中国科学技术大学计算机学院
《计算机组成原理实验》报告

实验题目：寄存器堆与队列

学生姓名：李昱祁

学生学号：PB18071496

完成日期：2020/05/07

计算机实验教学中心制

2019 年 09 月

【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。

【实验环境】

1. Vivado

2. Linux 操作系统

【实验内容】

一. 寄存器堆

1. 逻辑设计：

a. 使用 Verilog 代码定义宽度、深度均为 32 的二维数组作为寄存器；

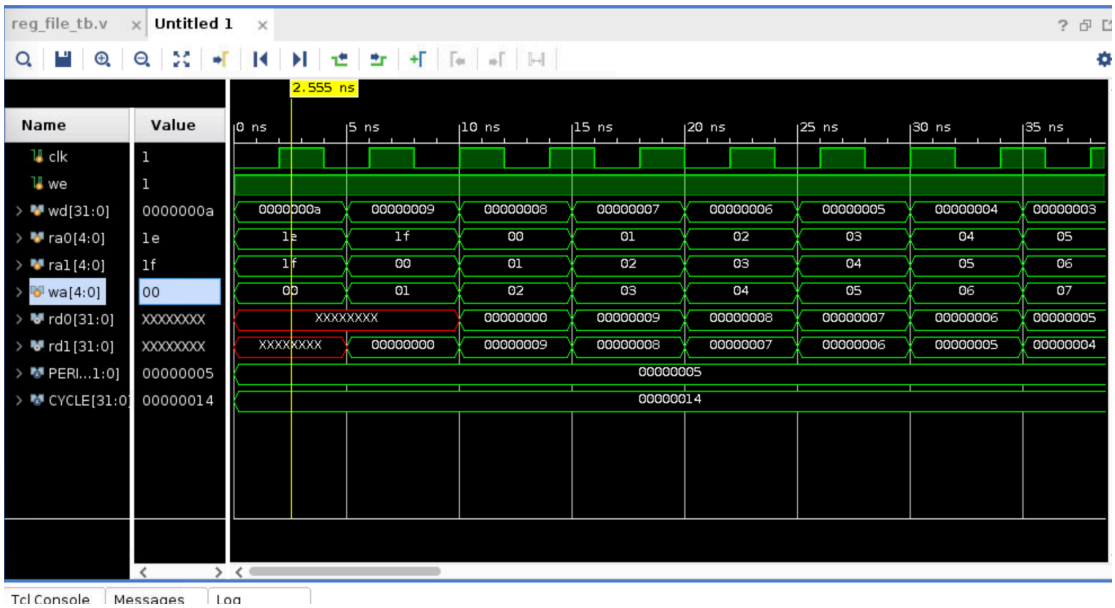
b. 异步读使用组合逻辑完成；同步写采用时序逻辑完成；0 号寄存器不能被写入非零值；

2. 核心代码：

```
module register_file
    #(  parameter WIDTH = 32,
        parameter DEEPTH = 32)
    (
        input clk,we,
        input [WIDTH - 1 : 0] wd,
        input [4 : 0] wa,
        input [4 : 0] ra0,
```

```
    input [4 : 0] ra1,

    output [WIDTH - 1 : 0] rd0,rd1
);
//32 个 32 位寄存器
reg [WIDTH - 1 : 0] r [DEEPTH - 1 : 0];
reg [WIDTH - 1 : 0] _rd0,_rd1;
assign rd0 = _rd0;
assign rd1 = _rd1;

always @(posedge clk)//同步写
begin
    if(we)
        if(wa)
            r[wa] <= wd;
        else
            r[wa] <= 0;
end

always @(*) //异步读
begin
    _rd0 = r[ra0];
    _rd1 = r[ra1];
end

endmodule
```

3. 仿真结果：



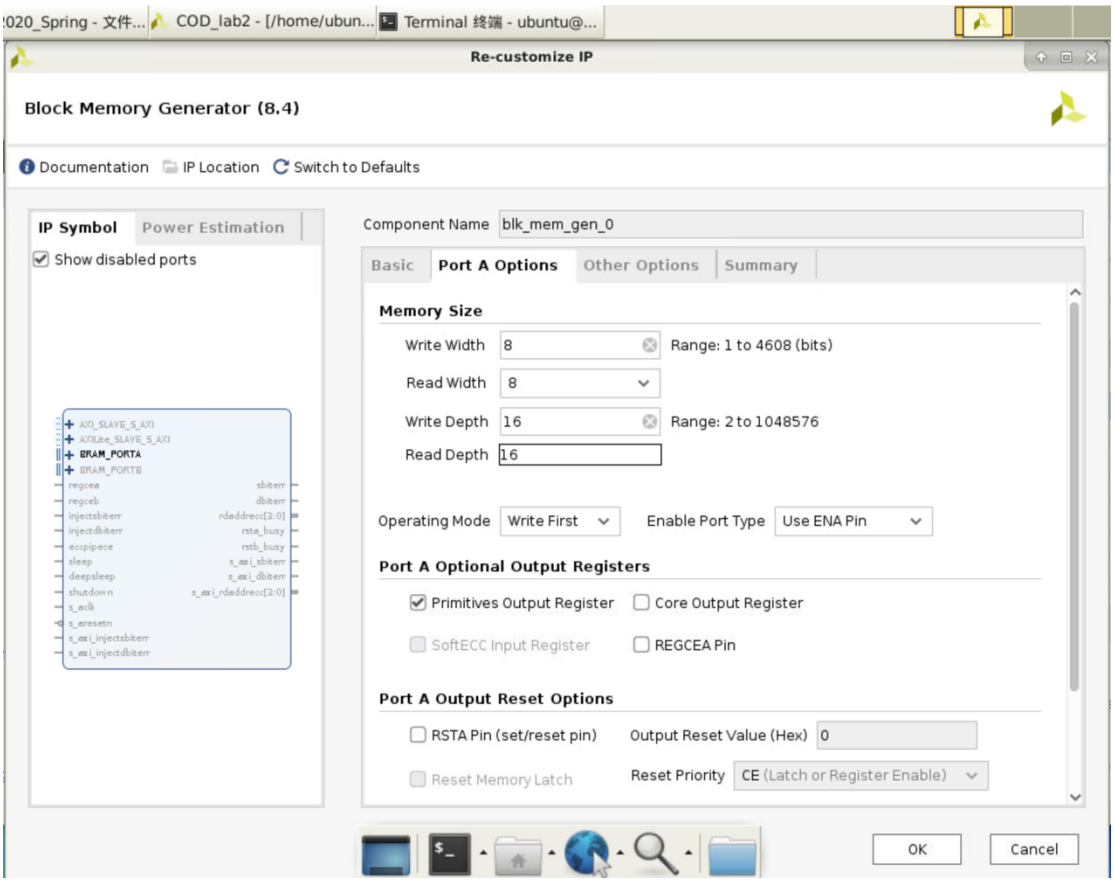
可见，寄存器堆正常进行写入和读取，并且 0 号寄存器的值始终为零，正确的完成了设计要求。

二． 分布式、块式 RAM 对比

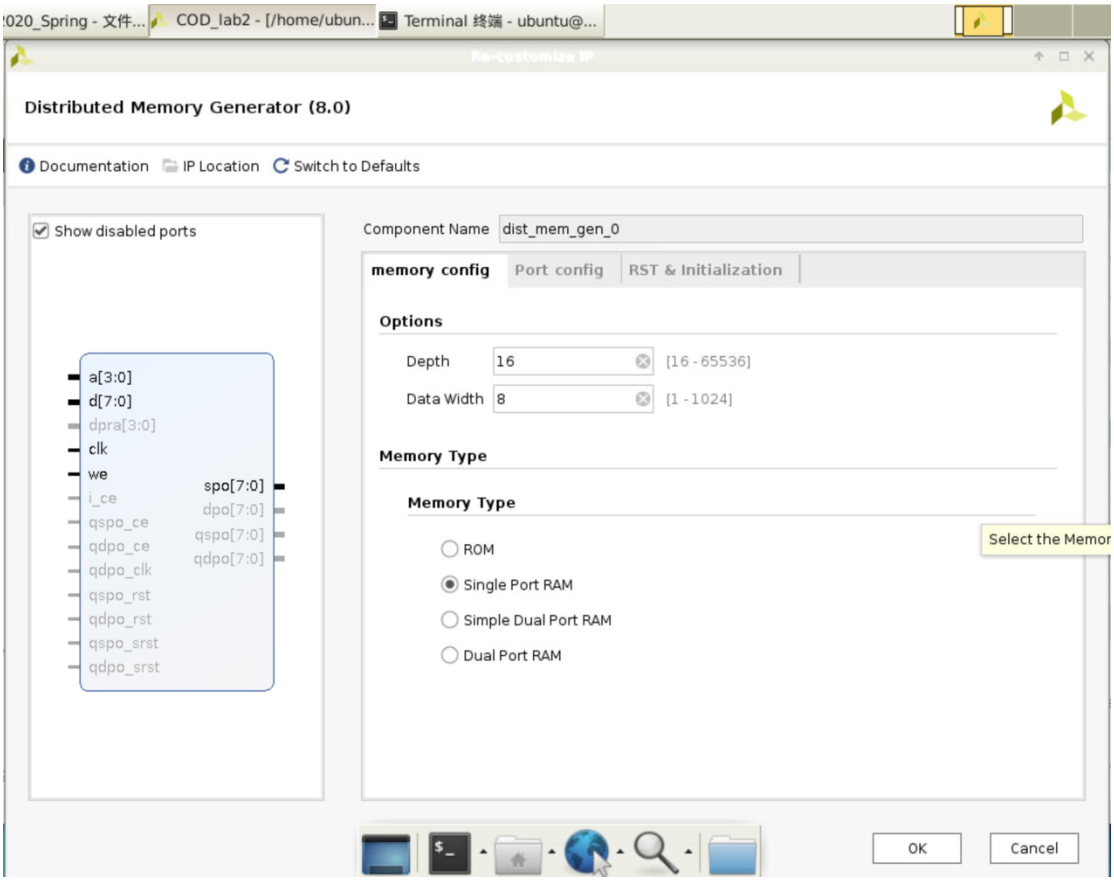
1. 逻辑设计：

例化两个 16*8 的 RAM,分别为块式、分布式：

块式：



分布式：



2. 核心代码：

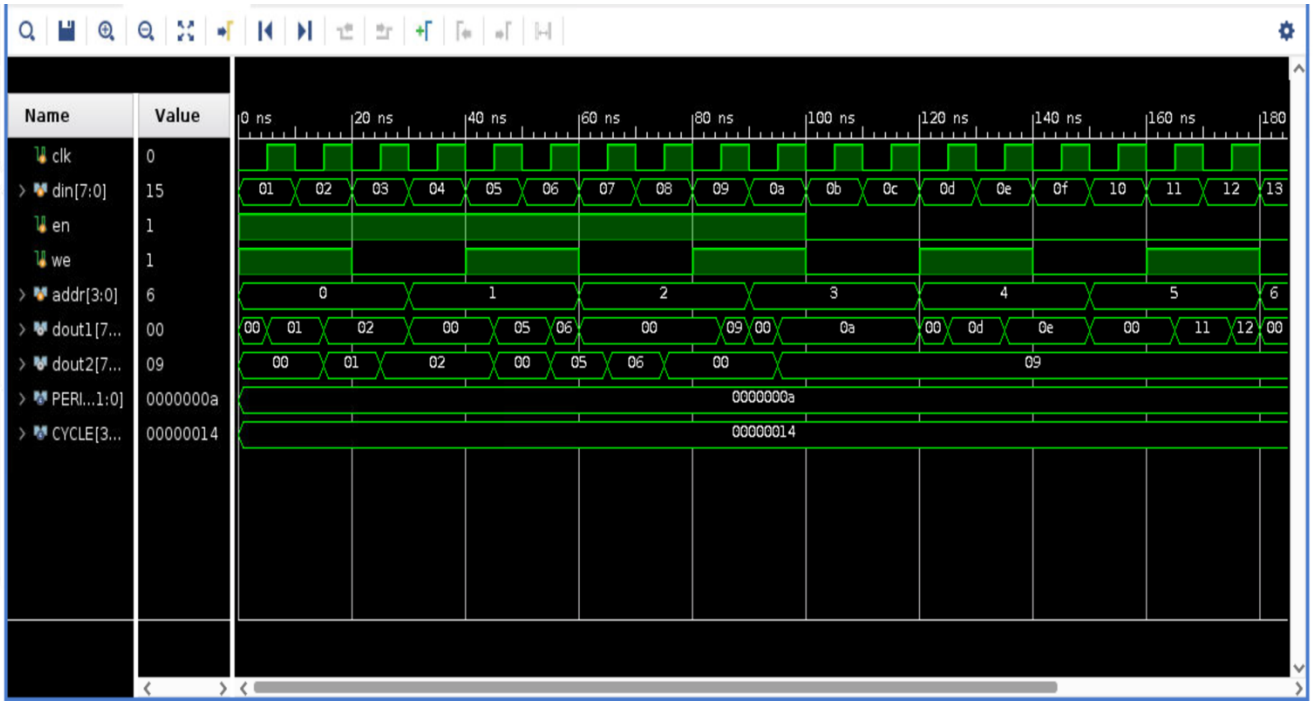
设计仿真文件，对比其不同行为特性即可。仿真文件 ram_tb.v 如下：

```
`timescale 1ns / 1ps

module ram_tb;
    reg clk;
    reg [7:0] din;
    reg en, we;
    reg [3:0] addr;
    wire [7:0] dout1, dout2;

    parameter PERIOD = 10;
    parameter CYCLE = 20;


    dist_mem_gen_0 dist_ram(addr, din ,clk, we, dout1);
    blk_mem_gen_0 blk_ram(clk, en, we, addr, din, dout2);

    initial
```

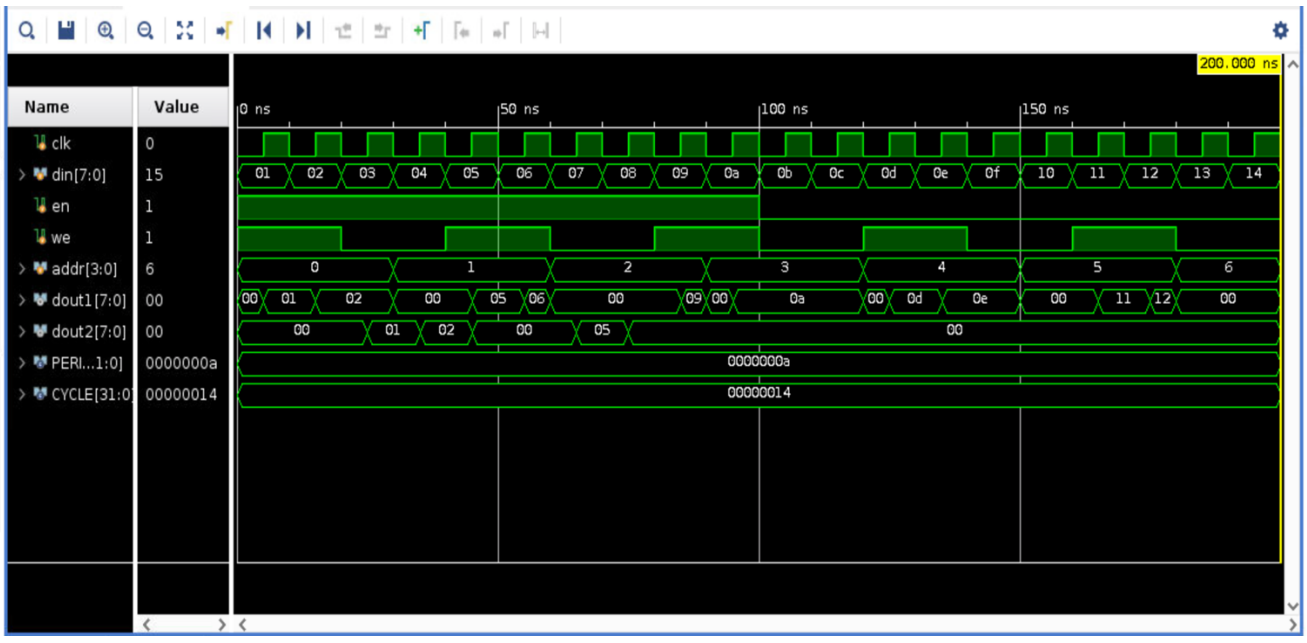
```
    begin
        clk = 0;
        repeat (2 * CYCLE)
            #(PERIOD/2) clk = ~clk;
        $finish;
    end

    initial
    begin
        we = 1;
        repeat (1 * CYCLE)
            #(PERIOD*2) we = ~we;
    end

    initial
    begin
        en = 1;
        repeat (2 * CYCLE)
            #(PERIOD*10) en = ~en;
    end

    initial
    begin
        addr = 4'b0000;
        repeat (2 * CYCLE)
            #(PERIOD*3) addr = (addr+1) % 16;
    end

    initial
    begin
        din = 8'h01;
        repeat (2 * CYCLE)
            #(PERIOD) din = (din+1) % 256;
    end

endmodule
```

3. 仿真结果：

块式 RAM 写优先 Write first：



块式 RAM 读优先 Read first：



其中 dout1 为分布式输出，dout2 为块式输出。可见：

（分布式 RAM）：读是异步的，在 addr 改变或者该地址被写入新内容后，dout 的内容会立即随之改变。在每次时钟上升沿，若 we=1，则向 RAM [addr]写入 din（可以看出 dout1 立刻输出了刚刚写入的 din）

（块式 RAM）：en=1 且 we=1 时，在第一个时钟上升沿准备好 addr 和 din，在第二个时钟上升沿向 RAM [addr]写入 din。接下来有两种情况：

Write First 时：在第二个时钟上升沿存入 din 后，dout2 立即输出存入的值 din。

Read First 时：在第二个时钟上升沿存入 din 后，dout2 先读出以前写入 RAM [addr]的内容，在在第三个时钟上升沿才输出 din。

三. Fifo 队列

1. 逻辑设计：

a. 使用取边沿模块(模块名“edg”)取得入队、出队信号边沿

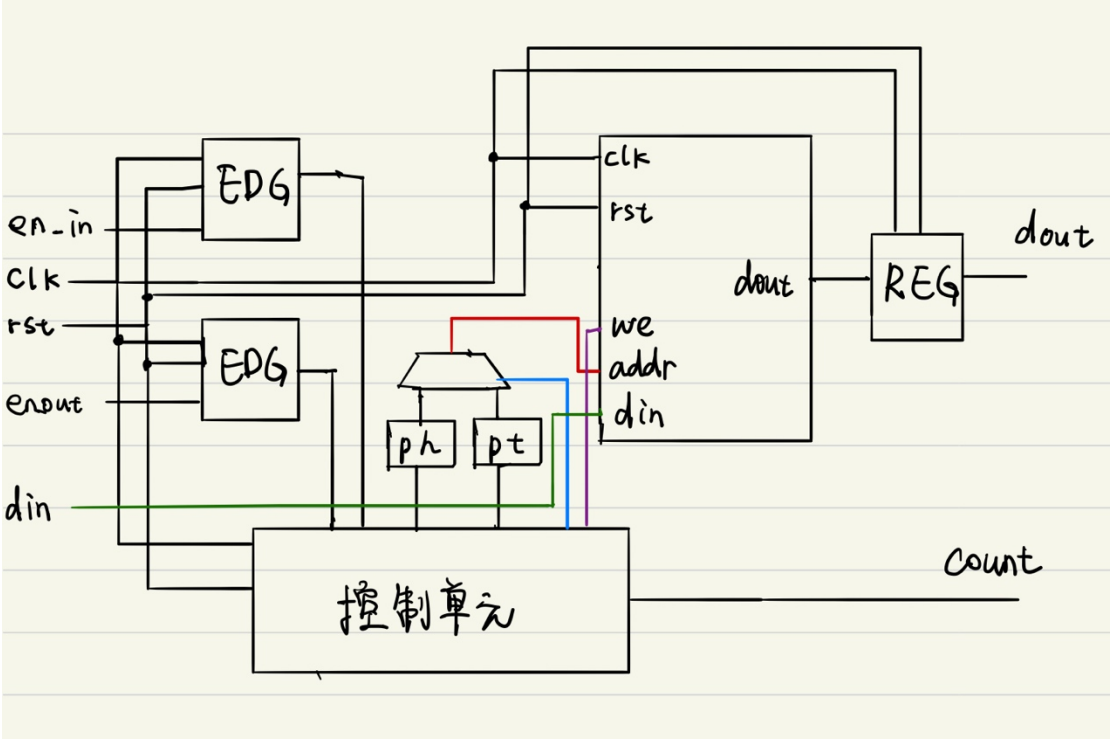
b. 根据获得的信号边缘判定状态，再有状态决定该周期完成什么任务：出队、入队、保持状态等

c. 维护头指针与尾指针，分别用于出队、入队；使用时，将要用的指针赋给 RAM 读入的地址；赋值完成后，使用的指针都会自增 1

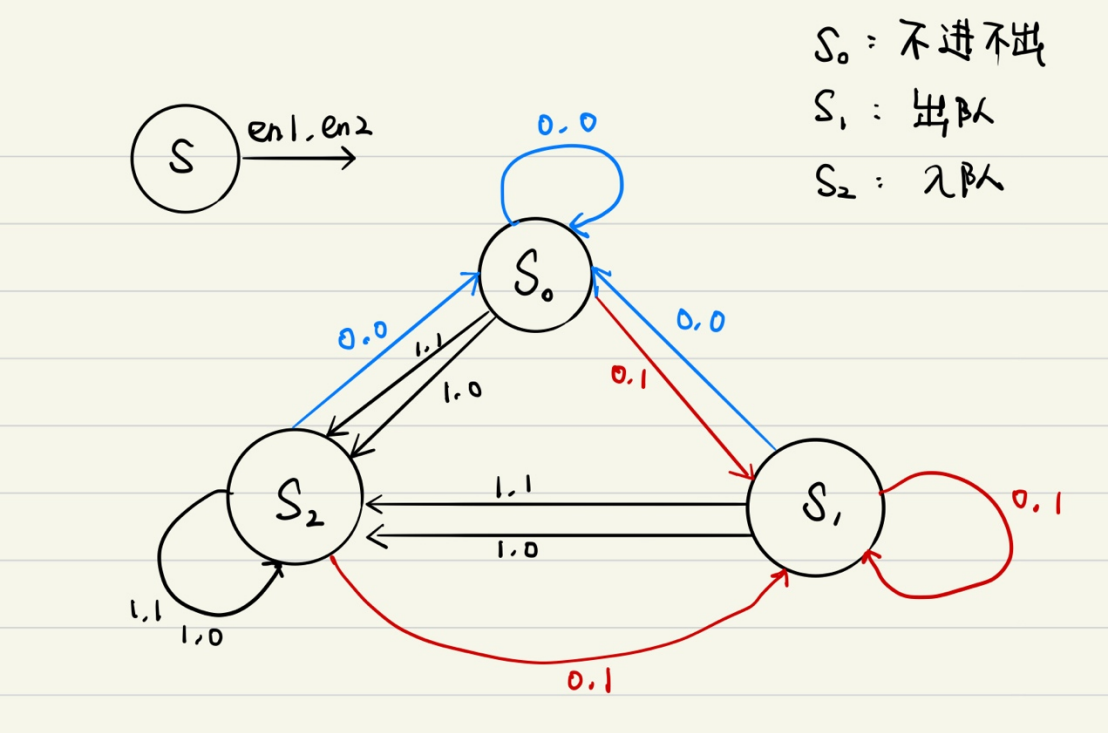
d. 寄存器变量 count 用于记录队列中现存的数据个数，入队时加 1，出队时减 1；根据 count 是否为 0 或 16 来判断队列是否空、满；队空信号无效(为 0)时才能出队，队满信号无效时才能入队

e. 在 RAM 的输出端接一个受使能信号控制的寄存器，只有在出队完成时才能让寄存器中的值更新为 RAM 的输出。这样，dout 的值总是上一个出队的值。

数据通路如下：



状态转换图如下：



（en1、en2 信号已经考虑了队满、队空的情况）

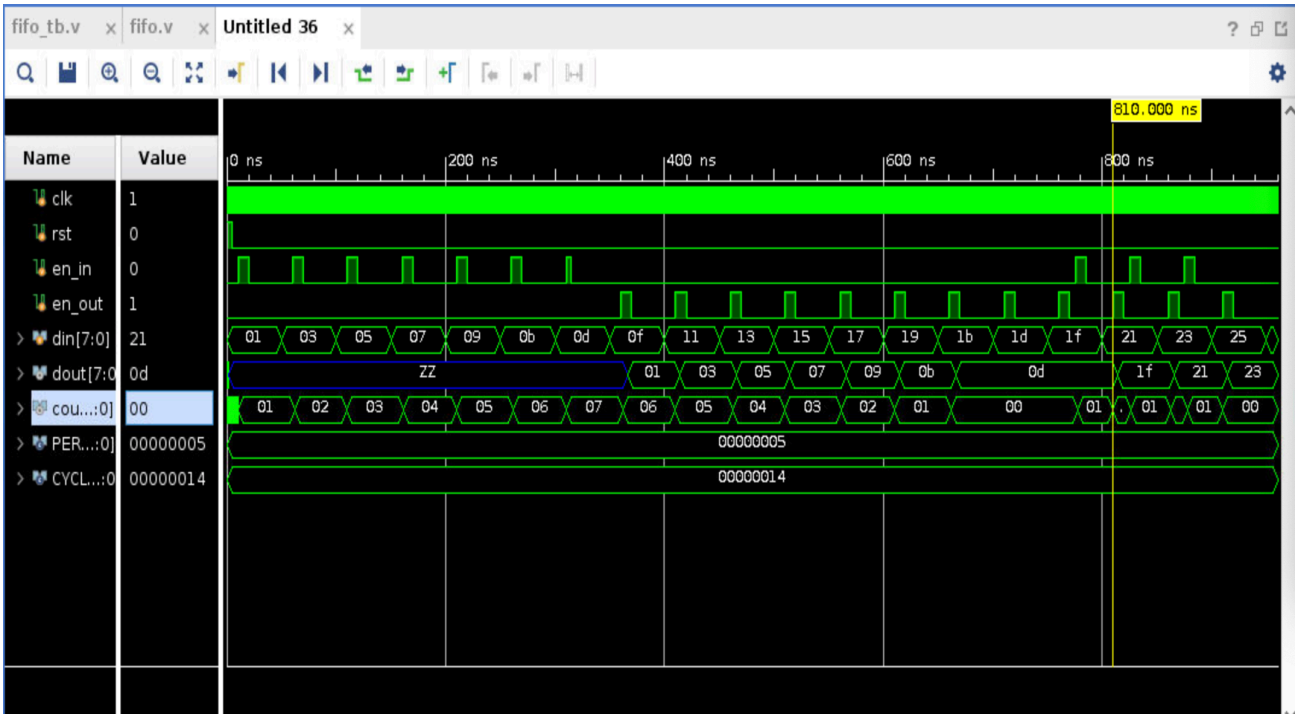
2. 核心代码：

```
module ptph
    (
        input clk, rst,
        input [7:0] din,
        input en_in,
        input en_out,

        output  [7:0] dout,
        output reg [4:0] count
    );

//队空、队满信号
wire full;
wire empty;

//去边沿得到的入队、出队使能信号
wire edg_in;
wire edg_out;

//加入队空队满判断后的使能信号
wire en1,en2;

//头尾指针与 RAM 端口的地址
```

```
reg [3:0]pt;   //pointer of tail
reg [3:0]ph;   //pointer of head
reg [3:0]addr;


assign full = (count == 5'b10000);
assign empty = (count == 5'b00000);

assign en1 = (edg_in & ~full);
assign en2 = (edg_out & ~empty);

reg [1:0] state;
wire [7:0] _dout;   //RAM 的输出

dist_mem_gen_0  mem(.a(addr),.d(din),.clk(clk),.we(edg_in),.spo(_dout));

edg edg_en_in(.y(en_in),.rst0(rst),.clk0(clk),.q(edg_in));
edg edg_en_out(.y(en_out),.rst0(rst),.clk0(clk),.q(edg_out));

//寄存器用来保存上一个出队的值
register R1(_dout,edg_out,rst,clk,dout);


always @(posedge clk)
begin
    if(rst)
    begin
        pt = 4'b0000;
        ph = 4'b0000;
        count = 5'b00000;
    end
end

always @(*)
    state = {en1,en2};

always @(state)
begin
    case(state)
        2'b00:
        begin
            count = count;
        end
        2'b01:
        begin
```

```
            addr = ph;
            count = (count - 1);
            ph = (ph + 1);
        end
        2'b10,2'b11://两信号都有效时，屏蔽出队信号
        begin
            addr = pt;
            count = count + 1;
            pt = (pt + 1);
        end
    endcase
end

endmodule
```

3. 仿真结果：



由仿真文件 fifo_tb.v 控制，先入队 7 次，之后出队 7 次，最后交替出入队；

可以看出：

a. 输出顺序即为输入顺序，满足了队列先入先出的要求

b. 队空时无法出队；

c. dout 结果一直存储为上一个出队的值(如 700ns 时的值“0d”)；

复位时 dout 的值被赋为高阻态

由仿真结果，可以验证设计的 fifo 电路在行为级上的正确性

【思考题】

1. 如何利用寄存器堆和适当电路设计实现可变个数的数据排序电路？

答：寄存器堆具有异步读、同步写的特点，可将其改为一个 fifo 队列，之后添加适当的电路，使其具有可变个数数据排序功能：

大致思路如下：

设有 n 个元素需要排序(n >= 2)：

首先前 2 个元素进行比较，之后按顺序入队

第 k 个元素到来时：

i. 队列里的元素按顺序出队；

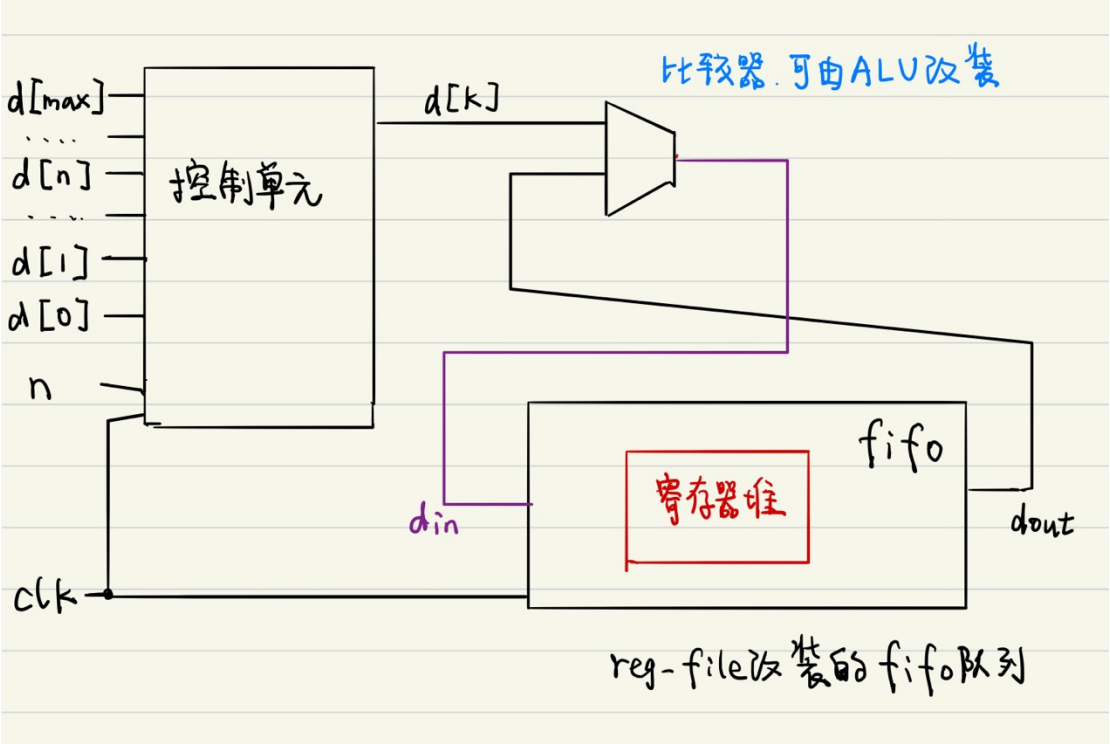
ii. 若出队元素小于等于第 k 个元素，则直接入队

iii. 当出队的第 i 个元素大于第 k 个元素时，第 k 个元素入队,之后再将第 k 个元素入队。后续元素出队后直接入队；之后，开始进行第 k+1 个元素到来时的操作

iv. 若队列里现有元素均小于等于第 k 个元素，则将第 k 个元素入队至队尾；之后，开始进行第 k+1 个元素到来时的操作

当第 n 个元素的插入完成后，此时该队列的出队顺序即为排序后的 n 个元素递增顺序。

电路图（数据通路）简要描述如下：



【实验总结】

1. 了解了块式 RAM 和分布式 RAM 之间的区别，并且通过仿真进一步熟悉其各自的特性
2. 回顾了取边沿电路的写法；学会了使用 FSM 来获取信号边沿
3. 通过 fifo 队列的设计，熟悉了控制器和数据通路的描述

改进建议：

无