基于 IEEE754 标准的 64 位二级流水线的乘法器设计

撰写人：杨登天-202028015926089

目录

基于 IEEE754 标准的 64 位二级流水线的乘法器设计

杨登天-202028015926089-微电子研究所

一、任务设计要求

任务要求：请设计一个 2 级流水的符合 IEEE 标准的浮点乘法器并用 RTL 实现并仿真

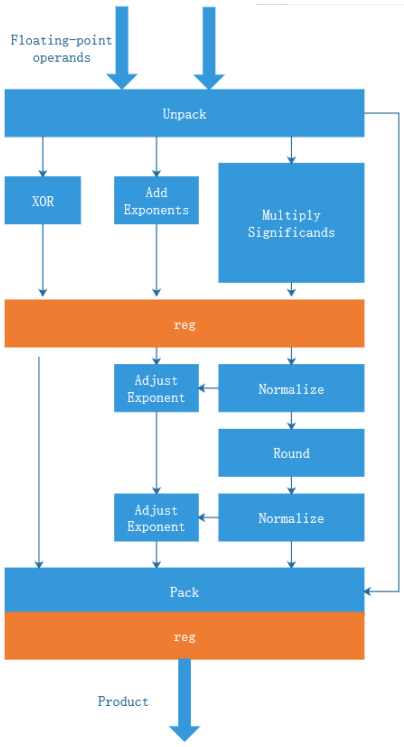
要求 1、加法操作可以直接用+号，乘法操作可以直接用*号实现；

要求 2、舍入操作根据一个输入信号 round_cfg 配置决定采用就近舍入还是 chopping

$$if\ round\_cfg\ == 0, \quad then\ chopping;$$

$$if\ round\_cfg\ == 1, \quad then\ round\ to\ near$$

要求 3、2 级流水线的位置如下所示



二、解答步骤一——64 位 RTL 分析

First level of pipeline
always @(posedge clk or negedge reset_n)
begin
reset_n 低有效的情况——将结果写入 3 个寄存器，符号位、指数位、底数位
reset_n 高无效的情况——将结果写入 3 个寄存器，符号位、指数位、底数位
end

```
Second level of pipeline
always @(posedge clk or negedge reset_n)
    begin
      reset_n 低有效的情况——将结果输出到 Pack 寄存器并输出失败信号
      reset_n 高无效的情况
            begin
            //****处理特殊情况
            if (multiplier1 == 1)  将 multiplier2 作为输出结果
            if (multiplier1 == -1 )  将-1*multiplier2 作为输出结果
            if (multiplier2 == 1)  将 multiplier1 作为输出结果
            if (multiplier2 == -1 )  将-1*multiplier1 作为输出结果
            //****处理正常情况

              //** step1 Normalize
               //***  考虑到两个 53 位 significand 相乘得到 106 位的结果
              if (significand_product(106) == 1)
                  significand_product << significand_product;
                  exponent ← exponent + 1;   //*** 13 位 exponent
                  //**** 55 位 st1_significand_product，高两位存储 01，即小数点
                  //****  前面部分的整数，中间 52 位存储 significand_product 的高
                  //**** 52 位，最低位存储舍入最高位
                  st1_significand_product [54:53] = 01;
                  st1_significand_product [52:1] = signficant_product[104:53];
                  st1_significand_product [0] = signficant_real[52];
              else (significand_product(106) == 0)
                  significand_product ← significand_product;
                  exponent ← exponent;
                  st1_significand_product [54:53] = 01;
                  st1_significand_product [52:1] = signficant_product[103:52];
                  st1_significand_product [0] = signficant_real[51];
               //** step2 Round
                  //**** 52 位 st2_significand_product，存储积的底数位
                  if(chopping)
                     st2_significand_product ← st1_significand_product [52:1];
                  if(rounding)
                     //****  判断被舍去的最高位 st1_significand_product [0]是否为 1
                    if(st1_significand_product [0] == 0)
                      st2_significand_product ← st1_significand_product [52:1];
                    else
                     st1_significand_product [54:1] ← st1_significand_product [54:53]+1;
                     //***  再次判断 st1_significand_product[54]是否为 1
                     if (st1_significand_product[54]==1)
                        exponent ← exponent + 1;
                        st2_significand_product ← st1_significand_product [53:2];
```

```
            else
               exponent ← exponent;
               st2_significand_product ← st1_significand_product [52:1];

        //** step3 Normalize
      //**** 判断指数是否溢出
      if(exponent[12:11] == 00)  指数未溢出;
      else  指数溢出;

     以寄存器存储方式输出结果;
```

三、解答步骤二——设计与验证

3.1 步骤一、编辑方式和编辑工具

按照上述伪代码的方式用 notepad++编辑 Verilog HDL。

3.2 步骤二、验证方式和验证工具

验证方式采用 vivado 内置仿真工具分析。（design 和 test 程序见附录）

3.3 步骤三、例子(一般情况和特殊情况——输入之一为 1 或者-1)

特殊情况是考虑到如果输入的数字之一是+1 或者－1 那么无需经过二级流水线直接输出结果。

接下来的分析是针对验证，因为进制问题不直观，所以分析以下 6 个例子在不同约化条件下判断结果是否溢出

在进行结果验证之前，设定 test 例子

1) $ina1 = 64'h4008\_0000\_0000\_0000$，$inb1 = 64'hC008\_0000\_0000\_0000$
   那么$ina1 = +2^1 \times (1.1)_2 = 3$，$inb1 = -2^1 \times (1.1)_2 = -3$
   结果应该是$result1 = -9 = -2^3 \times (1.001)_2$
   不管采用 chopping 还是 rounding，
   经过 64 位规格化处理之后，$\boldsymbol{result1 = 64'hC022\_0000\_0000\_0000}$
   结果没有溢出。
2) $ina2 = 64'hBFF8\_0000\_0000\_0000$，$inb2 = 64'hBFF8\_0000\_0000\_0000$
   那么$ina2 = -2^0 \times (1.1)_2 = -1.5$，$inb2 = -2^0 \times (1.1)_2 = -1.5$
   结果应该是$result2 = -2.25 = -2^1 \times (1.001)_2$
   不管采用 chopping 还是 rounding
   经过 64 位规格化处理之后，$\boldsymbol{result2 = 64'h4002\_0000\_0000\_0000}$
   结果没有溢出。
3) $ina1 = 64'hBFF8\_0000\_0000\_0000$，$inb1 = 64'h3FF0\_0000\_0000\_0000$
   那么$inb1 = 1$，$ina1 = -1.5$
   结果应该是$\boldsymbol{result1 = -1.5 = 64'hBFF8_0000_0000_0000}$
   不管采用 chopping 还是 rounding
   结果没有溢出。——特殊例子
4) $ina2 = 64'hBFF8\_0000\_0000\_0000$，$inb2 = 64'hBFF8\_0000\_0000\_0000$

那么$inb2=-1$，$ina2=-1.5$

结果应该是$result2=1.5=64'h3FF8_0000_0000_0000$

不管采用 chopping 还是 rounding

结果没有溢出。——特殊例子

5) $ina1=64'hBFF8\_0000\_0000\_0000$，$inb1=64'h3FF0\_0000\_0000\_0000$

那么$ina1=-2^0\times(1.1)_2=-1.5$，$inb1=1$

结果应该是$result1=-1.5=-2^0\times(1.1)_2$

不管采用 chopping 还是 rounding

经过 64 位规格化处理之后，$result3=64'hBFF8\_0000\_0000\_0000$

结果没有溢出。

6) $ina2=64'h0000\_0000\_0000\_0000$，$inb2=64'h0000\_0000\_0000\_0000$

那么$ina2=2^{-1023}\times(1.0)_2$，$inb2=2^{-1023}\times(1.0)_2$

结果应该是$result2=2^{-2046}\times(1.0)_2$

不管采用 chopping 还是 rounding

经过 64 位规格化处理之后，结果都是溢出。

7) $ina1=64'h7FFF\_FFFF\_FFFF\_FFFF$，$inb1=64'h7FFF\_FFFF\_FFFF\_FFFF$

那么

$$ina1=2^{1022}\times(1.F\_FFFF\_FFFF\_FFFF)_2$$
$$inb1=2^{1022}\times(1.F\_FFFF\_FFFF\_FFFF)_2$$

采用 rounding

经过 64 位规格化处理之后，溢出。

8) $ina2=64'h7FFF\_FFFF\_FFFF\_FFFF$，$inb2=64'h7FFF\_FFFF\_FFFF\_FFFF$

那么

$$ina2=2^{1022}\times(1.F\_FFFF\_FFFF\_FFFF)_2$$
$$inb2=2^{1022}\times(1.F\_FFFF\_FFFF\_FFFF)_2$$

采用 chopping

经过 64 位规格化处理之后，溢出。

四、解答步骤三——结果分析

说明 1：defectornon_signal 是数据输出有效/无效信号——也就是无溢出/溢出信号，低有效（无溢出），高无效（有溢出）。

说明 2：一点遗憾是没有找到能够使得输入数据相同，但是在两种不同的约化条件下产生不同未溢出结果的例子。但任务二-步骤二的 5)和 6)举例了同输入在两种约化条件下的溢出结果。

说明 3：ina1 和 inb1 相乘的结果观察 out1，ina2 和 inb2 相乘的结果观察 out2，若数据溢出或者 reset_n 有效时，out 输出信号在本程序中被设置为 64 位 x

说明 4：ina1、inb1、round_cfg1、out1、defectornon_signal1 都是第一个调用模块的参数，ina2、inb2、round_cfg2、out2、defectornon_signal2 都是第二个调用模块的参数。

4.1 步骤一、通过 vivado 内置仿真工具得到仿真图

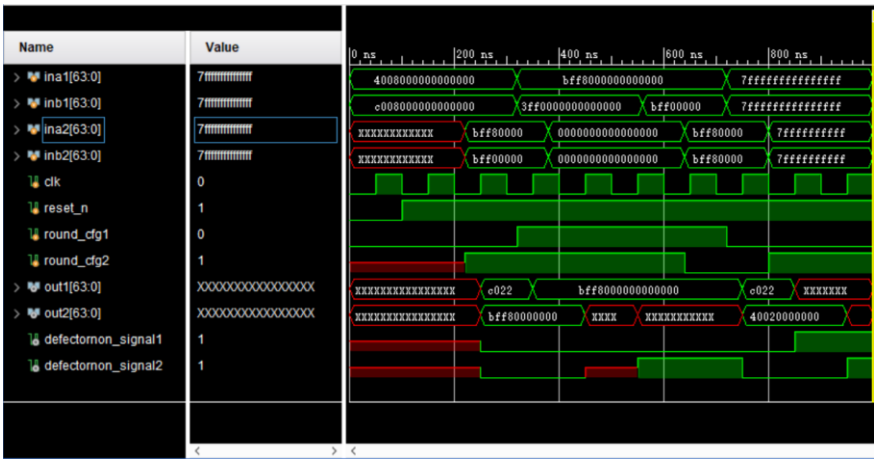


图 1 仿真图结果

能够顺利得到结果，表明程序运行无问题。

4.2 步骤二、分析仿真图内的流水线是否成立

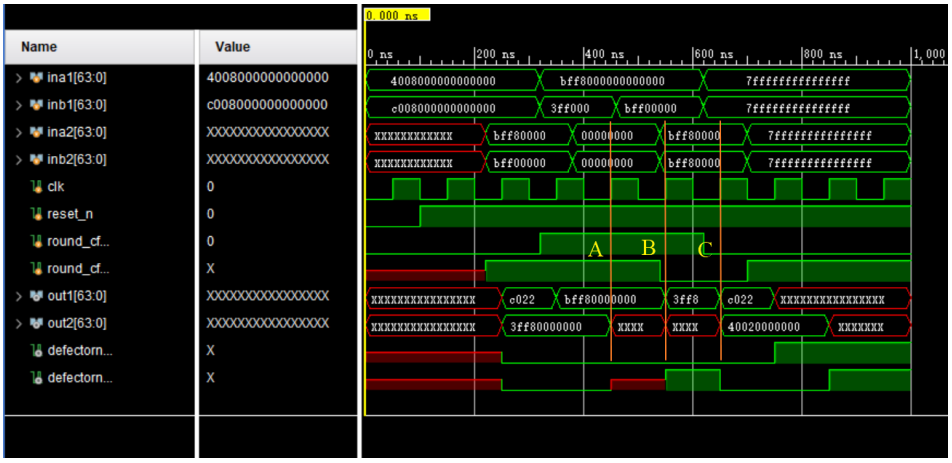


图 2 增加标志线和标志区后的仿真图

对图 2 进行分析，观察 3 条橘红色直线 A、B、C(字母同样也可以表达为过时钟上升沿的时间)。

4.2.1 结论 1——寄存器无问题

——ina2 和 inb2 的第二组数据信号(64 位 0 和 64 位 0)在直线 A 经过的时钟上升沿进入第一级寄存器，在直线 B 经过的时钟上升沿进入第二级寄存器，伴随输出(因为溢出，在本程序中设定溢出输出为 64 位 X)。**——说明两级寄存器的实现没有问题。**

4.2.2 结论 2——流水线无问题

——ina2 和 inb2 的**第二组数据信号**(64 位 0 和 64 位 0)在直线 A 经过的时钟上升沿进入第一级寄存器，在直线 B 经过的时钟上升沿进入第二级寄存器，同时 ina2 和 inb2 的**第三组数据**(图片中 bff80000 和 bff80000，对应十进制数字为-1.5 和-1.5)在直线 B 经过的时钟上升沿进入第一级寄存器，在直线 C 经过的时钟上升沿进入第二级寄存器，同时输出。在直线 B 到直线 C 这段时间内，ina2 和 inb2 **第二组数据**正在第二级流水线操作，ina2 和 inb2 **第三组数据**正在第一级流水线操作。**——说明两级流水线已然正确实现。**

4.3 步骤三、分析仿真图内的输入输出数据——一般情况

根据图 2 观察

4.3.1 结论 1——常见信号无问题

——ina1 和 inb1、ina2 和 inb2 的输入信号完全遵从任务二的设计例子，clk 信号没有问题，reset_n 复位信号也没有问题（上图中黄色框内，在 reset_n 复位信号为 0 时，本程序设定输出 out 为 64 位 x，在随后复位信号回到 1 时，程序可以正常输出任务二-步骤二的例子分析的结果）**——说明基本信号诸如数据信号、时钟信号和控制信号的实现没有问题。**

4.3.2 结论 2——约化信号无问题

——round_cfg1 和 round_cfg2 信号分别用于(ina1、inb1)和(ina2、inb2)两组数据，为了便于叙述，这里都认为是 round_cfg 信号，值为 0 则直接 chopping，值为 1 则 rounding。根据任务二-步骤二的例子分析的结果，对照上图，约化规则有效。**——说明约化规则如 chopping 和 rounding 没有问题。**

4.3.3 结论 3——溢出信号无问题

——defectornon_signal1 和 defectornon_signal2 就是(ina1,inb1)和(ina2,inb2)的相乘结果的溢出判断值，为 1 则溢出，为 0 则未溢出。根据任务二-步骤二的例子分析的结果，对照上图，溢出判断正确。**——说明溢出无问题。**

4.3.4 结论 4——一般数据输出正确

——输入数据无 1 和-1 的情形，这边给出两个例子，均来自于图 2

第一个实例、ina1 和 inb1 第一组数据如下，$ina1 = 64'h4008\_0000\_0000\_0000$，$inb1 = 64'hC008\_0000\_0000\_0000$，那么$ina1 = +2^1 \times (1.1)_2 = 3$，$inb1 = -2^1 \times (1.1)_2 = -3$，结果应该是$result1 = -9 = -2^3 \times (1.001)_2$，不管采用 chopping 还是 rounding，经过 64 位规格化处理之后，$result1 = 64'hC022\_0000\_0000\_0000$。结果和图片中输出 out1(C022)相符。

第二个实例、ina2 和 inb2 第三组数据如下，$ina2 = 64'hBFF8\_0000\_0000\_0000$，$inb2 = 64'hBFF8\_0000\_0000\_0000$，那么$ina2 = -2^0 \times (1.1)_2 = -1.5$，$inb2 = -2^0 \times (1.1)_2 = -1.5$，结果应该是$result2 = -2.25 = -2^1 \times (1.001)_2$，不管采用 chopping 还是 rounding，经过 64 位规格化处理之后，$result2 = 64'h4002\_0000\_0000\_0000$。结果和图片中输出 out1(4022)相符。

——两个例子说明一般数据的输出无问题。

4.4 步骤四、分析仿真图内的输入输出数据——特殊情况

4.4.1 结论 1——特殊数据输出正确

——输入数据有 1 和-1 的情形，这边给出两个例子，均来自于图 2

第一个实例、ina1 和 inb1 的第二组数据如下，$ina1 = 64'hBFF8\_0000\_0000\_0000$，$inb1 = 64'h3FF0\_0000\_0000\_0000$，那么$ina1 = -2^0 \times (1.1)_2 = -1.5$，$inb1 = 1$，结果应该是$result1 = -1.5 = -2^0 \times (1.1)_2$，不管采用 chopping 还是 rounding，经过 64 位规格化处理之后，$result1 = 64'hBFF8\_0000\_0000\_0000$。结果和图片中输出 out1(BFF8)相符。

第二个实例、ina2 和 inb2 的第一组数据如下，$ina2 = 64'hBFF8\_0000\_0000\_0000$，$inb2 = 64'hBFF8\_0000\_0000\_0000$，那么$inb2 = -1$，$ina2 = -1.5$，结果应该是$result2 = 1.5 = 64'h3FF8_0000_0000_0000$。结果和图片中输出 out2(3FF8)相符。

——两个例子说明特殊数据的输出无问题。

4.4.2 结论 2——特殊数据仅含一级流水

——仍然利用上述两个例子，可以看到步骤四下两组特殊例子(如图 3 的蓝色和黄色方框)在时钟沿上升信号到来后立刻显示结果，而不是隔一个时钟周期再输出结果(如图 3 的浅白色方框)。

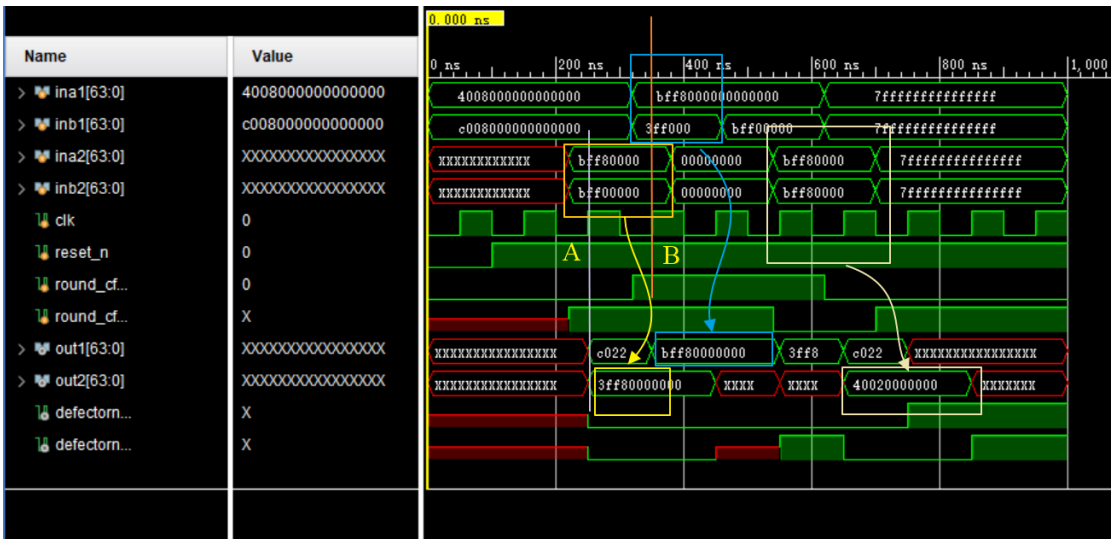


图 3 特殊情况下的一级流水和一般情况下的二级流水

蓝色方框和黄色方框是特殊情况——一级流水，而浅白色方框是一般情况——二级流水

五、上下溢判断[补充，2020.12.3]

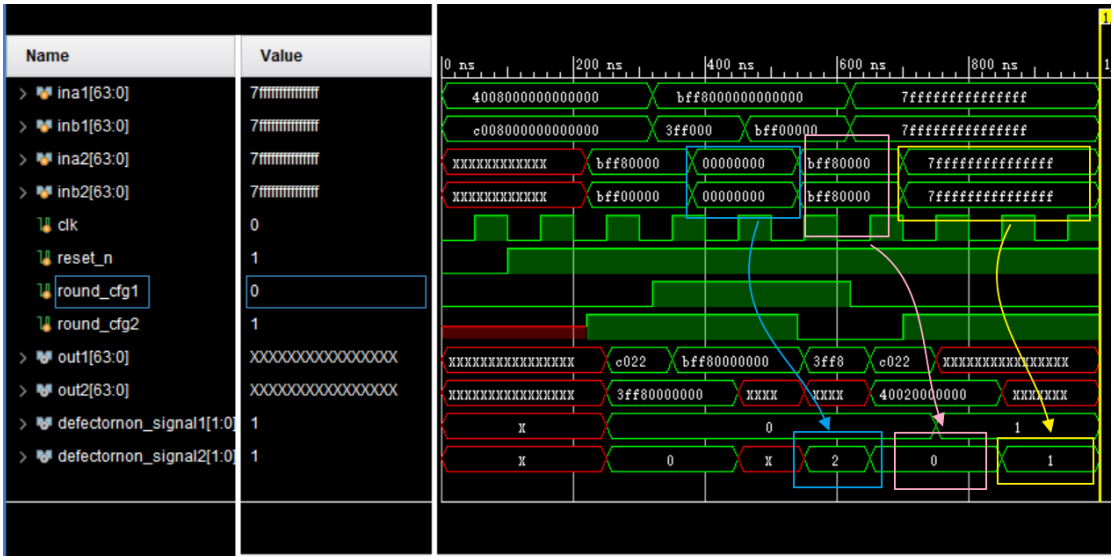


图 4 上下溢判断

图中如果上溢则 defectornon_signal 信号值为 1，如果下溢则为 2，如果无溢出则为 0，如果 reset_n 信号介入，则为 x。

在程序中的体现——其原则是在溢出的情况下，如果 ina[62]和 inb[62]都是 0，那么自然是下溢；如果 ina[62]和 inb[62]都是 1，那么自然是上溢；如果 ina 的指数部分是负值且绝对值大于指数部分是正值的 inb 的绝对值，那么就是下溢；如果 ina 的指数部分是正值且绝对值大于指数部分是负值的 inb 的绝对值，那么就是上溢。

因此根据图 4 的内容，上下溢的判断没有问题。

六、心得

这次作业难度稍大，但是总的来看是完成任务的，实现了以下要求

1、 特殊输入数据的乘法结果直接输出，实现“一级流水”；
2、 一般输入数据的乘法结果二级流水输出；
3、 以上两种情况的结果正确输出；
4、 溢出的判断；
5、 约化规则正确；
6、 使用数据满足 IEEE754 标准；
7、 后续的补充——判断上溢和下溢。

但没有做到的地方有如下：

1、 未能找到一组数据使得两种约化条件下输出不同的有效结果。（但根据程序，可以推断一旦找到这样的数据，在本程序中可以实现正确输出）

七、附录

附录一、design code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2020/10/22 09:29:55
// Design Name:
// Module Name: DouFloatMul
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//***本模块综合实现符合 IEEE754 标准的双精度浮点表示
//***与两个双精度浮点数的乘法计算
//***双精度浮点数的第 63 位为 1 则负，为 0 则正；第 62-52 位为指数位；剩余第 51-0 位为 significand
//两级流水线
module DouFloatMul
  (input [63:0] ina,
   input [63:0] inb,
   input clk,
```

```
    input reset_n,
    input round_cfg,
    output reg [63:0] out,
    output reg [1:0]defectornon_signal);     //0 表示没有溢出，1 表示上溢，2 表示下溢，x 表示 rstn 的输出。

    parameter exponent_bias = 11'd1023;

//***特殊数字的输入放到第二级流水线进行处理
//***特殊数字包括其中任何一个乘数为 0 或者为 1 或者为-1
//***一级流水部分
    reg sign_reg1;
    reg    [11:0]exponent_reg1;                    //存储指数相加以后的指数值并且多设置一位以检测是否溢出
    wire [52:0]a_signficant_real;
    wire [52:0]b_signficant_real;
    reg    [105:0]ab_signficant_real;   //赋值为 0 是为了便于之后对首位和次位的判断

    assign a_signficant_real    = {1,ina[51:0]}; //可认为对小数部分右移 52 位
    assign b_signficant_real    = {1,inb[51:0]}; //可认为对小数部分右移 52 位

//***  关于上溢和下溢的判断，如果同号容易，异号则取决于幂的绝对值更大
    reg [1:0] double_symbol;
    reg exponent_abs;     //0 则表明两数之间的负幂更大，1 则表明两数之间的正幂更大。
    wire [10:0]a_exponent_real;
    wire [10:0]b_exponent_real;

    assign a_exponent_real = ina[62]? (ina[62:52] - exponent_bias):(exponent_bias - ina[62:52]);
    assign b_exponent_real = inb[62]? (inb[62:52] - exponent_bias):(exponent_bias - inb[62:52]);



    //***二级流水线部分变量定义
    reg [54:0]ab_signficant_106_tmp;
    reg [54:0]ab_signficant_106_retmp;
    reg [51:0]ab_signficant_106;
    reg [12:0]exponent_reg21;
    reg [10:0]exponent_reg22_tmp;
    reg [1:0]defectornon_signal_tmp;

    //***  一级流水线部分
    always @(posedge clk or negedge reset_n)
    begin
        if(!reset_n) begin
```

```
            sign_reg1 <= 1'bx;
            exponent_reg1 <= 12'hxxx;
            ab_signficant_real <= 106'h00xx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx;
            double_symbol <= 2'bxx;
            exponent_abs <= 1'bx;
            end
        else begin
          if (!((ina[51:0]==0 && ina[63]==0 && ina[62:52]==exponent_bias)||(inb[51:0]==0 && inb[63]==0 && inb[62:52]==exponent_bias)||(ina[51:0]==0 && ina[63]==1 && ina[62:52]==exponent_bias)||(inb[51:0]==0 && inb[63]==1 && inb[62:52]==exponent_bias)))
            begin
            sign_reg1 <= ina[63]^inb[63]; //判断正负
            exponent_reg1 <= ina[62:52]+inb[62:52]-exponent_bias;
            ab_signficant_real <= a_signficant_real*b_signficant_real;
            double_symbol <= {ina[62],inb[62]};
            if ( ina[62]==1 || inb[62]==0 ) begin
                exponent_abs <=   (a_exponent_real>b_exponent_real) ? 1 : 0;
            end
            else if ( ina[62]==0 || inb[62]==1 ) begin
                exponent_abs <=   (a_exponent_real>b_exponent_real) ? 0 : 1;
            end
            else begin
                exponent_abs <= 1'bx;
            end
            end
           else
             begin
               if(ina[51:0]==0 && ina[63]==0 && ina[62:52]==exponent_bias) begin
                 out <= inb;
                 defectornon_signal <= 2'b00;
               end
               else if(inb[51:0]==0 && inb[63]==0 && inb[62:52]==exponent_bias) begin
                 out <= ina;
                 defectornon_signal <= 2'b00;
               end
               else if(ina[51:0]==0 && ina[63]==1 && ina[62:52]==exponent_bias) begin
                  out[62:0] <= inb[62:0];
                  defectornon_signal <= 2'b00;
                 if (inb[63]==1) begin
                       out[63] <= 1'd0;
                       end
                 else if (inb[63]==0) begin
                       out[63] <= 1'd1;
                       end
```

```
                end
                else if(inb[51:0]==0 && inb[63]==1 && inb[62:52]==exponent_bias) begin
                 out[62:0] <= ina[62:0];
                 defectornon_signal <= 2'b00;
                 if (ina[63]==1) begin
                       out[63] <= 1'd0;
                       end
                 else if (ina[63]==0) begin
                       out[63] <= 1'd1;
                       end
                end
              end
          end
    end

    //***指数部分第一次调整，根据首二位确定指数是否+1
    always @(posedge clk or negedge reset_n)
    begin
     if(!reset_n)
       begin
          out <= 64'dx;
          defectornon_signal <= 2'bxx;
          end

      else
       begin
//******  完成指数和底数的调整
    if (!((ina[51:0]==0 && ina[63]==0 && ina[62:52]==exponent_bias)||(inb[51:0]==0 && inb[63]==0 && inb[62:52]==exponent_bias)||(ina[51:0]==0 && ina[63]==1 && ina[62:52]==exponent_bias)||(inb[51:0]==0 && inb[63]==1 && inb[62:52]==exponent_bias)))
       begin
       case(ab_signficant_real[105])
        1'b1:
        begin
              exponent_reg21 = exponent_reg1 + 1'b1;
              ab_signficant_106_tmp[54:53] = 2'b01;
              ab_signficant_106_tmp[52:1] =ab_signficant_real[104:53];
              ab_signficant_106_tmp[0] = ab_signficant_real[52];
         end
        1'b0:
        begin
              exponent_reg21 = exponent_reg1;
              ab_signficant_106_tmp[54:53] = 2'b01;
              ab_signficant_106_tmp[52:1] = ab_signficant_real[103:52];
```

```
            ab_signficant_106_tmp[0] = ab_signficant_real[51];
        end
    endcase
    //***round 环节
    case(round_cfg)
    1'b0:     //chopping
    begin
        ab_signficant_106 = ab_signficant_106_tmp[52:1];
    end
    1'b1:     //舍入
    begin
        case(ab_signficant_106_tmp[0])   //判断被舍去的最高位
        1'b0:ab_signficant_106 = ab_signficant_106_tmp[52:1];
        1'b1:
        begin
            ab_signficant_106_retmp = ab_signficant_106_tmp[54:1]+1;
            case(ab_signficant_106_retmp[54])
            //***第二次调整指数
              1'b1:
              begin
                exponent_reg21 = exponent_reg21 + 1'b1;
                ab_signficant_106 = ab_signficant_106_retmp[53:2];
              end
              1'b0:
              begin
                exponent_reg21 = exponent_reg21;
                ab_signficant_106 = ab_signficant_106_retmp[52:1];
              end
              endcase
        end
        endcase
    end
    endcase
    //****判断指数部分是否溢出
    //****该部分为指数部分是否上溢出，也就是指数部分超过 IEEE754 规定的指数部分最大值
    case(exponent_reg21[12:11])
        2'b00:begin
            exponent_reg22_tmp = exponent_reg21[10:0];
            defectornon_signal_tmp = 2'b00;     //信号为低时，数据有效
            ab_signficant_106 = ab_signficant_106;
            end
        2'b01,2'b10,2'b11:begin          //****
            exponent_reg22_tmp = 11'dx;
```

```
                ab_signficant_106 = 52'dx;
                // 1 为上溢，2 为下溢
                if (double_symbol == 2'b11) begin
                   defectornon_signal_tmp = 2'b01;    //数据上溢
                end
                else if (double_symbol == 2'b00) begin
                         defectornon_signal_tmp = 2'b10; //数据下溢
                   end
                else begin
                     if (exponent_abs == 1) begin   //两数之间正幂更大，则上溢
                          defectornon_signal_tmp = 2'b01;
                          end
                     else begin                              //两数之间负幂更大，则下溢
                          defectornon_signal_tmp = 2'b10;
                          end
                     end
                     end
            endcase

//*****赋值
        out[63] <= sign_reg1;
        out[62:52] <= exponent_reg22_tmp;
        out[51:0] <= ab_signficant_106;
        defectornon_signal <= defectornon_signal_tmp;
       end
   end
   end
endmodule
```

附录二、test code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2020/10/22 15:15:36
// Design Name:
// Module Name: Test_DFM
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
```

```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//***双精度浮点数的第 63 位为 1 则负，为 0 则正；第 62-52 位为指数位；剩余第 51-0 位为 significand
//***63-60;59-56;55-52;51-48;47-44;43-40;39-36;35-32;31-28;27-24;23-20;19-16;15-12;11-8;7-4;3-0
//*** 1        2        3        4        5        6        7        8        9        10       11       12       13       14    15    16

module Test_DFM();
reg [63:0] ina1;
reg [63:0] inb1;
reg [63:0] ina2;
reg [63:0] inb2;

reg clk;
reg reset_n;
reg round_cfg1;
reg round_cfg2;

wire [63:0] out1;
wire [63:0] out2;

wire [1:0]defectornon_signal1;
wire [1:0]defectornon_signal2;

//wire symbol;
//wire [11:0] exponent_reg1;

DouFloatMul myDouFloatMul1(ina1,inb1,clk,reset_n,round_cfg1,out1,defectornon_signal1);
DouFloatMul myDouFloatMul2(ina2,inb2,clk,reset_n,round_cfg2,out2,defectornon_signal2);


always #50 clk = ~clk;
initial
   begin
     //reset_n = 1;
     reset_n = 0;
     clk = 0;
     ina1 = 64'h4008_0000_0000_0000;   //+2^1*(1.1)_2=3
```

```
    inb1 = 64'hC008_0000_0000_0000;  //-2^1*(1.1)_2=-3  计算得到-9  -1001=-2^3*(1.001)_2  64'hC022_0000_0000_0000
    round_cfg1 = 0;
    #100
    reset_n = 1;
    #120
    round_cfg2 = 1;
    ina2 = 64'hBFF8_0000_0000_0000;
    inb2 = 64'hBFF0_0000_0000_0000;  // inb2 =-1  结果为 -1.5 64'h3FF8_0000_0000_0000
    #100
    round_cfg1 = 1;
    ina1 = 64'hBFF8_0000_0000_0000;
    inb1 = 64'h3FF0_0000_0000_0000;  //inb1=+1  结果为 +1.5 64'hBFF8_0000_0000_0000

    #60
    round_cfg2 = 1;
    ina2 = 64'h0000_0000_0000_0000;  //
    inb2 = 64'h0000_0000_0000_0000;  // 结果溢出
   // #20 reset_n = 1;
    #80
    round_cfg1 = 1;
    ina1 = 64'hBFF8_0000_0000_0000;  //
    inb1 = 64'hBFF0_0000_0000_0000;  //2^0*(1.1)_2=1.5  计算得到 1.5 64'h3FF8_0000_0000_0000
    #80
    round_cfg2 = 0;
    ina2 = 64'hBFF8_0000_0000_0000;  //-2^0*(1.1)_2=-1.5
    inb2 = 64'hBFF8_0000_0000_0000;  //-2^0*(1.1)_2=-1.5  计算得到 2.25 10.01_2 = 2^1*(1.001)_2  64'h4002_0000_0000_0000
    #80
    round_cfg1 = 0;
    ina1 = 64'h7FFF_FFFF_FFFF_FFFF;
    inb1 = 64'h7FFF_FFFF_FFFF_FFFF;
    #80
    round_cfg2 = 1;
    ina2 = 64'h7FFF_FFFF_FFFF_FFFF;
    inb2 = 64'h7FFF_FFFF_FFFF_FFFF;
  end
endmodule
```