计算机组成原理 实验报告

姓名：宋玮　学号：PB20151793　实验日期：2022.3.30

一、实验题目：

Lab02 寄存器堆与存储器及其应用

二、实验目的：

- 掌握寄存器堆（Register File）和存储器的功能、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

三、实验平台：

Vivado, fpgaol

四、实验过程及结果：

如下所示：

● 寄存器堆

1.verilog 设计代码

注：m，width，sum 为三个参数，方便后续例化不同大小的寄存器堆

下面是 8×4 位的寄存器堆

```
module register_file#(
parameter m = 3, WIDTH = 4
)(
    input clk,
    input we,
    input [m-1:0] wa,
    input [m-1:0] ra0,ra1,
    input [WIDTH-1:0] wd,
    output [WIDTH-1:0] rd0,rd1
);

parameter sum = 8'b1 << m;
reg [WIDTH-1:0] regfile [0:sum-1];

assign rd0 = regfile[ra0],
```

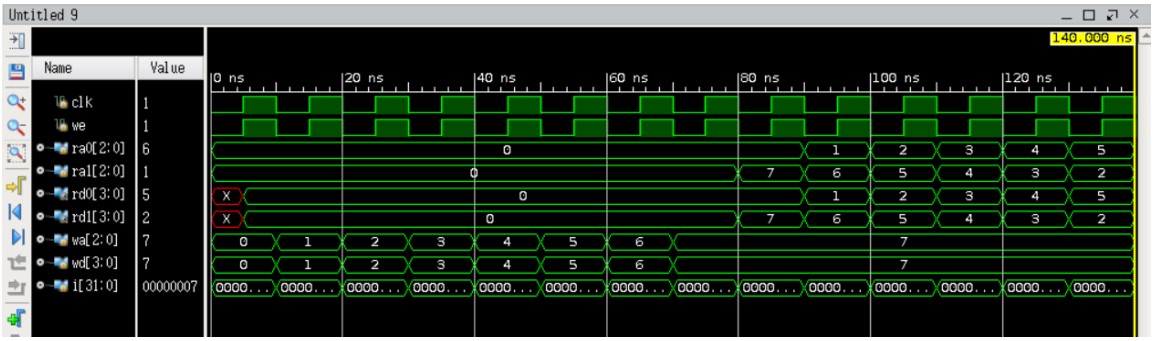
```
        rd1 = regfile[ra1];

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

2.仿真文件

```
module rf_sim();
reg clk;
reg we;
reg [2:0] ra0,ra1;
wire [3:0] rd0,rd1;
reg [2:0] wa;
reg [3:0] wd;
integer i;

register_file rf(.clk(clk),.we(we),.rd0(rd0),.rd1(rd1),.ra0(ra0),.ra1(ra1),.wa(wa),.wd(wd));

initial clk = 0;
always #5 clk = ~clk;

initial we = 0;
always #5 we = ~we;

initial
begin
    wa = 0; wd = 0; ra0 = 0; ra1 = 0;
    for(i=1;i<8;i=i+1)
    begin
        #10; wa = wa+1; wd = wd + 1;
    end
    for(i=0;i<7;i=i+1)
    begin
        #10; ra0 = i; ra1 = 7-i;
    end
    $finish;
end
endmodule
```

3.仿真结果

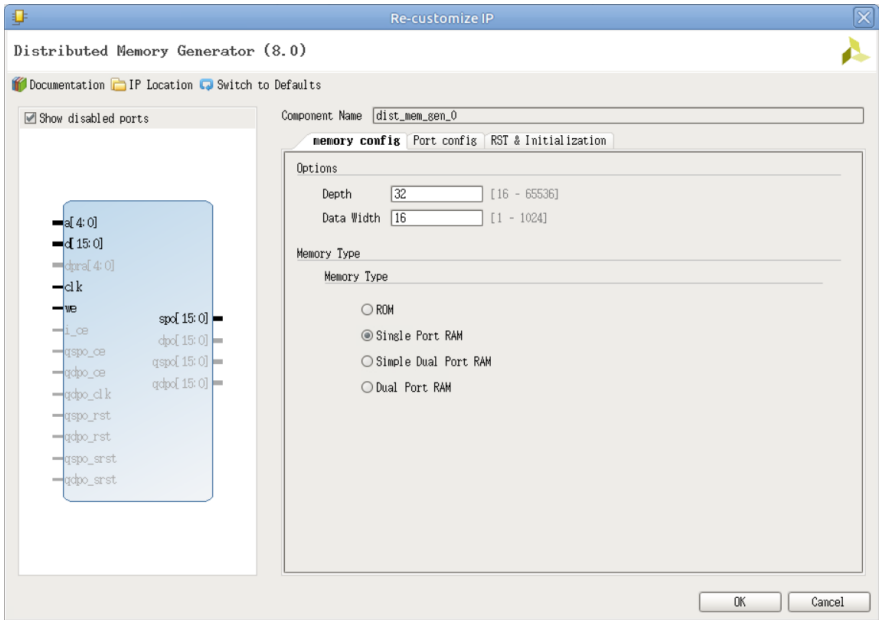
前面写入 0~7，后面通过 rd0，rd1 输出 0~7



● 存储器 IP 核

1.分布式存储器（Distributed Memory Generator）

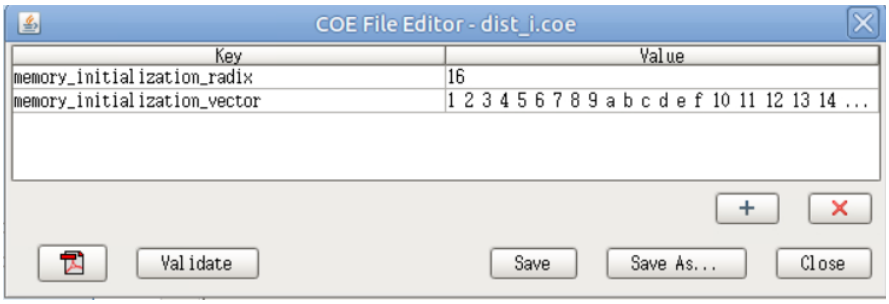
（1）在 IP Catalog 中生成分布式存储器 ip 核（dist_mem_gen_0）



（2）载入 coe 文件:

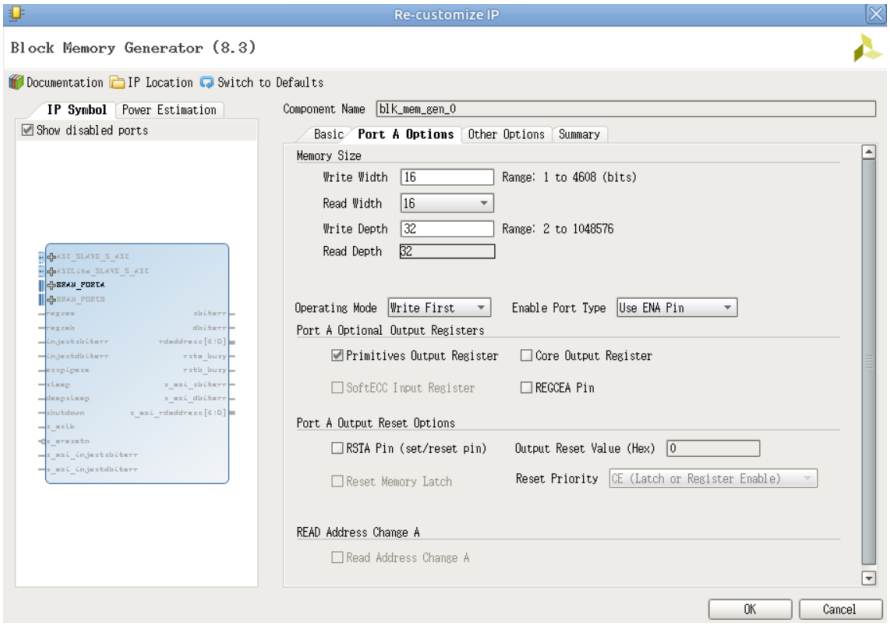
memory_initialization_radix = 16;

memory_initialization_vector = 1 2 3 4 5 6 7 8 9 a b c d e f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20;



2.块式存储器（Block Memory Generator）

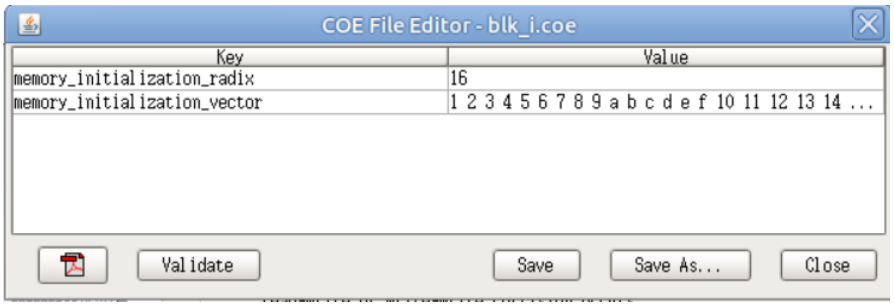
（1）在 IP Catalog 中生成块式存储器 ip 核（blk_mem_gen_0）



（2）载入 coe 文件:

memory_initialization_radix = 16;

memory_initialization_vector = 1 2 3 4 5 6 7 8 9 a b c d e f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20;



3.仿真文件

```
module dist();
reg clk,we,en;
reg [4:0] a;
reg [15:0] wd;
wire [15:0] distd, blkd;

dist_mem_gen_0 distmem (
   .a(a),          // input wire [4 : 0] a
   .d(wd),         // input wire [15 : 0] d
   .clk(clk),   // input wire clk
   .we(we),       // input wire we
   .spo(distd)   // output wire [15 : 0] spo
);
blk_mem_gen_0 blkmem (
   .clka(clk),       // input wire clka
   .ena(en),         // input wire ena
   .wea(we),         // input wire [0 : 0] wea
   .addra(a),   // input wire [4 : 0] addra
   .dina(wd),       // input wire [15 : 0] dina
   .douta(blkd)   // output wire [15 : 0] douta
);

initial clk = 0;
always #5 clk = ~clk;

initial
begin
en = 1; a = 1; wd = 0; we = 1;
#6    wd = 4;
#6 a = 2;
#8 wd = 5;
#10 wd = 6;
#1 a = 3;
#10 $finish;
end

endmodule
```

4.仿真结果

注意到：
分布式存储器为同步写，异步读；

块式存储器为同步写，并且在下一个周期读出（原因：块式存储器的内部结构，写入的数据在读出时要经过 D 触发器，导致慢一个周期，见图 2）

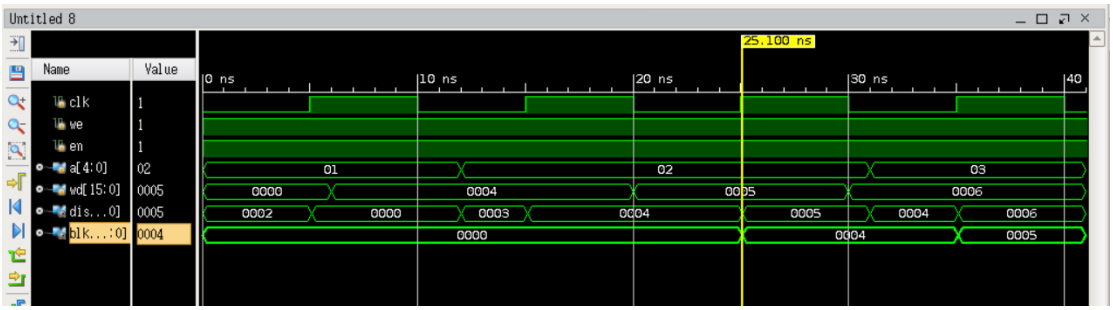


图 1

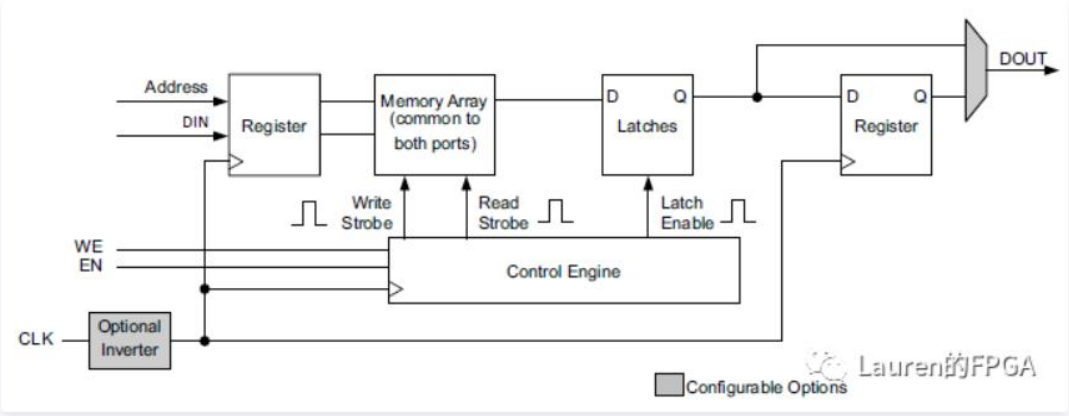


图 2

- **FIFO 队列**

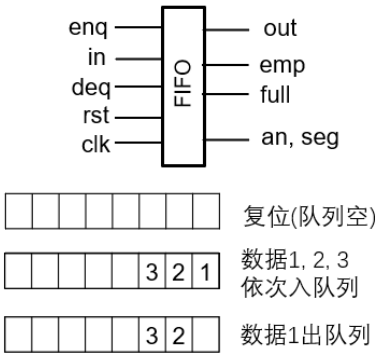
用三端口8×4寄存器堆实现最大长度为8的FIFO队列

deq, enq：出/入队列使能(互斥)，一次有效仅允许操作一项数据

out, in：出/入队列数据

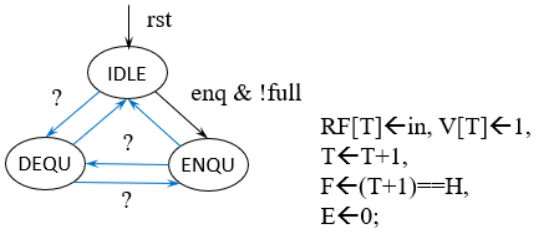
full, emp：队列满/空，满/空时忽略入/出队操作

an, seg：数码管控制信号，显示队列状态



1.设计文件

- 根据要求，设置了四个状态——rrst（复位状态），idle（闲置状态），enqu（入队状态），dequ（出队状态）；



- 也设置了队头指针（head）和队尾指针（tail），分别指向队头和队尾元素位置；
- 队列最大长度为 8，含有元素数量由 num 表示，有效位由 valid[]表示；

具体 verilog 代码如下：

```
module fifo(
input clk, rst,
input enq,
input [3:0] in,
input deq,
output [3:0] out,
output [2:0] an,
output [3:0] seg,
output full,
output empty
);

reg [2:0] head,tail;
reg we;
wire [2:0]ra0;
wire [3:0]rd0;
wire [3:0]rd1;

reg button_r1,button_r2,button_r3,button_r4;
wire enq_edge,deq_edge;

reg [2:0] led = 0;

wire t;
reg [31:0] cnt = 32'b0;

reg dequeue=0;              //out
reg [4:0]num;
reg [7:0]valid;
reg [1:0]ns,cs;
```

```
parameter idle = 2'b00, enqu = 2'b01, dequ = 2'b10, rrst = 2'b11;

//去抖动
always@(posedge clk)
    button_r1 <= enq;
always@(posedge clk)
    button_r2 <= button_r1;
assign enq_edge = button_r1 & (~button_r2);

always@(posedge clk)
    button_r3 <= deq;
always@(posedge clk)
    button_r4 <= button_r3;
assign deq_edge = button_r3 & (~button_r4);

register_file r_f(.clk(clk),.we(we),.ra0(ra0),.rd0(rd0),.ra1(an),.rd1(rd1),.wa(tail),.wd(in));

assign empty = (num == 0) ? 1 : 0;
assign full = (num == 8) ? 1 : 0;

//描述 ns
always@(*)
begin
    if(rst) ns = rrst;
    else
    begin
        case(cs)
            rrst:
                begin
                    if(enq_edge && !full) ns = enqu;
                    else if(deq_edge && !empty) ns = dequ;
                    else ns = rrst;
                end
            default:
                begin
                    if(enq_edge && !full) ns = enqu;
                    else if(deq_edge && !empty) ns = dequ;
                    else ns = idle;
                end
        endcase
    end
end

//描述 cs
```

```
always@(posedge clk)
begin
    cs <= ns;
end

always@(posedge clk)
begin
    case(ns)
        enqu:
            begin
                num <= num + 1;
                tail <= tail + 3'b1;
                we <= 1;
                valid[tail+3'b1] <= 1'b1;
            end
        dequ:
            begin
                num <= num - 1;
                head <= head + 3'b1;
                we <= 0;
                dequeue <= 1;
                valid[head] <= 1'b0;
            end
        rrst:
            begin
                head <= 0;
                tail <= 3'b111;
                num <= 0;
                we <= 0;
                dequeue <= 0;
                valid <= 8'b0;
            end
        idle:
            begin
                we <= 0;
            end
    endcase
end

assign ra0 = head ? head-1 : 3'b111;
assign out = dequeue ? rd0 : 0;

//时钟分时复用
always@(posedge clk)
```

```
begin
    if(cnt >= 400000)
        cnt <= 32'b0;
    else
        cnt <= cnt + 32'b1;
end
assign t = (cnt == 32'b1);

always@(posedge clk)
begin
    if(t)
    begin
        if( valid[led+1] == 1'b0 ) led <= head;
        else led <= led + 1;
    end
end

//描述输出
assign an = led;
assign seg = empty ? 15 : rd1;    //ra1:an

endmodule
```

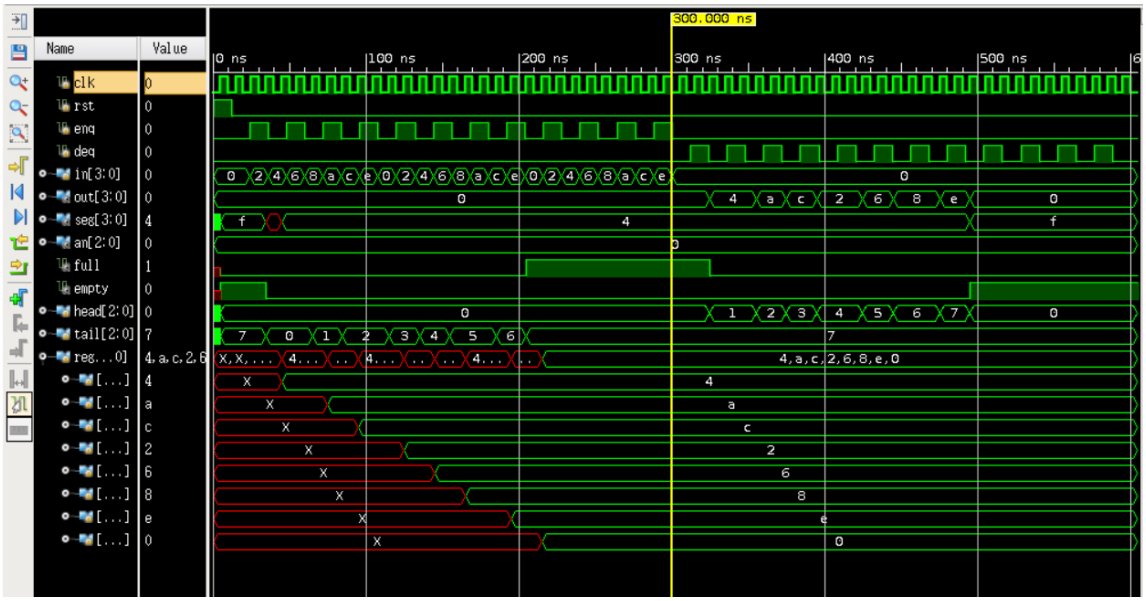
2.仿真文件

```
module fifo_sim();
reg clk, rst;
reg enq,deq;
reg [3:0] in;

wire [3:0] out,seg;
wire [2:0] an;
wire full,empty;

integer i;

initial clk = 0;
always #5 clk = ~clk;

fifo fi_fo(.clk(clk),.rst(rst),.enq(enq),.deq(deq),.in(in),.out(out),.seg(seg),.an(an),.full(full),.empty(empty));

initial
begin
```

```
            rst = 1; enq = 0; deq = 0; in = 0;
        #12     rst = 0;
        for (i=0; i<24; i=i+1)
        begin
            #12
            enq = ~enq;
            in = in + 2;
        end
        for(i=0; i<24; i=i+1)
        begin
            #12
            deq = ~deq ;
        end
        #16 $finish;
end
endmodule
```

3.仿真结果

首先，4,a,c,2,6,8,e,0 入队，之后队满，无法再入队，并且 full 变为高电平；
之后，4,a,c,2,6,8,e,0 依次出队，直至队空，显示 f（false），并且 empty 变为高电平



4.约束文件

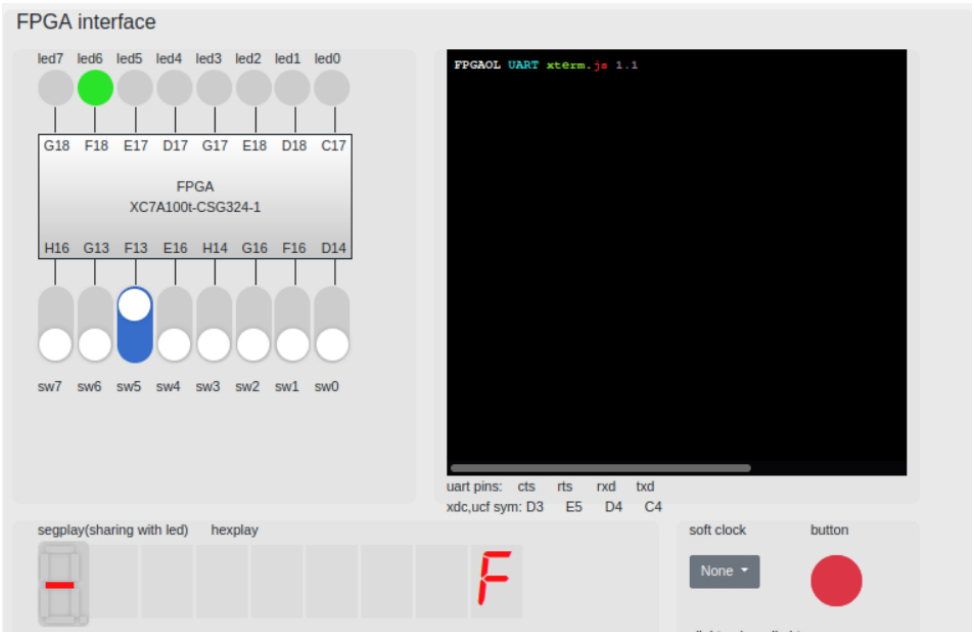
```
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports { clk }];
```

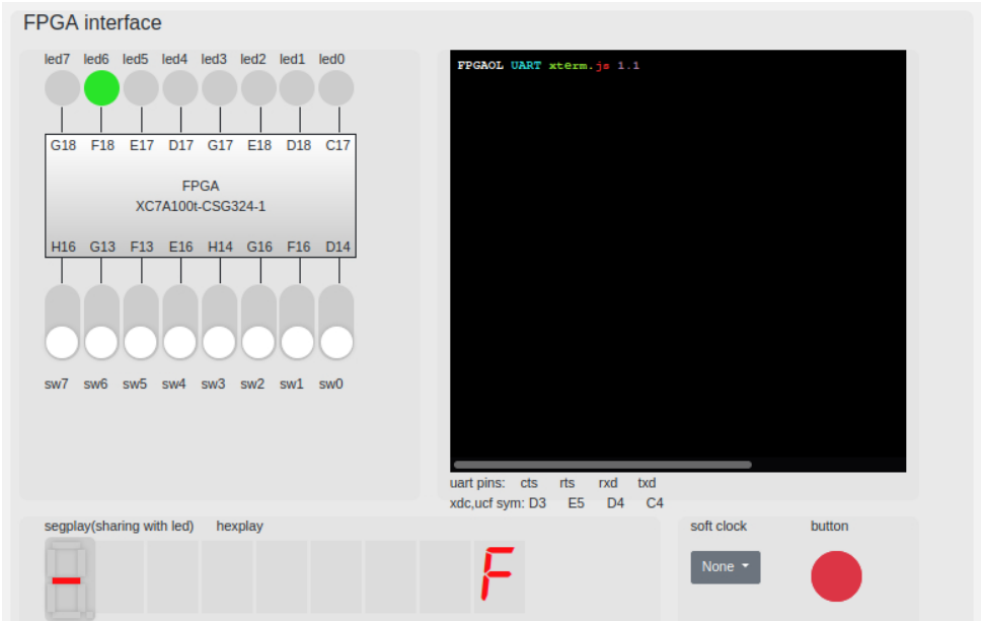
```
set_property -dict { PACKAGE_PIN F13 IOSTANDARD LVCMOS33} [get_ports { rst }];
set_property -dict { PACKAGE_PIN H16 IOSTANDARD LVCMOS33} [get_ports { enq }];
set_property -dict { PACKAGE_PIN G13 IOSTANDARD LVCMOS33} [get_ports { deq }];
set_property -dict { PACKAGE_PIN C17 IOSTANDARD LVCMOS33} [get_ports { out[0] }];
set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33} [get_ports { out[1] }];
set_property -dict { PACKAGE_PIN E18 IOSTANDARD LVCMOS33} [get_ports { out[2] }];
set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33} [get_ports { out[3] }];
set_property -dict { PACKAGE_PIN G18 IOSTANDARD LVCMOS33} [get_ports { full }];
set_property -dict { PACKAGE_PIN F18 IOSTANDARD LVCMOS33} [get_ports { empty }];
set_property -dict { PACKAGE_PIN D14 IOSTANDARD LVCMOS33} [get_ports { in[0] }];
set_property -dict { PACKAGE_PIN F16 IOSTANDARD LVCMOS33} [get_ports { in[1] }];
set_property -dict { PACKAGE_PIN G16 IOSTANDARD LVCMOS33} [get_ports { in[2] }];
set_property -dict { PACKAGE_PIN H14 IOSTANDARD LVCMOS33} [get_ports { in[3] }];
set_property -dict { PACKAGE_PIN B17 IOSTANDARD LVCMOS33} [get_ports { an[0] }];
set_property -dict { PACKAGE_PIN B16 IOSTANDARD LVCMOS33} [get_ports { an[1] }];
set_property -dict { PACKAGE_PIN A18 IOSTANDARD LVCMOS33} [get_ports { an[2] }];
set_property -dict { PACKAGE_PIN A14 IOSTANDARD LVCMOS33} [get_ports { seg[0] }];
set_property -dict { PACKAGE_PIN A13 IOSTANDARD LVCMOS33} [get_ports { seg[1] }];
set_property -dict { PACKAGE_PIN A16 IOSTANDARD LVCMOS33} [get_ports { seg[2] }];
set_property -dict { PACKAGE_PIN A15 IOSTANDARD LVCMOS33} [get_ports { seg[3] }];
```

5.fpga ol 测试 bit 文件

（1）首先，复位，sw5 为复位键

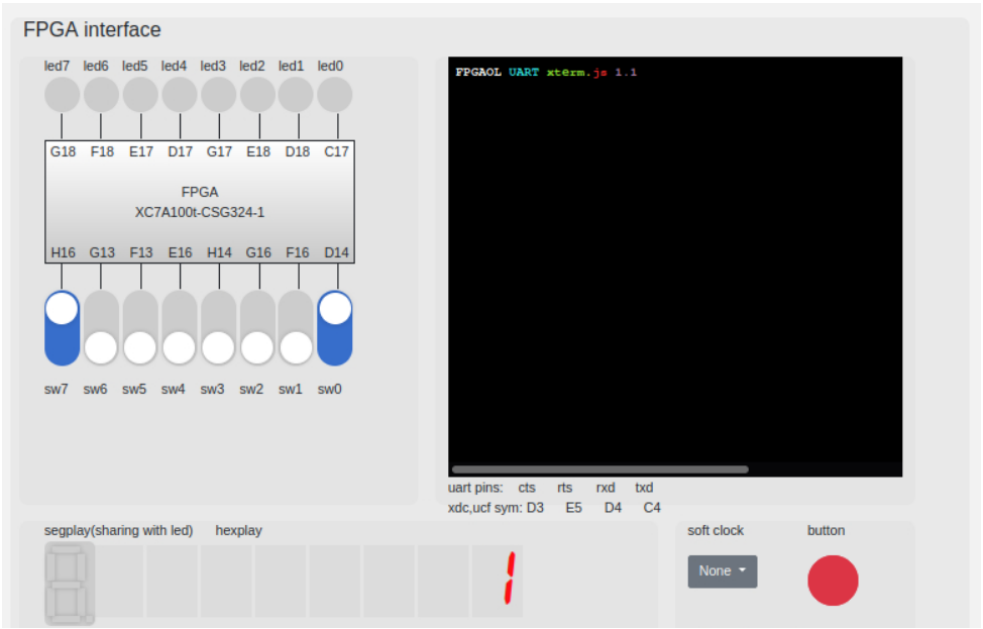
队空，led6（empty）亮起，并且数码管显示 f（false），表示此时队内没有元素

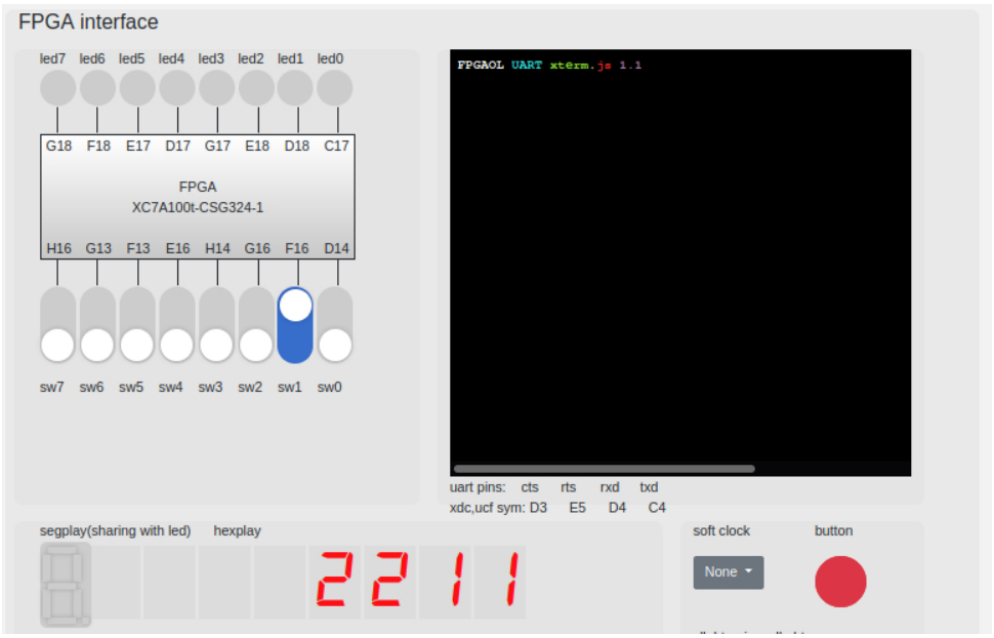


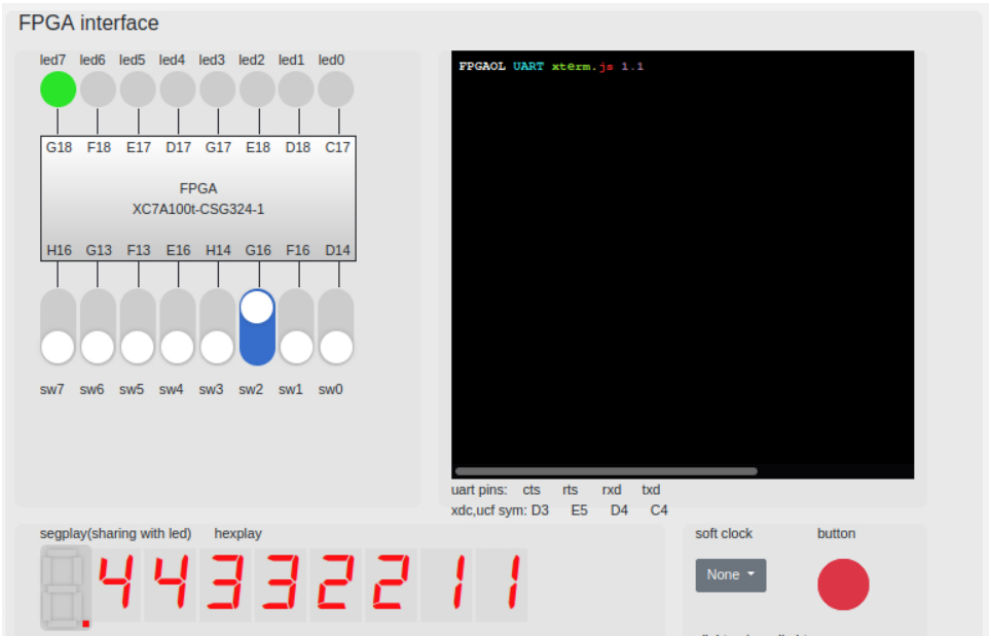


（2）入队 1,1,2,2,3,3,4,4，sw7 位入队按钮

最后队满，led7（full）亮起

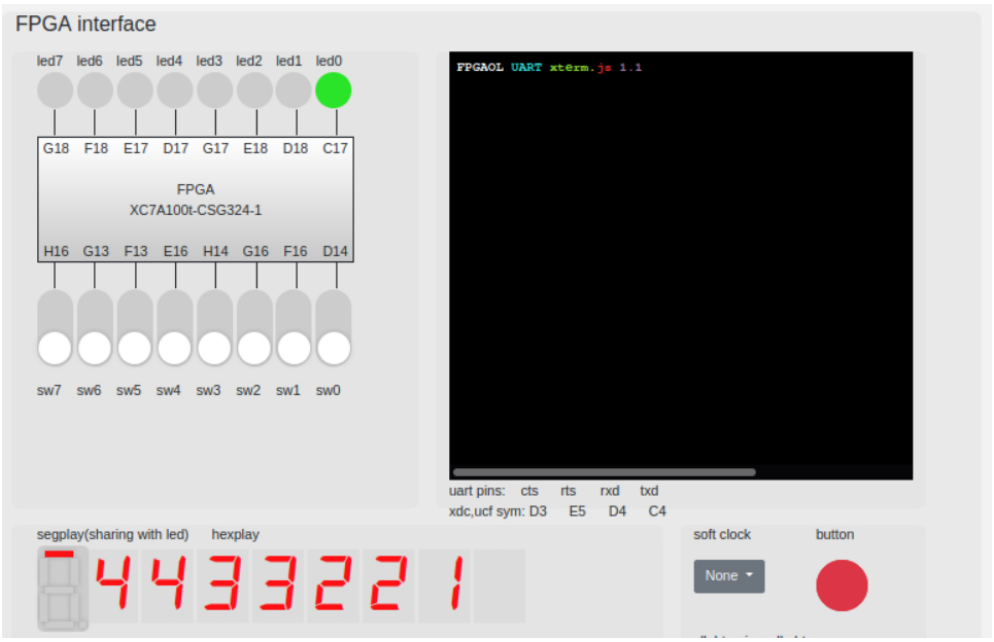


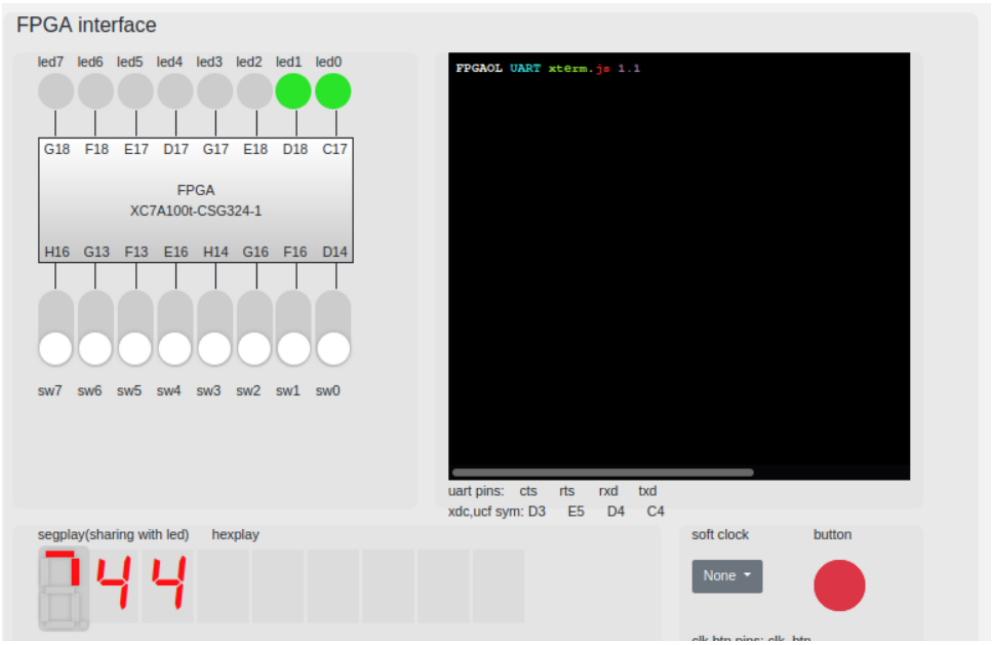


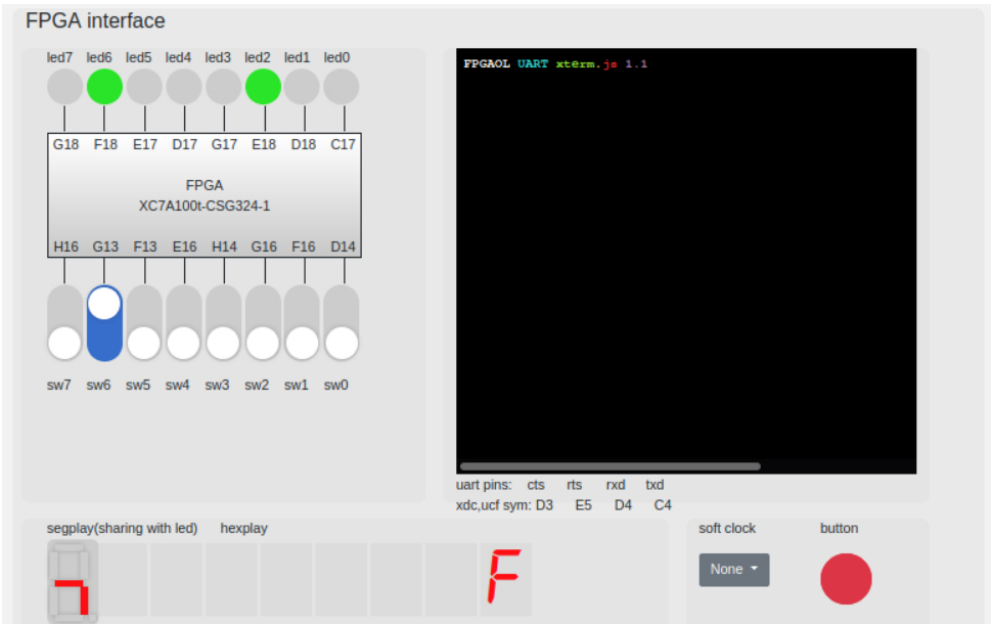


（3）1,1,2,2,3,3,4,4 依次出队，sw6 为出队按钮

最后 led6（empty）亮起，并且数码管显示 f（false），表示此时队内没有元素


FPGA interface
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4
segplay(sharing with led)
hexplay
4 4 3 3 2 2 1
soft clock
button
None


FPGA interface
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4
segplay(sharing with led)
hexplay
4 4
soft clock
button
None


FPGA interface
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4
segplay(sharing with led)
hexplay
F
soft clock
None
button