

Lab04 时序与状态机

PB15000102 王嵩超

功能要求

综合利用三次实验的结果，完成以下功能：

- 通过例化，向ram中0地址到13地址存入14个数，比如10-23；向ram中100地址到106地址存入7个数，比如0~6，分别代表运算符，向ram 107地址写入-1。
- 运算控制：
 - 从ram 0地址开始的地方取两个数，从ram 100地址开始的地方取一个运算符，计算之后，把结果存入ram地址200。
 - 从ram 2地址开始的地方取两个数，从ram 101地址开始的地方取一个运算符，计算之后，把结果存入ram地址201。
 - 如果取出操作符为-1，则结束。

设计要求

- 实现一个control模块，完成整个运算的控制。
- 实现一个顶层模块top
 - 调用Ram模块
 - 调用RegFile
 - 调用ALU完成加法运算
 - 调用control模块，完成运算控制

减少周期数的具体实现

每个周期要干多件事情，类似流水线的思想。

T0周期：

- 将上次的相加结果存入RAM。
- 把写入寄存器的寄存器号准备好，准备将上个周期T3周期从RAM读入的数据写入reg[0]：
- 将RAM读入的地址改为第二个操作数，准备在下个周期读入。

T1周期：

- 将上次从RAM读入的数据存入reg[0]；
- 开始从RAM读入第二个操作数，同时改寄存器的写地址，准备写到reg[1]，T1周期之后的不长时间便可以从dout得到数据。（然而此时时钟已过，故需等到下一周期才能把数据写入寄存器。）
- 将寄存器的两个读地址分别置为0，1（其实这两个地址一直不用变）。
- 将RAM的读地址改为运算符的对应位置。准备读运算符
- 将RAM的写地址改为相应位置。

T2周期：

接着：

- T0周期读入的第一个操作数已经在T1时存入reg[0]

接着：

- T1周期中读入的数据会立即存入reg[1]。
- 寄存器会更新两个读数据。

这两步相当于是写reg[1]的同时读reg[1]，事实证明，读入的是新数据。

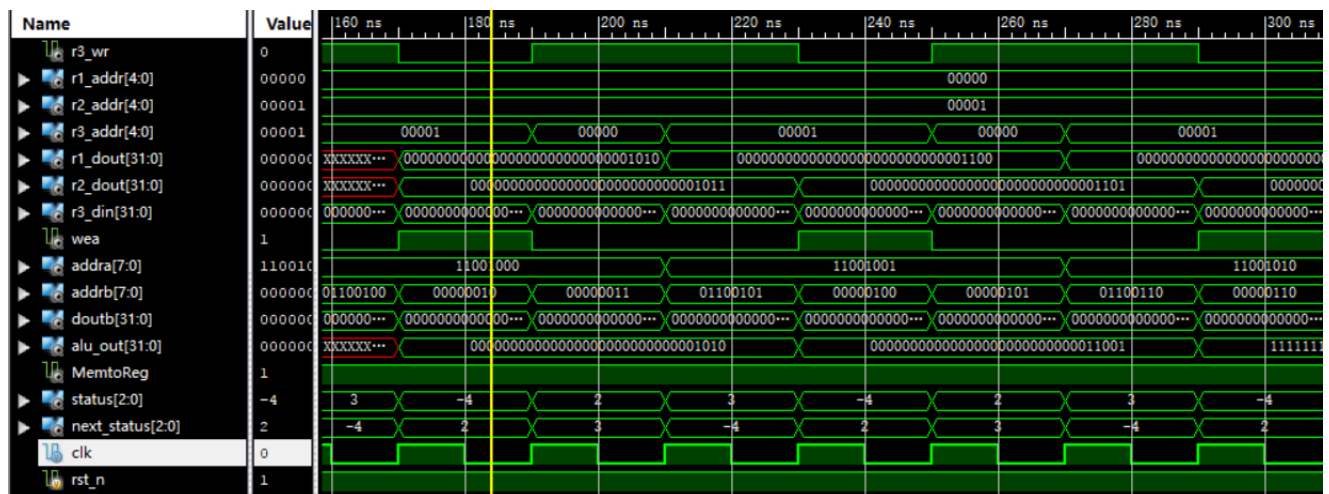
- RAM会开始读运算符，不久后运算符就通过线网输入到ALU。
- 打开RAM的写使能，准备在下周期写入。

然后：

- ALU是组合逻辑，在以上几步做完后可以立即算出结果。



再次返回T0周期。

仿真结果



RAM:

| | | | | |
|-----|----|----|-----|----|
| 196 | 0 | 0 | 0 | 0 |
| 200 | 10 | 25 | -1 | 16 |
| 204 | 19 | 1 | -24 | 0 |
| 208 | 0 | 0 | 0 | 0 |
| 212 | 0 | 0 | 0 | 0 |
| 216 | 0 | 0 | 0 | 0 |
| 220 | 0 | 0 | 0 | 0 |
| 224 | 0 | 0 | 0 | 0 |
| 228 | 0 | 0 | 0 | 0 |
| 232 | 0 | 0 | 0 | 0 |
| 236 | 0 | 0 | 0 | 0 |
| 240 | 0 | 0 | 0 | 0 |
| 244 | 0 | 0 | 0 | 0 |
| 248 | 0 | 0 | 0 | 0 |
| 252 | 0 | 0 | 0 | 0 |

Default.wcfg*

源代码

```

1  //alu.v
2  `timescale 1ns / 1ps
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
10 parameter
11     A_NOP = 5'h00, //空运算
12     A_ADD = 5'h01, //符号加
13     A_SUB = 5'h02, //符号减
14     A_AND = 5'h03, //与
15     A_OR  = 5'h04, //或
16     A_XOR = 5'h05, //异或
17     A_NOR = 5'h06; //或非
18
19
20 always @(*) begin
21     case(alu_op)
22         A_NOP: alu_out = alu_a;
23         A_ADD: alu_out = alu_a + alu_b;
24         A_SUB: alu_out = alu_a - alu_b;
25         A_AND: alu_out = alu_a & alu_b;
26         A_OR:  alu_out = alu_a | alu_b;
27         A_XOR: alu_out = alu_a ^ alu_b;
28         A_NOR: alu_out = ~(alu_a | alu_b);
29     endcase
30 end
31
32 endmodule

```

仿真模块

```

1  `timescale 1ns / 1ps
2
3  module top_test;
4
5      // Inputs
6      reg clk;
7      reg rst_n;
8
9      // wires
10     wire r3_wr;
11     wire [4:0] r1_addr;
12     wire [4:0] r2_addr;
13     wire [4:0] r3_addr;
14     wire [31:0] r1_dout;
15     wire [31:0] r2_dout;
16     wire [31:0] r3_din;
17     wire wea;
18     wire [7:0] addra;
19     wire [7:0] addrb;
20     wire [31:0] doutb;
21     wire [31:0] alu_out;
22     wire MemtoReg;
23     wire [2:0] status, next_status;
24
25     // Instantiate the Unit Under Test (UUT)
26     top uut (
27         .clk(clk),
28         .rst_n(rst_n),
29         .r3_wr(r3_wr),
30         .r1_addr(r1_addr),
31         .r2_addr(r2_addr),
32         .r3_addr(r3_addr),
33         .r1_dout(r1_dout),
34         .r2_dout(r2_dout),
35         .r3_din(r3_din),
36         .wea(wea),
37         .addra(addra),
38         .addrb(addrb),
39         .doutb(doutb),
40         .alu_out(alu_out),
41         .MemtoReg(MemtoReg),
42         .status(status),
43         .next_status(next_status)
44     );
45
46     initial begin
47         clk = 0;
48         #10;
49         forever begin
50             clk <= ~clk;
51             #10;
52         end
53     end

```

```
54
55     initial begin
56         // Initialize Inputs
57         clk = 0;
58         rst_n = 0;
59
60         // Wait 100 ns for global reset to finish
61         #100;
62
63         // Add stimulus here
64         rst_n = 1;
65     end
66
67 endmodule
68
```

顶层模块:

```

1  `timescale 1ns / 1ps
2  module top(
3      input clk,
4      input rst_n,
5      output r3_wr,
6      output [4:0] r1_addr,
7      output [4:0] r2_addr,
8      output [4:0] r3_addr,
9      output [31:0] r1_dout,
10     output [31:0] r2_dout,
11     output [31:0] r3_din,
12     output wea,
13     output [7:0] addra,
14     output [7:0] addrb,
15     output [31:0] doutb,
16     output [31:0] alu_out,
17     output MemtoReg,
18     output [2:0] status, next_status
19 );
20
21 parameter alu_op = 5'h01;
22
23
24
25 ram u_ram_32x512(
26     .clka(clk),
27     .wea(wea),
28     .addra(addra),
29     .dina(alu_out),    /*直接接入alu_out*/
30     .clkb(clk),
31     .addrb(addrb),
32     .doutb(doutb)
33 );
34
35 regfile_32x32 u_regfile_32x32(
36     .clk(clk),
37     .rst_n(rst_n),
38     .r3_wr(r3_wr),
39     .r1_addr(r1_addr),
40     .r2_addr(r2_addr),
41     .r3_addr(r3_addr),
42     .r3_din(r3_din),
43     .r1_dout(r1_dout),
44     .r2_dout(r2_dout)
45 );
46
47 alu u_alu(
48     .alu_a(r1_dout),
49     .alu_b(r2_dout),
50     .alu_op(doutb[4:0]),
51     .alu_out(alu_out)
52 );
53

```

```
54 control u_control(  
55     .clk(clk),  
56     .rst_n(rst_n),  
57     .doutb(doutb),  
58     .r3_din(r3_din),  
59     .alu_out(alu_out),  
60     .r3_wr(r3_wr),           // regfile 写使能  
61     .r3_addr(r3_addr),      // regfile 写地址  
62     .r1_addr(r1_addr),  
63     .r2_addr(r2_addr),      // regfile 读地址  
64     .wea(wea),              // ram 写使能  
65     .addra(addra),          // ram 写地址  
66     .addrb(addrb),          // ram 读地址  
67     .MemtoReg(MemtoReg),  
68     .status(status),  
69     .next_status(next_status)  
70 );  
71  
72 endmodule
```

regfile:


```

1  `timescale 1ns / 1ps
2
3  module regfile_32x32(
4      input clk,
5      input rst_n,
6      input r3_wr,
7      input [4:0] r1_addr,
8      input [4:0] r2_addr,
9      input [4:0] r3_addr,
10     input [31:0] r3_din,
11     output reg [31:0] r1_dout,
12     output reg [31:0] r2_dout
13 );
14
15 parameter regsize = 32;
16 reg [31:0] R [0:31];
17 integer k;
18
19 always @(posedge clk or negedge rst_n) begin
20     if (~rst_n) begin
21         end
22     else begin
23         r1_dout <= R[r1_addr];
24         r2_dout <= R[r2_addr];
25     end
26 end
27
28 always @(negedge clk or negedge rst_n) begin
29     if (~rst_n) begin
30         for (k = 0; k < 32; k = k + 1) begin
31             R[k] <= 0;
32         end
33     end
34     else if (r3_wr) begin
35         R[r3_addr] <= r3_din;
36     end
37 end
38
39 endmodule

```

control模块（内含MemtoReg选择器）

```

1  `timescale 1ns / 1ps
2  module control(
3      input clk,
4      input rst_n,
5      input [31:0] doutb,
6      input [31:0] alu_out,
7      output [31:0] r3_din,
8      output reg MemtoReg,
9      output reg r3_wr,                // regfile 写使能
10     output reg [4:0] r3_addr,         // regfile 写地址
11     output reg [4:0] r1_addr, r2_addr, // regfile 读地址
12     output reg wea,                  // ram 写使能
13     output reg [7:0] addra,          // ram 写地址
14     output reg [7:0] addrb,          // ram 读地址
15     output reg [2:0] status, next_status
16 );
17
18 reg [4:0] cur;
19
20 parameter
21     S0 = 0, S1 = 1,
22     S2 = 2, S3 = 3,
23     S4 = 4, S5 = 5,
24     S6 = 6, S7 = 7;
25
26 always @(posedge clk or negedge rst_n) begin
27     if (~rst_n) begin
28         status <= S0;
29     end
30     else begin
31         status <= next_status;
32     end
33 end
34
35 always @(*) begin
36     case(status)
37         S0: next_status = S1;
38         S1: next_status = S2;
39         S2: next_status = S3;
40         S3: next_status = S4;
41         S4: next_status = (doutb != -1) ? S2 : S5;
42         S5: next_status = S5;
43         default: next_status = S0;
44     endcase
45 end
46
47 //MemtoReg:
48 assign r3_din = MemtoReg ? doutb : alu_out;
49
50 always @(posedge clk or negedge rst_n) begin
51     if (~rst_n) begin
52         cur <= 0;
53         wea <= 0;

```

```

54     r3_wr <= 0;
55 end
56 else if (next_status == S1) begin
57     wea <= 0;
58     r3_wr <= 0;
59
60     addrb <= 2*cur;          /* read ram[cur] */
61 end
62 else if (next_status == S2) begin
63     wea <= 0;
64
65
66     r3_wr <= 1;
67     r3_addr <= 0;          /* write reg[0] */
68     MemtoReg <= 1;
69
70     addrb <= 2*cur + 1;    /* read ram[cur+1] */
71
72 end
73 else if (next_status == S3) begin
74     wea <= 0;
75
76     r3_wr <= 1;
77     r3_addr <= 1;
78     MemtoReg <= 1;
79
80     r1_addr <= 0;          /* read reg[0] */
81     r2_addr <= 1;          /* read reg[1] */
82     addrb <= cur + 100;    /* read ram[cur+100] 运算符 */
83     addra <= cur + 200;
84     cur <= cur + 1;
85 end
86 else if (next_status == S4) begin
87     if(doutb != -1)begin
88         wea <= 1;
89
90         r3_wr <= 0;
91
92         addrb <= 2*cur;    /* read ram[cur] */
93     end
94 end
95 else if (next_status == S5) begin
96     wea <= 0;
97     r3_wr <= 0;
98 end
99 end
100
101 endmodule
102
103

```

