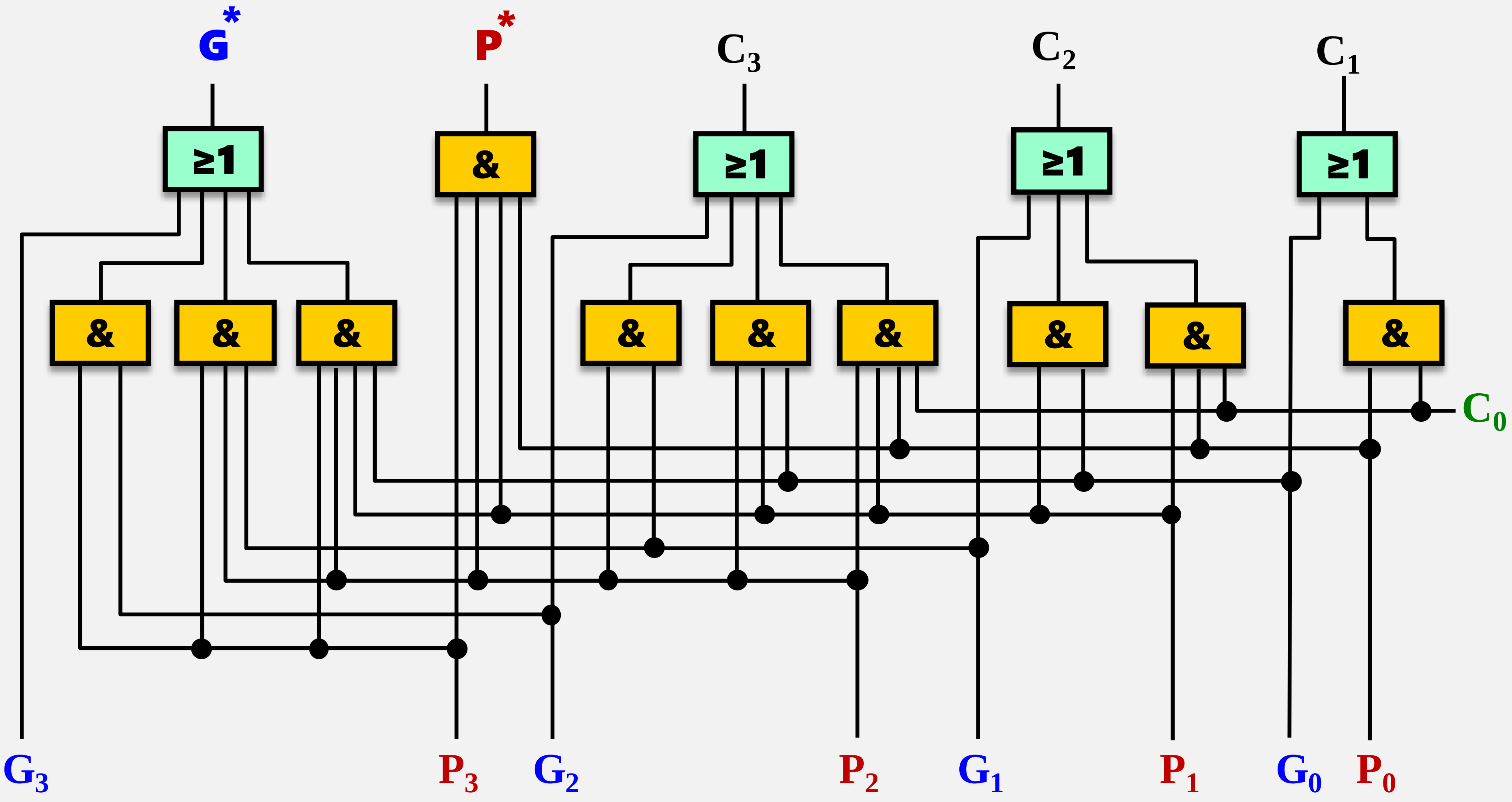
两式逻辑表达式完全一样！

？ 也应该完全一样！

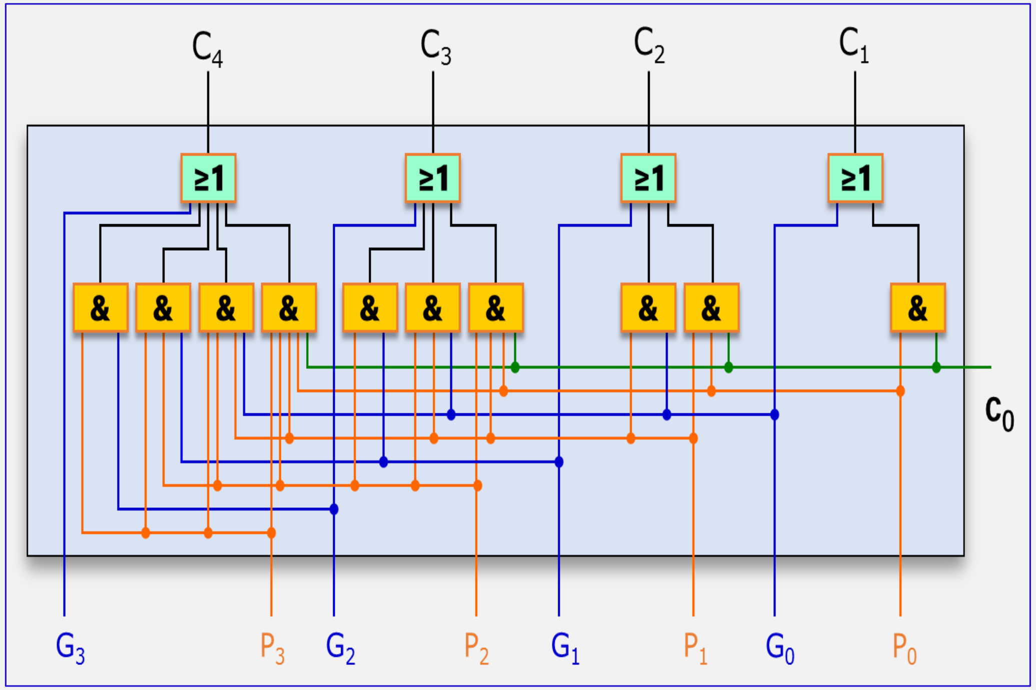
同理 可得： $C_8$ VS $C_2$ $C_{12}$VS $C_3$, $C_{16}$ VS $C_4$ 的表达式也一样

7. 快速加法器设计

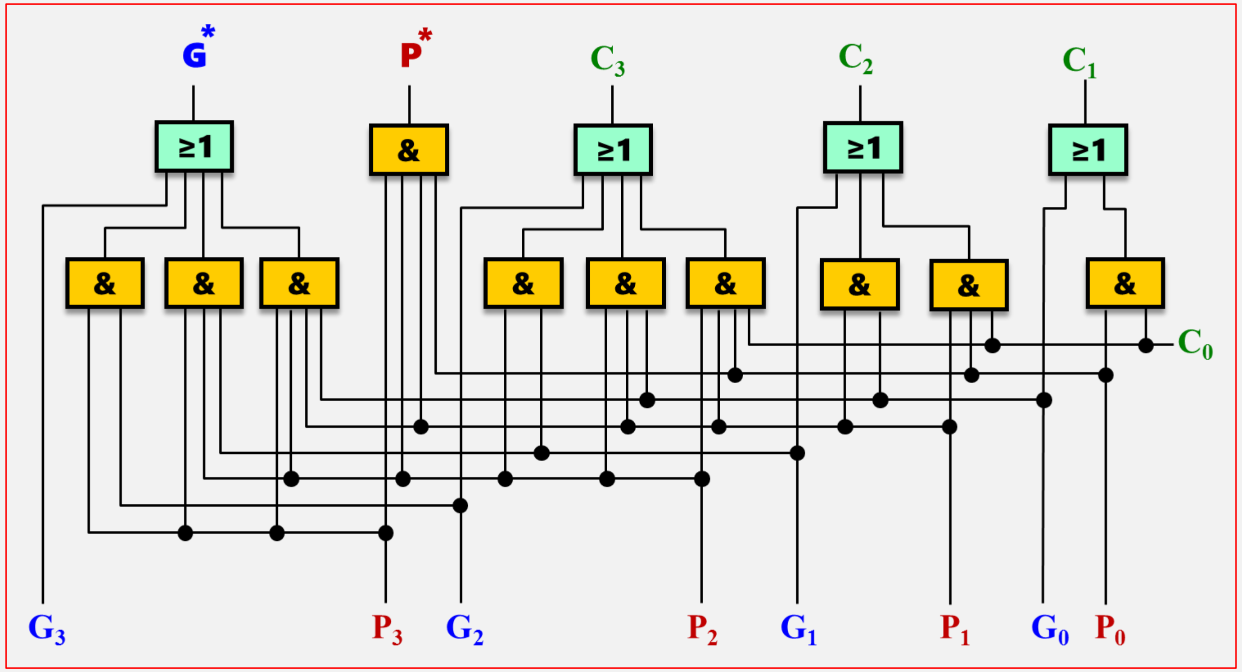
两级先行进位电路



## 7. 快速加法器设计

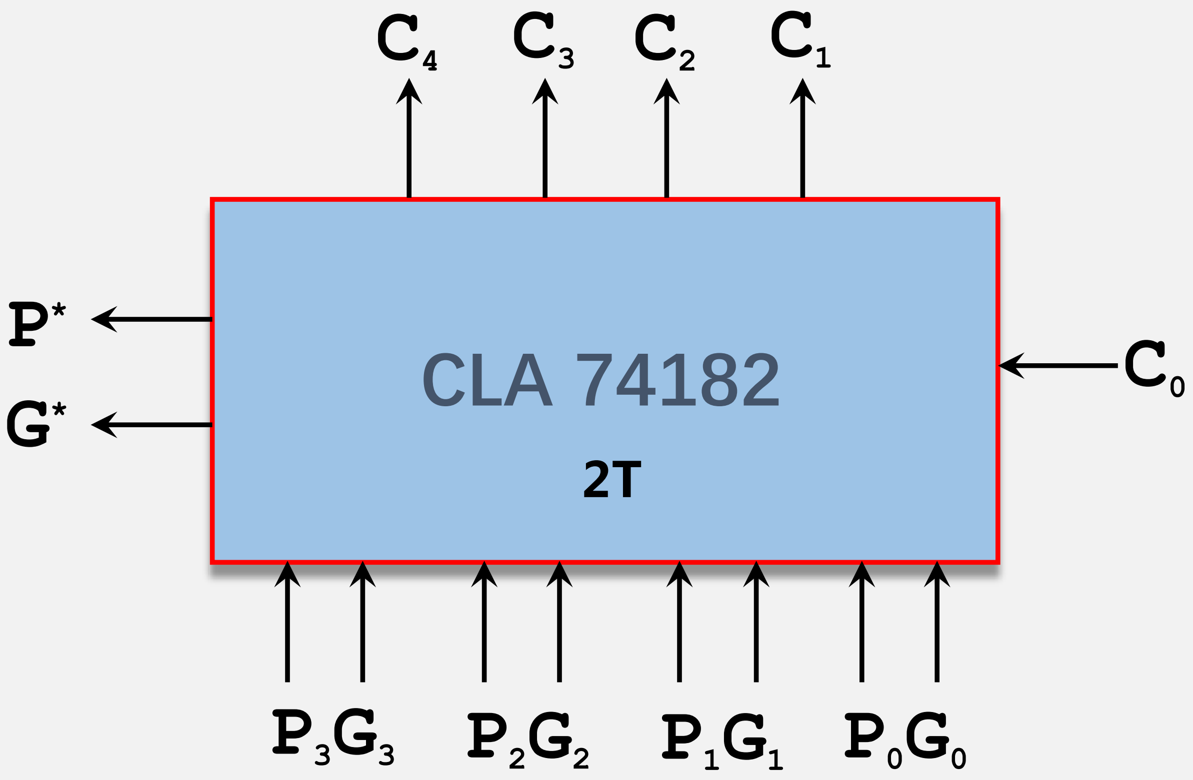


$C_4$
$C_3$
$C_2$
$C_1$
≥1
&
$C_0$
$G_3$
$P_3$
$G_2$
$P_2$
$G_1$
$P_1$
$G_0$
$P_0$

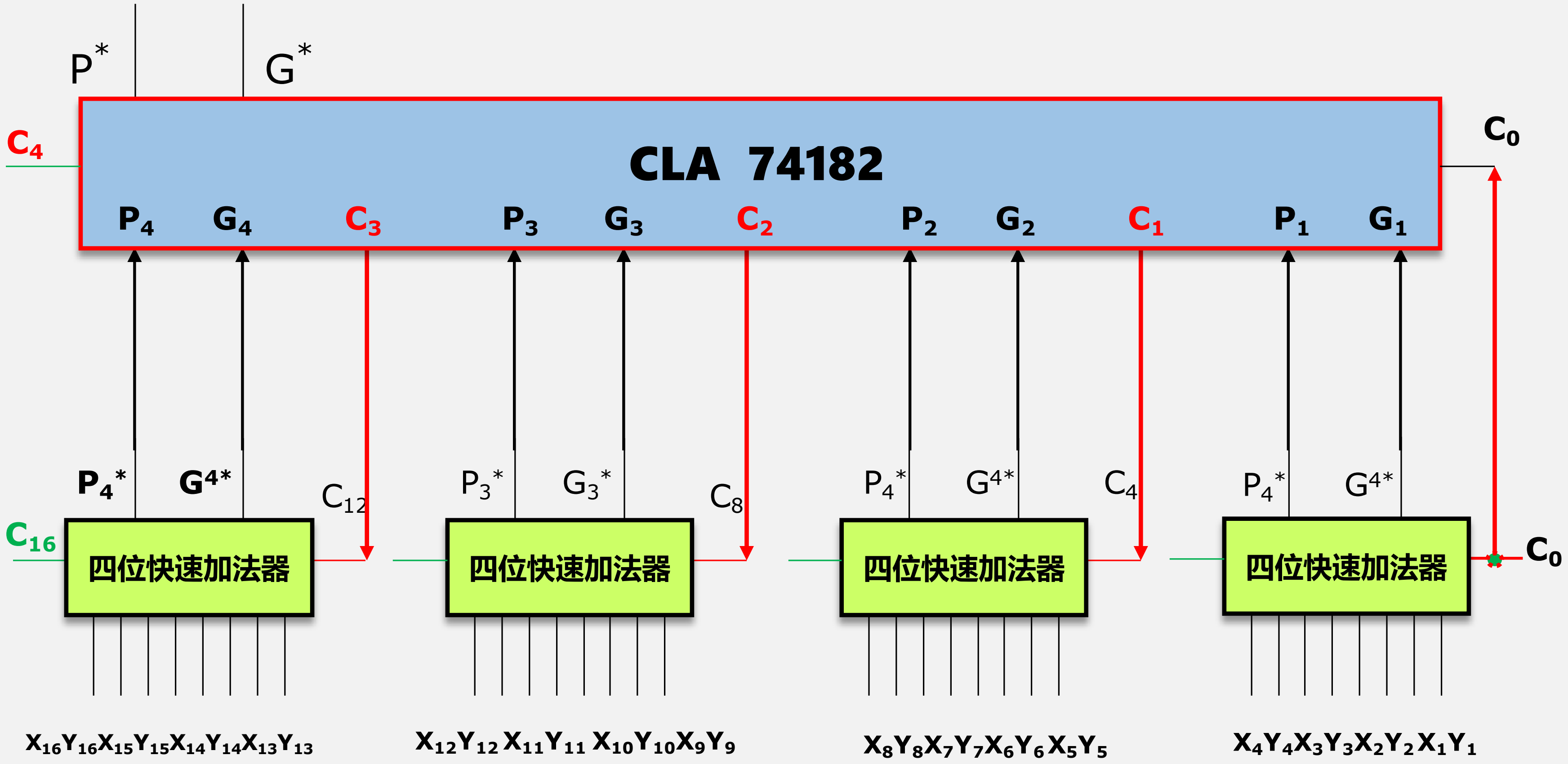
$G^*$
$P^*$
$C_3$
$C_2$
$C_1$
≥1
&
$C_0$
$G_3$
$P_3$
$G_2$
$P_2$
$G_1$
$P_1$
$G_0$
$P_0$

7. 快速加法器设计

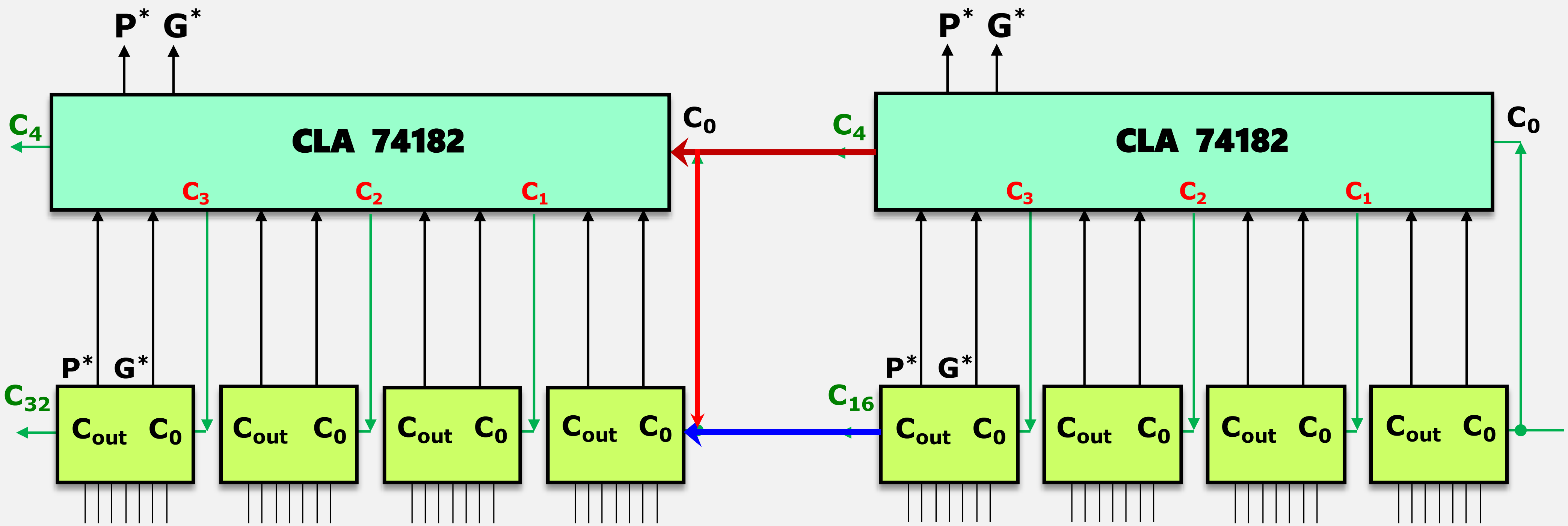


华中科技大学 计算机科学与技术学院 School of Computer Science & Technology, HUST

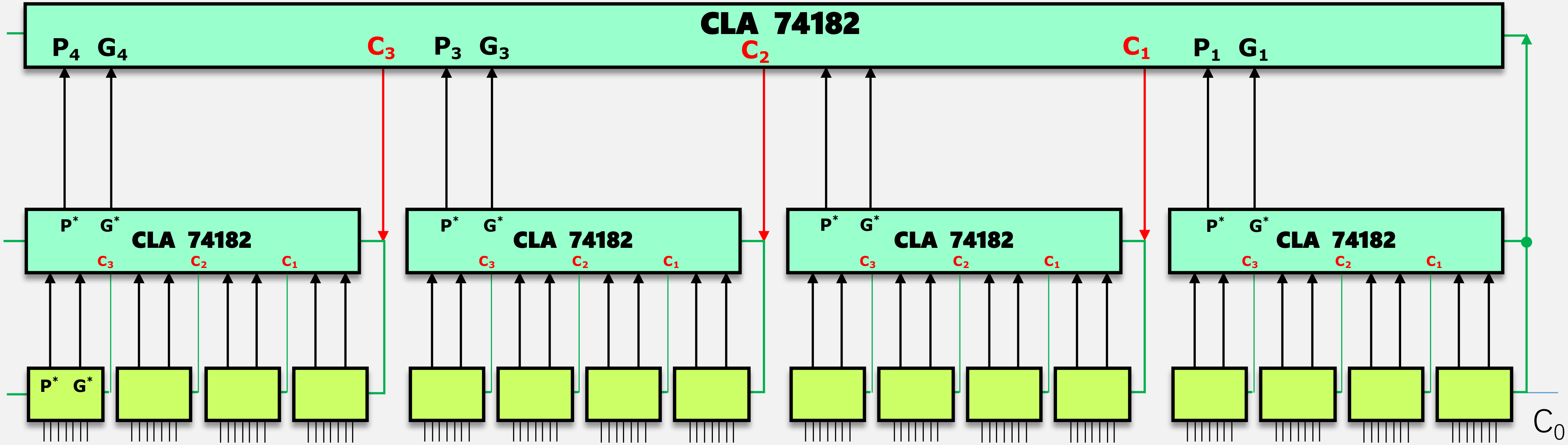
## 7. 快速加法器设计



## 7. 快速加法器设计

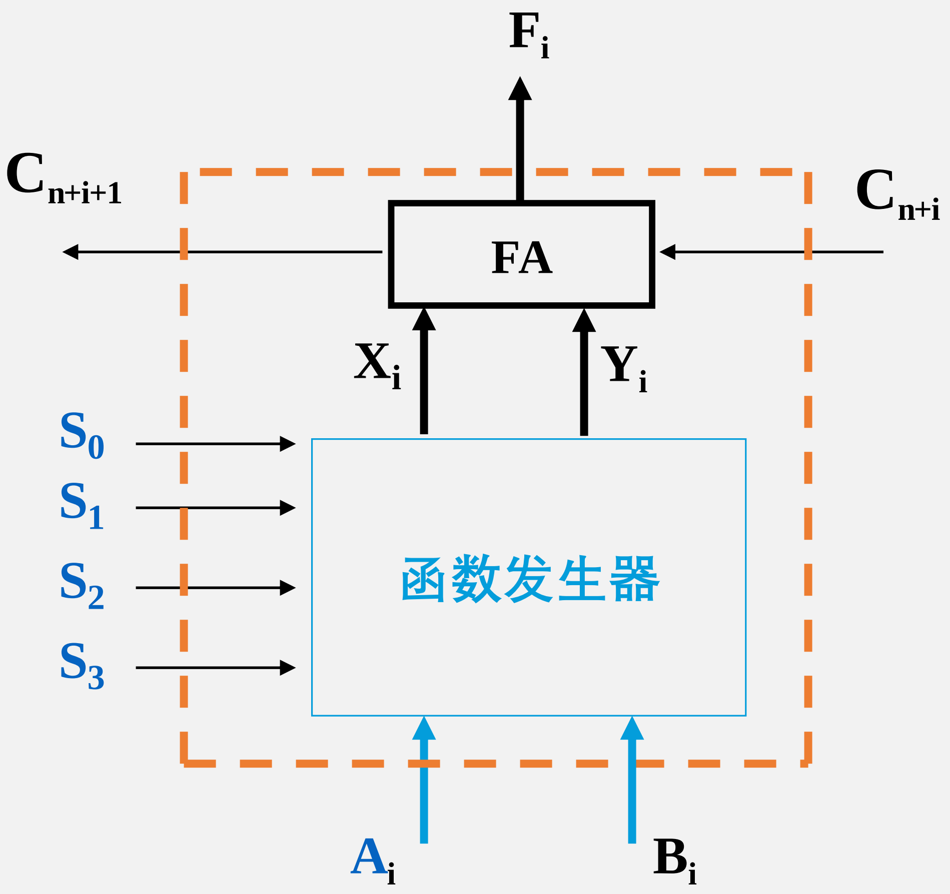

P* G*
CLA 74182
C4
C0
C3
C2
C1
P* G*
C32
Cout C0
Cout C0
Cout C0
Cout C0
P* G*
CLA 74182
C4
C0
C3
C2
C1
P* G*
C16
Cout C0
Cout C0
Cout C0
Cout C0

## 7. 快速加法器设计



华中科技大学
计算机科学与技术学院
School of Computer Science & Technology, HUST

## 8. 四位运算器芯片 SN74181




FA的逻辑表达式：

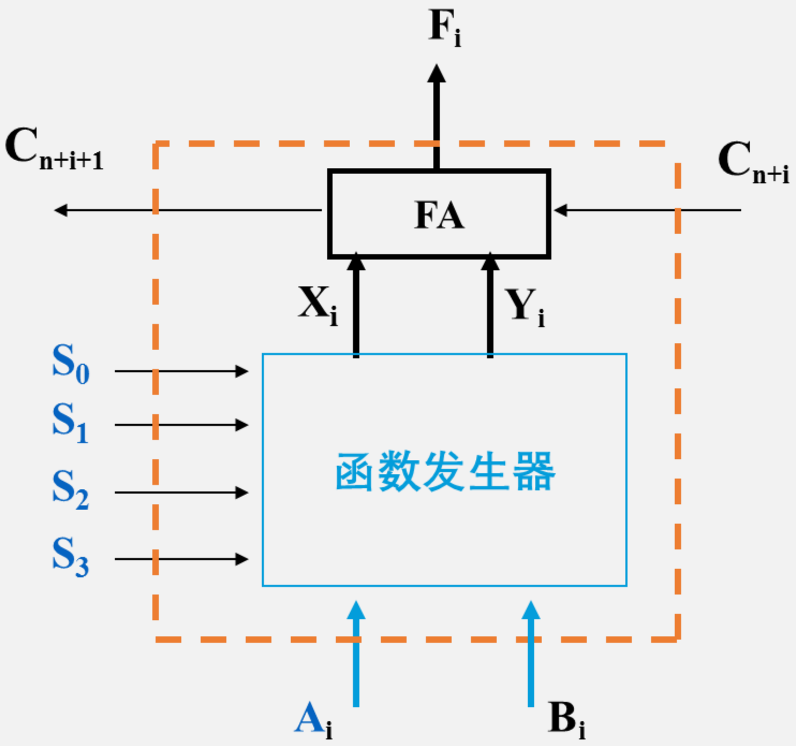
$F_i = X_i \oplus Y_i \oplus C_{n+i}$

$C_{n+i+1} = X_iY_i + Y_iC_{n+i} + X_iC_{n+I}$

# 3.1 定点数加/减运算

## 8. 四位运算器芯片 SN74181

| S0 S1 | $Y_i$ | S2 S3 | $X_i$ |
|---|---|---|---|
| 0 0 | $\bar{A}_i$ | 0 0 | 1 |
| 0 1 | $\bar{A}_iB_i$ | 0 1 | $\bar{A}_i+\bar{B}_i$ |
| 1 0 | $\bar{A}_i\ \bar{B}_i$ | 1 0 | $\bar{A}_i+B_i$ |
| 1 1 | 0 | 1 1 | $\bar{A}_i$ |




$$X_i=\bar{S}_2\bar{S}_3+\bar{S}_2S_3(\bar{A}_i+\bar{B}_i)+S_2\bar{S}_3(\bar{A}_i+B_i)+S_2S_3\bar{A}_i$$

$$Y_i=\bar{S}_0\bar{S}_1\bar{A}_i+\bar{S}_0S_1\bar{A}_iB_i+S_0\bar{S}_1\bar{A}_i\bar{B}_i$$

⇒

$$X_i=\overline{S_3A_iB_i+S_2A_i\bar{B}_i}$$

$$Y_i=\overline{A_i+S_0B_i+S_1\bar{B}_i}$$

⇒

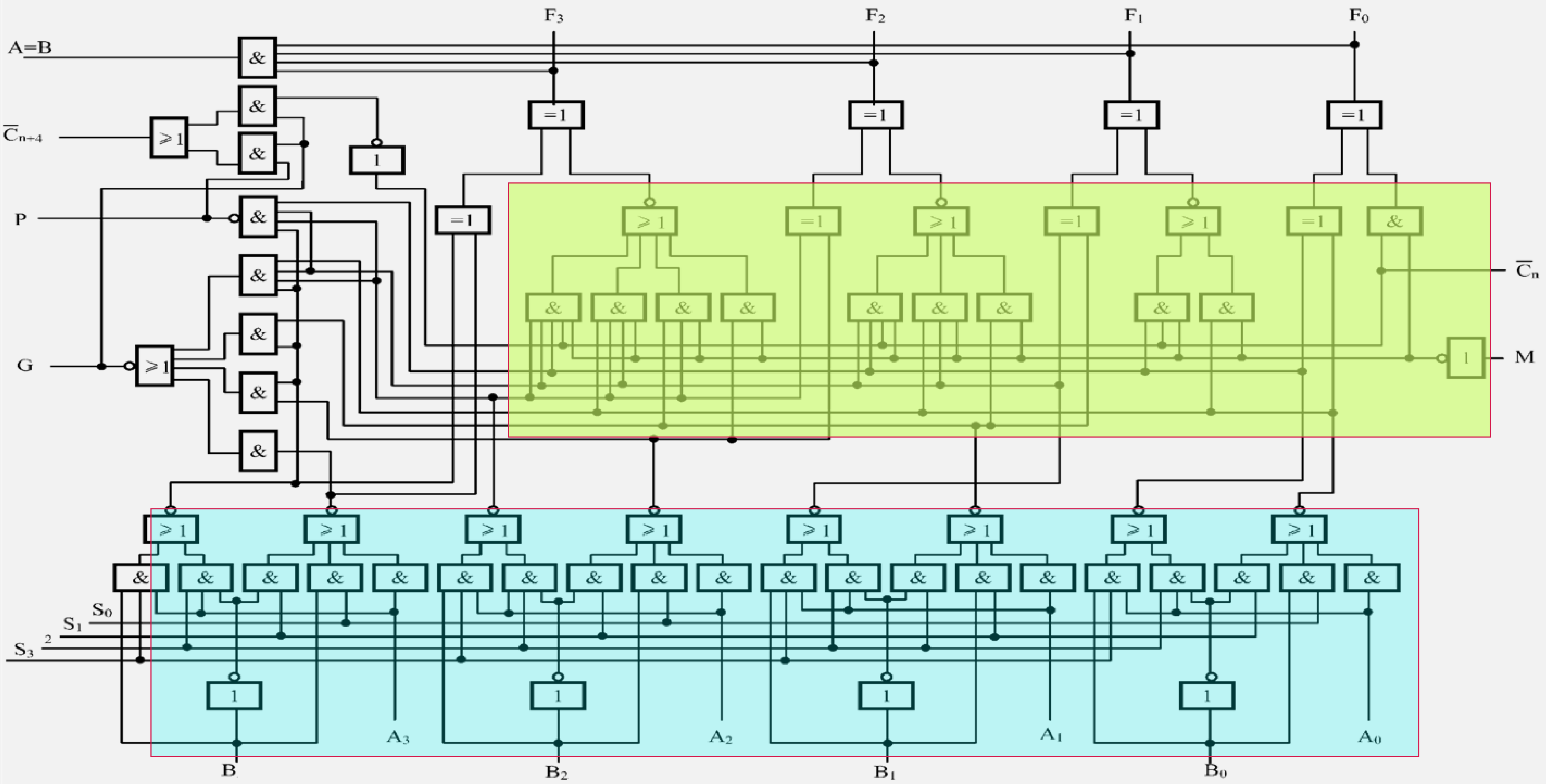
FA的逻辑表达式：

$F_i = X_i \oplus Y_i \oplus C_{n+i}$

$C_{n+i+1}=X_iY_i+Y_iC_{n+i}+X_iC_{n+l}$

8. 四位运算器芯片 SN74181





# 3.1 定点数加/减运算

## 8. 四位运算器芯片 SN74181

♦加、减: 算术运算， + :逻辑运算;

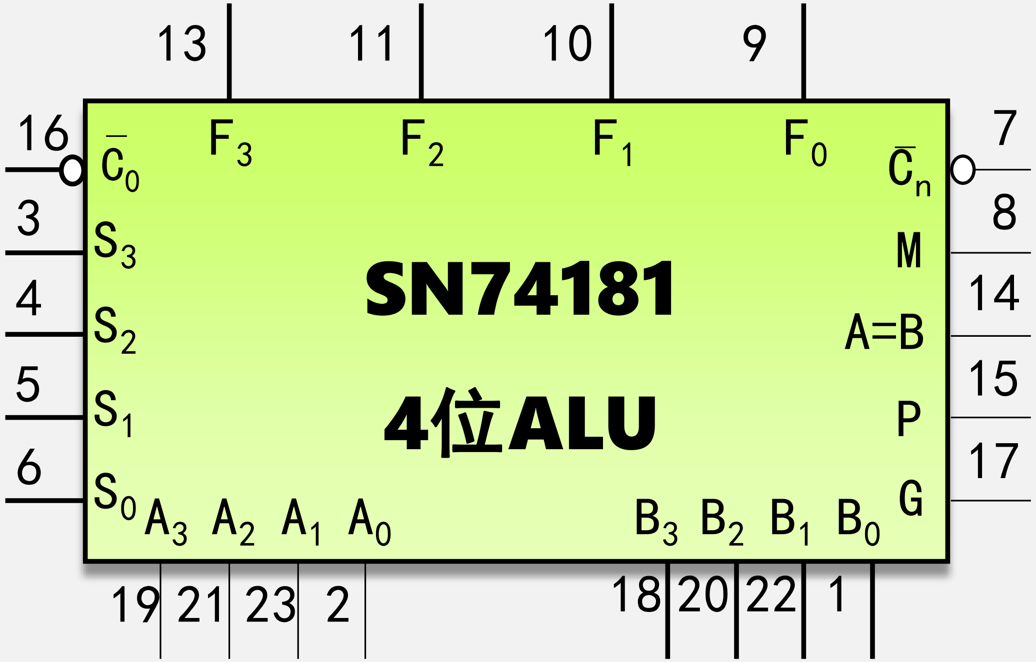
♦ M=H时逻辑运算， M=L 时算术运算;

♦算术运算结果和过程均用补码表示;

♦ 当$S_3S_2S_1S_0$ =LHHL，M=L , C=0时 ， F = ?

| $S_3S_2S_1S_0$ | 逻辑运算 M=H | 算术运算 M=L $G_n$ =H |
|---|---|---|
| LLLL | $F=\overline{A}$ | F=A |
| LLLH | $F=\overline{A+B}$ | F=A + B |
| LLHL | $F=\overline{A}B$ | $F=A+\overline{B}$ |
| LLHH | F=逻辑 0 | F= -1 |
| LHLL | $F=\overline{AB}$ | F=A 加 $A\overline{B}$ |
| LHLH | F=B | F=(A+B)加 $A\overline{B}$ |
| LHHL | $F=A\oplus B$ | F=A 减 B 减 1 |
| LHHH | $F=A\overline{B}$ | F=$A\overline{B}$减1 |
| HLLL | $F=\overline{A}+B$ | F=A 加 AB |
| HLLH | $F=\overline{A\oplus B}$ | F=A 加 B |
| HLHL | F=B | F=(A+$\overline{B}$)加 AB |
| HLHH | F=AB | F=AB减1 |
| HHLL | F=1 | F=A 加 A |
| HHLH | $F=A+\overline{B}$ | F=(A+B)加A |
| HHHL | F=A + B | F=(A+$\overline{B}$)加 A |
| HHHH | F=A | F=A减1 |

8. 四位运算器芯片 SN74181



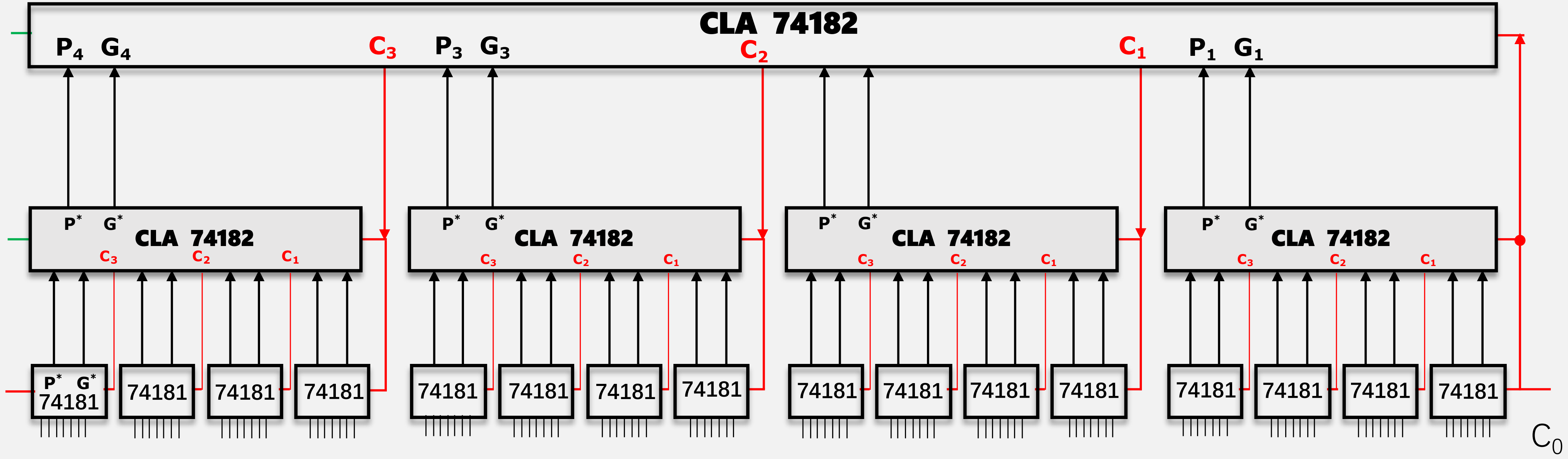

前面讲过的三种溢出检测方法可直接用于74181吗？

$$\text{Overflow}_1 = X_nY_n\overline{S_n} + \overline{X_n}\,\overline{Y_n}S_n$$

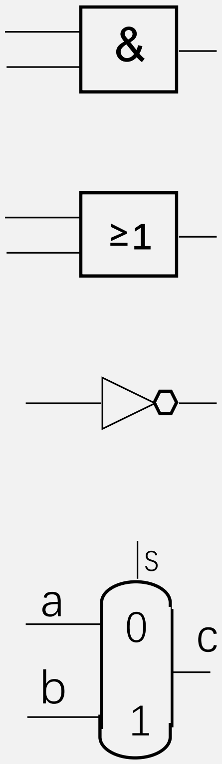
$$\text{Overflow}_2 = C_f \oplus C_n$$

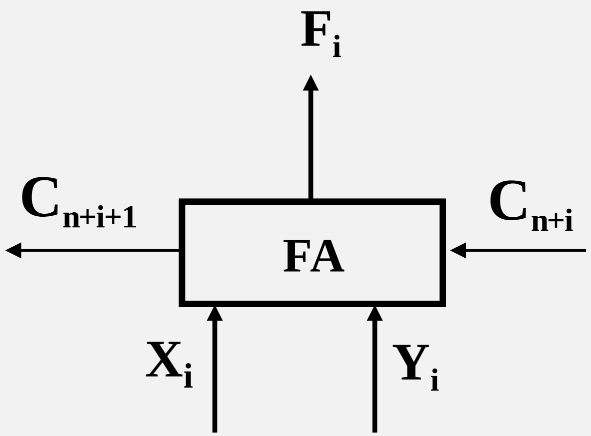
$$\text{Overflow}_3 = f_1 \oplus f_2$$

## 8. 四位运算器芯片 SN74181



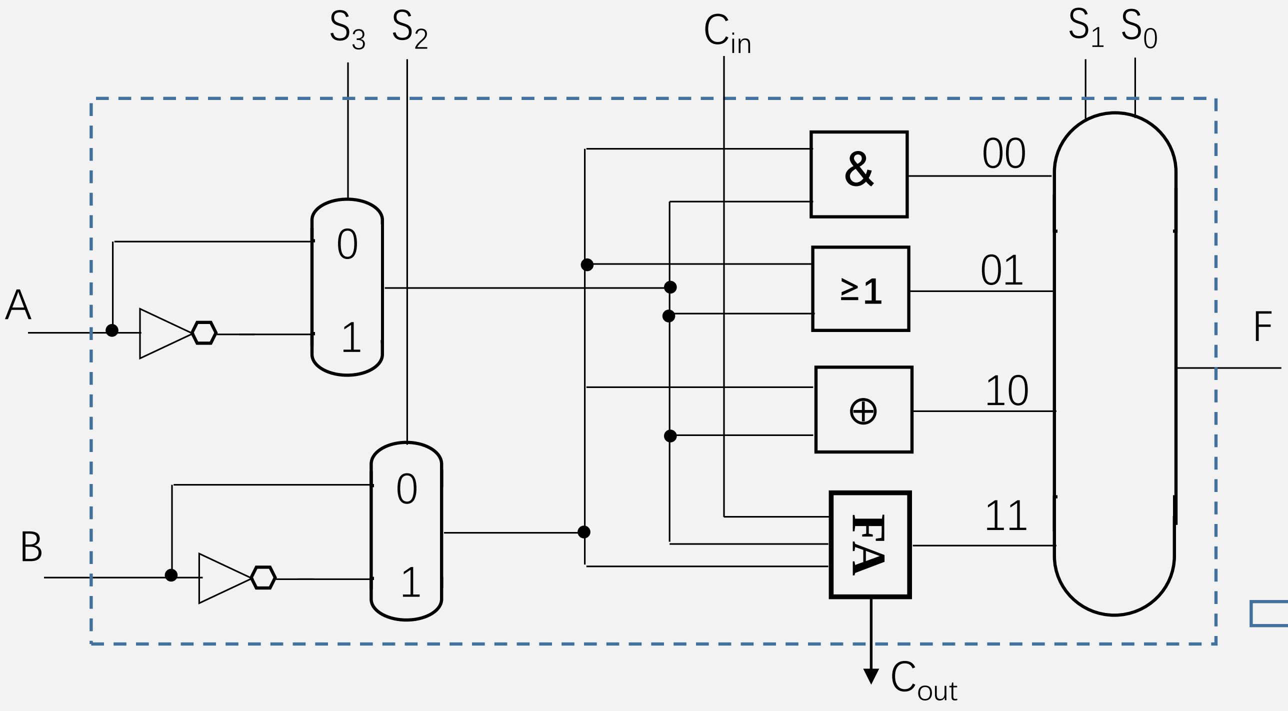
## 9. 运算器设计其他方法(结构)




**if (s= = 0,c = a ;  else c=b )**

## 9. 运算器设计其他方法(结构)




$S_3S_2\,S_1S_0$= 0000: F=AB

$S_3S_2\,S_1S_0$= 1100: $F=\bar{A}\bar{B}$

$S_3S_2\,S_1S_0$= 1000: $F=\bar{A}B$

$S_3S_2\,S_1S_0$= 0100: $F=A\bar{B}$

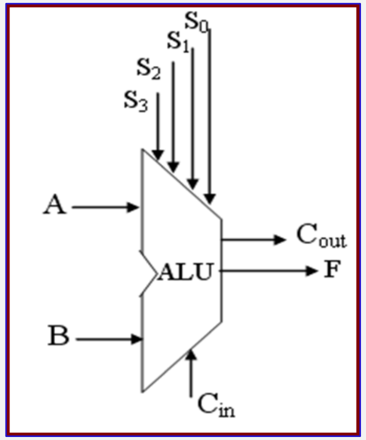
$S_3S_2\,S_1S_0$= 0001: F=A+B

$S_3S_2\,S_1S_0$= 0010: F=A⊕B

$S_3S_2\,S_1S_0$= 0011 $C_{in}$=0: F=A加B

$S_3S_2\,S_1S_0$= 0111 $C_{in}$=1: F=A减B

.........



9. 运算器设计其他方法(硬件描述语言：Verilog)

```
moudle  add_4 (x,y, sum ,cin, cout )
input[3:0] x,y ;
outpu[3:0] sum ;
output cout ;
input cin ;
assign  {cout, sum } = x + y + cin;
endmoudle
```

```
moudle  add_4 (x,y, sum ,cin, cout )
input[3:0] x,y ;
outpu[3:0] sum ;
output cout ;
input cin ;
sum < = x xor  y xor cin;
cout < = (x and y) or ( x and cin ) or ( y and cin )
endmoudle
```

配合FPGA使用 （Field Programmable Gate Array）

# 第一部分完