计算机组成原理

定点加减法运算

王浩宇,教授

haoyuwang@hust.edu.cn

https://howiepku.github.io/

目录

- 位运算与逻辑运算
- 定点加法、减法运算
- 二进制加法减法器

Boolean Algebra

- **Developed by George Boole in 19th Century**
  - Algebraic representation of logic
    - Encode “True” as 1 and “False” as 0

And

- **A&B = 1 when both A=1 and B=1**

&	0	1
0	0	0
1	0	1

Or

- **A|B = 1 when either A=1 or B=1**

|	0	1
0	0	1
1	1	1

Not

- **~A = 1 when A=0**

Exclusive-Or (Xor)

- **A^B = 1 when either A=1 or B=1, but not both**

General Boolean Algebras

- **Operate on Bit Vectors**
  - Operations applied bitwise

```
  01101001     01101001     01101001
& 01010101   | 01010101   ^ 01010101   ~ 01010101
  --------     --------     --------     --------
  01000001     01111101     00111100     10101010
```

- **All of the Properties of Boolean Algebra Apply**

Bit-Level Operations in C

- **Operations &, |, ~, ^ Available in C**
  - Apply to any “integral” data type
    - long, int, short, char, unsigned
  - View arguments as bit vectors
  - Arguments applied bit-wise

```c
#include <stdio.h>

int main(){
    unsigned int i = 0x12345678;
    unsigned int j = 0xFF;
    unsigned int k = i & j;
    printf("%X\n",k);
}
```

```
root@ubuntu:/opt/shell# ./hello
78
```

Contrast: Logic Operations in C

- **Contrast to Logical Operators**
  - &&, ||, !
    - View 0 as “False”
    - Anything nonzero as “True”
    - **Always return 0 or 1**
    - Early termination

```c
#include <stdio.h>

int main(){
      unsigned int x = 0x12345678;
      unsigned int i = !x;
      unsigned int j = ~x;
      unsigned int m = !!x;
      unsigned int n = ~~x;
      printf("%u %u\n",i,j);
      printf("%u %u\n",m,n);
}
```

```
root@ubuntu:/opt/shell# ./hello
0 3989547399
1 305419896
```

Shift Operations

- **Left Shift: `x << y`**
  - Shift bit-vector `x` left `y` positions
    - Throw away extra bits on left
  - Fill with 0's on right
- **Right Shift: `x >> y`**
  - Shift bit-vector `x` right `y` positions
    - Throw away extra bits on right
  - Logical shift
    - Fill with 0's on left
  - Arithmetic shift
    - Replicate most significant bit on left
- **Undefined Behavior**
  - Shift amount < 0 or ≥ word size

Argument x	01100010
<< 3	00010*000*
Log. >> 2	*00*011000
Arith. >> 2	*00*011000

Argument x	10100010
<< 3	00010*000*
Log. >> 2	*00*101000
Arith. >> 2	*11*101000

定点加法、减法运算

- 补码加法
- 补码减法
- 溢出概念与检验方法
- 基本的二进制加法、减法器

补码加法

- **补码加法运算基本公式**
  - 定点整数：$[x+y]_{补} = [x]_{补} + [y]_{补} \quad (\text{mod } 2^{n+1})$
    定点小数：$[x+y]_{补} = [x]_{补} + [y]_{补} \quad (\text{mod } 2)$
- **证明**

（1）

$$[x]_{补} = \begin{cases} x & 1 > x \geqslant 0 \\ 2+x = 2 - |x| & 0 \geqslant x \geqslant -1 \end{cases} \quad (\text{mod } 2)$$

（2）证明思路：分三种情况。

(a) x、y均为正值（x > 0， y > 0）

(b) x、y一正一负（x > 0， y < 0，或者 x <0， y >0）

(c) x、y均为负值（x <0， y <0）

补码加法公式证明（1/2）

证明：

(a) $x > 0, y > 0$

$$[x]_{补} + [y]_{补} = x + y = [x+y]_{补} \quad (\text{mod } 2)$$

(b) $x < 0, y < 0$

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

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

$= 2 + [x+y]_{补}$ （mod 2）

$= [x+y]_{补}$

补码加法公式证明（2/2）

(c) X > 0, Y <0 （x <0， y >0的证明与此相同）

∵ $[x]_{补}=x$ ， $[y]_{补}=2+y$

∴ $[x]_{补}+[y]_{补}= x+2+y =2+(x+y)$

当x+y>0时， 2+(x+y) > 2 ， 进位2必丢失；

因(x+y)> 0 ， 故$[x]_{补}+[y]_{补}= x+y =[x+y]_{补}$ (mod 2)

当x+y<0时， 2+(x+y) < 2

因(x+y) < 0 ， 故$[x]_{补}+[y]_{补}= 2+(x+y)$

$= [x+y]_{补}$ (mod 2)

定点数补码加法举例

[例11] $x = +1001,\ y = +0101$,

求 $x + y$。

解:

$[x]_{补} = 0\ 1001,\ [y]_{补} = 0\ 0101$

$[x]_{补}$	0 1001
$+[y]_{补}$	0 0101
$[x+y]_{补}$	0 1110

所以 $x + y = +1110$

[例12] $x = +1011,\ y = -0101$,

求 $x + y$。

解:

$[x]_{补} = 0\ 1011,\ [y]_{补} = 1\ 1011$

$[x]_{补}$	0 1011
$+[y]_{补}$	1 1011
$[x+y]_{补}$	10 0110

所以 $x + y = +0110$

补码减法

- **补码减法运算基本公式**
  - 定点整数：$[x-y]_{补}=[x]_{补}-[y]_{补}=[x]_{补}+[-y]_{补}$ （mod 2^{n+1}）
  - 定点小数：$[x-y]_{补}=[x]_{补}-[y]_{补}=[x]_{补}+[-y]_{补}$ （mod 2）
- **证明：只需要证明 $[-y]_{补}=-[y]_{补}$**
  - 已证明$[x+y]_{补}=[x]_{补}+[y]_{补}$，故$[y]_{补}=[x+y]_{补}-[x]_{补}$
  - 又$[x-y]_{补}=[x]_{补}+[-y]_{补}$，故$[-y]_{补}=[x-y]_{补}-[x]_{补}$
  - 可得$[y]_{补}+[-y]_{补}=[x+y]_{补}+[x-y]_{补}-[x]_{补}-[x]_{补}$

$$=[x+y+x-y]_{补}-[x]_{补}-[x]_{补}$$

$$=[x+x]_{补}-[x]_{补}-[x]_{补}=0$$

- **$[-y]_{补}$等于$[y]_{补}$的各位取反，末位加1 （lec-2）**

[例] x＝＋1101，y＝＋0110，求x－y。

- **解:**
  - $[x]_{补} = 0\ 1101$，$[y]_{补} = 0\ 0110$，$[-y]_{补} = 1\ 1010$
  - $[x-y]_{补} = [x]_{补} + [-y]_{补}$

$$= 0\ 1101 + 1\ 1010$$

$$= 10\ 0111$$

$$= 0\ 0111$$

$$\therefore \quad x - y = +0111$$

$$\begin{array}{rr} & 0\ 1101 \\ +) & 1\ 1010 \\ \hline & \boxed{1}0\ 0111 \end{array}$$

定点数补码加减法运算

- **基本公式**
  - 定点整数：

$$[x \pm y]_{补} = [x]_{补} + [\pm y]_{补} \quad (\mathrm{mod}\ 2^{n+1})$$

  - 定点小数：

$$[x \pm y]_{补} = [x]_{补} + [\pm y]_{补} \quad (\mathrm{mod}\ 2)$$

- **定点数补码加减法运算**
  - 符号位和数值位可同等处理；
  - 只要结果不溢出，将结果按2^{n+1}或2取模，即为本次运算结果。

例　设机器字长为8位，$[x]_{补}$=1010 0011，

$[y]_{补}$=0010 1101 ，求x−y。

解:

$[-y]_{补}$ = 1101 0011

$[x-y]_{补}$ = $[x]_{补}$ + $[-y]_{补}$

= 1010 0011 + 1101 0011

= 1 0111 0110

= 0111 0110

∴ x - y = + 118

1010 0011
＋) 1101 0011
1 0111 0110

x= −93，y= +45

计算过程中，产生了溢出！

−93−45=-138 < −128

溢出概念与检测方法

- **溢出**
  - 在定点数机器中，**数的大小超出定点数能表示范围**
- **上溢**
  - 数据大于机器所能表示的最大正数；
- **下溢**
  - 数据小于机器所能表示的最小负数；
- **例如，4位补码表示的定点整数，范围为[−8，+7]**
  - 若x = 5，y = 4，则x+y产生上溢
  - 若x = −5，y = −4，则x+y产生下溢
  - 若x = 5，y = −4，则x−y产生上溢

例题

[例] x = +0.1011, y = +0.1001,
求 x + y

[解:] $[x]_{补} = 0.1011$

$[y]_{补} = 0.1001$

$[x]_{补}$	0.1
$+ [y]_{补}$	0.1001
$[x + y]_{补}$	1.0100

正数+正数=负数

[例] x = -0.1101, y = -0.1111,
求 x + y

[解:] $[x]_{补} = 1.0011$

$[y]_{补} = 1.0001$

$[x]_{补}$	1.0011
$+ [y]_{补}$	1.0001
$[x + y]_{补}$	0.0101

负数+负数=正数

溢出！

溢出判别方法——直接判别法

- **方法：**
  - 同号补码相加，结果符号位与加数相反；
  - 异号补码相减，结果符号位与减数相同；
- **特点：硬件实现较复杂**
- **举例：**
  - 若$[x]_{补}$=0101，$[y]_{补}$=0100，则$[x+y]_{补}$=1001　**上溢**
  - 若$[x]_{补}$=1011，$[y]_{补}$=1100，则$[x+y]_{补}$=0111　**下溢**
  - 若$[x]_{补}$=0101，$[y]_{补}$=1100，则$[x-y]_{补}$=1001　**上溢**

溢出判别方法——变形补码判别法

- 变形补码，也叫**模4补码**：采用双符号位表示补码
- 判别方法：

双符号位	结果
00	正
01	上溢
10	下溢
11	负

- 特点：硬件实现简单，只需对结果符号位进行异或
- 举例：
  - 若$[x]_{补}$=00101，$[y]_{补}$=00100，则$[x+y]_{补}$=01001　　上溢
  - 若$[x]_{补}$=11011，$[y]_{补}$=11100，则$[x+y]_{补}$=10111　　下溢
  - 若$[x]_{补}$=00101，$[y]_{补}$=11100，则$[x-y]_{补}$=01001　　上溢

溢出判别方法——进位判别法

- **判别方法：**
  - 最高数值位的进位与符号位的进位是否相同；
- **判别公式**
  - 溢出标志$V=C_f \oplus C_{n-1}$

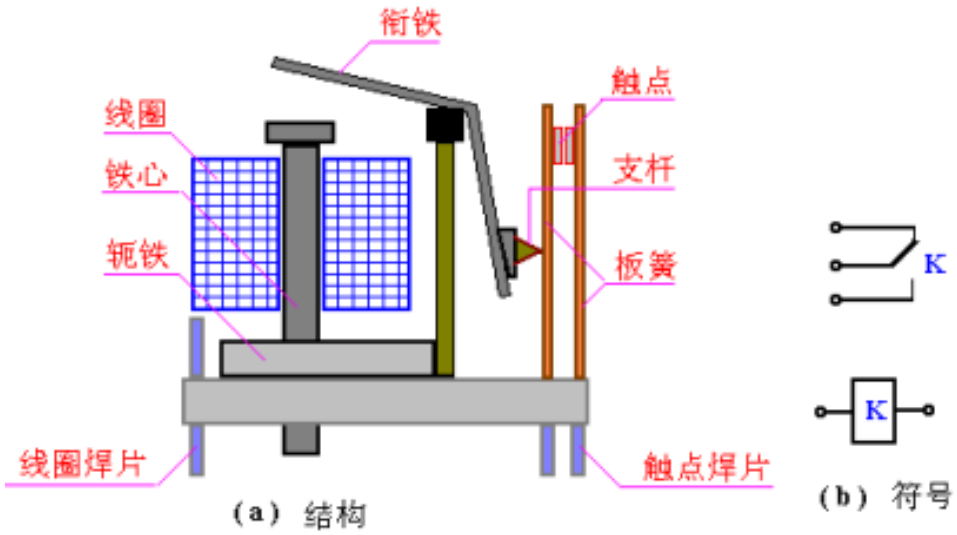
  其中C_f为符号位产生的进位，C_{n-1}为最高数值位产生的进位。
- **举例：**

```
   0101          0001
+) 0100       +) 0100
------        ------
   1001          0101
   ↓↓            ↓↓
   0 1           0 0
```

$V=0\oplus1=1$　　$V=0\oplus0=0$

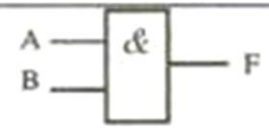
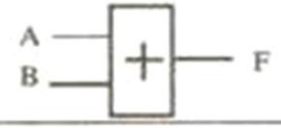
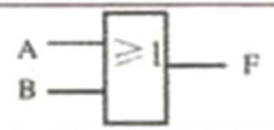
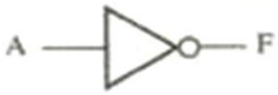
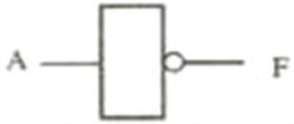
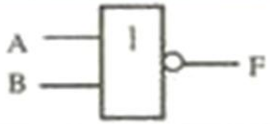
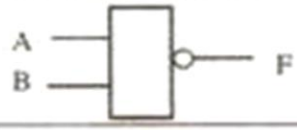
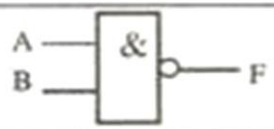
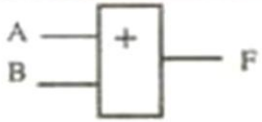
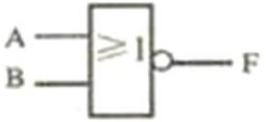
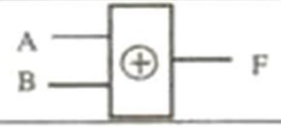
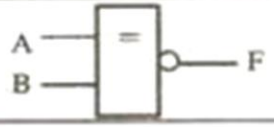
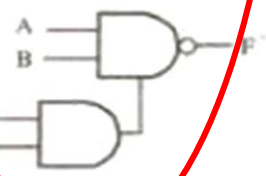
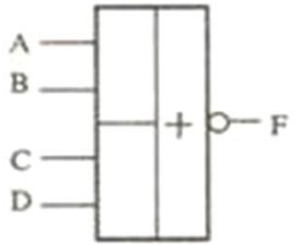
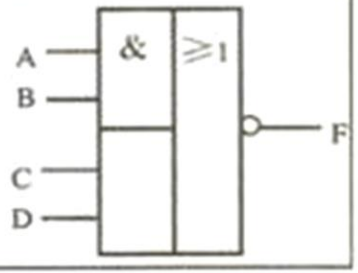
基本的二进制加法/减法器

- **继电器与逻辑门**
  - 继电器是逻辑门中的基本组成单元



  - **逻辑门电路** 把继电器按照不同的方式组合起来，形成不同的逻辑门，用来表示逻辑

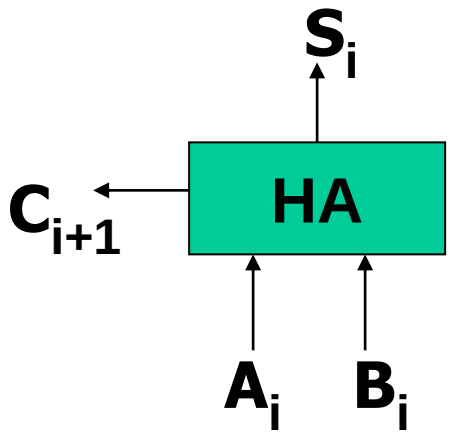
回顾逻辑门图形符号

逻辑门	IEEE 推荐符号	我国部颁符号	输出表达式	标准符号
"与"门			F=A·B	
"或"门			F=A+B	
"非"门			$F=\overline{A}$	
"与非"门			$F=\overline{A \cdot B}$	
"或非"门			$F=\overline{A+B}$	
"异或"门			$F=A \oplus B$ $=A\overline{B}+\overline{A}B$	
"与或非"门			$F=\overline{AB+CD}$	

基本的二进制加法/减法器

- 一位二进制数据的半加器：
  - 加数：A_i、B_i
  - 结果：S_i（和）

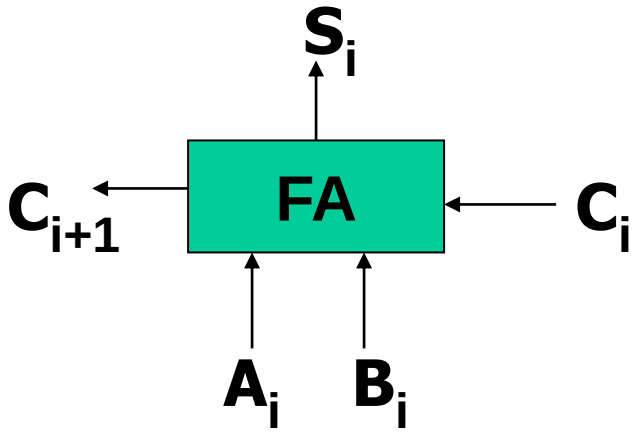
  C_{i+1}（本位向高位的进位）
- 一位半加器示意图：



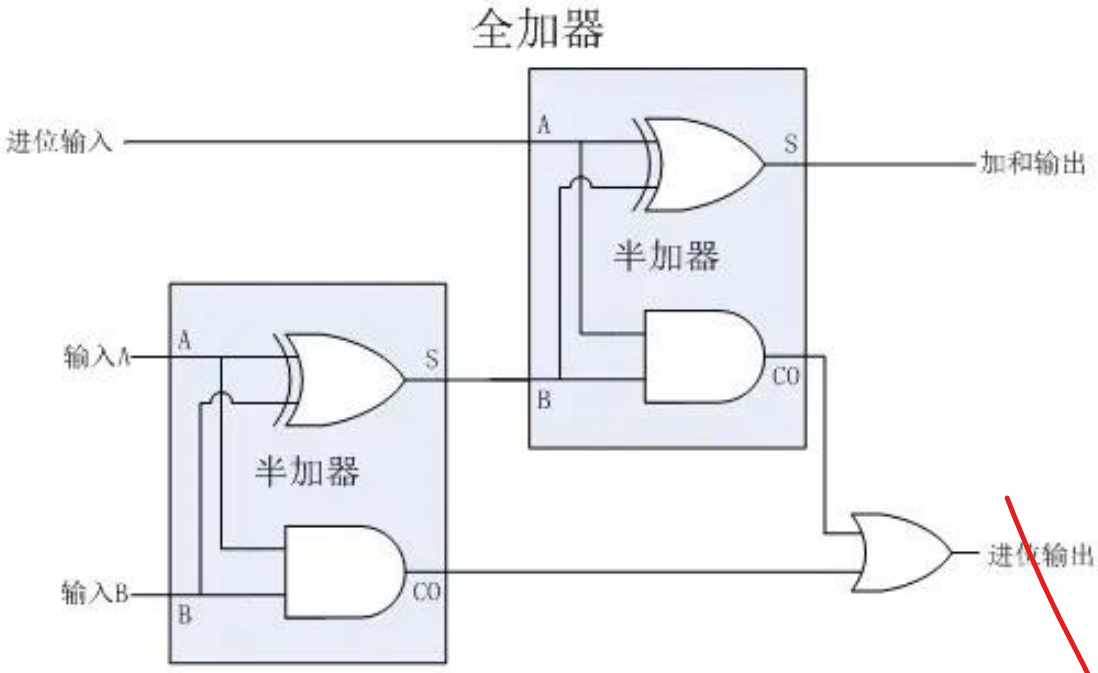
- 一位二进制数据的全加器：
  - 加数：A_i、B_i

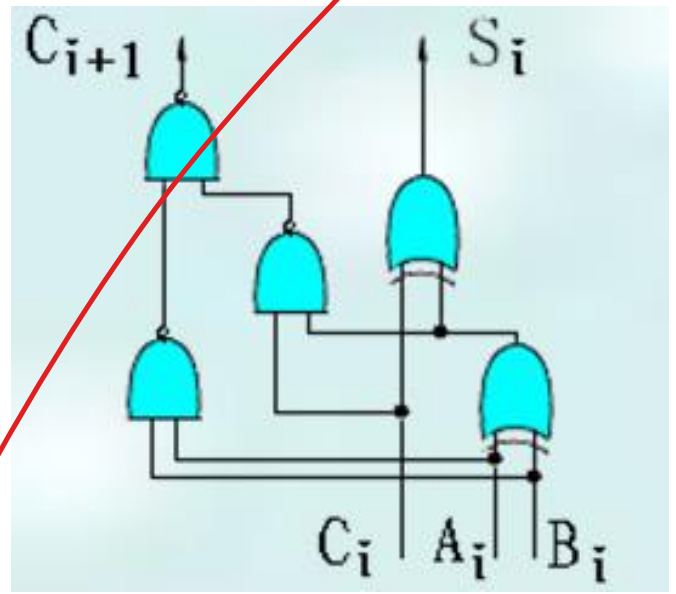
  C_i（低位向本位的进位）
  - 结果：S_i（和）

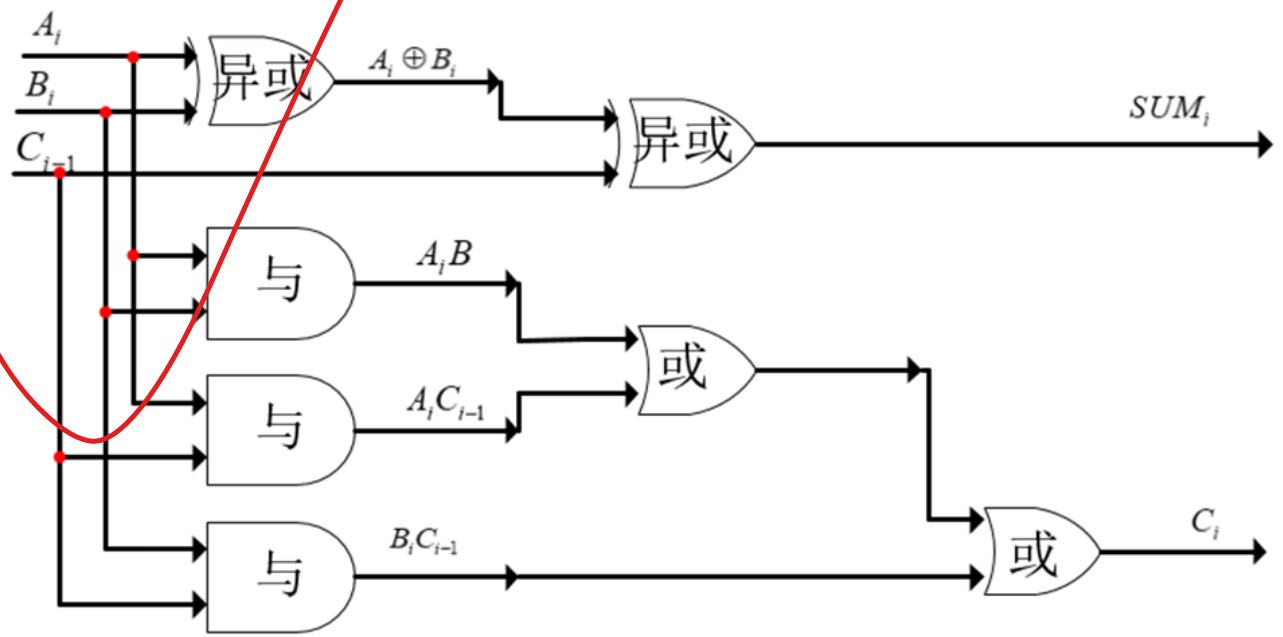
  C_{i+1}（本位向高位的进位）
- 一位全加器示意图：



基本的二进制加法/减法器



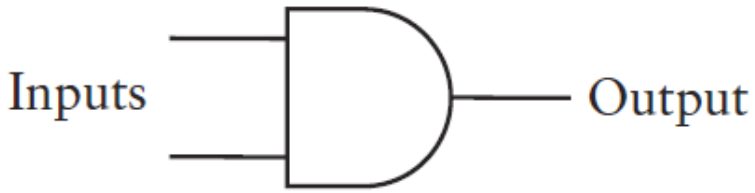




WHY?

计算和和计算进位逻辑电路构造

- **二进制进位的计算**
  - 当两个加数都0，结果为0
  - 其中一个是1，结果为0
  - 两个都是1，结果为1

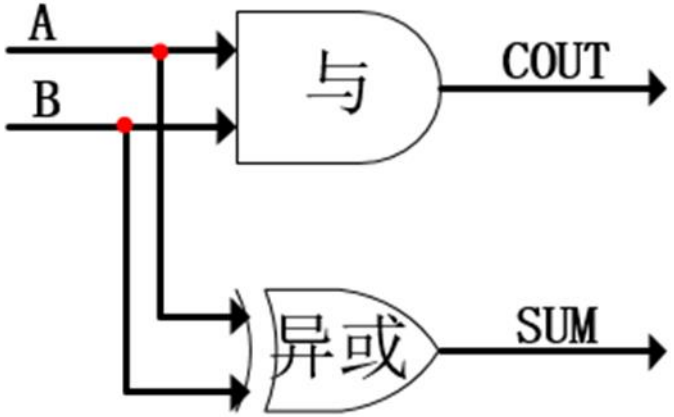


计算和的逻辑电路构造

+和	0	1
0	0	1
1	1	0

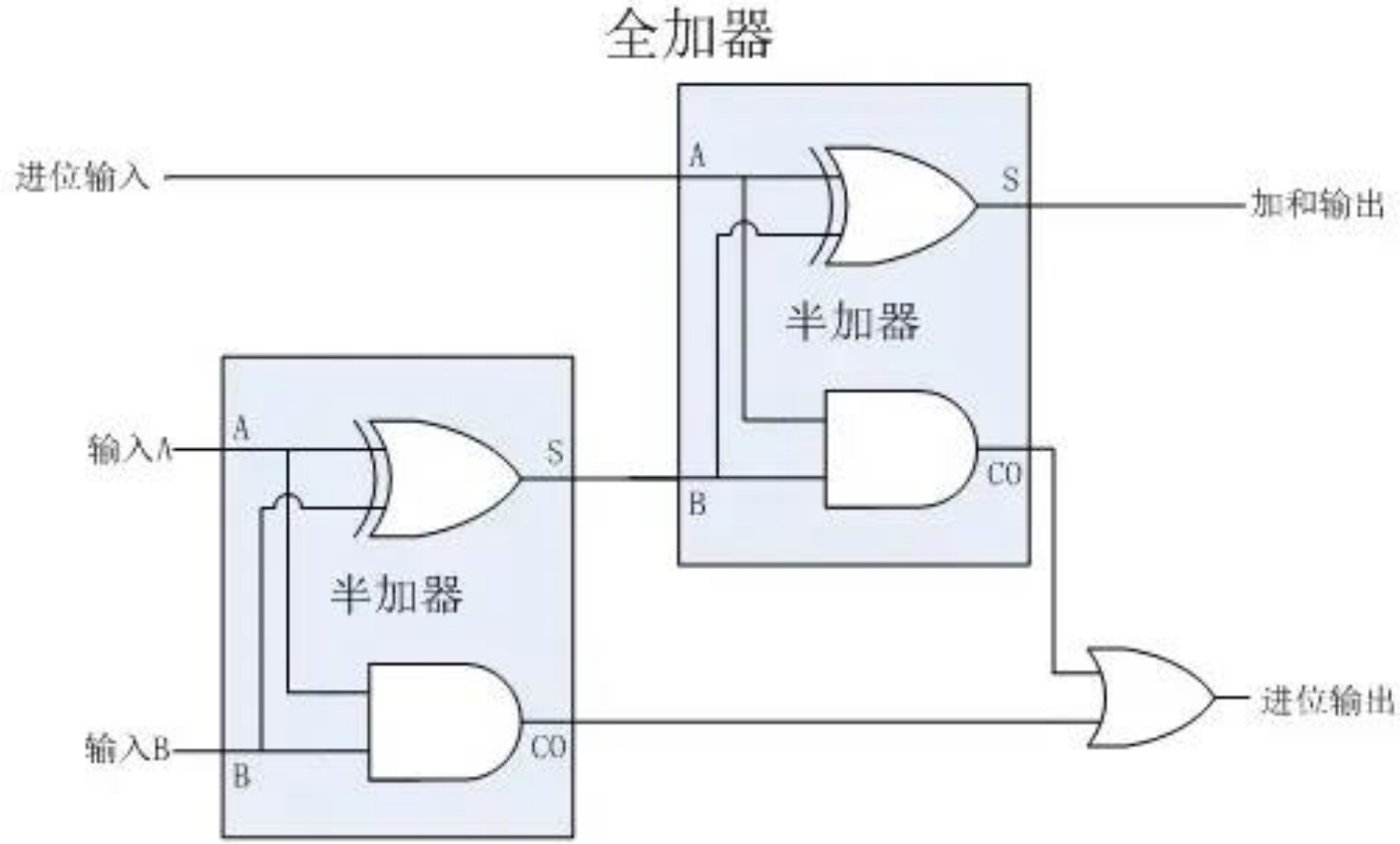
计算和

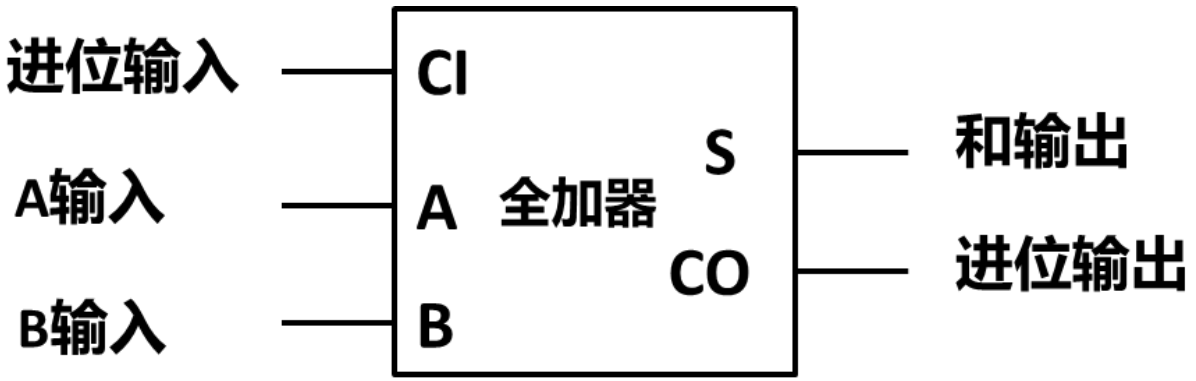
异或门



半加器

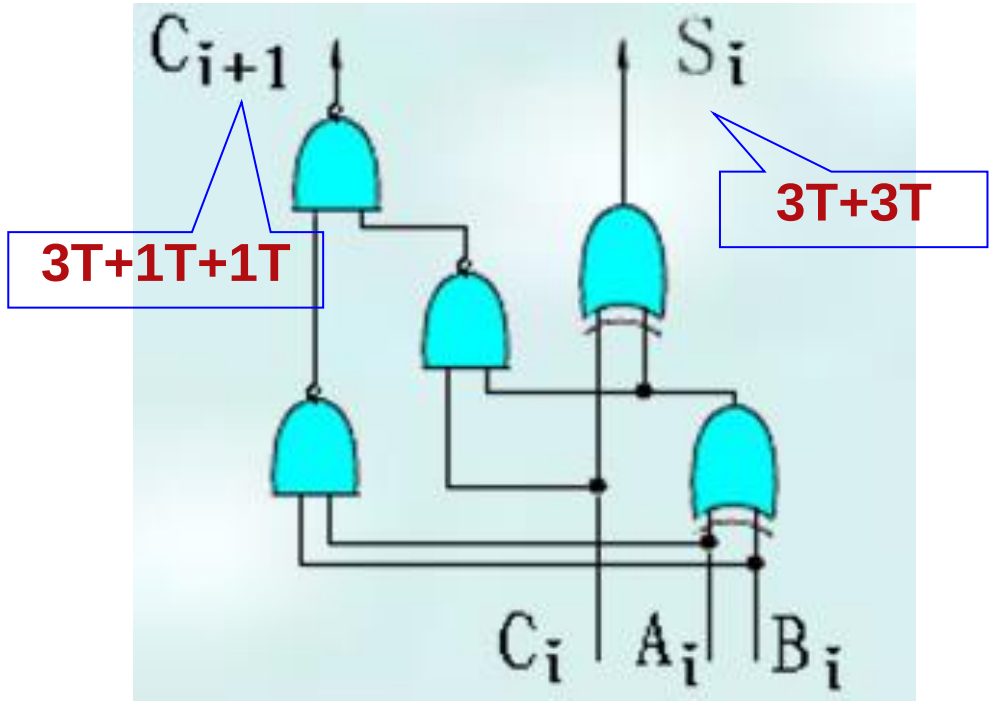
全加器


全加器
进位输入
A
S
加和输出
半加器
输入A
A
S
CO
B
半加器
输入B
B
CO
进位输出


进位输入
CI
A输入
A
全加器
S
和输出
CO
进位输出
B输入
B

一位二进制数据的全加器的逻辑结构

- 全加运算的真值表如右所示：
- 两个输出端的逻辑表达式
  - $S_i = A_i \oplus B_i \oplus C_i$
  - $C_{i+1} = A_iB_i + B_iC_i + C_iA_i = A_iB_i + (A_i \oplus B_i)C_i$
  - 全加器逻辑结构示例：



输入			输出	
A_i	B_i	C_i	S_i	C_{i+1}
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

补充内容

- 典型门电路的时间延迟

门的种类/延时符号	延迟时间
非门Tf	T
与门Ta	2T
或门To	2T
与非门Tf	T
或非门Tf	T
异或门Teo	3T

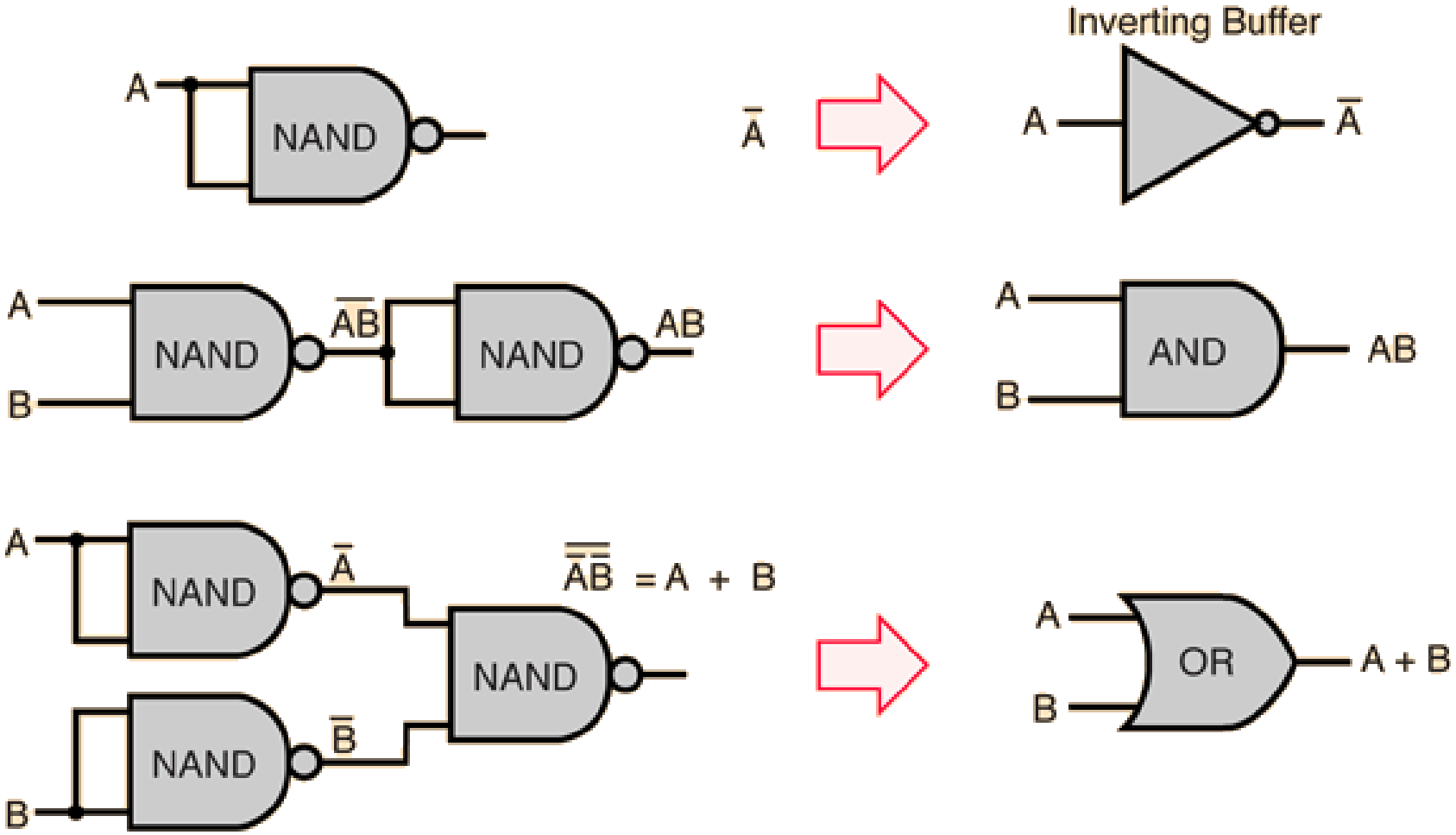
WHY?

T通常采用一个与非门或者一个或非门的时间延迟作为度量单位

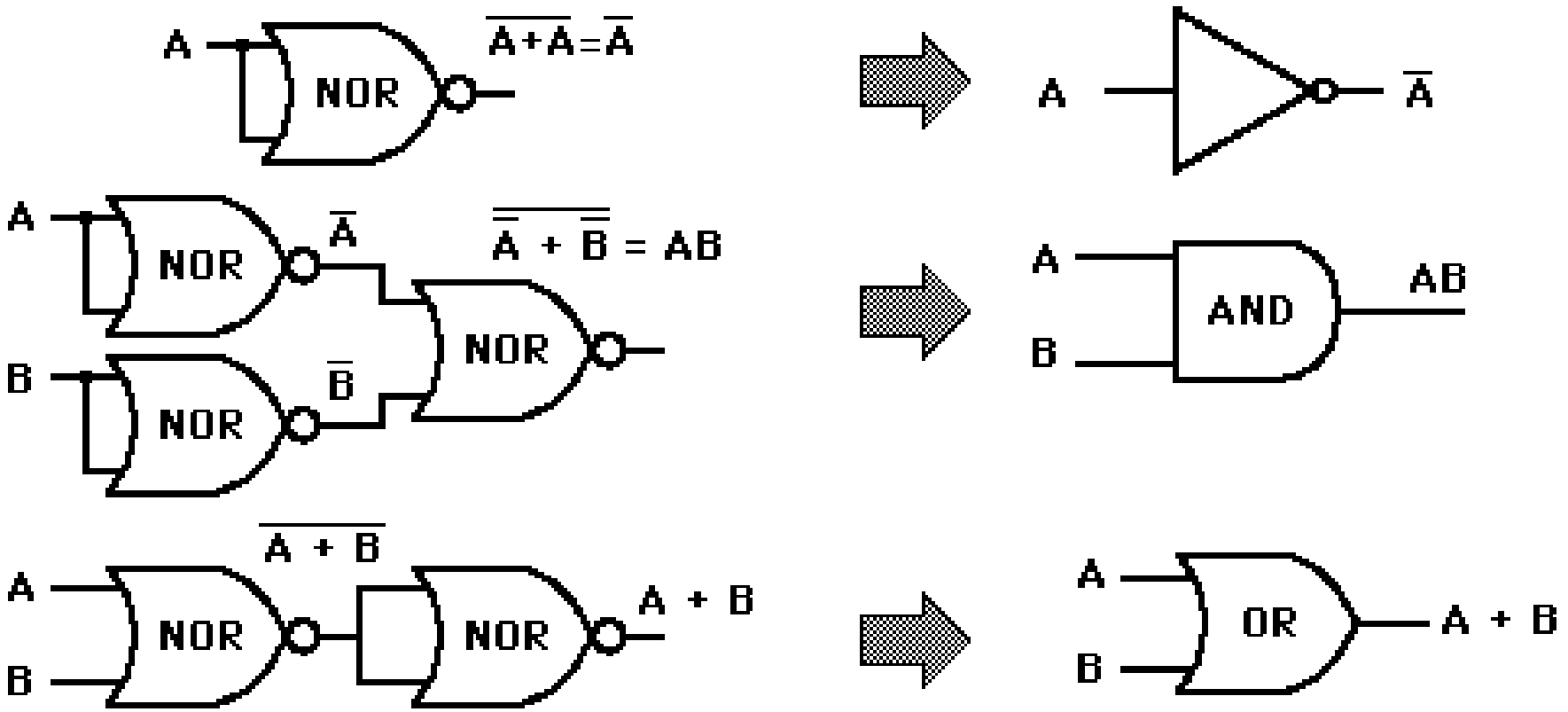
Universal Gates

- The NAND gate（与非门） and the NOR gate（或非门） are universal gates
- Combinations of them can be used to accomplish any of the basic operations and can thus produce an inverter, an OR gate or an AND gate
- The non-inverting gates do not have this versatility since they can't produce an invert.

NAND Gate Operations



NOR Gate Operations

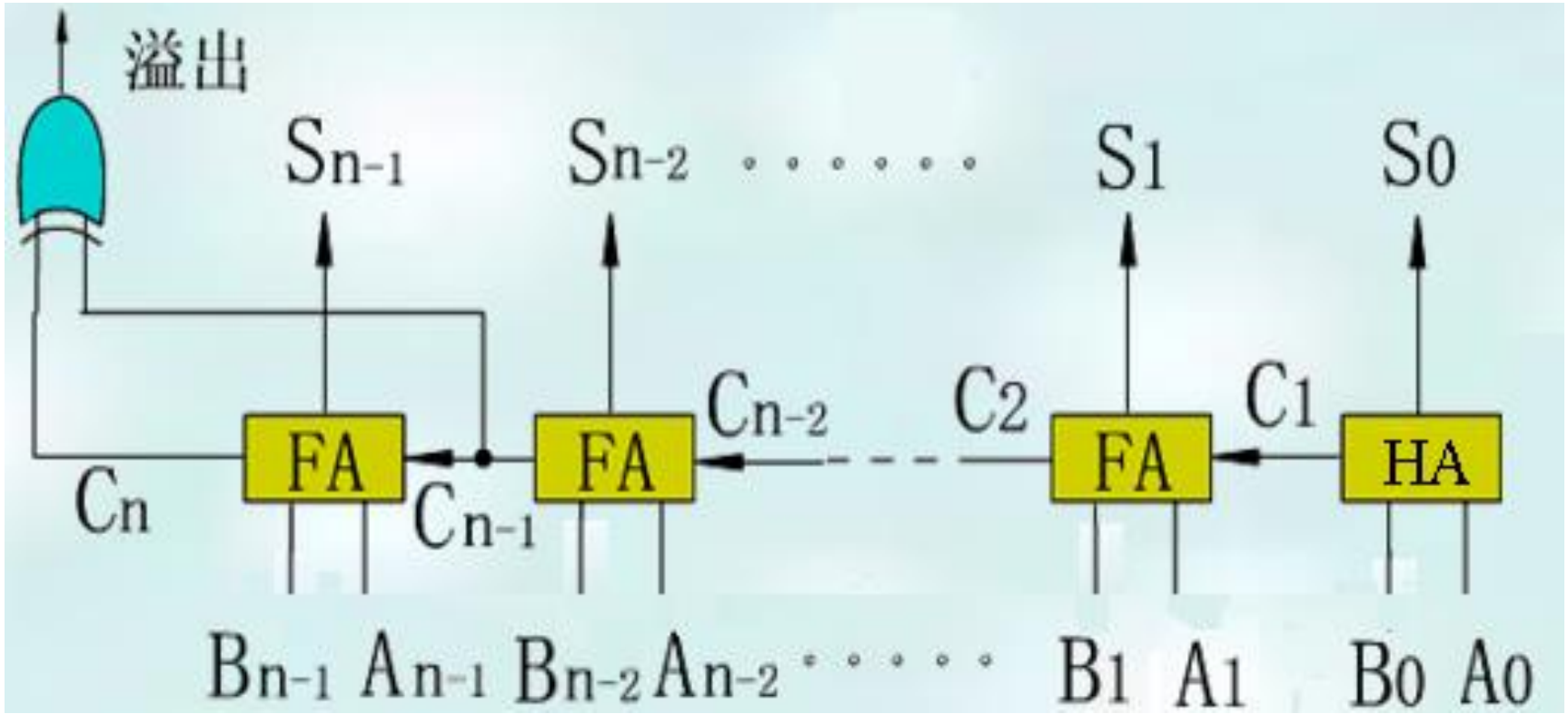


补充内容小结

- **与非门和或非门具有函数完备性**
  - 任何逻辑门都可以用**与非门**的组合或**或非门**的组合实现。
- 一个完整的处理器可以只用**与非门**或**或非门**制作出来。在使用多发射极晶体管的TTL集成电路中，与非门需要的晶体管也少于其他任何门电路
- **典型门电路的时间延迟（了解即可）**

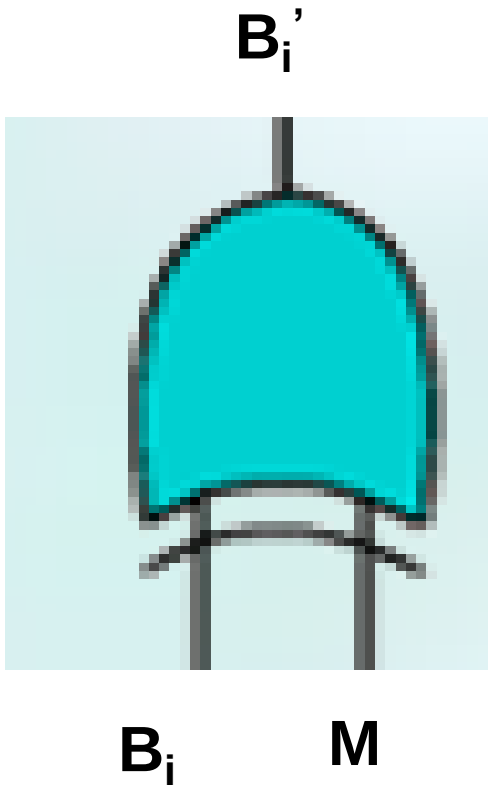
多位二进制数据加法器

- 两个n位的数据$A=A_{n-1}A_{n-2}\cdots A_1A_0$，$B=B_{n-1}B_{n-2}\cdots B_1B_0$
- 和$S=S_{n-1}S_{n-2}\cdots S_1S_0$
- 采用进位判别法判断运算的溢出： $V=C_n\oplus C_{n-1}$

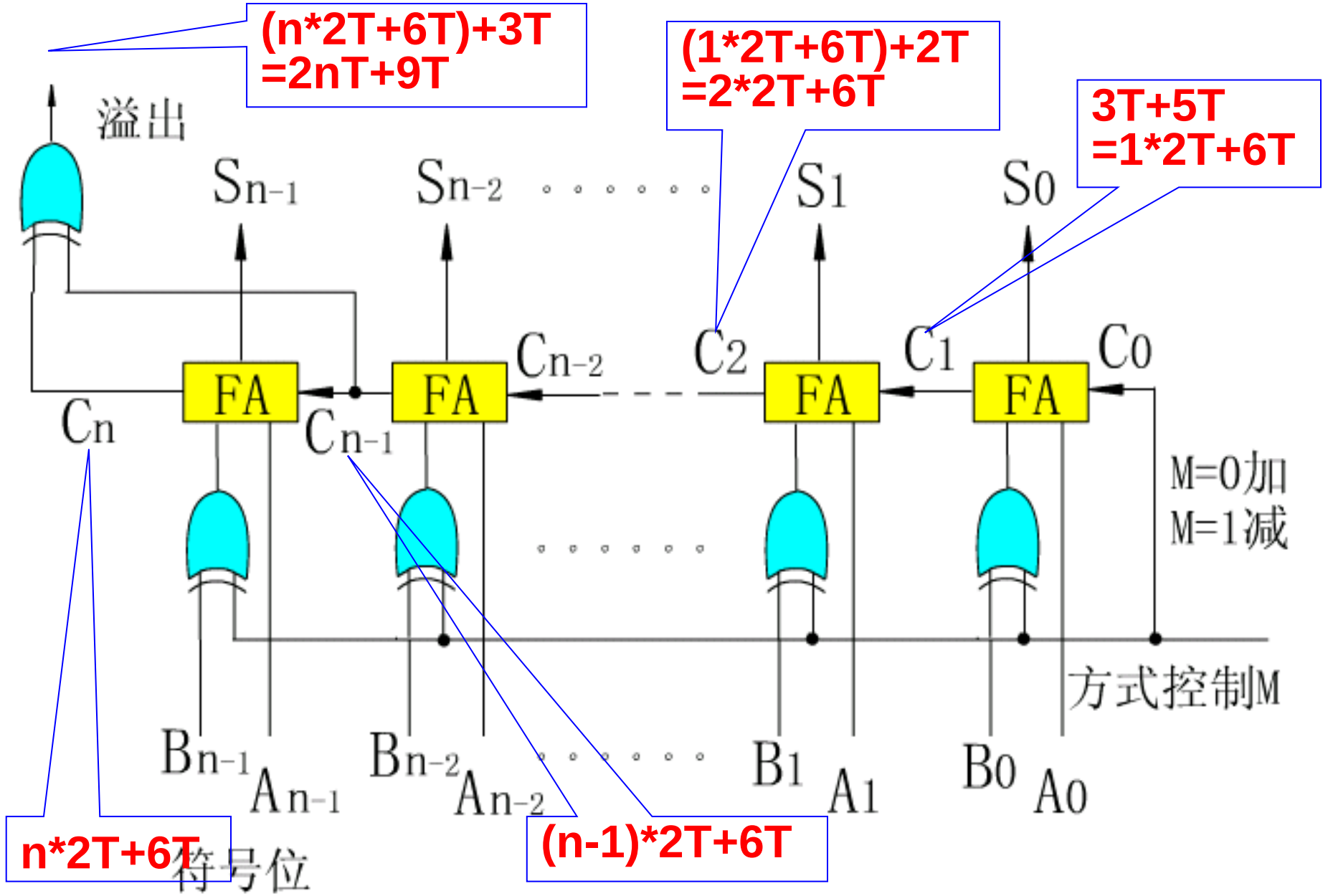


多位二进制数据加法/减法器

- 将减法转换成加法
  - $[A]_{补}-[B]_{补} = [A]_{补}+[-B]_{补}$
- 由$[B]_{补}$ 求$[-B]_{补}$
  - $[B]_{补}$ 求各位取反，末位加1；
- 将加减法电路合二为一
  - 使用异或运算；
  - 当M=0时，$B_i' = B_i$
  - 当M=1时，$B_i' = \neg B_i$ ；



多位二进制数据加法/减法器



多位二进制加法/减法器的输出延迟

- 假如每位均采用一位全加器并考虑溢出检测，n位行波进位加法器的延迟时间t_a为：

  $t_a = n*2T + 9T = (2n+9)T$

- 如果不考虑溢出，则延迟时间t_a由S_{n-1}的输出延迟决定：

  $t_a = (n-1)*2T + 6T + 3T = (2(n-1)+9)T$

- 延迟时间t_a
  - 输入稳定后，**在最坏情况下**加法器得到稳定的输出所需的最长时间
  - 显然这个时间越小越好。

推荐阅读

- The Hidden Language of Computer Hardware and Software， Charles Petzold
  编码的奥秘–隐匿在计算机背后的软硬件语言