实验报告二：寄存器文件

姓名：姜庆彩

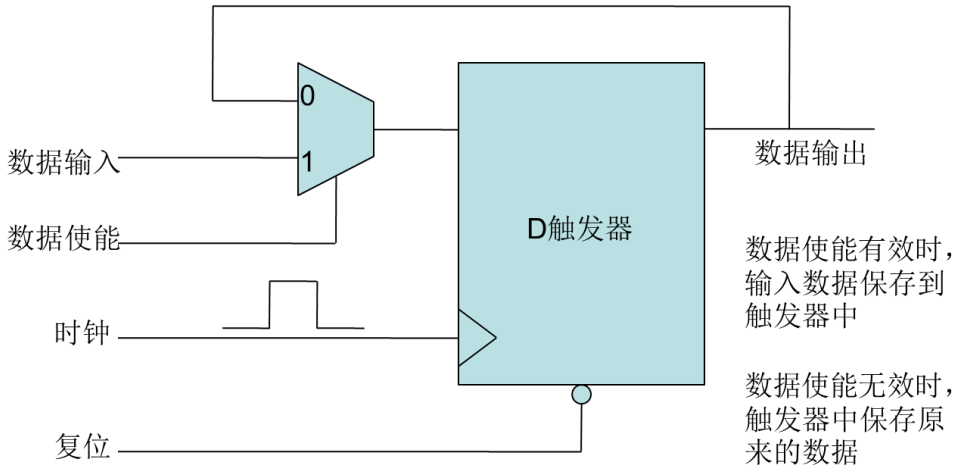
学院：计算机学院

学号：PB15051087

**实验原理：**

# 原理

- 寄存器如何存储数据？




**实验要求（寄存器文件）：**

- 设计一 32*32bit 的寄存器文件，即 32 个 32 位的寄存器文件（寄存器组）

具备两组读端口及一组写端口
通过读端口可从 0~31 号的任意地址读取数据
通过写端口可向 0~31 号的任意地址写入数据
寄存器的复位值自行制定

**实验要求（功能）：**

- 调用实验一 ALU，完成以下功能

寄存器文件组 r0,r1 初始化为 1，1，其他所有寄存器初始化为 0
在 clk 控制下，依次完成以下计算，注意每个 clk 至多允许完成一次计算
r0+r1->r2
**r1+r2->r3**
r2+r3->r4
……

- 结果在仿真中显示

**我的实现：**

- 本次实验的分成 Top 与 ALU，REG_FILE，control 四个模块完成。

- ALU 模块直接使用了上次实验实现的 ALU 模块。

- REG_FILE 模块主要实现的是寄存器的读写操作，初始时 rst_n 低电平，寄存器前两位数置为 1，其余位置置为 0，其中置数的部分使用了 for 循环，这个是网上搜索现学现用的；当 r3_wr 高电平时寄存器可以写入数据；r1_dout 与 r2_dout 则一直是相应位置寄存器存的数据。

- Control 模块实现的是控制 r1~r3 的地址与 r3 的写允许的值。状态机有两个状态，分别为暂停状态与运行状态。当 state 处于运行状态并且 r3_addr 的值小于 31 时，每个端口的地址加一。

源代码：

```
module alu(
  input  signed      [31:0] alu_a,
  input  signed      [31:0] alu_b,
  input               [4:0] alu_op,
  output      reg    [31:0] alu_out
    );
     always@(*)
     begin
     case(alu_op)
        1:  alu_out <= alu_a + alu_b;
        2:  alu_out <= alu_a - alu_b;
        3:  alu_out <= alu_a & alu_b;
        4:     alu_out <= alu_a | alu_b;
        5:     alu_out <= alu_a ^ alu_b;
        6:     alu_out <= ~(alu_a | alu_b);
        default:alu_out <= alu_out;
        endcase
        end

endmodule
```

```
module REG_FILE(
input    clk,
input    rst_n,
input  [4:0]  r1_addr,
input  [4:0]  r2_addr,
input  [4:0]  r3_addr,
input  [31:0] r3_din,
input   r3_wr,
output reg [31:0] r1_dout,
output reg [31:0]   r2_dout
    );
reg [31:0] register[0:31];
integer k,i;
always@(posedge clk or negedge rst_n)
begin
 if(~rst_n)
 begin
   register[0]=32'b1;
    register[1]=32'b1;
   for(k=2;k<32;k=k+1)
      register[k]=32'b0;
 end
 else if(r3_wr)
 begin
   i=r3_addr;
    register[i]=r3_din;
    i=r1_addr;
    r1_dout=register[i];
    i=r2_addr;
    r2_dout=register[i];
 end
 else
 begin
   i=r1_addr;
    r1_dout=register[i];
    i=r2_addr;
    r2_dout=register[i];
 end
end



endmodule
```

```
module control(
input clk,
input rst_n,
output reg [4:0] r1_addr,
output reg [4:0] r2_addr,
output reg [4:0] r3_addr,
output reg r3_wr
    );
reg state;
always@(posedge clk or negedge rst_n)
begin
   if(~rst_n)
    begin
      state=1'b0;
      r1_addr=5'b00;
      r2_addr=5'b01;
      r3_addr=5'b10;
      r3_wr=0;
    end
    else
    begin
      if(state==1'b0)
      begin
      r3_wr=1;
      state=1'b1;
      end
      else if(state==1'b1 && r3_addr<5'b11111)
      begin
      r3_wr=0;
      r3_addr=r3_addr+5'b1;
      r2_addr=r2_addr+5'b1;
      r1_addr=r1_addr+5'b1;
      state=1'b0;
      end
    end
end
endmodule
```

```
module top(
input clk,
input rst_n,
output [31:0] result
   );
   assign result=r3_din;
  wire [4:0] alu_op;
   wire [31:0] r1_dout;
   wire [31:0] r2_dout;
   wire [31:0] r3_din;
   wire [4:0] r1_addr;
   wire [4:0] r2_addr;
   wire [4:0] r3_addr;
   wire r3_wr;
   assign alu_op=5'h01;
alu uut
(
.alu_a(r1_dout),
.alu_b(r2_dout),
.alu_op(alu_op),
.alu_out(r3_din)
);
REG_FILE utt(
.clk(clk),
.rst_n(rst_n),
.r1_addr(r1_addr),
.r2_addr(r2_addr),
.r3_addr(r3_addr),
.r3_din(r3_din),
.r3_wr(r3_wr),
.r1_dout(r1_dout),
.r2_dout(r2_dout)
   );
control uuu(
.clk(clk),
.rst_n(rst_n),
.r1_addr(r1_addr),
.r2_addr(r2_addr),
.r3_addr(r3_addr),
.r3_wr(r3_wr)
);
Endmodule
```

```
module test;

// Inputs
    reg clk;
    reg rst_n;

    // Outputs
    wire [31:0] result;

    // Instantiate the Unit Under Test (UUT)
    top uut (
        .clk(clk),
        .rst_n(rst_n),
        .result(result)
    );

    initial begin
        // Initialize Inputs
        clk =1;
        rst_n = 0;

        // Wait 100 ns for global reset to finish
        #1;
      rst_n= 1;
        // Add stimulus here

    end
   always
    begin
       clk=~clk;
        #1;
    end
endmodule
```