

# 浙江大学

## 本科实验报告

课程名称: 计算机组成

姓 名: 应周骏

学 院: 计算机科学与技术学院

系: 计算机科学与技术系

专 业: 计算机科学与技术

学 号: 3200103894

指导教师: 马德

2022 年 4 月 30 日

# 浙江大学实验报告

课程名称：\_\_\_\_计算机组成\_\_\_\_实验类型：\_\_\_\_综合\_\_\_\_

实验项目名称：\_\_\_\_复杂操作实现\_\_\_\_

学生姓名：\_\_\_\_应周骏\_\_\_\_专业：\_\_\_\_计算机科学与技术\_\_\_\_学号：\_\_\_\_3200103894\_\_\_\_

同组学生姓名：\_\_\_\_无\_\_\_\_指导老师：\_\_\_\_马德\_\_\_\_

实验地点：\_\_\_\_东 4-509\_\_\_\_实验日期：\_\_\_\_2022\_\_\_\_年\_\_\_\_3\_\_\_\_月\_\_\_\_21\_\_\_\_日

## 一、实验目的和要求

1. 复习二进制加减、乘除的基本法则
2. 掌握补码的基本原理和作用
3. 了解浮点数的表示方法及加法运算法则
4. 进一步了解计算机系统的复杂运算操作

## 二、实验内容和原理

### 目标：

熟悉二进制原码补码的概念，了解二进制加减乘除的原理，掌握浮点加法的操作实现。

### 内容：

1. 设计实现乘法器；
2. 设计实现除法器；
3. 设计实现浮点加法器；

### 原理：

#### 1. 设计实现乘法器

因为是两个  $N$  位数相乘，所以结果应该是四个数加和得到的。先判断  $y$  的最低位是 0 还是 1，如果是 1，则需要把  $x$  加到部分积上，若为 0，则需要把 0 加到部分积上（实际上加 0 的这个过程计算机并不执行，因为加 0 对部分积没有

任何影响)，x 左移一位，之后再让 y 右移一位，若 y 为 0，则循环结束，否则继续此循环过程，一共循环 N 次。流程图和具体架构如下。

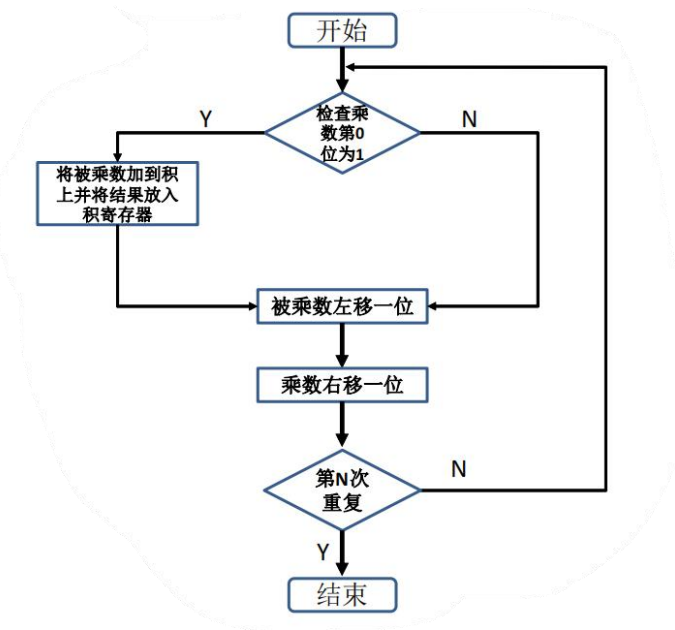


图 2.1.1 N 位乘法流程图

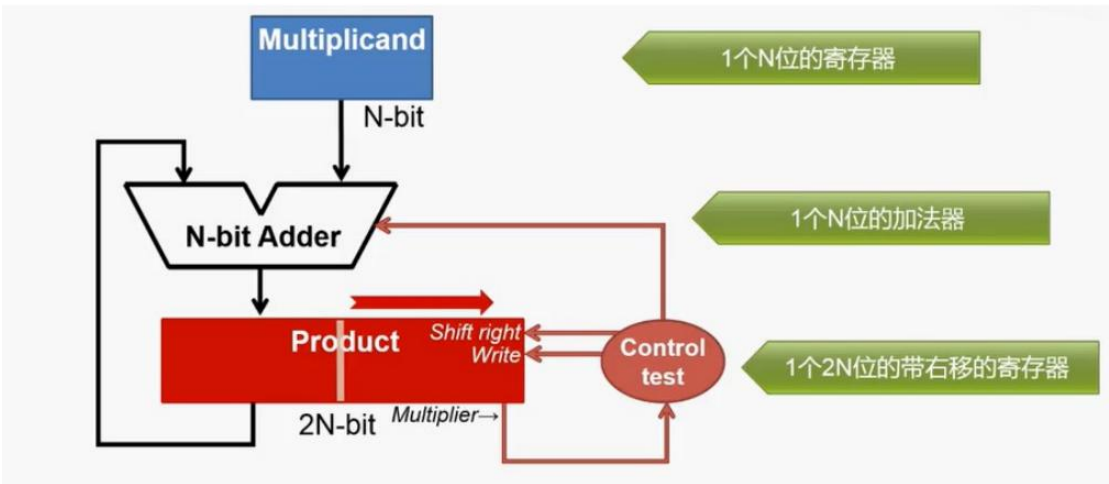


图 2.1.2 N 位乘法器设计架构

2. 除法器的实现

被除数 x 为 1001010，除数 y 为 1000，下面的除法过程是手工运算的一个步骤，而计算机在做除法时就是模拟手工运算的执行过程，具体计算机流程图如下。

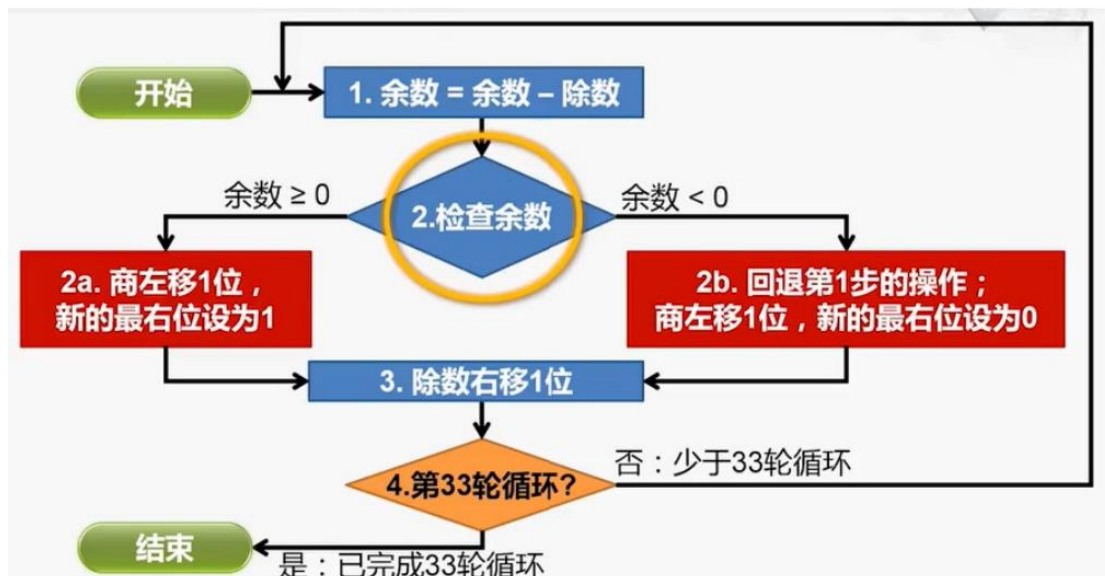


图 2.2.1 32 位除法流程图

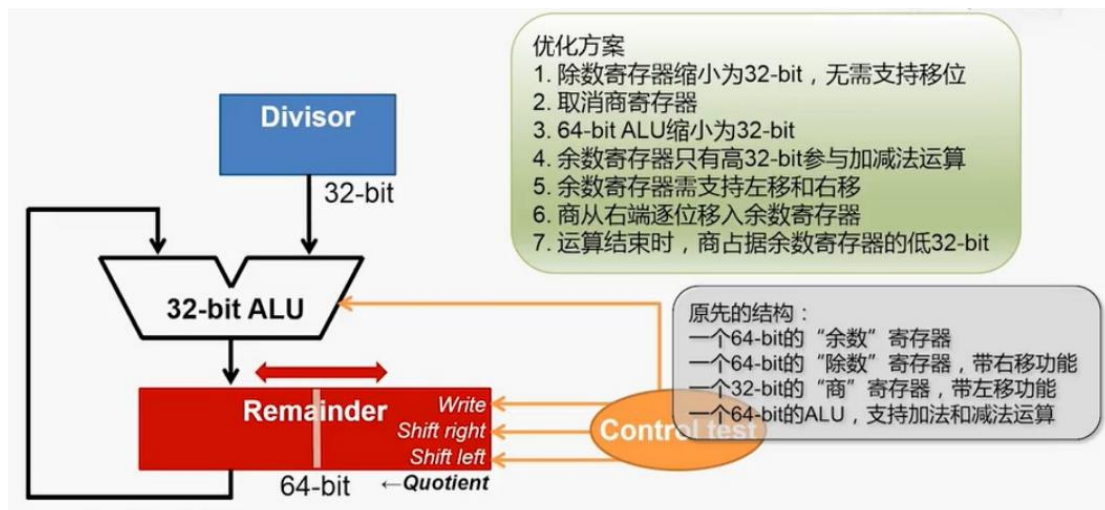


图 2.2.2 32 位除法流程图

### 3. 浮点数加法

完成浮点数加减运算的操作过程大体分为:

1. 0 操作数的检查;
2. 比较阶码大小并完成对阶;
3. 尾数进行加或减运算;
4. 结果规格化;
5. 舍入处理;
6. 溢出处理;

在舍入处理中, 本次实验我采用的是 0 舍 1 入。

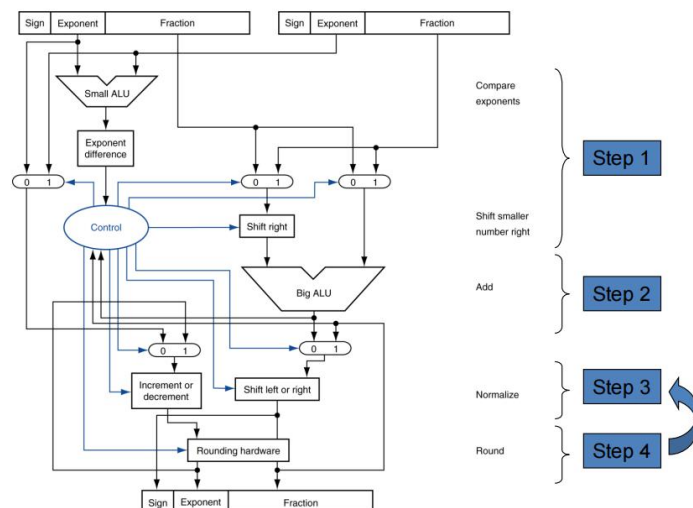


图 2.3.1 浮点数加减法硬件实现

### 三、实验过程和数据记录

#### 实验一：乘法器设计

##### 1. 工程文件建立

新建工程文件，命名为“OxExp03\_mul32”。

##### 2. 乘法模块设计

新建 Verilog 文件“mul32.v”，输入以下代码。

```

module mul32(
    input clk,
    input rst,
    input [31:0] multiplicand,
    input [31:0] multiplier,
    input start,
    output reg[63:0] product,
    output finish
);
    integer cnt = 0;
    wire [63:0] mul;
    reg fin;

    assign finish = fin;
    assign mul = (multiplicand, 32'h0);
    always @(posedge clk or posedge start or rst) begin
        if(rst) begin
            product <= 64'h0;
            fin <= 1'b0;
        end
        else if(start) begin
            product[31:0] <= multiplier;
            product[63:32] <= 32'h0;
            cnt <= 0;
            fin = 1'b0;
        end
        else begin
            if(cnt<32)begin
                if(multiplier[cnt] == 1'b1)
                    product <= (product + mul)>>1;
                else
                    product <= product >>1;
                //product <= product >> 1;
                cnt <= cnt+1;
                fin <= 1'b0;
            end
            else fin <= 1'b1;
        end
    end
endmodule

```

图 3.1.1 乘法器结构化描述（详见附件）

##### 3. 仿真

对乘法器进行仿真，输入仿真代码如下。

```

23 module mul32_tb();
24     reg clk;
25     reg rst;
26     reg[31:0] multiplicand;
27     reg[31:0] multiplier;
28     reg start;
29     wire[63:0] product;
30     wire finish;
31     initial begin
32         clk = 0;
33         rst = 1;
34         multiplicand = 0;
35         multiplier = 0;
36         start = 0;
37         #100
38         rst = 0;
39         start = 1;
40         multiplicand = 32'd2;
41         multiplier = 32'd3;
42         #350
43         start = 0;
44         #350
45         start = 1;
46         multiplicand = 32'd10;
47         multiplier = 32'd8;
48
49         #350
50         start = 0;
51         #350
52         start = 1;
53         multiplicand = 32'd9;
54         multiplier = 32'd9;
55         #350
56         start = 0;
57         #350
58         start = 1;
59         multiplicand = 32'd50;
60         multiplier = 32'd6;
61         #350
62         start = 0;
63         #350
64         start = 1;
65         multiplicand = 32'd6;
66         multiplier = 32'd60;
67         #350
68         start = 0;
69         #4000 $finish();
70     end
71     always #5 clk = ~clk;
72
73     mul32 mul32_u(
74         clk, rst, multiplicand, multiplier, start, product, finish
75     );
76 endmodule

```

图 3.1.2 乘法器仿真代码

得到仿真结果图如下：





图 3.1.3 乘法器仿真结果图

## 实验二：除法器设计

### 1.工程文件建立

新建工程文件“OxExp03\_div32”。

### 2. 除法器模块

新建 Verilog 文件“div32.v”，输入如下代码。

```

module div32(
    input clk,
    input rst,
    input start,
    input [31:0] dividend,
    input [31:0] divisor,
    output [31:0] quotient,
    output [31:0] remainder,
    output reg finish
);

    reg [63:0] rem;
    wire [63:0] div;
    integer cnt = 0;
    assign div = {divisor, 32'h0};
    assign quotient = rem[31:0];
    assign remainder = rem[63:32];

    always@(posedge clk or start or rst) begin
        if(rst) begin
            rem <= 64'h0;
            finish <= 1'b0;
            cnt <= 0;
        end
        else if(start) begin
            if(cnt == 0) begin
                rem <= {31'b0, dividend, 1'b0};
                finish <= 1'b0;
                cnt <= cnt+1;
            end
            else if(cnt < 33) begin
                if(rem < div) begin
                    rem <= rem << 1;
                end
                else if(rem >= div) begin
                    rem = (rem-div)<<1;
                    rem = rem + 1;
                end
                finish <= 1'b0;
                cnt <= cnt+1;
            end
            else begin
                rem[63:32] = rem[63:32] >> 1;
                finish <= 1'b1;
                cnt <= 0;
            end
        end
    end
end

```

图 3.2.1 除法器代码

对该模块进行仿真，输入如下仿真代码。

```

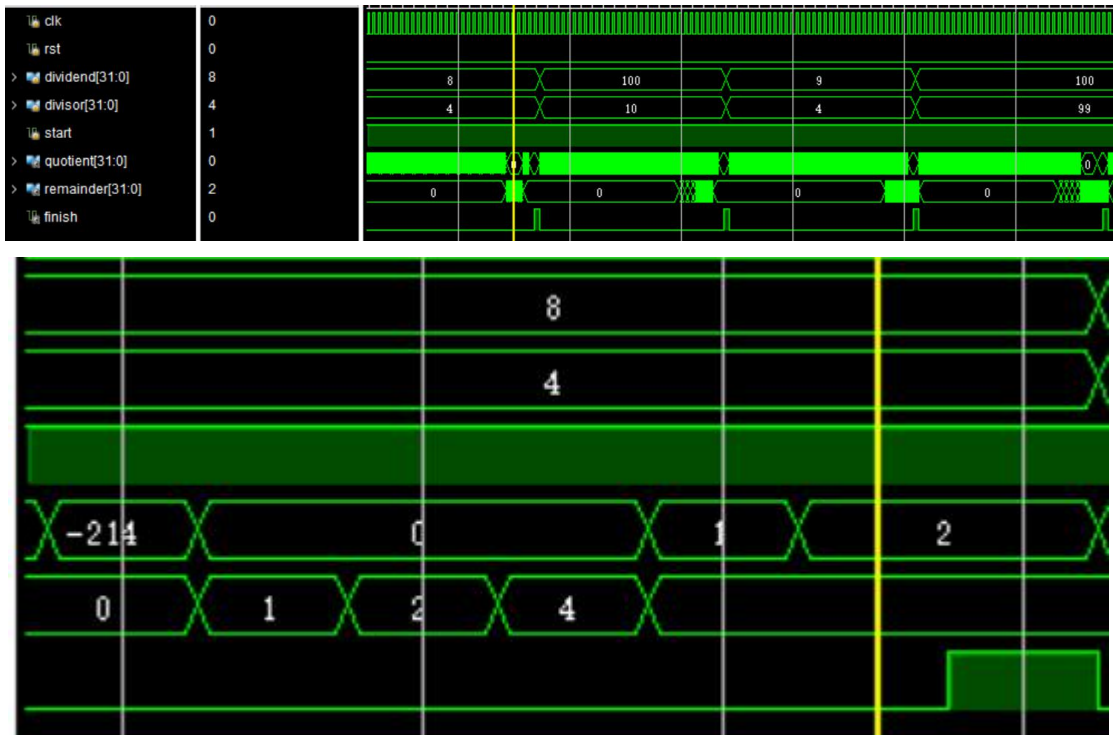
initial begin
    clk = 0;
    rst = 1;
    start = 0;
    #10
    rst = 0;
    start = 1;
        dividend = 32'd8;
        divisor  = 32'd4;
        #335
        dividend = 32'd100;
        divisor  = 32'd10;
        #335
        dividend = 32'd9;
        divisor  = 32'd4;
        #340
        dividend = 32'd100;
        divisor  = 32'd99;
        #350 $stop();

end

```

图 3.2.2 除法器仿真代码

得到仿真结果如下。





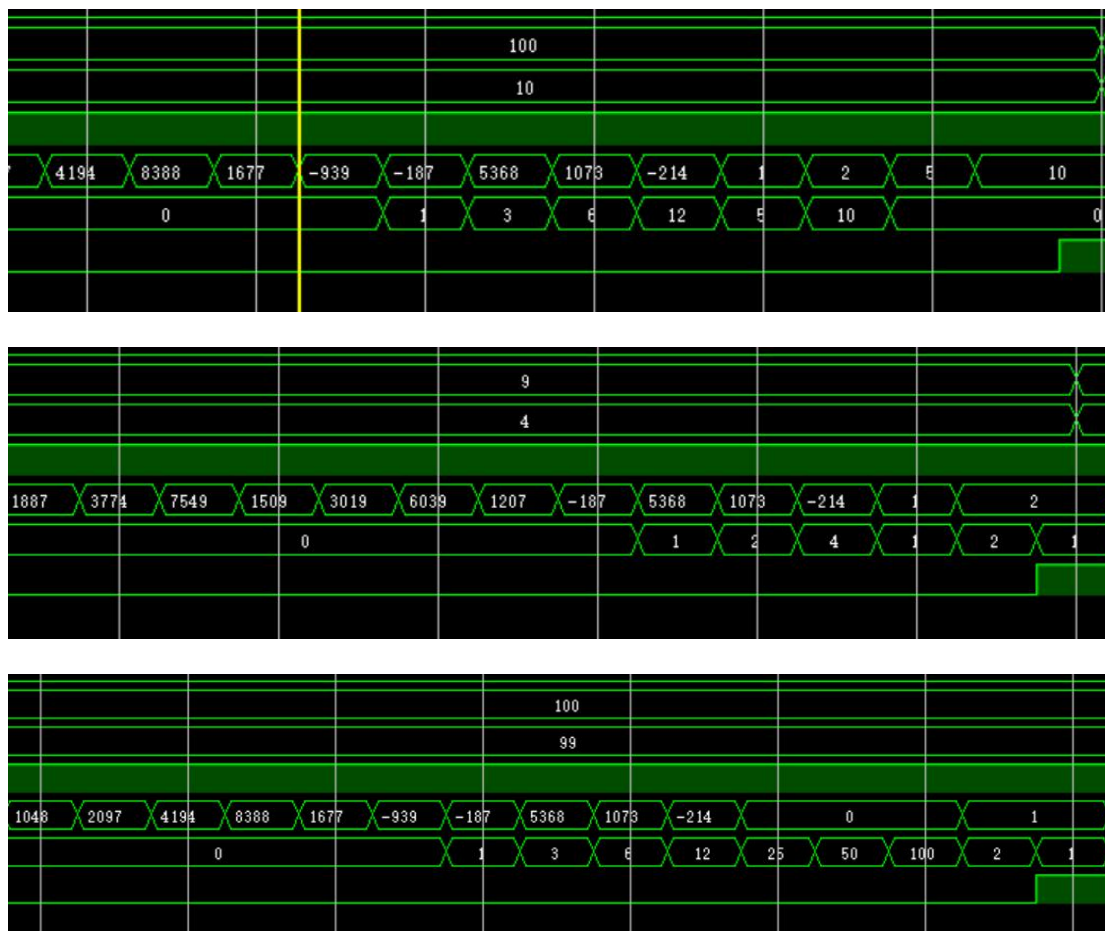


图 3.2.3 除法器仿真结果图

## 实验三：浮点数加减法

### 1. 工程文件建立

新建工程文件，命名“OxExp03\_floatadd”。

### 2. 状态机设计

新建 Verilog 文件“float\_add32.v”，输入代码(由于过长，附在附件中，此处截取部分，后面代码分析进一步展开)。

```
//000-pre 001-if zero? 010-normalize 011-add 100-dealwithalignment 101-finish
always @(posedge clk) begin
    case(state)
        3'b000: begin
            fin <= 0;
            signa <= A[31];
            signb <= B[31];
            expa <= A[30:23];
            expb <= B[30:23];
            fracA <= A[22:0];
            fracb <= B[22:0];
            state <= 3'b001;
        end
        3'b001: begin
            if(A == 0) begin
                sign <= signb;
                exp <= expb;
            end
        end
    endcase
end
```

图 3.3.1 浮点数加法器代码

对该原理图进行仿真，输入如下仿真代码。

```
always #10 clk = ~clk;
initial begin
rst = 1'b1;
clk = 1'b0; //Testbench采用固定输入激励以便观察结果
en = 1'b0;
A = 32'b0;
B = 32'b0;
#10
rst = 1'b0;
en = 1;
A=32'hc0a00000; //-5.0
B=32'hc0900000; //-4.5
//c1180000(-9.5)
#80
A=32'h40a00000; //+5.0
B=32'h40900000; //+4.5
#1000 $stop(); //41180000(+9.5)
end
```

图 3.3.2 浮点数加法仿真代码

得到仿真结果。

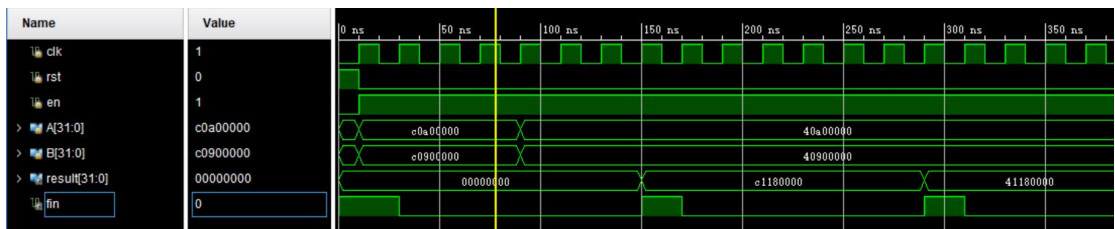


图 3.3.3 浮点数加法仿真结果图

## 四、实验结果分析

### 实验一：乘法器设计

#### 1. 乘法器代码分析

通过下述代码，对复位信号和开始信号做出响应，为后续算数实现做好准备。

```

if(rst) begin
product <= 64'h0;
fin <= 1'b0;
end
else if(start) begin
product[31:0] <= multiplier;
product[63:32] <= 32'h0;
cnt <= 0;
fin = 1'b0;
end

```

复位  
load操作数  
初始化product

图 4.1.1 乘法器初始部分

```

else begin
if(cnt<32)begin
if(multiplier[cnt] == 1'b1)
product <= (product + mul)>>1;
else
product <= product >>1;
//product <= product >> 1;
cnt <= cnt+1;
fin <= 1'b0;
end
else fin <= 1'b1;
end
end

```

为1时  
直接移位

图 4.1.2 乘法算术操作模拟

上述代码模拟实现了乘法操作，具体过程与乘法的流程对应。

## 2.乘法器仿真结果分析

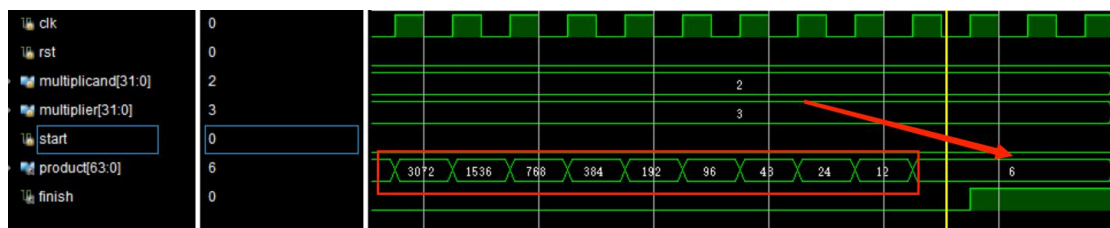


图 4.1.3 乘法器仿真结果分析

在仿真时，能够显示乘法器工作流程，并且结果也和预期相同，此处仅列出一组，其余不再赘述。

## 实验二：除法器设计

### 1. 代码分析

```

if(rst) begin
    rem <= 64'h0;
    finish<= 1'b0;
    cnt <= 0;
end

```

图 4.2.1 除法器复位

```

else if(start) begin
    if(cnt == 0) begin
        rem <= {31'b0, dividend, 1'b0};
        finish <= 1'b0;
        cnt <= cnt+1;
    end
    else if(cnt < 33) begin
        if(rem < div) begin
            rem <= rem << 1;
        end
        else if(rem >= div) begin
            rem = (rem-div)<<1;
            rem = rem + 1;
        end
        finish <= 1'b0;
        cnt <= cnt+1;
    end
    else begin
        rem[63:32] = rem[63:32] >> 1;
        finish <= 1'b1;
        cnt <= 0;
    end
end

```

初始赋值

减去div为负 只移位

否则减后移位

最后右移一位收尾

图 4.2.2 除法器操作流程

按照除法器流程图，编写如上代码，可以实现预期功能。

## 2. 仿真结果分析



图 4.2.3 除尽

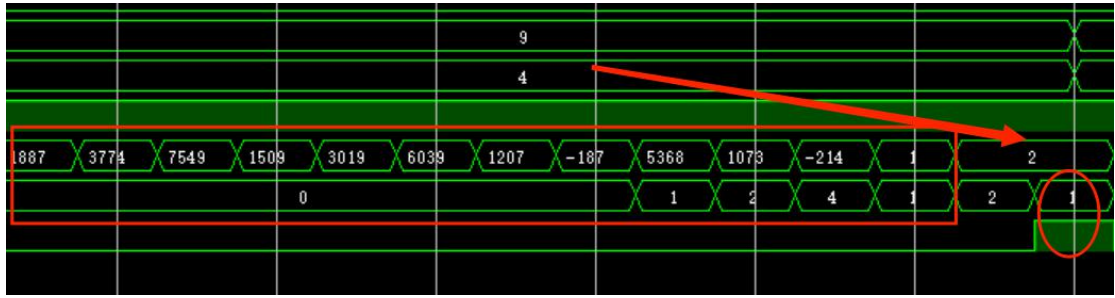


图 4.2.4 除不尽有余数

可以显示除法器运行细节，且能够正确输出除尽和除不尽有余数情况的结果，符合实验预期。

### 实验三：浮点数加法器设计

#### 1. 浮点数加法状态机及代码分析

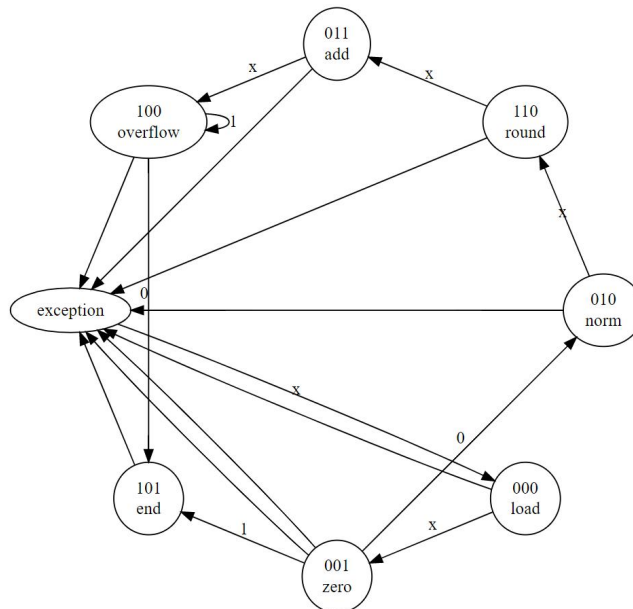


图 4.3.1 浮点数加法器状态机

依据设计的状态机，编写相应的 verilog 代码。状态 1 编码为 000，功能为将操作数进行分解，暂时存储在寄存器中。

```

always @(posedge clk) begin
    case(state)
    3'b000: begin
        fin <= 0;
        signa <= A[31];
        signb <= B[31];
        expa <= A[30:23];
        expb <= B[30:23];
        fracb <= A[22:0];
        fracb <= B[22:0];
        state <= 3'b001;
    end

```

图 4.3.2 状态 1 加载和分解操作数

状态 2 编码为 001，用于判断操作数中是否有 0，如果有 0 则直接转到结束，节省时间和资源。

```

3'b001: begin
    if(A == 0) begin
        sign <= signb;
        exp <= expb;
        frac = fracb;
        state <= 3'b101;
    end
    else
        state <= 3'b010;
    if(B == 0) begin
        sign <= signa;
        exp <= expa;
        frac = fracb;
        state <= 3'b101;
    end
    else
        state <= 3'b010;

```

图 4.3.3 状态 2 0 操作数判定

状态 3 编码为 010，用于判断两个操作数是同号还是异号，并且比较指数大小，确定对齐位数和结果的符号。

```

3'b010:begin
    if(signa != signb)
        sign_same <= 0;
    else
        sign_same <= 1;

    if(expa > expb) begin
        exp <= expa;
        sign <= signa;
        exp_dif <= expa-expb;
        fra_max <= {2'b01, fracb};
        fra_min <= {2'b01, fracb};
    end

    else if(expa < expb) begin
        exp <= expb;
        sign <= signb;
        exp_dif <= expb-expa;
        fra_max <= {2'b01, fracb};
        fra_min <= {2'b01, fracb};
    end

    else begin
        exp <= expa;
        exp_dif <= 0;
        if(fracb > fracb) begin
            sign <= signa;
            fra_max <= {2'b01, fracb};
            fra_min <= {2'b01, fracb};
        end
        else begin
            sign <= signb;
            fra_max <= {2'b01, fracb};
            fra_min <= {2'b01, fracb};
        end
    end
    state <= 3'b110;
end

```

比较指数大小

若指数相等

图 4.3.4 状态 3 同号判断和对齐

状态 4 编码 110，暂时存储舍去部分的最高位，并进行对齐操作。

```

3'b110: begin
    fra_min <= fra_min >> exp_dif;
    temp <= fra_min >> (exp_dif-1);
    state <= 3'b011;
end

```

暂存舍去部分最高位

图 4.3.5 状态 4 舍入位暂存

状态 5 编码 011，用于判断是否需要舍入，并完成浮点数加减。

```

3'b011: begin
    if(sign_same) begin //舍入
        if(round)
            frac <= fra_max + fra_min+1'b1;
        else
            frac <= fra_max + fra_min;
        end
    else
        if(round)
            frac <= fra_max - fra_min-1'b1;
        else
            frac <= fra_max - fra_min;
        state <= 3'b100;
    end
end

```

temp[0]



图 4.3.6 状态 5 舍入确认和加法

状态 6 编码 100，用于溢出判断，若上溢，则对指数进行加一调整；下溢，则持续该状态直到移位到符合要求的指数。

```
3'b100: begin
    if(frac[24] == 1'b1) begin
        exp <= exp + 1;
        frac <= {1'b0, frac[24:1]};
        state <= 3'b101;
    end
    else if(frac[24:23] == 1'b00) begin
        exp <= exp-1;
        frac <= {frac[23:0], 1'b0};
        state <= 3'b100;
    end
    else
        state <= 3'b101;
    end
end
```

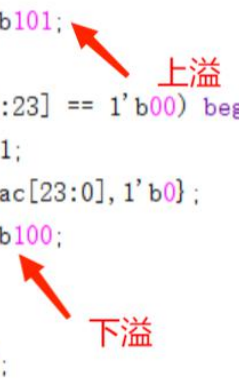


图 4.3.7 状态 6 溢出判断

状态 7 编码 101，结束后返回状态 1，并设置 fin 信号。

```
3'b101:begin
    fin <= 1;
    state <= 3'b000;
end
default: state <= 3'b000;
endcase
end
```

图 4.3.8 状态 7 结束判断

下面一段代码对 fin 信号进行响应，设置结果。



```

always @(posedge fin or posedge rst or negedge rst) begin
    if(en) begin
        res <= {sign, exp, frac[22:0]};
    end
    else if(rst) begin
        res <= 0;
        fin <= 1'b1;
    end
    else
        res <= res;
end

```

图 4.2.9 结果存储

## 2. 浮点数加法仿真分析

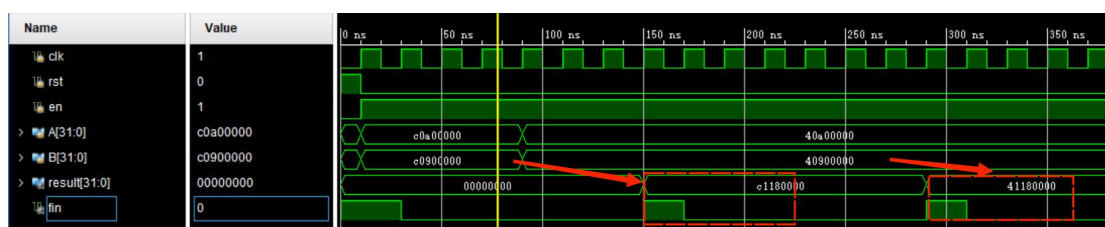


图 4.3.2 浮点数加法仿真结果分析

对于该段仿真，我们可以看到结果产生和输入改变之间存在一定的时延，因为浮点数的状态机产生结果需要多个周期，同时可以看到结果符合预期。

## 五、讨论与心得

1. 通过本次实验，我对基本算数操作（乘法、除法、浮点数加减）的硬件实现有了更全面的了解，能够更清晰地理解操作的各个流程。同时，从最初较多资源浪费的方法到后续优化方案的演进中，我也感受到设计者的巧妙思路。

2. 本次实验中，由于浮点数是选做实验，本来想放弃，但是最后还是坚持下来完成了该项实验，并通过这个实验，对较复杂状态机的编写进行了巩固和提高，有助于 Verilog 代码能力的进一步突破。

3. 本次实验整体相对顺利，实现了课堂内容和实验的交互以及互相促进。