组成原理实验课程第 二 次实验报告

实验名称	定点乘法			班级	李涛老师
学生姓名	辛浩然	学号	2112514	指导老师	董前琨
实验地点	实验楼 A 区 304			实验时间	2023/3/28

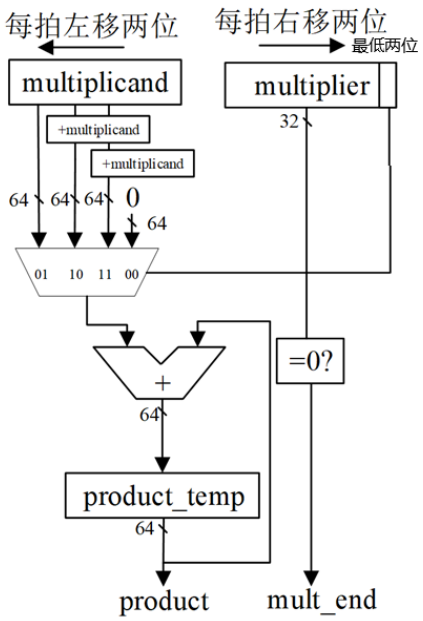
1 实验目的

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. 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. 将改进后的乘法器进行仿真验证。
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。

3 实验原理图



图中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。

迭代乘法是在模拟我们人算乘法的过程，乘数每次右移两位，被乘数每次左移两位。根据乘数的最低两位：如果乘数最低两位是 00，部分积为 0；如果乘数最低两位是 01，部分积为被乘数的

值；如果乘数最低两位是 10，部分积为被乘数的值的二倍（通过加法实现，即两个被乘数相加）；如果乘数最低两位是 11，部分积为被乘数的值的三倍。乘积不停地累加部分积最后就得到结果了。可以看到迭代乘法是用多次加法完成乘法操作的，故需要多拍时间，其结束标志为乘数所有位全为 0，故该乘法器需要 16 拍能完成一次乘法。

4 实验步骤

4.1 乘法器的实现

修改代码，使乘数每次**右移两位**，被乘数每次**左移两位**，根据**乘数最低两位**判断乘积累加的值。根据乘数的最低两位：如果乘数最低两位是 00，部分积为 0；如果乘数最低两位是 01，部分积为被乘数的值；如果乘数最低两位是 10，部分积为被乘数的值的二倍（通过加法实现，即两个被乘数相加）；如果乘数最低两位是 11，部分积为被乘数的值的三倍。

代码修改如下：

```
// 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'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'b0,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

// 部分积
```

```
wire [63:0] partial_product;
assign partial_product = multiplier[1] ? (multiplier[0] ? (multiplicand
    + multiplicand + multiplicand) : (multiplicand + multiplicand)):(
   multiplier[0] ? multiplicand : 64'd0);
// 三目运算符，先判断乘积倒数第二位，再判断乘积最后一位
```

4.2 修改 LCD 屏幕显示数据的位置

修改 display 文件输出到触摸屏显示部分。

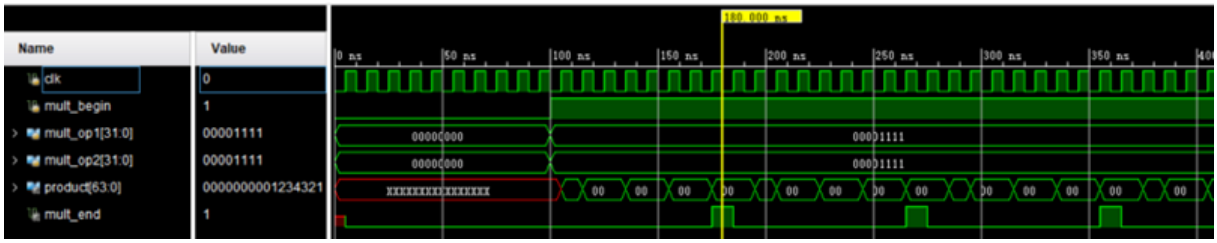
修改 case 语句中的 display_number. display_number 就是输出到外部说明当前需要显示的区域块为第几块，有效编号从 1 44，指示 44 块显示区域块。对其进行修改，实现位置显示的变换。

```
always @(posedge clk)
begin
   case(display_number)
      6'd5 :   // 修改，在第5块显示
      begin
         display_valid <= 1'b1;
         display_name  <= "M_OP1";
         display_value <= mult_op1;
      end
      6'd7 :   // 修改，在第7块显示
      begin
         display_valid <= 1'b1;
         display_name  <= "M_OP2";
         display_value <= mult_op2;
      end
      6'd9 :   // 修改，在第9块显示
      begin
         display_valid <= 1'b1;
         display_name  <= "PRO_H";
         display_value <= product_r[63:32];
      end
      6'd10 :   // 修改，在第10块显示
      begin
         display_valid <= 1'b1;
         display_name  <= "PRO_L";
         display_value <= product_r[31: 0];
      end
      default :
      begin
         display_valid <= 1'b0;
         display_name  <= 48'd0;
         display_value <= 32'd0;
```

```
33          end
34      endcase
35  end
```

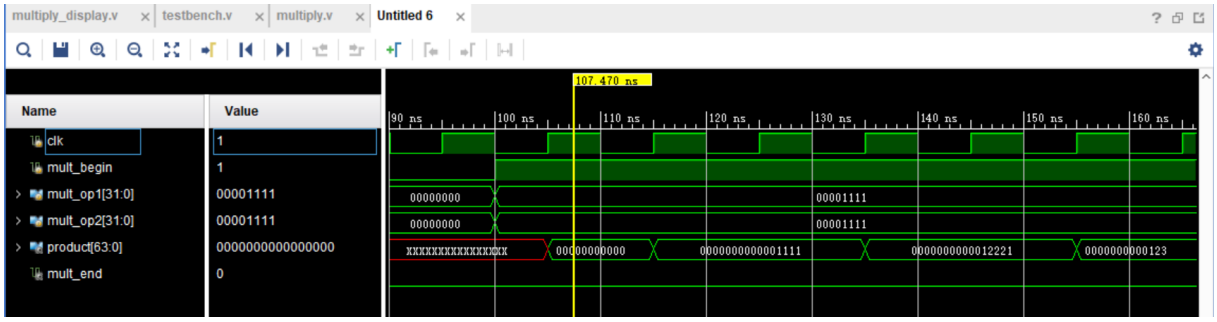
5 实验结果分析

5.1 仿真波形结果

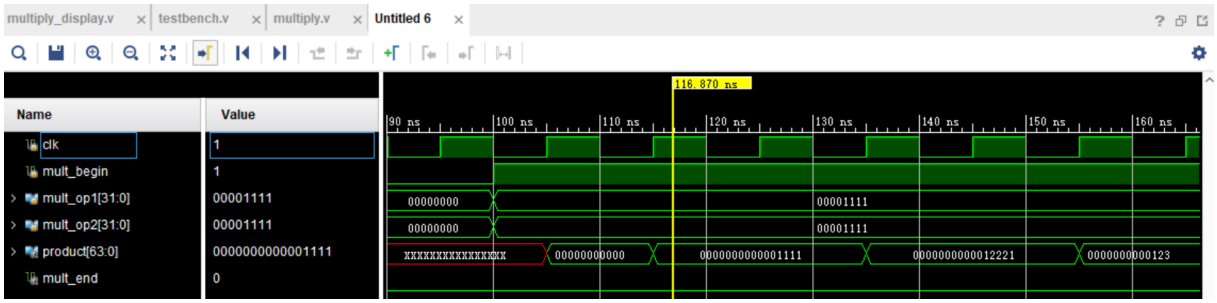


当 mult_end 为 1 时，product 为最终的乘积计算结果：00001111×00001111 = 0000000001234321，结果正确。

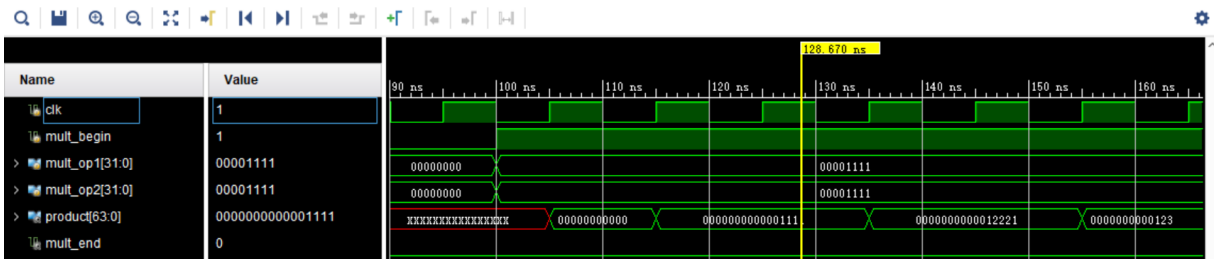
然后分析一下乘法过程：



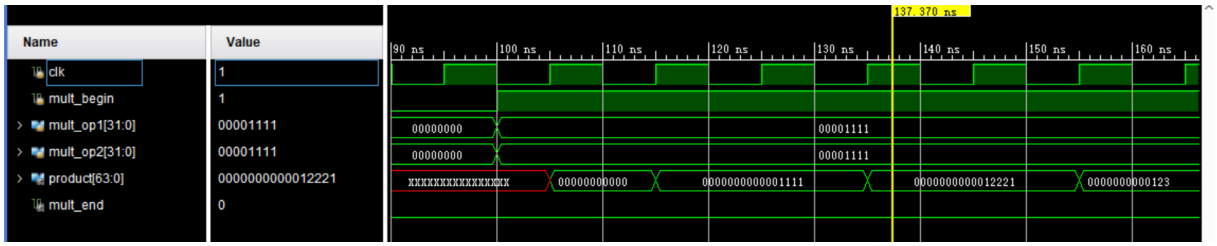
在 mult_begin 为 1 后，第一个时钟脉冲将乘积 product 置为 0。



上图为乘法开始后的第一个脉冲，乘数后两位为 01，部分积为被乘数的值即 1111(64 位，省略了前面的 0)，此时的乘积 product 为被乘数的值。

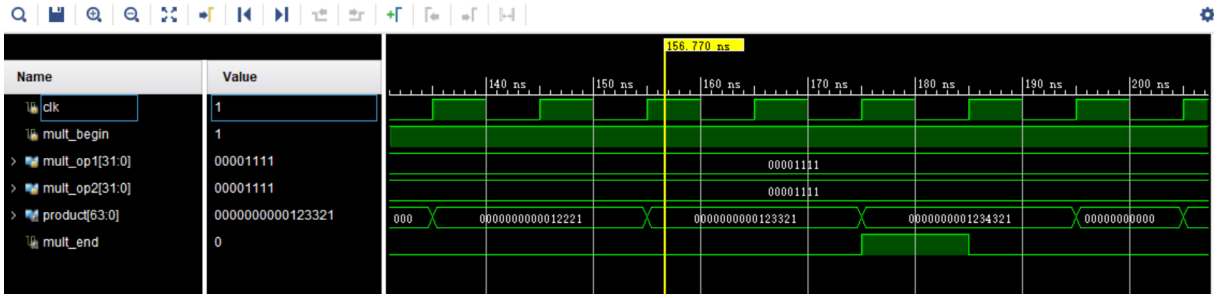


如上图，第二个脉冲后，乘数向右移动两位，被乘数左移两位，此时被乘数最后两位为 00，部分积为 0，此时的乘积 product 不变。



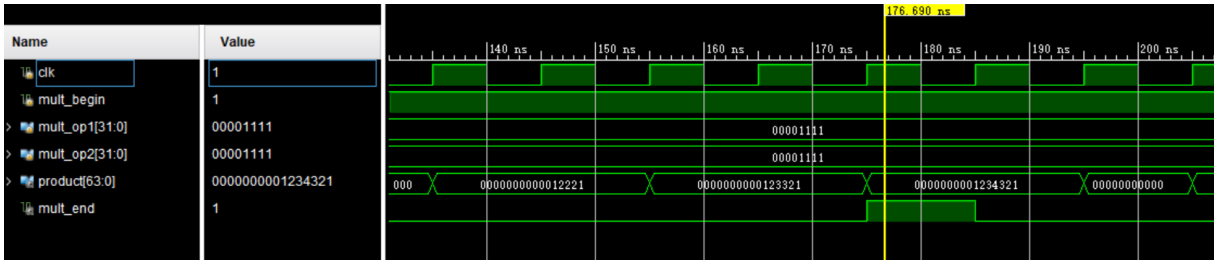
如上图，第三个脉冲后，乘数向右移动两位，被乘数左移两位，此时被乘数最后两位为 01，部分积为被乘数的值即 11110(64 位，省略了前面的 0)，乘积 product 加上 11110，值为 12221(64 位，省略了前面的 0)。

第四个脉冲后，乘数右移两位，最后两位为 00，product 值不变。



如上图，第五个脉冲后，乘数向右移动两位，被乘数左移两位，此时被乘数最后两位为 01，部分积为被乘数的值即 111100(64 位，省略了前面的 0)，乘积 product 加上 111100，值为 123321(64 位，省略了前面的 0)。

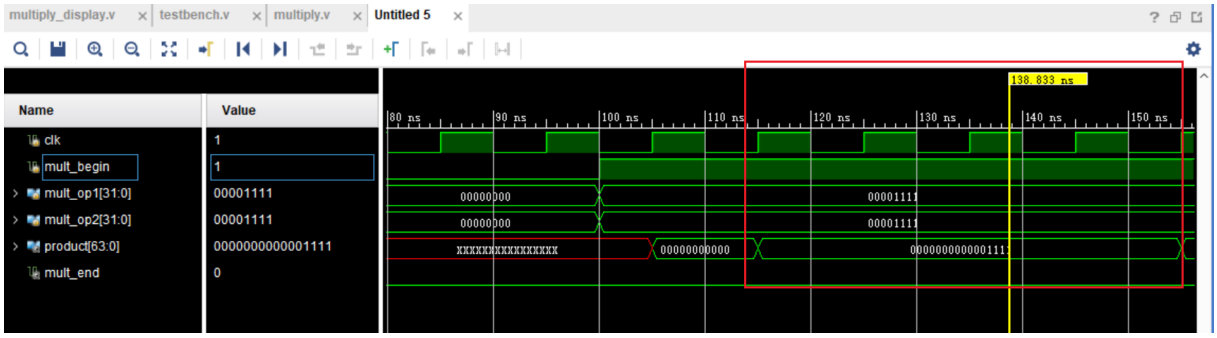
第六个脉冲后，乘数右移两位，最后两位为 00，product 值不变。



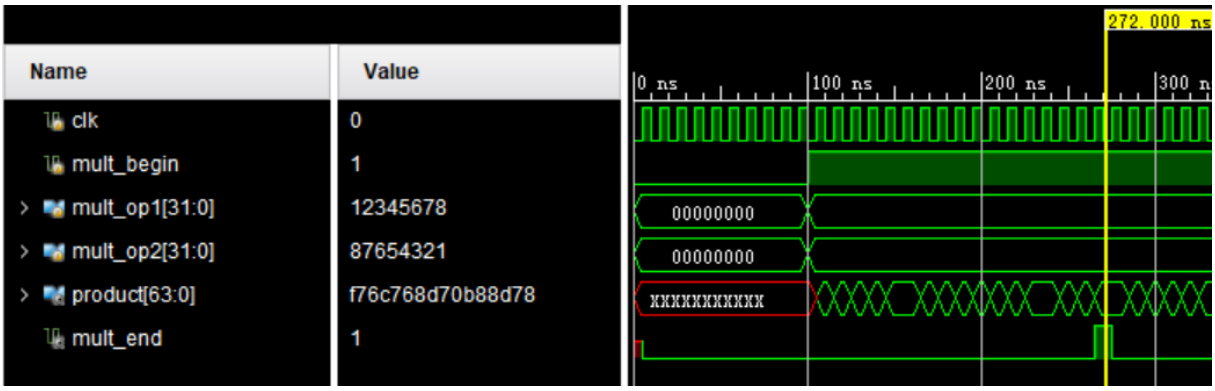
如上图，第七个脉冲后，乘数向右移动两位，被乘数左移两位，此时被乘数最后两位为 01，部分积为被乘数的值即 1111000(64 位，省略了前面的 0)，乘积 product 加上 1111000，值为 1234321(64 位，省略了前面的 0)。

此时乘数所有位都为 0，mult_end 置 1，乘法结束。

对比原乘法器，如下图，是原乘法器生成的仿真图像。可以发现，乘积由 1111 变为 12221 时，需要四个时钟脉冲，也就是乘数移动四位。而改进后的乘法器只需要两个时钟脉冲。每次乘法时钟脉冲数减少了一半，实现了对乘法器的优化。



修改 testbench 文件，修改被乘数为 1234 5678，乘数为 8765 4321，得到如下仿真结果：

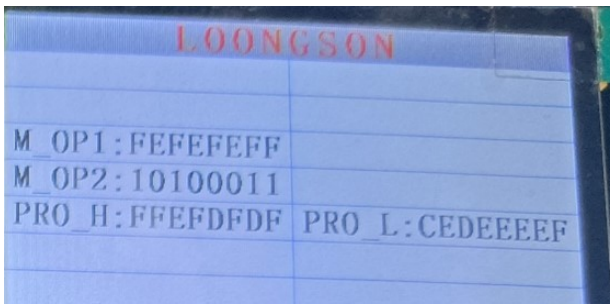


可以发现，经历 16 个时钟脉冲就实现了乘法操作。

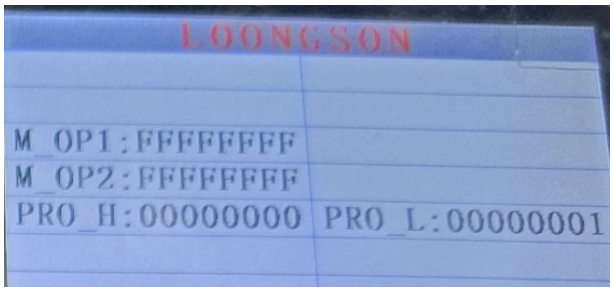
5.2 实验箱结果

在 display 文件中，将显示位置修改为第 5、7、9、10 个位置，通过实验箱显示可以发现，显示位置修改正确。

在实验箱验证中，拨码开关最左侧的开关用来选择触摸屏输入的数据为乘数 1 还是乘数 2。

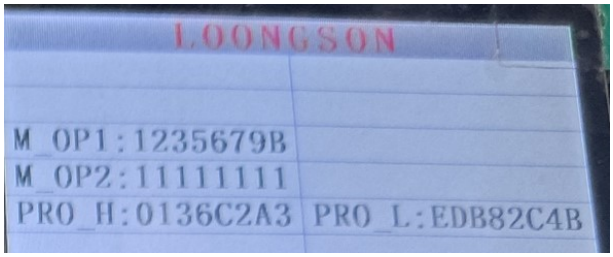


被乘数 FEFE FEFF，为负数，其绝对值为 0101 0101；乘数 1010 0011，为正数；乘积绝对值为 0010 2020 3121 1111。其为负数，最终乘积为 FFEF DFDF CEDE EEEF，结果正确。



被乘数 FFFF FFFF，为负数，其绝对值为 0000 0001；乘数 FFFF FFFF，为负数，其绝对值

为 0000 0001；乘积为正数，为 0000 0000 0000 0001，结果正确。



被乘数 1235 679B，为正数；乘数 1111 1111，为正数；乘积为正数，为 0136 C2A3 EDB8 2C4B，结果正确。

由上述实验箱结果可验证乘法器的正确性。

6 总结感想

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. 熟悉了 LS-CPU-EXB-002 实验箱和软件平台。
3. 掌握了利用实验箱各项功能开发组成原理和体系结构实验的方法。
4. 进一步熟悉并掌握了 Verilog 语言的一些关键语法。