第三讲 加法器设计

- 定点加法器设计
  - 进位链结构
  - 串行进位
  - 并行进位
- 浮点加法器设计
  - 规格化浮点数运算的基本原理
  - 浮点加法器设计实现

3.1 定点加法器设计

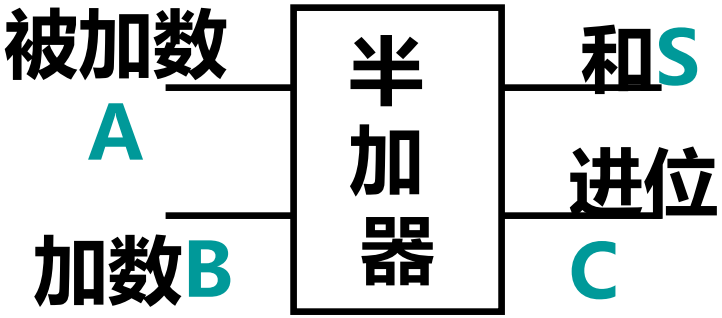
- 算术逻辑部件的核心单元是加法器。加法器是影响算术逻辑部件整体性能的关键部分。
- 定点多位加法器是指能够实现多位二进制数相加运算的电路。

	A	:	1	1	0	1 ← 被加数
+	B	:	1	0	1	1 ← 加数
			1	1	1	0 ← 低位进位
			1	1	0	0 0

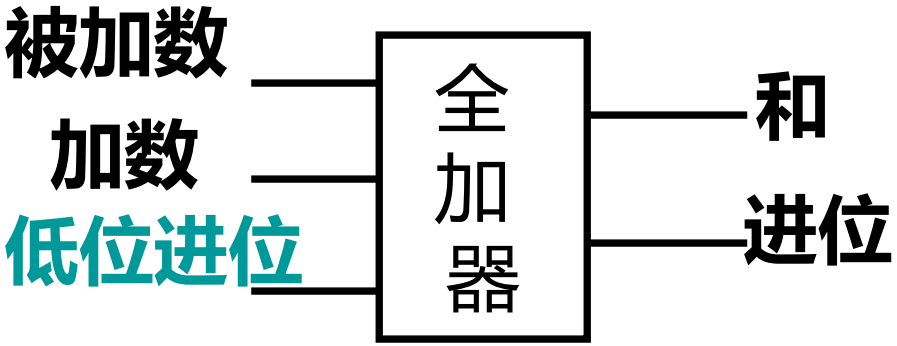
进位C　　和S

3.1 定点加法器设计

- 一位半加器 ---不考虑低位进位的一位加法器



- 一位全加器：---考虑低位进位的一位加法器



3.1.1 进位链结构

按形成进位的方式可以将多位加法器分为两类：

- **串行进位加法器**

  串行进位方式是将多个全加器的进位输出依次级联。

- **并行进位加法器**

  并行进位加法器设有专门的并行进位产生逻辑，运算速度较快。

3.1.1 进位链结构

串行进位加法器

- 每步操作只实现一位求和。
- 采用一位加法器设计n位全加器，则需将n位二进制求和运算分解为n步操作实现，每位的进位作为下一步求和操作的进位输入。
- 串行加法器所用元件很少，但速度太慢。

3.1.1 进位链结构

并行进位加法器

- 使用n个全加器一步实现n位相加，即n位数据同时求和。
- 计算机的运算器基本上都采用并行加法器，所用全加器的个数与操作位数相同。
- 并行加法器的运算速度不仅与全加器的速度有关，更取决于进位传递的速度。

3.1.1 进位链结构

- 从本质上来讲，进位的产生是从低位开始，逐级向高位传递的。
- 假定Cin为低位进位信号，则本位（第i位）产生的进位信号Cout为：

$$C_{out}=A_i B_i + C_{in}(A_i \oplus B_i)$$

$$C_{out}=A_i B_i + C_{in}(\overline{A_i} \oplus \overline{B_i})$$

$$C_{out}=A_i B_i + C_{in}(A_i + B_i)$$

3.1.1 进位链结构

- $Cout=G_i+P_iC_{in}$是构成各种进位链结构的基本逻辑式。
- $G_i=A_iB_i$称为第i位的进位产生函数，或称为本位进位或绝对进位。
- 若本位的两输入量均为1，必产生进位。这是不受进位传递影响的分量。

3.1.1 进位链结构

- Pi称为进位传递函数，而PiCin则称为传送进位或条件进位。
- Pi的逻辑含义是：若本位的两个输入至少一个为1时，则当低位有进位传来时，本位将产生进位。

3.1.2 串行进位

- 串行进位方式是指：逐级地形成各位进位，每一级进位直接依赖于上一级进位。
- 设n位并行进位加法器的序号是第一位为最低位，第n位为最高位，则各进位信号的逻辑式如下：

$$C1=G1+P1C0=A1B1+(A1B1)C0$$

$$C2=G2+P2C2=A2B2+(A2B2)C1$$

.

.

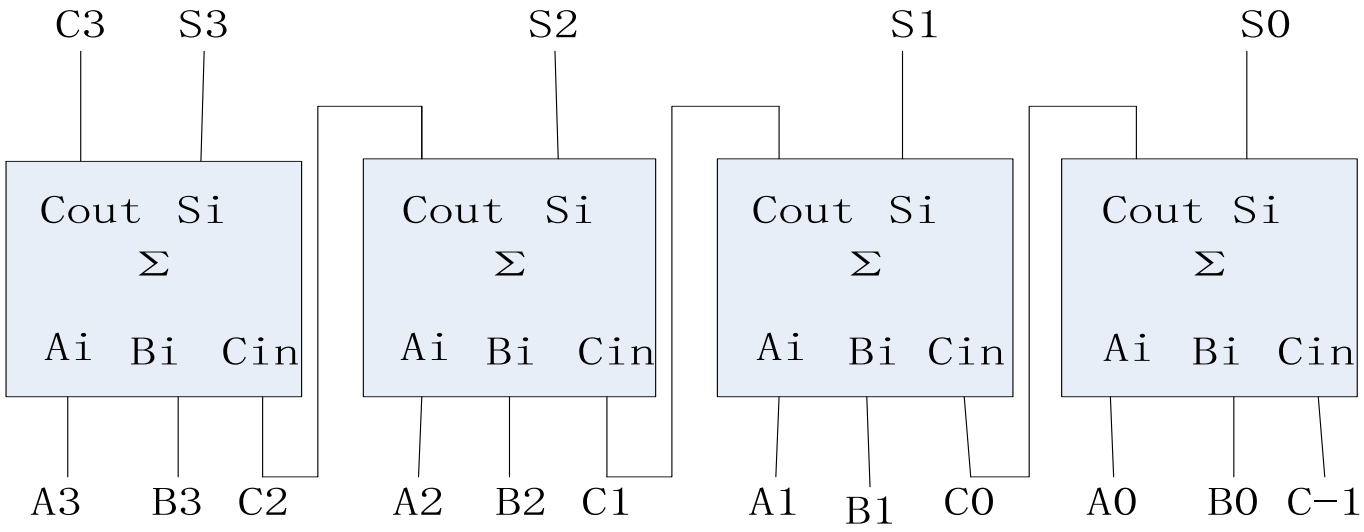
.

$$Cn=Gn+PnCn=AnBn+(AnBn)Cn-1$$

3.1.2 串行进位

- 两个多位数相加时，只要将低位全加器的进位输出端接到高位全加器的进位输入端，就可以构成串行进位加法器。
- 任一位的加法运算必须在低一位的加法运算完成之后才能进行。
- 在各级全加器之间，进位信号采用串联结构，所用元件最少，逻辑电路比较简单，但运算时间比较长。

3.1.2 串行进位



串行进位加法器

3.1.2 串行进位

- 可以通过使用1位全加器的串联行成多位串行进位加法器。
- 要实现8位串行进位加法器，只需要首先1位全加器模块，然后在顶层模块中对该1位全加器实例化，通过串联的方式产生8位全加器的各位输出。

3.1.2 串行进位

- 首先实现一个加法器模块

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity FAdder1 is
    Port ( p : in  STD_LOGIC;
           q : in  STD_LOGIC;
           c_in : in  STD_LOGIC;
           s : out  STD_LOGIC;
           c_out : out  STD_LOGIC);
end FAdder1;

architecture Behavioral of FAdder1 is

begin
s<=(p xor q) xor c_in;
c_out<=(p and q) or (c_in and p) or (c_in and q);
end Behavioral;
```

3.1.2 串行进位

实现8位全加器时，只要在顶层模块进行相应位的映射即可实现。

```
u0:FAdder1 port map(a(0),b(0),cin,sum(0),c(1));
u1:FAdder1 port map(a(1),b(1),c(1),sum(1),c(2));
u2:FAdder1 port map(a(2),b(2),c(2),sum(2),c(3));
u3:FAdder1 port map(a(3),b(3),c(3),sum(3),c(4));
u4:FAdder1 port map(a(4),b(4),c(4),sum(4),c(5));
u5:FAdder1 port map(a(5),b(5),c(5),sum(5),c(6));
u6:FAdder1 port map(a(6),b(6),c(6),sum(6),c(7));
u7:FAdder1 port map(a(7),b(7),c(7),sum(7),cout);
```

3.1.3并行进位

- 并行加法器又称为超前进位加法器。每位的进位只有加数和被加数决定，而与低位的进位无关，即在加法运算过程中各级进位信号同时送到各个全加器的进位输入端。

3.1.3 并行进位

- 根据进位产生函数Gi=AiBi及进位传递函数，可得到如下逻辑式：

$$C1=G1+P1C0$$

$$C2=G2+P2G1+P2P1P0$$

$$C3=G3+P3G2+P3P2G1+P3P2P1C0$$

.

.

.

$$Cn=Gn+PnGn-1+\ldots+(Pn\ldots P1)C0$$

3.1.3 并行进位

- 在并行进位结构中，各进位结构是独自形成的，并不直接依赖于前级。当加法器运算的有关输入（AiBiC0）稳定后，各级同时产生自己的Gi和Pi，也同时形成自己的进位信号Ci。

3.1.3 并行进位

- 4位并行进位加法器的设计采用数据流方式进行描述。其中，P表示进位传递信号，如果P为0，就否决前一级的进位输入，G表示绝对进位信号，如果g为1，表示一定会向后一级产生进位输出。
- pp信号和gg信号用于多个超前进位模块之间的连接，例如利用4个4位超前进位加法器模块构成16位超前进位加法器。

3.1.3 并行进位

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ParAdder is
    Port ( a : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
           b : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
           cin : in  STD_LOGIC;
           s : out  STD_LOGIC_VECTOR(3 DOWNTO 0);
           cout : out  STD_LOGIC;
           pp : out  STD_LOGIC;
           gg : out  STD_LOGIC);
end ParAdder;
architecture Behavioral of ParAdder is
signal c: std_logic_vector(3 downto 1);
signal p,g:std_logic_vector(3 downto 0);
begin
--绝对进位
   g(0)<=a(0) and b(0);
   g(1)<=a(1) and b(1);
   g(2)<=a(2) and b(2);
   g(3)<=a(3) and b(3);
--进位传递条件
   p(0)<=a(0) xor b(0);
   p(1)<=a(1) xor b(1);
   p(2)<=a(2) xor b(2);
   p(3)<=a(3) xor b(3);
--产生并行进位
   c(1)<=g(0) or (p(0) and cin);
   c(2)<=g(1) or (p(1) and g(0) ) or (p(1)and p(0) and cin);
   c(3)<=g(2) or (p(2) and g(1) ) or (p(2)and p(1) and g(0)) or (p(2)and p(1)and p(0)and cin);
   cout<=g(3)or(p(3) and g(2) )or(p(3)and p(2) and g(1))or(p(3)and p(2)
         and p(1) and g(0)) or(p(3)and p(2)and p(1)and p(0) and cin );
--本组进位传递条件
   pp<=p(0) and p(1) and p(2) and p(3);
   gg<=g(3) or (p(3) and g(2)) or (p(3)and p(2) and g(1)) or (p(3)and p(2) and p(1) and g(0));
--和输出
   s(0)<=p(0) xor cin;
   s(1)<=p(1) xor c(1);
   s(2)<=p(2) xor c(2);
   s(3)<=p(3) xor c(3);
end Behavioral;
```

3.2 浮点加法器

- 浮点数比定点数的表示范围宽，有效精度高，更适合于科学与工程计算的需要。
- 浮点数由阶码E和尾数M组成，其数值为：

$(-1)^{Ms} \times M \times B^{E}$

3.2.1 规格化浮点数加减运算基本原理

- 浮点数$X=Mx\cdot 2^{Ex}$ ± $Y=My\cdot 2^{Ey}$

（1）对阶

（2）尾数进行加（减）运算

（3）规格化

（4）舍入处理

3.2.1 规格化浮点数加减运算基本原理

- 对阶的原则：小阶对大阶。
- 当调整阶码时，尾数应同步地移位，以保证浮点数的值不变。如果阶码以2为低，则每当阶码增1时，尾数应右移一位。

3.2.1 规格化浮点数加减运算基本原理

- 规格化
- 1）左规
- 运算结果为11.1XXX或00.0XXX，尾数左移1位，阶码减1。
- 2）右规
- 运算结果为10.XXX或01.XXX，尾数右移1位，阶码加1。最多右移1次。

3.2.2 浮点加法器的设计

- 数据格式

S（1 b）	Exponent(8 b)	Mantissa（23b）

- 数据共32位，S(1b)为符号位，表示浮点数的正负，Exponent(8b)为阶码，Mantissa(23b)为尾数。

➢ 阶码采用移码表示

[E]阶=E+128

➢ 尾数采用2的补码表示形式

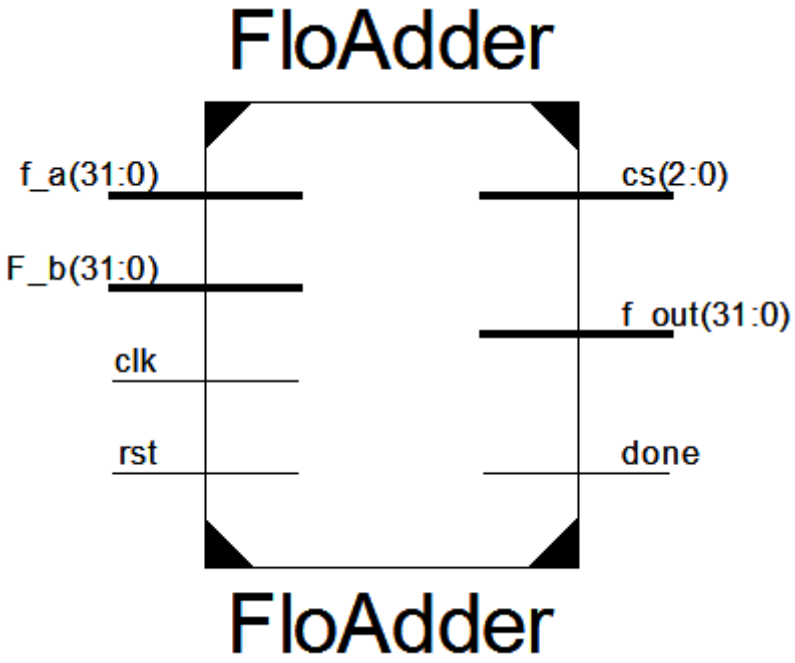
[M]补=2+M，

3.2.2 浮点加法器的设计

符号位在最前面(S)，最后的23位均为数值部分。本节设计的浮点加法器尾数采用补码表示，可以简化设计，而不必判断两数的绝对值大小关系。

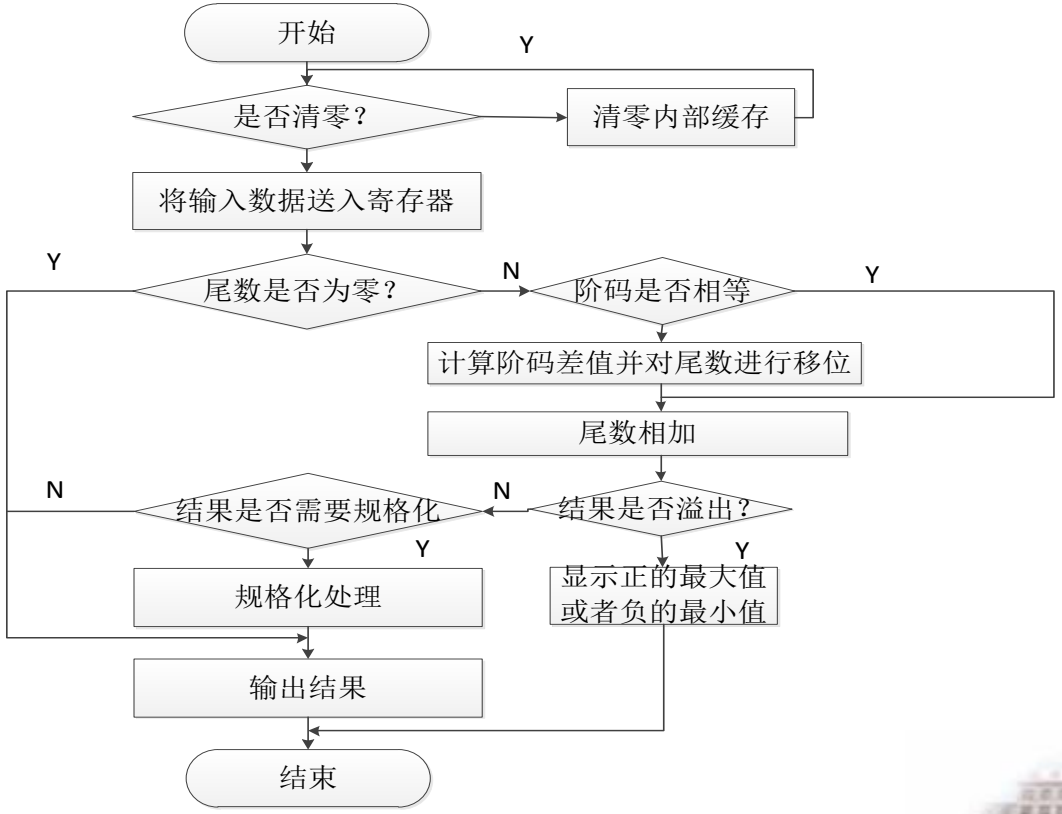
3.2.2 浮点加法器的设计

- 输入输出端口定义



3.2.2 浮点加法器的设计

- 浮点加法器的工作流程



3.2.2 浮点加法器的设计

浮点加法器的工作流程可以用状态描述。设计7个状态（读者也可自行根据流程图定义状态机，状态数可以多余或少于7个），分别表示运算过程的各个步骤，各状态的含义如表所示。

3.2.2 浮点加法器的设计

状态编码	执行的操作
S0：4’0000	初始化
S1：4’0001	检测操作数是否是零
S2：4’0010	比较阶码并计算阶码的差值
S3：4’0011	阶码小的尾数右移并修改阶码
S4：4’0100	尾数求和
S5：4’0101	判断结果是否溢出以及是否需要规格化
S6：4’0110	对结果进行规格化

3.2.2 浮点加法器的设计

浮点加法器的状态转换图