

## Lab2\_Regfile

金泽文 PB15111604

### 实验目的：

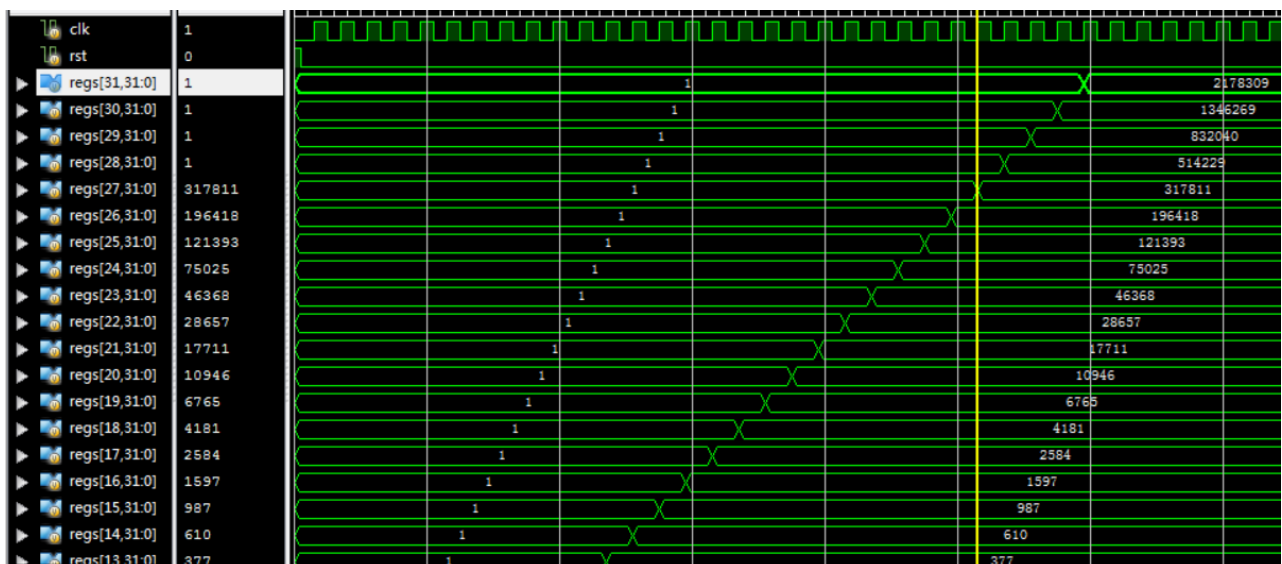
理解寄存器文件的原理

### 实验内容：

- 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

### 实验结果：

按要求完成。仿真结果如图：



### 实验要求：

设计一个顶层模块 Top

在 Top 中调用 ALU 完成加法运算

在 Top 中调用 REG\_FILE 完成数据存取

在 Top 中实现一控制逻辑 Control，完成在 CLK 控制下的斐波拉契加法运算

Control 也可以实现为一个模块，然后在 Top 中被调用。

#### **实验分析：**

ALU 模块完全同于上次实验。

Regfile 读取用一个周期完成。

Top 模块控制输入输出地址。


#### **意见建议：**

无

附录：

ALU 部分：

```
1 module ALU(  
2   input signed [31:0] alu_a,  
3   input signed [31:0] alu_b,  
4   input [4:0] alu_op,  
5   output reg [31:0] alu_out  
6   );  
7  
8   parameter A_NOP = 5'h00;  
9   parameter A_ADD = 5'h01;  
10  parameter A_SUB = 5'h02;  
11  parameter A_AND = 5'h03;  
12  parameter A_OR = 5'h04;  
13  parameter A_XOR = 5'h05;  
14  parameter A_NOR = 5'h06;  
15  
16  always@(*)  
17  begin  
18      case(alu_op)  
19          A_NOP: alu_out = 0;  
20          A_ADD: alu_out = alu_a + alu_b;  
21          A_SUB: alu_out = alu_a - alu_b;  
22          A_AND: alu_out = alu_a & alu_b;  
23          A_OR: alu_out = alu_a | alu_b;  
24          A_XOR: alu_out = (~alu_a & alu_b) | (alu_a & ~alu_b);  
25          A_NOR: alu_out = ~(alu_a | alu_b);  
26      endcase  
27  end  
28  
29 endmodule
```



Top 部分：

55%

0.8K

16.9K

```
1 |
2 module top(
3     input        clk,
4     input        rst
5 );
6
7 wire [31:0] r1_dout, r2_dout, r3_din;
8 wire [4:0]  r1_addr, r2_addr, r3_addr;
9 reg [4:0]   r1_addr_reg, r2_addr_reg, r3_addr_reg;
10
11 parameter alu_op = 5'h01;
12 assign r3_wr = clk;
13 assign r1_addr = r1_addr_reg;
14 assign r2_addr = r2_addr_reg;
15 assign r3_addr = r3_addr_reg;
16
17 always @(posedge clk or posedge rst) begin
18     if (rst) begin
19         r1_addr_reg <= 0;
20         r2_addr_reg <= 1;
21         r3_addr_reg <= 2;
22     end
23     else begin
24         r1_addr_reg <= r1_addr_reg + 1;
25         r2_addr_reg <= r2_addr_reg + 1;
26         r3_addr_reg <= r3_addr_reg + 1;
27     end
28 end
29
30 ALU ALU1(r1_dout, r2_dout, alu_op, r3_din);
31 REG_FILE RF(clk, rst, r1_addr, r2_addr, r3_addr, r3_din, r3_wr, r1_dout,
32             r2_dout);
33 endmodule
```

Regfile 部分：

```
1 module REG_FILE(  
2   input      clk,  
3   input      rst,  
4   input [4:0] r1_addr,  
5   input [4:0] r2_addr,  
6   input [4:0] r3_addr,  
7   input [31:0] r3_din,  
8   input      r3_wr,  
9   output [31:0] r1_dout,  
10  output [31:0] r2_dout  
11 );  
12  
13 reg [31:0] regs[31:0];  
14  
15 always @(posedge clk or posedge rst) begin  
16     if (rst) begin  
17         regs[0] <= 32'b1;  
18         regs[1] <= 32'b1;  
19         regs[2] <= 32'b1;  
20         regs[3] <= 32'b1;  
21         regs[4] <= 32'b1;  
22         regs[5] <= 32'b1;  
23         regs[6] <= 32'b1;  
24         regs[7] <= 32'b1;  
25         regs[8] <= 32'b1;  
26         regs[9] <= 32'b1;  
27         regs[10] <= 32'b1;  
28         regs[11] <= 32'b1;  
29         regs[12] <= 32'b1;  
30         regs[13] <= 32'b1;  
31         regs[14] <= 32'b1;  
32         regs[15] <= 32'b1;  
33         regs[16] <= 32'b1;  
34         regs[17] <= 32'b1;  
35         regs[18] <= 32'b1;  
36         regs[19] <= 32'b1;  
37         regs[20] <= 32'b1;  
38         regs[21] <= 32'b1;  
39         regs[22] <= 32'b1;  
40         regs[23] <= 32'b1;  
41         regs[24] <= 32'b1;  
42         regs[25] <= 32'b1;  
43         regs[26] <= 32'b1;  
44         regs[27] <= 32'b1;  
45         regs[28] <= 32'b1;  
46         regs[29] <= 32'b1;  
47         regs[30] <= 32'b1;  
48         regs[31] <= 32'b1;  
49     end  
50     else if (r3_wr) begin  
51         regs[r3_addr] <= r3_din;  
52     end  
53 end  
54  
55 assign r1_dout = regs[r1_addr];  
56 assign r2_dout = regs[r2_addr];  
57 endmodule  
58
```