实验报告四
PB14000556 陈晓彤

实验题目：运算控制 时序与状态机
实验要求：

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,……
结果在仿真中显示
要求尽量少的时钟周期数完成

设计思路：

1. 在 ISE 中建立一个 IP 核，选择双端口 RAM，a 为写入 RAM 端口，b 为读出 RAM 端口，
   根据实验要求，创建一个 coe 文件对 RAM 进行初始化，
   格式为 MEMORY_INITIALIZATION_RADIX=10;
   MEMORY_INITIALIZATION_VECTOR=
   内容为 1，1，0；
2. 建立控制模块，实现两阶段工作：
   a) 从 RAM[0],RAM[1]取数至 REG[0],REG[1],完成准备工作
   b) REG 与 ALU 交互工作，完成斐波那契数列计算，同时将结果存入 RAM
   实现方法：
   为了尽可能少的周期完成目标，同时也为了使实现便于移植和修改，将数据与地址分开处理，而数据的变化较简单，大部分数据使用 assign 语句一步完成互联，reg_din 需要分两阶段赋值，分别来源于 ram 和 alu；使用时钟分频完成初始化和后来计算的统一控制。
   具体实现：使用 3 位变量 state，复位时从 0 开始，随时钟上升沿每次递增 1，在 1 和 2 时完成两个初值的赋值，3 时完成计算环境初始化，如置 ena=1，wea=1，enb=0 等，到 4 时保持值为 4，进入计算阶段，alu 与 reg，ram 数据读写地址每周期增一。
   最终实现了时钟周期为 1 的运算过程，见截图。

源代码：

核心控制模块

```
module ctrl(
    input clk,
    input rst_n,
    output [31:0] alu_in1,
    output [31:0] alu_in2,
    output [31:0] alu_out
    );
    reg [4:0] reg_aout1;
    reg [4:0] reg_aout2;
    reg [4:0] reg_ain;
    wire [31:0] reg_dout1;
    wire [31:0] reg_dout2;
```

```
    reg [31:0] reg_din;
    reg [5:0] ram_ain;
    reg [5:0] ram_aout;
    wire [31:0] ram_din;
    wire [31:0] ram_dout;

    alu alu1(
    .alu_a      (alu_in1),
   .alu_b(alu_in2),
   .alu_op     (5'b1),
   .alu_out    (alu_out)
    );
    regfile reg1(
    .clk  (clk),
    .rst_n      (rst_n),
    .r1_addr  (reg_aout1),
    .r2_addr  (reg_aout2),
    .r3_addr  (reg_ain),
    .r3_din     (reg_din),
    .r3_wr      (1'b1),
    .r1_dout  (reg_dout1),
    .r2_dout  (reg_dout2)
    );
    ram ram1(
    .clka       (clk),
    .ena        (state[2]),
    .wea        (state[2]),
    .addra      (ram_ain),
    .dina               (ram_din),
    .clkb       (clk),
    .rstb       (1'b0),
    .enb        (~state[2]),
    .addrb      (ram_aout),
    .doutb      (ram_dout)
    );

    reg [2:0] state;
    assign ram_din=alu_out;
    assign alu_in1=reg_dout1;
    assign alu_in2=reg_dout2;
always@(posedge clk or negedge rst_n)
    begin
    if(~rst_n)
          state<=3'b0;
```

```
    else if(state==3'b100)
        state<=state;
    else
        state<=state+3'b1;
    end

always@(*)
    begin
    if(state[2])
        reg_din<=alu_out;
    else
        reg_din<=ram_dout;
    end

always@(posedge clk)
    begin
        case(state)
            3'b0:begin    ram_aout<=6'b0;reg_ain<=5'b0;end
            3'b1:begin    ram_aout<=6'b0;reg_ain<=5'b0;end
            3'b10:begin   ram_aout<=6'b1;reg_ain<=5'b1;end
            3'b11:begin
    reg_ain<=5'b10;ram_ain<=6'b10;reg_aout1<=5'b0;reg_aout2<=5'b1; end
            default:
            begin
                reg_ain<=reg_ain+5'b1;
                reg_aout1<=reg_aout1+5'b1;reg_aout2<=reg_aout2+5'b1;
                ram_ain<=ram_ain+6'b1;
            end
        endcase
    end

endmodule
```

寄存器模块

```
module regfile(
    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] regfile [31:0];
    integer i;
always@(posedge clk or negedge rst_n)
    begin
        if(~rst_n)
        begin
            for(i=0;i<32;i=i+1)
                regfile[i]<=32'b0;
        end
        else
            if(r3_wr)
                regfile[r3_addr]<=r3_din;
    end

always@(negedge clk)
    begin
        r1_dout<=regfile[r1_addr];
        r2_dout<=regfile[r2_addr];
    end
endmodule
```

运算器模块

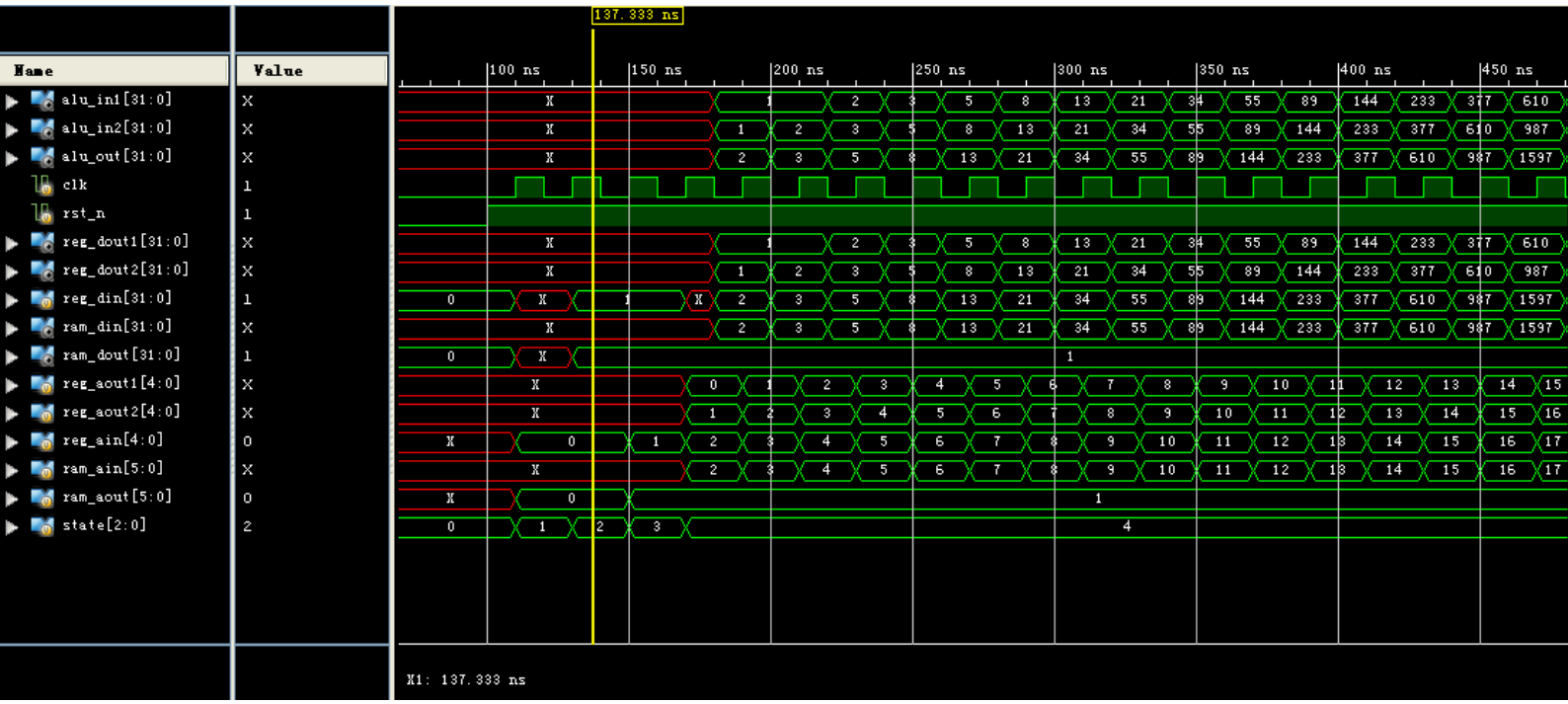
```
module alu(
  input   signed   [31:0] alu_a,
  input   signed   [31:0] alu_b,
  input                [4:0]       alu_op,
  output   reg        [31:0] alu_out
);
    parameter    A_NOP = 5'h00; //空运算
    parameter    A_ADD   = 5'h01;  //符号加
    parameter    A_SUB    = 5'h02;  //符号减
    parameter    A_AND = 5'h03;    //与
    parameter    A_OR    = 5'h04;    //或
    parameter    A_XOR = 5'h05;    //异或
    parameter    A_NOR = 5'h06;    //或非
    reg sign;
    always@(*)
    begin
        case(alu_op)
            A_ADD:{sign,alu_out}<=alu_a+alu_b;
            A_SUB:{sign,alu_out}<=alu_a-alu_b;
            A_AND:alu_out<=alu_a&alu_b;
            A_OR:alu_out<=alu_a|alu_b;
```

```
                A_XOR:alu_out<=alu_a^alu_b;
                A_NOR:alu_out<=~(alu_a|alu_b);
                A_NOP:alu_out<=alu_out;
                default:alu_out<=alu_out;
            endcase
        end
endmodule
```

测试模块

```
module test;

        // Inputs
        reg clk;
        reg rst_n;

        // Outputs
        wire [31:0] alu_in1;
        wire [31:0] alu_in2;
        wire [31:0] alu_out;

        // Instantiate the Unit Under Test (UUT)
        ctrl uut (
                .clk(clk),
                .rst_n(rst_n),
                .alu_in1(alu_in1),
                .alu_in2(alu_in2),
                .alu_out(alu_out)
        );

        initial begin
                // Initialize Inputs
                clk = 0;
                rst_n = 0;

                // Wait 100 ns for global reset to finish
                #100;
           rst_n = 1;
                forever #10 clk = ~clk;
                // Add stimulus here

        end

endmodule
```

仿真图：



	0	1	2	3
0x0	1	1	2	3
0x4	5	8	13	21
0x8	34	55	89	144
0xC	233	377	610	987
0x10	1597	2584	4181	6765
0x14	10946	17711	28657	46368
0x18	75025	121393	196418	317811
0x1C	514229	832040	1346269	2178309
0x20	3524578	5702887	9227465	14930352
0x24	24157817	39088169	63245986	102334155
0x28	165580141	267914296	433494437	0
0x2C	0	0	0	0
0x30	0	0	0	0
0x34	0	0	0	0
0x38	0	0	0	0
0x3C	0	0	0	0

	0	1	2	3
0x1F	2178309	1346269	832040	514229
0x1B	317811	196418	121393	75025
0x17	46368	28657	17711	10946
0x13	6765	4181	2584	1597
0xF	987	610	377	233
0xB	144	433494437	267914296	165580141
0x7	102334155	63245986	39088169	24157817
0x3	14930352	9227465	5702887	3524578