**组成原理实验课程第 2次实报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | 数据运算：定点乘法 | | | 班级 | 李涛 |
| 学生姓名 | 张丛 | 学号 | 2113662 | 指导老师 | 董前琨 |
| 实验地点 | B308 | | 实验时间 | 2023.04.04 | |

1. **实验目的**
2. 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。

1. **实验内容说明**

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。

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。

实验检查：

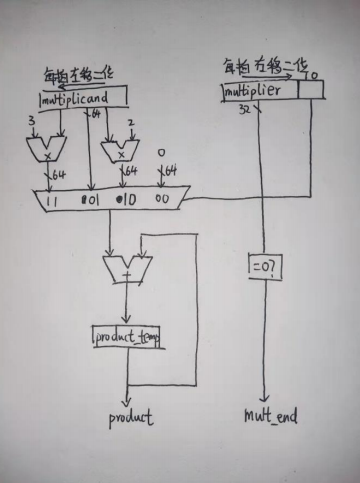
1. 完成上板验证后，让指导老师或助教进行检查，进行现场演示，可对演

示结果进行拍照作为实验报告结果一项的材料。

2. 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

1. **实验原理图**



每次周期，被乘数左移两位，乘数右移两位。

根据乘数最后两位的值，确定累加器加数的值。

分四种情况：

00,加数为0；

01，加数为multiplicand;

10,加数为multiplicand\*2；

1. 加数为multiplicand\*3。

在代码中可以直接判断乘数最后两位，也可以一位一位判断。

当乘数所有为均为0时，改变信号，乘法结束

1. **实验步骤**
2. 编写乘法运算代码，将移位一位变为两位

module multiply( // 乘法器

input clk, // 时钟

input mult\_begin, // 乘法开始信号

input [31:0] mult\_op1, // 乘法源操作数1

input [31:0] mult\_op2, // 乘法源操作数2

output [63:0] product, // 乘积

output mult\_end // 乘法结束信号

);

reg mult\_valid;

assign mult\_end = mult\_valid & ~(|multiplier); //当乘数全为0时，乘法结束

always @(posedge clk)

begin

if (!mult\_begin || mult\_end)

begin

mult\_valid <= 1'b0;

end

else

begin

mult\_valid <= 1'b1;

end

end

//两个源操作取绝对值

wire op1\_sign; //操作数的符号位

wire op2\_sign;

wire [31:0] op1\_absolute; //操作数的绝对值

wire [31:0] op2\_absolute;

assign op1\_sign = mult\_op1[31];

assign op2\_sign = mult\_op2[31];

assign op1\_absolute = op1\_sign ? (~mult\_op1+1) : mult\_op1;

assign op2\_absolute = op2\_sign ? (~mult\_op2+1) : mult\_op2;

//加载被乘数，运算时每次左移2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2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：根据乘数最后两位的情况，确定累加器中加数的值

wire [63:0] partial\_product;

wire [63:0] partial\_product1;

assign partial\_product = multiplier[0] ? multiplicand : 64'd0;

assign partial\_product1 = multiplier[1]?{multiplicand[62:0],1'b0}:64'd0;

//累加器

reg [63:0] product\_temp;

always @ (posedge clk)

begin

if (mult\_valid)

begin

product\_temp <= product\_temp + partial\_product + partial\_product1;

end

else if (mult\_begin)

begin

product\_temp <= 64'd0; // 乘法开始，乘积清零

end

end

//乘法结果的符号位和乘法结果

reg product\_sign;

always @ (posedge clk) // 乘积

begin

if (mult\_valid)

begin

product\_sign <= op1\_sign ^ op2\_sign;

end

end

//若乘法结果为负数，则需要对结果取反+1

assign product = product\_sign ? (~product\_temp+1) : product\_temp;

Endmodule

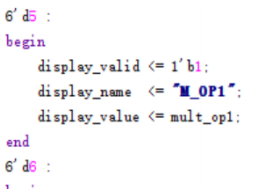
其中：

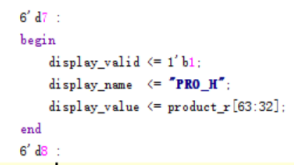
每个周期的被乘数左移两位，代码为multiplicand <= {multiplicand[61:0],2'b0}；

乘数右移两位，代码为multiplier <= {2'b0,multiplier[31:2]}；

修改了部分积这一块代码，累加器中的值将根据乘数的最后两位来确定。

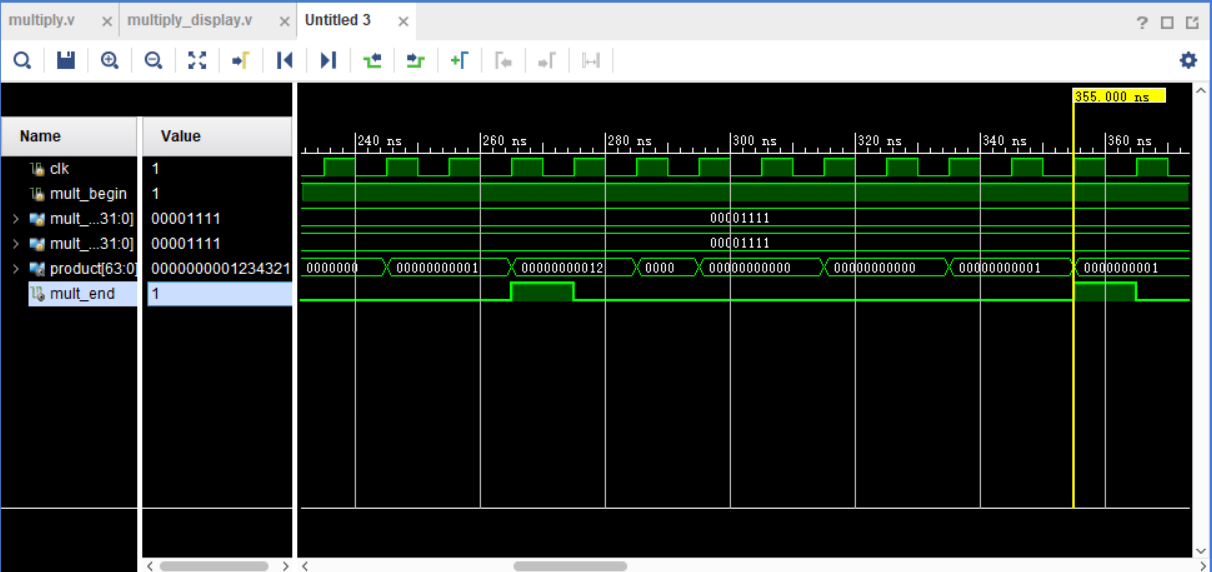
1. 添加仿真文件，进行仿真实验
2. 修改display文件，使得上箱验证时数据不显示在前四格

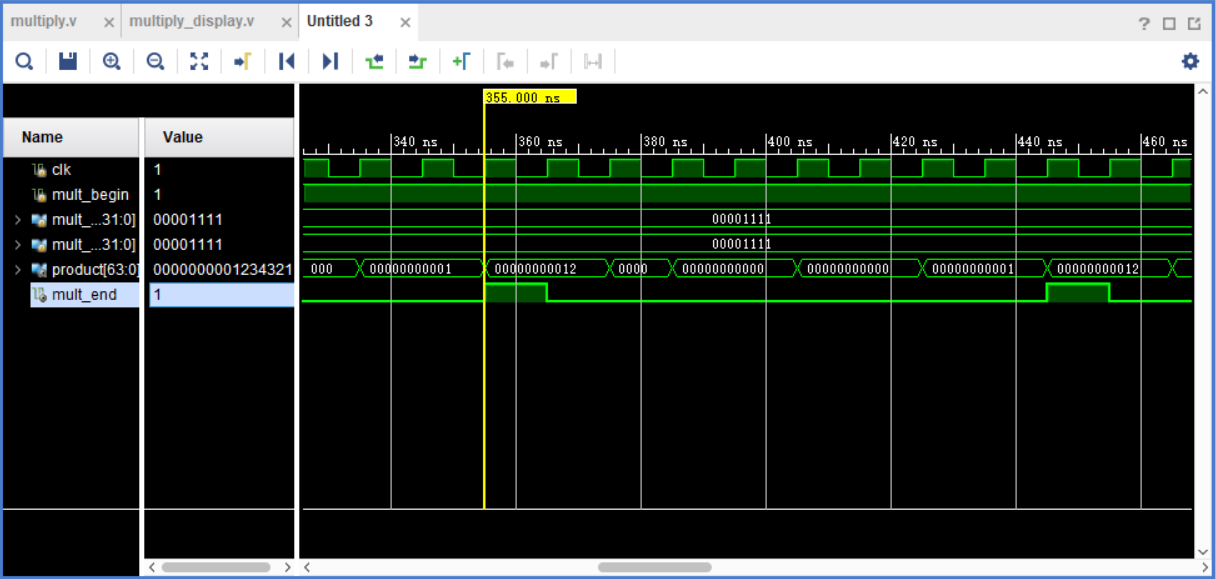




1. 添加展示外围模块，添加约束文件，连接实验箱进行上板验证
2. **实验结果分析**

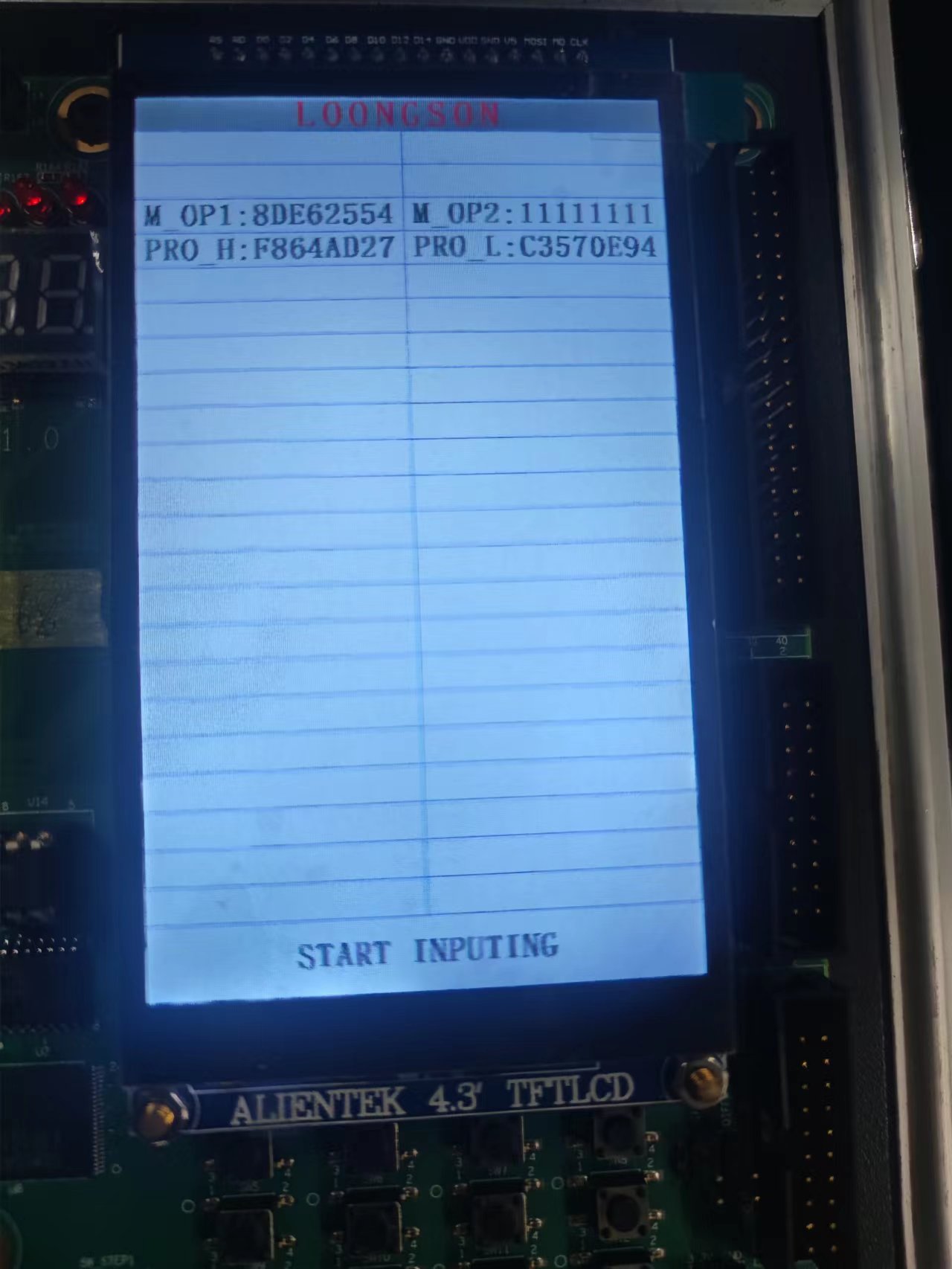
仿真实验：





即：1111\*1111=1234321,结果正确。

上箱验证：





同样可以验证乘法，如二图表示305419896 \* 305419896 = 93281312872650816。

1. **总结感想**

掌握了乘法的运算原理，并且实现了对乘法的时间效率优化。

进一步熟悉了verilog语言，以及迭代的思想。