

Lab3_Ram

金泽文 PB15111604

实验目的：

学习如何使用 ISE 的 IP 核

学习使用 Xilinx FPGA 内的 RAM 资源

例化一个简单双端口的 RAM (32bitx64)

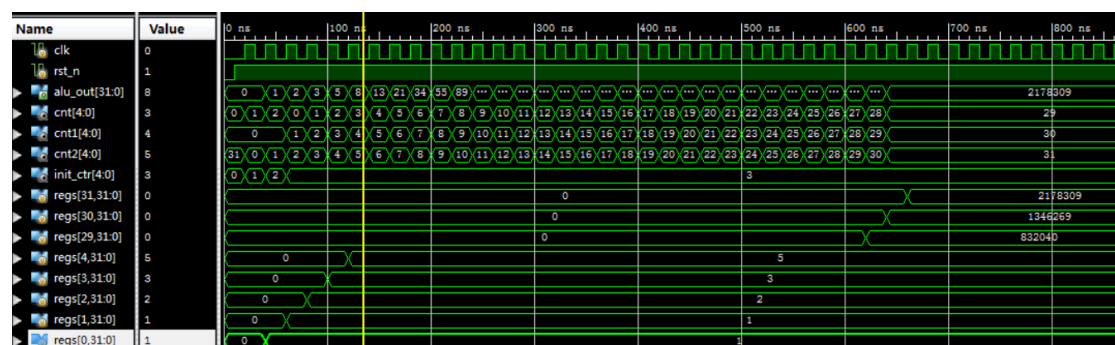
使用 coe 文件对 RAM 进行初始化

实验内容：

- 综合利用三次实验的结果，完成以下功能：
 - 从 ram 中 0 地址和 1 地址读取两个数， 分别赋给 reg0 和 reg1
 - 利用第二次实验的结果(ALU+Regfile)进行斐波拉契运算 ,运算结果保存在对应的寄存器
 - 运算结果同时保存在对应的 ram 地址中，即 $ram[0] <-----> reg0$,
 $ram[1] <-----> reg1$,
 $ram[2] <-----> reg2, \dots$

实验结果：

按要求完成。仿真结果如图：



	0	1
31	2178309	1346269
29	832040	514229
27	317811	196418
25	121393	75025
23	46368	28657
21	17711	10946
19	6765	4181
17	2584	1597
15	987	610
13	377	233
11	144	89
9	55	34
7	21	13
5	8	5
3	3	2
1	1	1

实验分析与设计：

ALU 模块完全同于上次实验。

Regfile 完全同于上次实验。

Top 不同于上次 ,利用 top 来控制两个阶段 :从 ram 读入与从 reg 读入 ,利用 control 模块输出的 step_ctr 来控制 reg 以及 ram 的使能实现。

新增的 control 利用 step_ctr 来控制从 reg 还是 ram 读入。刚开始初始化前两个 reg ,之后是相加与写入。鉴于 ram 输出的延迟 ,复位时用三个周期初始化。变量 cnt 负责写入后面寄存器阶段的计数。


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：

```
1  `timescale 1ns / 1ps
2
3  module top(
4  input clk,
5  input rst_n,
6  input [4:0] alu_op
7  );
8
9  wire [31:0] alu_out , r1_dout , r2_dout , doutb;
10 wire [4:0] cnt , cnt1 , cnt2 ;
11 wire ctr;
12 wire [4:0] step_control;
13 reg [31:0] r3 = 0;
14 reg we = 0;
15 ALU ALU1(r1_dout, r2_dout, alu_op, alu_out);
16 REG_FILE REG_FILE1(clk, rst_n, cnt, cnt1, cnt2, r3, ctr, r1_dout, r2_dout);
17 ram ram1(clk, we , cnt2, alu_out, clk, cnt, doutb);
18 control control1(clk, rst_n, cnt, cnt1, cnt2, ctr, step_control);
19
20 always@(*)
21 begin
22     if(step_control < 3)
23         r3 = doutb;
24     else
25         r3 = alu_out;
26 end
27
28 always@(*)
29 begin
30     if(step_control < 3)
31         we = 0;
32     else if( cnt < 29 )
33         we = 1;
34 end
35
36 endmodule
37
```

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
```

Control :

```
1  `timescale 1ns / 1ps
2
3  module control(
4  input clk,
5  input rst_n,
6  output reg [4:0] cnt,
7  output reg [4:0] cnt1,
8  output reg [4:0] cnt2,
9  output reg ctr,
10 output reg [4:0] init_ctr
11 );
12
13 reg [4:0] c_cnt = 0;
14 reg c_ctr = 0;
15 reg [4:0] init = 0;
16
17 always@(posedge clk,negedge rst_n) begin
18     if(~rst_n)
19         init <= 0;
20     else if(init < 3)
21         init <= init + 1;
22 end
23
24 always@(*)
25 begin
26     if(init == 3) begin
27         cnt = c_cnt ;
28         cnt1 = c_cnt + 1 ;
29         cnt2 = c_cnt + 2 ;
30         ctr = c_ctr ;
31     end
32     else begin
33         cnt = init ;
34         cnt1 = c_cnt ;
35         cnt2 = init - 1 ;
36         ctr = 1 ;
37     end
38 end
39
40 always@(*) begin
41     init_ctr = init;
```

```
22 end
23
24 always@(*)
25 begin
26     if(init == 3) begin
27         cnt = c_cnt ;
28         cnt1 = c_cnt + 1 ;
29         cnt2 = c_cnt + 2 ;
30         ctr = c_ctr ;
31     end
32     else begin
33         cnt = init ;
34         cnt1 = c_cnt ;
35         cnt2 = init - 1 ;
36         ctr = 1 ;
37     end
38 end
39
40 always@(*) begin
41     init_ctr = init;
42 end
43
44 always@(posedge clk,negedge rst_n) begin
45     if(~rst_n)
46         c_cnt <= 0;
47     else if(init < 3)
48         c_cnt <= 0 ;
49     else if(c_cnt <= 5'd28 && c_ctr == 1)
50         c_cnt <= c_cnt + 1;
51 end
52
53 always@(posedge clk,negedge rst_n) begin
54     if(~rst_n)
55         c_ctr <= 0;
56     else if(init == 1)
57         c_ctr <= 1;
58     else if(c_cnt == 5'd29)
59         c_ctr <= 0;
60 end
61
62 endmodule
```

Testfile :

```
24
25  module test;
26
27      // Inputs
28      reg clk;
29      reg rst_n;
30      reg [4:0] alu_op;
31
32      // Outputs
33
34
35      // Instantiate the Unit Under Test (UUT)
36      top uut (
37          .clk(clk),
38          .rst_n(rst_n),
39          .alu_op(alu_op)
40      );
41
42      initial begin
43          // Initialize Inputs
44          clk = 0;
45          rst_n = 0;
46          alu_op = 0;
47
48          // Wait 100 ns for global reset to finish
49
50          #10;
51          rst_n = 1;
52          alu_op = 1;
53          forever #10 clk = ~clk;
54          // Add stimulus here
55
56      end
57
58  endmodule
59
```

