组成原理实验课程第_2_次实验报告

实验名称	定点乘法			班级	李涛老师
学生姓名	孙蕗	学号	2112060	指导老师	董前琨
实验地点	A308		实验时间	2023.4.4	

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、 实验设备

（1） 装有 Xilinx Vivado 的计算机一台。

（2） LS-CPU-EXB-002 教学系统实验箱一套。

3、 实验要求

（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。（其他形式的优化也建议尝试）

（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

（4） 实验报告中的原理图为迭代乘法的算法图，不再是顶层模块图

（5） 使用三目运算符实现

4、 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，详细标出输入输出端口，本次实验的乘法器建议采用迭代的方式实现，如果能力有余的，也可以采用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。

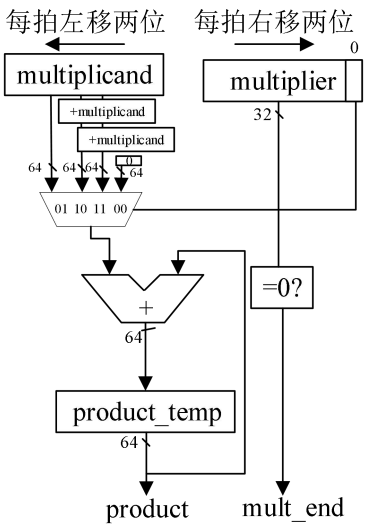
（3） 根据设计的实验方案，使用 verilog 编写相应代码。

（4） 对编写的代码进行仿真，得到正确的波形图。

（5） 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，见图 3.1。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。

（6） 将编写的代码进行综合布局布线，并下载到实验箱中的 FPGA 板子上进行演示

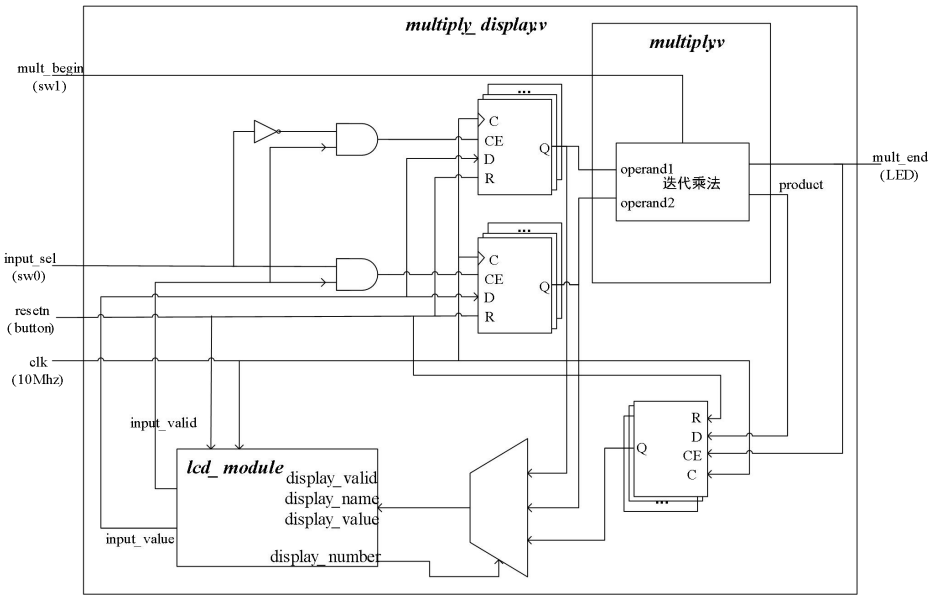
5、 实验原理图



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乘数每次右移两位，被乘数每次左移两位。

判断乘数的最低两位，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 2 倍；如果乘数最低两位是 11，部分积为被乘数的值的 3 倍。乘积不停地累加部分积即可就得到结果。当乘数所有位全为 0，标志计算完成。

定点乘法参考设计的顶层模块图：



6、 实验步骤

A. multiply.v 修改（使用 case 判断）

（1） multiply.v 修改，实现被乘数每次左移两位

42	//加载被乘数，运算时每次左移两位
43	reg [63:0] multiplicand;
44	always @ (posedge clk)
45	begin
46	if (mult_valid)
47	begin // 如果正在进行乘法，则被乘数每时钟左移两位

```
48                  multiplicand <= {multiplicand[61:0],2'b0};
49              end
50              else if (mult_begin)
51              begin     // 乘法开始，加载被乘数，为乘数 1 的绝对值
52                  multiplicand <= {32'd0,op1_absolute};
53              end
54          end
```

（2） multiply.v 修改，乘数每次右移两位

```
56      //加载乘数，运算时每次右移两位
57          reg    [31:0] multiplier;
58          always @ (posedge clk)
59          begin
60              if (mult_valid)
61              begin     // 如果正在进行乘法，则乘数每时钟右移两位
62                  multiplier <= {2'b0,multiplier[31:2]};
63              end
64              else if (mult_begin)
65              begin     // 乘法开始，加载乘数，为乘数 2 的绝对值
66                  multiplier <= op2_absolute;
67              end
68          end
```

（3）

```
70  // 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;
71          reg [63:0] partial_product;
72          always@(multiplier[1:0])
73          begin
74              case(multiplier[1:0]) // 根据乘数最低两位判断
75                  2'd00: // 乘数最低两位为 00，部分积为 0
76                          partial_product <= 64'd0;
77                  2'd01: // 乘数最低两位为 01，部分积为被乘数的值
78                          partial_product <= multiplicand;
79                  2'd11: // 乘数最低两位为 11，部分积为被乘数的值的三倍
80                          partial_product <= multiplicand + multiplicand +multiplicand;
81                  2'd10: // 乘数最低两位为 10，部分积为被乘数的值的二倍
82                          partial_product <= multiplicand + multiplicand;
83              endcase
84          end
```

B. multiply.v 修改（使用三目运算符）

（1） multiply.v 修改，实现被乘数每次左移两位

```
42      //加载被乘数，运算时每次左移两位
43          reg    [63:0] multiplicand;
44          always @ (posedge clk)
45          begin
```

```
46          if (mult_valid)
47          begin       // 如果正在进行乘法，则被乘数每时钟左移两位
48              multiplicand <= {multiplicand[61:0],2'b00};
49          end
50          else if (mult_begin)
51          begin     // 乘法开始，加载被乘数，为乘数 1 的绝对值
52              multiplicand <= {32'd0,op1_absolute};
53          end
54      end
```

（2） multiply.v 修改，乘数每次右移两位

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

（3）

```
70  // 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;
71      wire[63:0] partial_product;//64 位数，保存部分积
72    assign partial_product = multiplier[1] ? (multiplier[0]?multiplicand+multiplicand+
    multiplicand:multiplicand+multiplicand):(multiplier[0]?multiplicand:64'd0);//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 2 倍；如果乘数最低两位是 11，部分积
    为被乘数的值的 3 倍。乘积不停地累加部分积即可就得到结果。
```

C. multiply_display.v

修改 multiply_display.v，改变实验箱上输出的格子位置。

```
129 //-----{输出到触摸屏显示}begin
130 //根据需要显示的数修改此小节，
131 //触摸屏上共有 44 块显示区域，可显示 44 组 32 位数据
132 //44 块显示区域从 1 开始编号，编号为 1~44，
133     always @(posedge clk)
134     begin
135         case(display_number)
136             6'd5 ://第 5 个格子显示
137             begin
```

```
138                         display_valid <= 1'b1;
139                         display_name    <= "M_OP1";
140                         display_value <= mult_op1;
141                 end
142                 6'd6 ://第 6 个格子显示
143                 begin
144                         display_valid <= 1'b1;
145                         display_name    <= "M_OP2";
146                         display_value <= mult_op2;
147                 end
148                 6'd7 ://第 7 个格子显示
149                 begin
150                         display_valid <= 1'b1;
151                         display_name    <= "PRO_H";
152                         display_value <= product_r[63:32];
153                 end
154                 6'd8 ://第 8 个格子显示
155                 begin
156                         display_valid <= 1'b1;
157                         display_name    <= "PRO_L";
158                         display_value <= product_r[31: 0];
159                 end
160                 default :
161                 begin
162                         display_valid <= 1'b0;
163                         display_name    <= 48'd0;
164                         display_value <= 32'd0;
165                 end
166             endcase
167         end
```

7、 实验结果分析

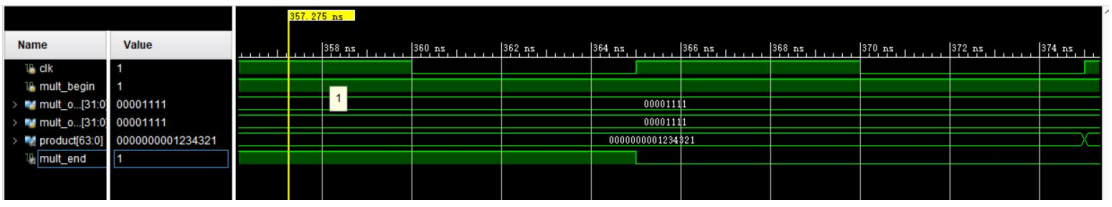
（1） 仿真波形分析

multiplicand=00001111

multiplier=00001111

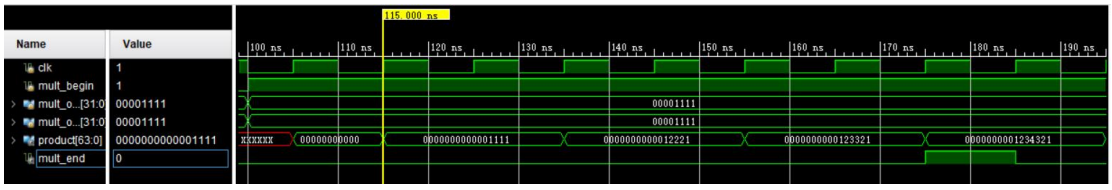
00001111×00001111=0000 0000 0123 4321

A. 仿真波形最终结果



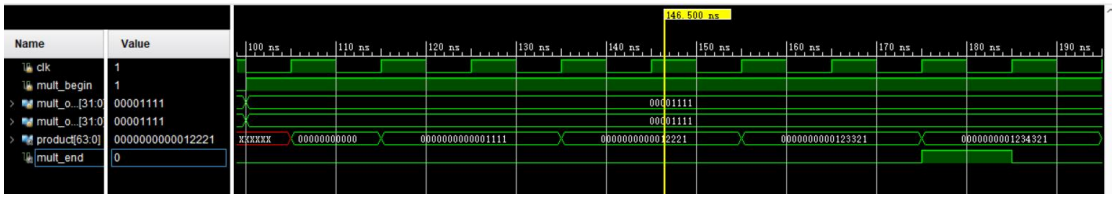
B. 过程分析

（1） 第一个脉冲，乘数 multiplier 后两位为 01，部分积 partial_product 应为 0000 0000 0000 1111，即被乘数的值 1111，此时的 product 也为被乘数的值 0000 0000 0000 1111。



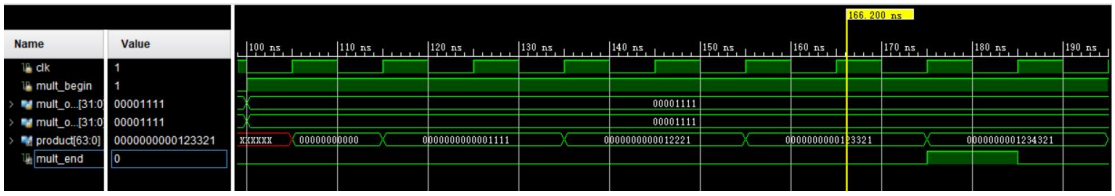
（2） 第二个脉冲，乘数 multiplier 向右移动两位，被乘数 multiplicand 向左移动两位，乘数 multiplier 最后两位为 00，部分积 partial_product 为 0，此时乘积 product 不变。

（3） 第三个脉冲，乘数 multiplier 向右移动两位，被乘数 multiplicand 向左移动两位，乘数 multiplier 最后两位为 01，部分积 partial_product 为被乘数的值 0000 0000 0001 1110，乘积 product 应加上 partial_product,变成 0000 0000 0001 2221。

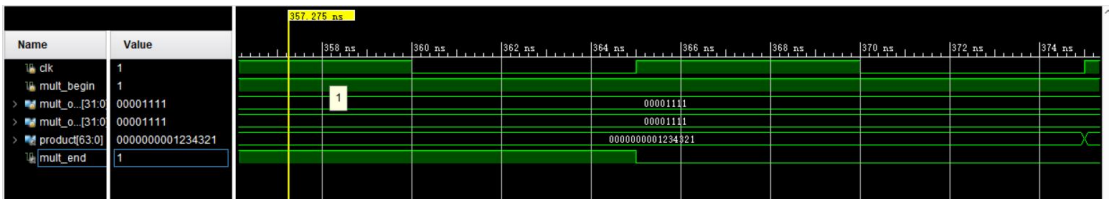


（4） 第四个脉冲，乘数 multiplier 向右移动两位，被乘数 multiplicand 向左移动两位，乘数最后两位为 00，partial_product=0，product 不变。

（5） 乘数 multiplier 向右移动两位，被乘数 multiplicand 向左移动两位，乘数最后两位为 01，部分积 partial_product 为被乘数的值 0000 0000 0011 1100，乘积 product 应加上 partial_product,变成 0000 0000 0012 3321。



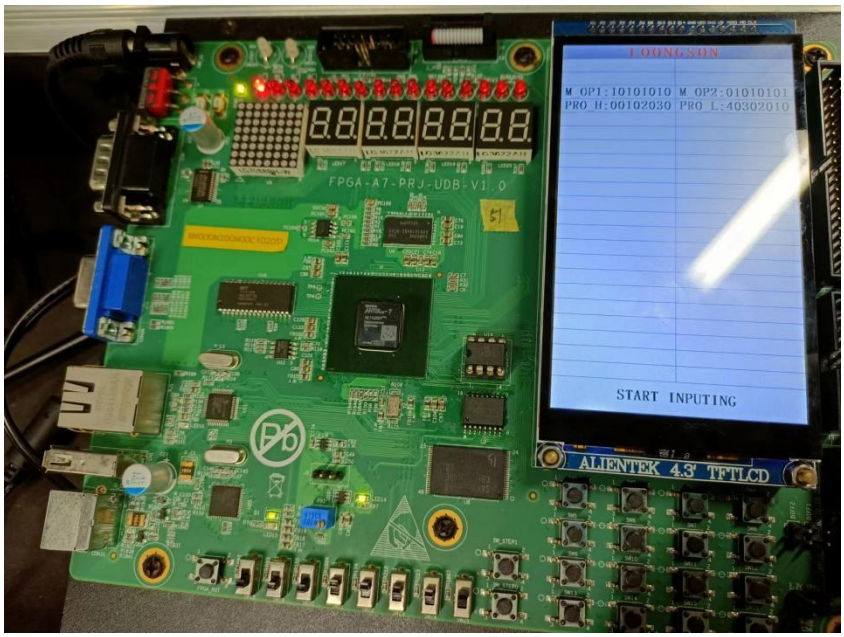
（6） 乘数 multiplier 向右移动两位，被乘数 multiplicand 向左移动两位，乘数最后两位为 01，部分积 partial_product 为被乘数的值 0000 0000 0111 1000，乘积 product 应加上 partial_product,变成 0000 0000 0123 4321。



（2） 实验箱

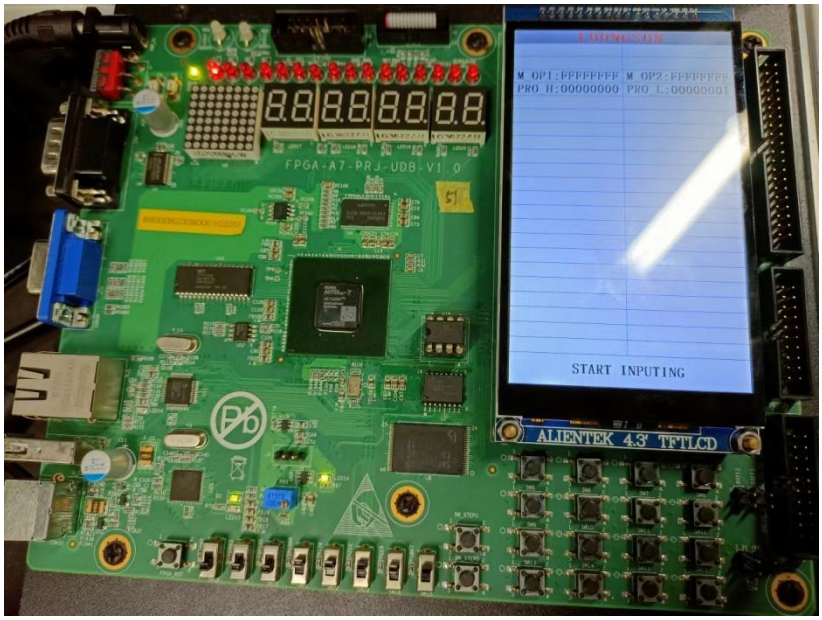
M_OP1 显示在第 5 格，M_OP2 显示在第 6 格，PRO_H 显示在第 7 格，PRO_L 显示在第 8 格。

A.



被乘数为 10101010，为正数
乘数为 01010101，为正数
乘积为 0010 2030 4030 2010，结果正确

B.



被乘数为 FFFF FFFF，是负数，绝对值为 0000 0001
乘数位 FFFF FFFF，是负数，绝对值为 0000 0001
乘积为正数，为 0000 0000 0000 0001，结果正确

8、 总结感想

通过这个实验，对 Verilog 的应用能力有了更深入的了解和提升。通过将每个时钟周期的移位改为移两位，可以提高计算的精度和效率。同时，使用三目运算符可以简化代码的编写和理解，使代码更加清晰。在修改移位时，需要注意运算的溢出问题；在使用三目运算符时，需要注意不同运算符的优先级和使用方式。