

乘法器实验

实验介绍

本实验将实现无符号乘法器和带符号乘法器

实验目标

- 了解 32 位带符号、无符号乘法器的实现原理
- 使用 verilog 实现 32 位无符号乘法器和带符号乘法器

实验原理

接口定义，可以直接复制到文件中

（请务必按照接口定义编写代码，在将来的实验中也是如此，模块名也请按照给出的定义命名）

- 无符号乘法器功能为：将两个 32 位无符号数相乘，得到一个 64 位无符号数

```
module MULTU(  
    input    clk,           // 乘法器时钟信号  
    input    reset,  
    input [31:0] a,         // 输入 a(被乘数)  
    input [31:0] b,         // 输入 b(乘数)  
    output [63:0] z         // 乘积输出 z  
);
```

2. 带符号乘法器功能为：将两个 32 带无符号数相乘，得到一个 64 位带符号数

```
module MUL(  
  
    input  clk,           // 乘法器时钟信号  
  
    input  reset,  
  
    input [31:0]  a,       // 输入 a(被乘数)  
  
    input [31:0]  b,       // 输入 b(乘数)  
  
    output [63:0] z        // 乘积输出 z  
  
);
```

无符号乘法器功能为：将两个 32 位无符号数相乘，得到一个 64 位无符号数。将低 32 位存放在专用寄存器 L0 中，高 32 位存放在寄存器 HI 中。带符号乘法器功能为：将两个 32 带无符号数相乘，得到一个 64 位带符号数。只选取低 32 位存入指定寄存器。执行乘法指令过程中不产生异常。本实验不允许使用行为级实现。

以下提供几种实验实现思路，仅供参考：

1. 两个二进制数 a 和 b 相乘，可以认为是 a 和 b 的每一位相乘后移位后的结果相加。关于 a 与 b 的每一位相乘产生的中间结果，如果 b 那位是 0，那么中间结果就是 0；如果是 1，那么中间结果就是在 a 前后补上相应位数的零通过字符拼接的方式表示。然后将这些中间乘积相加就是最后的结果。

2. 二进制的乘法可以用加法和移位操作完成，可以使用循环迭代的方法实现。每次循环时，判断 b 的值是否为 1，然后决定是否将中间值加上 a。每次循环，a 左移一位，b 右移一位。循环结束，最后的中间值就是最后的乘积。

3. 可以从 Wallace Tree 乘法算法的角度出发实现。有兴趣实现的同学可以自行查阅此算法实现。

下面以思路 1 举例 8 位无符号数乘 8 位无符号数的一种实现方式：

```
module MULTU(  
  
    input clk, // 乘法器时钟信号  
  
    input reset,  
  
    input [7:0] a, // 输入 a(被乘数)
```

```

input [7:0] b, // 输入 b(乘数)

output [15:0] z // 乘积输出 z

);

    // 申请寄存器

    reg [15:0] temp;

    reg [15:0] stored0;

    reg [15:0] stored1;

    reg [15:0] stored2;

    reg [15:0] stored3;

    reg [15:0] stored4;

    reg [15:0] stored5;

    reg [15:0] stored6;

    reg [15:0] stored7;

    reg [15:0] add0_1;

    reg [15:0] add2_3;

    reg [15:0] add4_5;

    reg [15:0] add6_7;

    reg [15:0] add0t1_2t3;

    reg [15:0] add4t5_6t7;

    reg [15:0] add0t3_4t7;


    always @(posedge clk or negedge reset)

    begin

        // reset 置零

        if(reset) begin

```

```
temp <= 0;

stored0 <= 0;

stored1 <= 0;

stored2 <= 0;

stored3 <= 0;

stored4 <= 0;

stored5 <= 0;

stored6 <= 0;

stored7 <= 0;

add0_1 <= 0;

add2_3 <= 0;

add4_5 <= 0;

add6_7 <= 0;

add0t1_2t3 <= 0;

add4t5_6t7 <= 0;
```

```
end
```

```
else begin
```

```
    //通过字符拼接方式表示出中间相乘值，并相加
```

```
stored0 <= b[0]? {8'b0, a} : 16'b0;

stored1 <= b[1]? {7'b0, a, 1'b0} : 16'b0;

stored2 <= b[2]? {6'b0, a, 2'b0} : 16'b0;

stored3 <= b[3]? {5'b0, a, 3'b0} : 16'b0;

stored4 <= b[4]? {4'b0, a, 4'b0} : 16'b0;

stored5 <= b[5]? {3'b0, a, 5'b0} : 16'b0;

stored6 <= b[6]? {2'b0, a, 6'b0} : 16'b0;
```

```

        stored7 <= b[7]? {1'b0, a, 7'b0} :16'b0;

        add0_1 <= stored1 + stored0;

        add2_3 <= stored2 + stored3;

        add4_5 <= stored4 + stored5;

        add6_7 <= stored6 + stored7;

        add0t1_2t3 <= add0_1 + add2_3;

        add4t5_6t7 <= add4_5 + add6_7;

        temp <= add0t1_2t3 + add4t5_6t7;

    end

end

assign z = temp;

endmodule

```

对于写完的模块，采取了以下数据进行了测试，大家可以参考对自己写完的模块进行测试：

```

a = 0, b = 0; a = 0, b = 8'b11111111;

a = 8'b10110011, b = 0; a = 8'b11111111, b = 8'b11111111;

a = 8'b10000000, b = 8'b10101010; a = 8'b10101010, b = 8'b10000000;

a = 8'b101101 ; b= 8'b1101000; a = 8'b1000111, b = 8'b1110

```

(注意：在写 32 位乘法器的时候用 32 位的数据进行测试)

至于带符号数乘法器的实现只需要进行简单的变动，请大家自己思考实现。

本次实验不允许使用行为级(乘号)方式实现。

实验步骤

1. 新建 Vivado 工程
2. 编写各个模块
3. 用 modelsim 仿真测试各模块