模块接口在除法器的模块接口基础上添加了乘除法选择信号输入mul\_div\_sel，该信号为0时进行乘法计算，为1时进行除法计算。

module mul\_div(

input wire rst, //复位

input wire clk, //时钟

input wire signed\_div\_i, //是否为有符号乘除法运算，1位有符号

input wire[31:0] opdata1\_i, //被除数或乘数1

input wire[31:0] opdata2\_i, //除数或乘数2

input wire start\_i, //是否开始运算

input wire annul\_i, //是否取消运算，1为取消

output reg[63:0] result\_o, //乘除法运算结果

output reg ready\_o, //乘除法运算是否结束

input wire mul\_div\_sel //0为乘法，1为除法

);

乘除法器内部设置了64位的乘法中间结果寄存器mulend和65位的除法中间结果寄存器dividend，6位寄存器cnt记录计算进行轮数，2位寄存器state记录乘除法器当前所处状态，64位寄存器mul\_tmp储存左移后的乘数1，32为寄存器temp\_op1, temp\_op2用来保存被除数、除数或两个乘数，在乘法操作时temp\_op2内的值会随迭代次数逐渐右移。

乘除法的控制逻辑相同，case语句完全可复用，初始化阶段如果运算为除法则会进行除数是否为0的判断

if(opdata2\_i == `ZeroWord && mul\_div\_sel == 1'b1) begin //如果是除法且除数为0

state <= `DivByZero;

end

如果不是除法或除数不为0，则进行初始化操作，temp\_op1, temp\_op2的初始化操作乘除法完全相同故可以复用

if(signed\_div\_i == 1'b1 && opdata1\_i[31] == 1'b1) begin //被除数或乘数1为负数

temp\_op1 = ~opdata1\_i + 1;

end else begin

temp\_op1 = opdata1\_i;

end

操作数temp\_op2的初始化代码同上，初始化阶段乘除法的区别在于乘法需要初始化mulend和mul\_tmp而除法需要初始化dividend和divisor。

if(mul\_div\_sel) begin

dividend <= {`ZeroWord, `ZeroWord};

dividend[32: 1] <= temp\_op1;

divisor <= temp\_op2;

end

else begin

mulend <= {`ZeroWord, `ZeroWord};

mul\_tmp <= {`ZeroWord,temp\_op1};

end

在运算过程中，对cnt的累加和判断逻辑相同，可以复用，而对操作数的处理需要分开处理。乘法操作的主要逻辑是判断temp\_op2的最低位是否是1，如果是1则需要将mul\_tmp与mulend值相加，并置于mulend寄存器，其中temp\_op2的最低位即为乘数2的第cnt位，mul\_tmp为乘数1左移cnt次的值，若temp\_op2的最低位为0则不需要进行加法操作，上述操作完成后需要将mul\_tmp左移一位，temp\_op2右移一位。

若为除法操作，则需要在div\_temp[32]为1时将dividend左移一位，否则dividend置为div\_temp[31:0]，dividend[31:0]和1的连接，代码如下。

if(cnt != 6'b100000) begin

if(mul\_div\_sel) begin

if (div\_temp[32] == 1'b1) begin

dividend <= {dividend[63:0],1'b0};

end else begin

dividend <= {div\_temp[31:0],dividend[31:0], 1'b1};

end

end

else begin

if(temp\_op2[0]) begin

mulend <= mulend + mul\_tmp;

end

mul\_tmp <= mul\_tmp << 1;

temp\_op2 <= temp\_op2 >> 1;

end

cnt <= cnt +1; //运算次数加一

end

如果cnt值已经到达32说明乘法或除法运算已经结束，此时如果为有符号乘除法需要还原符号位，乘法可以直接将64位的mulend按位取反加一，除法需要将商和余数分别还原。还原后需要将乘除法器状态置为运算结束并将cnt置0，这部分代码可以复用，代码如下

else begin

if (mul\_div\_sel) begin

if ((signed\_div\_i == 1'b1) && ((opdata1\_i[31] ^ opdata2\_i[31]) == 1'b1)) begin

dividend[31:0] <= (~dividend[31:0] + 1);

end

if ((signed\_div\_i == 1'b1) && ((opdata1\_i[31] ^ dividend[64]) == 1'b1)) begin

dividend[64:33] <= (~dividend[64:33] + 1);

end

end

else begin

if ((signed\_div\_i == 1'b1) && ((opdata1\_i[31] ^ opdata2\_i[31]) == 1'b1)) begin

mulend[63:0] <= (~mulend[63:0] + 1);

end

end

state <= `DivEnd;

cnt <= 6'b000000;

end

运算结束后如果则直接输出mulend，除法则跳过dividend的第32位，其余关于取消运算已经状态转移和异常结束等控制电路均可以复用。