组成原理实验课程第_2_次实验报告

实验名称	乘法器改进			班级	李涛
学生姓名	阿斯雅	学号	2210737	指导老师	董前琨
实验地点	实验楼 A 区 308		实验时间	3 月 28 中午	

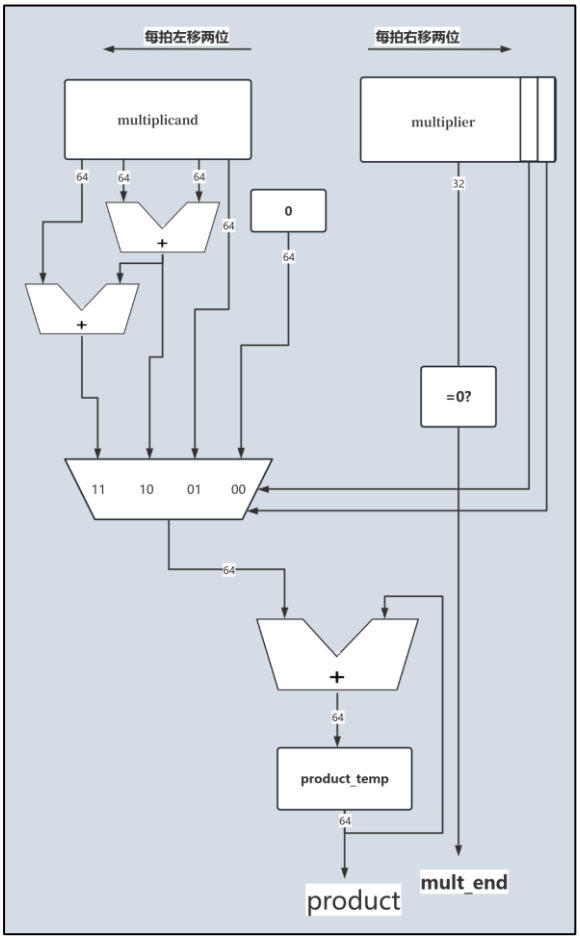
1、 实验目的

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。

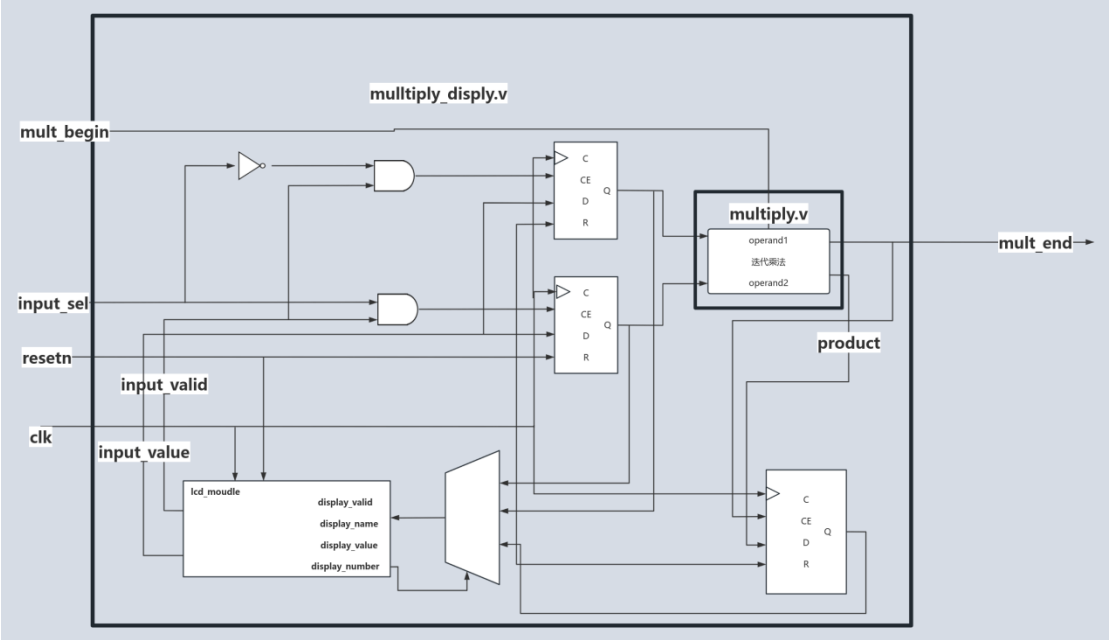
2、 实验内容

结合实验指导手册中的实验二（定点乘法器实验）完成性能改进，不在是原始的最长 32 个时钟周期完成乘法。

3、 实验原理图



（迭代乘法原理图）



（实验顶层模块框图）

4、 实验步骤

4.1、分析

因为原先乘法模块实现的是通过每次时钟沿把被乘数左移一位，乘数右移一位来实现两个 32 位数的乘法。而我们如果要想乘法运算可以更快，我们可以在每次时钟沿把被乘数和乘数左右移动两位，这样可以在原先的一半时间内就可以运算完乘法。

而要想每次移动两位，我们就需要判断乘数的最后两位是多少。显然，一共有四种情况：00,01,10,11。对每一种情况，都需要我们去改变当前的 product_temp 来让去累加。比如当乘数最后两位是 00 时 product_temp 就为 0，而当乘数最后两位等于 11 时 product_temp 就为 multiplicand+multiplicand+multiplicand，以此类推。

除了乘法运算的原理稍微有点变化，其他的模块都没有受到影响，所以我们修改代码时就可以只修改 multiply.v 模块，而其他的模块不用改动。

4.2、修改 multiply.v 模块

首先，我们需要改变左右移代码，实现每拍左右移动两位。对被乘数来说，每拍我们可以把被乘数的低 62 位送到高位，而最后两位填充为 0，这样就可以实现被乘数左移两位。而对乘数来说，我们可以把高 30 位送到低位上，而最高两位填充为 0，这样就可以实现乘数右移两位。

```
//加载被乘数，运算时每次左移位两
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两位
        multiplicand <= {multiplicand[61:0],2'b00};
    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
```

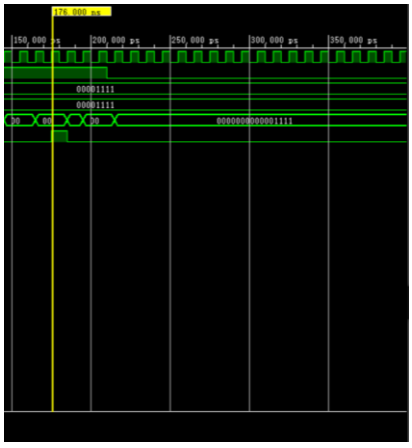
```
//加载乘数，运算时每次右移两位
reg  [31:0] multiplier;
always @ (posedge clk)
begin
    if (mult_valid)
    begin   // 如果正在进行乘法，则乘数每时钟右移两位
        multiplier <= {2'b00,multiplier[31:2]};
    end
    else if (mult_begin)
    begin   // 乘法开始，加载乘数，为乘数2的绝对值
        multiplier <= op2_absolute;
    end
end
```

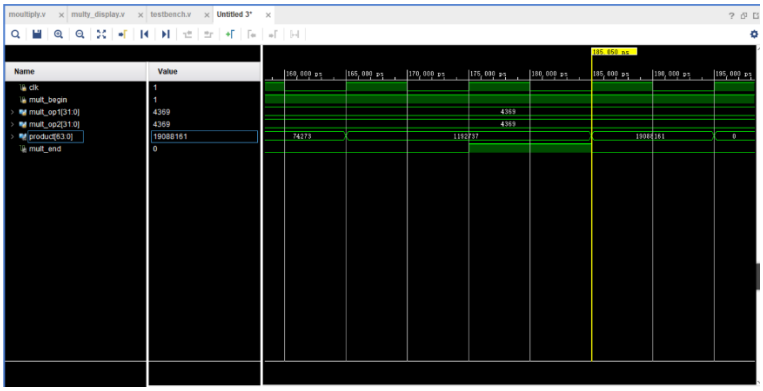
然后就到了乘法运算部分，因为乘数最后两位共有四种情况，所以我们可以使用 case 语句来实现对不同的情况做出不同的动作。对情况 00 来说，临时变量 product_temp 就为 0,对情况 01 来说,product_temp 就等于 multiplicand，对情况 10 来说 product_temp=multiplicand+multiplicand，而对情况 11 来说 product_temp 就为 multiplicand+multiplicand+multiplicand。

```
reg [63:0] partial_product;

always @(posedge clk)
    begin
        case({multiplier[1],multiplier[0]})
            2'b00 :
            begin
                partial_product <= 64'd0;
            end
            2'b01 :
            begin
                 partial_product<= multiplicand;
            end
            2'b10 :
            begin
                partial_product <= multiplicand+multiplicand;
            end
            2'b11 :
            begin
                partial_product <= multiplicand+multiplicand+multiplicand;

            end
       endcase
    end
```

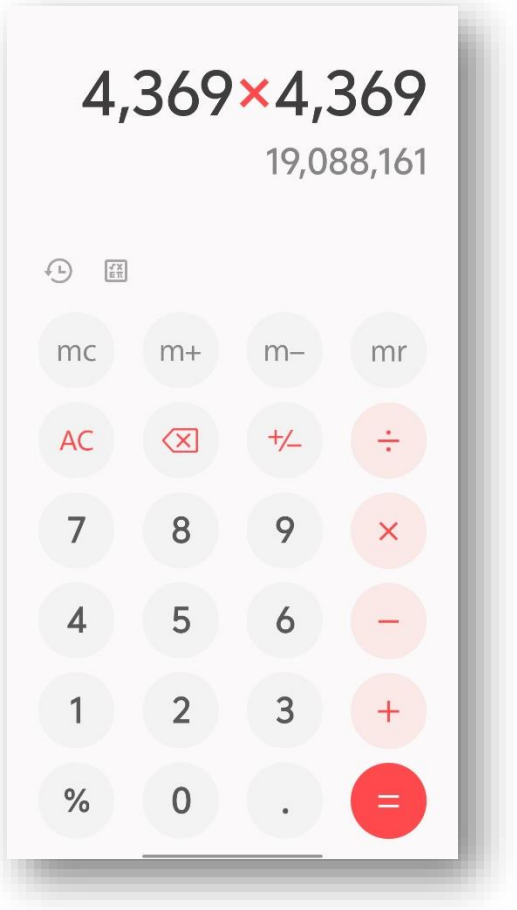
5、 实验结果分析

5.1、仿真模拟图

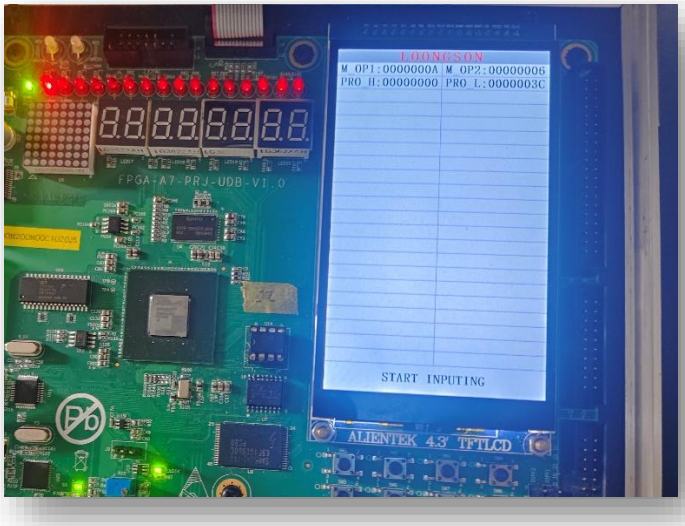




结果分析：因为乘法运算需要好几个时钟周期才能运算完，所以仿真模拟并不会立即出现正确答案，而是会出现当前累加器的结果。而要想看到正确答案，我们可以当 mult_end=1 的时候，因为 mult_end=1 就标志着乘法运算的结束。在仿真模拟中，我们可以先找到 mult_end 出现 1 的那个方波，紧跟在它后面的 product 就是乘法的正确答案。在本次实验中，我们从图片可以看出两个乘数都是 4369，而 mult_end=1 后面的 product=19088161。我们可以在手机计算器上验证相应的值：



5.2、实验箱上机图片



6、 心得体会

在这次乘法改进实验中，我深入探究了迭代乘法运算的原理。通过了解每

次如何对被乘数和乘数进行左右移，我逐渐领悟到了乘法运算背后的逻辑。然而，在改进代码的过程中，我发现这一次与之前的实验有所不同。这次的乘法实验涉及到时序逻辑，因此需要更加细致地考虑时钟信号的作用。除此之外，我还意识到需要修改最核心的乘法模块，这意味着我们必须重新思考整个运算的实现原理。因此，对 Verilog 语言的熟悉程度变得尤为重要。只有通过对 Verilog 语言的深入了解，我们才能准确地修改代码，确保整个程序能够顺利运行，并且在性能和效率上都有所提升。