# COD LAB2 实验报告

姓名：刘芷辰

学号：PB21111728

日期：2023.4.11

## 1. 实验题目

寄存器堆与储存器及其应用

## 2. 实验目标

- 掌握寄存器堆功能、时序及其应用
- 掌握存储器的功能、时序
- 熟练掌握数据通路和控制器的设计和描述方法

## 3. 实验内容

### 3.1 PART 1（寄存器堆设计与仿真）

> 逻辑设计

- 寄存器堆

```
`timescale 1ns / 1ps
// 32 * WIDTH Register File
module register_file #(parameter WIDTH = 32)
(
input clk,
input [4:0] ra0,                              // 读端口0地址
input [4:0] ra1,                              // 读端口1地址
input [4:0] wa,                               // 写端口地址
input we,                                     // 写使能（pos）
input [WIDTH-1:0] wd,                         // 写端口数据
output [WIDTH-1:0] rd0,                       // 读端口0数据
output [WIDTH-1:0] rd1                        // 读端口1数据
```

```verilog
);
reg [WIDTH-1:0] regfile [0:31];
assign rd0 = regfile[ra0];
assign rd1 = regfile[ra1];


always @(posedge clk) begin
    if(wa==0)   regfile[0]<=0;
    else if (we) regfile[wa] <= wd;
end
endmodule
```

为了实现寄存器堆x0为0的功能，控制当wa为0时，写入的数据恒为0

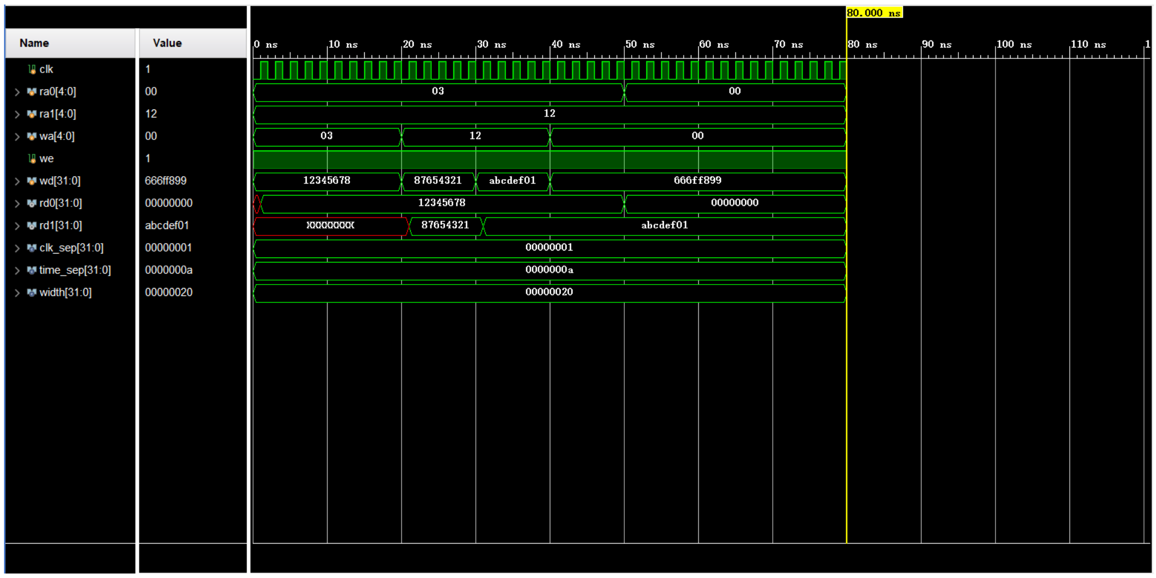
## 寄存器堆功能仿真

- 仿真文件

```verilog
`timescale 1ns / 1ps
module testbench();
parameter clk_sep = 1;
parameter time_sep = 10;
parameter width = 32;
reg clk;
reg [4:0] ra0;
reg [4:0] ra1;
reg [4:0] wa;
reg we;
reg [width-1:0] wd;
wire [width-1:0] rd0;
wire [width-1:0] rd1;
register_file regfile(
.clk(clk),
.ra0(ra0),
.ra1(ra1),
.wa(wa),
.we(we),
.wd(wd),
.rd0(rd0),
.rd1(rd1)
);
initial begin
clk = 0;
ra0 = 5'h03;
```

```
ra1 = 5'h12;
forever #clk_sep clk = ~clk;
end
initial begin
we = 1'b1;
wa = 5'h03;
wd = 32'h12345678;
#time_sep
we = 1'b1;
#time_sep
wa = 5'h12;
wd = 32'h87654321;
#time_sep
we = 1'b1;
wd = 32'habcdef01;
#time_sep
we = 1'b1;
wa = 5'h0;
wd = 32'h666ff899;
#time_sep
we = 1'b1;
ra0 = 0;
#time_sep
we = 1'b1;
wa = 5'h0;
#time_sep
we = 1'b1;
#time_sep
$finish;
end
endmodule
```

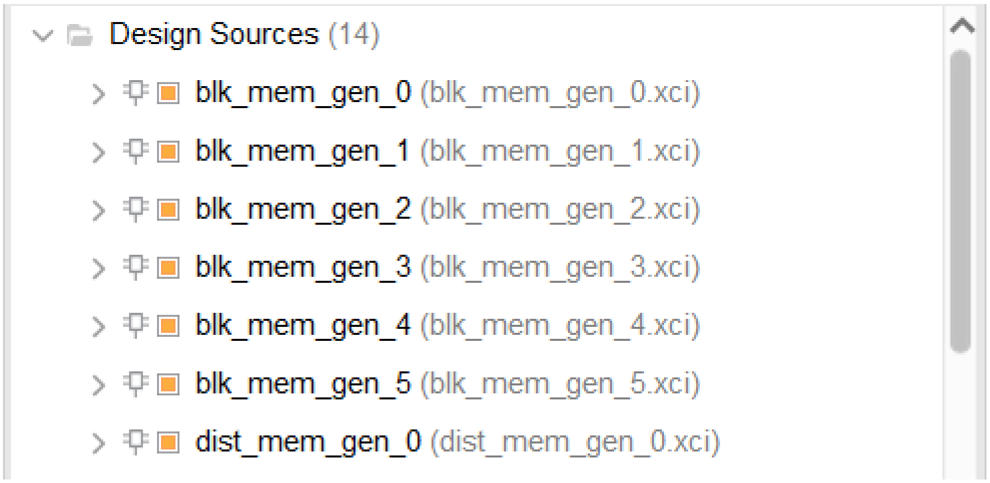
- 仿真波形



可以看出，在ra和wa为0时，ra显示的数据为0，并不是wd的写入数据，实现了x0为0的功能

## 3.2 PART 2（存储器IP核例化及仿真）

IP核例化（含选做）



Design Sources (14)
- blk_mem_gen_0 (blk_mem_gen_0.xci)
- blk_mem_gen_1 (blk_mem_gen_1.xci)
- blk_mem_gen_2 (blk_mem_gen_2.xci)
- blk_mem_gen_3 (blk_mem_gen_3.xci)
- blk_mem_gen_4 (blk_mem_gen_4.xci)
- blk_mem_gen_5 (blk_mem_gen_5.xci)
- dist_mem_gen_0 (dist_mem_gen_0.xci)

含选做部分一共七个IP核例化，从上到下分别为

| IP核 | 参数设置 |
|---|---|
| blk_men_gen_0 | 块式、write first、always enabled、non register |
| blk_men_gen_1 | 块式、read first、always enabled、non register |
| blk_men_gen_2 | 块式、no change、always enabled、non register |
| blk_men_gen_3 | 块式、write first、always enabled、primitives output register |
| blk_men_gen_4 | 块式、write first、always enabled、core output register |
| blk_men_gen_5 | 块式、write first、always enabled、two output register |
| dist_men_gen_0 | 分布式、non register |

## 功能仿真

- 测试文件

```
`timescale 1ns / 1ps
module tb();
reg clk;
reg [3:0] addr;
reg [7:0] in;
reg we;
initial begin
clk <= 1'b0;
forever
#1 clk <= ~clk;
end
initial begin
addr <= 4'h0;
in <= 8'h00;
we <= 1'b0;
#10 addr <= 4'h1;
#10 addr <= 4'h2;
#10 addr <= 4'h3;
#10 addr <= 4'h4;
#10 addr <= 4'h5;
#10 addr <= 4'h6;
#10 addr <= 4'h7;
#10 addr <= 4'h8;
#10 addr <= 4'h9;
```

```
#10 addr <= 4'hA;
#10 addr <= 4'hB;
#10 addr <= 4'hC;
#10 addr <= 4'hD;
#10 addr <= 4'hE;
#10 addr <= 4'hF;
#10 addr <= 4'h0;
in <= 8'h00;
we <= 1'b1;
#10 addr <= 4'h1;
in <= 8'h11;
#10 addr <= 4'h2;
in <= 8'h22;
#10 addr <= 4'h3;
in <= 8'h33;
#10 addr <= 4'h4;
in <= 8'h44;
#10 addr <= 4'h5;
in <= 8'h55;
#10 addr <= 4'h6;
in <= 8'h66;
#10 addr <= 4'h7;
in <= 8'h77;
#10 addr <= 4'h8;
in <= 8'h88;
#10 addr <= 4'h9;
in <= 8'h99;
#10 addr <= 4'hA;
in <= 8'hAA;
#10 addr <= 4'hB;
in <= 8'hBB;
#10 addr <= 4'hC;
in <= 8'hCC;
#10 addr <= 4'hD;
in <= 8'hDD;
#10 addr <= 4'hE;
in <= 8'hEE;
#10 addr <= 4'hF;
in <= 8'hFF;
#10 addr <= 4'h0;
we <= 1'b0;
#10 addr <= 4'h1;
#10 addr <= 4'h2;
#10 addr <= 4'h3;
#10 addr <= 4'h4;
```

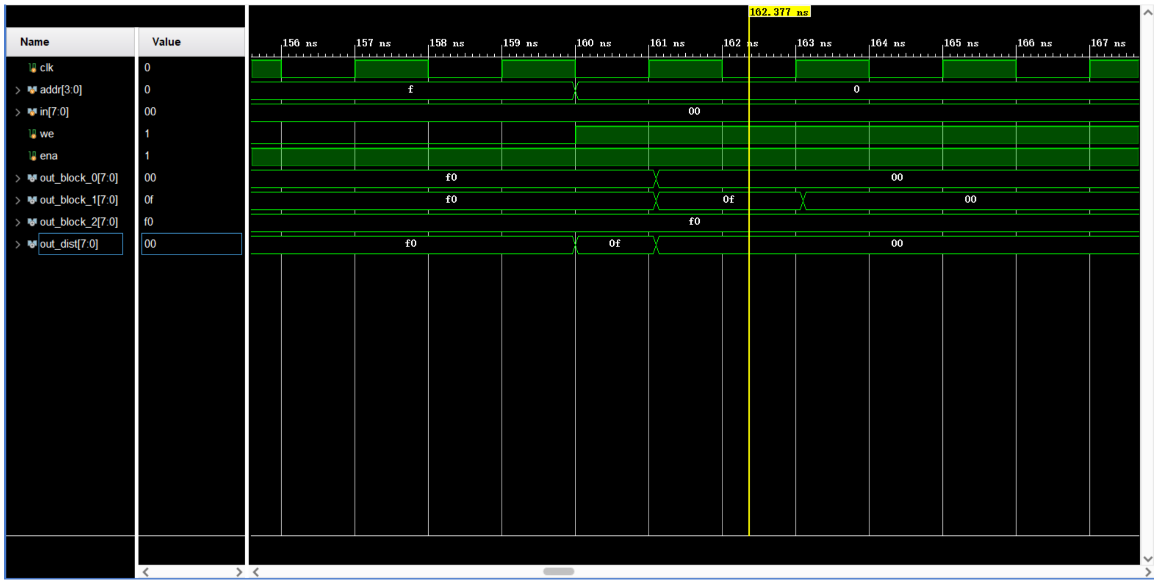
```
#10 addr <= 4'h5;
#10 addr <= 4'h6;
#10 addr <= 4'h7;
#10 addr <= 4'h8;
#10 addr <= 4'h9;
#10 addr <= 4'hA;
#10 addr <= 4'hB;
#10 addr <= 4'hC;
#10 addr <= 4'hD;
#10 addr <= 4'hE;
#10 addr <= 4'hF;
#10 $finish;
end
// block memory
reg ena;
wire [7:0] out_block_0;
wire [7:0] out_block_1;
wire [7:0] out_block_2;
initial begin
ena <= 1'b1;
#330 ena <= 1'b0;
forever
#5 ena <= ~ena;
end
blk_mem_gen_0 test_block_0(
.clka(clk),
.addra(addr),
.dina(in),
.douta(out_block_0),
.wea(we)
);
blk_mem_gen_1 test_block_1(
.clka(clk),
.addra(addr),
.dina(in),
.douta(out_block_1),
.wea(we)
);
blk_mem_gen_2 test_block_2(
.clka(clk),
.addra(addr),
.dina(in),
.douta(out_block_2),
.wea(we)
);
```

```
// distributed memory
wire [7:0] out_dist;
dist_mem_gen_0 test_dist(
.clk(clk),
.a(addr),
.d(in),
.we(we),
.spo(out_dist)
);

endmodule
```

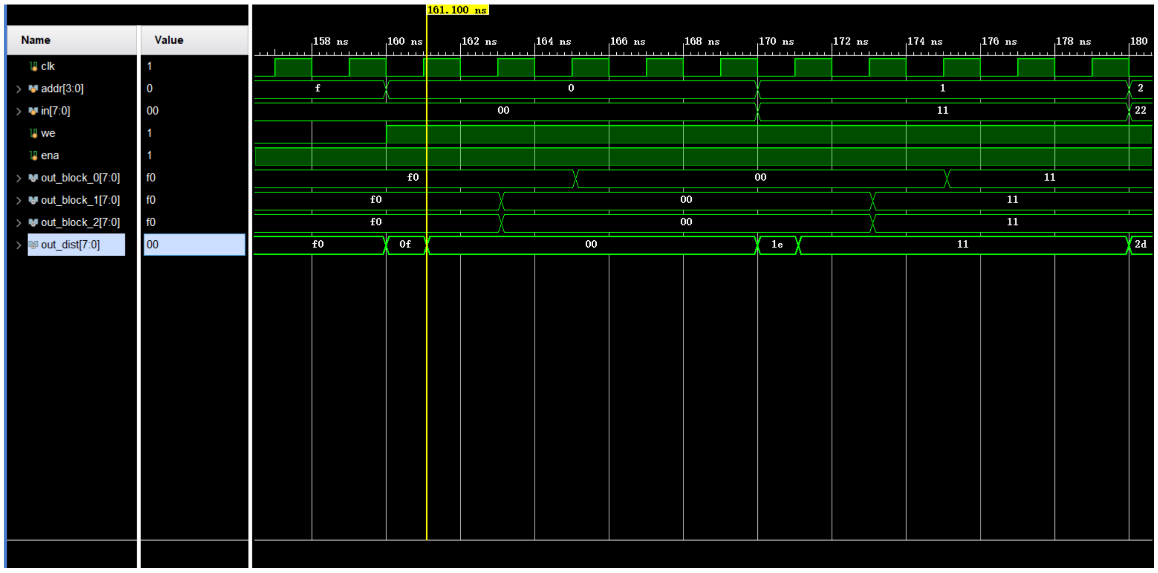
- 仿真结果

  必做部分：



可以看出块式比分布式时序输出延后半个周期，这是因为块式是同步读端口，分布式是异步读端口，而写优先直接显示了写入的数据，读优先先显示了之前的初始数据，再显示了写入的数据，no change一旦写有效则读无效，因此没有显示出写入的数据，直到we变为低电平

选做部分：（对仿真代码实例化部分修改例化模块即可）

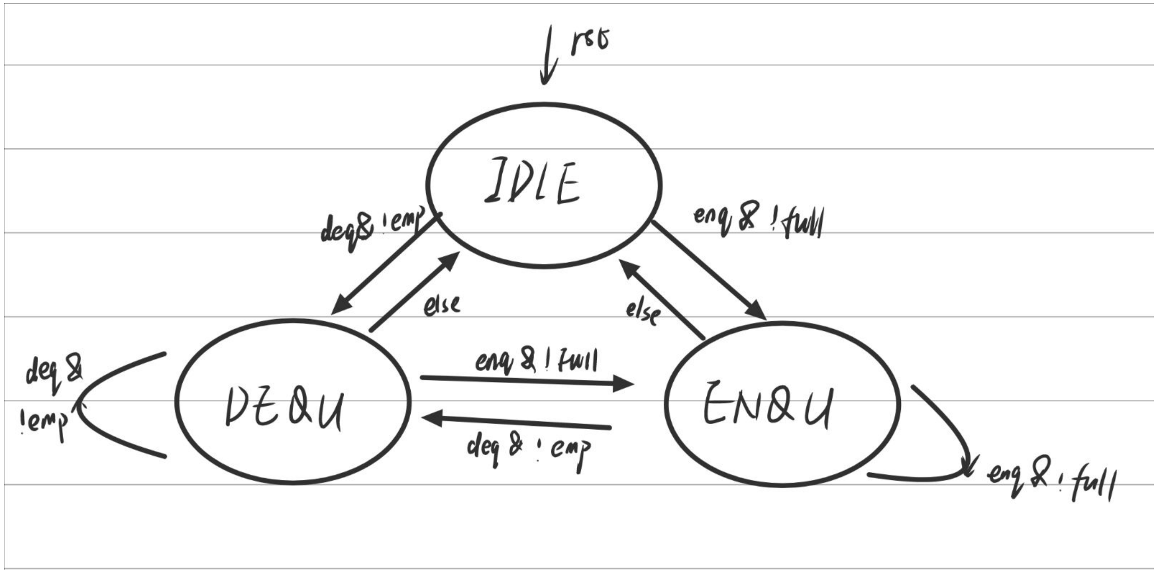


可以看出块式比分布式时序输出延后，而同时勾选两个寄存器比只勾选一个寄存器还延后一个周期，这是因为通过了两个寄存器进行输出

## 3.3 PART 3（寄存器堆的应用：FIFO队列）

LCU模块设计

1. FSM设计
    - 状态机：



    - 代码：

```
module FSM(
  input clk,              //时钟信号
  input enq,              //入队标志
  input deq,              //出队标志
  input rst,              //同步复位(高电平有效)
  input full,             //队满标志
  input emp,              //队空标志
  output reg [2:0] state      //状态
);


parameter IDLE = 2'b00;//保持现状
parameter ENQU = 2'b01;//入队状态
parameter DEQU = 2'b10;//出队状态
// parameter IDLE = 2'b11;

reg [2:0] current_state;
reg [2:0] next_state;

//描述状态切换
always @(posedge clk) begin
  if(rst) begin
    current_state <= IDLE;
  end
  else begin
    current_state <= next_state;
  end
  end


//判断状态转移条件，描述状态转移规律
always @(*) begin
  case (current_state)
    IDLE:
    if(enq & ~full) begin
      next_state = ENQU;
    end
    else if(deq & ~emp) begin
      next_state = DEQU;
    end
    else begin
      next_state = IDLE;
    end

    ENQU:
     if(enq & ~full) begin
      next_state = ENQU;
    end
```

```
      else if(deq & ~emp) begin
        next_state = DEQU;
      end
      else begin
        next_state = IDLE;
      end

      DEQU:
       if(enq & ~full) begin
        next_state = ENQU;
      end
      else if(deq & ~emp) begin
        next_state = DEQU;
      end
      else begin
        next_state = IDLE;
      end


      default:
        next_state = IDLE;
    endcase
    end

  //描述状态输出
  always @(posedge clk) begin
    state = current_state;
  end

  endmodule
```

**2.** LCU模块（调用FSM）

```
module list_control_unit(
    input                   clk,

    input                   rst,
    input           [3:0]   in,             // 入队数据
    input                   enq,            // 入队边缘
    input                   deq,            // 出队边缘
    input           [3:0]   rd,             // 写端口数据
    output                  full,
    output                  emp,            // 队列空
    output reg      [3:0]   out,            // 出队数据
    output          [2:0]   ra,             // 读端口地址
    output                  we,             // 写使能
```

```verilog
    output       [2:0]  wa,          // 写端口地址
    output       [3:0]  wd,          // 写端口数据
    output reg   [7:0]  valid        // 数据有效
);
    reg [2:0] head;                 // 头指针
    reg [2:0] tail;                 // 尾指针
    wire [1:0] state;               // 状态

    assign full = &valid;           // 当标志的每一位都为1时,说明队列已满
    assign emp  = ~(|valid);        // 当标志的每一位都为0时,说明队列为空

    assign ra = head;               // 出队,读端口地址等于队头
    assign we = enq & ~full & ~rst;
    assign wa = tail;               // 入队，写端口地址等于队尾
    assign wd = in;

    //状态产生
    FSM fsm(
      .clk(clk),
      .enq(enq),
      .deq(deq),
      .rst(rst),
      .full(full),
      .emp(emp),
      .state(state)
    );

    always @(posedge clk) begin
        if(rst) begin
            valid <= 8'h0;
            head  <= 3'h0;
            tail  <= 3'h0;
            out   <= 3'h0;
        end
        else if (state == 2'b00) begin
            valid <= valid;
            head  <= head;
            tail  <= tail;
            out   <= out;
        end

        else if(state == 2'b01) begin
            valid[tail] <= 1'b1;
            tail        <= tail + 3'h1;
        end

        else if(state == 2'b10) begin
            valid[head] <= 1'b0;
            head        <= head + 3'h1;
            out         <= rd;
        end

    end

endmodule
```

SDU模块设计

在显示单元中, 输入的时钟信号是 100MHz 的, 对于数码管来说频率过快, 因此这里对其降频到 400Hz.

使用一个 18 位的模 250000 计数器, 在每个 100MHz 的时钟上升沿进行计数, 并在计数器值大于等于 249999)时对输出到寄存器堆的ra0进行加一, 实现对寄存器堆400Hz 的扫描.

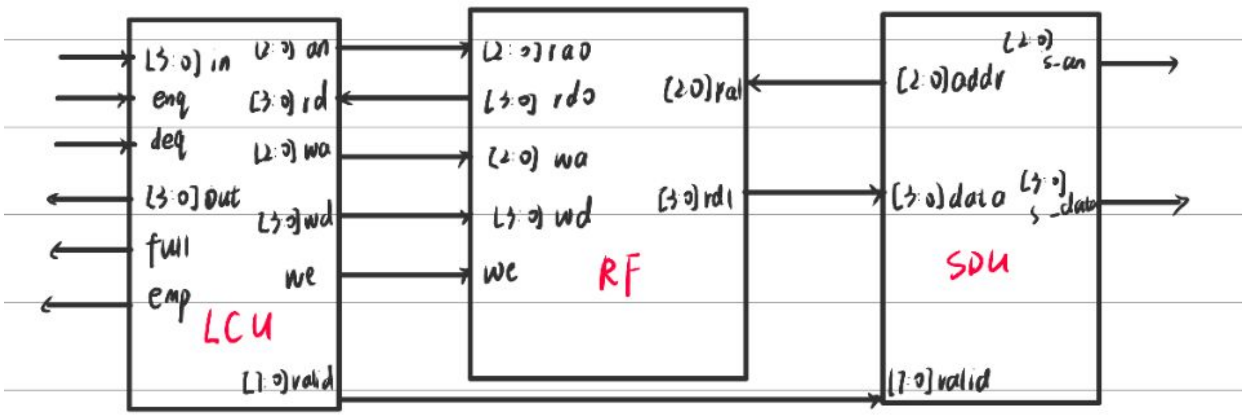
```verilog
`timescale 1ns / 1ps
module segplay_unit(
input clk_100mhz,
input [3:0] data,
input [7:0] valid,
output reg [2:0] addr,
output [2:0] segplay_an,
output [3:0] segplay_data
);
// 降频到400Hz（250000倍）
wire clk_400hz;
reg [17:0] clk_cnt;
assign clk_400hz = ~(|clk_cnt); // 每满250000翻转时钟
always @(posedge clk_100mhz) begin
if (clk_cnt >= 18'h3D08F) begin // clk_cnt >= 249999
clk_cnt <= 18'h00000;
addr <= addr + 3'b001;//取遍八位，将有数据的输出
end else
clk_cnt <= clk_cnt + 18'h00001;
end

reg [2:0] segplay_an_reg;
reg [3:0] segplay_data_reg;
always @(posedge clk_100mhz) begin
```

```
if (clk_400hz && valid[addr]) begin //有数据在addr，则数码管展示出来
segplay_an_reg <= addr;
segplay_data_reg <= data;
end
end
assign segplay_data = (|valid) ? segplay_data_reg : 4'h0;
assign segplay_an = (|valid) ? segplay_an_reg : 3'h0;//队列为空则在最低位输出0
endmodule
```

## FIFO顶层模块

将各部分例化，包括取边缘部分，在此给出框图：（省略clk和rst）



## 仿真

- 仿真文件：

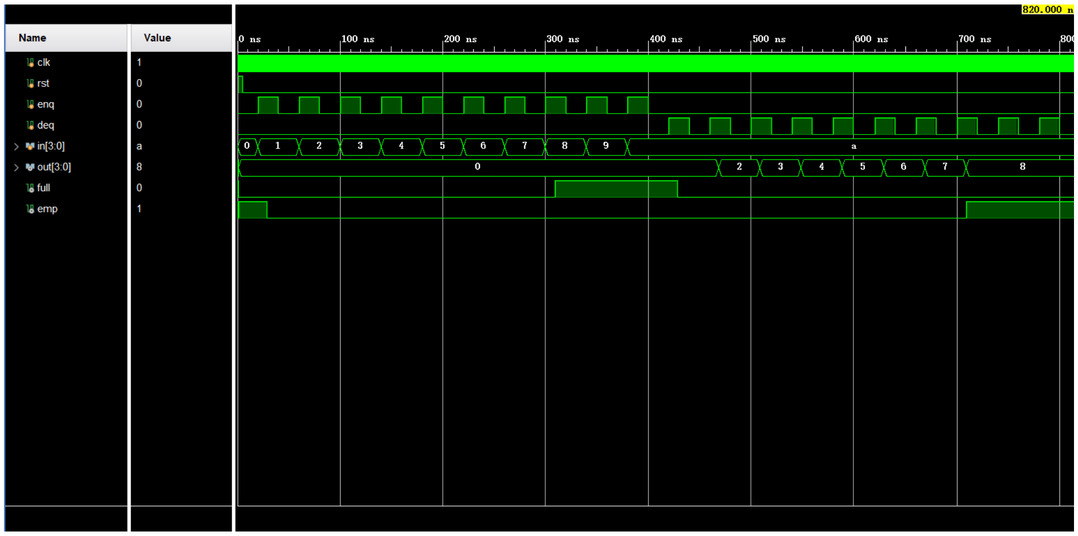
```
`timescale 1ns / 1ps
module tb();
reg clk;
reg rst;
reg enq;
reg deq;
reg [3:0] in;
wire [3:0] out;
wire full;
wire emp;
fifo test(
.clk(clk),
.rst(rst),
```

```verilog
.enq(enq),
.deq(deq),
.in(in),
.out(out),
.full(full),
.emp(emp)
);
initial begin
clk <= 1'b0;
forever
#1 clk <= ~clk;
end
initial begin
rst <= 1'b1;
#5 rst <= 1'b0;
end
initial begin
enq <= 1'b0;
deq <= 1'b0;
in <= 4'h0;
#20 enq <= 1'b1; // 1st enqueue
in <= 4'h1;
#20 enq <= 1'b0;
#20 enq <= 1'b1; // 2nd enqueue
in <= 4'h2;
#20 enq <= 1'b0;
#20 enq <= 1'b1; // 3rd enqueue
in <= 4'h3;
#20 enq <= 1'b0;
#20 enq <= 1'b1; // 4th enqueue
in <= 4'h4;
#20 enq <= 1'b0;
#20 enq <= 1'b1; // 5th enqueue
in <= 4'h5;
#20 enq <= 1'b0;
#20 enq <= 1'b1; // 6th enqueue
in <= 4'h6;
#20 enq <= 1'b0;
#20 enq <= 1'b1; // 7th enqueue
in <= 4'h7;
#20 enq <= 1'b0;
#20 enq <= 1'b1; // 8th enqueue
in <= 4'h8;
#20 enq <= 1'b0;
#20 enq <= 1'b1; // 9th enqueue (invalid)
```

```
in <= 4'h9;
#20 enq <= 1'b0;
#20 enq <= 1'b1; // 10th enqueue (invalid)
in <= 4'hA;
#20 enq <= 1'b0;
#20 deq <= 1'b1; // 1st dequeue
#20 deq <= 1'b0;
#20 deq <= 1'b1; // 2nd dequeue
#20 deq <= 1'b0;
#20 deq <= 1'b1; // 3rd dequeue
#20 deq <= 1'b0;
#20 deq <= 1'b1; // 4th dequeue
#20 deq <= 1'b0;
#20 deq <= 1'b1; // 5th dequeue
#20 deq <= 1'b0;
#20 deq <= 1'b1; // 6th dequeue
#20 deq <= 1'b0;
#20 deq <= 1'b1; // 7th dequeue
#20 deq <= 1'b0;
#20 deq <= 1'b1; // 8th dequeue
#20 deq <= 1'b0;
#20 deq <= 1'b1; // 9th dequeue (invalid)
#20 deq <= 1'b0;
#20 deq <= 1'b1; // 10th dequeue (invalid)
#20 deq <= 1'b0;
#20 $finish;
end
endmodule

```

- 仿真波形：



可以看出在x0入队时，full变为低电平，说明有数据入队，但是输出仍为0，表示x0恒为0，符合设计

> 下载至FPGA测试

已在线下检查，正确

## 4. 总结

本次实验难度适中

在完成的过程学习到了更多的取信号边缘方法，以及降频的方法

但是ppt的讲解不清晰，读了很久也难以理解最终要求，而且贴图和实验要求不符，希望之后实验能有更加 详细且准确的实验文档