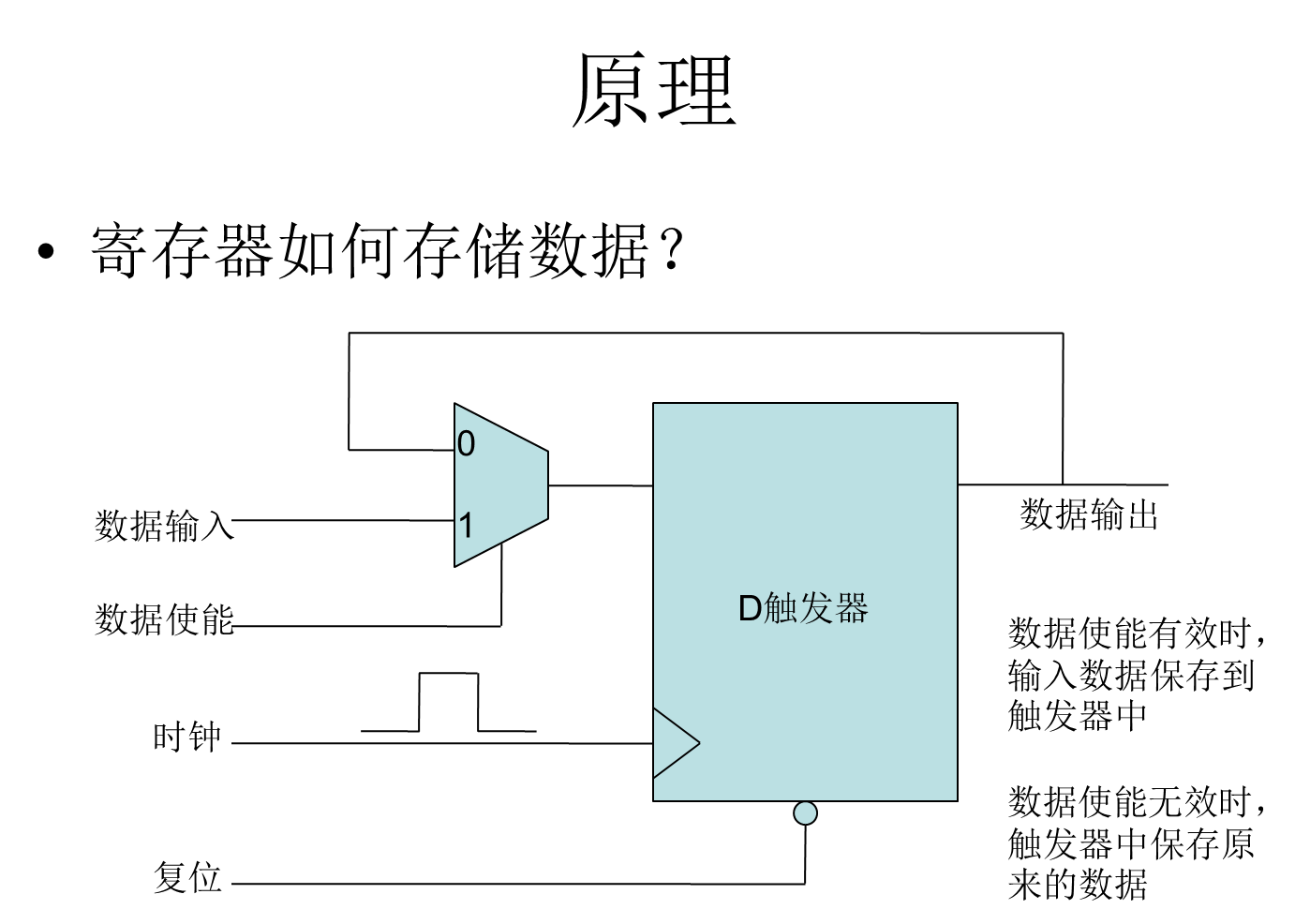
实验报告二：寄存器文件

姓名：姜庆彩

学院：计算机学院

学号：PB15051087

**实验原理：**

****

**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）
  1. 具备两组读端口及一组写端口
  2. 通过读端口可从0~31号的任意地址读取数据
  3. 通过写端口可向0~31号的任意地址写入数据
  4. 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
  1. 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
  2. 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
     1. r0+r1->r2
     2. **r1+r2->r3**
     3. r2+r3->r4
     4. ……
* 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