组成原理实验课程第_2_次实报告

实验名称	数据运算：定点乘法			班级	李涛老师
学生姓名	曹瑜	学号	2212794	指导老师	董前琨
实验地点	实验楼 A 区 306		实验时间	2024.03.28	

1、 实验目的

1.理解定点乘法的不同实现算法的原理，掌握基本实现算法。

2.熟悉并运用 verilog 语言进行电路设计。

3.为后续设计 cpu 的实验打下基础。

2、 实验内容说明

1.学习并理解计算机中定点乘法器的多种实现算法的原理，重点掌握迭代乘法的实现算法。；

2.自行设计本次实验的方案，画出结构框图，本次实验的乘法器建议采用迭代或其他效率更高的算法实现。本次实验要求实现的乘法为有符号乘法，因此需要注意计算机存储的有符号数都是补码的形式，设计方案传递进来的数也需是补码；

3.根据设计的实验方案，使用 verilog 编写相应代码；

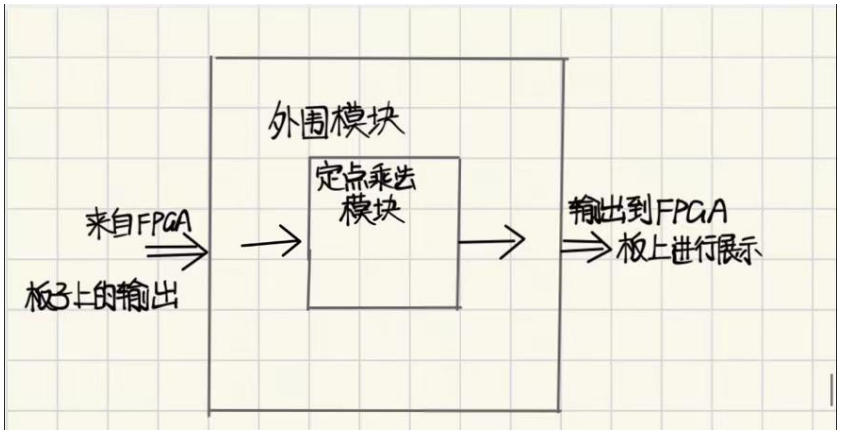
4.对编写的代码进行仿真，得到正确的波形图；

将以上设计作为一个单独的模块，设计一个外围模块去调用该模块，外围模块中需调用封装好的 LCD 触摸屏模块，显示两个乘数和乘法结果，且需要利用触摸功能输入两个乘数；

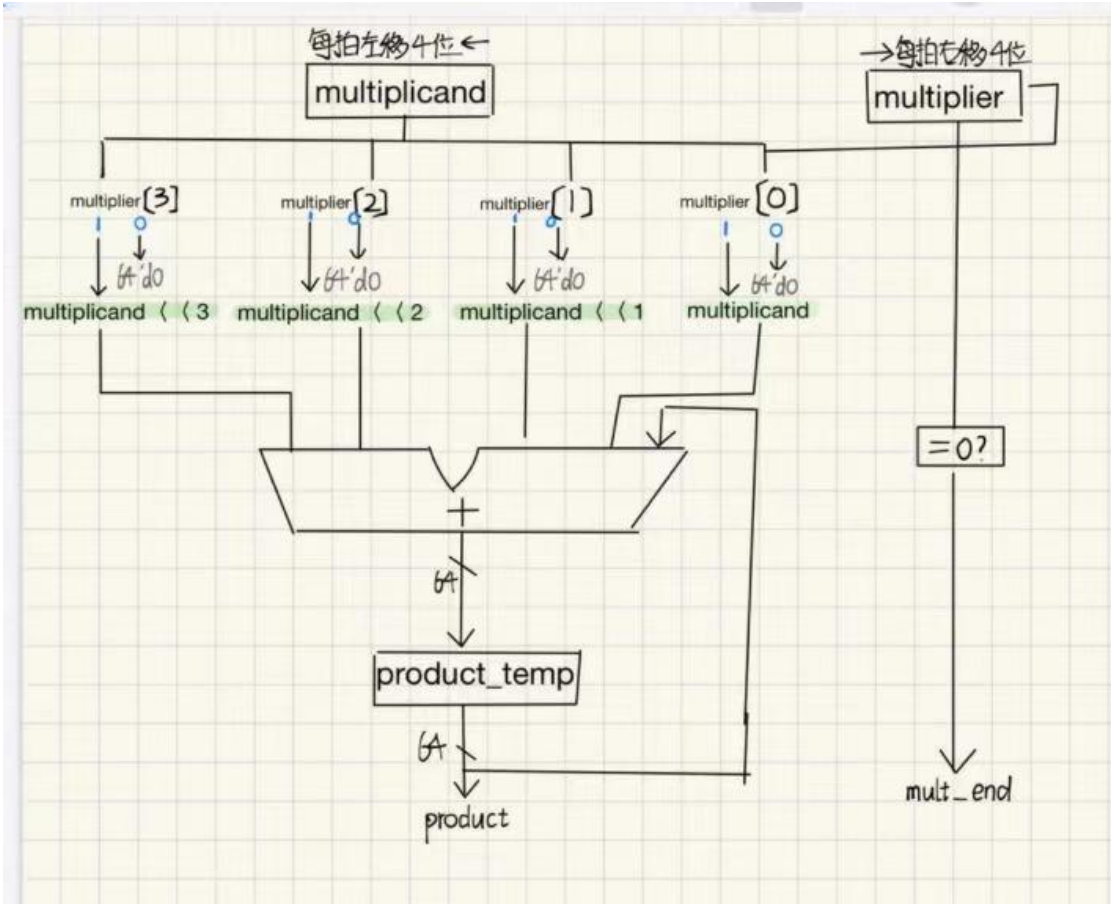
3、 实验原理图

乘数每次右移 4 位，根据最低 4 位，判断是加被乘数移 0/1/2/3 位后的值还是加 0，不停地累加最后得到乘积，用多次加法完成乘法操作，故需要多拍时间，其结束标志为乘数移位后为 0，此时对于 32 位乘法，最多需要 8 拍就能完成一次乘法；

参与运算的为两个乘数的绝对值，乘法结果也是绝对值，需要单独判断符号位后校正乘积；

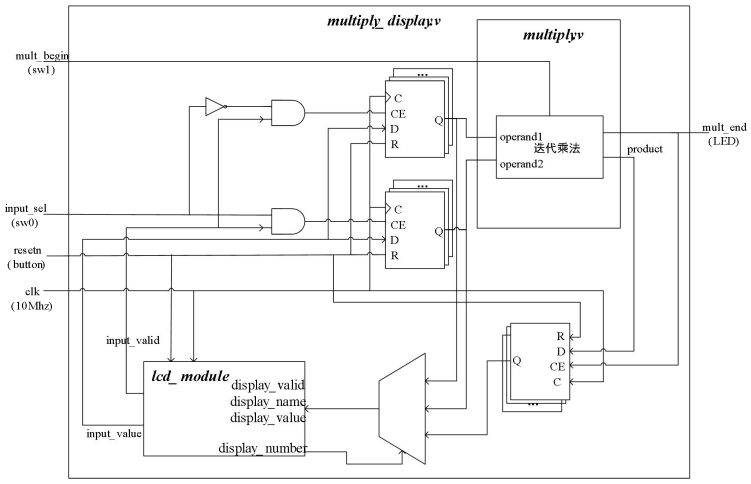


定点乘法设计实验的顶层模块大致框图



迭代乘法算法的原理图（优化版）

优化后的乘法器一次性会左移/右移 4 位，对应地，生成部分积时要根据乘数的末 4 位来决定，[0]位对应着当前被乘数，[1]位对应着当前被乘数左移 1 位的结果，[2]位对应着当前被乘数左移 2 位的结果，[3]位对应着当前被乘数左移 3 位的结果，末四位对应数的累加结果即为部分积，不断更新部分积进行累加即得到最后结果。



定点乘法参考设计的顶层模块框图（未更改）

4、 实验步骤

1、multiply 模块中修改移位操作：（使运算周期缩减为原来的 1/4）

若正在进行乘法，被乘数每时钟左移 4 位，右边用 4 个 0 补齐；

```
//加载被乘数，运算时每次左移4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4位
        multiplicand <= {multiplicand[59:0],4'b0000};
    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 4 位，左边用 4 个 0 补齐；

```
//加载乘数，运算时每次右移4位
reg  [31:0] multiplier;
always @ (posedge clk)
begin
    if (mult_valid)
    begin   // 如果正在进行乘法，则乘数每时钟右移4位
        multiplier <= {4'b0000,multiplier[31:4]};
    end
    else if (mult_begin)
    begin   // 乘法开始，加载乘数，为乘数2的绝对值
        multiplier <= op2_absolute;
    end
end
```

2、multiply 模块中修改根据乘数最后 4 位加载对应的部分积操作

末 4 位为 1 时依次对应：

{multiplicand[60:0],3'b000}

{multiplicand[61:0],2'b00}　　为 0 时均对应：64'd0

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

Multiplicand

根据末 4 位的具体值组装部分积；

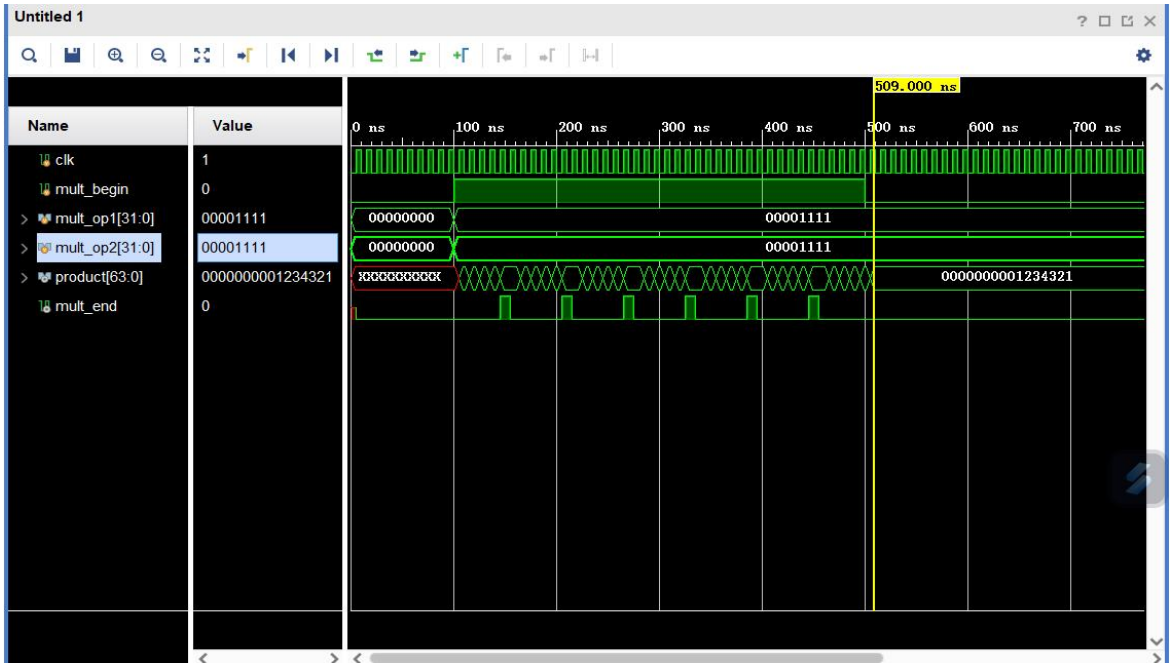
```
// 部分积：根据乘数的最后4位的具体值确定
reg [63:0] partial_product;
always @ *
begin
    case (multiplier[1:0])
        4'b0000: partial_product = 64'd0;
        4'b0001: partial_product = multiplicand;
        4'b0010: partial_product = {multiplicand[62:0],1'b0} ;
        4'b0011: partial_product = multiplicand+{multiplicand[62:0],1'b0};

        4'b0100: partial_product = {multiplicand[61:0],2'b00};
        4'b0101: partial_product = multiplicand+{multiplicand[61:0],2'b00};
        4'b0110: partial_product = {multiplicand[62:0],1'b0}+{multiplicand[61:0],2'b00} ;
        4'b0111: partial_product = multiplicand+{multiplicand[62:0],1'b0}+{multiplicand[61:0],2'b00};

        4'b1000: partial_product = {multiplicand[60:0],3'b000};
        4'b1001: partial_product = multiplicand+{multiplicand[60:0],3'b000};
        4'b1010: partial_product = {multiplicand[62:0],1'b0}+{multiplicand[60:0],3'b000} ;
        4'b1011: partial_product = multiplicand+{multiplicand[62:0],1'b0}+{multiplicand[60:0],3'b000};

        4'b1100: partial_product = {multiplicand[60:0],3'b000}+{multiplicand[61:0],2'b00};
        4'b1101: partial_product = multiplicand+{multiplicand[60:0],3'b000}+{multiplicand[61:0],2'b00};
        4'b1110: partial_product = {multiplicand[62:0],1'b0}+{multiplicand[60:0],3'b000}+{multiplicand[61:0],2'b00};
        4'b1111: partial_product = multiplicand+{multiplicand[62:0],1'b0}+{multiplicand[61:0],2'b00}+{multiplicand[60:0],3'b000};
    endcase
 end
```

5、 实验结果分析

[仿真结果]：

输入：op1: 0000 1111　　op2: 0000 1111

输出: 0000000001234321　　仿真结果正确



[**实验箱试验结果 1**]

输入：op1: FFFFFFFF　　op2: FFFFFFFF

预期输出:　0000000000000001

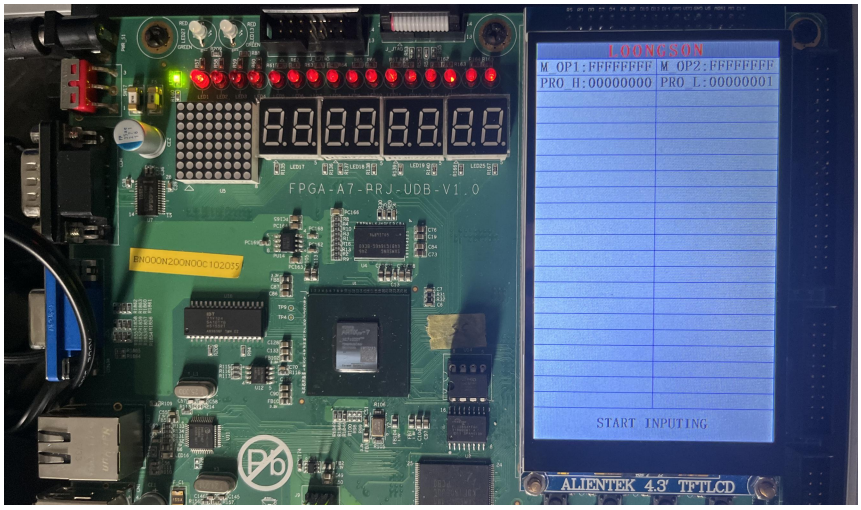
实际输出：0000000000000001　　结果正确

（2 进制表示）：

op1:　　11111111111111111111111111111111　　（负数）

op2:　　11111111111111111111111111111111　　（负数）

Cout:　0000 0000 0000 0000 0000 0000 0000 0001　　（正数）



[**实验箱试验结果 2**]

输入：op1: 0000FFFF　　op2: 00001111

预期输出:　0000000001110EEEF

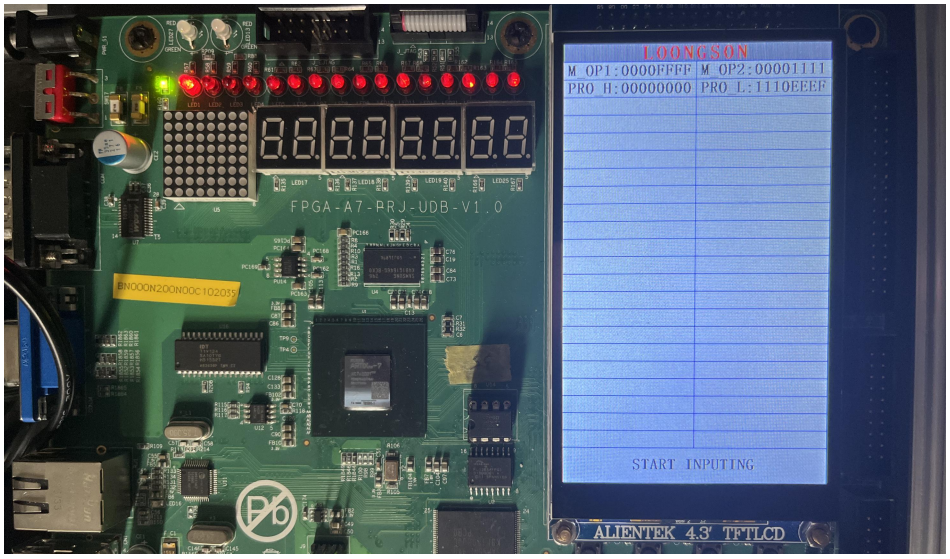
实际输出：0000000001110EEEF　　结果正确

（2 进制表示）：

op1:　　0000 1111 1111 1111　　（正数）

op2:　　0001 0001 0001 0001　　（正数）

Cout:　0001 0001 0001 0000 1110 1110 1110 1111　　（正数）



[**实验箱试验结果 3**]

输入：op1: A1111111　　op2:11111111

预期输出：　F9ABCDF017654321

实际输出：F9ABCDF017654321　　结果正确

（2 进制表示）：

op1:　　1010 0001 0001 0001 0001 0001 0001 0001　　（负数）

op2：　　0001 0001 0001 0001 0001 0001 0001 0001　　（正数）

Cout：　　1111 1001 1010 1011 1100 1101 1111 0000

　　0001 0111 0110 0101 0100 0011 0010 0001　　（负数）

6、 总结感想

通过本次实验，掌握了定点乘法迭代算法的原理并能理解代码实现复现仿真，同时通过移位位数的增加实现了算法时间性能的改进，进一步熟悉了基本的 verilog 语言编写和仿真流程，以及如何对 verilog 语言进行综合布局布线，并下载到实验箱中的 FPGA 板上进行演示，对乘法器的实现有了更深一步认知，但对 vivado 的熟悉程度仍然不够，独自进行项目创建和模块配置时还有问题，希望能在之后的实验中改进。