

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

```
1  `timescale 1ns / 1ps
2  // 32 * WIDTH Register File
3  module register_file #(parameter WIDTH = 32)
4  (
5      input clk,
6      input [4:0] ra0,           // 读端口0地址
7      input [4:0] ra1,           // 读端口1地址
8      input [4:0] wa,           // 写端口地址
9      input we,                 // 写使能 (pos)
10     input [WIDTH-1:0] wd,       // 写端口数据
11     output [WIDTH-1:0] rd0,     // 读端口0数据
12     output [WIDTH-1:0] rd1     // 读端口1数据
```

```

13 );
14 reg [WIDTH-1:0] regfile [0:31];
15 assign rd0 = regfile[ra0];
16 assign rd1 = regfile[ra1];
17
18
19 always @(posedge clk) begin
20     if(wa==0)    regfile[0]<=0;
21     else if (we) regfile[wa] <= wd;
22 end
23 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

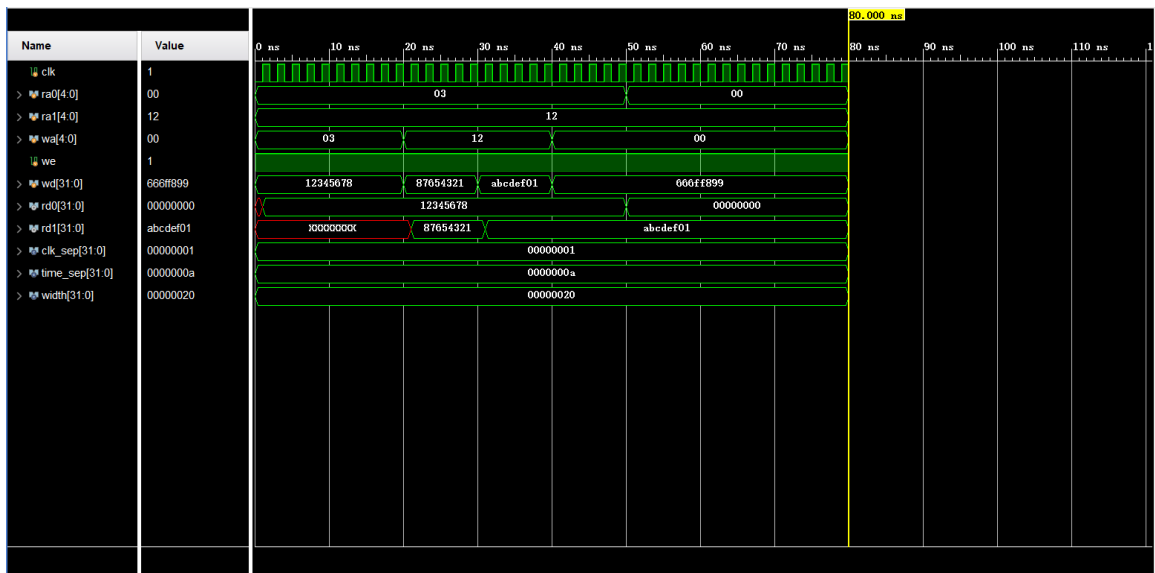
```

1  `timescale 1ns / 1ps
2  module testbench();
3      parameter clk_sep = 1;
4      parameter time_sep = 10;
5      parameter width = 32;
6      reg clk;
7      reg [4:0] ra0;
8      reg [4:0] ra1;
9      reg [4:0] wa;
10     reg we;
11     reg [width-1:0] wd;
12     wire [width-1:0] rd0;
13     wire [width-1:0] rd1;
14     register_file regfile(
15         .clk(clk),
16         .ra0(ra0),
17         .ra1(ra1),
18         .wa(wa),
19         .we(we),
20         .wd(wd),
21         .rd0(rd0),
22         .rd1(rd1)
23     );
24     initial begin
25         clk = 0;
26         ra0 = 5'h03;

```

```
27  ral = 5'h12;
28  forever #clk_sep clk = ~clk;
29  end
30  initial begin
31  we = 1'b1;
32  wa = 5'h03;
33  wd = 32'h12345678;
34  #time_sep
35  we = 1'b1;
36  #time_sep
37  wa = 5'h12;
38  wd = 32'h87654321;
39  #time_sep
40  we = 1'b1;
41  wd = 32'habcdef01;
42  #time_sep
43  we = 1'b1;
44  wa = 5'h0;
45  wd = 32'h666ff899;
46  #time_sep
47  we = 1'b1;
48  ra0 = 0;
49  #time_sep
50  we = 1'b1;
51  wa = 5'h0;
52  #time_sep
53  we = 1'b1;
54  #time_sep
55  $finish;
56  end
57  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）



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

```

1  `timescale 1ns / 1ps
2  module tb();
3      reg clk;
4      reg [3:0] addr;
5      reg [7:0] in;
6      reg we;
7      initial begin
8          clk <= 1'b0;
9          forever
10             #1 clk <= ~clk;
11         end
12         initial begin
13             addr <= 4'h0;
14             in <= 8'h00;
15             we <= 1'b0;
16             #10 addr <= 4'h1;
17             #10 addr <= 4'h2;
18             #10 addr <= 4'h3;
19             #10 addr <= 4'h4;
20             #10 addr <= 4'h5;
21             #10 addr <= 4'h6;
22             #10 addr <= 4'h7;
23             #10 addr <= 4'h8;
24             #10 addr <= 4'h9;

```

```
25 #10 addr <= 4'hA;
26 #10 addr <= 4'hB;
27 #10 addr <= 4'hC;
28 #10 addr <= 4'hD;
29 #10 addr <= 4'hE;
30 #10 addr <= 4'hF;
31 #10 addr <= 4'h0;
32 in <= 8'h00;
33 we <= 1'b1;
34 #10 addr <= 4'h1;
35 in <= 8'h11;
36 #10 addr <= 4'h2;
37 in <= 8'h22;
38 #10 addr <= 4'h3;
39 in <= 8'h33;
40 #10 addr <= 4'h4;
41 in <= 8'h44;
42 #10 addr <= 4'h5;
43 in <= 8'h55;
44 #10 addr <= 4'h6;
45 in <= 8'h66;
46 #10 addr <= 4'h7;
47 in <= 8'h77;
48 #10 addr <= 4'h8;
49 in <= 8'h88;
50 #10 addr <= 4'h9;
51 in <= 8'h99;
52 #10 addr <= 4'hA;
53 in <= 8'hAA;
54 #10 addr <= 4'hB;
55 in <= 8'hBB;
56 #10 addr <= 4'hC;
57 in <= 8'hCC;
58 #10 addr <= 4'hD;
59 in <= 8'hDD;
60 #10 addr <= 4'hE;
61 in <= 8'hEE;
62 #10 addr <= 4'hF;
63 in <= 8'hFF;
64 #10 addr <= 4'h0;
65 we <= 1'b0;
66 #10 addr <= 4'h1;
67 #10 addr <= 4'h2;
68 #10 addr <= 4'h3;
69 #10 addr <= 4'h4;
```

```
70 #10 addr <= 4'h5;
71 #10 addr <= 4'h6;
72 #10 addr <= 4'h7;
73 #10 addr <= 4'h8;
74 #10 addr <= 4'h9;
75 #10 addr <= 4'hA;
76 #10 addr <= 4'hB;
77 #10 addr <= 4'hC;
78 #10 addr <= 4'hD;
79 #10 addr <= 4'hE;
80 #10 addr <= 4'hF;
81 #10 $finish;
82 end
83 // block memory
84 reg ena;
85 wire [7:0] out_block_0;
86 wire [7:0] out_block_1;
87 wire [7:0] out_block_2;
88 initial begin
89   ena <= 1'b1;
90   #330 ena <= 1'b0;
91   forever
92     #5 ena <= ~ena;
93   end
94 blk_mem_gen_0 test_block_0 (
95   .clka(clk),
96   .addra(addr),
97   .dina(in),
98   .douta(out_block_0),
99   .wea(we)
100 );
101 blk_mem_gen_1 test_block_1 (
102   .clka(clk),
103   .addra(addr),
104   .dina(in),
105   .douta(out_block_1),
106   .wea(we)
107 );
108 blk_mem_gen_2 test_block_2 (
109   .clka(clk),
110   .addra(addr),
111   .dina(in),
112   .douta(out_block_2),
113   .wea(we)
114 );
```

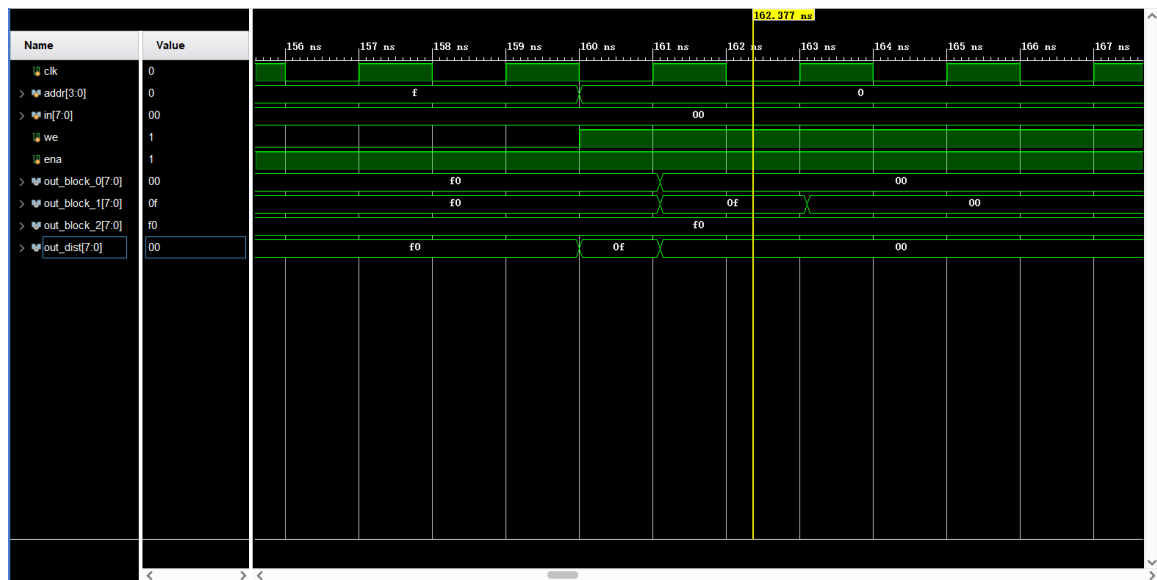
```

115 // distributed memory
116 wire [7:0] out_dist;
117 dist_mem_gen_0 test_dist(
118 .clk(clk),
119 .a(addr),
120 .d(in),
121 .we(we),
122 .spo(out_dist)
123 );
124
125 endmodule

```

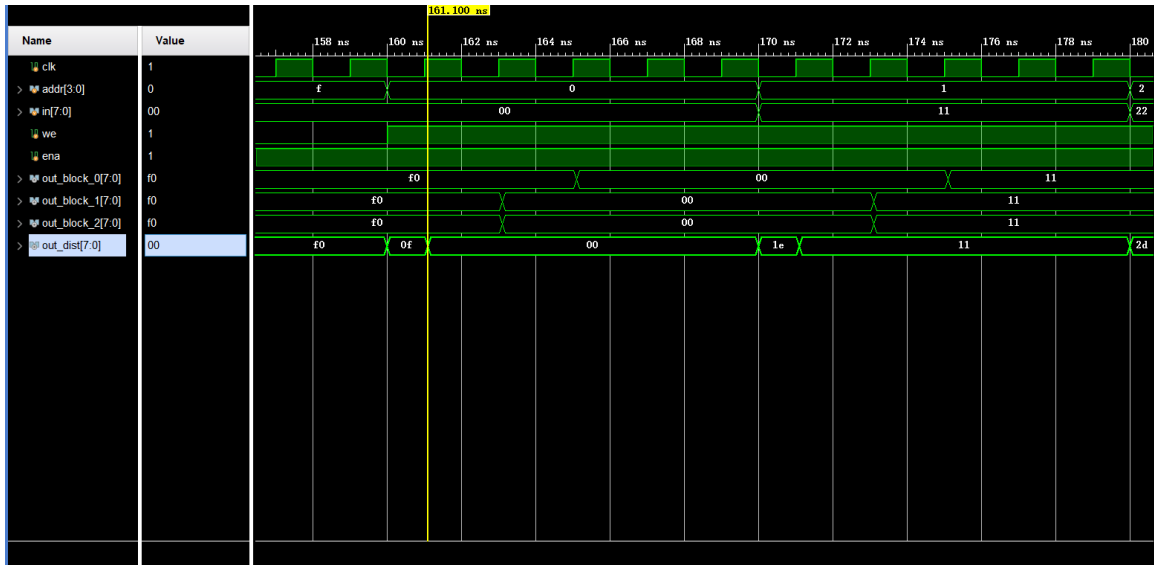
- 仿真结果

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）



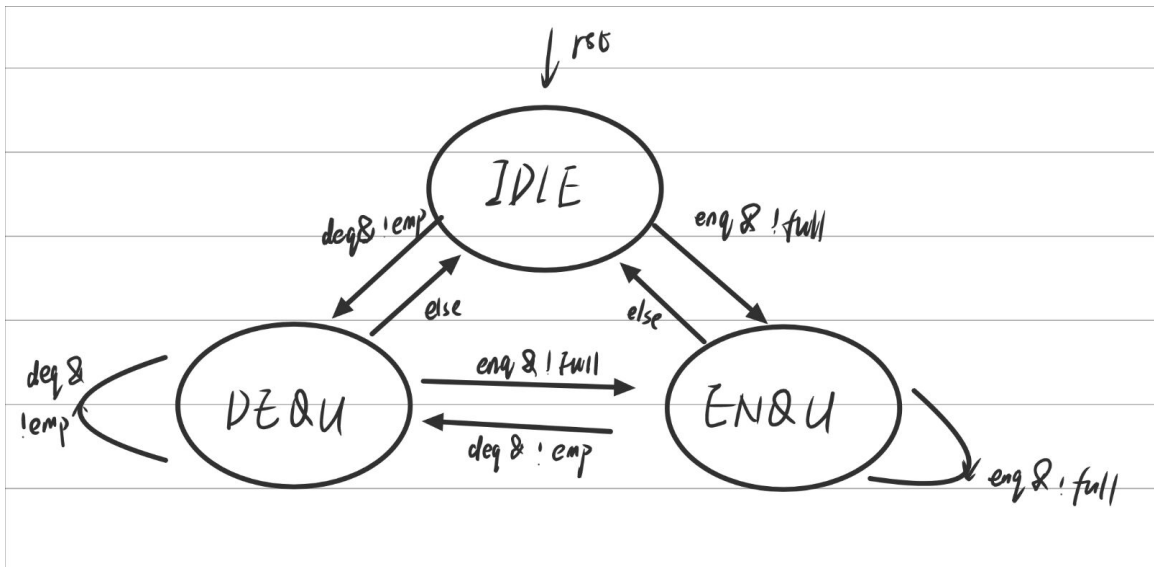
可以看出块式比分布式时序输出延后，而同时勾选两个寄存器比只勾选一个寄存器还延后一个周期，这是因为通过了两个寄存器进行输出

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：

```
1 module FSM(  
2     input clk,           //时钟信号  
3     input enq,           //入队标志
```

```

4      input deq,           //出队标志
5      input rst,          //同步复位(高电平有效)
6      input full,         //队满标志
7      input emp,          //队空标志
8      output reg [2:0] state //状态
9  );
10
11
12  parameter IDLE = 2'b00; //保持现状
13  parameter ENQU = 2'b01; //入队状态
14  parameter DEQU = 2'b10; //出队状态
15  // parameter IDLE = 2'b11;
16
17  reg [2:0] current_state;
18  reg [2:0] next_state;
19
20  //描述状态切换
21  always @(posedge clk) begin
22      if(rst) begin
23          current_state <= IDLE;
24      end
25      else begin
26          current_state <= next_state;
27      end
28  end
29
30
31  //判断状态转移条件，描述状态转移规律
32  always @(*) begin
33      case (current_state)
34          IDLE:
35              if(enq & ~full) begin
36                  next_state = ENQU;
37              end
38              else if(deq & ~emp) begin
39                  next_state = DEQU;
40              end
41              else begin
42                  next_state = IDLE;
43              end
44
45          ENQU:
46              if(enq & ~full) begin
47                  next_state = ENQU;
48              end

```

```

49     else if(deq & ~emp) begin
50         next_state = DEQU;
51     end
52     else begin
53         next_state = IDLE;
54     end
55
56     DEQU:
57         if(enq & ~full) begin
58             next_state = ENQU;
59         end
60         else if(deq & ~emp) begin
61             next_state = DEQU;
62         end
63         else begin
64             next_state = IDLE;
65         end
66
67
68     default:
69         next_state = IDLE;
70 endcase
71 end
72
73 //描述状态输出
74 always @(posedge clk) begin
75     state = current_state;
76 end
77
78 endmodule

```

2. LCU模块（调用FSM）

```

1  module list_control_unit(
2      input                clk,
3
4      input                rst,
5      input                [3:0] in,        // 入队数据
6      input                enq,            // 入队边缘
7      input                deq,            // 出队边缘
8      input                [3:0] rd,        // 写端口数据
9      output               full,
10     output               emp,            // 队列空
11     output reg           [3:0] out,        // 出队数据
12     output               [2:0] ra,        // 读端口地址
13     output               we,            // 写使能

```

```

14     output      [2:0]  wa,           // 写端口地址
15     output      [3:0]  wd,           // 写端口数据
16     output reg   [7:0]  valid        // 数据有效
17 );
18     reg [2:0] head;                  // 头指针
19     reg [2:0] tail;                  // 尾指针
20     wire [1:0] state;                // 状态
21
22     assign full = &valid;            // 当标志的每一位都为1时,说明队列已满
23     assign emp  = ~(|valid);         // 当标志的每一位都为0时,说明队列为空
24
25     assign ra = head;                // 出队,读端口地址等于队头
26     assign we = enq & ~full & ~rst;
27     assign wa = tail;                // 入队,写端口地址等于队尾
28     assign wd = in;
29
30     //状态产生
31     FSM fsm(
32         .clk(clk),
33         .enq(enq),
34         .deq(deq),
35         .rst(rst),
36         .full(full),
37         .emp(emp),
38         .state(state)
39     );
40
41     always @(posedge clk) begin
42         if(rst) begin
43             valid <= 8'h0;
44             head  <= 3'h0;
45             tail  <= 3'h0;
46             out   <= 3'h0;
47         end
48         else if (state == 2'b00) begin
49             valid <= valid;
50             head  <= head;
51             tail  <= tail;
52             out   <= out;
53         end
54
55         else if (state == 2'b01) begin
56             valid[tail] <= 1'b1;
57             tail        <= tail + 3'h1;
58         end

```

```

59
60         else if(state == 2'b10) begin
61             valid[head] <= 1'b0;
62             head         <= head + 3'h1;
63             out          <= rd;
64         end
65
66     end
67
68 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.

```

1  `timescale 1ns / 1ps
2  module segplay_unit(
3      input clk_100mhz,
4      input [3:0] data,
5      input [7:0] valid,
6      output reg [2:0] addr,
7      output [2:0] segplay_an,
8      output [3:0] segplay_data
9  );
10 // 降频到400Hz (250000倍)
11 wire clk_400hz;
12 reg [17:0] clk_cnt;
13 assign clk_400hz = ~(|clk_cnt); // 每满250000翻转时钟
14 always @(posedge clk_100mhz) begin
15     if (clk_cnt >= 18'h3D08F) begin // clk_cnt >= 249999
16         clk_cnt <= 18'h00000;
17         addr <= addr + 3'b001; // 取遍八位, 将有数据的输出
18     end else
19         clk_cnt <= clk_cnt + 18'h00001;
20     end
21
22 reg [2:0] segplay_an_reg;
23 reg [3:0] segplay_data_reg;
24 always @(posedge clk_100mhz) begin

```

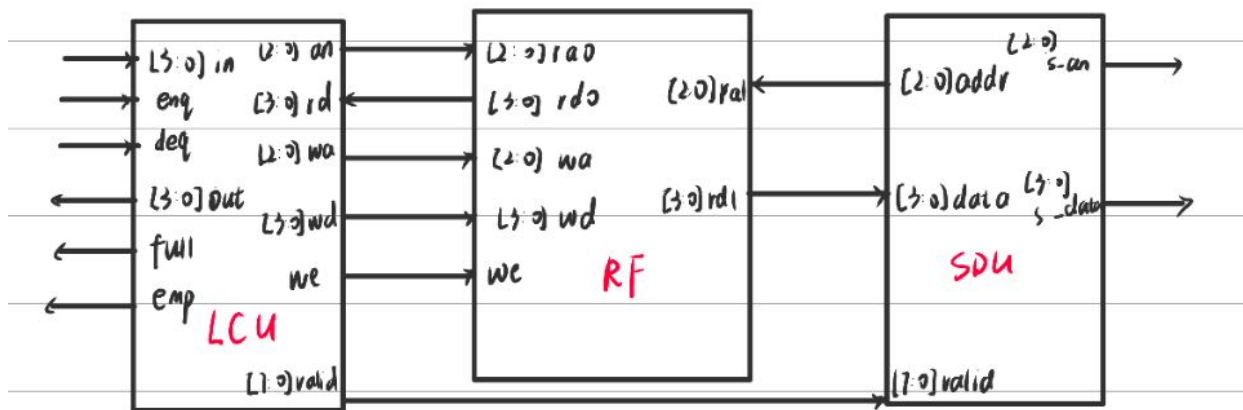
```

25  if (clk_400hz && valid[addr]) begin //有数据在addr，则数码管展示出来
26  segplay_an_reg <= addr;
27  segplay_data_reg <= data;
28  end
29  end
30  assign segplay_data = (!valid) ? segplay_data_reg : 4'h0;
31  assign segplay_an = (!valid) ? segplay_an_reg : 3'h0; //队列为空则在最低位
    输出0
32  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:

```

1  `timescale 1ns / 1ps
2  module tb();
3  reg clk;
4  reg rst;
5  reg enq;
6  reg deq;
7  reg [3:0] in;
8  wire [3:0] out;
9  wire full;
10 wire emp;
11 fifo test(
12 .clk(clk),
13 .rst(rst),

```

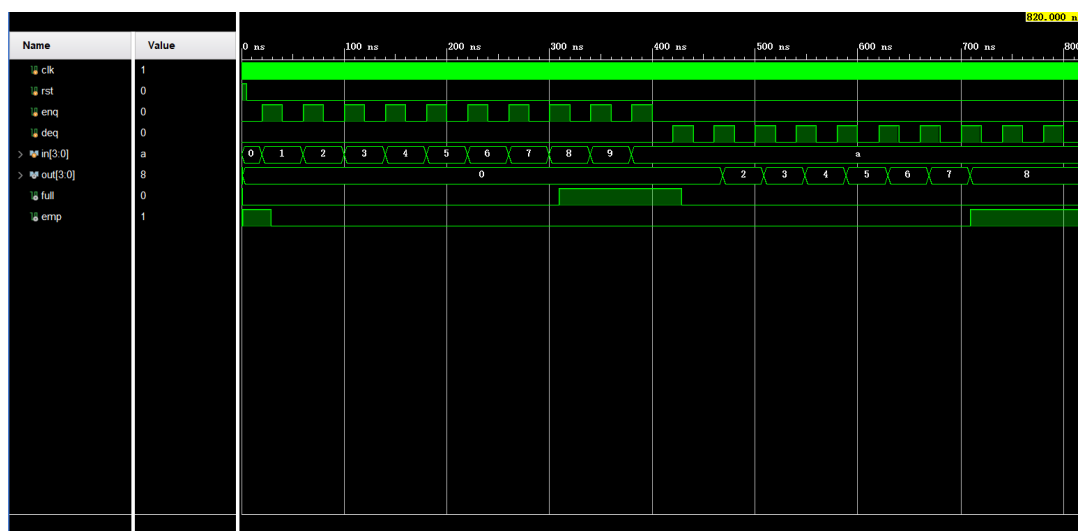
```
14 .enq(enq),
15 .deq(deq),
16 .in(in),
17 .out(out),
18 .full(full),
19 .emp(emp)
20 );
21 initial begin
22     clk <= 1'b0;
23     forever
24         #1 clk <= ~clk;
25     end
26     initial begin
27         rst <= 1'b1;
28         #5 rst <= 1'b0;
29     end
30     initial begin
31         enq <= 1'b0;
32         deq <= 1'b0;
33         in <= 4'h0;
34         #20 enq <= 1'b1; // 1st enqueue
35         in <= 4'h1;
36         #20 enq <= 1'b0;
37         #20 enq <= 1'b1; // 2nd enqueue
38         in <= 4'h2;
39         #20 enq <= 1'b0;
40         #20 enq <= 1'b1; // 3rd enqueue
41         in <= 4'h3;
42         #20 enq <= 1'b0;
43         #20 enq <= 1'b1; // 4th enqueue
44         in <= 4'h4;
45         #20 enq <= 1'b0;
46         #20 enq <= 1'b1; // 5th enqueue
47         in <= 4'h5;
48         #20 enq <= 1'b0;
49         #20 enq <= 1'b1; // 6th enqueue
50         in <= 4'h6;
51         #20 enq <= 1'b0;
52         #20 enq <= 1'b1; // 7th enqueue
53         in <= 4'h7;
54         #20 enq <= 1'b0;
55         #20 enq <= 1'b1; // 8th enqueue
56         in <= 4'h8;
57         #20 enq <= 1'b0;
58         #20 enq <= 1'b1; // 9th enqueue (invalid)
```

```

59  in <= 4'h9;
60  #20 enq <= 1'b0;
61  #20 enq <= 1'b1; // 10th enqueue (invalid)
62  in <= 4'hA;
63  #20 enq <= 1'b0;
64  #20 deq <= 1'b1; // 1st dequeue
65  #20 deq <= 1'b0;
66  #20 deq <= 1'b1; // 2nd dequeue
67  #20 deq <= 1'b0;
68  #20 deq <= 1'b1; // 3rd dequeue
69  #20 deq <= 1'b0;
70  #20 deq <= 1'b1; // 4th dequeue
71  #20 deq <= 1'b0;
72  #20 deq <= 1'b1; // 5th dequeue
73  #20 deq <= 1'b0;
74  #20 deq <= 1'b1; // 6th dequeue
75  #20 deq <= 1'b0;
76  #20 deq <= 1'b1; // 7th dequeue
77  #20 deq <= 1'b0;
78  #20 deq <= 1'b1; // 8th dequeue
79  #20 deq <= 1'b0;
80  #20 deq <= 1'b1; // 9th dequeue (invalid)
81  #20 deq <= 1'b0;
82  #20 deq <= 1'b1; // 10th dequeue (invalid)
83  #20 deq <= 1'b0;
84  #20 $finish;
85  end
86  endmodule
87

```

- 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