计算机组成原理 实验报告

姓名：龚小航　学号：PB18151866　实验日期：2020-4-29

一、实验题目：

Lab01 寄存器堆与队列

二、实验目的

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

设计参数化的寄存器堆，其中寄存器堆含有 32 个寄存器，寄存器宽度由参数决定，具有 2 个异步读端口和 1 个同步写端口。

用行为描述、分布式 IP 核和块式 IP 核设计容量为 16x8 位的存储器。

利用前述存储器的 IP 核设计一数据宽度为 8 位，最大长度为 16 的 FIFO 队列。

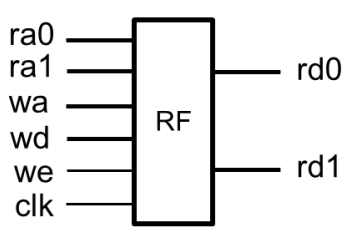
三、实验平台：

Vivado

四、实验过程：

1. 设计寄存器堆：

先画出该部分的输入输出关系，逻辑图如下所示：



设计参数化的寄存器堆，逻辑符号如图所示。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。

这部分的代码实现较为简明。只需声明一个寄存器组，包含 32 个寄存器，每个寄存去的位数由输入的参数指定。

具体实现如下所示。由于寄存器类型数据是易失性的，不需要对其进行初始化。

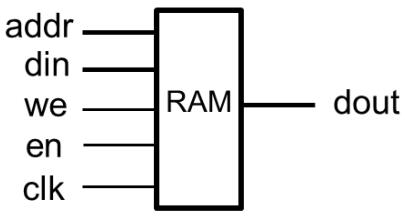
```
00 module Register_File
01  #(parameter WIDTH=32)
02   (
03    input clk,                  //clock
04    input [4:0] ra0,            //读端口0地址
05    output [WIDTH-1:0] rd0,     //读端口0数据
06    input [4:0] ra1,            //读端口1地址
07    output [WIDTH-1:0] rd1,    //读端口1数据
08    input [4:0] wa,            //写端口地址
09    input we,                  //写使能，高电平有效
10    input [WIDTH-1:0] wd       //写端口数据
11    );     //异步读 同步写
12
13    reg [WIDTH-1:0] register[31:0];
14
15     always @(posedge clk)begin
16        if(we)begin
17            register [wa] <= wd;
18        end
19    end
20
21    assign rd0 = register[ra0];
22    assign rd1 = register[ra1];
23
24
25 endmodule
```

2. 设计存储器：

存储器可以通过行为描述构建一个模块以满足需求，也可以直接例化软件提供的 IP 核直接生成。

先介绍自制的存储器，通过行为描述方法实现一个单端口 RAM。

逻辑符号如下所示：



端口说明： addr 为输入的读/写地址，din 为要写入的数据，we 为写使能，en 为芯片总使能，clk 为时钟信号， dout 为读出的数据。具体实现代码如下：

```
1  `timescale 1ns / 1ps
2  module ram_16x8                  //16x8 位单端口 RAM
3      (input clk,                  //时钟(上升沿有效)
4       input en,we,                //写使能
5       input [3:0] addr,           //操作地址
6       input [7:0] din,            //输入数据
7       output [7:0] dout           //读出数据
8      );
9      reg [3:0] addr_reg;
10     reg [7:0] mem[15:0];
11
12     //初始化 RAM 的内容
13     initial
14         $readmemb("D:/Init.txt",mem);
15
16     assign dout = mem[addr_reg];
17
18     always @(posedge clk)begin
19         if(en)begin
20             addr_reg <= addr;
21             if(we)
22                 mem[addr] <= din;
23         end
24     end
25
26 endmodule
```

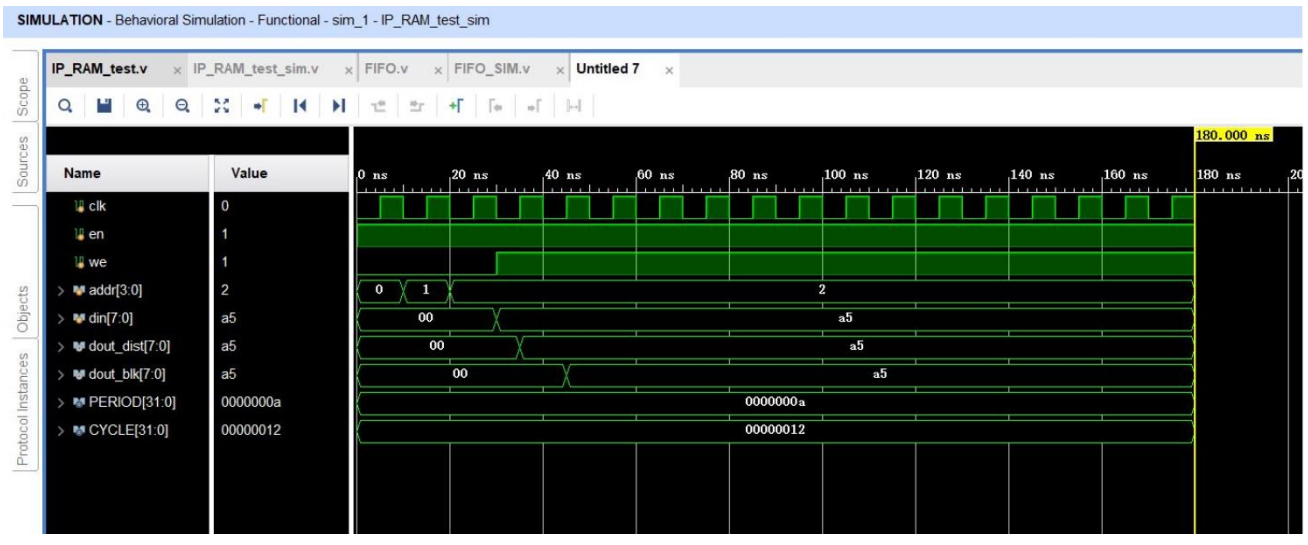
其中初始化文件放在 D 盘根目录下，是一个标准的 coe 文件。

再利用 Vivado 软件附带的 IP 核实现存储器功能。软件提供了两种类型的存储器 IP 核，分别是块 RAM（blockRam）和分布式 RAM（distributRam）。将其参数设置为 16 位深，8 位宽的存储器，再分别例化这两个 IP 核，写仿真文件对比。

测试两种 IP 核的代码如下所示：

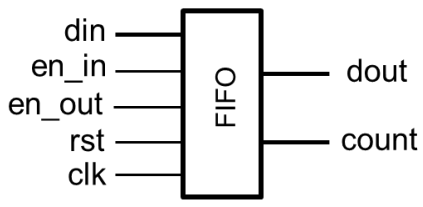
```
00 module IP_RAM_test(
01     input clk,
02     input [3:0] addr,
03     input en,
04     input we,              //写使能
05     input [7:0] din,
06     output [7:0] dout_dist,dout_blk
07     );
08
09     dist_mem_16depth_8bit dist_test
   (.a(addr), .d(din), .clk(clk), .we(we), .spo(dout_dist));
10     blk_mem_16depth_8bit  blk_test
   (.addra(addr), .clka(clk), .dina(din), .douta(dout_blk), .ena(en), .wea(we));
11
12 endmodule
```

仿真波形如下所示，仿真代码附于源码文件中。



3.设计先进先出队列（FIFO）：

队列的存储依靠的是例化的分布式 RAM。本例中数据宽度均为 8 位，队列的最大长度为 16.逻辑符号如下所示：



接口介绍：en_in：入队列使能，高电平有效；

en_out：出队列使能，高电平有效；

din： 入队列数据

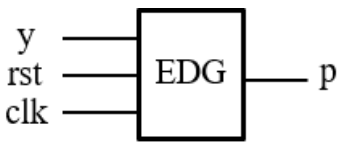
dout：出队列数据

count：队列数据计数

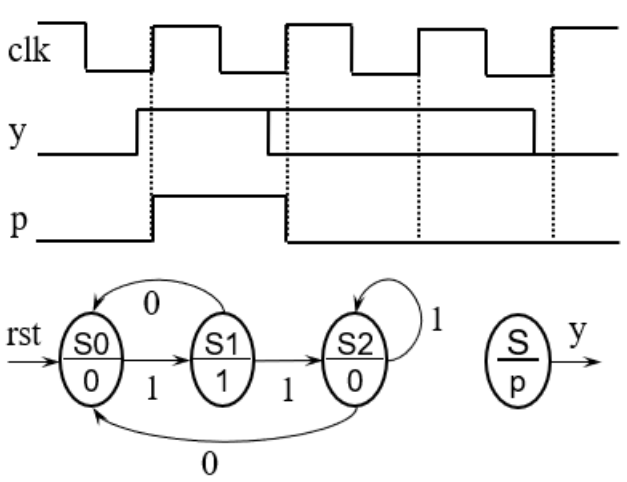
当入队列使能 en_in 有效时，将数据 din 加入队尾，当出队列使能 en_out 有效时，将队头的数据从 dout 输出，队列数据计数 count 表示队列中有效数据的个数。当 count=16 时不能入队，当 count=0 时不能出队。

在入队使能信号的一次有效持续时间， 仅允许最多入队一个数据，出队操作类似，因此需要对信号 en_in 和 en_out 取边沿。

取边沿可以通过状态机实现。逻辑符号如下所示：



输出的信号经过取边沿，有效时间只能持续一个时钟周期。实现原理可以用下图来表示：



具体实现代码如下：

```
00 module Take_edge
01  (
02    input clk,rst,
03    input  in,
04    output out
05    );
06
07  reg [1:0] current_state,next_state;
08  localparam [1:0] S0=2'b00,S1=2'b01,S2=2'b10;
09  initial current_state=S0;
10  initial next_state=S0;
11
12//state logic
13 always @(posedge clk, posedge rst)
14  if (rst) current_state <= S0;
15  else current_state <= next_state;
16
17//next state logic
18 always @(*) begin
19  next_state = current_state;
20  case (current_state)
21    S0: if (in) next_state = S1; else next_state = S0;
22    S1: if (in) next_state = S2; else next_state = S0;
23    S2: if (in) next_state = S2; else next_state = S0;
24    default: next_state = S0;
25  endcase
26 end
27
28//output logic
29  assign  out = (current_state==S1);
30
31 endmodule
```

完成该模块后，利用这个模块对入队信号、出队信号取边沿。利用有限状态机来描述 FIFO 队列，本例实现的队列利用了一个尾指针，再加上队列中现有的元素个数 count，这两个量即可完整的描述队列的状态，且具有队列的一切功能。本例实现了循环队列，初始化操作仅需将 count 置 0 而任何时候都无需关心 rear 尾指针的具体位置。实现代码如下所示：为了调式方便，将状态编码也作为模块输出。

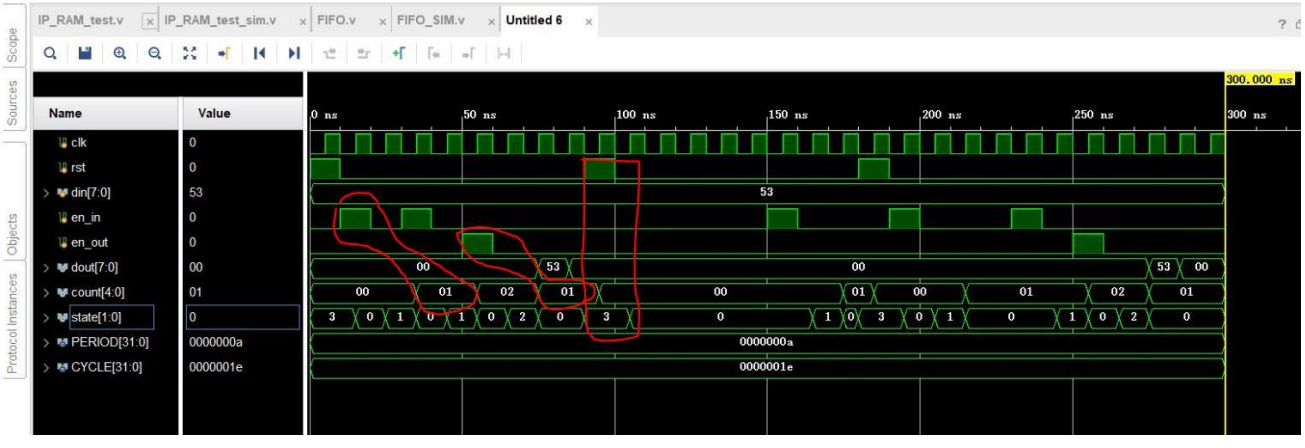
```
000 module FIFO(
001     input clk, rst,          //时钟（上升沿有效）、异步复位（高电平有效）
002     input [7:0] din,      //入队列数据
003     input en_in,             //入队列使能，高电平有效
004     input en_out,            //出队列使能，高电平有效
005     output [7:0] dout,       //出队列数据
006     output [4:0] count,      //队列数据计数
007        output [1:0] s
008     );
009
010
011   wire  en_in_edge,  en_out_edge,  mem_en;
012   wire [7:0] mem_out;
013   wire [3:0] addr;
014
015   Take_edge IN  (clk,rst,en_in,en_in_edge);
016   Take_edge OUT (clk,rst,en_out,en_out_edge);
017   dist_mem_16depth_8bit MEM
        (.a(addr), .d(din), .clk(clk), .we(mem_en), .spo(mem_out));
018
019    reg [1:0] current_state,next_state;
020   localparam [1:0] WAIT=2'b00,INQ=2'b01,OUTQ=2'b10,CLEAR=2'b11;
021    reg [3:0] rear;//记录队尾编号
022    reg [3:0] addr_reg;  reg mem_en_reg; reg [4:0] count_reg;
023    reg [7:0] dout_reg;
024  initial current_state=WAIT;
025  initial next_state=WAIT;
026  initial rear=0;
027  initial count_reg=0;
028  initial dout_reg=0;
029
030  //描述次态
031 always @(*) begin
032  if(rst) next_state=CLEAR;
033  else
034  begin
035    case (current_state)
036      WAIT:begin          //优先处理入队
037          if (en_in_edge) next_state = INQ;
038          else if(en_out_edge) next_state =  OUTQ;
039          else next_state = WAIT;
040        end
041      INQ : begin
042       if(en_out_edge) next_state = OUTQ; else  next_state = WAIT;
043        end
044      OUTQ:begin
045       if (en_in_edge) next_state = INQ;  else  next_state = WAIT;
046        end
047      default: next_state = WAIT;
048    endcase
049  end
050 end
051
052   reg x;
053  //描述输出:
054  always @(posedge clk) begin
055    case (current_state)
056      WAIT:begin
057          dout_reg<=0;
058          mem_en_reg<=0;
059          count_reg<=count_reg;
060        end
061      INQ : begin     //入队
062         dout_reg<=0;
063         if(count_reg<5'b01111) begin
064            mem_en_reg<=1;
065               if({1'b0,rear}+count_reg>5'b01111)
066                 {x,addr_reg} <= {1'b0,rear}+count_reg-5'b01111;
067               else                                          //写的位置
068                   {x,addr_reg} <= {1'b0,rear}+count_reg;
069            count_reg<=count_reg+1;
070          end
071         else  mem_en_reg<=0;
072        end
073      OUTQ: begin    //出队
074         mem_en_reg<=0;
075         if(count_reg>0) begin
076            count_reg<=count_reg-1;
077            addr_reg<=rear;
078              if(rear==4'b1111)
079                  rear<=0;
080              else rear<=rear+1;
081            dout_reg<=mem_out;
082          end
083         else ;
084        end
085      CLEAR: begin
```

```
086          dout_reg<=0;
087          count_reg<=0;
088          mem_en_reg<=0;
089        end
090      default: ;
091    endcase
092 end
093
094
095  //状态变化
096 always @(posedge clk, posedge rst)
097  if (rst) current_state <= CLEAR;
098  else current_state <= next_state;
099
100
101 assign count=count_reg;
102 assign mem_en=mem_en_reg;
103 assign dout=dout_reg;
104 assign addr=addr_reg;
105
106  assign s=current_state;
107
108 endmodule
```

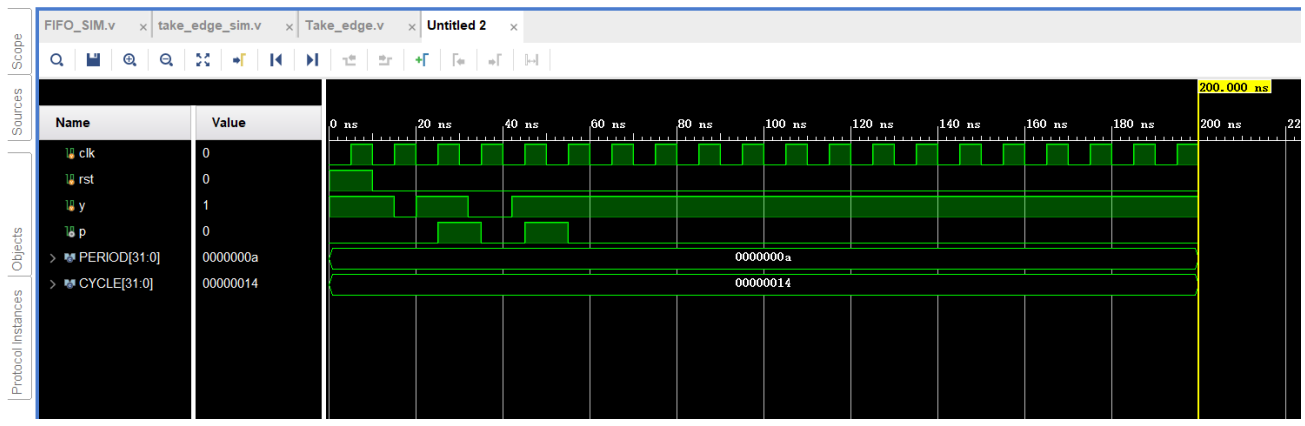
五、实验结果：

FIFO 队列仿真文件附于源码中。当 rst=1 时，将队列清空，即令count = 0.

为结果清晰，还将状态变量 state 也作为输出显示其波形，0 代表无操作，1 为入队，2 为出队，3 为清空状态。由于线路和门延时，输出比输入晚约 20ns。



其中利用了取边沿模块 take_edge ，利用状态机取入队、出队信号边沿。设计该模块的仿真代码，仿真结果如下所示：



可见输出信号高电平持续一个时钟周期，达到设计要求。

六、心得体会：

本次实验实现了寄存器堆，还实现了数据的存储模块，也介绍了了 IP 核的使用和状态机的描述方法。本次实验重点在于 FIFO 队列的描述，需要设计出状态图和数据通路，并且正确执行，较易出错。

本次实验重视 Vivado 的仿真检查。当没有实体开发板时，通过写仿真文件也可以较为准确的判断出代码情况。