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；

1. 建立控制模块，实现两阶段工作：
   1. 从RAM[0],RAM[1]取数至REG[0],REG[1],完成准备工作
   2. 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;

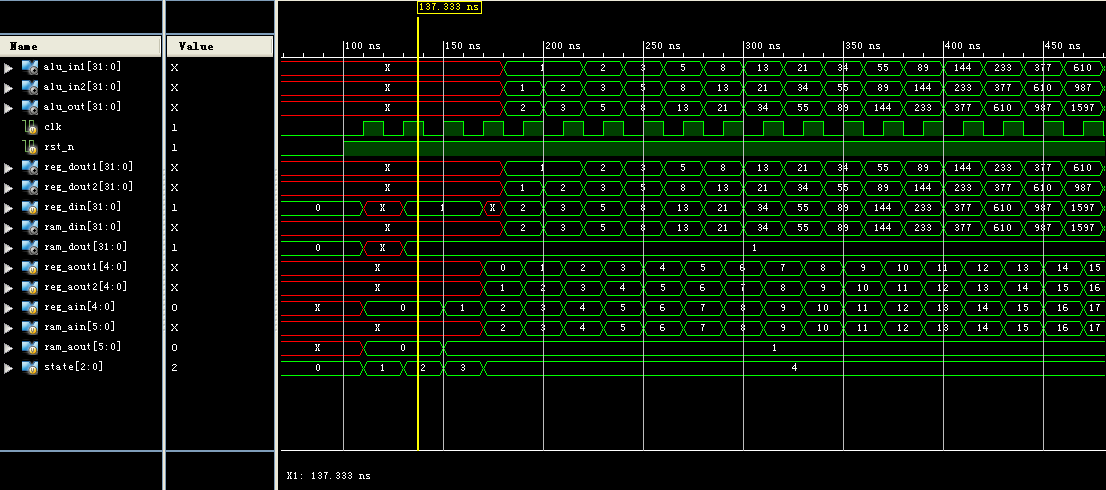
rst\_n = 1;

forever #10 clk = ~clk;

// Add stimulus here

end

endmodule

仿真图：

