组成原理实验课程第二次实验报告

实验名称	数据运算：定点乘法			班级	张金老师
学生姓名	蒋薇	学号	2110957	指导老师	董前琨
实验地点	A306		实验时间	2023.4.3	

1、 实验目的

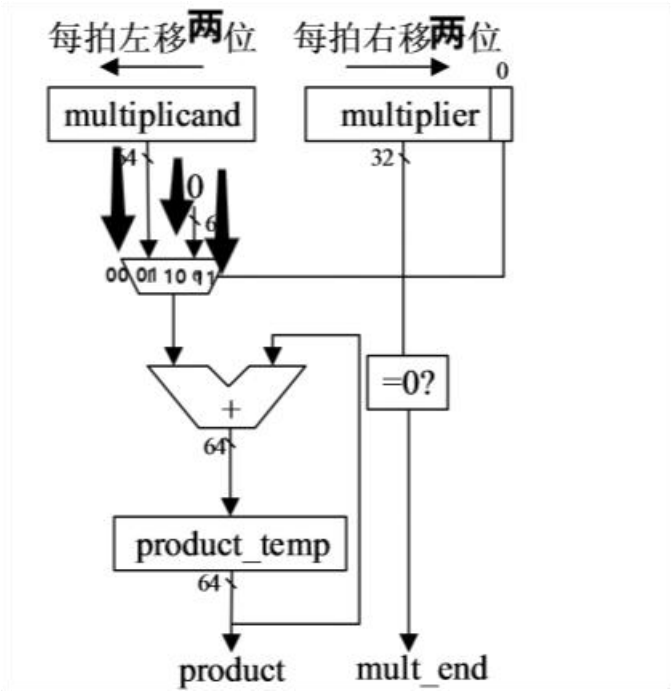
（1） 理解定点乘法的不同实现算法的原理，掌握基本实现算法。
（2） 熟悉并运用 verilog 语言进行电路设计。
（3） 为后续设计 cpu 的实验打下基础。

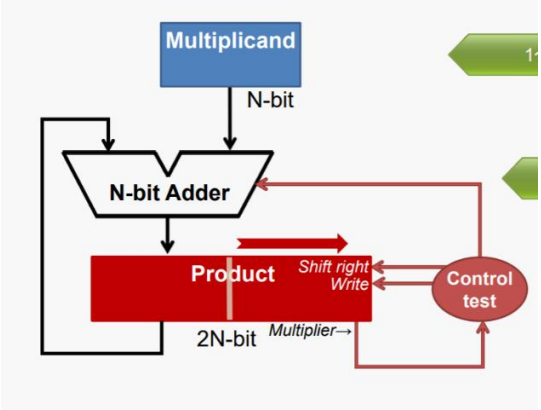
2、 实验内容说明

（1）重点掌握迭代乘法的实现算法，将原有的迭代乘法改进成两位乘法，即每个时钟周期移位移两位，从而提高乘法效率。
（2）仿真得到正确的波形图，将改进后的乘法器进行仿真验证
（3）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。将改进后的乘法器进行上实验箱验证，上箱验证时调整数据不在前 4 格显示
（4）画出结构框图，详细标出输入输出端口，原理图为迭代乘法的算法图，不再是顶层模块图
（5）介绍分析的内容，针对仿真的波形图和实验箱照片，要解释图中信息，是否验证成功。

3、 实验原理图

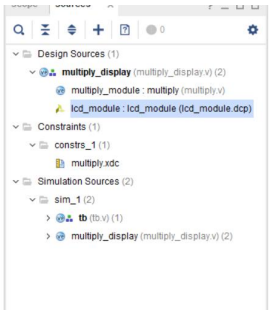
（画图并简要说明）迭代乘法的算法图，不再是顶层模块图





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。

4、 实验步骤



（1）将原有的迭代乘法改进成两位乘法，即每个时钟周期移位移两位，从而提高乘法效率。

修改被乘数左移：

```
//加载被乘数，运算时每次左移两位
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
```

修改乘数右移：

```
// 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   //    如果正在进乘法，被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
```

修改部分积：

```
//部分积
wire [63:0] partial_product;
wire [63:0] partial_product2;//定义两个临时变量
    assign partial_product = multiplier[0] ? multiplicand : 64'd0;
    //乘数末位为1，临时变量1是被乘数本身；乘数末位为0，临时变量1是0
    assign partial_product2 = multiplier[1]?{multiplicand[62:0],1'b0}:64'd0;
    //乘数倒数第二位为1，临时变量由被乘数左移一位得到；乘数倒数第二位为0，临时变量2为0
 reg [63:0] product_temp;
```

修改累加器

```
g [63:0] product_temp;
    always @ (posedge clk)
    begin
        if (mult_valid)
        begin
            product_temp <= product_temp + partial_product + partial_product2;//两位
        end
        else if (mult_begin)
        begin
            product_temp <= 64'd0;  //  _ 乘法开始，乘积清零  '
        end
```

由于改进后的代码比源代码只多增加了一个 wire 的临时变量，故对 multiply.xdc、multiply_display.v 等文件不需要修改。

（3）将改进后的乘法器进行上实验箱验证，上箱验证时调整数据不在前 4 格显示。从第五个开始显示，修改如下：

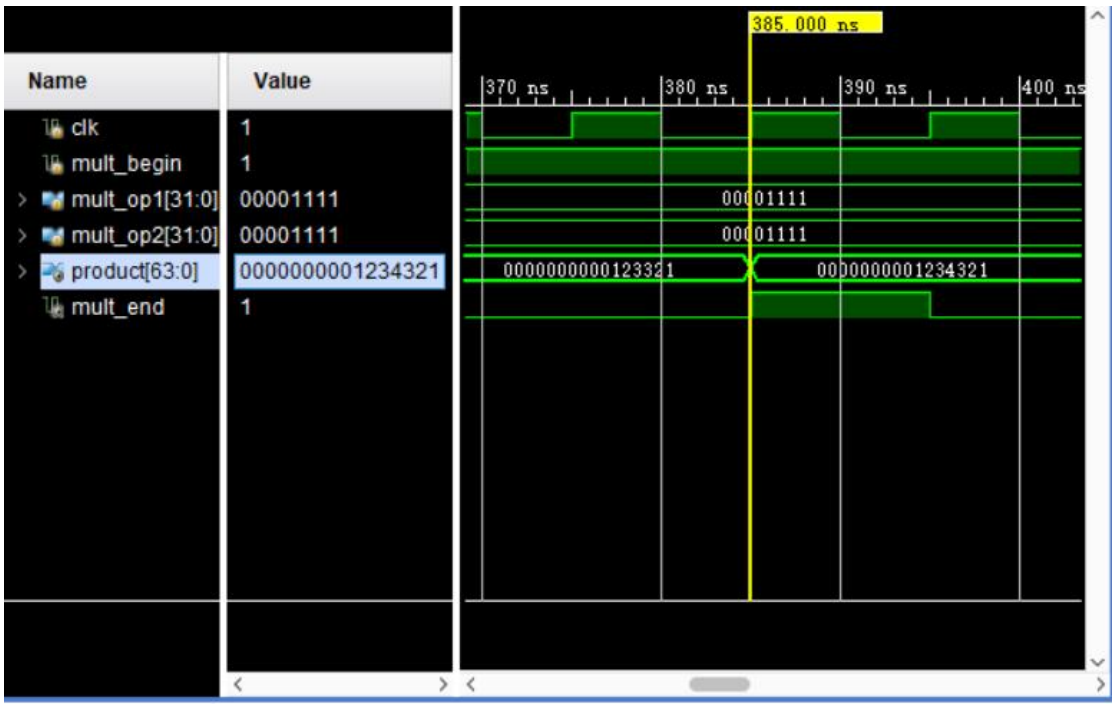
```
151             6'd5 :
152             begin
153                 display_valid <= 1'b1;
154                 display_name  <= "M_OP1";
155                 display_value <= mult_op1;
156             end
157             6'd6 :
158             begin
159                 display_valid <= 1'b1;
160                 display_name  <= "M_OP2";
161                 display_value <= mult_op2;
162             end
163             6'd7 :
164             begin
165                 display_valid <= 1'b1;
166                 display_name  <= "PRO_H";
167                 display_value <= product_r[63:32];
168             end
169             6'd8 :
```

试验箱显示如下：

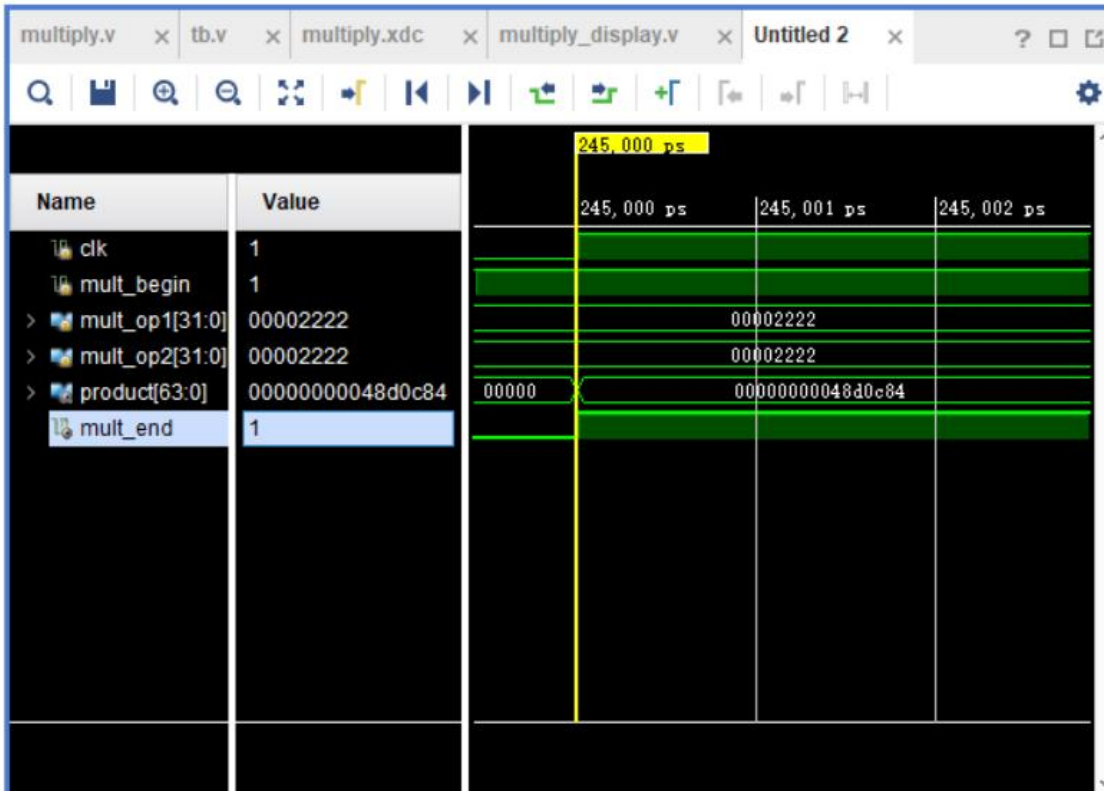
综上，该功能实现。

（分布介绍依次完成了哪些代码修改，从而实现了什么样的功能）

5、 实验结果分析

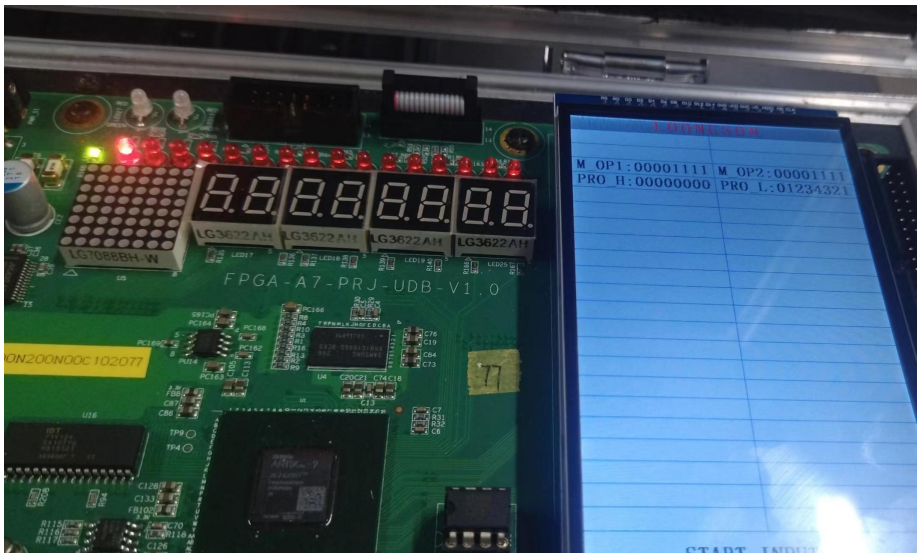


mult_op1 = 00001111,mult_op2 = 00001111,product = 0000000001234321。

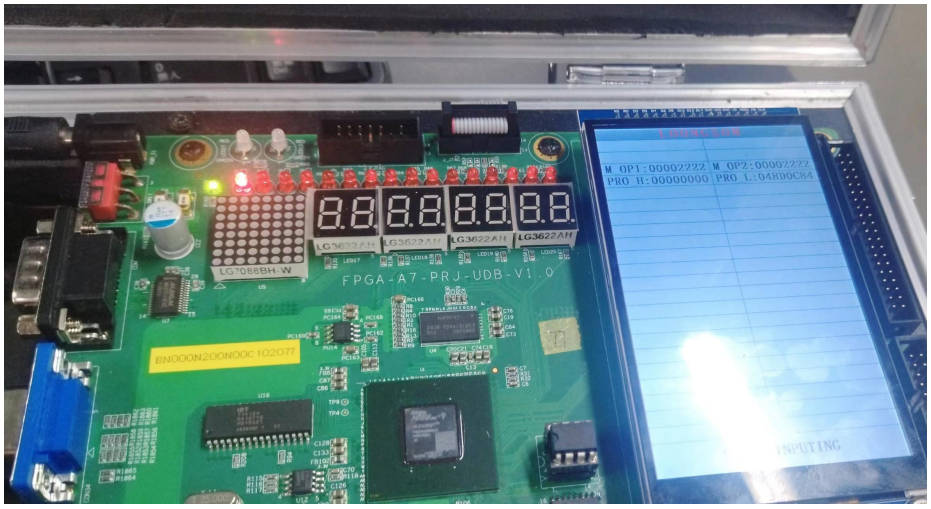


Mult_op1 = 00002222,mult_op2 = 00002222,product = 00000000048d0c84。

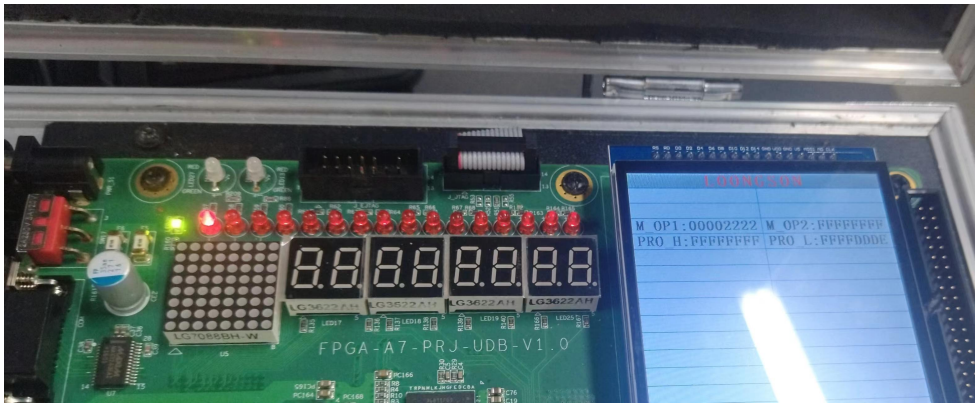
仿真结果表明，改进后的乘法器能够正确地进行两位乘法运算，并得到正确的乘积。



M_OP1 = 00001111,M_OP2 = 00001111,PRO_H=0000000,PRO_L = 01234321。



M_OP1 = 00001111,M_OP2 = 00001111,PRO_H=00000000,PRO_L = 048d0c84。



M_OP1= 00002222， M_OP2 = FFFFFFFF， PRO_H + PRO_L = FFFFFFFF FFFFDDDE.

实验结果表明，改进后的乘法器能够正确地进行两位乘法运算，并得到正确的乘积。

上箱验证如图，可知达到要求。

6、 总结感想

（说说本次实验的总结感想）

①乘法运算可用移位和加法来实现，当两个四位数相乘，总共需做四次加法和四次移位。
②由乘数的末位值确定被乘数是否与原部分积相加，然后右移一位，形成新的部分积；同时，乘数也右移一位，由次低位作新的末位，空出最高位放部分积的最低位。
③每次做加法时，被乘数仅仅与原部分积的高位相加，其低位被移至乘数所空出的高位位置

迭代乘法算法的原理是将两个数相乘，分解为两个数各自的高位和低位相乘的结果，然后再将这些结果相加得到最终结果。

例如，对于两个三位数相乘的情况，可以将它们分解为四个两位数相乘的结果，然后将这些结果相加得到最终结果。

具体步骤如下：

将两个数分别表示为 a 和 b，其中 a 的位数为 n，b 的位数为 m。

将 a 和 b 分别分解为高位和低位，例如 a 可以表示为 a1 * 10^(n/2) + a2，b 可以表示为 b1 * 10^(m/2) + b2，其中 a1 和 b1 为高位，a2 和 b2 为低位。

对于 a1 和 b1，使用迭代乘法算法计算它们的乘积 c1。

对于 a2 和 b2，使用迭代乘法算法计算它们的乘积 c2。

对于 a1 和 a2 的乘积 c3，使用迭代乘法算法计算它们的乘积 c4。

对于 b1 和 b2 的乘积 c5，使用迭代乘法算法计算它们的乘积 c6。

将 c1 * 10^(n) + (c4 + c5) * 10^(n/2) + c2 作为最终结果返回。