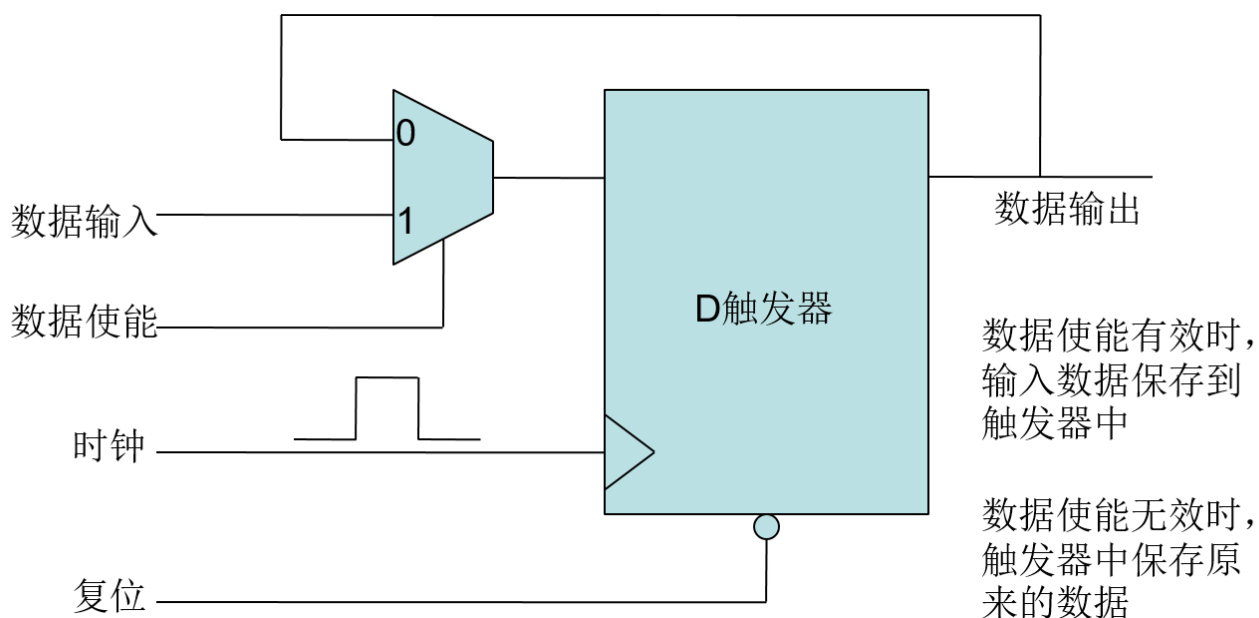


Lab02 寄存器文件

PB15000102 王嵩超

实验原理

寄存器如何存储数据：



实验目的与要求

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

寄存器接口

我使用了供参考的接口设计：

```

1 module REG_FILE(
2     input          clk,
3     input          rst_n,
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 );

```

功能要求

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

为了与计算机内CPU的构造类似，所有的寄存器输入值都应由ALU计算而获得。如置零可用类似xor \$ax, \$ax的方法完成。

实验设计

设计一个顶层模块Top

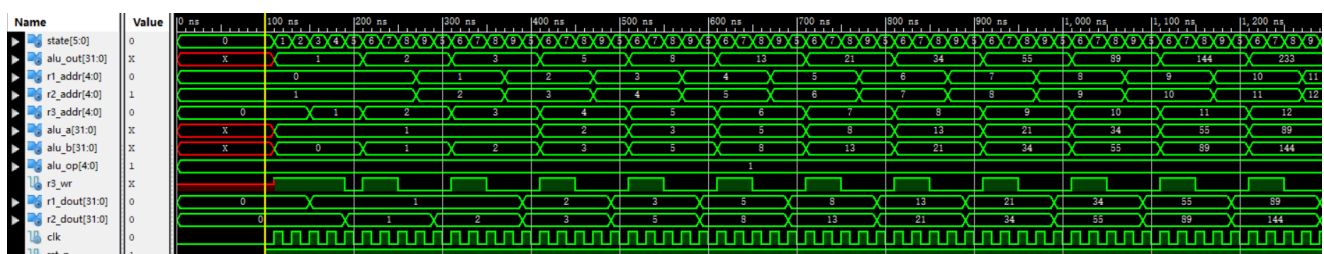
- 在Top中调用ALU完成加法运算
- 在Top中调用REG_FILE完成数据存取
- 在Top中实现一状态机，完成在CLK控制下的斐波拉契加法运算

在CPU的设计中，可以预见的是：因为指令格式已经预制好，ALU的输入端口之一直接连往寄存器的输出端口，另一输入端口则通过选择器，要么连往寄存器输出端口，要么连往立即数端口。而在本实验中并不需要处理指令，所以ALU的输入端口仍由我们自己赋值。

设计仿真模块Simu

- 产生复位信号，使各寄存器初始化
- 周期性地产生clk信号

仿真结果



其中最上一行state是状态机的状态寄存器。

可看出运算一次由5个clock完成。

rx_addr均为寄存器文件的地址寄存器，每次计算完后都向后移一位。

alu_out显示了每次计算后的结果。

总结

本次实验完成较快，但也花了不少时间用来调试。主要的bug在于ALU计算进行得过早。在前一周期 t_0 ，寄存器堆的两个读地址自增后，需要等到下一个时钟(t_0+1)后，才能使r1_dout, r2_dout的值被改变，如果在(t_0+1)对ALU操作数赋值进行计算，两操作数则仍是移位以前的值。故需要等一个周期到(t_0+2)时再用ALU计算和。

如图所示：

```
8://move
begin
    r1_addr <= r1_addr + 1;
    r2_addr <= r2_addr + 1;
    state <= 9;
end
9://waiting This is important, because r1_out and r2_out take time to output.
begin
    state <= 5;
end
```

源代码

ALU模块：

```

1  `timescale 1ns / 1ps
2
3  module alu(
4      input signed [31:0] alu_a,
5      input signed [31:0] alu_b,
6      input [4:0] alu_op,
7      output reg [31:0] alu_out
8  );
9  parameter A_NOP = 5'h00; //空运算
10 parameter A_ADD = 5'h01; //符号加
11 parameter A_SUB = 5'h02; //符号减
12 parameter A_AND = 5'h03; //与
13 parameter A_OR = 5'h04; //或
14 parameter A_XOR = 5'h05; //异或
15 parameter A_NOR = 5'h06; //或非
16
17 always@(*)
18 begin
19     case(alu_op)
20         A_NOP:
21             begin
22
23             end
24         A_ADD:
25             begin
26                 alu_out <= alu_a + alu_b;
27             end
28         A_SUB:
29             begin
30                 alu_out <= alu_a - alu_b;
31             end
32         A_AND:
33             begin
34                 alu_out <= alu_a & alu_b;
35             end
36         A_OR:
37             begin
38                 alu_out <= alu_a | alu_b;
39             end
40         A_XOR:
41             begin
42                 alu_out <= alu_a ^ alu_b;
43             end
44         A_NOR:
45             begin
46                 alu_out <= ~(alu_a | alu_b);
47             end
48     endcase
49 end
50 endmodule

```

寄存器模块:

```

1 //r1 r2 read; r3 write
2 module REG_FILE(
3     input                clk,
4     input                rst_n,
5     input [4:0]          r1_addr,
6     input [4:0]          r2_addr,
7     input [4:0]          r3_addr,
8     input [31:0]         r3_din,
9     input                r3_wr,
10    output reg [31:0]     r1_dout,
11    output reg [31:0]     r2_dout
12 );
13
14 reg [31:0] R[31:0];
15 integer i;
16 always@(posedge clk, negedge rst_n)
17 begin
18     if(~rst_n)
19     begin
20         for ( i=0; i<32; i=i+1 ) R[i] <= 0;
21         r1_dout <=0;
22         r2_dout <=0;
23     end
24     else
25     begin
26         r1_dout = R[r1_addr];
27         r2_dout = R[r2_addr];
28         if(r3_wr) //write enabled
29             R[r3_addr] = r3_din;
30     end
31 end
32 endmodule

```

top控制模块:

```

1  //top.v
2  module top(input clk, input rst_n,
3  output reg [5:0] state,
4  output [31:0] alu_out,
5  output reg [4:0]          r1_addr,
6  output reg [4:0]          r2_addr,
7  output reg [4:0]          r3_addr,
8  output reg [31:0] alu_a,
9  output reg [31:0] alu_b,
10 output reg [4:0] alu_op,
11 output reg    r3_wr,
12 output [31:0] r1_dout,
13 output [31:0] r2_dout);
14
15
16
17 //reg [31:0]          r3_din; //substituted by alu_out.
18 //registers can only be set values via ALU.
19
20
21
22 ALU myALU (
23     .alu_a(alu_a),
24     .alu_b(alu_b),
25     .alu_op(alu_op),
26     .alu_out(alu_out)
27 );
28 REG_FILE myreg(
29     .clk(clk),
30     .rst_n(rst_n),
31     .r1_addr(r1_addr),
32     .r2_addr(r2_addr),
33     .r3_addr(r3_addr),
34     .r3_din(alu_out),
35     .r3_wr(r3_wr),
36     .r1_dout(r1_dout),
37     .r2_dout(r2_dout)
38 );
39 always@(posedge clk, negedge rst_n)
40 begin
41     if(~rst_n)
42     begin
43         state <= 0;
44         r1_addr <= 0;
45         r2_addr <= 1;
46         r3_addr <= 0;
47         alu_op <= 5'h01;
48     end
49 else
50 begin
51     case(state)
52     0://initialize I R[0]=1
53         begin

```

```

54         r3_wr <= 1;
55         alu_a <= 1;
56         alu_b <= 0;
57         state <= 1;
58     end
59     1://waiting
60     begin
61         state <= 2;
62     end
63     2://initialize II R[1]=1
64     begin
65         r3_addr <= 1;
66         state <= 3;
67     end
68     3://waiting
69     begin
70         state <= 4;
71     end
72     4://turn off writing
73     begin
74         r3_wr <= 0;
75         state <= 5;
76     end
77     5://calculate
78     begin
79         r3_addr <= r2_addr + 1; //adjust the writing port
80         r3_wr <= 1;
81         alu_a <= r1_dout; //R[n-1]
82         alu_b <= r2_dout; //R[n-2]
83         state <= 6;
84     end
85     6://writing
86     begin
87         state <= 7;
88     end
89     7://judge
90     begin
91         r3_wr <= 0;
92         if(r3_addr ==31) //reach the end
93             state <= 10;
94         else state <= 8;
95     end
96     8://move
97     begin
98         r1_addr <= r1_addr + 1;
99         r2_addr <= r2_addr + 1;
100        state <= 9;
101    end
102    9://waiting This is important, because r1_out and r2_out take time to
output.
103    begin
104        state <= 5;
105    end

```

```
106         10://finish
107     begin
108         state <= 10;
109     end
110     default:
111     begin
112         state <= 0;
113     end
114 endcase
115 end
116 end
117
118 endmodule
```

仿真模块:


```

1  `timescale 1ns / 1ps
2
3  ///////////////////////////////////////////////////////////////////
4  // Company:
5  // Engineer:
6  //
7  // Create Date:    20:37:14 03/20/2017
8  // Design Name:    top
9  // Module Name:     D:/ISE Project/COD/RegFile/simu.v
10 // Project Name:    RegFile
11 // Target Device:
12 // Tool versions:
13 // Description:
14 //
15 // Verilog Test Fixture created by ISE for module: top
16 //
17 // Dependencies:
18 //
19 // Revision:
20 // Revision 0.01 - File Created
21 // Additional Comments:
22 //
23 ///////////////////////////////////////////////////////////////////
24
25 module simu;
26
27     // Inputs
28     reg clk;
29     reg rst_n;
30
31
32     // Outputs
33     wire [5:0] state;
34     wire [31:0] alu_out;
35     wire [4:0] r1_addr;
36     wire [4:0] r2_addr;
37     wire [4:0] r3_addr;
38     wire [31:0] alu_a;
39     wire [31:0] alu_b;
40     wire [4:0] alu_op;
41     wire r3_wr;
42     wire [31:0] r1_dout;
43     wire [31:0] r2_dout;
44
45
46     // Instantiate the Unit Under Test (UUT)
47     top uut (
48         .clk(clk),
49         .rst_n(rst_n),
50         .state(state),
51         .alu_out(alu_out),
52         .r1_addr(r1_addr),
53         .r2_addr(r2_addr),

```

```
54         .r3_addr(r3_addr),
55         .alu_a(alu_a),
56         .alu_b(alu_b),
57         .alu_op(alu_op),
58         .r3_wr(r3_wr),
59         .r1_dout(r1_dout),
60         .r2_dout(r2_dout)
61     );
62
63     initial begin
64         // Initialize Inputs
65         clk = 0;
66         rst_n = 0;
67
68         // Wait 100 ns for global reset to finish
69         #100;
70         rst_n = 1;
71         // Add stimulus here
72         forever #10 clk = ~clk;
73     end
74
75 endmodule
76
```