**组成原理实验课程第二次实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | 定点乘法器改进 | | | 班级 | 张金老师班 |
| 学生姓名 | 姚知言 | 学号 | 2211290 | 指导老师 | 董前琨 |
| 实验地点 | 实验楼A308 | | 实验时间 | 2024年4月7日 | |

1. **实验目的**

理解定点乘法的不同实现算法的原理，掌握基本实现算法。

熟悉并运用 verilog 语言进行电路设计。

为后续设计 cpu 的实验打下基础。

1. **实验内容说明**

掌握定点乘法的多种实现算法的原理。

确定定点乘法的输入输出端口设计。

在课前画好设计框图或实验原理图。

确认定点乘法的设计框图的正确性。

编写 verilog 代码。

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。

完成调用定点乘法模块的外围模块的设计，并编写代码。

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。

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

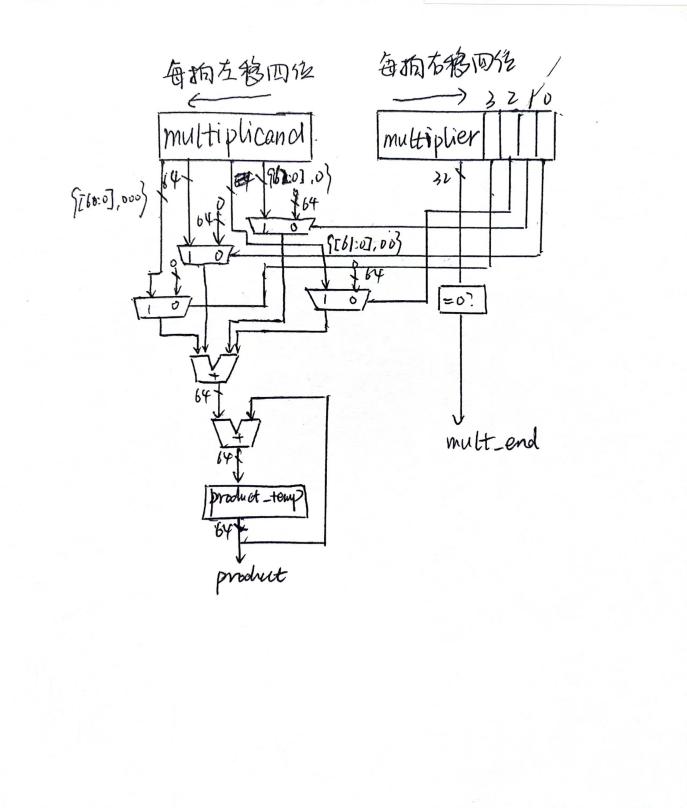
示结果进行拍照作为实验报告结果一项的材料。

对乘法器进行改进，以在更少的时钟周期内完成乘法计算。

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**

四位乘的改装图如下：



1. **实验步骤**

（1）multiply.v

//以四位乘为例进行展示，同理可得更多位乘的设计方法

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

);

......//略去了运算开始信号和结束信号的确立，以及对操作数取符号位和补码的操作

//加载被乘数，运算时每次左移4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4位

multiplicand <= {multiplicand[59:0],4'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，运算时每次右移一位

reg [31:0] multiplier;

always @ (posedge clk)

begin

if (mult\_valid)

begin // 如果正在进行乘法，则乘数每时钟右移一位

multiplier <= {4'b0,multiplier[31:4]};

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

// 对于要处理的4位计算部分积并求和：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，同时根据处理的位数对被乘数进行左移，通过增加此部分被加数的数量，可以更多的增加单个时钟周期中处理的位数

//然而，在实际应用中，时钟周期可能不会这么长，在这种情况下，一味的增加元器件的复杂度并不可取，必定存在一个折中的数量

wire [63:0] partial\_product;

assign partial\_product =(multiplier[0] ? multiplicand : 64'b0)+(multiplier[1] ? {multiplicand[62:0],1'b0} :64'b0)+(multiplier[2] ? {multiplicand[61:0],2'b0} : 64'b0)+(multiplier[3] ? {multiplicand[60:0],3'b0} :64'b0);

......//略去了部分和累加以及负数反码处理的操作

（2）tb.v

module tb;

......//略去了输入输出和乘法器模块引用的操作

initial begin

clk = 0;

mult\_begin = 0;

mult\_op1 = 0;

mult\_op2 = 0;

// Wait 100 ns for global reset to finish

#100;

mult\_begin = 1;

mult\_op1 = 32'H00001111;

mult\_op2 = 32'H00001111;

#400;

mult\_begin = 0;

#500;

mult\_begin = 1;

mult\_op1 = 32'H00001111;

mult\_op2 = 32'H00002222;

#400;

mult\_begin = 0;

#500;

mult\_begin = 1;

mult\_op1 = 32'H00000002;

mult\_op2 = 32'HFFFFFFFF;

#400;

mult\_begin = 0;

#500;

mult\_begin = 1;

mult\_op1 = 32'H00000002;

mult\_op2 = 32'H80000000;

#400;

mult\_begin = 0;

while(1)

begin

#500;

mult\_begin = 1;

mult\_op1 = $random;

mult\_op2 = $random;

#400;

mult\_begin = 0;

end

//由于原模块仅仅对四个固定乘法组进行了验证，这有利于我们确定结果的正确性，但为了保证设计的正确性，仍然增加了随机数生成器以验证普遍现象

end

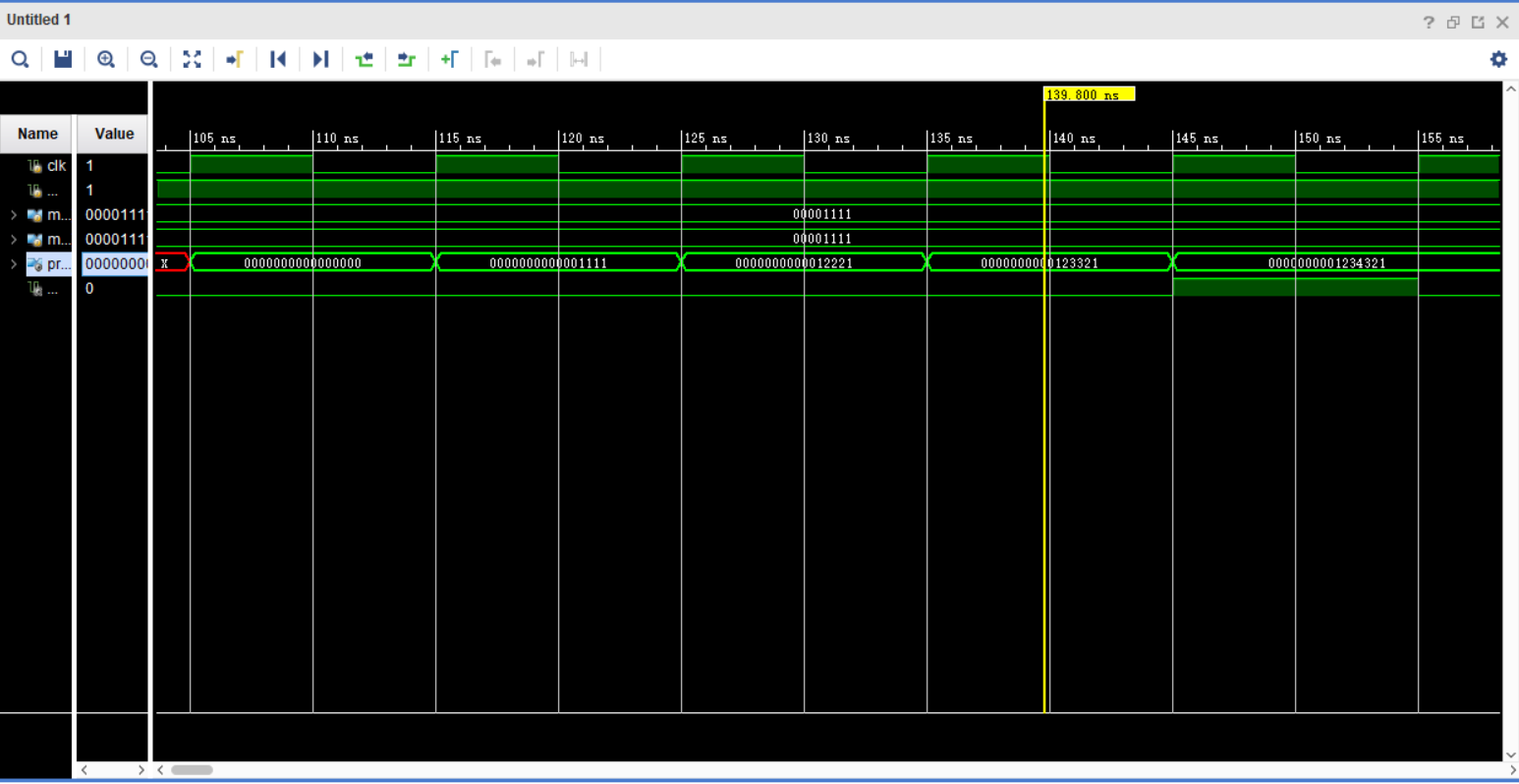
always #5 clk = ~clk;

endmodule

1. **实验结果分析**

（1）仿真结果

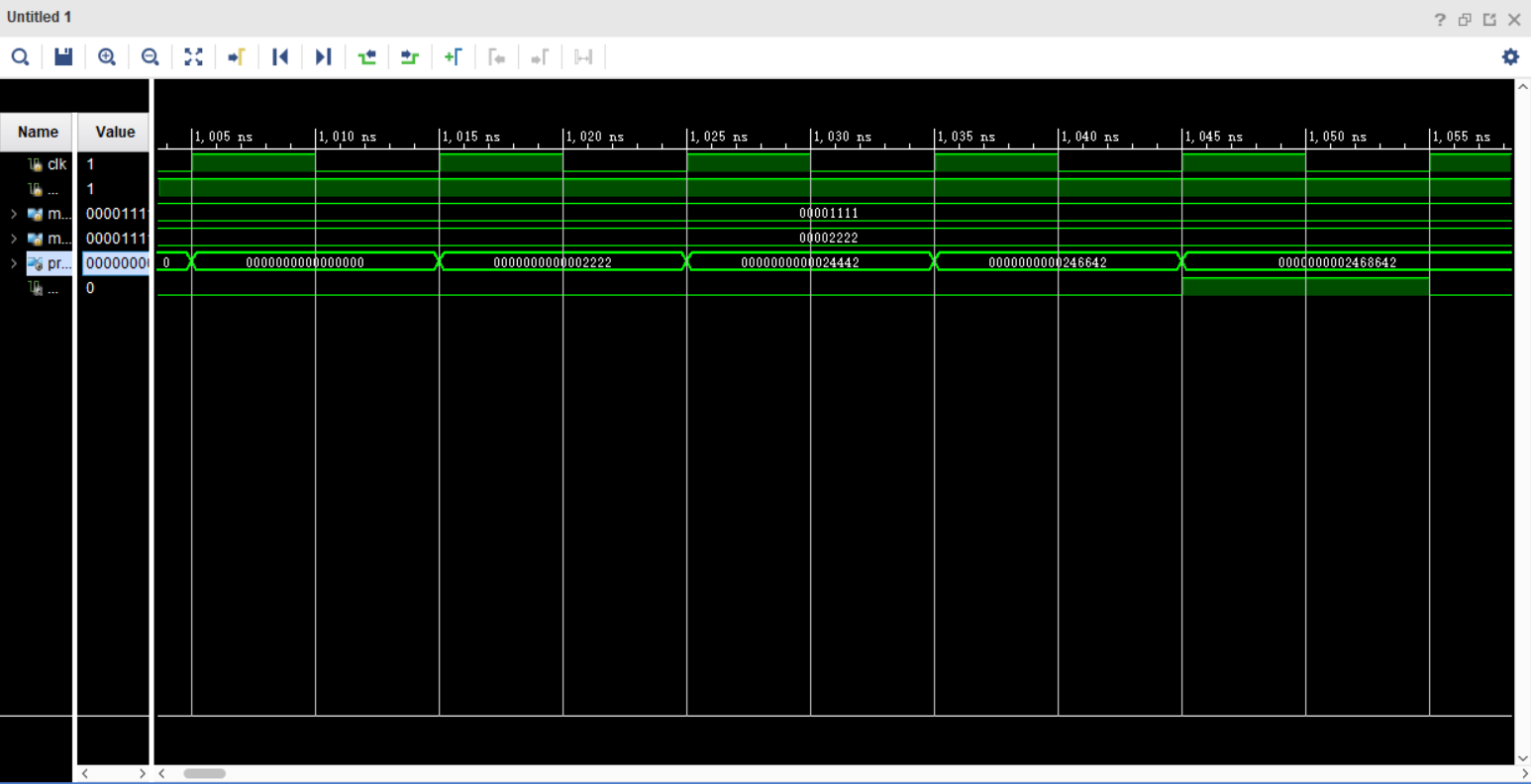
i.时间周期分析



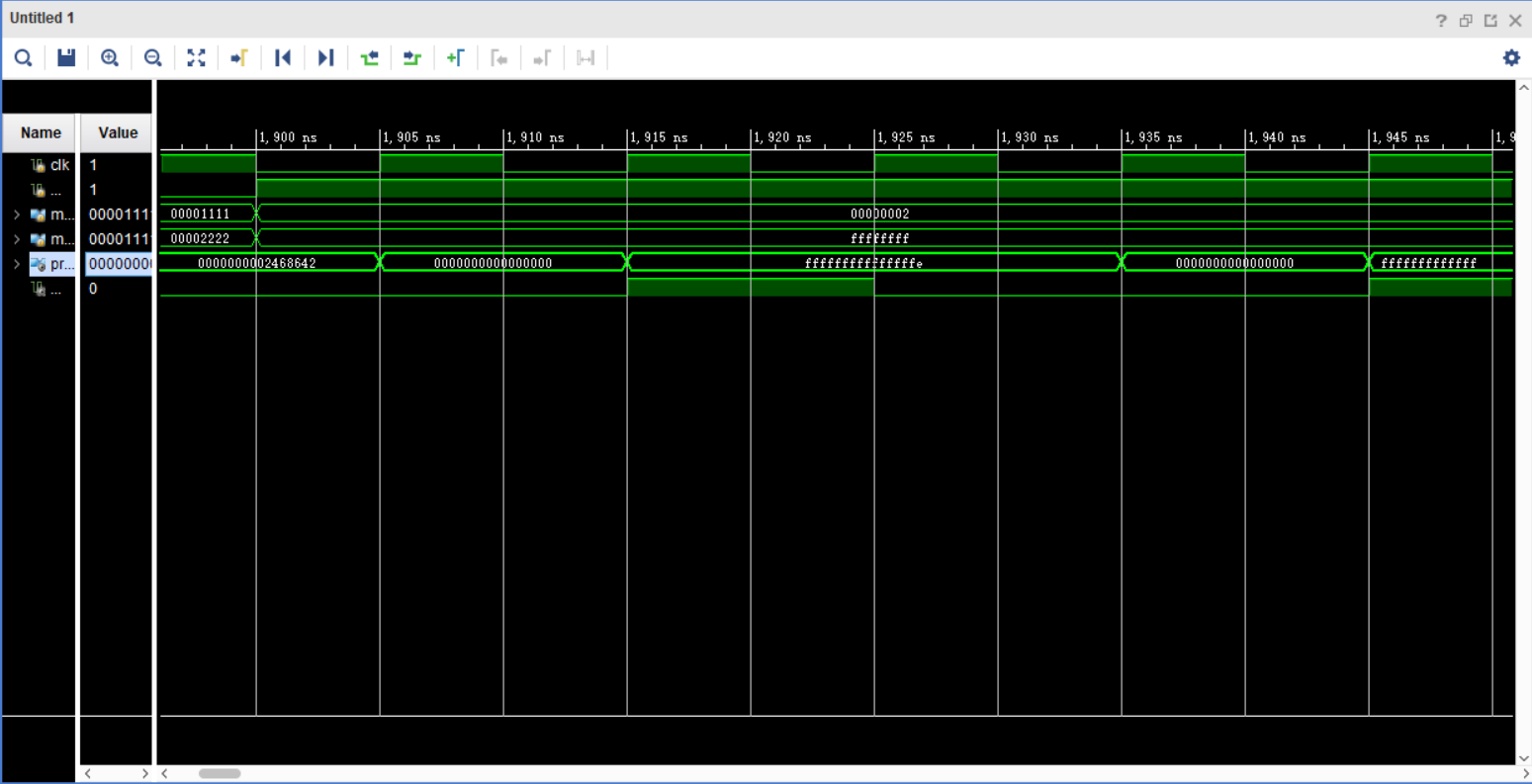
根据对数据0x1111与0x1111的乘积样例分析，仅仅经过4个周期就得到了正确的结果0x1234321，每个周期后的累加结果分别是0x1111，0x12221，0x123321，0x1234321。

既验证了改进乘法器运算逻辑的正确性，也验证了该算法确实是单次进行4位的计算。

ii.对其他样例数据组的分析

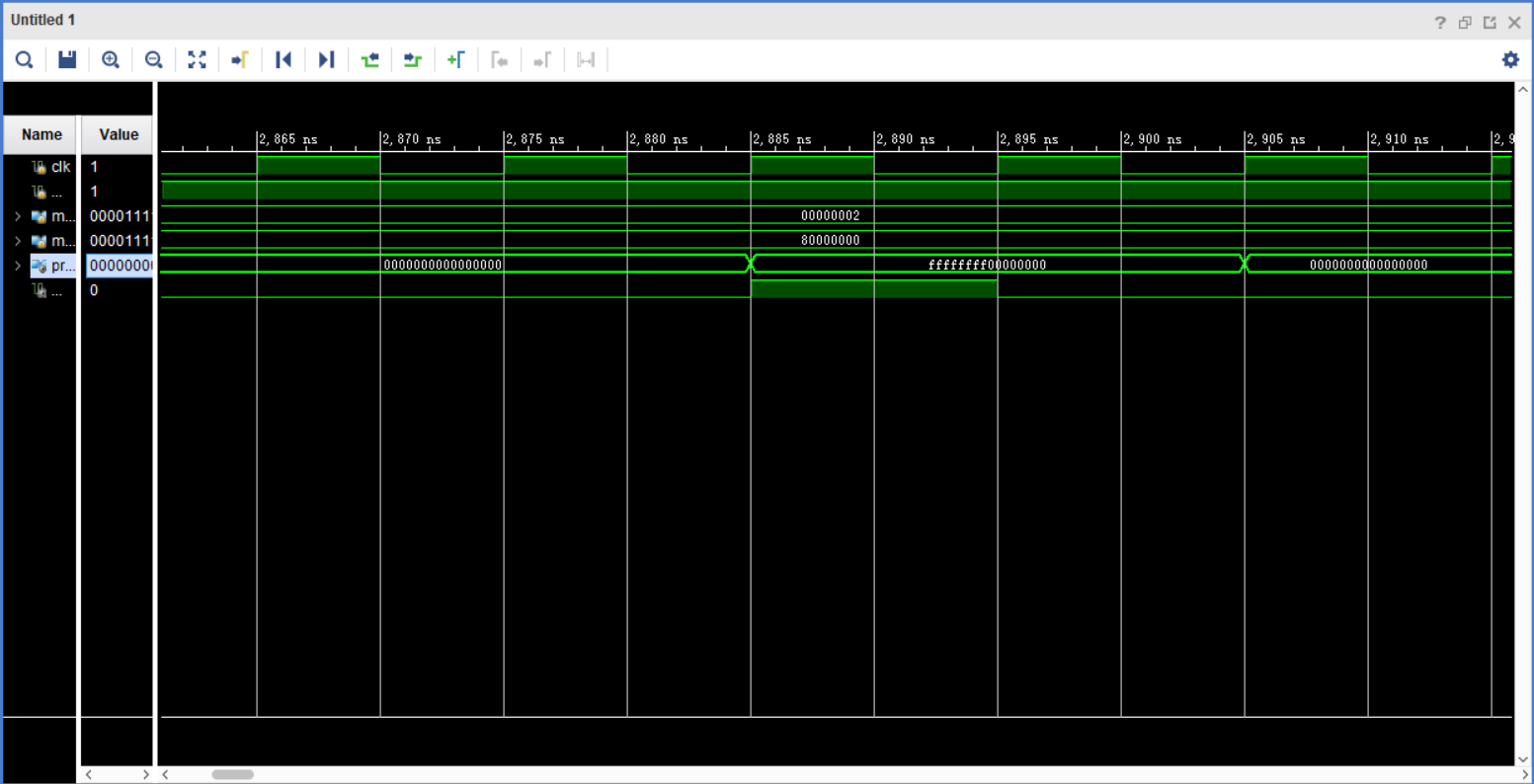


0x1111与0x2222的乘积为0x2468642（4周期），正确。



0x2与0xffffffff的乘积为0xfffffffffffffffe（1周期），正确。

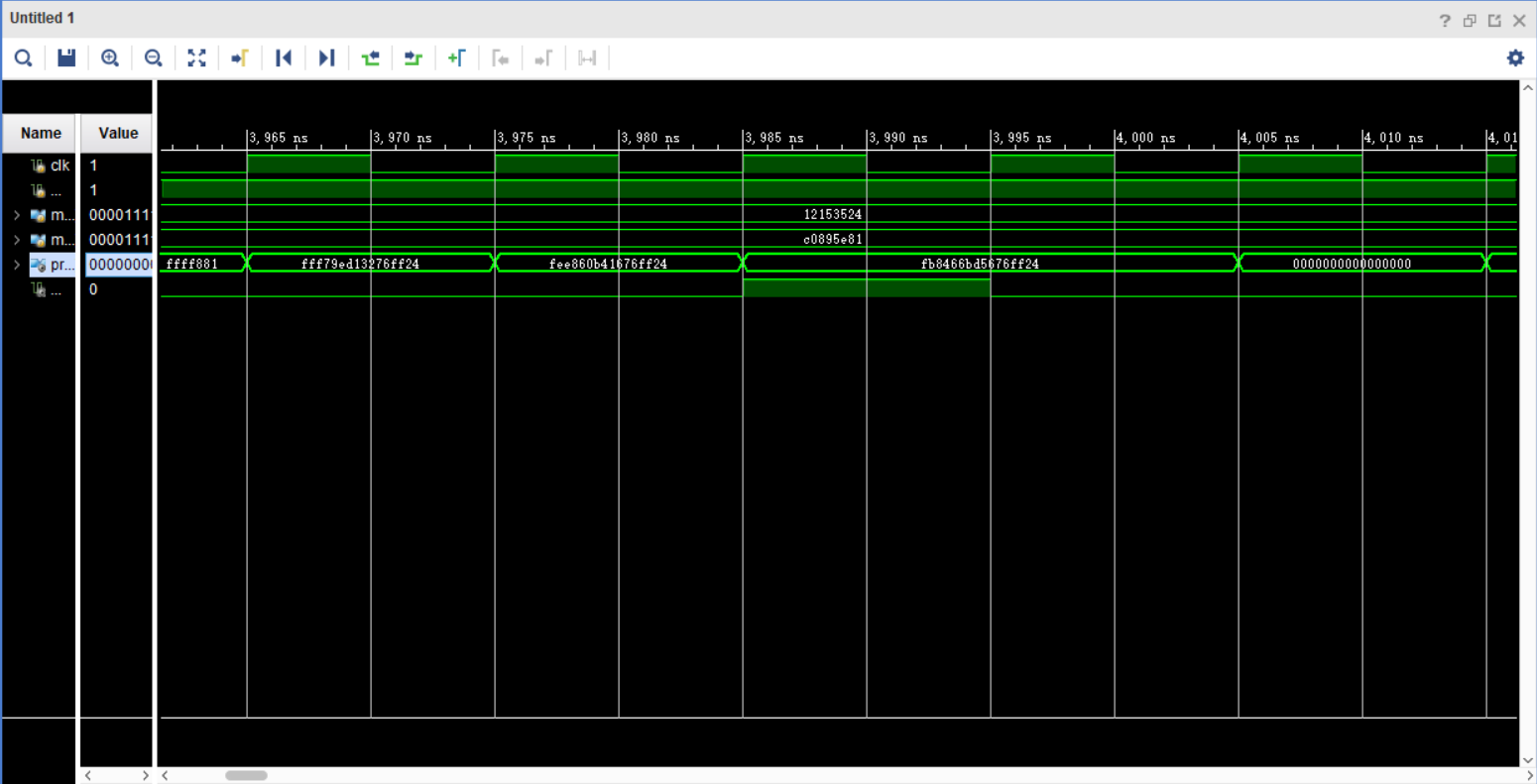
说明：0xffffffff的符号位为1，是负数，补码为0x1（所以只需要1周期），补码相乘结果为0x2，结果是负数，所以取补码为0xfffffffffffffffe。



0x2与0x80000000的乘积为0xfffffff00000000（8周期），正确。

说明：0x80000000的符号位为1，是负数，补码为0x80000000（8周期），补码相乘结果为0x100000000，结果是负数，所以取补码为0xffffffff00000000。

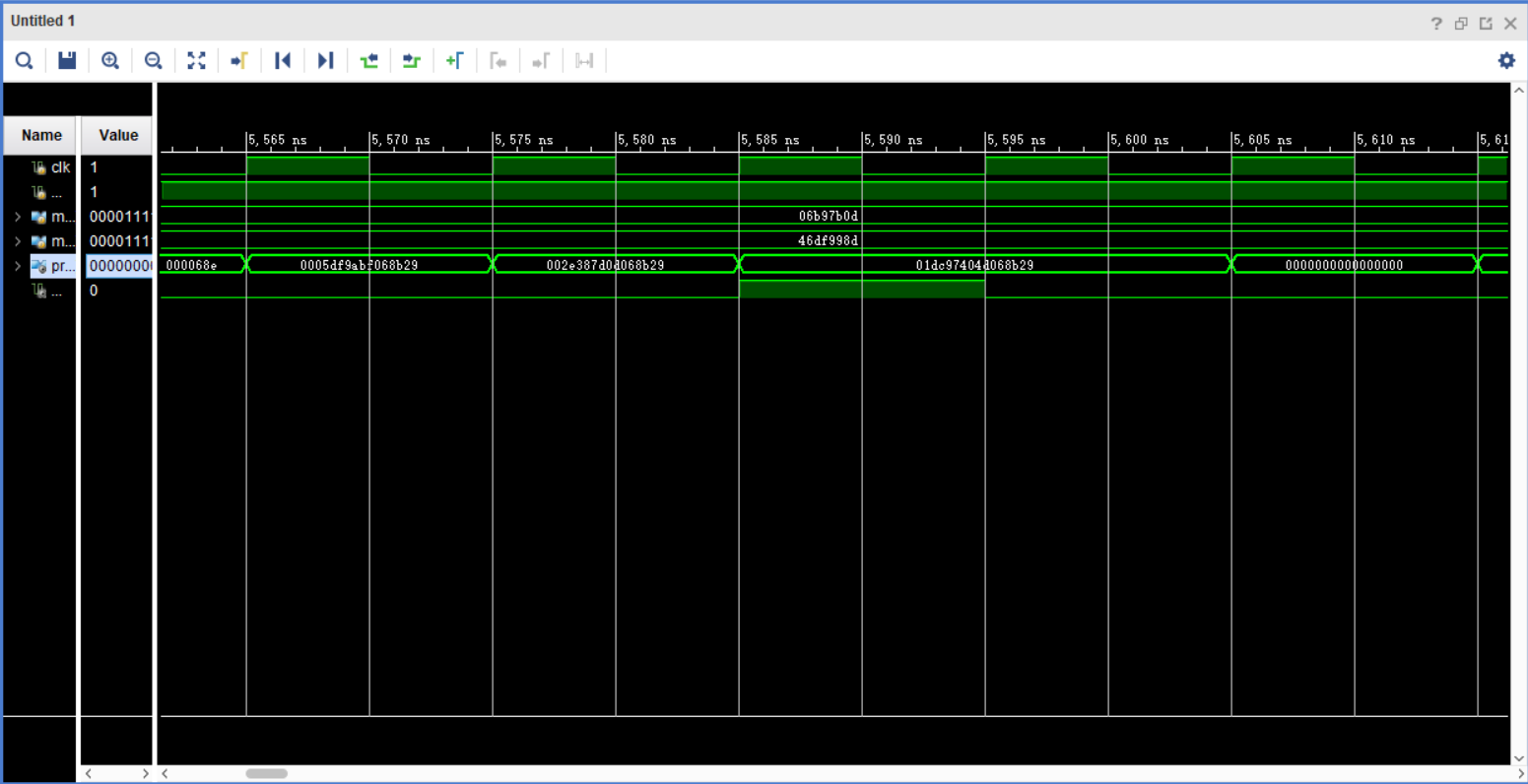
iii.随机生成数据分析



以此例代表正数与负数相乘的情况。

0x12153524与0xc0895e81的乘积为0xfb8466bd5676ff24，正确。

说明：0xc0895e81的符号位为1，是负数，补码为0x3f76a17f，补码相乘结果为0x47b9942a98900dc，结果是负数，所以取补码为0xfb8466bd5676ff24。



以此例代表正数与正数相乘的情况。

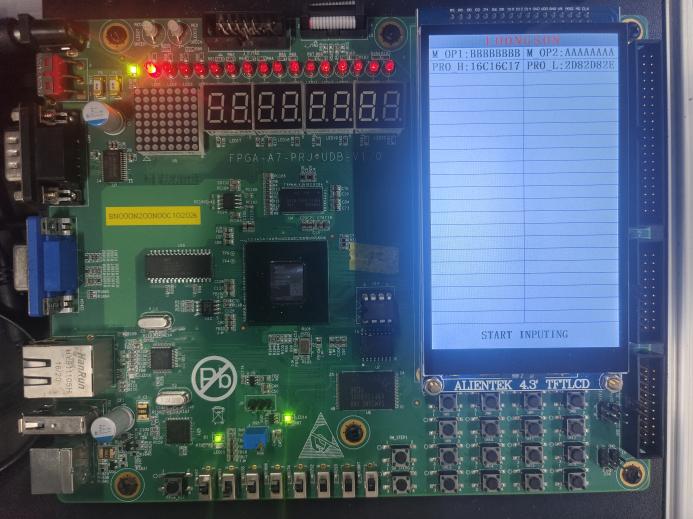
0x6b97b0d与0x46df998d的乘积为0x1dc94704d068b29，正确。

（2）实验箱运行结果

第一个led灯亮，表示乘法运算已经结束，第一个拨码开关切换两个数的输入，两个乘数和乘积的输出如显示屏所示。



以此例代表正数与正数相乘的情况。0x20040426和0x20240411的乘积为0x405019785acde86，正确。



以此例代表负数与负数相乘的情况。0xbbbbbbbb和0xaaaaaaaa的乘积为0x16c16c172d82d82e。

说明：0xbbbbbbbb和0xaaaaaaaa都是负数，补码分别为0x44444445和0x55555556，乘积为0x16c16c172d82d82e，乘积为正数。



以此例代表正数与负数相乘的情况。0x62626262和0xaaaaaaaa的乘积为0xdf3489df13be6914。

说明：0xaaaaaaaa是负数，补码为0x55555556，乘积为0x20cb7620ec4196ec，为负数，补码为0xdf3489df13be6914。

综上，我们可以认为此次改装是正确且高效的。

1. **总结感想**

通过这次实验，我复习了vivado和实验箱的操作流程，对vivado的文件架构更为熟悉。同时，我通过本次对multiplier.v和tb.v的学习和修改，进一步熟练了乘法器的移位流程和补码的计算方法，熟悉了对仿真实验和实验箱实验的理解，将实验课所学与理论课内容串联起来，很有收获。