

## 实验三：存储器 RAM

姓名：姜庆彩

学号：PB15051087

- 实验内容

- 1.学习如何使用 ISE 的 IP 核
- 2.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

- 实验功能要求

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,.....

- 实验设计要求

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

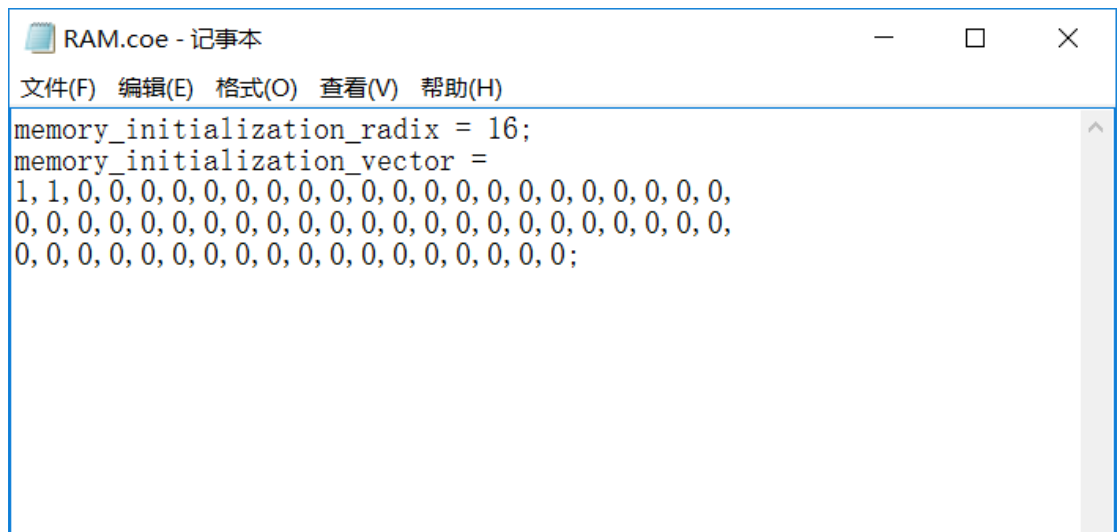
调用 Ram 模块

调用 RegFile

调用 ALU 完成加法运算

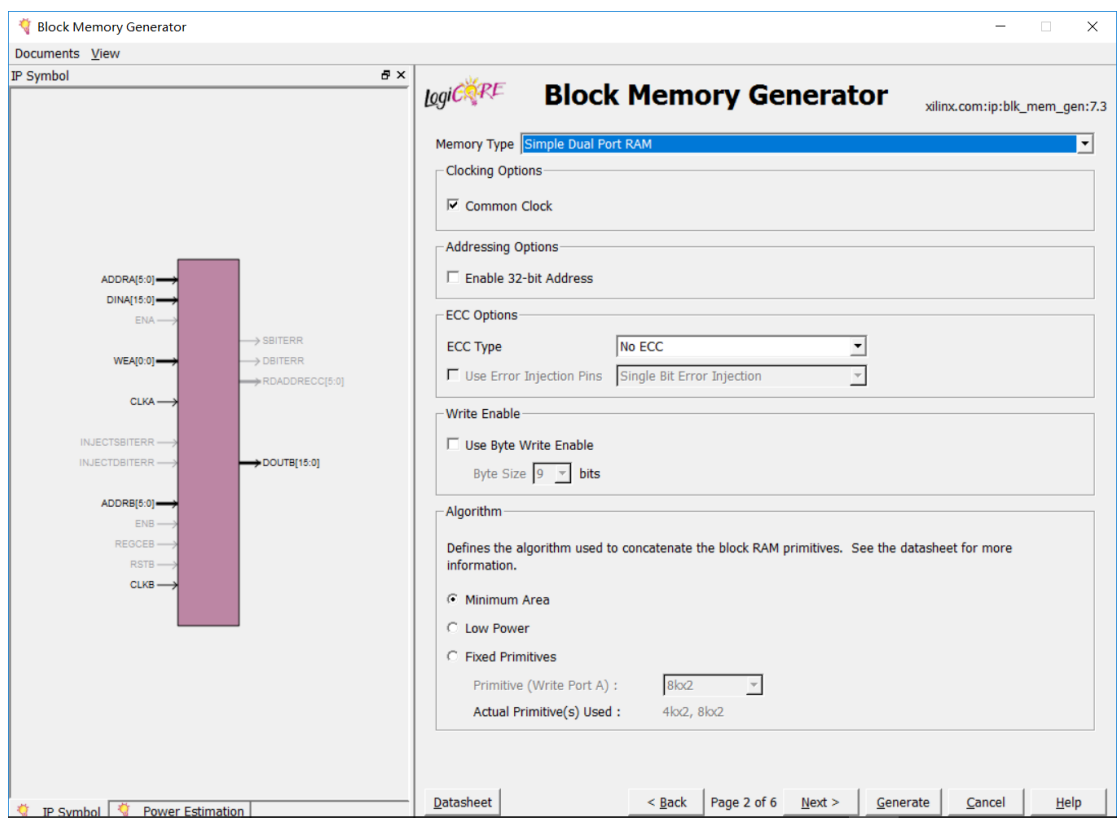
调用 control 模块，完成运算控制

- 我的实现过程



```
memory_initialization_radix = 16;
memory_initialization_vector =
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
```

首先写一个 coe 文件

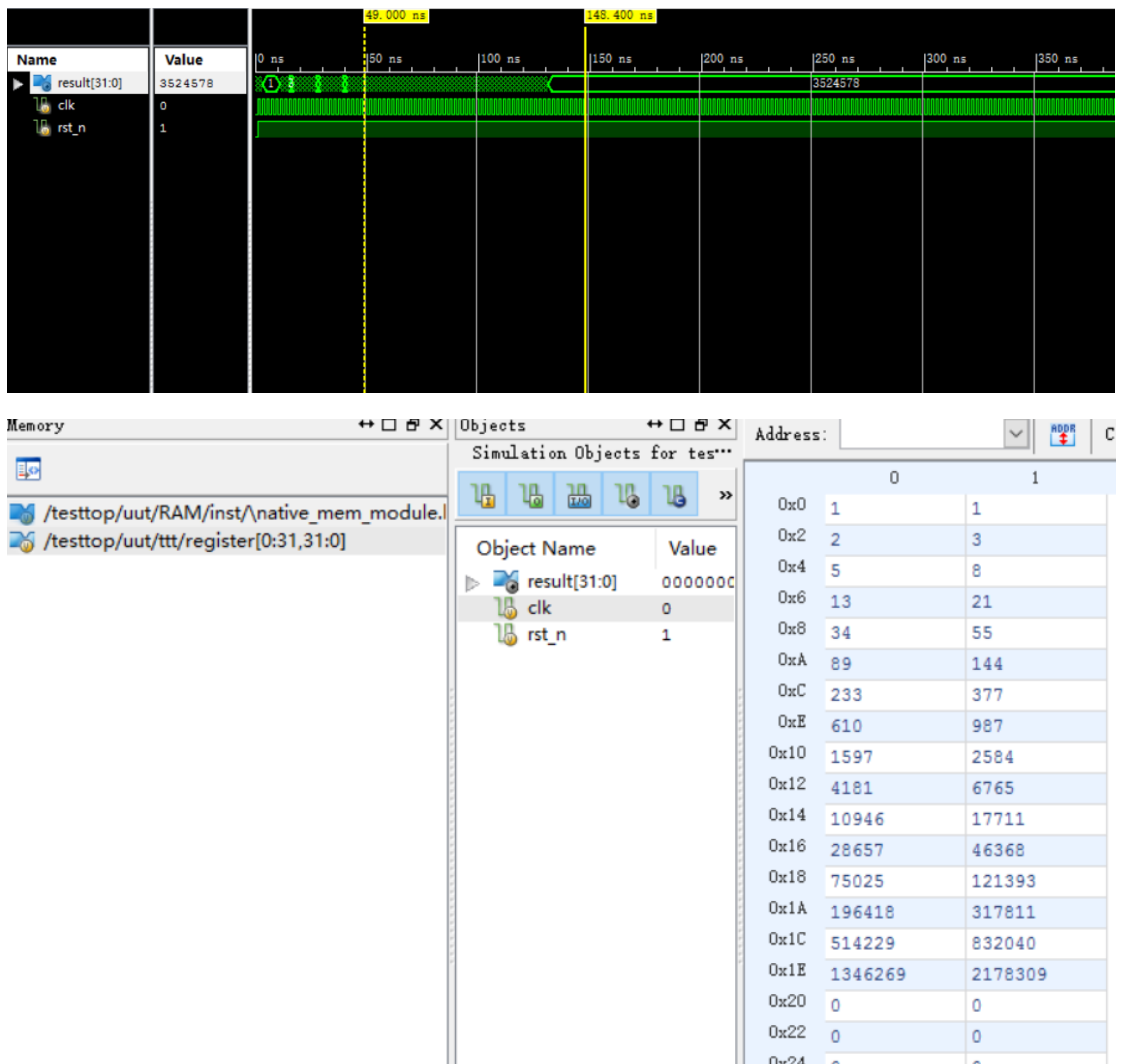


接着，例化一个 ram  
然后把代码部分完成，调试，得到正确的仿真结果。

- 代码部分  
这次的代码部分中的 alu 模块与 REG\_FILE 模块与上次相同，control 模块中则添加了一个 always 块，用来判断 din 是由 ram 给出还是有 alu 运算

给出（前两次由 ram 读出，后面的由 alu 给出）

● 实验结果



- 源代码:

```
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  
);  
always@(*)  
begin  
    case(alu_op)  
        1:  alu_out <= alu_a + alu_b;  
        2:  alu_out <= alu_a - alu_b;  
        3:  alu_out <= alu_a & alu_b;  
        4:      alu_out <= alu_a | alu_b;  
        5:      alu_out <= alu_a ^ alu_b;  
        6:      alu_out <= ~(alu_a | alu_b);  
        default: alu_out <= alu_out;  
    endcase  
end  
  
endmodule
```

```

        module REG_FILE(
            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] register[0:31];
        integer k,i;
        always@(posedge clk or negedge rst_n)
        begin
            if(~rst_n)
            begin
                register[0]=32'b0;
                register[1]=32'b0;
                for(k=2;k<32;k=k+1)
                    register[k]=32'b0;
            end
            else if(r3_wr)
            begin
                i=r3_addr;
                register[i]=r3_din;
                i=r1_addr;
                r1_dout=register[i];
                i=r2_addr;
                r2_dout=register[i];
            end
            else
            begin
                i=r1_addr;
                r1_dout=register[i];
                i=r2_addr;
                r2_dout=register[i];
            end
        end
    end

```

endmodule

```

module control(
input clk,
input rst_n,
input [31:0] doutb,
input [31:0] aluin,
output reg [31:0] din,
output reg wea,
output reg [5:0] addra,
output reg [5:0] addrb,
output reg [4:0] r1_addr,
output reg [4:0] r2_addr,
output reg [4:0] r3_addr,
output reg r3_wr
);
    reg state;
    reg [2:0] read;
always@(posedge clk or negedge rst_n)
begin
    if(~rst_n)
    begin
        read=2'b00;
        addrb=6'b00;
    end
    else if(read<=2'b10)
    begin
        read=read+1'b1;
        addrb=addrb+1'b1;
        din=doutb;
    end
    else
        din=aluin;
end
always@(posedge clk or negedge rst_n)
begin
    if(~rst_n)
    begin
        state=1'b0;
        wea=1'b0;
        addra=5'b01;
        r1_addr=5'b00;
        r2_addr=5'b01;
        r3_addr=5'b00;
        r3_wr=1;
    end

```

```

else
begin
if(state==1'b0&&read>2'b01)
begin
r3_wr=1;
state=1'b1;
end
else if(state==1'b1 &&r3_addr<5'b101111)
begin
r3_wr=0;
wea=0;
r3_addr=r3_addr+1;
state=1'b0;
if(r3_addr>=5'b10)
begin
r1_addr=r1_addr+5'b1;
r2_addr=r2_addr+5'b1;
addra=addra+5'b1;
wea=1;
end
end
else
begin
wea=0;
r3_wr=0;
end
end
end

```

```

endmodule

```

```

module top(
input clk,
input rst_n,
output [31:0] result
);
    assign result=r3_din;
    wire [4:0] alu_op;
    wire [31:0] r1_dout;
    wire [31:0] r2_dout;
    wire [31:0] r3_din;
    wire [4:0] r1_addr;
    wire [4:0] r2_addr;
    wire [4:0] r3_addr;
    wire r3_wr;
    wire wea;
    wire [31:0] doutb;
    wire [5:0] addra;
    wire [5:0] addrb;
    wire [31:0] alu_in;
    assign alu_op=5'h01;
    ram RAM(
        .clka(clk),
        .wea(wea),
        .addra(addra),
        .dina(r3_din),
        .clkb(clk),
        .addrb(addrb),
        .doutb(doutb)
    );
    control uut(
        .clk(clk),
        .rst_n(rst_n),
        .doutb(doutb),
        .aluin(alu_in),
        .din(r3_din),
        .wea(wea),
        .addra(addra),
        .addrb(addrb),
        .r1_addr(r1_addr),
        .r2_addr(r2_addr),
        .r3_addr(r3_addr),
        .r3_wr(r3_wr)
    );
    alu utt(

```



```
        .alu_a(r1_dout),  
        .alu_b(r2_dout),  
        .alu_op(alu_op),  
        .alu_out(alu_in)  
    );  
    REG_FILE ttt(  
        .clk(clk),  
        .rst_n(rst_n),  
        .r1_addr(r1_addr),  
        .r2_addr(r2_addr),  
        .r3_addr(r3_addr),  
        .r3_din(r3_din),  
        .r3_wr(r3_wr),  
        .r1_dout(r1_dout),  
        .r2_dout(r2_dout)  
    );  
endmodule
```

```

module testtop;

    // Inputs
    reg clk;
    reg rst_n;

    // Outputs
    wire [31:0] result;

    // Instantiate the Unit Under Test (UUT)
    top uut (
        .clk(clk),
        .rst_n(rst_n),
        .result(result)
    );

    initial begin
        // Initialize Inputs
        clk = 1;
        rst_n = 0;

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

    end
    always
    begin
        clk = ~clk;
        #1;
    end
endmodule

```