组成原理实验课程第_二_次实报告

实验名称	数据运算：定点加法			班级	李涛老师
学生姓名	王昱	学号	2212046	指导老师	董前琨
实验地点	A306		实验时间	2024.3.28	

一、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。

二、实验内容说明

1. 做好预习：

1） 掌握定点乘法的多种实现算法的原理；

2） 确定定点乘法的输入输出端口设计；

3） 在课前画好设计框图或实验原理图；

4） 如果对 FPGA 板了解的话，可确定设计中与 FPGA 板上交互的接口，画出包含外围模块的整体设计框图，即补充完善图 3.1。

2. 实验实施：

1） 确认定点乘法的设计框图的正确性；

2） 编写 verilog 代码；

3） 对该模块进行仿真，得出正确的波形，截图作为实验报告结果一项的材料；

4） 完成调用定点乘法模块的外围模块的设计，并编写代码；

5） 对代码进行综合布局布线下载到实验箱里 FPGA 板上，进行上板验证。

3. 实验检查：

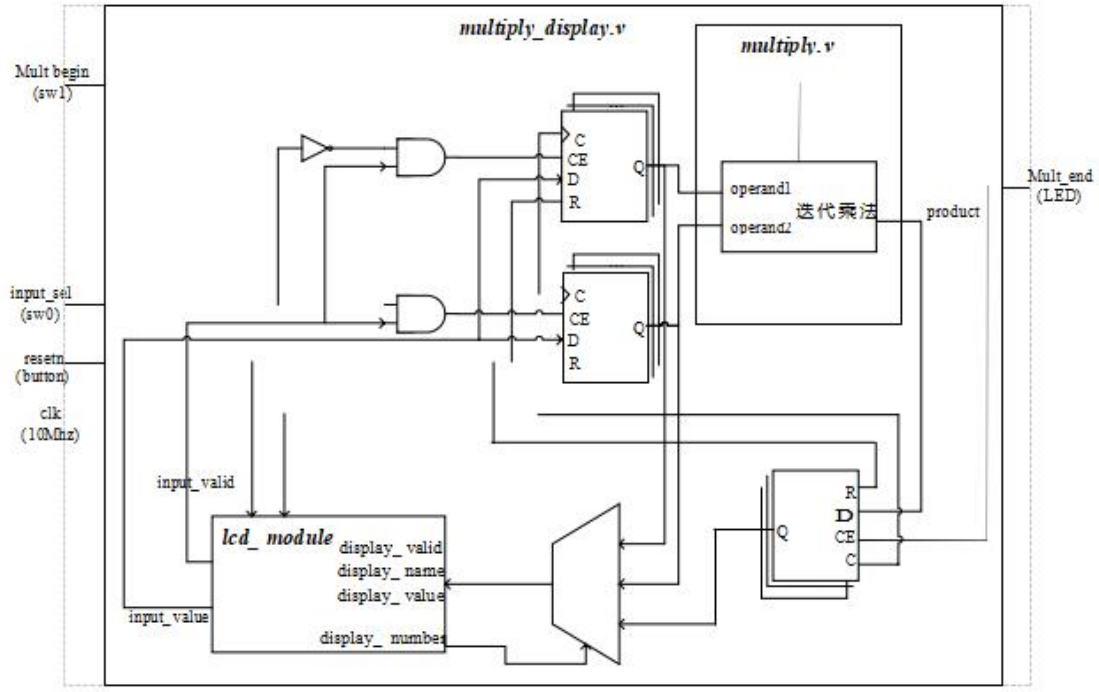
1） 完成上板验证后，让指导老师或助教进行检查，进行现场演示，可对演示结果进行拍照作为实验报告结果一项的材料。

4. 实验报告的撰写：

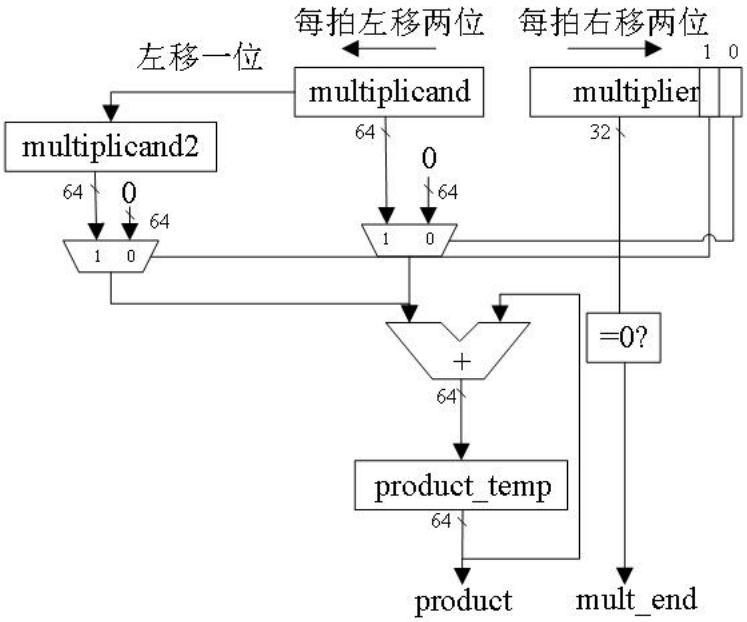
1） 实验结束后，需按照规定的格式完成实验报告的撰写。

三、实验原理图

乘法器顶层模块框图：



实验原理图：



实验原理图改进说明：

迭代乘法是在模拟我们人算乘法的过程，乘数每次右移一位，根据最低位，判断是加被乘数移位后的值还是加 0，不停地累加最后就得到乘积了。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，其结束标志为乘数移位后为 0，故对于 32 位乘法，如

果每次只移一位的话最多需要 32 拍才能完成一次乘法。

本次实验的目的是将乘法器改成被乘数每次左移两位，从而减少一半的运算周期。由实验原理图可知：改进后，每次移两位，这样最多需要 16 拍即可完成一次乘法。乘法开始时，将被乘数扩展为六十四位，高位补 0。每一拍将被乘数左移两位，乘数右移两位。然后对乘数的末尾两位分别进行判断：

（1）乘数末位为 1，将乘数本身与结果相加；乘数末位为 0，将结果加 0；

（2）乘数倒数第二位为 1，将被乘数左移一位与结果相加得到；乘数倒数第二位为 0，将结果加 0。

当乘数每一位都为 0 时，程序结束。累计求和得到的就是相乘的结果。

四、实验步骤

1.修改关键代码：

主要修改 multipler.v 模块：

（1）将被乘数从每个周期左移一位改进为左移两位：

```
//加载被乘数，运算时每个时钟周期左移两位
reg   [63:0] multiplicand;
      always @ (posedge clk)
      begin
            if (mult_valid)
            begin
                  multiplicand <= {multiplicand[61:0],2'b00};
            end
            else if (mult_begin)
            begin
                  multiplicand <= {32'd0,op1_absolute};
            end
      end
```

解释说明：

乘法开始时(mult_begin 为真），将被乘数寄存器向左移动 32 位，将高 32 位补零；

运算过程中（mult_valid 为真），将被乘数向左移动两位，低位用 0 补充，为下一次部分积的加法做准备。

（2）将乘数从每个周期右移一位改进为右移两位：

```
//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
                  multiplier <= {2'b00,multiplier[31:2]};
            end
```

```
else if (mult_begin)
            begin
                  multiplier <= op2_absolute;
            end
      end
```

解释说明：

乘法开始时(mult_begin 为真），将乘数寄存器向加载为 op2_absolute 的值；

运算过程中（mult_valid 为真），将被乘数向右移动两位，高位用 0 补充，为下一次部分积的加法做准备。

（3）修改部分积的计算方法：

```
 // 部分积
      wire [63:0] partial_product1;
       wire [63:0] partial_product2;
       assign partial_product1 = multiplier[0] ? multiplicand : 64'd0;
       assign partial_product2 = multiplier[1]?{multiplicand[62:0],1'b0} :
64'd0;
```

解释说明：

partial_product1 是根据 multiplier 的最低位（multiplier[0]）来计算的：

（1）如果是 1，partial_product1 就等于 multiplicand 的值；

（2）如果是 0，partial_product1 就等于 0。

partial_product2 是根据 multiplier 的第二低位（multiplier[1]）来计算的：

（1）如果是 1，partial_product2 就等于 multiplicand 左移一位的结果；

（2）如果是 0，partial_product2 就等于 0。

这相当于在乘数的对应位为 1 时，将被乘数乘以 2 然后加到部分积上。

（4）累加器的修改：

```
//累加器
      reg [63:0] product_temp;
      always @ (posedge clk)
      begin
            if (mult_valid)
            begin
                  product_temp <= product_temp + partial_product1+
partial_product2;
            end
            else if (mult_begin)
            begin
                  product_temp <= 64'd0;
            end
      end
```

解释说明：

这部分对部分积的结果进行累加：

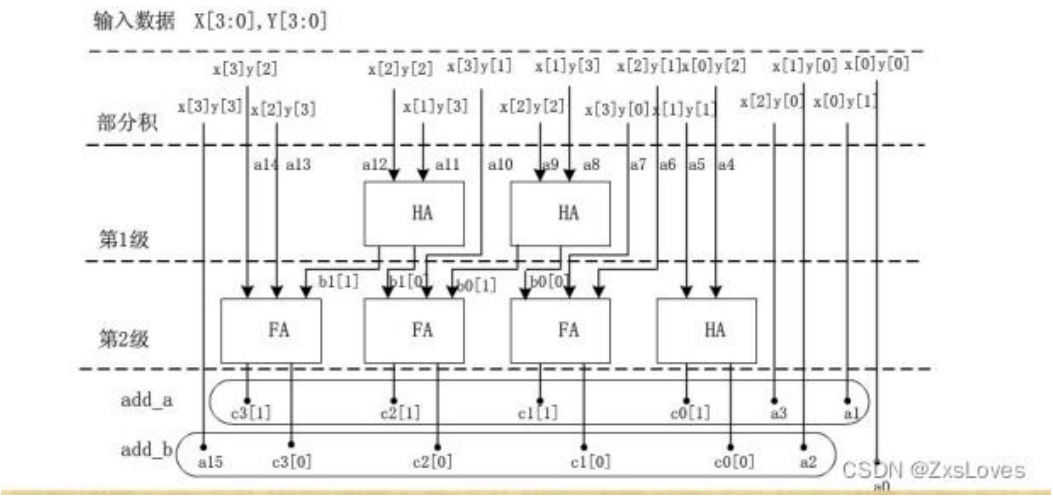
乘法开始时(mult_begin 为真），将积寄存器向扩展为 64 位并且清零；

运算过程中（mult_valid 为真），product_temp 寄存器会将其当前值与两个部分积 partial_product1 和 partial_product2 相加。

2.华莱士树优化

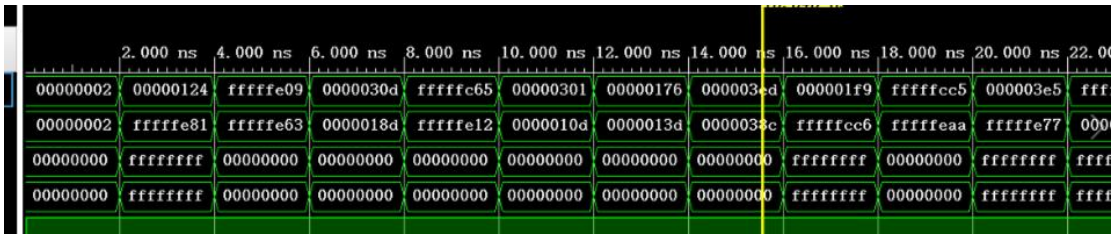
华莱士树乘法器的设计思想是将乘法操作分解为多个部分积的加法操作。每个部分积都是由两个二进制数的某一位相乘得到的。简单地讲即许多个加数求和，每 3 个加数分为一组，压缩至 2 个加数，循环往复。

在乘法器中，乘法的积为许多个部分和之和。Wallace 结构可以加快乘法器的计算速度。为了提高计算效率，华莱士树乘法器采用了一种特殊的结构。在 32 位乘法中，全加器个数为 14。这样的设计保证了在首位上有足够的空闲进位要求。



（以上为华莱士树四位乘法器的结构）

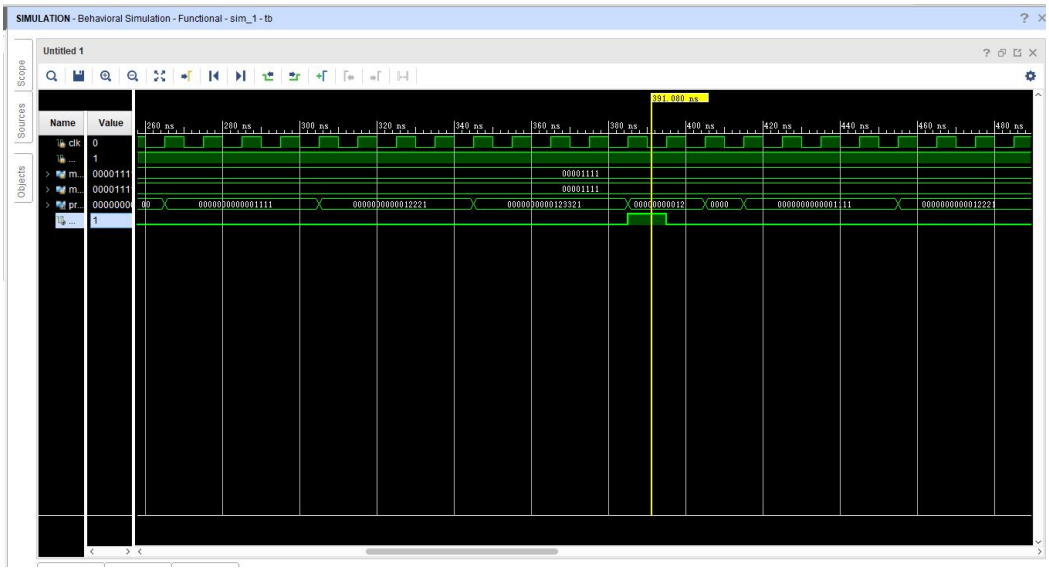
仿真结果：



3.实验结果分析

(1)仿真结果截图与分析：

一位乘法器：



解释说明：

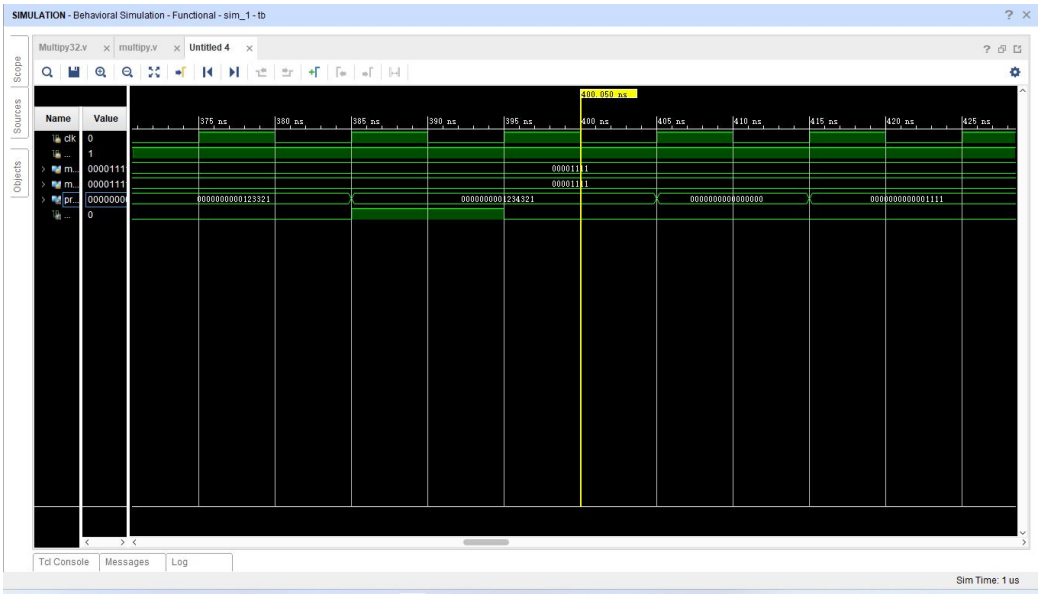
M_OP1=00001111
M_OP2=00001111

用 python 程序计算得：00001111*00001111=000000001234321

由图可知当 muti_end=1 时，结果为 000000001234321

因此可知实验结果是正确的。

二位乘法器：



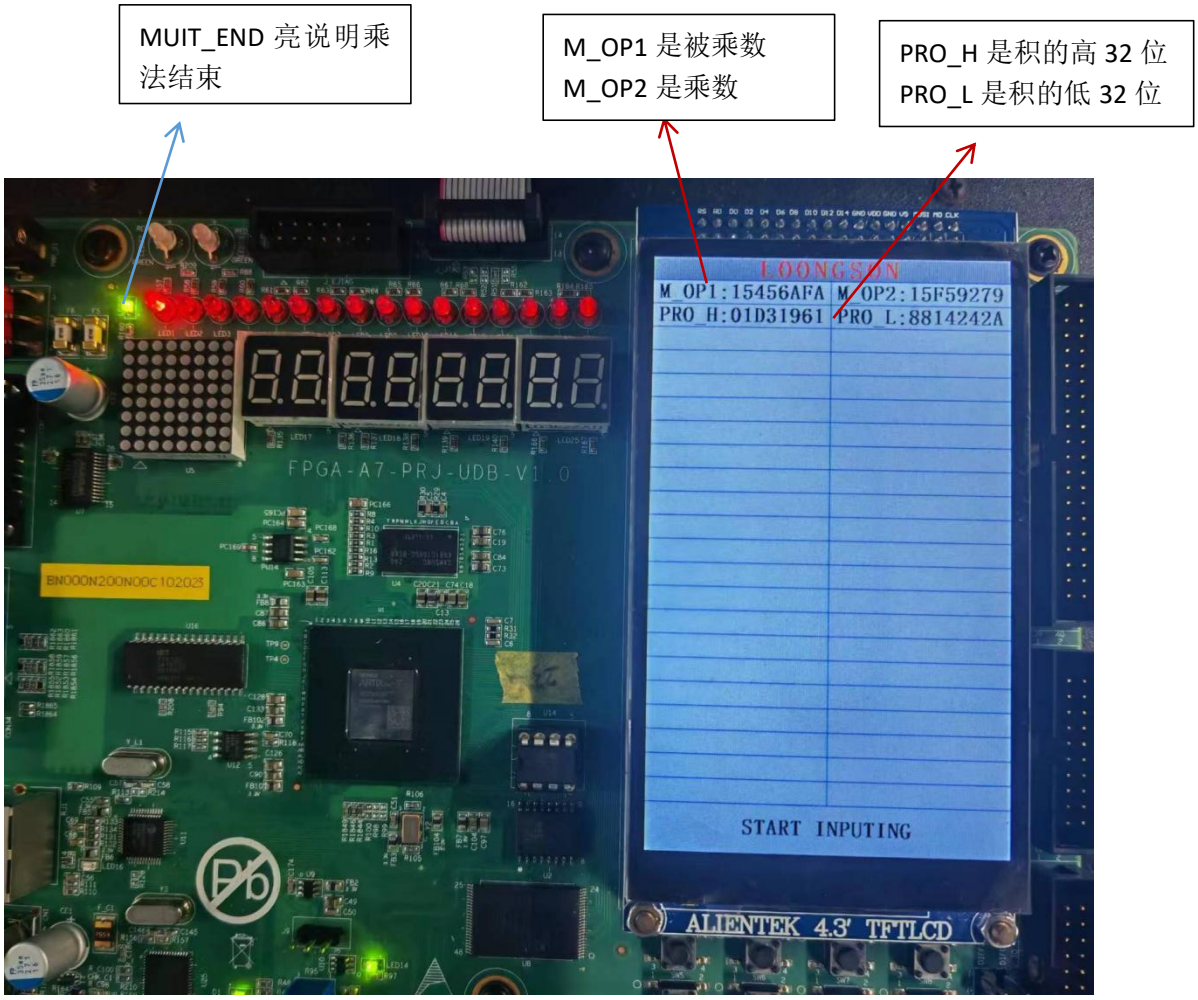
- **解释说明：**
  同上。

- **结果分析：**

由仿真图像可知，计算相同的结果，一位乘法器所用的时钟周期是两位乘法器所用的时钟周期的二倍，因此将最多用 32 个周期的乘法器优化为最多用 16 个周期的乘法器，提升了乘法器的性能，达到了更快的运行速度。

(2)实验箱结果验证：

（二位乘法器）



结果分析：

左起第一个 LED 灯亮，说明运算结束。

15456AFA*15F59279=01D319618814242A

程序验证结果：

```
0x1d319618814242a
```

结果相同，可知乘法器运算结果是正确的。

五.总结感想

（1）通过老师的指导以及自己课后的学习，对 verilog 语言的理解与掌握更进一步。

（2）对 vivado 软件和实验箱的使用更加得心应手，为之后的实验打下了牢固的基础。

（3）深刻体会了 32 位乘法器的实现原理。并且学会如何用 verilog 如何实现一个 32 位迭代加法器，并且通过自己进行修改，改进为了效率更高的乘法器。

（4）这次实验也让我认识到了理论知识与实际应用之间的联系。在计组课堂上学到的抽象概念，在实验中得到了具体的应用，这种实践经验对于我的学习过程非常宝贵。

（3）学习了更加高效的乘法器改进方法——华莱士树，对我启发很大。