组成原理课程第二次实报告

实验名称：数据运算--乘法器与乘加器

学号：2310422　姓名：谢小珂　　班次：　1078

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）复现 32 位乘法器

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。

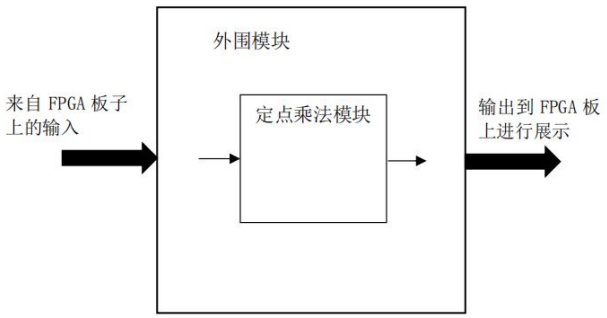


图 3.1 定点乘法设计实验的顶层模块大致框图

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。

（2）改进乘法器

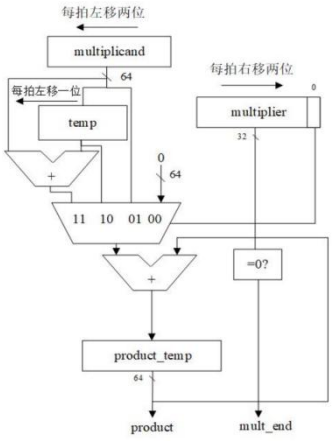
1、将原始的补码一位乘法修改成补码两位乘法，也就是每个时钟周期移两位。

2、将乘法器修改成乘加器，实现 A*B+C 的效果，模块至少有三个 32 位的输入操作数和一个 64 位的输出操作数（由于有加法，可以考虑添加进位输入和输出，也可以不考虑）。

3、仿真文件原始代码中延迟 400、500 的时间单位过长，大家可以修改成 40/50，这样看波形比较直观，注意波形图上应该是 multi_end 为 1 时的输出才是正确输出。

4、上实验箱进行验证时，注意要 lcd 屏上需要输入三个数据，input_sel 不能再用 1 位了，此外 lcd 屏上还需要显示 ABC 三个数。

三、实验原理图



实验原理图

四、实验步骤

（一）设计乘加器

·编写 multiply 文件

把原始的补码一位乘法修改为补码两位乘法,乘数移位逻辑改为每次移两位。

1. 添加加数变量 mult_add

```
input  [31:0] mult_op1,    // 被乘数 A
input  [31:0] mult_op2,   // 乘数 B
input  [31:0] mult_add,  // 加数 C
output [63:0] result,   // 结果 A*B+C
```

2. 修改被乘数和乘数每个时钟周期移动的位数

在每个时钟周期，被乘数通过以下方式更新:保留原被乘数的低 62 位（即去掉最高 2 位）同时在低位补 2 个 0 位，并将这两部分拼接形成新的 64 位被乘数。

```
//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'b00};
       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
```

在每个时钟周期，乘数通过以下方式更新：丢弃最低 2 位，其余位向右移动，同时在最高 2 位补 0，以保持 32 位宽度。

```
//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'b00, 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
```

3. 修改积的计算方式

将生成的积拆分为两个独立的部分积，分别由乘数的最低两位控制：

1>partial_product1→ 由乘数第 0 位（最低位）决定：

位 0=0：取 0

位 0=1：取被乘数原值（×1）

2>partial_product2→ 由 乘数第 1 位（次低位）决定：

位 1=0：取 0

位 1=1：取被乘数左移 1 位（×2）

具体如表格所示：

乘数最低两位	操作含义	partial_product1	partial_product2	实际积
00	+0	0	0	0
01	+A（被乘数）	A	0	A
10	+A*2	0	被乘数左移一位	A*2
11	+A*3	A	被乘数左移一位	A+A*2=A*3

代码如下：

```
// 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;
```

4. 将拆分后的两个部分积同时累加到乘法结果中，确保每次迭代正确更新累加值。

```
//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; // 乘法开始，乘积清零
end
end
```

5. 将结果汇总得到最终结果即 A*B+C

```
wire [63:0] add_result;
assign add_result = product + {32'd0, mult_add};// 加上 C
```

・仿真验证--编写 testbench（）文件

主要修改为添加了加数 C 即变量 mult_add，故不再赘述。

・实验箱验证--修改 multiply_display 文件（仅展示修改的代码部分）

```
module multiply_display(

    // 拨码开关，用于选择输入数
    input input_sel_1, // 0: 输入为被乘数 A; 1: 输入为乘数 B
    input input_sel_2,  // 0: 输入为加数
    input sw_begin,
);
    //-----{调用乘法器模块}begin
    wire        mult_begin;
    reg  [31:0] mult_op1;
    reg  [31:0] mult_op2;
    reg  [31:0] mult_add;
    wire [63:0] result;
    wire        mult_end;
    assign mult_begin = sw_begin;
    assign led_end = mult_end;

    multiply multiply_module (
        .clk       (clk       ),
        .mult_begin(mult_begin),
        .mult_op1  (mult_op1  ),
        .mult_op2  (mult_op2  ),
        .mult_add  (mult_add ),
        .result    (result  ),
        .mult_end  (mult_end  )
    );

//-----{从触摸屏获取输入}begin
```

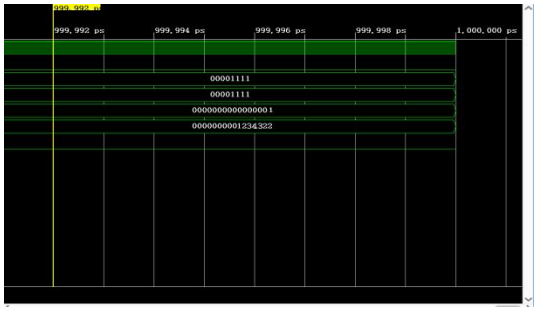
```
// 根据实际需要输入的数修改此小节
    // 当 input_sel_1 为 0 时，表示输入数为被乘数 A
    always @(posedge clk)
    begin
        if (!resetn)
        begin
            mult_op1 <= 32'd0;
        end
        else if (input_valid && (input_sel_1==2'b00))
        begin
            mult_op1 <= input_value;
        end
    end

    // 当 input_sel_1 为 1 时，表示输入数为乘数 B
    always @(posedge clk)
    begin
        if (!resetn)
        begin
            mult_op2 <= 32'd0;
        end
        else if (input_valid && (input_sel_1==2'b01))
        begin
            mult_op2 <= input_value;
        end
    end

      // 当 input_sel_2 为 0 时，表示输入数为加数 C
    always @(posedge clk)
    begin
        if (!resetn)
        begin
            mult_add<= 32'd0;
        end
        else if (input_valid && (input_sel_2 == 2'b00))
        begin
            mult_add <= input_value;
        end
    end
//-----{从触摸屏获取输入}end

//-----{输出到触摸屏显示}begin
    always @(posedge clk)
    begin
```

```
        case(display_number)
            6'd1 : // 显示被乘数 A
            begin
                display_valid <= 1'b1;
                display_name  <= "M_OP1";
                display_value <= mult_op1;
            end
            6'd2 : // 显示乘数 B
            begin
                display_valid <= 1'b1;
                display_name  <= "M_OP2";
                display_value <= mult_op2;
            end
            6'd3://显示加数 C
            begin
                display_valid <= 1'b1;
                display_name  <= "M_ADD";
                display_value <= myaddr;
            end
             6'd4 :
            begin
                display_valid <= 1'b1;
                display_name  <= "RES_H";
                display_value <= result_r[63:32];
            end
            6'd5 :
            begin
                display_valid <= 1'b1;
                display_name  <= "RES_L";
                display_value <= result_r[31: 0];
            end
            default :
            begin
                display_valid <= 1'b0;
                display_name  <= 48'd0;
                display_value <= 32'd0;
            end
        endcase
    end
//-----{输出到触摸屏显示}end
endmodule
```

- 添加 lcd_moudle 文件和 multiply.xdc 约束文件。

五、实验结果分析

• 仿真结果 simulation 波形图



A=00001111 B=00001111 C=00000001 result=0000000001234322
由波形知，结果正确。

• 实验箱结果

左图中，A=00000000 B=00123456 C=00000789 RES_H=00000000 RES_L=00000789
右图中，A=00001111 B=00001111 C=00001111 RES_H=00000000 RES_L=01235432 经验证，结果均正确。

六、总结感想

1.通过本次实验，我进一步熟悉了 Verilog 语言的使用，掌握了模块化设计思想，在实现补码两位乘法器和乘加功能时，对数据移位、符号扩展和累加器的设计有了更深刻的理解。

2. 本次实验的核心是理解并实现迭代乘法算法，通过拆分部分积（partial_product1 和 partial_product2），我认识到如何通过硬件并行优化计算效率。

3.相比第一次实验，本次实验的难度显著提升，尤其是从补码一位乘法扩展到两位乘法，以及新增的乘加功能。初期对部分积累加和符号扩展的逻辑不够清晰，导致仿真结果出现偏差。通过逐步调试波形图和对比理论值，最终定位到问题（部分积漏算），这锻炼了我的调试能力和耐心。