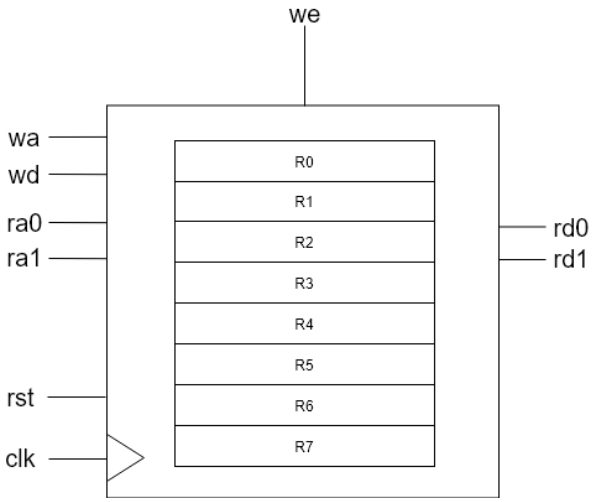
实验三 寄存器堆与计数器

PB17111623 范睿

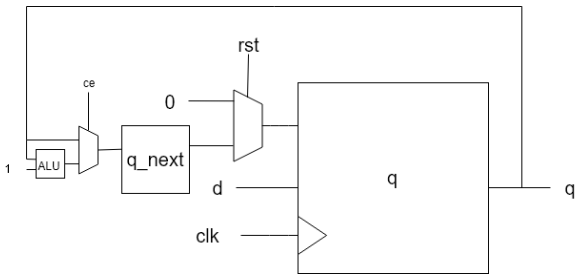
逻辑设计

寄存器堆RF

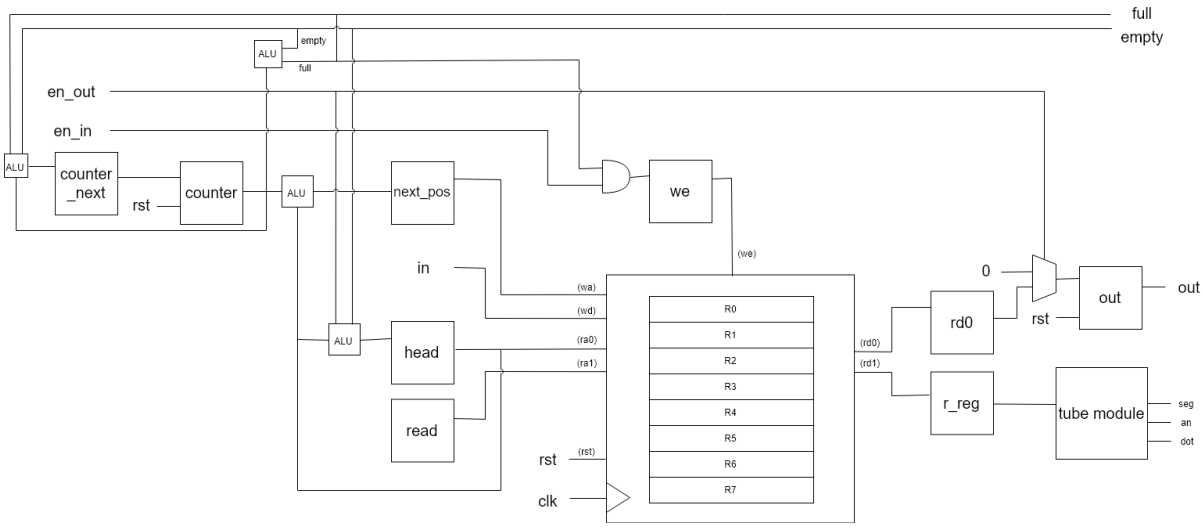
数据通路：



计数器



循环队列：



核心代码

计数器

```
module Counter(
    input [2:0]d,
    input pe, //同步装数使能
    input ce, //计数器使能
    input clk,
    input rst,
    output reg [2:0]q
    );
    reg [2:0]q_next;
    always@(posedge clk or posedge rst)begin    //每当时钟上升沿到来时
        if(rst) q = 3'b0;   //若rst有效，q置为0
        else q = q_next;    //否则将q_next赋给q
    end
    always@(*)begin //更新q_next
        if(ce) q_next = q + 3'b001;//若计数使能有效，q_next为q+1
        else if(pe) q_next = d;//若同步装数使能有效，q_next为d
        else q_next = q;//否则q值不改变
    end
endmodule
```

循环队列：

```
    assign head_next = (head + 4'b0001)%4'b1000;//head_next为下一个头指针的位置
//RF为寄存器堆模块（具体解释见下）
    RF
R(.ra0(head),.ra1(read),.wa(next_pos),.we(we),.clk(clk),.rst(rst),.wd(in),.rd0(rd0),.rd1
(r_seg));
//更新we
    always@(*)begin
        if(en_in&&!full)we = 1;//当en_in为1且队列没满时，将寄存器堆的写使能置为1
        else we = 0;
    end
//更新out，out为出队的数字
    always@(posedge clk or posedge rst)begin
        if(rst) out = 4'b0;
        else begin
            if(en_out) out = rd0;
            else out = 4'b0;
        end
    end
 //更新队头指针
    always@(posedge clk or posedge rst)begin
        if(rst)begin
            head = 4'b0000;
        end
        else begin
            if(!empty)begin //队列空时忽略出
                if(en_out)begin//只有出队了才更新队头指针
                    head = head_next;
                end
            end
        end
    end
```

```
//更新counter，记录此时队列中有多少个数字
    always@(posedge clk or posedge rst)begin
        if(rst)begin
            counter = 4'b0000;
        end
        else begin
            counter = counter_next;
        end
    end
//更新next_pos，next_pos表示队尾
    always@(*)begin
        if(!full) begin//若没满才更新
            if(head + counter > 4'b0111) next_pos = head + counter - 4'b1000;
            else next_pos = head + counter;
        end
    end
//更新counter_next，counter的下一个值
    always@(*)begin
        if(en_in)begin//若入队，counter_next=counter+1
            if(!full) counter_next = counter + 4'b0001;
            else counter_next = counter;
        end
        if(en_out)begin//若出队，counter_next=counter-1
            if(!empty) counter_next = counter - 4'b0001;
            else counter_next = counter;
        end
    end
//判断队满，full
    assign full = (counter == 4'b1000);//队列中有8个数时队满
//判断对空
    assign empty = (counter == 4'b0000);//队列中有0个数时队满
```

- RF模块输入输出解释：

  输入：

  ra0：head，队头地址

  ra1：read，当前数码管位选地址，数码管显示用

  wa：next_pos，下一个写入的位置

  we：we，写使能

  wd：in，入队元素

  输出：

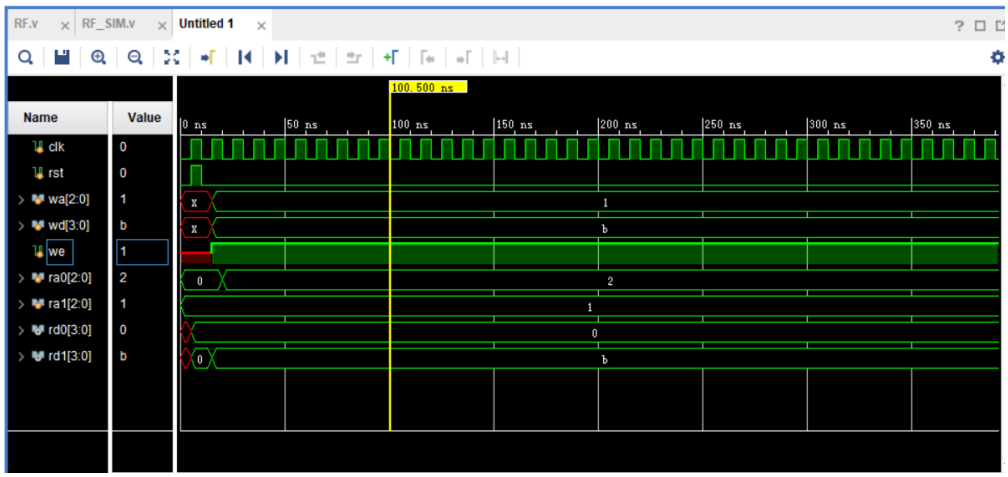
  rd0：rd0，出队元素

  rd1：r_reg，当前位选地址的数字（显示到数码管上）

仿真及下载

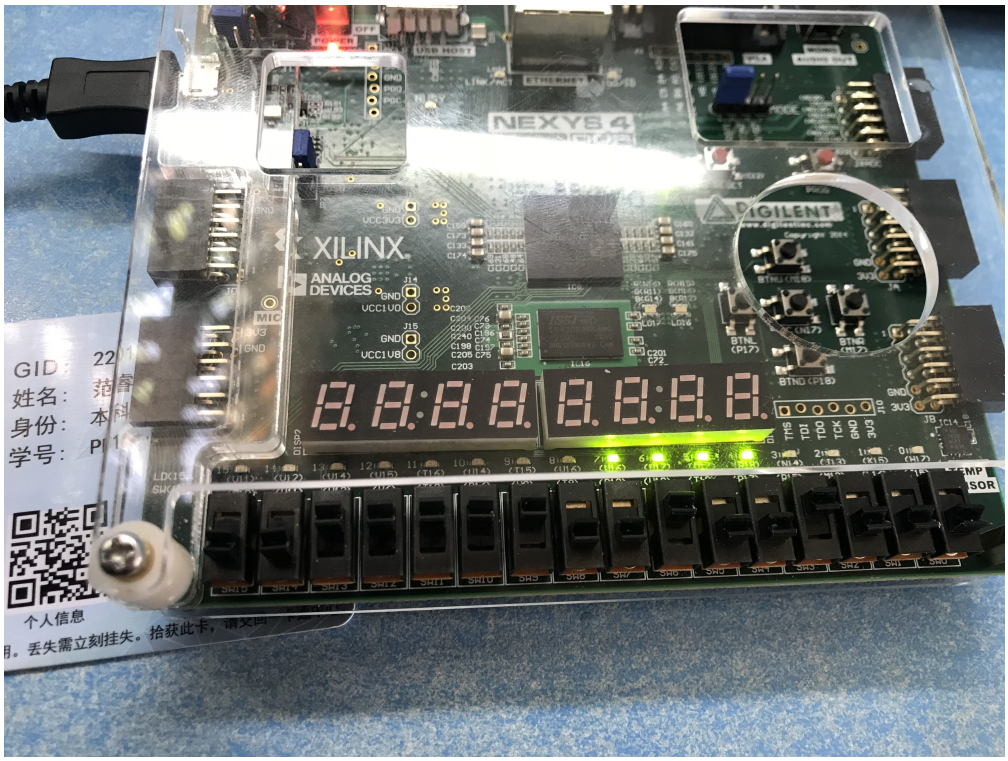
RF：

仿真：



t=15ns时，向地址1中写入d

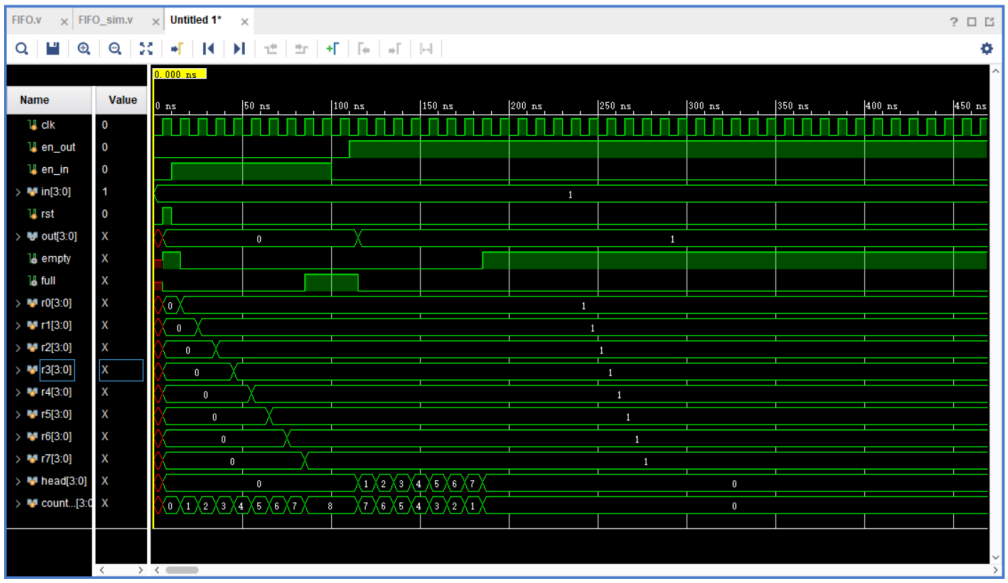
同时读地址1和地址2



- s[0]-s[2] ~ ra0[0]-ra0[2]
- s[3]-s[5] ~ ra1[0]-ra1[2]
- s[6]-s[8] ~ wa[0]-wa[2]
- s[9]-s[12] ~ wd[0]-wd[2]
- s[13] ~ we
- s[14] ~rst
- s[15] ~ clk
- led[0]-led[3] ~ rd0[0]-rd0[3]
- led[4]-led[7] ~ rd1[0]-rd1[3]
- 此时向地址1写入了1111，读地址0和地址1

FIFO:

仿真：



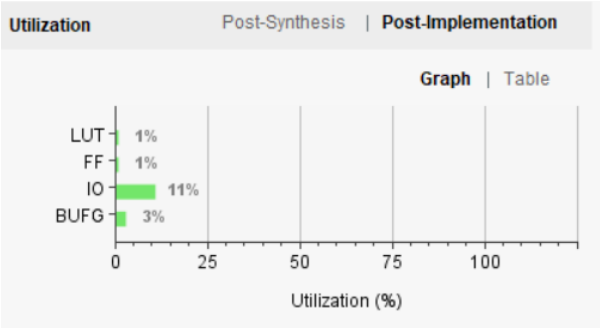
- 10ns时，入队有效，1一直入队，直到队满（full=1）
- 110ns时，出队有效，1一直出队（out一直为1），直到对空（empty=1）
- 最后队中无元素，head和counter都为0

下载：

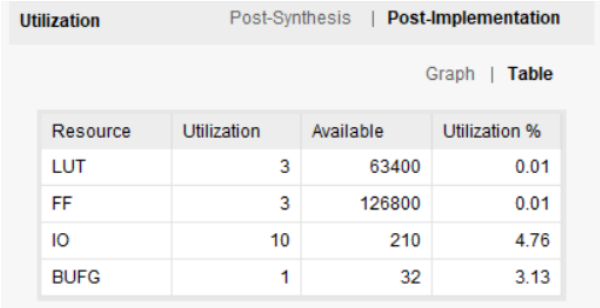
- s[0]-s[3] ~ in[0]-in[3]
- s[12] ~ en_out
- s[13] ~ en_in
- s[14] ~ rst
- s[16] ~clk
- led[0]-led[3] ~ out[0]-out[3]

电路性能及资源使用情况

RF:



counter:



Resource	Utilization	Available	Utilization %
LUT	3	63400	0.01
FF	3	126800	0.01
IO	10	210	4.76
BUFG	1	32	3.13

实验总结

在本次实验中，我实现了寄存器堆模块，计数器模块和基于寄存器堆模块的循环队列，复习了数码管的书写。本次实验给我最大的感受是：在大工程中，把任务分给不同模块去做，每个模块只实现一个小功能，将接口处的参数传递对接好，代码效率会提高很多。本来我写循环队列的时候没有调用已经写好的的RF模块，但是经过同学指点，我改变了实现方法，调用了RF模块，一个口读出队元素，一个口读数码管显示元素，十分方便。

设计及仿真代码

RF设计代码：

```
module RF(
    input [2:0] ra0,//读的地址
    input [2:0] ra1,//读的地址
    input [2:0] wa,//写的地址
    input [3:0] wd,//写的数据
    input we,//写的使能
    input clk,
    input rst,
    output reg [3:0] rd0,//被读的数据
    output reg [3:0] rd1 //被读的数据
    );
    reg [3:0]r0;//寄存器堆
    reg [3:0]r1;
    reg [3:0]r2;
    reg [3:0]r3;
    reg [3:0]r4;
    reg [3:0]r5;
    reg [3:0]r6;
    reg [3:0]r7;
```

```
20
21      always@(posedge clk or posedge rst)begin //写模块
22          if(rst) begin
23              r0 <= 4'd0;
24              r1 <= 4'd0;
25              r2 <= 4'd0;
26              r3 <= 4'd0;
27              r4 <= 4'd0;
28              r5 <= 4'd0;
29              r6 <= 4'd0;
30              r7 <= 4'd0;
31          end
32          else begin
33              if(we)begin
34                  case(wa)
35                      3'b000:r0<=wd;
36                      3'b001:r1<=wd;
37                      3'b010:r2<=wd;
38                      3'b011:r3<=wd;
39                      3'b100:r4<=wd;
40                      3'b101:r5<=wd;
41                      3'b110:r6<=wd;
42                      3'b111:r7<=wd;
43                  endcase
44              end
45          end
46      end
47
48      always@(*)begin //读模块
49          case(ra0)
50              3'b000:rd0<=r0;
51              3'b001:rd0<=r1;
52              3'b010:rd0<=r2;
53              3'b011:rd0<=r3;
54              3'b100:rd0<=r4;
55              3'b101:rd0<=r5;
56              3'b110:rd0<=r6;
57              3'b111:rd0<=r7;
58          endcase
59          case(ra1)
60              3'b000:rd1<=r0;
61              3'b001:rd1<=r1;
62              3'b010:rd1<=r2;
63              3'b011:rd1<=r3;
64              3'b100:rd1<=r4;
65              3'b101:rd1<=r5;
66              3'b110:rd1<=r6;
67              3'b111:rd1<=r7;
68          endcase
69      end
70
71  endmodule
```

RF仿真代码：

```
1   module RF_SIM(
2
3       );
4
5       reg clk;
```

```
    reg rst;
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
    initial begin
        rst = 0;
        #5 rst = 1;
        #5 rst = 0;
    end
    reg [2:0]wa;
    reg [3:0]wd;
    reg we;
    reg [2:0]ra0;
    reg [2:0]ra1;
    wire [3:0]rd0;
    wire [3:0]rd1;
    initial begin
        ra0 = 3'b000;
        ra1 = 3'b001;
        #20 ra0 = 3'b001;
        ra0 = 3'b010;
    end
    initial begin
        #15 wa = 3'b001;
        wd = 4'b1011;
        we = 1;
    end
    RF
L(.ra0(ra0),.ra1(ra1),.wa(wa),.wd(wd),.we(we),.clk(clk),.rst(rst),.rd0(rd0),.rd1(rd1));
endmodule
```

counter设计代码：

```
module Counter(
    input [2:0]d,
    input pe,
    input ce, //计数器使能
    input clk,
    input rst,
    output reg [2:0]q
    );
    reg [2:0]q_next;
    always@(posedge clk or posedge rst)begin
        if(rst) q = 3'b0;
        else q = q_next;
    end
    always@(*)begin
        if(ce) q_next = q + 3'b001;
        else if(pe) q_next = d;
        else q_next = q;
    end
endmodule
```

FIFO设计代码：

```
module FIFO(
    input clk_100M,
    input en_out,
    input en_in,
    input [3:0]in,
```

```
    input rst,
    input clk,
    output reg [3:0] out,
    output empty,
    output full,
    output [6:0]seg,
    output reg [7:0]an,
    output dot
    );
    wire [3:0] rd0,rd1;
    wire [3:0]r_seg;
    reg we;
    reg [3:0]head; //队头
    reg [3:0]counter; //队中数字个数
    reg [3:0]next_pos;
    wire [3:0]head_next;
    reg [3:0]counter_next;
    reg [3:0]read;
    assign head_next = (head + 4'b0001)%4'b1000;
    RF
R(.ra0(head),.ra1(read),.wa(next_pos),.we(we),.clk(clk),.rst(rst),.wd(in),.rd0(rd0),.rd
1(r_seg));
    always@(*)begin //更新we
        if(en_in&&!full)we = 1;
        else we = 0;
    end
    always@(posedge clk or posedge rst)begin //更新out
        if(rst) out = 4'b0;
        else begin
            if(en_out) out = rd0;
            else out = 4'b0;
        end
    end
    always@(posedge clk or posedge rst)begin //更新队头指针
        if(rst)begin
            head = 4'b0000;
        end
        else begin
            if(!empty)begin //空时忽略出
                if(en_out)begin
                    head = head_next;
                end
            end
        end
    end
    always@(posedge clk or posedge rst)begin //更新counter
        if(rst)begin
            counter = 4'b0000;
        end
        else begin
            counter = counter_next;
        end
    end
    always@(*)begin //更新next_pos
        if(!full) begin
            if(head + counter > 4'b0111) next_pos = head + counter - 4'b1000;
            else next_pos = head + counter;
        end
    end
    always@(*)begin
```

```
64          if(en_in)begin
65              if(!full) counter_next = counter + 4'b0001;
66              else counter_next = counter;
67          end
68          if(en_out)begin
69              if(!empty) counter_next = counter - 4'b0001;
70              else counter_next = counter;
71          end
72      end
73      assign full = (counter == 4'b1000);
74      assign empty = (counter == 4'b0000);
75
76      wire locked;
77      wire clk_5M;
78      reg clk_500;
79      reg [21:0]count;
80
81      //数码管:
82      clk_wiz_0 clk1(clk_5M,rst,locked,clk_100M);
83
84      always@(posedge clk_5M)begin
85          if(count>=21'd9999)begin
86              count<=0;
87          end
88          else begin
89              count<=count+1;
90          end
91      end
92      always@(posedge clk_5M)begin
93          clk_500<=(count>=21'd4999)?1:0;
94      end
95      initial begin
96          an = 8'b01111111;
97          count = 22'b0;
98      end
99      always@(posedge clk_500)begin
100         case(an)
101             8'b11111110:an<=8'b11111101;
102             8'b11111101:an<=8'b11111011;
103             8'b11111011:an<=8'b11110111;
104             8'b11110111:an<=8'b11101111;
105             8'b11101111:an<=8'b11011111;
106             8'b11011111:an<=8'b10111111;
107             8'b10111111:an<=8'b01111111;
108             8'b01111111:an<=8'b11111110;
109         endcase
110     end
111     always@(*)begin
112         case(an)
113             8'b11111110:read = 3'b111;
114             8'b11111101:read = 3'b110;
115             8'b11111011:read = 3'b101;
116             8'b11110111:read = 3'b100;
117             8'b11101111:read = 3'b011;
118             8'b11011111:read = 3'b010;
119             8'b10111111:read = 3'b001;
120             8'b01111111:read = 3'b000;
121         endcase
122     end
123     wire display;
```

```
    tube L1(.r_seg(r_seg),.an(an),.head(head),.counter(counter),.dot(dot),.seg(seg),.display(display));

endmodule
```

tube模块：

```
module tube(
    input [3:0]r_seg,
    input [7:0]an,
    input [3:0]head,
    input [3:0]counter,
    output dot,
    output reg [6:0]seg,
    output display
    );
    assign dot = ~(an[4'b0111-head]==0);
    reg [3:0]flag;
    wire [3:0]last;
    assign last = (head + counter - 4'b0001)%4'b1000;
    always@(*)begin
        case(an)
            8'b11111110:flag = 4'd7;
            8'b11111101:flag = 4'd6;
            8'b11111011:flag = 4'd5;
            8'b11110111:flag = 4'd4;
            8'b11101111:flag = 4'd3;
            8'b11011111:flag = 4'd2;
            8'b10111111:flag = 4'd1;
            8'b01111111:flag = 4'd0;
        endcase
    end
    assign display = (((head<=last)&&((head <= flag) && (flag<=last)))||((head > last)&&(flag>=head || flag <= last)))&&(counter != 4'b0);
    always@(*)begin
        if(display)begin
            case(r_seg)
                4'b0000:seg = 7'b1000000;
                4'b0001:seg = 7'b1111001;
                4'b0010:seg = 7'b0100100;
                4'b0011:seg = 7'b0110000;
                4'b0100:seg = 7'b0011001;
                4'b0101:seg = 7'b0010010;
                4'b0110:seg = 7'b0000010;
                4'b0111:seg = 7'b1111000;
                4'b1000:seg = 7'b0000000;
                4'b1001:seg = 7'b0010000;
                4'b1010:seg = 7'b0001000;
                4'b1011:seg = 7'b0000011;
                4'b1100:seg = 7'b1000110;
                4'b1101:seg = 7'b0100001;
                4'b1110:seg = 7'b0000110;
                4'b1111:seg = 7'b0001110;
            endcase
        end
        else seg = 7'b1111111;
    end
endmodule
```

仿真代码：

```
module FIFO_sim(

    );

    reg clk_100M;
    reg clk;
    reg en_out;
    reg en_in;
    reg [3:0] in;
    reg rst;
    wire [3:0] out;
    wire empty;
    wire full;
    wire [6:0]seg;
    wire [7:0]an;
    wire dot;

    initial begin
        clk_100M = 0;
        forever #1 clk_100M = ~clk_100M;
    end
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
    initial begin
        rst = 0;
        #5 rst = 1;
        #5 rst = 0;
    end
    initial begin
        en_out = 0;
        en_in = 0;
        in = 4'b0001;
        #10 en_in = 1;
        #10 en_in = 1;
        #10 en_in = 1;
        #10 en_in = 1;
        #10 en_in = 1;
        #10 en_in = 1;
        #10 en_in = 1;
        #10 en_in = 1;
        #10 en_in = 1;
        #10 en_in = 0;
        #10 en_out = 1;
        #10 en_out = 1;
        #10 en_out = 1;
        #10 en_out = 1;
        #10 en_out = 1;
        #10 en_out = 1;
        #10 en_out = 1;
        #10 en_out = 1;
    end
    FIFO L(.clk_100M(clk_100M),.en_out(en_out),.en_in(en_in),.in(in),.
rst(rst),.clk(clk),.out(out),.empty(empty),.full(full),.seg(seg),.an(an),.dot(dot));
endmodule
```