9 浮点数算法和Verilog中的ALU设计

英语

1. underbars下划线
2. significand有效值
3. sticky粘性
4. *even*偶数
1. guard保护
2. mantissa尾数
3. *statisfy*满足
4. *odd*奇数

9.1 IEEE 754浮点数格式

IEEE 754标准定义了两种浮点数格式：32位的单精度（1,8,23）和64(1,11,52)位的双精度，其格式如下：

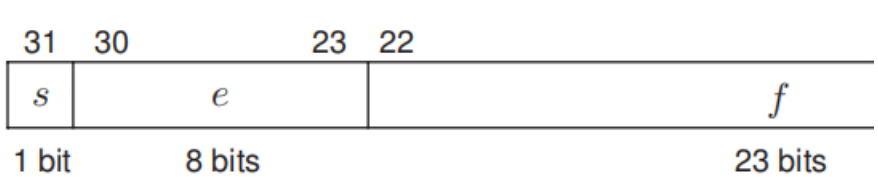

Figure 9.1 IEEE 754 format of a single-precision float

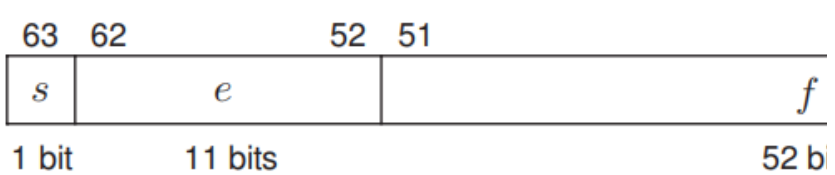

Figure 9.2 IEEE 754 format of a double-precisior

s位表示符号位，1为正0为负

e位表示阶码，采用偏置码[注释1]表示

f位表示尾数位

📌按照e的大小，我们将其分为以下几种情况

- 浮点数的值分类
  - 阶码值为0
    - 非正则化浮点数
      - 阶码值为0但尾数不为0
      - 尾数隐含位为0
      - 无穷小视为0
    - 0
      - 阶码值为0，尾数值也为0

        - 隐含位为0
    - 正则化浮点数
        - 阶码值在1~254/1~2046之间
        - 尾数隐含位为1
    - 阶码值为255/2047
        - 无穷
            - 阶码值为255/1047，尾数位全0
            - 隐含位为1
        - Nan
            - 阶码值为255/1047，尾数位不全为0
            - 隐含位为1

- Normalized: If $0 < e < 255$, then $V = (-1)^s \times 1.f \times 2^{e-127}$;
- $+0, -0$: If $e = 0$ and $f = 0$, then $V = (-1)^s \times 0$;
- Denormalized: If $e = 0$ and $f \neq 0$, then $V = (-1)^s \times 0.f \times 2^{-126}$;
- $+\infty, -\infty$: If $e = 255$ and $f = 0$, then $V = (-1)^s \infty$; and
- NaN (not a number): If $e = 255$ and $f \neq 0$, then $V =$ NaN.

- Normalized: If $0 < e < 2047$, then $V = (-1)^s \times 1.f \times 2^{e-1023}$;
- $+0, -0$: If $e = 0$ and $f = 0$, then $V = (-1)^s \times 0$;
- Denormalized: If $e = 0$ and $f \neq 0$, then $V = (-1)^s \times 0.f \times 2^{-1022}$;
- $+\infty, -\infty$: If $e = 2047$ and $f = 0$, then $V = (-1)^s \infty$; and
- NaN: If $e = 2047$ and $f \neq 0$, then $V =$ NaN.

9.2浮点数和整数之间的转换

9.1浮点数→整数

32位整数的范围是$-2^{31} \leq d \leq +2^{31} - 1$，这意味着有很多的浮点数不能转换为整数invalid为1

符号位s为0时，e≤157,f任意，即0_10011101_11111111111111111111111

符号位s为1时，e≤158,f有条件的任意(e=158时f必须全0，其余阶码任意),即1_11111111_00000000000000000000000

浮点数转换为整数的算法是：

1. 分别得到符号位s、阶码e、尾数s
2. 根据e和s先判断是否该数能够正确转换为浮点数，若不能则置其值为最小负数0x8000_0000
3. 根据e和s划分浮点数值的类别，并根据类别做处理

非正则数视为0，且有精度损失

若e-127小于等于0，那么就赋值0，并根据s判断精度是否有损失

此外，右移阶码位小数得到最终整数值，精度损失取决于移位到的小数点后面的值

```
`timescale 1ns / 1ps

module float2int(
    input [31:0]a,
    output reg [31:0]d,
    output reg p_lost,//精度损失
    output reg invalid,//超出整数表示范围
    output reg denorm//非正则数
);
    wire sign=a[31];
    wire [7:0]e=a[30:23];
    wire [54:0]s={|e,a[22:0],31'b0};
    reg [54:0]t;
    integer i;
    always @(a) begin//注意这里是跟a变
        i=158;//注意i的初始化
        if (e>=8'd158) begin
            p_lost=0;denorm=0;invalid=1;
            d=32'h8000_0000;
            if ((e==8'd158)&(sign==1)&(s[53:31]==23'b0)) begin
                invalid=0;
            end
        end else if (e<8'd127) begin//视作0
            d=32'b0;
            p_lost=|s;denorm=0;invalid=0;
            if (e==8'd0) begin//非正则、0
```

```
                if (s[54:31]==32'b0) begin
                    p_lost=0;
                end  else begin
                    denorm=1;
                end
            end
        end else begin
            i=i-e;
            t=s>>i;
            d=sign?~t[54:23]+32'b1:t[54:23];//负数的处理
            p_lost=|t[22:0];invalid=0;denorm=0;
        end
    end
endmodule
```

Verilog

9.2整数→浮点数

所有的整数都可以转变为浮点数，但是因为浮点数的精度只有23位，所以存在失损精度的问题

转换的方法如下：

1. 记录符号sign，并将原数换为绝对值——$-x = barx + 1$
2. 小数点本来在0位后，这里设置为31位后——已移位31次，c=31
3. 若[31]为0，那么左移数，直至[31]=1,每左移一次，c-1
4. 最后得到s=[30:8]、sign=原[31]、e=c+127、plost=|[7:0]

```
`timescale 1ns / 1ps

module int2float(
    input [31:0]a,
    output reg[31:0]f,
    output reg p_lost
);
    integer i;
```

```
    reg [31:0]temp;
    reg sign;
    always @(a) begin
        temp=a;sign=0;i=31;//注意i每次都要设初值
        if (a[31]==1) begin
            sign=1;
            temp=~a+1;//负数处理换成绝对值
        end
        while (~temp[31]) begin
            temp=temp<<1;
            i=i-1;
        end
        f[31]=sign;
        f[30:23]=i+127;
        f[22:0]=temp[30:8];
        p_lost=|temp[7:0];
    end
endmodule
```

Verilog

9.3浮点数加法器设计FADD

9.3.1浮点数加法算法

浮点数加法的计算分为三步：对阶、计算、正则化

9.3.1.1对阶——小向大看齐

首先在计算过程中，可能会有a、b符号不同的情况，如果这时执行的是加法，那么实际执行的是减法——绝对值大的减去绝对值小的。因此需要判断a、b[30:0]的大小，若b大则调换顺序

$$exchange = a[30:0] > b[30:0] ? 0 : 1;$$

然后进行对阶，小的数阶向大的数阶看齐，如果是规格化数则移动差值位，否则移动差值-1位[注释2]

> *a and b in the example above are two normalized float numbers. Although we calculate a + b, the signs of a and b are different, so we must perform subtraction on their absolutes. By checking their exponents, we know that the absolute of b is larger than that of a, so we will perform |b| − |a|. The sign of s is negative, the same as the number that has a larger absolute. And the exponent of s is 125 temporarily.*

同时，**为了得到更准确的结果**，将大数和小数后缀均增加3位，大数的LSB为000，小数的LSB为移出的保护位g、四舍五入位r，以及表示后续不全为0的粘性位s——**也因为使用了grs，那么小数右移不会右移超过26位**

$$a = 2^{120-127} \times 1.11000000000000000010001$$
$$= 2^{125-127} \times 0.00001110000000000000000\_10001\ 101$$

9.3.1.2计算

计算时**增加一位符号位**——因为1.f+1.f有可能会大于2但是不会大于4

		grs	
	01.00000100000000000000000	000	(significand of b)
−	00.00001110000000000000000	101	(significand of a)
	00.11110101111111111111111	011	(significand of s)

9.3.1.3正则化

计算得到的“00.11110101111111111111111001”需要变为“1.f”的格式

对于“0x.xxxxx”形式的结果：则不断左移减阶码，使得出现“1.f”的格式

对于“1x.xxxxx”形式的结果：则只需要右移一位，阶码+1，即可出现“1.f”的格式

对于最后的grs位需要采用舍入的方法：

1. 最近舍入：若x grs距离x更近，那么就去掉grs，否则距离x+1更近，那么就进1
   grs为100时，若x=1则进1否则进0
2. 截断：直接截取grs
3. 向正无穷舍入：若为正数，则+1舍去grs；若为负数则-1舍去grs
4. 向负无穷舍入：若为正数，则-1舍去grs；若为负数则+1舍去grs

9.3.1.4特殊情况

当a和b中有一个是Nan的话，那么结果也是Nan

此外：$a + (\pm\infty) = \pm\infty, a - (\pm\infty) = \mp\infty, (\pm\infty) \pm b = \pm\infty$

Table 9.2 Results of operations on two infinity numbers

sub	a	b	s	Comment
0	$+\infty$	$+\infty$	$+\infty$	$(+\infty)+(+\infty)$
0	$-\infty$	$-\infty$	$-\infty$	$(-\infty)+(-\infty)$
1	$+\infty$	$-\infty$	$+\infty$	$(+\infty)-(-\infty)$
1	$-\infty$	$+\infty$	$-\infty$	$(-\infty)-(+\infty)$
0	$+\infty$	$-\infty$	NaN	$(+\infty)+(-\infty)$
0	$-\infty$	$+\infty$	NaN	$(-\infty)+(+\infty)$
1	$+\infty$	$+\infty$	NaN	$(+\infty)-(+\infty)$
1	$-\infty$	$-\infty$	NaN	$(-\infty)-(-\infty)$

9.3.1.5加法器的实现细节

1. 小的数右移不会超过26位——grs3+尾数23
2. 舍入的方法
   1. 最近舍入：结合最后的第23位和grs一块判断——注意grs=100时结合最末位一块判断

      若grs>100，那么需要尾数+1；若grs<100，那么尾数不变

      若grs=100，则结合第23位判断：若第23位为1则尾数+1，若为0则尾数不变
   2. 截断舍入：直接舍去grs，尾数+0
   3. 向+∞舍入：若为正数且grs≠0，那么尾数+1——grs须不为0
   4. 向－∞舍入：若为负数且grs≠0，那么尾数+1——grs须不为0

   下面是总结的尾数+1的情况：用casex分类讨论

Table 9.4 Fraction increment when rounding

rm[1:0] 舍入方法	Frac	g	r	s	Sign	frac_plus_1	Rounding mode
00	1	1	0	0	x	1	Round to even 看Frac
00	x	1	{not 00}		x	1	Round to nearest
01	x	{not 0 0 0}			1	1	Round toward $-\infty$
10	x	{not 0 0 0}			0	1	Round toward $+\infty$

3. 四舍五入后结果可能仍需要正则化调整

28位小数，若[27]==1则右移一位，指数+1；若[27]==0,[26]==0,不断左移减指数至[26]=1

$$
\begin{array}{ll}
 & 01.11111111111111111111111\ (1.f) \\
+ & 00.00000000000000000000001\ (\texttt{frac\_plus\_1}) \\
\hline
= & 10.00000000000000000000000 \\
\rightarrow & 01.00000000000000000000000\ (1.f)
\end{array}
$$

指数+1

4. 溢出的判断——正则化结果的指数位是255/2047

使用四舍五入到最近的方法——最终的结果凭正则化结果的符号变为±无穷

使用截断法——最终的结果凭正则化结果的符号变为±的正则化数值

使用四舍五入到－∞：正数到最大的正的正则化数值，负数到－∞

使用四舍五入到＋∞：正数到＋∞，负数到负最大的正则化数值

9.3.2 FADD实现

浮点加法器的结构如下所示，分为对阶、计算和规格化三部分

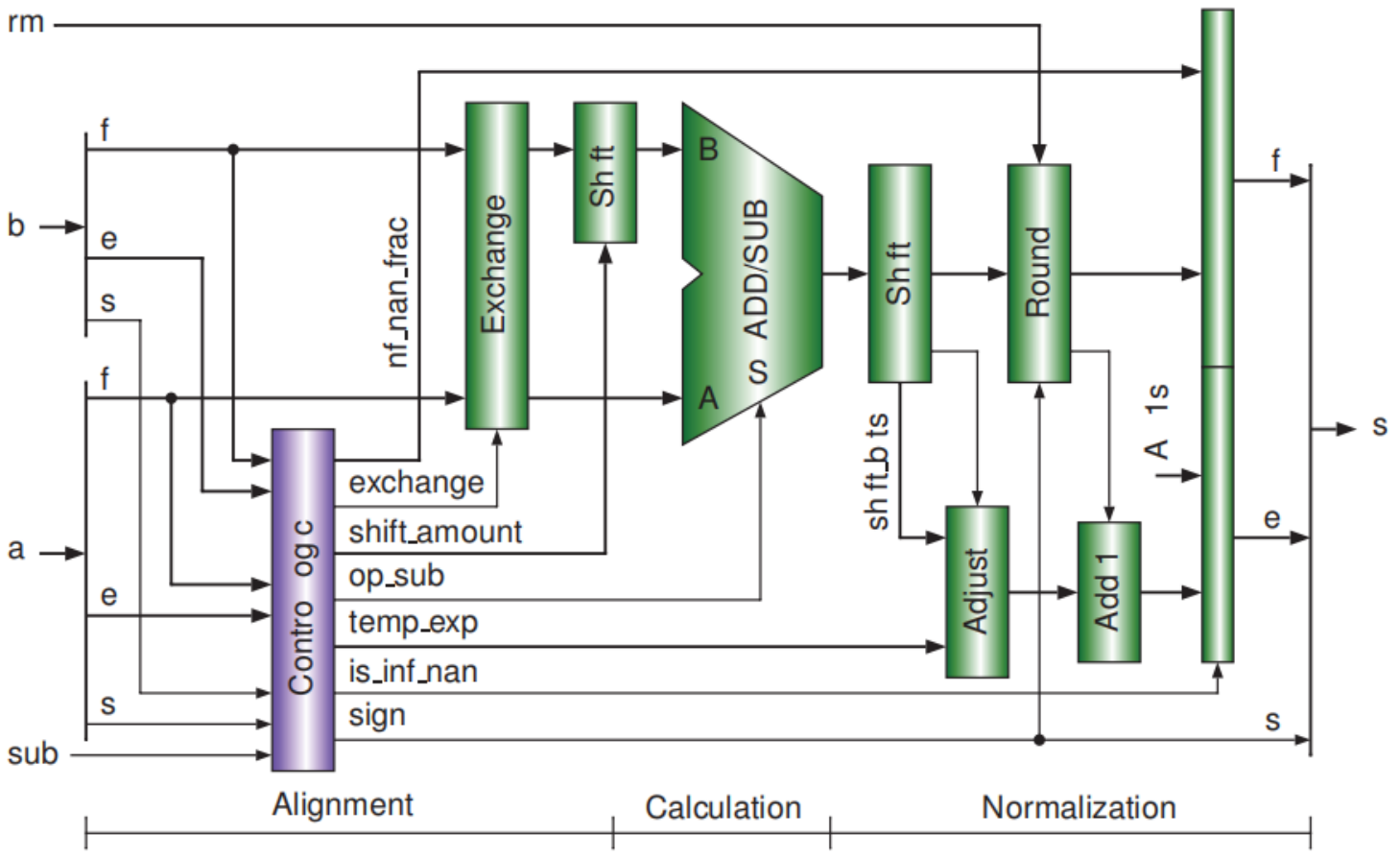

Figure 9.8 Block diagram of a floating-point adder

```
module fadder (a,b,sub,rm,s);
    input [31:0] a,b; // fp inputs a and b
    input sub;        // 1: sub; 0: add
    input [1:0]  rm;  // round mode
    output [31:0] s;  // fp output
```

模块输入是加数a、b，运算方式sub，舍入方式rm；模块输出是s

当sub=1时，a-b；否则a+b

9.3.2.1对阶+交换

1. 首先整体判断大小（结合阶码位数一块判断），并确定是否需要交换a，b

```
//判断是否需要交换
wire exchange=(a[30:0]<b[30:0])?1:0;
wire [31:0]fp_large=exchange?b:a;
wire [31:0]fp_small=exchange?a:b;
```

Verilog

2. 然后再确定两个浮点数的隐含位——若阶码非全0则隐含位为1，否则为0

```
//判断隐含位
wire fp_large_hid=|fp_large[30:23];
wire fp_small_hid=|fp_small[30:23];
```

Verilog

3. 然后确定两个浮点数的尾数"1.f"/"x.f"，结果的符号以及实际进行的运算，暂时确定结果的指数

两个浮点数的尾数即{隐含位，[22:0]}

结果的符号：若没有交换，那么结果符号和a相同；

若交换了，那么执行加法结果符号和大数符号相同，执行减法a-b结果和b的符号相反

实际进行的运算：当a、b符号相同sub=1时，当a，b两符号相反sub=0时执行减法，其余均加法

结果的暂时指数为大数的指数

a	b	sub	op_sub
0	0	0	0
0	0	1	1

0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

```
//得到尾数、判断结果的符号，阶码，实际进行的运算
wire [23:0]fp_large_frac={fp_large_hid,fp_large[22:0]};
wire [23:0]fp_small_frac={fp_small_hid,fp_small[22:0]};
wire sign=exchange?sub^b[31]:a[31];
wire [7:0]temp_e=fp_large[30:23];
wire op_sub=sub^fp_large[31]^fp_small[31];
```

Verilog

4. 确定a、b中是否有Nan，或者是否有无穷

Nan：阶码为全f，尾数不全为0

无穷：阶码为全f，尾数全为0

当a、b中有nan或者是正无穷加负无穷或者正无穷-正无穷或者负无穷-负无穷即结果为NAN，其余为无穷。结果是无穷时，尾数全0；结果是Nan时，尾数选较大的

```
//判断是否有inf、nan
 wire fp_large_e1=&fp_large[30:23];
 wire fp_small_e1=&fp_small[30:23];
 wire fp_large_frac0=~|fp_large[22:0];
 wire fp_small_frac0=~|fp_small[22:0];
 wire fp_large_isnan=fp_large_e1&~fp_large_frac0;
 wire fp_small_isnan=fp_small_e1&~fp_small_frac0;
 wire fp_large_isinf=fp_large_e1&fp_large_frac0;
 wire fp_small_isinf=fp_small_e1&fp_small_frac0;
 wire isNan=fp_large_isnan|fp_small_isnan|(fp_large_isinf&fp_small_isinf&op_sub);
```

```
    wire isInf=(fp_large_isinf|fp_small_isinf)&~isNan;
    wire nan_frac=(a[22:0]>b[22:0])?{1'b1,a[22:0]}:{1'b1,b[22:0]};
    wire inf_frac=23'b0;
```

Verilog

5. 对阶

注意如果a，b均是规格化数，那么移位量是阶码差值；如果小数不是规格化数，那么右移位数是阶码差值-1，因为**非规格化数的阶码是-126，不是-127**

fp_small 是非规格化数，则右移位数等于 exp_diff − 1。这是因为规格化数的绝对值等于 $2^{e-127} \times 1.f$，而非规格化数的绝对值等于 $2^{0-126} \times 0.f$。图 9.7 说明当右移位数大

```
   wire [7:0]exp_diff=fp_large[30:23]-fp_small[30:23];
   wire small_den_only=(fp_large[30:23]!=0)&(fp_small[30:23]==0);//小数是非规格化数
   wire [7:0]shift_amount=small_den_only?exp_diff-8'h1:exp_diff;
   wire
[49:0]small_frac50=(shift_amount>26)?{26'h0,fp_small_frac}:{fp_small_frac,26'h0}>>shift_amount;
   wire[26:0]small_frac27={small_frac50[49:24],|small_frac50[23:0]};//24位尾数+3位grs x.xxxx xxx
   wire [27:0]aligned_large_frac={1'b0,fp_large_frac,3'b000};//加一个符号位 28位
   wire [27:0]aligned_small_frac={1'b0,small_frac27};
```

Verilog

9.3.2.2计算

```
wire [27:0]cal_frac=op_sub?aligned_large_frac-
aligned_small_frac:aligned_large_frac+aligned_small_frac;
```

Verilog

9.3.2.3规格化

当结果尾数cal_frac是"1x.xxxx"格式时：右移一位尾数，并使阶码+1

当结果尾数cal_frac是"0x.xxxx"格式时：不断左移使得cal_frac[26]=1,每移动一次阶码-1

四舍五入：使用casex根据rm[2:0]，cal_frac[3]和cal_frac[2:0]得到进位

然后再进行规格化

然后判断溢出casex判断

```
`timescale 1ns / 1ps

module fadd_design(
    input [31:0]a,b,
    input [1:0]rm,
    input sub,
    output reg [31:0]s
);
    //判断是否需要交换
    wire exchange=(a[30:0]<b[30:0])?1:0;
    wire [31:0]fp_large=exchange?b:a;
    wire [31:0]fp_small=exchange?a:b;
    //判断隐含位
    wire fp_large_hid=|fp_large[30:23];
    wire fp_small_hid=|fp_small[30:23];
    //得到尾数、判断结果的符号，阶码，实际进行的运算
    wire [23:0]fp_large_frac={fp_large_hid,fp_large[22:0]};
    wire [23:0]fp_small_frac={fp_small_hid,fp_small[22:0]};
    wire sign=exchange?sub^b[31]:a[31];
    wire [7:0]temp_e=fp_large[30:23];
    wire op_sub=sub^fp_large[31]^fp_small[31];
    //判断是否有inf、nan
    wire fp_large_e1=&fp_large[30:23];
    wire fp_small_e1=&fp_small[30:23];
    wire fp_large_frac0=~|fp_large[22:0];
    wire fp_small_frac0=~|fp_small[22:0];
    wire fp_large_isnan=fp_large_e1&~fp_large_frac0;
    wire fp_small_isnan=fp_small_e1&~fp_small_frac0;
```

```
    wire fp_large_isinf=fp_large_e1&fp_large_frac0;
    wire fp_small_isinf=fp_small_e1&fp_small_frac0;
    wire isNan=fp_large_isnan|fp_small_isnan|(fp_large_isinf&fp_small_isinf&op_sub);
    wire isInf=(fp_large_isinf|fp_small_isinf)&~isNan;
    wire nan_frac=(a[22:0]>b[22:0])?{1'b1,a[22:0]}:{1'b1,b[22:0]};
    wire inf_frac=23'b0;

    wire [7:0]exp_diff=fp_large[30:23]-fp_small[30:23];
    wire small_den_only=(fp_large[30:23]!=0)&(fp_small[30:23]==0);
    wire [7:0]shift_amount=small_den_only?exp_diff-8'h1:exp_diff;
    wire
[49:0]small_frac50=(shift_amount>26)?{26'h0,fp_small_frac}:{fp_small_frac,26'h0}>>shift_amount;
    wire[26:0]small_frac27={small_frac50[49:24],|small_frac50[23:0]};
    wire [27:0]aligned_large_frac={1'b0,fp_large_frac,3'b000};
    wire [27:0]aligned_small_frac={1'b0,small_frac27};

    //计算
    wire [27:0]cal_frac=op_sub?aligned_large_frac-
aligned_small_frac:aligned_large_frac+aligned_small_frac;
    reg [27:0]temp_frac;
    integer  i;
    //规格化
    always @(a,b,rm,sub) begin
        i=0;
        if (isInf) begin//a和b中有无穷，且a、b均不是nan
            s={fp_large[31],8'hff,inf_frac};//无穷
        end else if (isNan) begin//ab中有nan
            s={fp_large[31],8'hff,nan_frac};//nan
        end else begin
            if (cal_frac[27]==1) begin
                temp_frac=cal_frac>>1;
```

```
            i=i+1;
            i=temp_e+i;
        end else if (cal_frac[26]==0) begin
            temp_frac=cal_frac<<1;//左移1位
            i=i+1;
            while (~temp_frac[26]&&i<27) begin
                temp_frac=temp_frac<<1;
                i=i+1;
            end
            i=(temp_e<i)?0:temp_e-i;//移位超过了现有阶码
        end else begin
            temp_frac=cal_frac;
            i=temp_e;
        end
        casex({rm,temp_frac[3],temp_frac[2:0],sign})
            7'b00_1_100_x:temp_frac=temp_frac+28'b1000;
            7'b00_x_1xx_x:begin
                if (temp_frac[2:0]>3'b100) begin
                    temp_frac=temp_frac+28'b1000;
                end
            end
            7'b01_x_xxx_1:begin
                if (temp_frac[2:0]>3'b000) begin
                    temp_frac=temp_frac+28'b1000;
                end
            end
            7'b10_x_xxx_0:begin
                if (temp_frac[2:0]>3'b000) begin
                    temp_frac=temp_frac+28'b1000;
                end
            end
            default :   temp_frac=temp_frac;
```

```
            endcase
            if (temp_frac[27]==1) begin
                temp_frac=temp_frac>>1;
                i=i+1;
            end else if (temp_frac[26]==0) begin
                while (~temp_frac[26]) begin
                    temp_frac=temp_frac<<1;
                    i=i-1;
                end
            end
            casex({i[7:0]==8'hff,rm,sign})
                4'b0_xx_x:s={sign,i[7:0],temp_frac[25:3]};
                4'b1_00_x:s={sign,i[7:0],23'b0};//正负无穷
                4'b1_01_0:s={sign,8'hfe,23'h7fffff};//向最大正数
                4'b1_01_1:s={sign,8'hff,23'b0};//向负无穷
                4'b1_10_0:s={sign,8'hff,23'b0};//向正无穷
                4'b1_10_1:s={sign,8'hfe,23'h7fffff};//向最小负数
                4'b1_11_x:s={sign,8'hfe,23'h7fffff};//最大正/负数
            endcase
        end
    end
endmodule
```

Verilog

9.3.3 流水线浮点加法器设计

流水线浮点加法器即在“对阶、计算、对齐”中间插入流水线寄存器，用于数据和控制信号的传递，每一阶段使用上一阶段提供的数据、信号进行运算输出并将结果输出给下一级。下图即为流水线浮点加法器的示意图，流水线寄存器的写信号是为了方便CPU用来阻塞

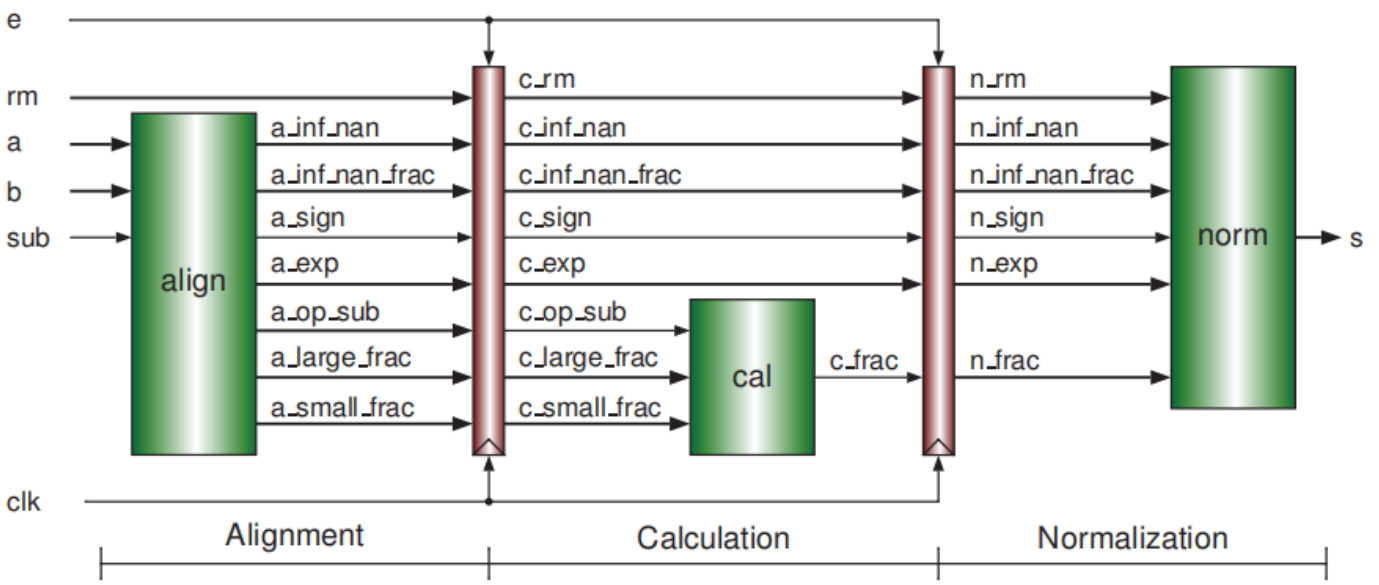

Figure 9.12 Block diagram of a pipelined floating-point adder

对齐阶段代码

```
`timescale 1ns / 1ps

module align_design(
    input [31:0]a,b,
    input [1:0]rm,
    input sub,

    output op_sub,isNan,isInf,sign,
    output [27:0]aligned_large_frac,aligned_small_frac,
    output [23:0]nan_frac,inf_frac,
    output [31:0]fp_large,fp_small,
    output [7:0]temp_e,
    output [1:0]crm
);
    //判断是否需要交换
    wire exchange=(a[30:0]<b[30:0])?1:0;
    assign fp_large=exchange?b:a;
    assign fp_small=exchange?a:b;
```

```
    //判断隐含位
    wire fp_large_hid=|fp_large[30:23];
    wire fp_small_hid=|fp_small[30:23];
    //得到尾数、判断结果的符号，阶码，实际进行的运算
    wire [23:0]fp_large_frac={fp_large_hid,fp_large[22:0]};
    wire [23:0]fp_small_frac={fp_small_hid,fp_small[22:0]};
    assign sign=exchange?sub^b[31]:a[31];
    assign temp_e=fp_large[30:23];
    assign op_sub=sub^fp_large[31]^fp_small[31];
    //判断是否有inf、nan
    wire fp_large_e1=&fp_large[30:23];
    wire fp_small_e1=&fp_small[30:23];
    wire fp_large_frac0=~|fp_large[22:0];
    wire fp_small_frac0=~|fp_small[22:0];
    wire fp_large_isnan=fp_large_e1&~fp_large_frac0;
    wire fp_small_isnan=fp_small_e1&~fp_small_frac0;
    wire fp_large_isinf=fp_large_e1&fp_large_frac0;
    wire fp_small_isinf=fp_small_e1&fp_small_frac0;
    assign isNan=fp_large_isnan|fp_small_isnan|(fp_large_isinf&fp_small_isinf&op_sub);
    assign isInf=(fp_large_isinf|fp_small_isinf)&~isNan;
    assign nan_frac=(a[22:0]>b[22:0])?{1'b1,a[22:0]}:{1'b1,b[22:0]};
    assign inf_frac=23'b0;

    wire [7:0]exp_diff=fp_large[30:23]-fp_small[30:23];
    wire small_den_only=(fp_large[30:23]!=0)&(fp_small[30:23]==0);
    wire [7:0]shift_amount=small_den_only?exp_diff-8'h1:exp_diff;
    wire
[49:0]small_frac50=(shift_amount>26)?{26'h0,fp_small_frac}:{fp_small_frac,26'h0}>>shift_amount;
    wire[26:0]small_frac27={small_frac50[49:24],|small_frac50[23:0]};
    assign aligned_large_frac={1'b0,fp_large_frac,3'b000};
    assign aligned_small_frac={1'b0,small_frac27};
```

```
    assign crm=rm;
endmodule
```

Verilog

计算阶段代码

```
`timescale 1ns / 1ps

module cal_design(
    input [27:0]aligned_large_frac,aligned_small_frac,
    input op_sub,
    output [27:0]cal_frac
);
    assign cal_frac=op_sub?aligned_large_frac-
aligned_small_frac:aligned_large_frac+aligned_small_frac;

endmodule
```

Verilog

规格化阶段代码

```
`timescale 1ns / 1ps

module normal_design(
    input [27:0]cal_frac,
    input isNan,isInf,sign,
    input [23:0]nan_frac,inf_frac,
    input [31:0]fp_large,fp_small,
    input [7:0]temp_e,
    input [1:0]crm,

    output reg[31:0]s
);
```

```
reg [27:0]temp_frac;
integer  i;
//规格化
always @(*) begin
    i=0;
    if (isInf) begin
        s={fp_large[31],8'hff,inf_frac};
    end else if (isNan) begin
        s={fp_large[31],8'hff,nan_frac};
    end else begin
        if (cal_frac[27]==1) begin
            temp_frac=cal_frac>>1;
            i=i+1;
            i=temp_e+i;
        end else if (cal_frac[26]==0) begin
            temp_frac=cal_frac<<1;//左移1位
            i=i+1;
            while (~temp_frac[26]&&i<27) begin
                temp_frac=temp_frac<<1;
                i=i+1;
            end
            i=(temp_e<i)?0:temp_e-i;
        end else begin
            temp_frac=cal_frac;
            i=temp_e;
        end
        casex({rm,temp_frac[3],temp_frac[2:0],sign})
            7'b00_1_100_x:temp_frac=temp_frac+28'b1000;
            7'b00_x_1xx_x:begin
                if (temp_frac[2:0]>3'b100) begin
                    temp_frac=temp_frac+28'b1000;
                end
```

```
                end
                7'b01_x_xxx_1:begin
                    if (temp_frac[2:0]>3'b000) begin
                        temp_frac=temp_frac+28'b1000;
                    end
                end
                7'b10_x_xxx_0:begin
                    if (temp_frac[2:0]>3'b000) begin
                        temp_frac=temp_frac+28'b1000;
                    end
                end
                default :   temp_frac=temp_frac;
            endcase
            if (temp_frac[27]==1) begin
                temp_frac=temp_frac>>1;
                i=i+1;
            end else if (temp_frac[26]==0) begin
                while (~temp_frac[26]) begin
                    temp_frac=temp_frac<<1;
                    i=i-1;
                end
            end
            casex({i[7:0]==8'hff,crm,sign})
                4'b0_xx_x:s={sign,i[7:0],temp_frac[25:3]};
                4'b1_00_x:s={sign,i[7:0],23'b0};
                4'b1_01_0:s={sign,8'hfe,23'h7fffff};//向负无穷
                4'b1_01_1:s={sign,8'hff,23'b0};
                4'b1_10_0:s={sign,8'hff,23'b0};
                4'b1_10_1:s={sign,8'hfe,23'h7fffff};//向正无穷
                4'b1_11_x:s={sign,8'hfe,23'h7fffff};//截断
            endcase
        end
```

```
    end
endmodule
```

Verilog

顶层模块

设置clear导致仿真卡住

```
`timescale 1ns / 1ps

module top_design(
    input clk,
    input [31:0]a,b,
    input [1:0]rm,
    input sub,
    output [31:0]s
);
    wire op_sub,isNan,isInf,sign;
    wire [27:0]aligned_large_frac,aligned_small_frac;
    wire [23:0]nan_frac,inf_frac;
    wire [31:0]fp_large,fp_small;
    wire [7:0]temp_e;
    wire [1:0]crm;
    align_design  align_design_inst (
        .a(a),
        .b(b),
        .rm(rm),
        .sub(sub),
        .op_sub(op_sub),
        .isNan(isNan),
        .isInf(isInf),
        .sign(sign),
        .aligned_large_frac(aligned_large_frac),
        .aligned_small_frac(aligned_small_frac),
```

```
        .nan_frac(nan_frac),
        .inf_frac(inf_frac),
        .fp_large(fp_large),
        .fp_small(fp_small),
        .temp_e(temp_e),
        .crm(crm)
    );
    wire cop_sub,cisNan,cisInf,csign;
    wire [27:0]caligned_large_frac,caligned_small_frac;
    wire [23:0]cnan_frac,cinf_frac;
    wire [31:0]cfp_large,cfp_small;
    wire [7:0]ctemp_e;
    wire [1:0]ccrm;
    reg_design
#(.WIDTH(182))align_cal(clk,1,0,{op_sub,isNan,isInf,sign,aligned_large_frac,aligned_small_frac,nan_f
rac,inf_frac,
fp_large,fp_small,temp_e,crm},{cop_sub,cisNan,cisInf,csign,caligned_large_frac,caligned_small_frac,c
nan_frac,cinf_frac,
    cfp_large,cfp_small,ctemp_e,ccrm});

    wire [27:0]cal_frac;
    cal_design  cal_design_inst (
        .aligned_large_frac(caligned_large_frac),
        .aligned_small_frac(caligned_small_frac),
        .op_sub(cop_sub),
        .cal_frac(cal_frac)
    );
    wire nisNan,nisInf,nsign;
    wire [23:0]nnan_frac,ninf_frac;
    wire [31:0]nfp_large,nfp_small;
    wire [7:0]ntemp_e;
    wire [1:0]ncrm;
```

```
    wire [27:0]ncal_frac;
    reg_design #(.WIDTH(153))cal_normal(clk,1,0,{cisNan,cisInf,csign,cnan_frac,cinf_frac,
    cfp_large,cfp_small,ctemp_e,ccrm,cal_frac},{nisNan,nisInf,nsign,nnan_frac,ninf_frac,
    nfp_large,nfp_small,ntemp_e,ncrm,ncal_frac});
    normal_design  normal_design_inst (
        .cal_frac(ncal_frac),
        .isNan(nisNan),
        .isInf(nisInf),
        .sign(nsign),
        .nan_frac(nnan_frac),
        .inf_frac(ninf_frac),
        .fp_large(nfp_large),
        .fp_small(nfp_small),
        .temp_e(ntemp_e),
        .crm(ncrm),
        .s(s)
    );
endmodule
```

Verilog

9.4浮点数乘法器FMUL

9.4.1浮点数乘法算法

浮点数乘法算法相比于浮点数加法算法来说：**不需要对阶——只有计算和规格化两步**

浮点数的乘法分为以下几种：

1. 两个规格数相乘

给定$a=\{s_a,e_a,f_a\},b=\{s_b,e_b,f_b\}$，那么$c=\{s_c,e_c,f_c\}$

$$s_c=s_a bigoplus s_b$$

$$|c|=|a||b|=(2^{e\_a-127}\times 1.f_a)\times(2^{e\_b-127}\times 1.f_b)=2^{(e\_a+e\_b-127)-127}\times(1.f_a\times .1.f_b)$$

而1<1.fa✖1.fb<4，若1<1.fa✖1.fb<2，则$e_c=e_a+e_b-127$

否则$e_c=e_a+e_b-127+1$,$1.fc=(1.fa\times 1.fb)>>1$

而1≤ea, eb≤254, 则-125≤ea+eb-127≤381，-124≤ea+eb-127+1≤382，则ec超出了1~254

若ec在1~254那么就是个规格数，ec>254即为无穷，若ec<1，c≥$2^{-126}*2^{-23}=2^{-149}$，则c是一个非规格数，否则是0

2. 规格数乘以非规格数

给定$a=\{s_a,e_a,f_a\},b=\{s_b,e_b,f_b\}$，那么$c=\{s_c,e_c,f_c\}$

$$s_c=s_a bigoplus s_b$$

$$|c|=|a||b|=(2^{e\_a-127}\times 1.f_a)\times(2^{-126}\times 0.f_b)=2^{(e\_a-253)}\times(1.f_a\times .0.f_b)$$

则最大时，ea=254，$(2-2^{-23})\times(1-2^{-23})\times 2$，是规格化数

则最小时，ea=1，$2^{-252}\times 1\times 2^{-23}=2^{-275}$，视为0

3. 两个非规格数相乘

给定$a=\{s_a,e_a,f_a\},b=\{s_b,e_b,f_b\}$，那么$c=\{s_c,e_c,f_c\}$

$$s_c=s_a bigoplus s_b$$

$$|c|=|a||b|=(2^{-126}\times 0.f_a)\times(2^{-126}\times 0.f_b)=2^{-252}\times(0.f_a\times .0.f_b)$$

结果比最小的非正规化数要小视为0

4. 无穷乘以0

无穷乘以0结果为NaN

5. 无穷乘以一个不为0不为Nan的数

如果b既不等于0，也不是Nan，那么无穷乘以b=无穷

6. 一个不为0不为Nan的数乘以Nan

如果b既不等于0，也不是Nan，那么Nan乘以b=Nan

浮点数乘法算法

1. 判断特殊情况的乘法 iszero、isnan、isinf

无穷乘以0➞Nan

无穷乘以b➞无穷（b既不是0也不是nan）

Nan乘以b➞Nan（b既不是0也不是nan）

> **Nan的尾数取较大的乘数的尾数，无穷的尾数为0**

2. 求结果的符号位，和暂时的指数位

结果=乘数符号异或

暂时的指数位需要考虑非规格化数，非规格数是0-127+1

那么**暂时的指数=a的指数位+b的指数位-127+a指数位是否是0+b指数位是否是0**

3. Wallace数计算尾数相乘的结果——**流水的话这个是第一阶段**
4. 将Wallace模块输出的sum和c相加得到最终的尾数结果xx.xxxxxxx48位——**流水第二阶段**
5. 对结果进行规格化处理——**流水第三阶段**
    1. 进行规格化"1x.xxxxx"和"0x.xxxxx","01.xxxxx"直接进入到下一步
    2. grs，结果={结果[46:21],|[20:0]}进行舍入处理
    3. 判断指数小于1时是非规格数还是0
    4. 再进行规格化

9.4.2 浮点FMUL实现

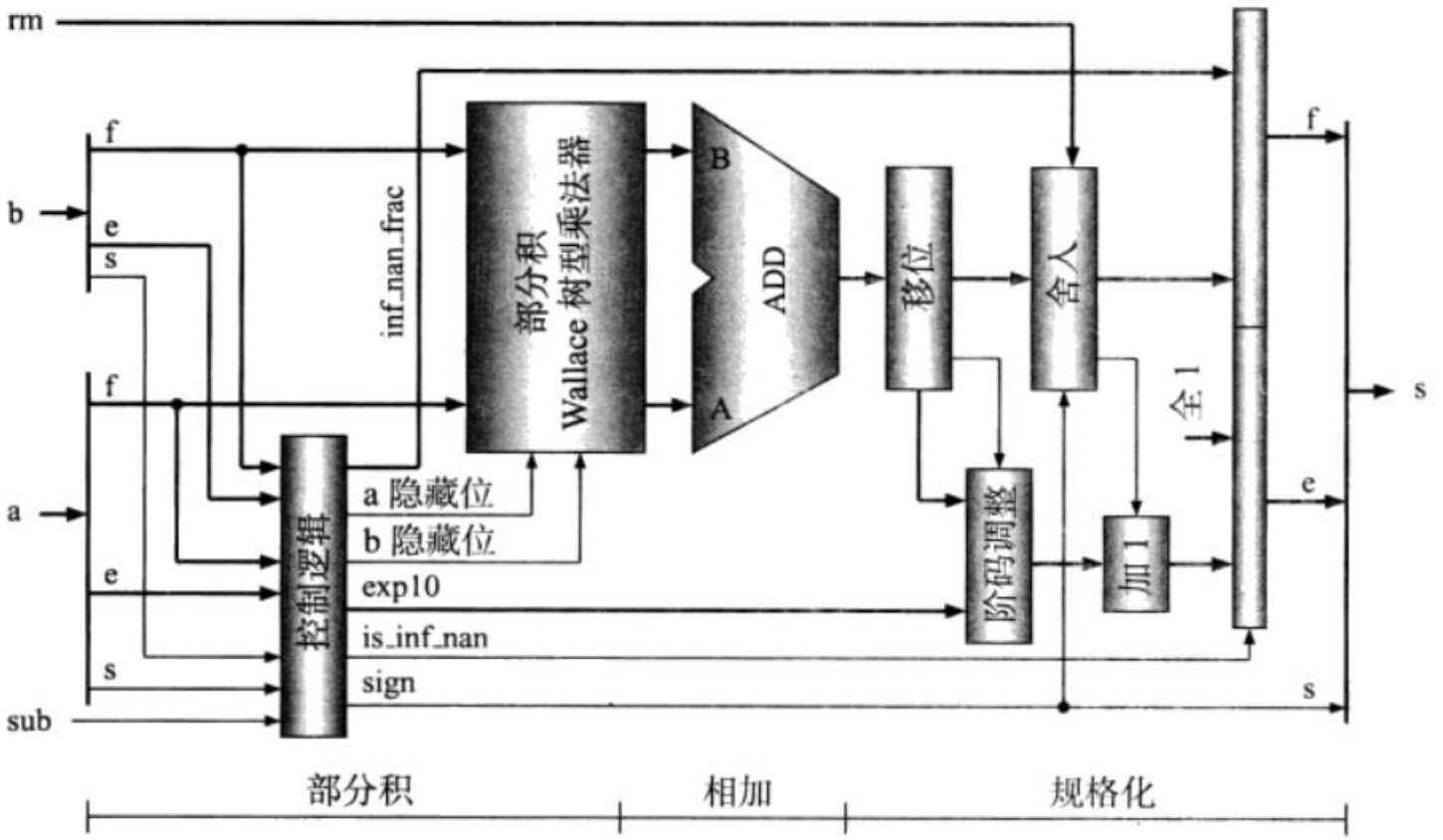

图 9.13　浮点乘法器电路的总体模块图

```
`timescale 1ns / 1ps

module fmul_design(
    input [31:0]a,b,
    input [1:0]rm,
    output reg[31:0]s
```

```
);
    //判断特殊情况
    wire aexp_zero=~|a[30:23];
    wire bexp_zero=~|b[30:23];
    wire aexp_f=&a[30:23];
    wire bexp_f=&b[30:23];
    wire a_frac0=~|a[22:0];
    wire b_frac0=~|b[22:0];
    wire aiszero=aexp_zero&a_frac0;
    wire biszero=bexp_zero&b_frac0;
    wire aisinf=aexp_f&a_frac0;
    wire bisinf=bexp_f&b_frac0;
    wire aisnan=aexp_f&~a_frac0;
    wire bisnan=bexp_f&~b_frac0;
    wire [23:0]nan_frac={1'b0,b[22:0]}>{1'b0,a[22:0]}?{1'b1,b[22:0]}:{1'b1,a[22:0]};
    //计算结果的符号，和结果暂时的指数
    wire sign=a[31]^b[31];
    reg [9:0]temp_e;
    //wallace计算结果
    wire [23:0]a_frac={~aexp_zero,a[22:0]};
    wire [23:0]b_frac={~bexp_zero,b[22:0]};
    wire [47:0]sum,c;
    wallace24 wallace24_inst(a_frac,b_frac,sum,c);
    wire [47:0]temp=sum+(c<<1);
    reg [47:0]temp_frac;//xx.xxxxx
    reg [27:0]frac;
    wire [149:0]min={149'b0,1'b1};
    reg [149:0]cmin;
    //规格化
    always @(*) begin
        temp_frac=temp;
```

```
        temp_e={2'b0,a[30:23]}+{2'b0,b[30:23]}-10'd127+aexp_zero+bexp_zero;
        if (aiszero&bisinf|biszero&aisinf) begin
            s={sign,8'hff,nan_frac[22:0]};
        end else if (aisinf|bisinf) begin
            s={sign,8'hff,23'b0};
        end else if (aisnan|bisnan) begin
            s={sign,8'hff,nan_frac[22:0]};
        end else begin
            if (temp_frac[47]==1) begin//1x.xxxx
                temp_frac=temp_frac>>1;
                temp_e=temp_e+1;
            end else begin//0x.xxxx
                while (~temp_frac[46]&&temp_e>0) begin
                    temp_frac=temp_frac<<1;
                    temp_e=temp_e-1;
                end
            end
            frac={temp_frac[47:21],|temp_frac[20:0]};
            casex({rm,frac[3],frac[2:0],sign})
                7'b00_1_100_x:frac=frac+28'b1000;
                7'b00_x_1xx_x:begin
                    if (frac[2:0]>3'b100) begin
                        frac=frac+28'b1000;
                    end
                end
                7'b01_x_xxx_1:begin
                    if (frac[2:0]>3'b000) begin
                        frac=frac+28'b1000;
                    end
                end
                7'b10_x_xxx_0:begin
                    if (frac[2:0]>3'b000) begin
                        frac=frac+28'b1000;
```

```
                end
            end
            default:frac=frac;
        endcase
        if (frac[27]==1) begin
            frac=frac>>1;
            temp_e=temp_e+1;
        end else if (frac[26]==0) begin
            while (~frac[26]&&temp_e>0) begin
                frac=frac<<1;
                temp_e=temp_e-1;
            end
        end
        if (temp_e[9]==1|temp_e[7:0]<1)begin//小于1，为正小于1或者为负
            cmin[149:122]=frac;
            cmin=cmin>>~(temp_e-127)+1;
            if (cmin>min) begin
                temp_e=0;
                frac=frac>>1;//非规格化数
            end else begin
                temp_e=0;
                frac=0;
            end
        end
        casex({temp_e[9:0]>=10'h0ff,rm,sign})
            4'b0_xx_x:s={sign,temp_e[7:0],frac[25:3]};
            4'b1_00_x:s={sign,8'hff,23'b0};//无穷
            4'b1_01_0:s={sign,8'hfe,23'h7fffff};
            4'b1_01_1:s={sign,8'hff,23'b0};//向负无穷
            4'b1_10_0:s={sign,8'hff,23'b0};//向正无穷
            4'b1_10_1:s={sign,8'hfe,23'h7fffff};
```

```
                4'b1_11_x:s={sign,8'hfe,23'h7fffff};//截断
            endcase

        end
    end
endmodule
```

Verilog

9.4.3流水浮点FMUL实现

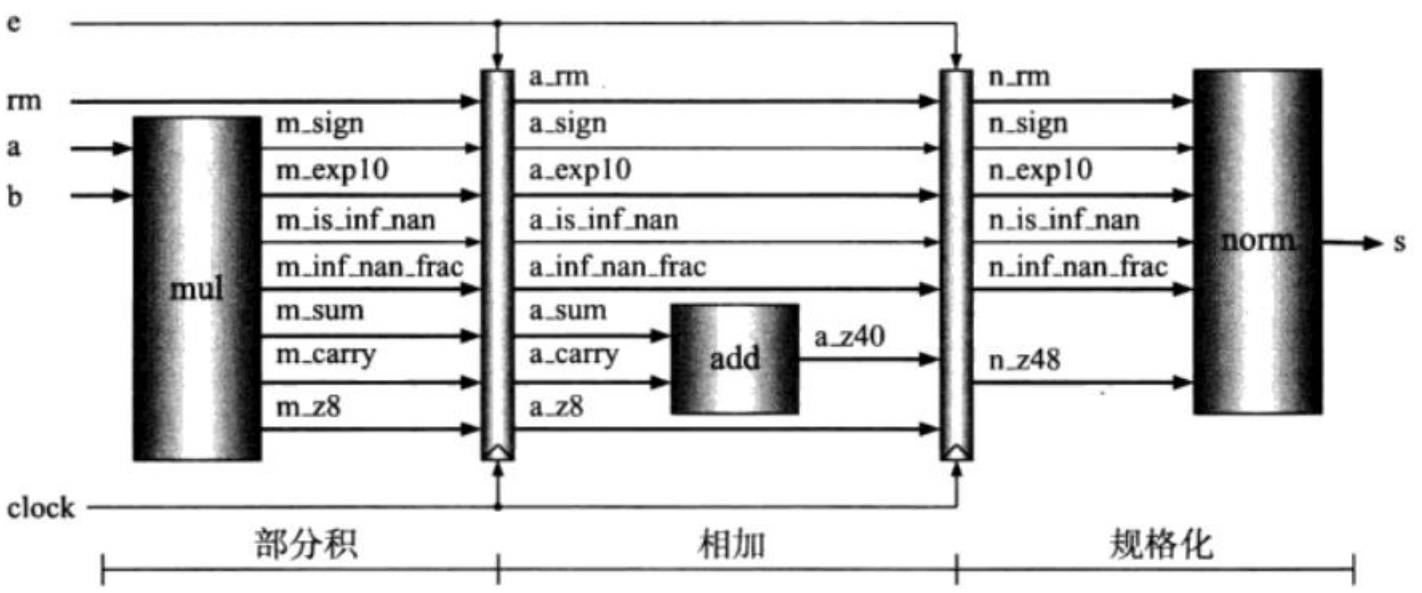

图 9.18　流水线浮点乘法器

顶层模块

```
`timescale 1ns / 1ps

module pipe_fmul_top(
    input clk,
    input [31:0]a,b,
    input [1:0]rm,
    output [31:0]s
```

```
);
    wire aisinf,aiszero,aisnan,bisinf,biszero,bisnan;
    wire [23:0]nan_frac;
    wire sign;
    wire [9:0]temp_e;
    wire [47:0]sum,c;
    partical  partical_inst (
        .a(a),
        .b(b),
        .aisinf(aisinf),
        .aiszero(aiszero),
        .aisnan(aisnan),
        .bisinf(bisinf),
        .biszero(biszero),
        .bisnan(bisnan),
        .nan_frac(nan_frac),
        .sign(sign),
        .temp_e(temp_e),
        .sum(sum),
        .c(c)
    );

    wire aaisinf,aaiszero,aaisnan,abisinf,abiszero,abisnan;
    wire [23:0]anan_frac;
    wire asign;
    wire [9:0]atemp_e;
    wire [47:0]asum,ac;
    wire [1:0]arm;
    reg_design #(.WIDTH(139))
partical_add(clk,1,0,{aisinf,aiszero,aisnan,bisinf,biszero,bisnan,nan_frac,sign,temp_e,sum,c,rm},
    {aaisinf,aaiszero,aaisnan,abisinf,abiszero,abisnan,anan_frac,asign,atemp_e,asum,ac,arm});
    wire [47:0]res;
    add  add_inst (
```

```
        .sum(asum),
        .c(ac),
        .res(res)
    );
    wire naisinf,naiszero,naisnan,nbisinf,nbiszero,nbisnan;
    wire [23:0]nnan_frac;
    wire nsign;
    wire [9:0]ntemp_e;
    wire [47:0]nres;
    wire [1:0]nrm;
    reg_design #(.WIDTH(91))
add_normal(clk,1,0,{aaisinf,aaiszero,aaisnan,abisinf,abiszero,abisnan,anan_frac,asign,atemp_e,res,ar
m},
    {naisinf,naiszero,naisnan,nbisinf,nbiszero,nbisnan,nnan_frac,nsign,ntemp_e,nres,nrm});

    normal  normal_inst (
        .aisinf(naisinf),
        .aiszero(naiszero),
        .aisnan(naisnan),
        .bisinf(nbisinf),
        .biszero(nbiszero),
        .bisnan(nbisnan),
        .rm(nrm),
        .nan_frac(nnan_frac),
        .sign(nsign),
        .exp(ntemp_e),
        .res(nres),
        .s(s)
    );
endmodule
```

Verilog

部分积模块

```
`timescale 1ns / 1ps

module partical(
    input [31:0]a,b,
    output aisinf,aiszero,aisnan,bisinf,biszero,bisnan,
    output [23:0]nan_frac,
    output sign,
    output [9:0]temp_e,
    output [47:0]sum,c
);
    //判断特殊情况
    wire aexp_zero=~|a[30:23];
    wire bexp_zero=~|b[30:23];
    wire aexp_f=&a[30:23];
    wire bexp_f=&b[30:23];
    wire a_frac0=~|a[22:0];
    wire b_frac0=~|b[22:0];
    assign aiszero=aexp_zero&a_frac0;
    assign biszero=bexp_zero&b_frac0;
    assign aisinf=aexp_f&a_frac0;
    assign bisinf=bexp_f&b_frac0;
    assign aisnan=aexp_f&~a_frac0;
    assign bisnan=bexp_f&~b_frac0;
    assign nan_frac={1'b0,b[22:0]}>{1'b0,a[22:0]}?{1'b1,b[22:0]}:{1'b1,a[22:0]};
    //计算结果的符号，和结果暂时的指数
    assign sign=a[31]^b[31];
    assign temp_e={2'b0,a[30:23]}+{2'b0,b[30:23]}-10'd127+aexp_zero+bexp_zero;
    //wallace计算结果
    wire [23:0]a_frac={~aexp_zero,a[22:0]};
    wire [23:0]b_frac={~bexp_zero,b[22:0]};
```

```
    wallace24 wallace24_inst(a_frac,b_frac,sum,c);
endmodule
```

Verilog

相加模块

```
`timescale 1ns / 1ps

module add(
    input [47:0]sum,
    input [47:0]c,
    output [47:0]res
);
    assign res=sum+(c<<1);
endmodule
```

Verilog

规格化模块

```
`timescale 1ns / 1ps

module normal(
    input aisinf,aiszero,aisnan,bisinf,biszero,bisnan,
    input [1:0]rm,
    input [23:0]nan_frac,
    input sign,
    input [9:0]exp,
    input [47:0]res,
    output reg[31:0]s
);
    reg [47:0]temp_frac;//xx.xxxxx
    reg [9:0]temp_e;
    reg [27:0]frac;
```

```
    wire [149:0]min={149'b0,1'b1};
    reg [149:0]cmin;
    //规格化
    always @(*) begin
        temp_frac=res;
        temp_e=exp;
        if (aiszero&bisinf|biszero&aisinf) begin
            s={sign,8'hff,nan_frac[22:0]};
        end else if (aisinf|bisinf) begin
            s={sign,8'hff,23'b0};
        end else if (aisnan|bisnan) begin
            s={sign,8'hff,nan_frac[22:0]};
        end else begin
            if (temp_frac[47]==1) begin//1x.xxxx
                temp_frac=temp_frac>>1;
                temp_e=temp_e+1;
            end else begin//0x.xxxx
                while (~temp_frac[46]&&temp_e>0) begin
                    temp_frac=temp_frac<<1;
                    temp_e=temp_e-1;
                end
            end
            frac={temp_frac[47:21],|temp_frac[20:0]};
            casex({rm,frac[3],frac[2:0],sign})
                7'b00_1_100_x:frac=frac+28'b1000;
                7'b00_x_1xx_x:begin
                    if (frac[2:0]>3'b100) begin
                        frac=frac+28'b1000;
                    end
                end
                7'b01_x_xxx_1:begin
                    if (frac[2:0]>3'b000) begin
                        frac=frac+28'b1000;
```

```
                end
            end
            7'b10_x_xxx_0:begin
                if (frac[2:0]>3'b000) begin
                    frac=frac+28'b1000;
                end
            end
            default:frac=frac;
        endcase
        if (frac[27]==1) begin
            frac=frac>>1;
            temp_e=temp_e+1;
        end else if (frac[26]==0) begin
            while (~frac[26]&&temp_e>0) begin
                frac=frac<<1;
                temp_e=temp_e-1;
            end
        end
        if (temp_e[9]==1|temp_e[7:0]<8'b1)begin
            cmin[149:122]=frac;
            cmin=cmin>>~(temp_e-127)+1;
            if (cmin>min) begin
                temp_e=0;
                frac=frac>>1;
            end else begin
                temp_e=0;
                frac=0;
            end

        end
        casex({temp_e[9:0]>=10'h0ff,rm,sign})
            4'b0_xx_x:s={sign,temp_e[7:0],frac[25:3]};
            4'b1_00_x:s={sign,8'hff,23'b0};//无穷
```

```
                4'b1_01_0:s={sign,8'hfe,23'h7fffff};//向负无穷
                4'b1_01_1:s={sign,8'hff,23'b0};
                4'b1_10_0:s={sign,8'hff,23'b0};
                4'b1_10_1:s={sign,8'hfe,23'h7fffff};//向正无穷
                4'b1_11_x:s={sign,8'hfe,23'h7fffff};//截断
            endcase

    end
    end
endmodule
```

Verilog

9.5浮点数除法器FDIV

9.5.1浮点数除法算法

浮点除法和浮点乘法比较相似，但是有两个不同点：

1. 指数的计算采用减法
2. 结果小数的计算是采用除法——这里用Newton-Rapthon算法

下面根据浮点数的分类来讨论除法：

1. 两个规格化浮点数的除法

给定$a=\{s_a,e_a,f_a\},b=\{s_b,e_b,f_b\}$，那么$c=\{s_c,e_c,f_c\}$

$$s_c = s_a bigoplus s_b$$

$$|c| = |a|/|b| = (2^{e\_a-127}\times 1.f_a)\times(2^{e\_b-127}\times 1.f_b) = 2^{(e\_a-e\_b+127)-127}\times(1.f_a \div .1.f_b)$$

而0.5<1.f_a/1.f_a<2（最大处最小得到上界，最小除最大得到下界）

若0.5<1.fa/1.fb<1，则$e_c = ea-eb+127-1$,$1.fc=(1.fa\div 1.fb)<<1$*因为规格化*

若1≤1.fa/1.fb<2，则$e_c = ea-eb+127$,$1.fc=1.fa\div 1/fb$

因此ec的范围是-126≤ec≤380或-127≤ec≤379，超过了1~254

则若ec在1~254则视为规格化数，若ec>254则视为无穷

若ec<1且$c\geq 2^{-126-23}$,则c为非规格化数否则为0

2. 规格化浮点数除以非规格化浮点数

给定$a=\{s_a,e_a,f_a\},b=\{s_b,e_b,f_b\}$，那么$c=\{s_c,e_c,f_c\}$

$$s_c=s_a bigoplus s_b$$

$$|c|=|a|/|b|=(2^{e\_a-127}\times 1.f_a)\times(2^{-126}\times 0.f_b)=2^{(e\_a+126)-127}\times(1.f_a\div .0.f_b)$$

最大值是$2^{254-1}\times(2-2^{-23})/2^{-23}=2^{276}\times(2-2^{-23})$，大于最大值视为无穷

最小值是$2^0\times(1/(1-2^{-23})$，是一个非规格化数

3. 非规格化浮点数除以规格化浮点数

给定$a=\{s_a,e_a,f_a\},b=\{s_b,e_b,f_b\}$，那么$c=\{s_c,e_c,f_c\}$

$$s_c=s_a bigoplus s_b$$

$$|c|=|a|/|b|=(2^{-126}\times 0.f_a)\times(2^{eb-127}\times 1.f_b)=2^{(1-e\_b)}\times(0.f_a\div .1.f_b)$$

最大值是$2^0\times((1-2^{-23})/1.0)=1-2^{-23}$，是一个规格化数

最小值是$2^{-253}\times(2^{-23}/(2-2^{-23})$视为0

4. 两个非规格化浮点数相除

两个非规格化数相除$|c|=|a|/|b|=0.fa/0.fb$

无论fa、fb取何值均能够得到规格化数

5. 特殊情况

Finally, consider some special calculations: (i) if $a\neq\infty$ and $a\neq$ NaN, $a/\infty=0$; (ii) if $a\neq 0$ and $a\neq$ NaN, $a/0=\infty$; (iii) $0/0=$ NaN; (iv) $\infty/\infty=$ NaN; and (v) NaN$/b=$ NaN.

📌浮点数除法计算器采用Newton-Rapthon算法，计算所得结果需要四步

1. 使用Newton-Rapthon迭代xn：迭代公式$x_{i+1}=x_i(2-x_i b)$

牛顿迭代法要求0.5≤a,b≤1，因此需要先将非规格化数转为规格化数，然后取{隐含位,[22:0]}共同作为小数——小数点位于最左

为了向浮点乘法流水线靠近——迭代期间（**三次迭代**）阻塞流水线，使得能够和浮点数乘法一样在执行阶段用到wallace乘法器

迭代的计算包括两个乘法和一个减法，乘法采用Wallace树需要两个时钟周期减法需要一个时钟周期，故每次迭代需要5个时钟周期——在count=1(取x0)、6、11、16时更新

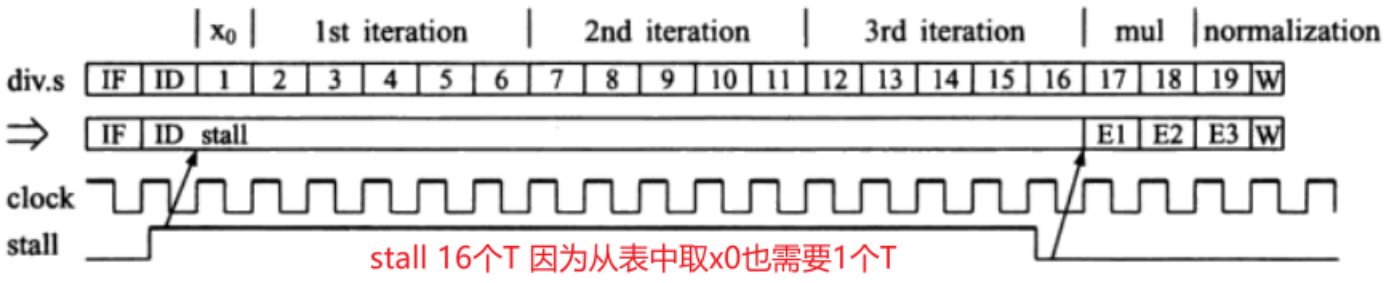

图 9.21　浮点除法指令的流水线

2. 使用Wallace树乘法计算sum、c：xn*a
3. 相加sum、c
4. 规格化

> *The Newton－Raphson division algorithm consists of two parts: (i) iterative calculation: xi+1 = xi (2 − xi b) and (ii) quotient calculation: q = a × xn. The second part performs a multiplication, which can be implemented with a pipelined Wallace tree. Therefore, the operation of the float division takes the following four steps:*
>
> *1. Newton iteration, which calculates xn;*
>
> *2. partial product, which calculates carry and sum with Wallace CSA array;*
>
> *3. addition, which calculates the product by adding the carry and sum;*
>
> *4. normalization, which generates the final result in the IEEE float format.*

> x86中Nan值是32'h7fc00000

9.5.2 FDIV设计实现

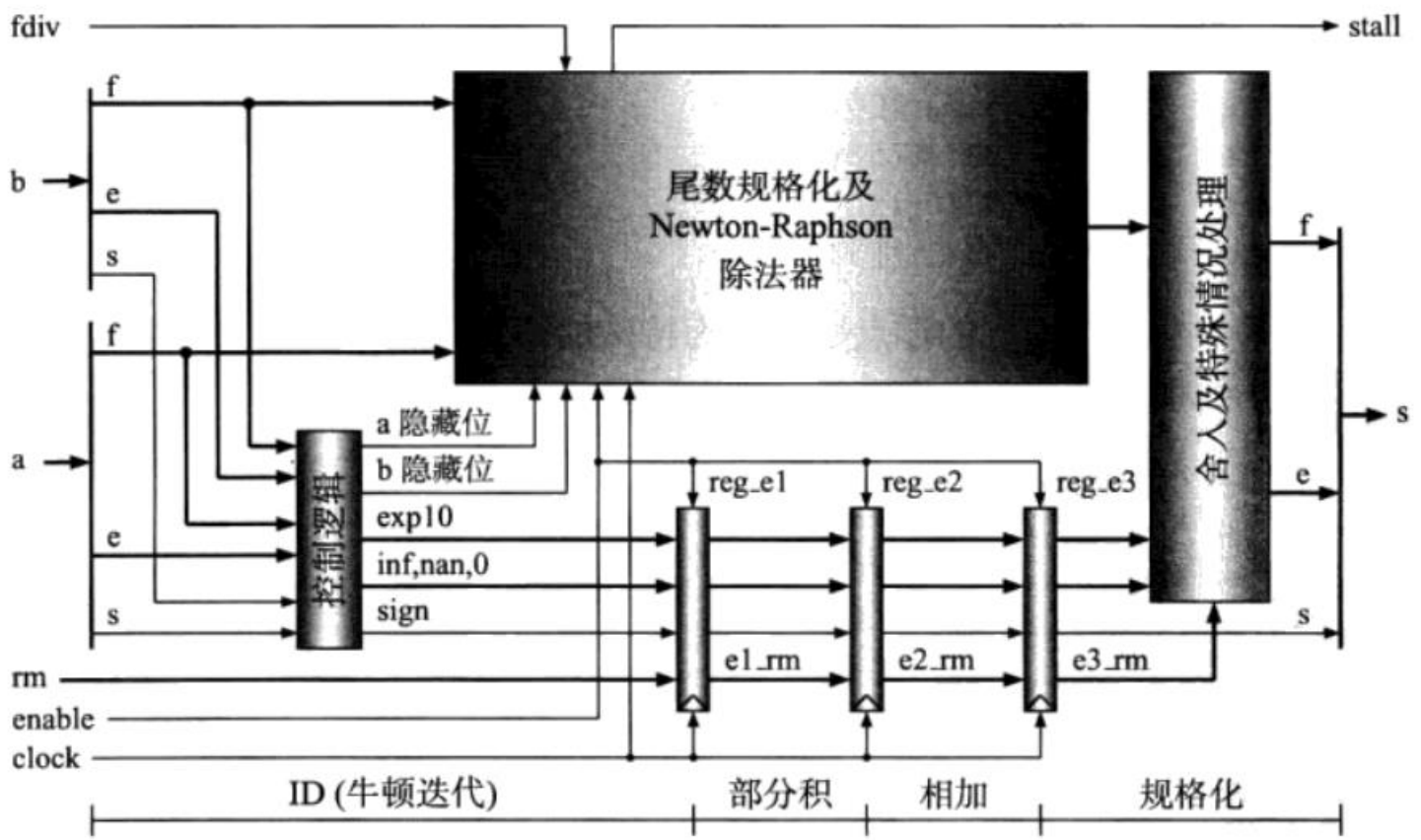

图 9.20　Newton-Raphson 浮点除法器电路的总体模块图

设计流程：

1. 判断特殊数inf、nan、zero

   a/b，当a不是Nan且不是无穷，a/∞=0；当a不是0不是Nan，a/0=∞

   0/0=Nan、Nan/b=Nan、∞/∞=Nan

2. 暂定结果符号和阶码，以及运算数的尾数（非规格化数左移1位以使得阶码不用分类）

   结果符号是操作数符号异或

   阶码暂定为a阶-b阶+127

   运算数的尾数为规格化数{1'b1,[22:0]}，非规格化数{[22:0],1'b0}

3. 确定输入到计算器的浮点数满足尾数的最高位为1

   对非规格化数操作不断左移，至最高位是1，并记录移位次数，调整阶码

4. 使用newton-raphson24位除法模块，计算结果

   采用24*26、26*26的Wallace做乘法，clk计数

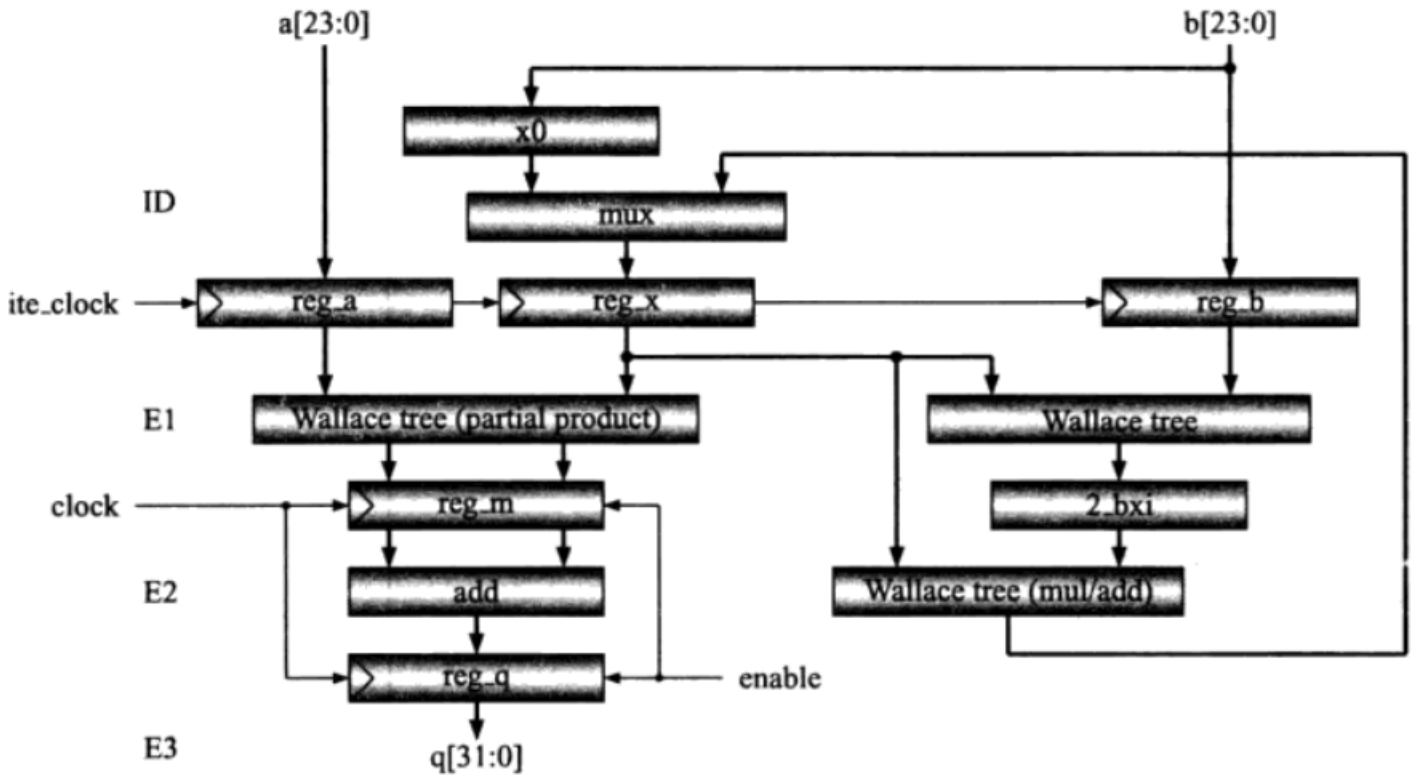

图 9.22　Newton-Raphson 除法器

初始值xo通过查ROM表得到，用一个时钟周期取x0

与我们在第3章介绍的32位版本不同，我们这里使用了实际的时钟周期进行计数，而不是对迭代次数进行计数。计数器值为0时表示指令处在ID级。如果当前指令是浮点除法且计数器值为0，把计数器设为1，并令busy信号也为1。在下一个时钟上升沿处，把数据xo,a和b分别打入寄存器regx,rega和regb。

一次迭代用5个时钟周期：两次乘法用4个周期，一次减法用一个周期。在计算过程中，当计数值为6,11和16时，我们修改寄存器x的内容(3次迭代)。当计数器为15时，清除busy。当计数器为16时，清零，以允许下一条指令的执行

当xn计算出之后，下一个周期使用Wallace树型乘法算法计算出部分积的和ms与进位mc

5. 规格化

Wallace24x26

```
`timescale 1ns / 1ps

module wallace26x24(
    input [25:0]a,
    input [23:0]b,
    output [49:0]sum,c
```

```
);
    reg[49:0]ab[23:0];
    integer i,j;
    always @(*) begin
        for ( i= 0; i<24;i=i+1) begin
            for (j = 0; j<26; j=j+1) begin
                ab[i][j]=b[i]*a[j];
            end
            ab[i][49:26]=32'b0;
        end
    end
    wire
[49:0]sum1_1,c1_1,sum1_2,c1_2,sum1_3,c1_3,sum1_4,c1_4,sum1_5,c1_5,sum1_6,c1_6,sum1_7,c1_7,sum1_8,c1_
8;
    csa #(.WIDTH(50))csa_1_1(ab[0],ab[1]<<1,ab[2]<<2,sum1_1,c1_1);
    csa #(.WIDTH(50))csa_1_2(ab[3]<<3,ab[4]<<4,ab[5]<<5,sum1_2,c1_2);
    csa #(.WIDTH(50))csa_1_3(ab[6]<<6,ab[7]<<7,ab[8]<<8,sum1_3,c1_3);
    csa #(.WIDTH(50))csa_1_4(ab[9]<<9,ab[10]<<10,ab[11]<<11,sum1_4,c1_4);
    csa #(.WIDTH(50))csa_1_5(ab[12]<<12,ab[13]<<13,ab[14]<<14,sum1_5,c1_5);
    csa #(.WIDTH(50))csa_1_6(ab[15]<<15,ab[16]<<16,ab[17]<<17,sum1_6,c1_6);
    csa #(.WIDTH(50))csa_1_7(ab[18]<<18,ab[19]<<19,ab[20]<<20,sum1_7,c1_7);
    csa #(.WIDTH(50))csa_1_8(ab[21]<<21,ab[22]<<22,ab[23]<<23,sum1_8,c1_8);

    wire [49:0]sum2_1,c2_1,sum2_2,c2_2,sum2_3,c2_3,sum2_4,c2_4,sum2_5,c2_5;
    csa #(.WIDTH(50))csa_2_1(sum1_1,c1_1<<1,sum1_2,sum2_1,c2_1);
    csa #(.WIDTH(50))csa_2_2(c1_2<<1,sum1_3,c1_3<<1,sum2_2,c2_2);
    csa #(.WIDTH(50))csa_2_3(sum1_4,c1_4<<1,sum1_5,sum2_3,c2_3);
    csa #(.WIDTH(50))csa_2_4(c1_5<<1,sum1_6,c1_6<<1,sum2_4,c2_4);
    csa #(.WIDTH(50))csa_2_5(sum1_7,c1_7<<1,sum1_8,sum2_5,c2_5);//c1_8未计算

    wire [49:0]sum3_1,c3_1,sum3_2,c3_2,sum3_3,c3_3;
    csa #(.WIDTH(50))csa_3_1(sum2_1,c2_1<<1,sum2_2,sum3_1,c3_1);
    csa #(.WIDTH(50))csa_3_2(c2_2<<1,sum2_3,c2_3<<1,sum3_2,c3_2);
```

```
    csa #(.WIDTH(50))csa_3_3(sum2_4,c2_4<<1,sum2_5,sum3_3,c3_3);//c2_5,c1_8未计算

    wire [49:0]sum4_1,c4_1,sum4_2,c4_2;
    csa #(.WIDTH(50))csa_4_1(sum3_1,c3_1<<1,sum3_2,sum4_1,c4_1);
    csa #(.WIDTH(50))csa_4_2(c3_2<<1,sum3_3,c3_3<<1,sum4_2,c4_2);//c2_5,c1_8未计算


    wire [63:0]sum5_1,c5_1,sum5_2,c5_2;
    csa csa_5_1(sum4_1,c4_1<<1,sum4_2,sum5_1,c5_1);
    csa csa_5_2(c4_2<<1,c2_5<<1,c1_8<<1,sum5_2,c5_2);

    wire [63:0]sum6_1,c6_1;
    csa csa_6_1(sum5_1,c5_1<<1,sum5_2,sum6_1,c6_1);//剩c5_2

    wire [63:0]sum7_1,c7_1;
    csa csa_7_1(sum6_1,c6_1<<1,c5_2<<1,sum,c);
endmodule
```

Verilog

Wallace26x26

```
`timescale 1ns / 1ps

module wallace26x26(
    input [25:0]a,
    input [25:0]b,
    output [51:0]sum,c
);
    reg[51:0]ab[25:0];
    integer i,j;
    always @(*) begin
        for ( i= 0; i<26;i=i+1) begin
```

```
            for (j = 0; j<26; j=j+1) begin
                ab[i][j]=b[i]*a[j];
            end
            ab[i][51:26]=26'b0;
        end
    end
    wire
[51:0]sum1_1,c1_1,sum1_2,c1_2,sum1_3,c1_3,sum1_4,c1_4,sum1_5,c1_5,sum1_6,c1_6,sum1_7,c1_7,sum1_8,c1_
8;
    csa #(.WIDTH(52))csa_1_1(ab[0],ab[1]<<1,ab[2]<<2,sum1_1,c1_1);
    csa #(.WIDTH(52))csa_1_2(ab[3]<<3,ab[4]<<4,ab[5]<<5,sum1_2,c1_2);
    csa #(.WIDTH(52))csa_1_3(ab[6]<<6,ab[7]<<7,ab[8]<<8,sum1_3,c1_3);
    csa #(.WIDTH(52))csa_1_4(ab[9]<<9,ab[10]<<10,ab[11]<<11,sum1_4,c1_4);
    csa #(.WIDTH(52))csa_1_5(ab[12]<<12,ab[13]<<13,ab[14]<<14,sum1_5,c1_5);
    csa #(.WIDTH(52))csa_1_6(ab[15]<<15,ab[16]<<16,ab[17]<<17,sum1_6,c1_6);
    csa #(.WIDTH(52))csa_1_7(ab[18]<<18,ab[19]<<19,ab[20]<<20,sum1_7,c1_7);
    csa #(.WIDTH(52))csa_1_8(ab[21]<<21,ab[22]<<22,ab[23]<<23,sum1_8,c1_8);//ab[24],ab[25]

    wire [51:0]sum2_1,c2_1,sum2_2,c2_2,sum2_3,c2_3,sum2_4,c2_4,sum2_5,c2_5,sum2_6,c2_6;
    csa #(.WIDTH(52))csa_2_1(sum1_1,c1_1<<1,sum1_2,sum2_1,c2_1);
    csa #(.WIDTH(52))csa_2_2(c1_2<<1,sum1_3,c1_3<<1,sum2_2,c2_2);
    csa #(.WIDTH(52))csa_2_3(sum1_4,c1_4<<1,sum1_5,sum2_3,c2_3);
    csa #(.WIDTH(52))csa_2_4(c1_5<<1,sum1_6,c1_6<<1,sum2_4,c2_4);
    csa #(.WIDTH(52))csa_2_5(sum1_7,c1_7<<1,sum1_8,sum2_5,c2_5);
    csa #(.WIDTH(52))csa_2_6(c1_8<<1,ab[24]<<24,ab[25]<<25,sum2_6,c2_6);

    wire [51:0]sum3_1,c3_1,sum3_2,c3_2,sum3_3,c3_3,sum3_4,c3_4;
    csa #(.WIDTH(52))csa_3_1(sum2_1,c2_1<<1,sum2_2,sum3_1,c3_1);
    csa #(.WIDTH(52))csa_3_2(c2_2<<1,sum2_3,c2_3<<1,sum3_2,c3_2);
    csa #(.WIDTH(52))csa_3_3(sum2_4,c2_4<<1,sum2_5,sum3_3,c3_3);
    csa #(.WIDTH(52))csa_3_4(c2_5<<1,sum2_6,c2_6<<1,sum3_4,c3_4);

    wire [51:0]sum4_1,c4_1,sum4_2,c4_2;
```

```
    csa #(.WIDTH(52))csa_4_1(sum3_1,c3_1<<1,sum3_2,sum4_1,c4_1);
    csa #(.WIDTH(52))csa_4_2(c3_2<<1,sum3_3,c3_3<<1,sum4_2,c4_2);//sum3_4,c3_4


    wire [51:0]sum5_1,c5_1,sum5_2,c5_2;
    csa #(.WIDTH(52))csa_5_1(sum4_1,c4_1<<1,sum4_2,sum5_1,c5_1);
    csa #(.WIDTH(52))csa_5_2(c4_2<<1,sum3_4,c3_4<<1,sum5_2,c5_2);

    wire [51:0]sum6_1,c6_1;
    csa #(.WIDTH(52))csa_6_1(sum5_1,c5_1<<1,sum5_2,sum6_1,c6_1);//剩c5_2

    wire [51:0]sum7_1,c7_1;
    csa #(.WIDTH(52))csa_7_1(sum6_1,c6_1<<1,c5_2<<1,sum,c);
endmodule
```

Verilog

移位

```
`timescale 1ns / 1ps

module shift_toMSB1(
    input [23:0]a,
    output reg[23:0]b,
    output reg[4:0]amount
);
    always @(*) begin
        amount=5'b0;
        b=a;
        while (~b[23]) begin
            b=b<<1;
            amount=amount+1;
        end
```

```
    end
endmodule
```

Verilog

Fdiv_design

```
`timescale 1ns / 1ps

module fdiv_design(
    input clk,enable,clear,
    input [1:0]rm,
    input [31:0]a,b,
    input ifdiv,
    output reg [31:0]res,
    output stall,
    output reg busy,
    output reg[4:0]count
);
    //首先判断特殊情况
    wire aexp_zero=~|a[30:23];
    wire bexp_zero=~|b[30:23];
    wire aexp_f=&a[30:23];
    wire bexp_f=&b[30:23];
    wire a_frac0=~|a[22:0];
    wire b_frac0=~|b[22:0];
    wire aiszero=aexp_zero&a_frac0;
    wire biszero=bexp_zero&b_frac0;
    wire aisinf=aexp_f&a_frac0;
    wire bisinf=bexp_f&b_frac0;
    wire aisnan=aexp_f&~a_frac0;
    wire bisnan=bexp_f&~b_frac0;
```

```
    //结果的符号，阶码和尾数
    wire sign=a[31]^b[31];
    wire [9:0]exp1={2'b0,a[30:23]}-{2'b0,b[30:23]}+10'd127;
    wire [23:0]a_tempfrac=aexp_zero?{a[22:0],1'b0}:{1'b1,a[22:0]};
    wire [23:0]b_tempfrac=bexp_zero?{b[22:0],1'b0}:{1'b1,b[22:0]};

    //求满足计算的尾数
    wire [23:0]a_frac,b_frac;
    wire [4:0]amount_a,amount_b;
    shift_toMSB1 getaFrac(a_tempfrac,a_frac,amount_a);
    shift_toMSB1 getbFrac(b_tempfrac,b_frac,amount_b);
    wire [9:0]exp2=exp1-amount_a+amount_b;

    reg [25:0]reg_x;//xx.xxxx
    reg [23:0]reg_b;//x.xxxx
    //迭代求xn
    wire [49:0]bxi,sum1,c1;//xx.xxxxxxxx
    wire [51:0]x52,sum2,c2;//xxx.xxxxxxxx
    wallace26x24 wallace26_24(reg_x,reg_b,sum1,c1);
    assign bxi=sum1+(c1<<1);
    wire [25:0]temp=~bxi[48:23]+1'b1;
    wallace26x26 wallace26_26(reg_x,temp,sum2,c2);
    assign x52=sum2+(c2<<1);
    always @(posedge clk or posedge clear) begin
        if (clear) begin
            count<=5'b0;
            busy<=1'b0;
        end else begin
            if (ifdiv&(count==5'b0)) begin
                count<=5'b1;
                busy<=1'b1;
            end else begin
```

```
                if (count==5'b1) begin
                    reg_x<={2'b01,rom(b_frac[22:19]),16'b0};
                    reg_b<=b_frac;
                end
                if (count!=0) count<=count+1;
                if (count==5'd15) busy<=5'd0;
                if (count==5'd16) count<=5'd0;
                if (count==5'd6|count==5'd11|count==5'd16) reg_x<=x52[50:25];
            end
        end
    end
    assign stall=ifdiv&(count==5'b0|busy);
    //寄存器
    wire [23:0]wa_frac;
    wire [25:0]wreg_x;
    wire [9:0]wtemp_e;
    wire [1:0]wrm;
    wire waiszero,waisinf,waisnan,wbisinf,wbisnan,wbiszero,wsign;
    reg_design #(.WIDTH(69))r1(clk,~stall,clear,
    {aiszero,aisinf,aisnan,biszero,bisinf,bisnan,rm,reg_x,a_frac,exp2,sign},
    {waiszero,waisinf,waisnan,wbiszero,wbisinf,wbisnan,wrm,wreg_x,wa_frac,wtemp_e,wsign});
    //wallace求sum、c
    wire [49:0]sum,c;
    wallace26x24 wallace26_24_(wreg_x,wa_frac,sum,c);
    //寄存器
    wire [49:0]csum,cc;
    wire caiszero,caisinf,caisnan,cbiszero,cbisinf,cbisnan,csign;
    wire [1:0]crm;
    wire [9:0]ctemp_e;
    reg_design #(.WIDTH(119))r2(clk,~stall,clear,
    {waiszero,waisinf,waisnan,wbiszero,wbisinf,wbisnan,wrm,wtemp_e,wsign,sum,c},
    {caiszero,caisinf,caisnan,cbiszero,cbisinf,cbisnan,crm,ctemp_e,csign,csum,cc});
```

```
//求和
wire [49:0]q=csum+(cc<<1);//xx.xxxxx
//寄存器
wire [49:0]nq;
wire naiszero,naisinf,naisnan,nbiszero,nbisinf,nbisnan,nsign;
wire [1:0]nrm;
wire [9:0]ntemp_e;
reg_design #(.WIDTH(69))r3(clk,~stall,clear,
{caiszero,caisinf,caisnan,cbiszero,cbisinf,cbisnan,crm,ctemp_e,csign,q},
{naiszero,naisinf,naisnan,nbiszero,nbisinf,nbisnan,nrm,ntemp_e,nsign,nq});

//规格化
reg [49:0]temp_frac;//xx.xxxxx
reg [9:0]temp_e;
reg [27:0]frac;
wire [149:0]min={149'b0,1'b1};
reg [149:0]cmin;
//规格化
always @(*) begin
    temp_frac=nq;
    temp_e=ntemp_e;
    cmin=150'b0;
    if (naiszero&nbiszero|naisinf&nbisinf|naisnan) begin
        res=32'h7fc00000;//nan
    end else if (~naisinf&~naisnan&nbisinf) begin
        res={nsign,8'b0,23'b0};
    end else if (~naiszero&~naisnan&nbiszero) begin
        res={nsign,8'hff,23'b0};
    end else begin
        if (temp_frac[49]==1) begin//1x.xxxx 2.48
```

```
            temp_frac=temp_frac>>1;
            temp_e=temp_e+1;
        end else begin//0x.xxxx
            while (~temp_frac[48]&&temp_e>0) begin
                temp_frac=temp_frac<<1;
                temp_e=temp_e-1;
            end
        end
        frac={temp_frac[49:23],|temp_frac[22:0]};
        casex({nrm,frac[3],frac[2:0],nsign})
            7'b00_1_100_x:frac=frac+28'b1000;
            7'b00_x_1xx_x:begin
                if (frac[2:0]>3'b100) begin
                    frac=frac+28'b1000;
                end
            end
            7'b01_x_xxx_1:begin
                if (frac[2:0]>3'b000) begin
                    frac=frac+28'b1000;
                end
            end
            7'b10_x_xxx_0:begin
                if (frac[2:0]>3'b000) begin
                    frac=frac+28'b1000;
                end
            end
            default:frac=frac;
        endcase
        if (frac[27]==1) begin
            frac=frac>>1;
            temp_e=temp_e+1;
        end else if (frac[26]==0) begin
            while (~frac[26]&&temp_e>0) begin
```

```
                frac=frac<<1;
                temp_e=temp_e-1;
            end
        end
        if (temp_e[9]==1|temp_e<1)begin
            cmin[149:122]=frac;
            cmin=cmin>>~(temp_e-127)+1;
            if (cmin>min) begin
                temp_e=0;
                frac=frac>>1;
            end else begin
                temp_e=0;
                frac=0;
            end
        end
        casex({temp_e[9:0]>=10'h0ff,nrm,nsign})
            4'b0_xx_x:res={sign,temp_e[7:0],frac[25:3]};
            4'b1_00_x:res={sign,8'hff,23'b0};//无穷
            4'b1_01_0:res={sign,8'hfe,23'h7fffff};
            4'b1_01_1:res={sign,8'hff,23'b0};//向负无穷
            4'b1_10_0:res={sign,8'hff,23'b0};//向正无穷
            4'b1_10_1:res={sign,8'hfe,23'h7fffff};
            4'b1_11_x:res={sign,8'hfe,23'h7fffff};//截断
        endcase

    end
end
function [7:0]rom;//隐含位1
    input [3:0]b;
    case (b)
        4'h0:rom=8'hff;
```

```
            4'h1:rom=8'hd4;
            4'h2:rom=8'hc3;
            4'h3:rom=8'haa;
            4'h4:rom=8'h93;
            4'h5:rom=8'h7f;
            4'h6:rom=8'h6d;
            4'h7:rom=8'h5c;
            4'h8:rom=8'h4d;
            4'h9:rom=8'h3f;
            4'ha:rom=8'h33;
            4'hb:rom=8'h27;
            4'hc:rom=8'h1c;
            4'hd:rom=8'h12;
            4'he:rom=8'h08;
            4'hf:rom=8'h00;
        endcase
    endfunction
endmodule
```

Verilog

9.6浮点数开发器FSQRT

9.6.1浮点开方算法

浮点开方算法相较来说比较简单，因为只有一个源操作数：$mathrm = \{mathrm_{mathrm}, mathrm_{mathrm}, mathrm_{mathrm}\}$且d如果是负数的话，结果是Nan；使得讨论情况分为：**规格化正数、非规格化正数、特殊情况（Nan、∞）**

1. 规格化正数

   若d是一个规格化正数，则$sqrt = sqrt$，因为要使用Newton-Raphson计算平方根，所以需要使1.fd→0.1xxxx或者0.01xxxxx的形式，即**将1.fd右移一位或者两位，同时指数+1/+2,平方根结果是0.1xxxx，再左移一位得到1.fq的标准格（不改变阶码）**

   分两种情况：ed-127为偶数、ed-127为奇数

1) 如果 $e_d - 127$ 为偶数，即 e_d 为奇数，右移 $1.f_d$ 两位，$1.f_q = \sqrt{1.f_d >> 2} << 1$，
$e_q = (e_d - 127 + 2)/2 + 127 - 1 = e_d >> 1 + 63 + e_d \% 2$；
2) 如果 $e_d - 127$ 为奇数，即 e_d 为偶数，右移 $1.f_d$ 一位，$1.f_q = \sqrt{1.f_d >> 1} << 1$，
$e_q = (e_d - 127 + 1)/2 + 127 - 1 = e_d >> 1 + 63 + e_d \% 2$。

可以看到无论ed奇偶，结果均是$e_q = e_d > 1 + 63 + e_d\%2, e_d\%2 = d[23]$——这个得到是最终的阶码

2. 非规格化正数

若d是一个非规格化正数，则$sqrt = sqrt$同样因为Newton-Raphson计算平方根，所以需要使0.fd→0.1xxxx或者0.01xxxxx的形式，且指数已为偶数，则尾数移位偶数得到想要的尾数格式计算更为方便。计算得到的结果0.1xxxxx左移一位得到1.fq的标准格式(不改变阶码)，指数$eq = (-126 - b)/2 - 1 + 127 = 63 - b/2$——这个得到是最终的阶码（63可以由规格化的eq得到）

3. 特殊情况

之前提到的若d是负数，则$sqrt = Nan$;若d是+∞，则$sqrt = +infin$;若d是Nan则$sqrt = Nan$

9.6.2 Newton-Raphson浮点开方器实现

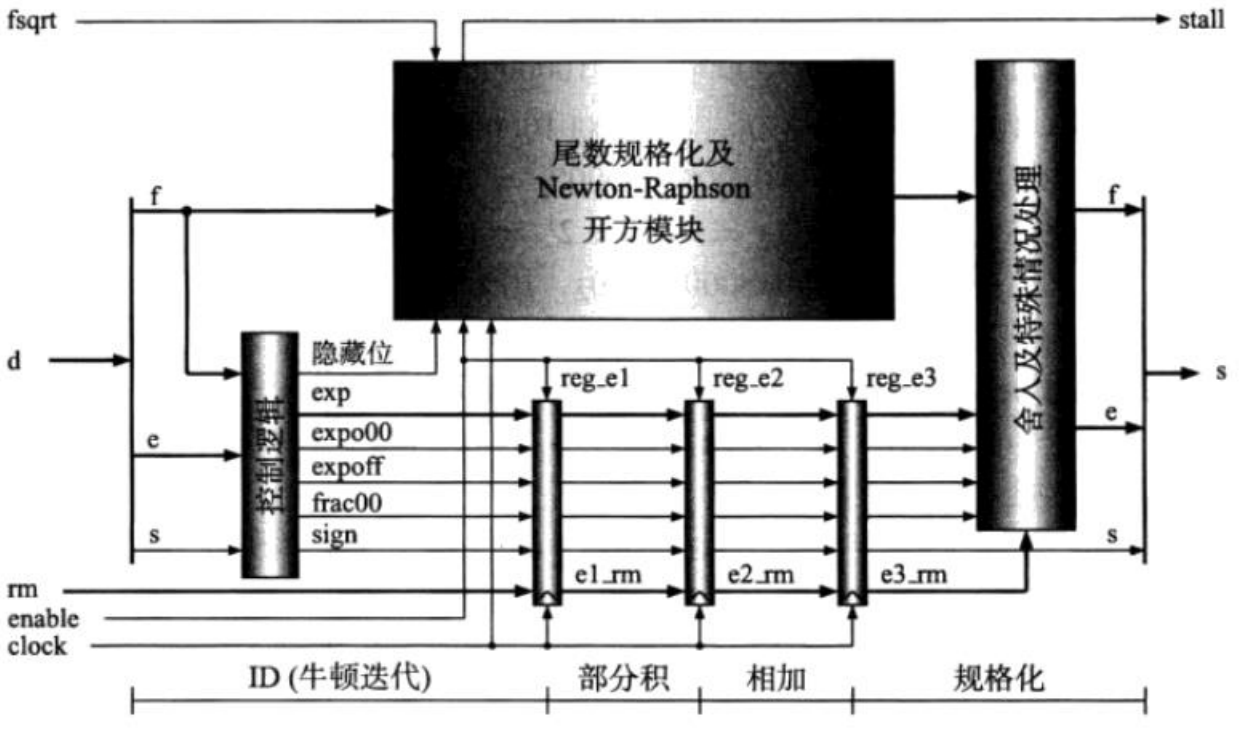

图 9.25 浮点开方电路的总体模块图

📌和采用Newton-Raphson算法的除法器FDIV一样，分为迭代、求积、规格化三步。求积采用Wallace数，故求积分为CSA阵列、求和

迭代方程：$x_{i+1} = x_i(3 - x_i^2 d)/2$，迭代过程中暂停流水线。

图 9.26 是 Newton-Raphson 浮点开方器的流水线示意图。查表得出 x_0 用一个周期，3 次牛顿迭代用 21 个周期。这 22 个周期由 stall 信号暂停流水线。然后是两个周期的乘法和一个周期的规格化。这部分与浮点乘法器类似，用流水线方式实现。将在下面描述如何产生 stall 信号。

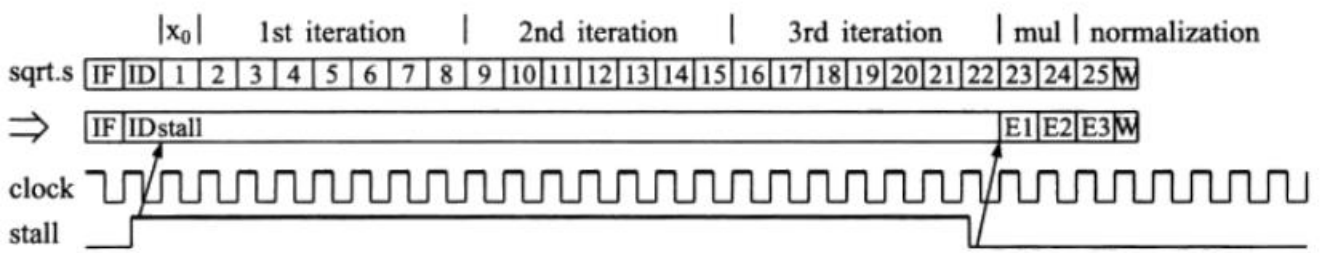

图 9.26　浮点开方指令的流水线

每次迭代需要7个T

当求积时启动流水线——FMUL、FDIV、FSQRT都可以使用乘法的流水线

1. 判断d是否是特殊数：零、Nan、Inf
2. 求临时阶码、求尾数

利用$e_q = e_d >> 1 + 63 + e_d\%2, e_d\%2 = d[23]$求临时阶码；

求尾数，这里尾数已经是0.xxxxxx的形式[注释3]，则若d是规格化数，那么若d[23]是偶数则不改变尾数，否则右移一位尾数0.01xxxx；若d是非规格化数，则调用子模块确定往左移多少偶数位并调整阶码值

3. 利用Newton-Raphson求迭代xn

当ID阶段时count=0，译得指令为fsqrt时，ifsqrt有效，则根据stall=ifsqrt&(count==5'b0|busy)产生stall有效阻塞流水线。且ifsqrt有效count==0使得count==5'b1；当count==5'b1时则装载reg_x，reg_n，此后每一个clk上升沿，count+1；

一次迭代需要做3次Wallace乘法，一次减法（/2可以在减法时移位计算）共需要7个T

则在count=8、15、22时更新reg_x，当count=21时停止stall，当count=22时清零count

4. 利用Wallace计算xn*reg_d得到的sum、c
5. 求和sum、c
6. 规格化——后续都是左移，且exp在之前计算出来后不需要再变动，因此规格化只需要处理尾数即可

wallace24x28

```
`timescale 1ns / 1ps

module wallace24x28(
    input [27:0]a,
    input [23:0]b,

    output [51:0]sum,
    output [51:0]c
);
    reg[51:0]ab[23:0];
    integer i,j;
    always @(*) begin
        for (i= 0; i<24;i=i+1) begin
            for (j = 0; j<28; j=j+1) begin
                ab[i][j]=b[i]*a[j];
            end
            ab[i][51:28]=24'b0;
        end
    end
    wire
[51:0]sum1_1,c1_1,sum1_2,c1_2,sum1_3,c1_3,sum1_4,c1_4,sum1_5,c1_5,sum1_6,c1_6,sum1_7,c1_7,sum1_8,c1_
8;
    csa #(.WIDTH(52))csa_1_1(ab[0],ab[1]<<1,ab[2]<<2,sum1_1,c1_1);
    csa #(.WIDTH(52))csa_1_2(ab[3]<<3,ab[4]<<4,ab[5]<<5,sum1_2,c1_2);
    csa #(.WIDTH(52))csa_1_3(ab[6]<<6,ab[7]<<7,ab[8]<<8,sum1_3,c1_3);
    csa #(.WIDTH(52))csa_1_4(ab[9]<<9,ab[10]<<10,ab[11]<<11,sum1_4,c1_4);
    csa #(.WIDTH(52))csa_1_5(ab[12]<<12,ab[13]<<13,ab[14]<<14,sum1_5,c1_5);
    csa #(.WIDTH(52))csa_1_6(ab[15]<<15,ab[16]<<16,ab[17]<<17,sum1_6,c1_6);
    csa #(.WIDTH(52))csa_1_7(ab[18]<<18,ab[19]<<19,ab[20]<<20,sum1_7,c1_7);
    csa #(.WIDTH(52))csa_1_8(ab[21]<<21,ab[22]<<22,ab[23]<<23,sum1_8,c1_8);

    wire [51:0]sum2_1,c2_1,sum2_2,c2_2,sum2_3,c2_3,sum2_4,c2_4,sum2_5,c2_5;
    csa #(.WIDTH(52))csa_2_1(sum1_1,c1_1<<1,sum1_2,sum2_1,c2_1);
```

```
    csa #(.WIDTH(52))csa_2_2(c1_2<<1,sum1_3,c1_3<<1,sum2_2,c2_2);
    csa #(.WIDTH(52))csa_2_3(sum1_4,c1_4<<1,sum1_5,sum2_3,c2_3);
    csa #(.WIDTH(52))csa_2_4(c1_5<<1,sum1_6,c1_6<<1,sum2_4,c2_4);
    csa #(.WIDTH(52))csa_2_5(sum1_7,c1_7<<1,sum1_8,sum2_5,c2_5);//c1_8

    wire [51:0]sum3_1,c3_1,sum3_2,c3_2,sum3_3,c3_3;
    csa #(.WIDTH(52))csa_3_1(sum2_1,c2_1<<1,sum2_2,sum3_1,c3_1);
    csa #(.WIDTH(52))csa_3_2(c2_2<<1,sum2_3,c2_3<<1,sum3_2,c3_2);
    csa #(.WIDTH(52))csa_3_3(sum2_4,c2_4<<1,sum2_5,sum3_3,c3_3);//c2_5,c1_8

    wire [51:0]sum4_1,c4_1,sum4_2,c4_2;
    csa #(.WIDTH(52))csa_4_1(sum3_1,c3_1<<1,sum3_2,sum4_1,c4_1);
    csa #(.WIDTH(52))csa_4_2(c3_2<<1,sum3_3,c3_3<<1,sum4_2,c4_2);//c2_5,c1_8

    wire [51:0]sum5_1,c5_1,sum5_2,c5_2;
    csa #(.WIDTH(52))csa_5_1(sum4_1,c4_1<<1,sum4_2,sum5_1,c5_1);
    csa #(.WIDTH(52))csa_5_2(c4_2<<1,c2_5<<1,c1_8<<1,sum5_2,c5_2);


    wire [51:0]sum6_1,c6_1;
    csa #(.WIDTH(52))csa_6_1(sum5_1,c5_1<<1,sum5_2,sum6_1,c6_1);//剩c5_2

    wire [51:0]sum7_1,c7_1;
    csa #(.WIDTH(52))csa_7_1(sum6_1,c6_1<<1,c5_2<<1,sum,c);
endmodule
```

Verilog

wallace26x26

```
`timescale 1ns / 1ps

module wallace26x26(
    input [25:0]a,
    input [25:0]b,
```

```
    output [51:0]sum,c
);
    reg[51:0]ab[25:0];
    integer i,j;
    always @(*) begin
        for ( i= 0; i<26;i=i+1) begin
            for (j = 0; j<26; j=j+1) begin
                ab[i][j]=b[i]*a[j];
            end
            ab[i][51:26]=26'b0;
        end
    end
    wire
[51:0]sum1_1,c1_1,sum1_2,c1_2,sum1_3,c1_3,sum1_4,c1_4,sum1_5,c1_5,sum1_6,c1_6,sum1_7,c1_7,sum1_8,c1_
8;
    csa #(.WIDTH(52))csa_1_1(ab[0],ab[1]<<1,ab[2]<<2,sum1_1,c1_1);
    csa #(.WIDTH(52))csa_1_2(ab[3]<<3,ab[4]<<4,ab[5]<<5,sum1_2,c1_2);
    csa #(.WIDTH(52))csa_1_3(ab[6]<<6,ab[7]<<7,ab[8]<<8,sum1_3,c1_3);
    csa #(.WIDTH(52))csa_1_4(ab[9]<<9,ab[10]<<10,ab[11]<<11,sum1_4,c1_4);
    csa #(.WIDTH(52))csa_1_5(ab[12]<<12,ab[13]<<13,ab[14]<<14,sum1_5,c1_5);
    csa #(.WIDTH(52))csa_1_6(ab[15]<<15,ab[16]<<16,ab[17]<<17,sum1_6,c1_6);
    csa #(.WIDTH(52))csa_1_7(ab[18]<<18,ab[19]<<19,ab[20]<<20,sum1_7,c1_7);
    csa #(.WIDTH(52))csa_1_8(ab[21]<<21,ab[22]<<22,ab[23]<<23,sum1_8,c1_8);//ab[24],ab[25]

    wire [51:0]sum2_1,c2_1,sum2_2,c2_2,sum2_3,c2_3,sum2_4,c2_4,sum2_5,c2_5,sum2_6,c2_6;
    csa #(.WIDTH(52))csa_2_1(sum1_1,c1_1<<1,sum1_2,sum2_1,c2_1);
    csa #(.WIDTH(52))csa_2_2(c1_2<<1,sum1_3,c1_3<<1,sum2_2,c2_2);
    csa #(.WIDTH(52))csa_2_3(sum1_4,c1_4<<1,sum1_5,sum2_3,c2_3);
    csa #(.WIDTH(52))csa_2_4(c1_5<<1,sum1_6,c1_6<<1,sum2_4,c2_4);
    csa #(.WIDTH(52))csa_2_5(sum1_7,c1_7<<1,sum1_8,sum2_5,c2_5);
    csa #(.WIDTH(52))csa_2_6(c1_8<<1,ab[24]<<24,ab[25]<<25,sum2_6,c2_6);

    wire [51:0]sum3_1,c3_1,sum3_2,c3_2,sum3_3,c3_3,sum3_4,c3_4;
```

```
    csa #(.WIDTH(52))csa_3_1(sum2_1,c2_1<<1,sum2_2,sum3_1,c3_1);
    csa #(.WIDTH(52))csa_3_2(c2_2<<1,sum2_3,c2_3<<1,sum3_2,c3_2);
    csa #(.WIDTH(52))csa_3_3(sum2_4,c2_4<<1,sum2_5,sum3_3,c3_3);
    csa #(.WIDTH(52))csa_3_4(c2_5<<1,sum2_6,c2_6<<1,sum3_4,c3_4);

    wire [51:0]sum4_1,c4_1,sum4_2,c4_2;
    csa #(.WIDTH(52))csa_4_1(sum3_1,c3_1<<1,sum3_2,sum4_1,c4_1);
    csa #(.WIDTH(52))csa_4_2(c3_2<<1,sum3_3,c3_3<<1,sum4_2,c4_2);//sum3_4,c3_4


    wire [51:0]sum5_1,c5_1,sum5_2,c5_2;
    csa #(.WIDTH(52))csa_5_1(sum4_1,c4_1<<1,sum4_2,sum5_1,c5_1);
    csa #(.WIDTH(52))csa_5_2(c4_2<<1,sum3_4,c3_4<<1,sum5_2,c5_2);

    wire [51:0]sum6_1,c6_1;
    csa #(.WIDTH(52))csa_6_1(sum5_1,c5_1<<1,sum5_2,sum6_1,c6_1);//剩c5_2

    wire [51:0]sum7_1,c7_1;
    csa #(.WIDTH(52))csa_7_1(sum6_1,c6_1<<1,c5_2<<1,sum,c);
endmodule
```

Verilog

wallace26x24

```
`timescale 1ns / 1ps

module wallace26x24(
    input [25:0]a,
    input [23:0]b,
    output [49:0]sum,c
);
    reg[49:0]ab[23:0];
    integer i,j;
```

```
    always @(*) begin
        for ( i= 0; i<24;i=i+1) begin
            for (j = 0; j<26; j=j+1) begin
                ab[i][j]=b[i]*a[j];
            end
            ab[i][49:26]=32'b0;
        end
    end
    wire
[49:0]sum1_1,c1_1,sum1_2,c1_2,sum1_3,c1_3,sum1_4,c1_4,sum1_5,c1_5,sum1_6,c1_6,sum1_7,c1_7,sum1_8,c1_
8;
    csa #(.WIDTH(50))csa_1_1(ab[0],ab[1]<<1,ab[2]<<2,sum1_1,c1_1);
    csa #(.WIDTH(50))csa_1_2(ab[3]<<3,ab[4]<<4,ab[5]<<5,sum1_2,c1_2);
    csa #(.WIDTH(50))csa_1_3(ab[6]<<6,ab[7]<<7,ab[8]<<8,sum1_3,c1_3);
    csa #(.WIDTH(50))csa_1_4(ab[9]<<9,ab[10]<<10,ab[11]<<11,sum1_4,c1_4);
    csa #(.WIDTH(50))csa_1_5(ab[12]<<12,ab[13]<<13,ab[14]<<14,sum1_5,c1_5);
    csa #(.WIDTH(50))csa_1_6(ab[15]<<15,ab[16]<<16,ab[17]<<17,sum1_6,c1_6);
    csa #(.WIDTH(50))csa_1_7(ab[18]<<18,ab[19]<<19,ab[20]<<20,sum1_7,c1_7);
    csa #(.WIDTH(50))csa_1_8(ab[21]<<21,ab[22]<<22,ab[23]<<23,sum1_8,c1_8);

    wire [49:0]sum2_1,c2_1,sum2_2,c2_2,sum2_3,c2_3,sum2_4,c2_4,sum2_5,c2_5;
    csa #(.WIDTH(50))csa_2_1(sum1_1,c1_1<<1,sum1_2,sum2_1,c2_1);
    csa #(.WIDTH(50))csa_2_2(c1_2<<1,sum1_3,c1_3<<1,sum2_2,c2_2);
    csa #(.WIDTH(50))csa_2_3(sum1_4,c1_4<<1,sum1_5,sum2_3,c2_3);
    csa #(.WIDTH(50))csa_2_4(c1_5<<1,sum1_6,c1_6<<1,sum2_4,c2_4);
    csa #(.WIDTH(50))csa_2_5(sum1_7,c1_7<<1,sum1_8,sum2_5,c2_5);//c1_8未计算

    wire [49:0]sum3_1,c3_1,sum3_2,c3_2,sum3_3,c3_3;
    csa #(.WIDTH(50))csa_3_1(sum2_1,c2_1<<1,sum2_2,sum3_1,c3_1);
    csa #(.WIDTH(50))csa_3_2(c2_2<<1,sum2_3,c2_3<<1,sum3_2,c3_2);
    csa #(.WIDTH(50))csa_3_3(sum2_4,c2_4<<1,sum2_5,sum3_3,c3_3);//c2_5,c1_8未计算
```

```
    wire [49:0]sum4_1,c4_1,sum4_2,c4_2;
    csa #(.WIDTH(50))csa_4_1(sum3_1,c3_1<<1,sum3_2,sum4_1,c4_1);
    csa #(.WIDTH(50))csa_4_2(c3_2<<1,sum3_3,c3_3<<1,sum4_2,c4_2);//c2_5,c1_8未计算



    wire [63:0]sum5_1,c5_1,sum5_2,c5_2;
    csa csa_5_1(sum4_1,c4_1<<1,sum4_2,sum5_1,c5_1);
    csa csa_5_2(c4_2<<1,c2_5<<1,c1_8<<1,sum5_2,c5_2);

    wire [63:0]sum6_1,c6_1;
    csa csa_6_1(sum5_1,c5_1<<1,sum5_2,sum6_1,c6_1);//剩c5_2

    wire [63:0]sum7_1,c7_1;
    csa csa_7_1(sum6_1,c6_1<<1,c5_2<<1,sum,c);
endmodule
```

Verilog

移位

```
`timescale 1ns / 1ps

module shift_even(
    input [23:0]a,
    output reg[23:0]out,
    output reg[4:0]amount
);
    always @(*) begin
        out=a;
        amount=0;
        while(out[23]!=1'd1&out[22]!=1'd1&amount<5'd24) begin
            out=out<<2;
            amount=amount+5'd1;
```

```
        end
    end
endmodule
```

Verilog

FSQRT

```
`timescale 1ns / 1ps

module fsqrt_design(
    input clk,clear,
    input [31:0]d,
    input [1:0]rm,
    input ifsqrt,

    output stall,
    output reg busy,
    output reg[4:0]count,
    output reg[31:0]res
);
    //1.判断特殊情况
    wire exp_isZero=~|d[30:23];
    wire exp_isF=&d[30:23];
    wire frac_isZero=~|d[22:0];
    wire isZero=exp_isZero&frac_isZero;
    wire isNeg=d[31];
    wire isInf=exp_isF&frac_isZero;
    wire isNan=exp_isF&~frac_isZero;
    //2.计算暂时的阶码，尾数
    wire [7:0]temp_exp={1'b0,d[30:24]}+8'd63+d[23];
    wire [23:0]tempFrac=exp_isZero?{d[22:0],1'b0}:{1'b1,d[22:0]};
    wire [23:0]temp_frac_=d[23]?{1'b0,tempFrac[23:1]}:tempFrac;//偶数-127为奇数，移1位即可。奇数-127为
```

偶数，移两位

```
    wire [23:0]d_frac;
    wire [4:0]amount;
    shift_even shift_init(temp_frac_,d_frac,amount);
    wire [7:0]exp=temp_exp-{3'b0,amount};
    //3.迭代
    reg [25:0]reg_x;//xx.xxxxx
    reg [23:0]reg_d;//.xxx
    wire [51:0]x_2,sum1,c1;//xi*xi;//xxx.xx
    wire [51:0]x_2d,sum2,c2;//x_2*d;//xxx.xxx
    wire [51:0]_3subx_2d;//xxx.xxxx
    wire [51:0]x52,sum3,c3;//xi(3-xi^2d)
    wallace26x26 w1(reg_x,reg_x,sum1,c1);
    assign x_2=sum1+(c1<<1);
    wallace24x28 w2(x_2[51:24],reg_d,sum2,c2);
    assign x_2d=sum2+(c2<<1);
    assign _3subx_2d=26'h3000000-x_2d[49:24];
    wallace26x26 w3(reg_x,_3subx_2d,sum3,c3);
    assign x52=sum3+(c3<<1);
    always @(posedge clk or posedge clear) begin
        if (clear) begin
            count<=5'b0;
            busy<=1'b0;
        end else begin
            if (count==5'b0&ifsqrt) begin
                busy<=1'b1;
                count<=5'b1;
            end begin
               if (count==5'b1) begin
                    reg_x<={2'b01,rom(d_frac[23:19]),16'b0};
                    reg_d<=d_frac;
               end
```

```
            if (count!=5'b0) count<=count+1'b1;
            if (count==5'h15) busy<=1'b0;
            if (count==5'h16) count<=5'b0;
            if (count==5'h8|count==5'hf|count==5'h16) reg_x=x52[50:25];
        end
    end
end
assign stall=ifsqrt&(count==5'b0|busy);

//4.求sum、c
wire [23:0]w_reg_d;//.xxxxx
wire [25:0]w_reg_x;//xx.xxxxx
wire wisZero,wisNan,wisInf,wisNeg;
wire [7:0]wexp;
wire [1:0]wrm;
reg_design #(.WIDTH(64))r1(clk,~stall,clear,
    {reg_d,reg_x,isZero,isNan,isInf,isNeg,exp,rm},
    {w_reg_d,w_reg_x,wisZero,wisNan,wisInf,wisNeg,wexp,wrm});
    wire [49:0]sum,c;//x.xxxxx
    wallace26x24 w4(w_reg_x,w_reg_d,sum,c);
    //5.求和
    wire cisZero,cisNan,cisInf,cisNeg;
    wire [7:0]cexp;
    wire [1:0]crm;
    wire [49:0]csum,cc;//xx.xxxxxx
    reg_design
#(.WIDTH(114))r2(clk,~stall,clear,{sum,c,wisZero,wisNan,wisInf,wisNeg,wexp,wrm},{csum,cc,cisZero,cis
Nan,cisInf,cisNeg,cexp,crm});
    wire [49:0]q=sum+(c<<1);//xx.xxxxxxx
    //6.规格化
    wire [49:0]nq;//xx.xxxxxx
    wire nisZero,nisNan,nisInf,nisNeg;
```

```
    wire [7:0]nexp;
    wire [1:0]nrm;
    reg_design
#(.WIDTH(64))r3(clk,~stall,clear,{q,cisZero,cisNan,cisInf,cisNeg,cexp,crm},{nq,nisZero,nisNan,nisInf
,nisNeg,nexp,nrm});
    reg [25:0]temp_frac;//xxxxxxx
    reg [7:0]temp_e;
    //对temp_e是做减法，不会上溢
    always @(*) begin
        temp_frac={nq[47:23],|nq[22:0]};
        temp_e=nexp;
        if (clear) begin
            res=0;
        end else if(~stall)begin//当流水线不阻塞时才允许计算
            if (nisZero) begin
                res={nisNeg,8'h0,23'h0};//0
            end else if (nisNan|nisNeg) begin
                res=32'h7fc00000;//负数开方和Nan开方结果均为Nan
            end else if (nisInf) begin
                res={nisNeg,8'hff,23'b0};
            end else begin
                while (~temp_frac[25]&temp_e>1) begin
                        temp_frac=temp_frac<<1;//不用移阶码
                end
                temp_frac=temp_frac<<1;//不用移阶码
                casex({nrm,temp_frac[3],temp_frac[2:0],nisNeg})//隐藏位已设置为1，这里对rgs处理后，不用
再改变隐藏位
                    7'b00_1_100_x:temp_frac=temp_frac+28'b1000;
                    7'b00_x_1xx_x:begin
                        if (temp_frac[2:0]>3'b100) begin
```

```
                        temp_frac=temp_frac+26'b1000;
                    end
                end
                7'b01_x_xxx_1:begin
                    if (temp_frac[2:0]>3'b000) begin
                        temp_frac=temp_frac+26'b1000;
                    end
                end
                7'b10_x_xxx_0:begin
                    if (temp_frac[2:0]>3'b000) begin
                        temp_frac=temp_frac+26'b1000;
                    end
                end
            endcase
            res={nisNeg,temp_e[7:0],temp_frac[25:3]};
        end
    end


    end
    function [7:0] rom; // a rom table: 1/d ^{1/2}
        input [4:0] d;
        case (d)
            5'h08: rom = 8'hff; 5'h09: rom = 8'he1;
            5'h0a: rom = 8'hc7; 5'h0b: rom = 8'hb1;
            5'h0c: rom = 8'h9e; 5'h0d: rom = 8'h9e;
            5'h0e: rom = 8'h7f; 5'h0f: rom = 8'h72;
            5'h10: rom = 8'h66; 5'h11: rom = 8'h5b;
            5'h12: rom = 8'h51; 5'h13: rom = 8'h48;
            5'h14: rom = 8'h3f; 5'h15: rom = 8'h37;
            5'h16: rom = 8'h30; 5'h17: rom = 8'h29;
            5'h18: rom = 8'h23; 5'h19: rom = 8'h1d;
            5'h1a: rom = 8'h17; 5'h1b: rom = 8'h12;
            5'h1c: rom = 8'h0d; 5'h1d: rom = 8'h08;
```

```
            5'h1e: rom = 8'h04; 5'h1f: rom = 8'h00;
            default: rom = 8'hff; // 0 - 7: not be accessed
        endcase
    endfunction
endmodule
```

Verilog

习题

1. 使用高级语言生成任意位的Wallace乘法Verilog代码

```
from collections import deque
def printWallace(a_length,b_length):
    print("""
    module wallace{}x{}(
        input [{}:0]a,
        input [{}:0]b,
        output [{}:0]sum,c
    );
    """.format(a_length,b_length,a_length-1,b_length-1,a_length+b_length-1));
    print("""
        reg[{}:0]ab[{}:0];
        integer i,j;
        always @(*) begin
            for ( i= 0; i<{};i=i+1) begin
                for (j = 0; j<{}; j=j+1) begin
                    ab[i][j]=b[i]*a[j];
                end
                ab[i][{}:{}]={}'b0;
            end
        end
    """.format(a_length+b_length-1,b_length-1,b_length,a_length,a_length+b_length-
1,a_length,b_length))
```

```
    A=deque()
    if(b_length%3==2):
        A.append([b_length-2,0])
        A.append([b_length-1,0])
    if(b_length%3==1):
        A.append([b_length-1,0])
    i=0
    while (i<b_length/3):
        print("""       wire [{}:0]sum{},c{};""".format(a_length+b_length-1,i,i))
        print("""       csa #(.WIDTH({}))csa_{}(ab[{}]<<{},ab[{}]<<{},ab[{}]<<{},sum{},c{});"""
            .format(a_length+b_length,i,i*3,i*3,i*3+1,i*3+1,i*3+2,i*3+2,i,i))
        A.append([i,1])#1不移位
        A.append([i,2])#2移一位
        i=i+1
    while(len(A)!=2):
        print("""       wire [{}:0]sum{},c{};""".format(a_length+b_length-1,i,i))
        a1=A.popleft()
        a2=A.popleft()
        a3=A.popleft()
        str1=""
        str2=""
        str3=""
        if(a1[1]==0):
            str1="ab[{}]<<{}".format(a1[0],a1[0])
        elif (a1[1]==1):
            str1="sum{}".format(a1[0])
        else:
            str1="c{}<<1".format(a1[0])
        if(a2[1]==0):
            str2="ab[{}]<<{}".format(a2[0],a2[0])
        elif (a2[1]==1):
            str2="sum{}".format(a2[0])
```

```python
        else:
            str2="c{}<<1".format(a2[0])
        if (a3[1]==1):
            str3="sum{}".format(a3[0])
        else:
            str3="c{}<<1".format(a3[0])
        print("""       csa #(.WIDTH({}))csa_{}({},{},{},sum{},c{});"""
            .format(a_length+b_length,i,str1,str2,str3,i,i))
        A.append([i,1])#1不移位
        A.append([i,2])#2移一位
        i=i+1

    print("""       assign sum=sum{};""".format(i-1))
    print("""       assign c=c{};""".format(i-1))
    print("""   endmodule""")
printWallace(28,24)
```

Python

2. 试着使用GoldSchmdit做FDIV、FSQRT
3. 使用舍入方式的浮点数和整数之间的转换
4. 设计双精度、浮点数、整数之间的转换

[注释1] 原码+2^(n-1)-1,n为二进制位数

[注释2] 非规格化数指数位是-126

[注释3] 小数点位于最左端