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：寄存器堆与存储器及其应用

学生姓名：吴毅龙

学生学号：PB19111749

完成日期：2021/4/12

计算机实验教学中心制

2020 年 09 月

【实验题目】

寄存器堆与存贮器及其应用

【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

【实验环境】

FPGAOL: fpgaol.ustc.edu.cn（或 Nexys4 DDR）

Vivado

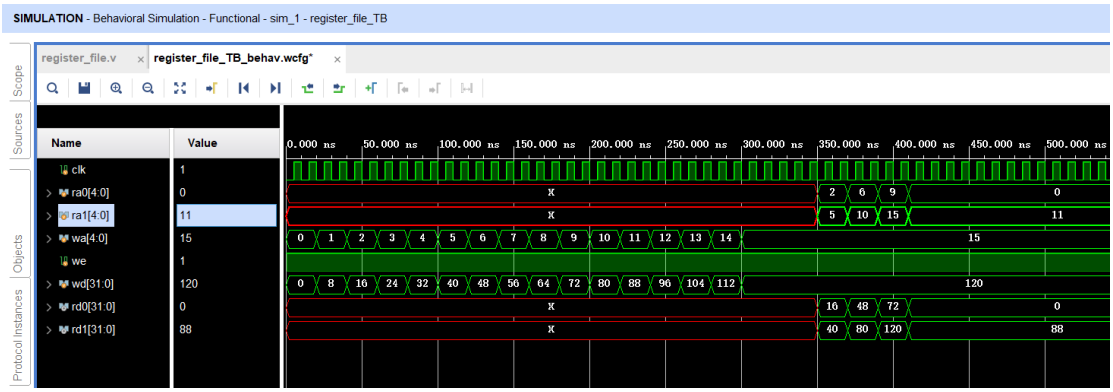
【实验练习】

1.行为方式参数化描述寄存器堆，功能仿真

如下为寄存器堆模块对应的代码：

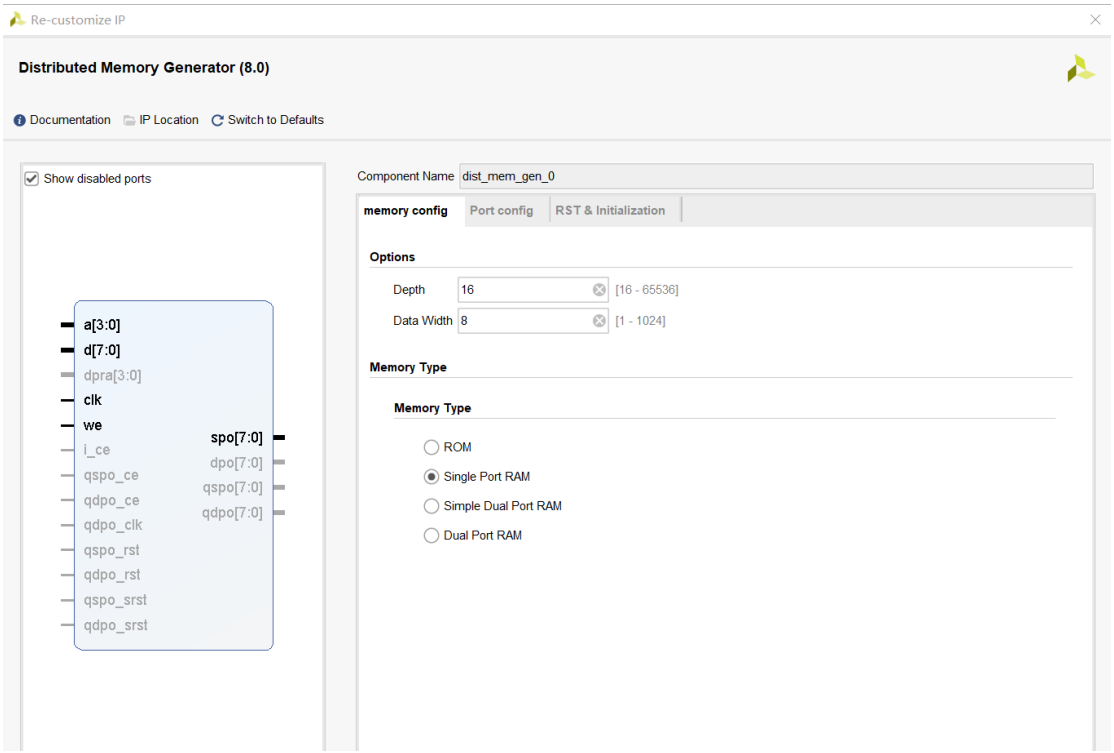
```
module register_file
#(parameter WIDTH = 32)
(input clk,
input [4:0]ra0,
input [4:0]ra1,
input [4:0]wa,
input we,
input [WIDTH - 1:0]wd,
output [WIDTH - 1:0]rd0,
output [WIDTH - 1:0]rd1);

reg [WIDTH - 1:0] ram [15:0];

assign rd0 = ram[ra0];
assign rd1 = ram[ra1];
always@(posedge clk)
begin
    if(we)
        ram[wa] <= wd;
end
endmodule
```

如下为仿真结果：



2. IP 例化分布式和块式 16x8 位单端口 RAM，功能仿真和对比

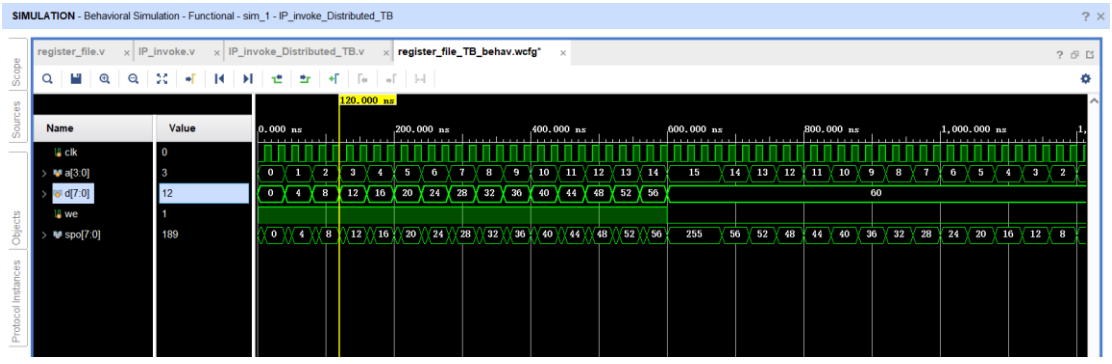
分布式 RAM 例化与功能仿真：



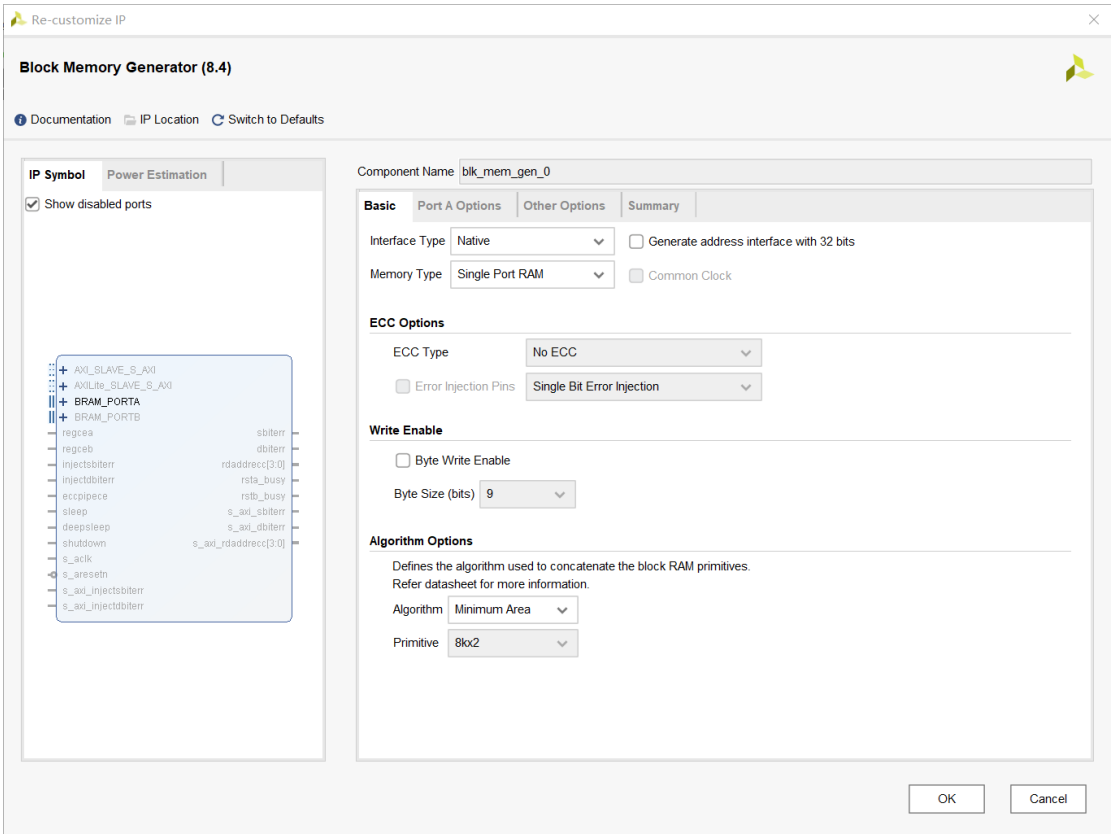

```
1.  module IP_invoke_Distributed(
2.  input clk,
3.  input [3:0]a,
4.  input [7:0]d,
5.  input we,
6.  output [7:0]spo);
7.  dist_mem_gen_0 IP_invoke_Distributed(
8.  .a(a),
9.  .d(d),
10. .clk(clk),
```

```
11. .we(we),
12. .spo(spo));
13. endmodule
```

```
1.  module IP_invoke_Distributed_TB();
2.  reg clk;
3.  reg [3:0]a;
4.  reg [7:0]d;
5.  reg we;
6.  wire [7:0]spo;
7.  IP_invoke_Distributed IP_invoke_Distributed(clk, a, d, we, spo);
8.  initial
9.  begin
10.     clk = 0; a = 0; we = 1; d = 0;
11.     #600 we = 0;
12. end
13. always #10 clk = ~clk;
14. initial
15. begin
16.     repeat(15)
17.     begin
18.         #40 a = a + 1;
19.         d = d + 4;
20.     end
21.     #40
22.     repeat(15)
23.         #40 a = a - 1;
24. end
25. endmodule
```



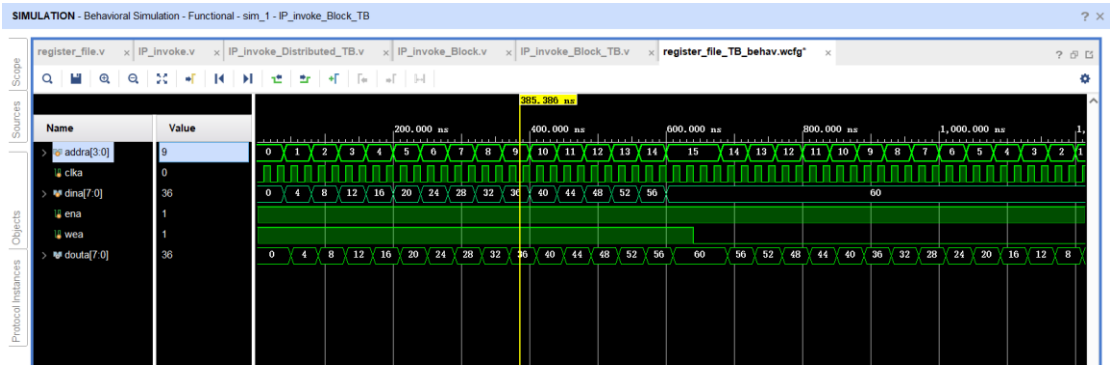
块式 RAM 例化与功能仿真：




```
1.  module IP_invoke_Block(
2.  input [3:0]addra,
3.  input clka,
4.  input [7:0]dina,
5.  input ena,
6.  input wea,
7.  output [7:0]douta);
8.  blk_mem_gen_0 IP_invoke_Block(
9.  .addra(addra),
10. .clka(clka),
11. .dina(dina),
12. .douta(douta),
13. .ena(ena),
14. .wea(wea));
15. endmodule
```

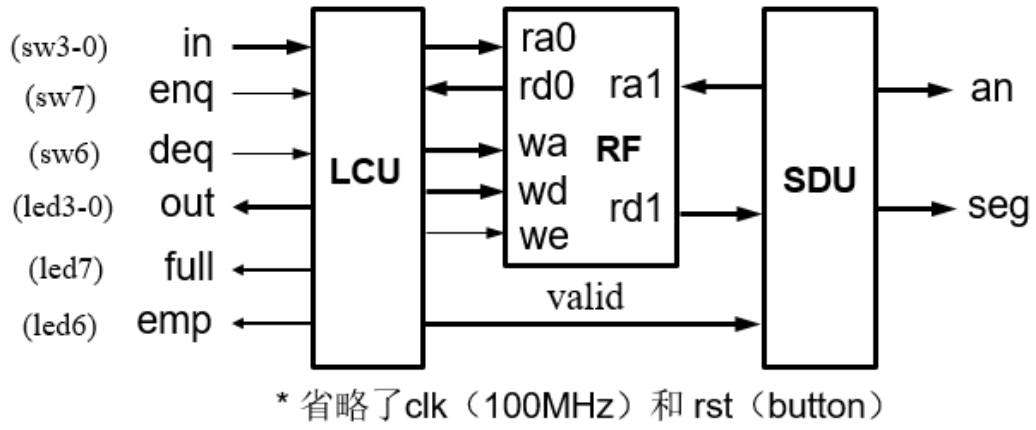
```
1.     module IP_invoke_Block_TB();
2.  reg [3:0]addra;
3.  reg clka;
4.  reg [7:0]dina;
5.  reg ena;
6.  reg wea;
```

```
7.  wire [7:0]douta;
8.  IP_invoke_Block IP_invoke_Block(addra, clka, dina, ena, wea, douta);
9.
10. initial
11. begin
12.     clka = 0; ena = 1; wea = 1; addra = 0; dina = 0;
13.     #640 ena = 1; wea = 0;
14. end
15.
16. always #10 clka = ~clka;
17. initial
18. begin
19.     repeat(15)
20.     begin
21.         #40 addra = addra + 1;
22.         dina = dina + 4;
23.     end
24.     #40
25.     repeat(15)
26.         #40 addra = addra - 1;
27. end
28.
29. endmodule
```



3. 设计 FIFO 队列电路的数据通路和控制器，结构化方式描述数据通路，Moore 型 FSM 描述控制器

FIFO 队列电路的数据通路：



//封装 FIFO 模块

```
1.  module FIFO(
2.  input enq,          //入队使能，一次有效仅允许操作一项数据 sw7
3.  input deq,          //出队使能，一次有效仅允许操作一项数据 sw6
4.  input [3:0]in,      //入队数据 sw3-0
5.  input rst,          //复位信号 button
6.  input clk,
7.
8.  output [3:0]out,    //出队数据 led3-0
9.  output emp,         //队空信号 led6
10. output full,        //队满信号 led7
11. output [2:0]an,     //数码管选择
12. output [3:0]seg); //数码管数据
13.
14. wire [2:0]ra0,ra1,wa;
15. wire [3:0]rd0,rd1,wd;
16. wire we;
17. wire [7:0]valid;  //有效标志
18. LCU LCU(.clk(clk),.rst(rst),.in(in),.enq(enq),.deq(deq),.out(out),.full(full
    ),.emp(emp),.ra0(ra0),.rd0(rd0),.wa(wa),.wd(wd),.we(we),.valid(valid));
19. RF RF(.clk(clk),.ra0(ra0),.rd0(rd0),.wa(wa),.wd(wd),.we(we),.ra1(ra1),.rd1(r
    d1));
20. SDU SDU(.clk(clk),.ra1(ra1),.rd1(rd1),.an(an),.seg(seg),.valid(valid));
21.
22. endmodule
```

//RF 模块

```
1.  module RF(
2.   input we,                      //写使能
3.   input clk,
4.   input [2:0] wa, ra0, ra1,     //写端口地址，读端口 0 地址，读端口 1 地址
```

```
  input [3:0] wd,               //写端口数据
  output [3:0] rd0, rd1         //读端口 0 数据，读端口 1 数据

  reg [3:0] regfile[0:7];
  assign rd0 = regfile[ra0];
  assign rd1 = regfile[ra1];
  always @(posedge clk)
  begin
      if(we) regfile[wa] <= wd;
  end
module
```

//LCU 模块

```
  module LCU(
input clk,
input rst,
input [3:0]in,            //入队数据
input enq, deq,
input [3:0]rd0,           //读端口 0 数据

output reg [3:0]out,      //出队列数据
output reg full, emp,     //队空与队满的标志
output [2:0]ra0,          //读端口 0 地址
output [2:0]wa,           //写端口地址
output reg [3:0]wd,       //写端口数据
output reg we,            //写使能信号
output reg [7:0]valid);   //有效标志，第 i 位对应 Ri 的状态

reg [2:0]cs, ns;          //当前状态，下一状态
reg [2:0]tail;            //队尾，指向入队列数据
reg [2:0]head;            //队头，指向出队列数据
assign ra0 = head;
assign wa = tail-1;

    always@(posedge clk)
        if (rst)   cs <= 3'b000;
        else   cs <= ns;


    always@(posedge clk)
        case(cs)
        3'b000:begin      //初始化
            valid <= 8'b0;
            tail <= 0;
```

```
32.             head <= 0;
33.             full <= 0;
34.             emp <= 0;
35.             we <= 0;
36.         end
37.         3'b001:begin       //入队一个数据
38.             wd <= in;
39.             valid[tail] <= 1;
40.             tail <= (tail + 1) % 8;
41.             full <= ((tail + 1) % 8 == head) & !emp;
42.             emp <= 0;
43.             we <= 1;
44.         end
45.         3'b010:begin        //出队一个数据
46.             out <= rd0;
47.             valid[head] <= 0;
48.             head <= (head + 1) % 8;
49.             emp <= (tail == (head + 1)) & !full;
50.             full <= 0;
51.             we <= 0;
52.         end
53.         default:
54.             we <= 0;
55.         endcase
56.
57.
58.     always@(*)
59.     begin
60.         case (cs)
61.         3'b000:begin
62.             if(enq & !full)       //当入队信号有效且队列不满时可以入队
63.                 ns = 3'b001;      //下一状态转到数据入队状态
64.             else if(deq &! emp)  //当出队信号有效且队列不空时可以出队
65.                     ns = 3'b010; //下一状态转到数据出队状态
66.                 else if((enq & full)|(deq & emp))  //当入队信号有效且队列满或当
    出队信号有效且队列空时
67.                     ns = 3'b011;
68.                     else ns = 3'b000;
69.         end
70.         3'b001:ns = 3'b011;
71.         3'b010:ns = 3'b011;
72.         3'b011:begin
73.             if(!enq &! deq)
74.                 ns = 3'b100;
```

```
            else
                ns = 3'b011;
        end
        3'b100:begin
            if((enq & full)|(deq & emp))
                ns = 3'b011;
            else if(enq & !full)
                ns = 3'b001;
                else if(deq  & !emp)
                    ns = 3'b010;
                else ns = 3'b100;
        end
        default:
            ns = 3'b000;
        endcase
    end

endmodule
```

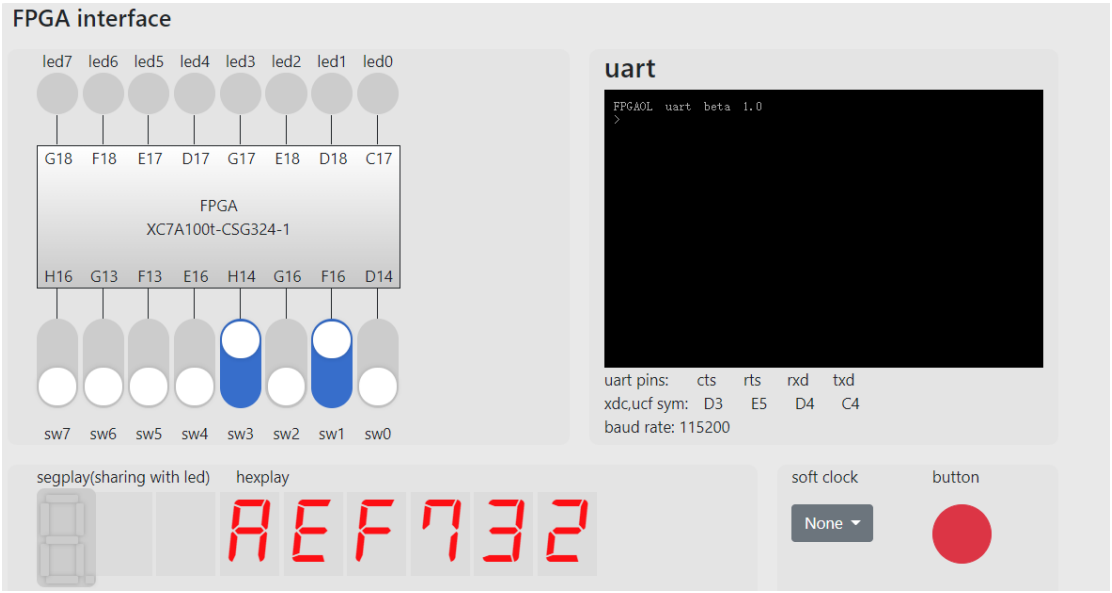
//SDU 模块

```
module SDU(
input clk,
input [3:0]rd1,        //读端口 1 数据
input [7:0]valid,      //有效标志
output reg [2:0]ra1,   //读端口 1 地址
output reg[2:0]an,     //数码管选择
output reg[3:0]seg);   //数码管数据
reg [9:0]cnt;

initial begin
    ra1 = 0;
    cnt = 0;
end

always@(posedge clk)
begin
    cnt <= cnt + 1;
    if(cnt == 0)begin
        ra1 <= ra1 + 1;
        if(valid[ra1])begin
            an <= ra1;
            seg <= rd1;
        end
        else begin
```

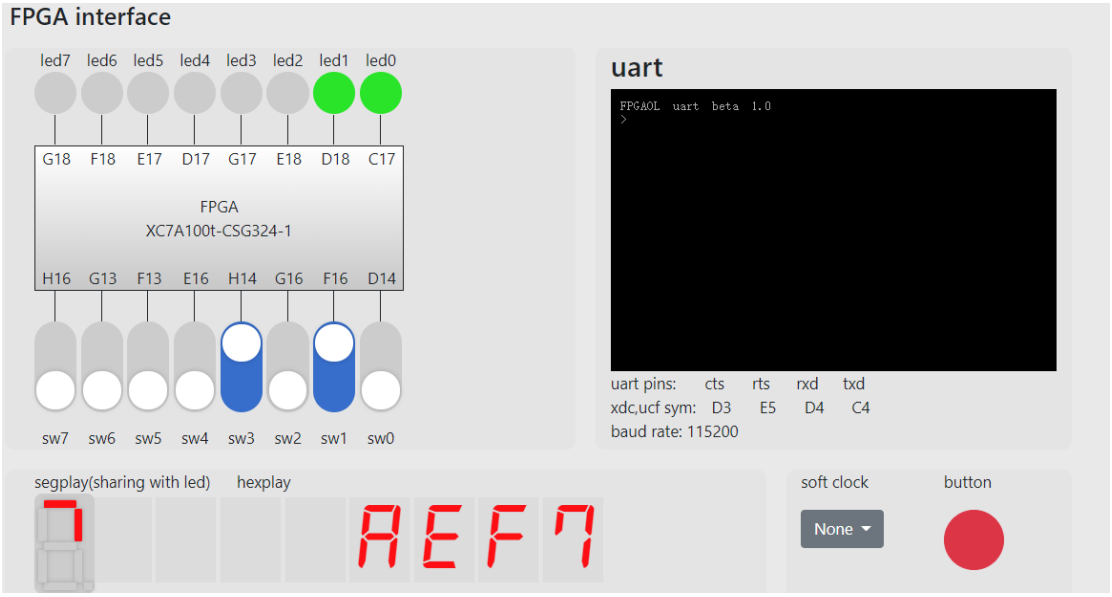
```
            an <= an;
            seg <= seg;
        end
    end
end
endmodule
```

4.FIFO 队列电路下载至 FPGA 中测试

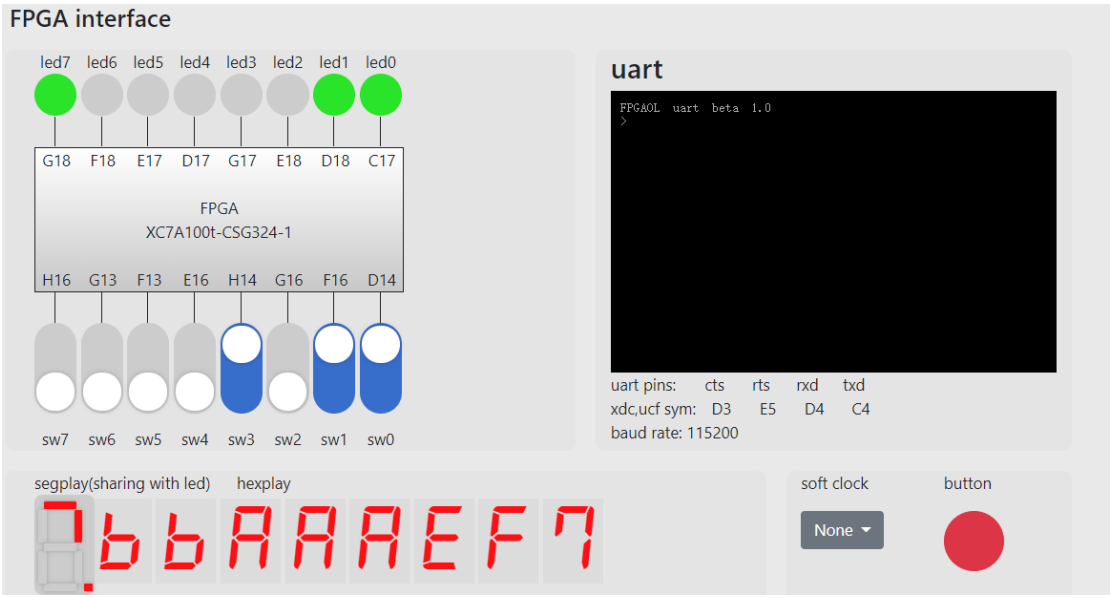
数据入队：



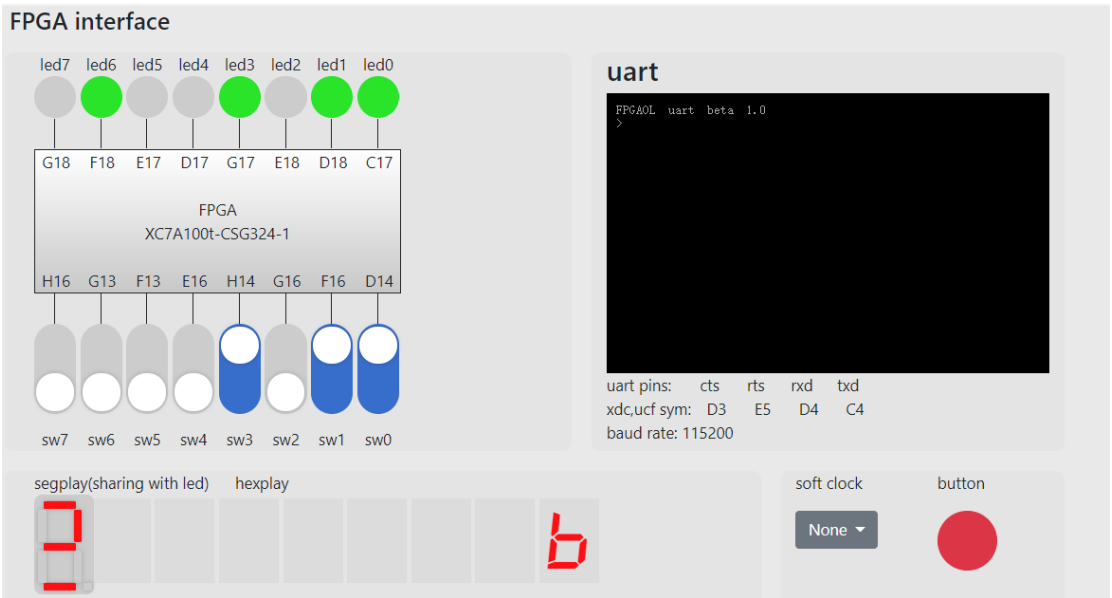
数据出队：



队满指示：



队空指示：



【总结与思考】

通过本次实验，我利用 FPGAOL 平台进行实验，并例化使用 IP 核，实现了 FIFO 队列。本次实验难度不大，根据实验说明的指导就可以完成实验操作，实验题目的难度层层递进，有基础操作的考核，也有所学知识的综合，难易结合，既有复习又有思考，让所学在实践中得

以运用，加深了我对逻辑电路知识的理解。希望今后实验可以保持本次实验中详细实验指导描述的优点，辅助完成每项试验内容。