

计算机组成原理 实验报告

姓名：魏钊 学号：PB18111699 实验日期：2020-5-4

一、实验题目：

实验二 寄存器堆与队列

二、实验目的：

- 1.掌握寄存器堆（Register File）和存储器（Memory）的功能、时序及其应用；
- 2.熟练掌握数据通路和控制器的设计和描述方法。

三、实验平台：

ISE / Vivado（暂不支持其他 Verilog HDL 开发环境的检查）

四、实验过程：

寄存器堆：

```
module register_file

#(parameter WIDTH = 32)

(

    input clk,                      //时钟（上升沿有效）

    input [4:0] ra0,                 //读端口 0 地址

    output [WIDTH-1:0] rd0,          //读端口 0 数据

    input [4:0] ra1,                 //读端口 1 地址

    output [WIDTH-1:0] rd1,          //读端口 1 数据

    input [4:0] wa,                  //写端口地址

    input we,                        //写使能，高电平有效

    input [WIDTH-1:0] wd             //写端口数据

);

reg [4:0] addr_reg;

reg [WIDTH-1:0] mem[0:31];
```

```

assign rd0=mem[ra0];

assign rd1=mem[ra1];


always@(*)

    mem[0]=1'b0;


always@(posedge clk)

begin

    if(we)

        if(~(wa==5'b0))

            mem[wa]=wd;

    mem[0]=0;

end

endmodule

```

该寄存器堆含有 32 个寄存器（r0 ~ r31，其中 r0 的内容恒定为零），寄存器的位宽由参数 **WIDTH** 指定，具有 2 个异步读端口和 1 个同步写端口。

存储器：

例如，设计一容量为 16 x 8 位（即深度 DEPTH：16，宽度 WIDTH：8）的单端口 RAM，其逻辑符号如图-2 所示。用行为方式描述的 Verilog 代码如下：（请补充代码中空缺的参数）

```

module ram_16x8                                //16x8 位单端口 RAM
(
    input clk,                                //时钟（上升沿有效）
    input en, we,                             //使能，写使能
    input [ __3: 0__ ] addr, //地址
    input [ _7: 0__ ] din,                    //输入数据
    output [ __7: 0__ ] dout //输出数据
);
reg [ __3: 0__ ] addr_reg;
reg [ __7: 0__ ] mem[ __0: 16__ ];

//初始化 RAM 的内容
initial
    $readmemh("初始化数据文件名", mem);

```

```
assign dout = mem[addr_reg];
```

```
always@(posedge clk) begin
```

```
  if(en) begin
```

```
    addr_reg <= addr;
```

```
    if(we)
```


```
      mem[addr] <= din;
```

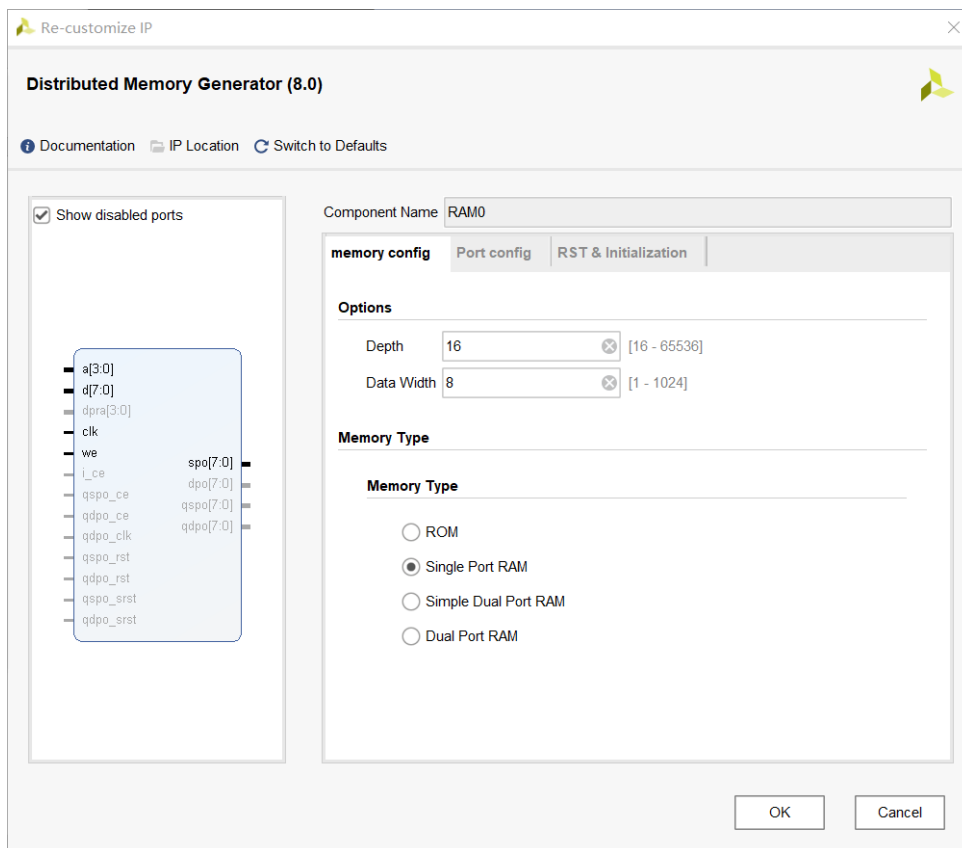
```
  end
```

```
end
```

```
endmodule
```

分布式：

>  RAM0 (RAM0.xci) (1)



The image shows the 'Re-customize IP' window for the 'Distributed Memory Generator (8.0)'. The window has a title bar with a close button. Below the title bar, there are three tabs: 'Documentation', 'IP Location', and 'Switch to Defaults'. The main area is divided into two panes. The left pane, titled 'Show disabled ports', contains a list of ports: a[3:0], d[7:0], dpra[3:0], clk, we, i_ce, qspo_ce, qdpo_ce, qdpo_clk, qspo_rst, qdpo_rst, qspo_srst, and qdpo_srst. The right pane, titled 'Component Name RAM0', contains three tabs: 'memory config', 'Port config', and 'RST & Initialization'. The 'memory config' tab is active. It has two sections: 'Options' and 'Memory Type'. The 'Options' section has two fields: 'Depth' with a value of 16 and a range of [16 - 65536], and 'Data Width' with a value of 8 and a range of [1 - 1024]. The 'Memory Type' section has four radio buttons: 'ROM', 'Single Port RAM' (which is selected), 'Simple Dual Port RAM', and 'Dual Port RAM'. At the bottom right of the window are 'OK' and 'Cancel' buttons.

Re-customize IP

Distributed Memory Generator (8.0)

Documentation IP Location Switch to Defaults

Show disabled ports

Component Name RAM0

memory config Port config RST & Initialization

Options

Depth 16 [16 - 65536]

Data Width 8 [1 - 1024]

Memory Type

Memory Type

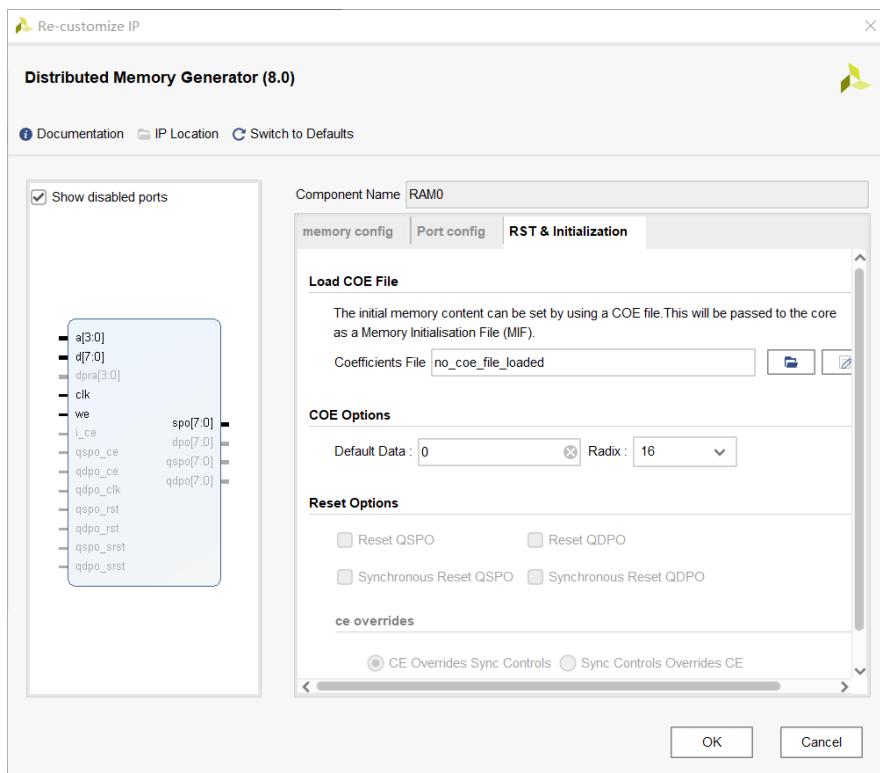
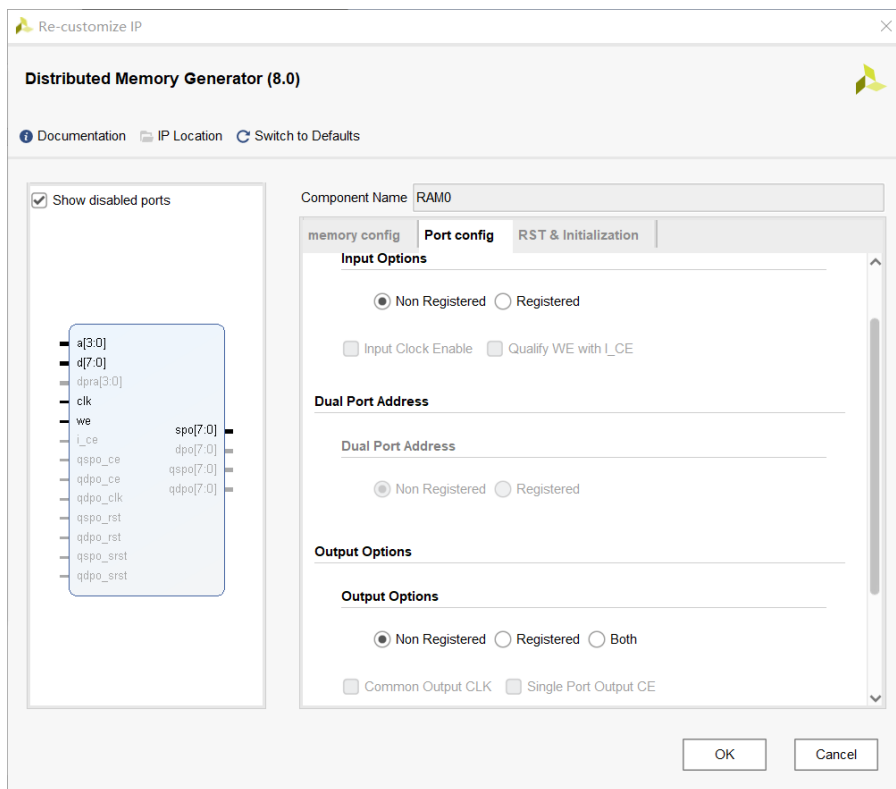
☐ ROM

☒ Single Port RAM


☐ Simple Dual Port RAM

☐ Dual Port RAM

OK Cancel

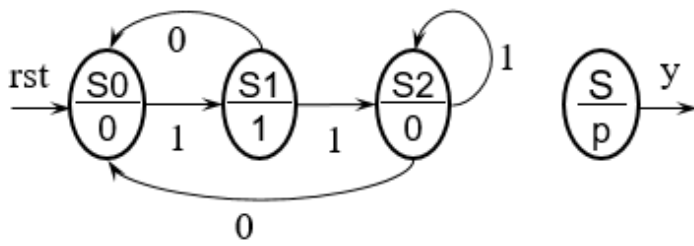


块式:

>  ram : RAM1 (RAM1.xci)

FIFO:

取边沿:



```
module EDG(  
    input clk,  
    input y,  
    input rst,  
    output p  
);  
  
parameter S0=2'b00;  
parameter S1=2'b01;  
parameter S2=2'b10;  
  
reg [1:0] state=S0,next_state;  
  
assign p=(state==S1);  
  
always@(posedge clk or posedge rst)  
    if(rst)  
        state<=S0;  
    else  
        state<=next_state;
```

```

        always@(*)
        begin
            case(state)
                S0:if(y)
                    next_state=S1;
                else
                    next_state=S0;
                S1:if(y)
                    next_state=S2;
                else
                    next_state=S0;
                S2:if(y)
                    next_state=S2;
                else
                    next_state=S0;
            endcase
        end
    endmodule

```

FIFO 声明及相关变量:

```

module FIFO(
    input clk,
    input rst,
    input en_in,
    input [7:0] din,
    input en_out,
    output [7:0] dout,
    output reg [4:0] count

```



```

    );

    reg [3:0] head,tail;//c1 为队头指针, c2 为队尾指针

    wire in,out;

    reg en;

    reg [3:0] addr=4'b0;
reg [1:0] state,next_state;

```

输入输出使能取边沿:

```
EDG edg1(clk,en_in,rst,in);//输入使能取边沿
```

```
EDG edg2(clk,en_out,rst,out);//输出使能取边沿
```

利用例化的存储器 IP（16 x 8 位块式的单端口 RAM）：

```
RAM1 ram(.clka(clk),.ena(en),.wea(in),.addra(addr),.dina(din),.douta(dout));
```

两段式 FSM 控制电路:

```
always@(posedge clk,posedge rst)
```

```
begin
```

```
if(rst)
```

```
begin
```

```
state<=2'b00;
```

```
count<=5'b0;
```

```
head<=4'b0;
```

```
tail<=4'b0;
```

```
addr<=4'b0;
```

```
end
```

```
else
```

```
state<=next_state;
```

```
end
```

```
always@(*)
```

```
begin
```

```
case(state)
```

```
2'b00:
```

```
begin
```

```
count=count;
```

```

        if((in==1'b1) && ~(count==5'b10000))//输入使能且未满足
        begin
            addr=tail;//地址指向尾指针
            en=1'b1;
            next_state=2'b10;
        end
    else
        if((out==1'b1) && ~(count==5'b0))//输出使能且非空
        begin
            addr=head;
            en=1'b1;
            next_state=2'b01;
        end
    else
        begin
            addr=4'bX;
            next_state=2'b00;
            en=1'b0;
        end
    end

end

2'b01://输出
begin
    count=count-5'b1;
    head=head+4'b1;
    next_state=2'b0;
end

2'b10://输入
begin
    count=count+5'b1;
    tail=tail+4'b1;
    next_state=2'b0;
end

default:
    next_state=2'b00;
endcase

```

end

endmodule

五、实验结果：

寄存器堆仿真：

```
module register_file_test;

    reg [31:0] WriteData;

    reg [4:0] ReadRegNum1, ReadRegNum2, WriteRegNum;

    reg Clock, RegWrite;

    wire [31:0] ReadData1, ReadData2;

    register_file regf0
    (.clk(Clock), .rd0(ReadData1), .rd1(ReadData2), .ra0(ReadRegNum1), .ra1(ReadRegNum2), .wd(WriteData), .wa
    (WriteRegNum), .we(RegWrite));

    always

        #5 Clock = ~ Clock;

    initial

    begin

        #0 Clock = 0;

        #0 RegWrite = 1'b1;

        #10 WriteRegNum = 1;

        #0 WriteData = 1;

        #10 WriteRegNum = 2;

        #0 WriteData = 2;

        #10 WriteRegNum = 3;

        #0 WriteData = 3;

        #10 WriteRegNum = 4;
```

```

#0 WriteData = 4;

#10 WriteRegNum = 0;

#0 WriteData = 1;

#10 WriteRegNum = 5;

#0 WriteData = 5;

#10 RegWrite = 1'b0;

#5 ReadRegNum1 = 1;

#0 ReadRegNum2 = 2;

#5 ReadRegNum1 = 3;

#0 ReadRegNum2 = 0;

#5 ReadRegNum1 = 4;

#0 ReadRegNum2 = 5;

#5 WriteRegNum = 1;

#0 RegWrite = 1'b1;

#0 WriteData = 7;

#5 ReadRegNum1 = 1;

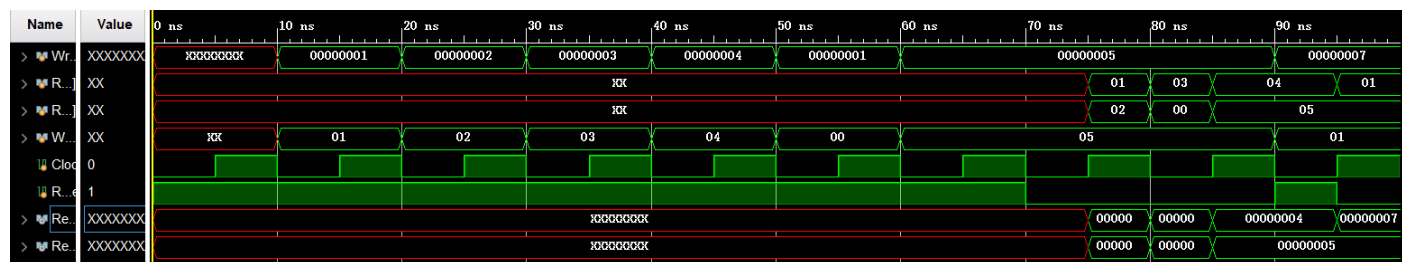
#0 RegWrite = 1'b0;

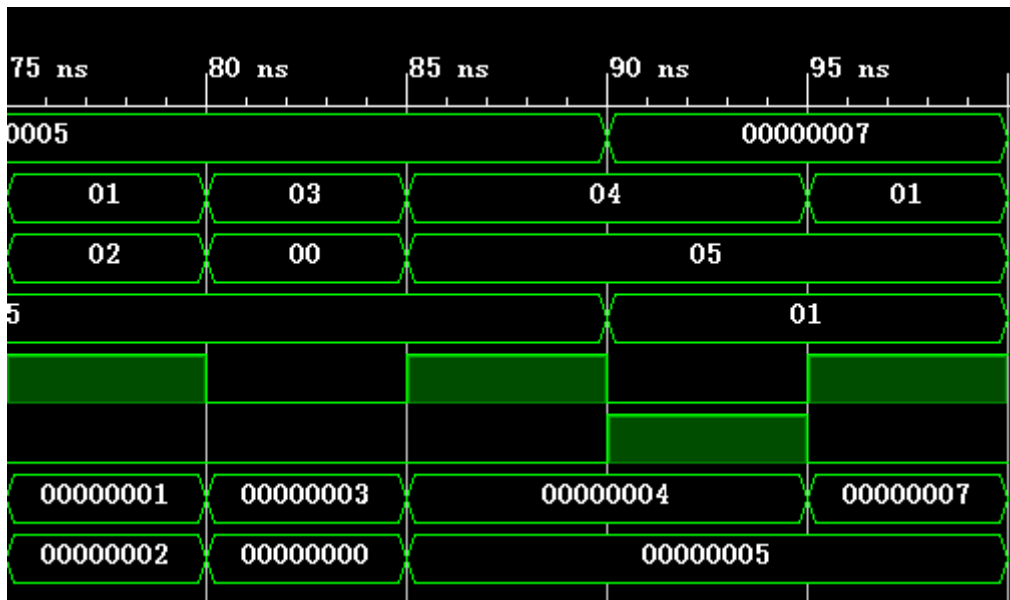
#5 $finish;

```

```
end
```

```
endmodule
```





分布式 RAM:

```

module RAM0_test;

    reg [3 : 0] a;
    reg [7 : 0] d;
    reg clk;
    reg we;
    wire [7 : 0] spo;

    RAM0 ram0(a, d, clk, we, spo);

    always
        #5 clk = ~ clk;

    initial
        begin
            #0 clk = 1'b0;
            #0 we = 1'b1;

            #5 a=1;
            #0 d=1;

            #10 a=4;

```

```

#0 d=4;

#10 a=10;

#0 d=10;

#10 a=15;

#0 d=15;

#10 a=7;

#0 d=7;

#10 a=1;

#0 we=0;

#10 a=4;

#10 a=10;

#10 a=15;

#10 a=7;

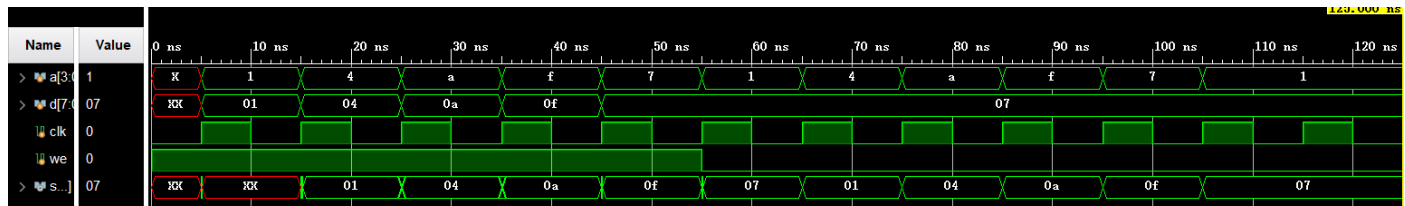
#10 a=1;

#20 $finish;

end

```

```
endmodule
```



块式 RAM:

```

module RAM1_test;

    reg clk;

    reg en;

    reg we;

    reg [3:0] a;

```

```
reg [7:0] d;  
wire [7:0] douta;  
  
RAM1 ram1 (clk, en, we, a, d, douta);
```

```
always  
    #5 clk = ~ clk;
```

```
initial  
    begin  
        #0 clk = 1'b0;  
        #0 we = 1'b1;  
        #0 en=1'b1;
```

```
        #5 a=1;  
        #0 d=1;
```

```
        #10 a=4;  
        #0 d=4;
```

```
        #10 a=10;  
        #0 d=10;
```

```
        #10 a=15;  
        #0 d=15;
```

```
        #10 a=4'b0;  
        #0 d=8'b110;
```

```
        #10 a=7;
```

```

#0 d=7;

#10 a=1;
#0 we=0;
#10 a=4;
#10 a=10;
#10 a=15;
#10 a=0;

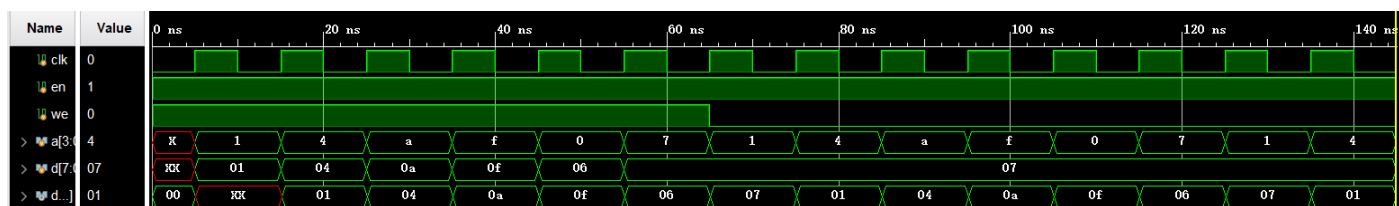
#10 a=7;
#10 a=1;
#10 a=4;

#10 $finish;

end

endmodule

```



块 RAM 和 分布式 RAM

- ① Xilinx 的 FPGA 结构主要由 CLB、IOB、IR、Block RAM 组成，其中 CLB 是最最重要的资源。
- ② 以 V5 为例，1 个 CLB 包括的 2 个 Slice，每个 Slice 包括 4 个 6 输入查找表，4 个 FlipFlop 和相关逻辑。在这里需要注意的是 Slice 分两种，SliceM 和 SliceL，它们都包括前面的东西，但是很特别的是 SliceM 还增加了基于查找表的分布式 RAM 和移位寄存器。
- ③ 每个 CLB 中都包含 SliceL，但并不是每个 CLB 中都包含 SliceM，整个一块 V5 芯片中 SliceM 和 SliceL 的比例为 1: 3。SliceM 的放置有一定的规则，这里不做阐述。
- ④ Xilinx 的 FPGA 中有 分布式 RAM 和 Block RAM 两种存储器。用分布式 RAM 时其实要用到其所在的 SliceM，所以要占用其中的逻辑资源；而 Block RAM 是单纯的存储资源，但是要一块一块的用，不像分布式 RAM 想要多少 bit 都可以。
- ⑤ 用户申请存储资源，FPGA 先提供 Block RAM，当 Block RAM 数量不够时再用分布式 RAM 补充。

FIFO:

```
module FIFO_test;

    reg clk;

    reg rst;

    reg en_in;

    reg [7:0] din;

    reg en_out;

    wire [7:0] dout;

    wire [4:0] count;

    FIFO fifo(clk, rst, en_in, din, en_out, dout, count);

    always

        #5 clk = ~ clk;

    initial

        begin

            #0 clk = 1'b0;

            #0 en_in=1'b0;

            #0 en_out=1'b0;

            #0 rst=1'b1;

            #10 rst=1'b0;

            #5 en_in=1'b1;

            #0 din=1;

            #20 en_in=1'b0;

            #20 en_in=1'b1;

            #0 din=2;

            #20 en_in=1'b0;
```

```
#20 en_in=1'b1;  
#0 din=3;  
    #20 en_in=1'b0;
```

```
#20 en_in=1'b1;  
#0 din=4;  
#20 en_in=1'b0;
```

```
#20 en_in=1'b1;  
#0 din=5;  
#20 en_in=1'b0;
```

```
#20 en_in=1'b1;  
#0 din=6;  
#20 en_in=1'b0;
```

```
#20 en_in=1'b1;  
#0 din=7;  
#20 en_in=1'b0;
```

```
#20 en_in=1'b1;  
#0 din=8;  
#20 en_in=1'b0;
```

```
#20 en_in=1'b1;  
#0 din=9;  
#20 en_in=1'b0;
```

```
#20 en_in=1'b1;  
#0 din=10;  
#20 en_in=1'b0;
```

#20 en_in=1' b1;

#0 din=11;

#20 en_in=1' b0;

#20 en_in=1' b1;

#0 din=12;

#20 en_in=1' b0;

#20 en_in=1' b1;

#0 din=13;

#20 en_in=1' b0;

#20 en_in=1' b1;

#0 din=14;

#20 en_in=1' b0;

#20 en_in=1' b1;

#0 din=15;

#20 en_in=1' b0;

#20 en_in=1' b1;

#0 din=7;

#20 en_in=1' b0;

#20 en_in=1' b1;

#0 din=8;

#20 en_in=1' b0;

#20 en_out=1' b1;

#20 en_out=1' b0;

#20 en_out=1' b1;

#20 en_out=1' b0;

#20 en_out=1' b1;

#20 en_out=1' b0;

#20 en_out=1' b1;

#20 en_out=1' b0;

#20 en_in=1' b1;

#0 din=9;

#20 en_in=1' b0;

#20 en_in=1' b1;

#0 din=11;

#20 en_in=1' b0;

#20 en_out=1' b1;

#20 en_out=1' b0;

#20 en_out=1' b1;

#20 en_out=1' b0;

#20 en_out=1' b1;

#20 en_out=1' b0;

#20 en_out=1' b1;

#20 en_out=1' b0;

#20 en_out=1' b1;

#20 en_out=1' b0;

#20 en_out=1' b1;

```
#20 en_out=1'b0;
```

```
#20 en_out=1'b1;
```

```
#20 en_out=1'b0;
```

```
#20 en_out=1'b1;
```

```
#20 en_out=1'b0;
```

```
#20 en_out=1'b1;
```

```
#20 en_out=1'b0;
```

```
#20 en_out=1'b1;
```

```
#20 en_out=1'b0;
```

```
#20 en_out=1'b1;
```

```
#20 en_out=1'b0;
```

```
#20 en_out=1'b1;
```

```
#20 en_out=1'b0;
```

```
#20 en_out=1'b1;
```

```
#20 en_out=1'b0;
```

```
#20 en_out=1'b1;
```

```
#20 en_out=1'b0;
```

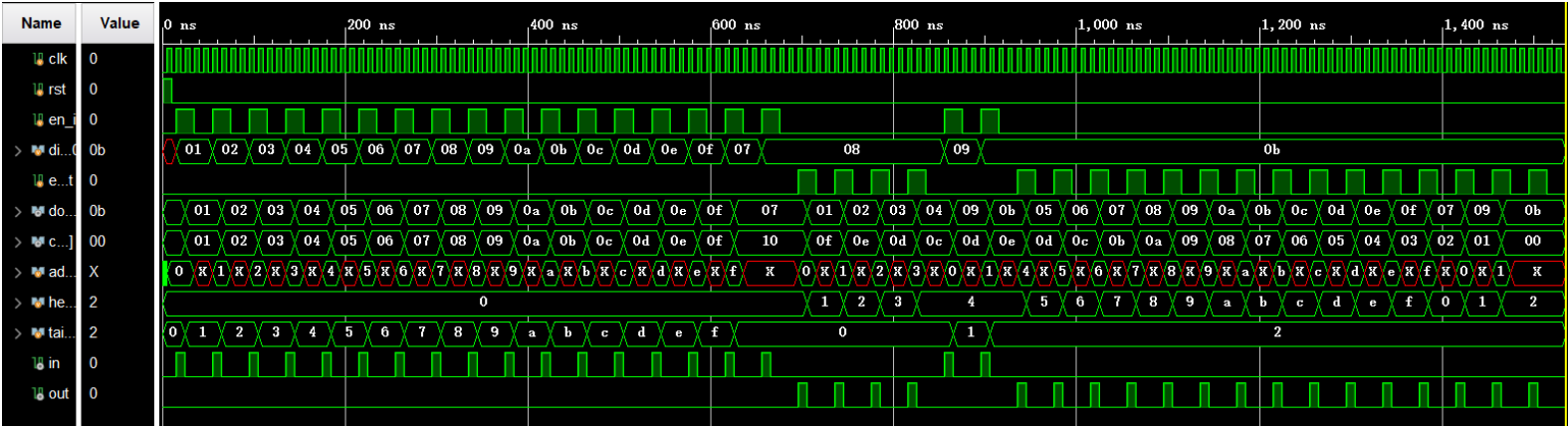
```
#20 en_out=1'b1;
```

```
#20 en_out=1'b0;
```

```
#20 $finish;
```

```
end
```

```
endmodule
```



六、 心得体会：

通过本次实验掌握了寄存器堆（Register File）和存储器（Memory）的功能、时序及其应用；熟练掌握了数据通路和控制器的设计和描述方法。