

实验三 32 位除法器

1. 实验介绍

通过本次试验，了解除法器的实现原理，并学习如何实现一个除法器，本实验将实现 32 位无符号除法器 and 32 位有符号除法器

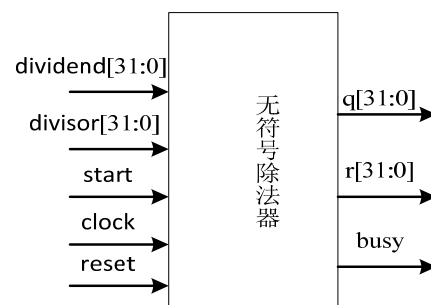
2. 实验目标

- 了解 32 位有符号、无符号除法器的实现原理
- 使用 Verilog 实现一个 32 位有符号除法器和一个 32 位无符号除法器

3. 实验原理

1) 无符号除法器

无符号除法器功能为将两个 32 位无符号数相除，得到一个 32 位商和 32 位余数。本实验分别实现 32 位有符号和无符号除法器，结果为 32 位商 `quotient` 和 32 位余数 `remainder`，分别存放在 CPU 的专用寄存器 LO 和 HI 中。除法器时钟信号下降沿时检查 `start` 信号，有效时开始执行，执行除法指令时，`busy` 标志位置 1。在执行除法指令时，任何情况下不产生算术异常，当除数为 0 时，运算结果未知，对除法器除数为 0 和溢出情况的发生通过汇编指令中其他指令进行检查和处理。

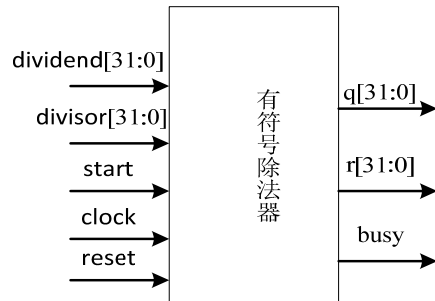


- 接口定义:

```
module DIVU(
    input [31:0]dividend,    //被除数
    input [31:0]divisor,     //除数
    input start,             //启动除法运算
    input clock,
    input reset,             //高电平有效
    output [31:0]q,          //商
    output [31:0]r,          //余数
    output busy              //除法器忙标志位
);
```

2) 有符号除法器

带符号除法器功能为：将两个 32 位有符号数相除，得到一个 32 位商和余数，基本和无符号除法器类似，注意余数符号与被除数符号相同。



● 接口定义：

```
module DIV(
    input [31:0]dividend,      //被除数
    input [31:0]divisor,       //除数
    input start,               //启动除法运算
    input clock,
    input reset,               //高电平有效
    output [31:0]q,            //商
    output [31:0]r,            //余数
    output busy                //除法器忙标志位
);
```

3) 参考思路

● 基于移位、减法的恢复余数除法器：

对于 32 位无符号除法，可将被除数 a 转换成高 32 位为 0 低 32 位是 a 的数 $temp_a$ ，在每个周期开始时 $temp_a$ 向左移动一位，最后一位补零，然后判断 $temp_a$ 的高 32 位是否大于等于除数 b ，如是则 $temp_a$ 的高 32 位减去 b 并且加 1，得到的值赋给 $temp_a$ ，如果不是则直接进入下一步，执行结束后 $temp_a$ 的高 32 位即为余数，低 32 位即为商。对于 32 位有符号除法，可先将有符号数转换成无符号数除法，根据被除数和除数的符号判断商的符号，被除数是负数时余数为负，否则为正。

● 不恢复余数除法器：

不恢复余数即不管相减结果是正还是负，都把它写入 reg_r ，若为负，下次迭代不是从中减去除数而是加上除数。

以下是无符号不恢复余数除法器的参考代码：

```
module DIVU(input [7:0]dividend,      //dividend
            input [7:0]divisor,       //divisor
```

```

        input start,          //start = is_div & ~busy
        input clock,
        input reset,
        output [7:0]q,
        output [7:0]r,
        output reg busy
    );
    wire ready;
    reg[2:0]count;
    reg [7:0] reg_q;
    reg [7:0] reg_r;
    reg [7:0] reg_b;
    reg busy2,r_sign;
    assign ready = ~busy & busy2;
    wire [8:0] sub_add = r_sign?({reg_r,q[7]} + {1'b0,reg_b}):({reg_r,q[7]} - {1'b0,reg_b}); //加、
    减法器
    assign r = r_sign? reg_r + reg_b : reg_r;
    assign q = reg_q;
    always @ (posedge clock or posedge reset)begin
        if (reset == 1) begin                                //重置
            count <=3'b0;
            busy <= 0;
            busy2 <= 0;
        end else begin
            busy2 <= busy;
            if (start) begin                                  //开始除法运算，初始化
                reg_r <= 8'b0;
                r_sign <= 0;
                reg_q <= dividend;
                reg_b <= divisor;
                count <= 3'b0;
                busy <= 1'b1;
            end else if (busy) begin                          //循环操作
                reg_r <= sub_add[7:0];                        //部分余数
                r_sign <= sub_add[8];                         //如果为负，下次相加
                reg_q <= {reg_q[6:0],~sub_add[8]};
                count <= count +3'b1;                         //计数器加一
                if (count == 3'h7) busy <= 0;                 //结束除法运算
            end
        end
    end
end
endmodule

```

三个寄存器 reg_q 初始化为被除数，结果为商；reg_b 初始化为除数；reg_r 初始化

为零，结果为余数，做减法时，减数是 `reg_b` 中的内容（除数），被减数是 `reg_r` 的内容（余数）左移一位，最低位由 `reg_q`（被除数）的最高位补充。为了判断相减结果的正负，加、减法器的位数要比除数的位数多一位。如果相减结果为正（减法器输出的最高位是0），商 1，把相减结果写入 `reg_r`，`reg_q` 的内容左移一位，最低位放入商 1，如果相减结果为负，商 0，把被减数写入 `reg_r`，`reg_q` 内容左移一位，最低位放入商 0，循环直到被除数全被移出 `reg_q` 为止。测试数据包括：

被除数 `0x00000000`、`0xffffffff`、`0xaaaaaaaa`、`0x55555555`、`0x7ffffff`

除数 `0xffffffff`、`0xaaaaaaaa`、`0x55555555`、`0x7ffffff`

4. 实验步骤

1. 新建工程
2. 编写有符号除法器模块
3. 编写无符号除法器模块
4. 用 ModelSim 仿真测试各模块