**中国海洋大学计算机科学与技术系**

**实验报告**

**姓名：岳宇轩 年级：2019** **专业：** 19慧与

**科目：计算机组成原理** **题目：定点乘法**

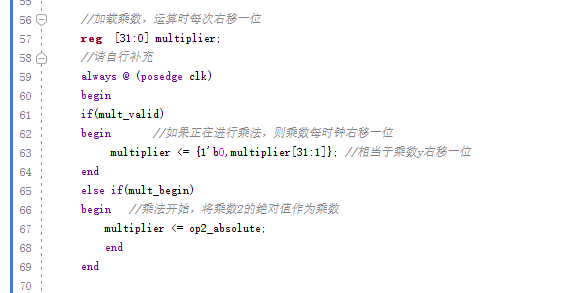
**实验时间:** 2021 年4 月 27日  **实验教师:** 张巍

**一、实验结果及截图分析：**

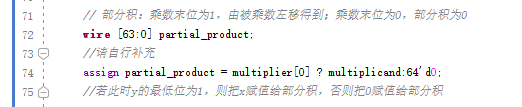
（※代码挖空的部分必须截图或复制）

1. **添加multiply.v文件，并补全代码和注释**

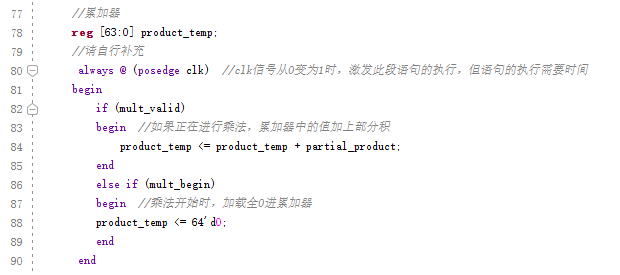
补充1



补充2



补充3



**2.添加testbench.v文件**

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// > 文件名: multiply.v

// > 描述 ：乘法器模块，低效率的迭代乘法算法，使用两个乘数绝对值参与运算

// > 作者 : LOONGSON

// > 日期 : 2016-04-14

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

module multiply( // 乘法器

input clk, // 时钟

input mult\_begin, // 乘法开始信号

input [31:0] mult\_op1, // 乘法源操作数1

input [31:0] mult\_op2, // 乘法源操作数2

output [63:0] product, // 乘积

output mult\_end // 乘法结束信号

);

//乘法正在运算信号和结束信号

reg mult\_valid;

assign mult\_end = mult\_valid & ~(|multiplier); //乘法结束信号：乘数全0

always @(posedge clk)

begin

if (!mult\_begin || mult\_end)

begin

mult\_valid <= 1'b0;

end

else

begin

mult\_valid <= 1'b1;

end

end

//两个源操作取绝对值，正数的绝对值为其本身，负数的绝对值为取反加1

wire op1\_sign; //操作数1的符号位

wire op2\_sign; //操作数2的符号位

wire [31:0] op1\_absolute; //操作数1的绝对值

wire [31:0] op2\_absolute; //操作数2的绝对值

assign op1\_sign = mult\_op1[31];

assign op2\_sign = mult\_op2[31];

assign op1\_absolute = op1\_sign ? (~mult\_op1+1) : mult\_op1;

assign op2\_absolute = op2\_sign ? (~mult\_op2+1) : mult\_op2;

//加载被乘数，运算时每次左移一位

reg [63:0] multiplicand;

always @ (posedge clk)

begin

if (mult\_valid)

begin // 如果正在进行乘法，则被乘数每时钟左移一位

multiplicand <= {multiplicand[62:0],1'b0};

end

else if (mult\_begin)

begin // 乘法开始，加载被乘数，为乘数1的绝对值

multiplicand <= {32'd0,op1\_absolute};

end

end

//加载乘数，运算时每次右移一位

reg [31:0] multiplier;

//请自行补充

always @ (posedge clk)

begin

if(mult\_valid)

begin //如果正在进行乘法，则乘数每时钟右移一位

multiplier <= {1'b0,multiplier[31:1]}; //相当于乘数y右移一位

end

else if(mult\_begin)

begin //乘法开始，将乘数2的绝对值作为乘数

multiplier <= op2\_absolute;

end

end

// 部分积：乘数末位为1，由被乘数左移得到；乘数末位为0，部分积为0

wire [63:0] partial\_product;

//请自行补充

assign partial\_product = multiplier[0] ? multiplicand:64'd0;

//若此时y的最低位为1，则把x赋值给部分积，否则把0赋值给部分积

//累加器

reg [63:0] product\_temp;

//请自行补充

always @ (posedge clk) //clk信号从0变为1时，激发此段语句的执行，但语句的执行需要时间

begin

if (mult\_valid)

begin //如果正在进行乘法，累加器中的值加上部分积

product\_temp <= product\_temp + partial\_product;

end

else if (mult\_begin)

begin //乘法开始时，加载全0进累加器

product\_temp <= 64'd0;

end

end

//乘法结果的符号位和乘法结果

reg product\_sign;

always @ (posedge clk) // 乘积

begin

if (mult\_valid)

begin

product\_sign <= op1\_sign ^ op2\_sign;

end

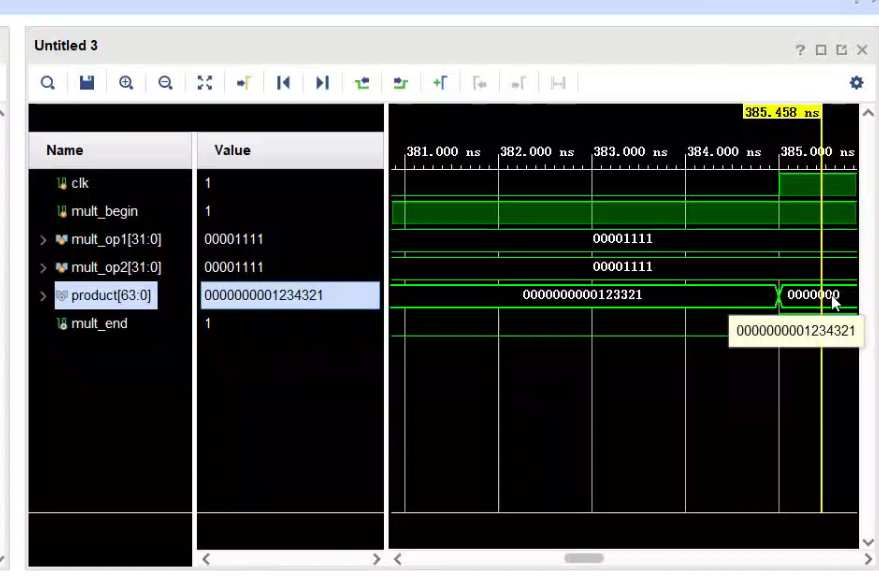
end

//若乘法结果为负数，则需要对结果取反+1

assign product = product\_sign ? (~product\_temp+1) : product\_temp;

endmodule

1. **完成仿真，生成仿真波形图像**



1. **实验总结**

本次实验学习了定点乘法器的编写。程序的大致思路如下：

1.程序开始时，mult\_begin初始化为1，并给出乘数mult\_op1和被乘数mult\_op2。

2.首先对这两个操作数进行取符号和取绝对值的操作，使得符号位和数值位分开进行运算。

3.乘法开始时，加载乘数进入multiplier，此后每时钟右移一位。

4.加载部分积进入partial\_product，乘数末位为1，由被乘数左移得到；乘数末位为0，部分积为0。

5.乘法开始时，加载全0进累加器product\_temp，此后每个时钟，累加器+=部分积。

6.最后，对于符号位product\_sign，如果乘法有效，计算结果为乘数的符号和被乘数的符号进行异或。

7.最终结果produc的运算：若乘法结果为负数（即product\_sign表示为负），则需要对结果取反+1(~product\_temp+1)；否则product\_temp就是最终结果。

通过此次试验，我对于定点乘法的基本原理有了更清楚的理解，包括数值符号分开运算，乘法过程中的具体步骤，以及最终结果的取值等等。同时，我对于verilog的编写和理解感到了有一定的提升。