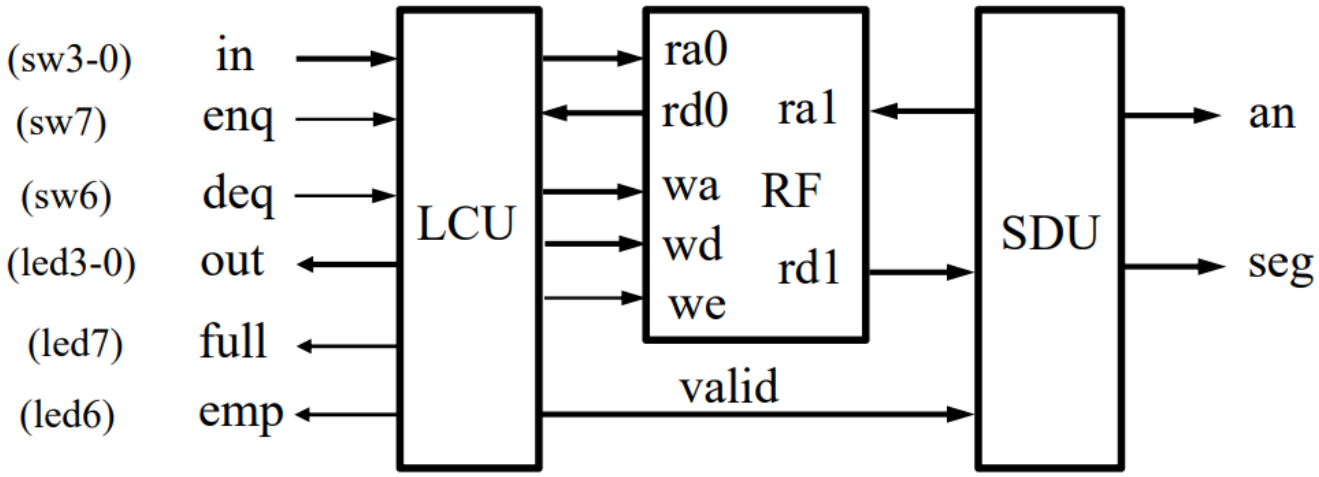
实验目的与内容

- 掌握寄存器堆（Register File）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
- 实现一个寄存器堆，并用它实现一个FIFO队列

逻辑设计

· 数据通路

采用PPT的数据通路，如图：



此外，还需要增加一条从LCU到SDU的线传输head，以及clk和rst接口。

· 寄存器堆RF

```
module RF
#(parameter WIDTH = 32) /
(   input clk,
    input[4 : 0] ra0,
    output[WIDTH - 1 : 0] rd0,
    input[4: 0] ra1,
    output[WIDTH - 1 : 0] rd1,
    input[4 : 0] wa,
    input we,
    input[WIDTH - 1 : 0] wd
);
reg [WIDTH - 1 : 0] regfile[0 : 31];
assign rd0 = regfile[ra0], rd1 = regfile[ra1];
always @ (posedge clk) begin
    if (we && wa != 0) regfile[wa] <= wd;
    else if(!we) regfile[wa] <= regfile[wa];
    else regfile[wa] <= 0;
```

```
end
endmodule
```

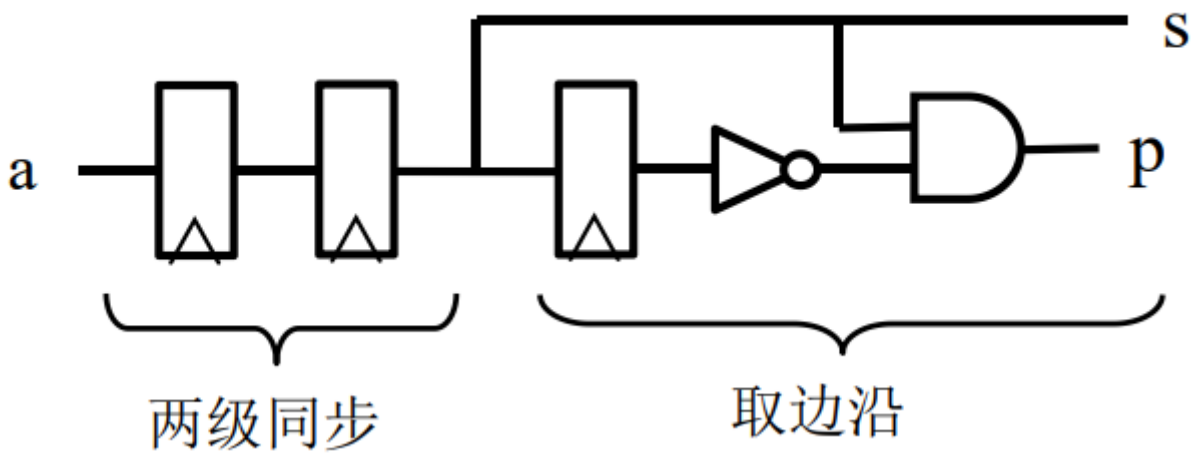
基本与PPT提供的一致，只在always块中用if-else进行写入判断：

- 当we有效且wa不是0时，写入数据；
- 当we无效时，不变；
- 当向x0写入数据时，写入0.

需要注意，we无效时应该不变，而不是写入0.

· 取边沿模块getEdge

数据通路如下：

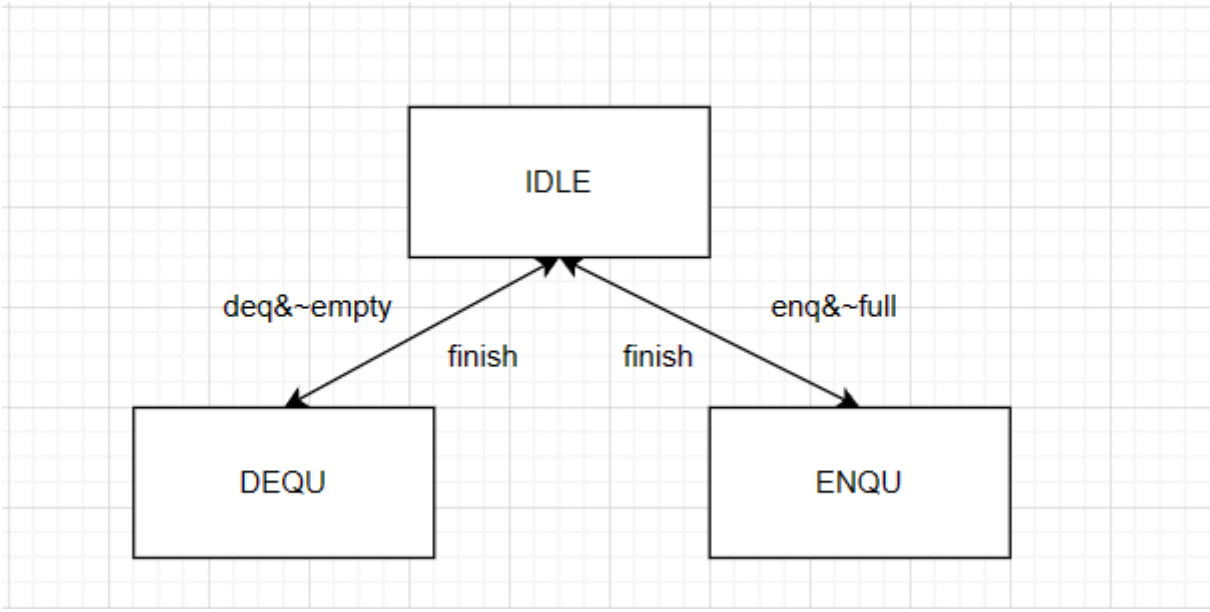


代码如下：

```
module getEdge(
    input clk,
    input in,
    output p,//edge
    output s //sychronized
    );
    reg r0, r1, r2;
    always @(posedge clk) begin
        r0 <= in;
        r1 <= r0;
        r2 <= r1;
    end
    assign s = r1;
    assign p = r1 & ~r2;
endmodule
```

· 列控制单元LCU

LCU有三个状态，分别为IDLE、DEQU、ENQU，状态转换关系如图:



为实现FIFO的控制，LCU需要两个寄存器分别存储队头和队尾，并在deq和enq信号的边缘从IDLE状态转移到相应状态，执行完队列操作后回到IDLE状态。

此外还需要进行full、empty、we等信号的判断。代码如下:

```
module LCU // depth = 8, width = 4
(
    input clk,
    input rst,
    input [3:0] in,
    input [3:0] rd,
    input enq_edge,
    input deq_edge,
    output full,
    output empty,
    output reg [3:0] out,
    output [2:0] ra,
    output [2:0] wa,
    output [3:0] wd,
    output we,
    output reg [7:0] valid,
    output head
    );

    assign full = &valid;
    assign empty = ~(|valid);

    reg [2:0] head;
    reg [2:0] tail;

    assign ra = head;
    assign wa = tail;

    assign wd = in;
```

```
    assign we = enq_edge & ~full & ~rst;

    always @(posedge clk) begin
        if(rst) begin
            valid <= 8'b0;
            head <= 3'b0;
            tail <= 3'b0;
            out <= 4'b0;
        end
        else if(enq_edge & ~full) begin
            valid[tail] <= 1'b1;
            tail <= tail + 1;
        end
        else if(deq_edge & ~empty) begin
            valid[head] <= 1'b0;
            head <= head + 1;
            out <= rd;
        end
        else begin
            head <= head;
            tail <= tail;
        end
    end

endmodule
```

· 数码管显示模块SDU

代码如下:

```
module SDU(
    input clk,
    input [3:0] data,
    input [7:0] valid,
    input [2:0] head,
    output reg [2:0] adrs,
    output [2:0] an,
    output [3:0] seg
    );
    // 100M Hz to 100 Hz
    wire clk_100;
    reg [19:0] clk_cnt;
    assign clk_100 = ~(|clk_cnt);
    always @(posedge clk) begin
        if (clk_cnt >= 20'd99_9999) begin
            clk_cnt <= 20'd0;
            adrs <= adrs + 1;
        end
        else
            clk_cnt <= clk_cnt + 20'd1;
    end
```

```
    reg [2:0] temp_an;
    reg [3:0] temp_seg;
    always @(posedge clk_100) begin
        if(valid[adrs] == 1) begin
        temp_an <= adrs - head;
        temp_seg <= data;
        end
        else begin
        temp_an <= temp_an;
        temp_seg <= temp_seg;
        end
    end
    assign an = (|valid)?temp_an:0;
    assign seg = (|valid)?temp_seg:0;
endmodule
```

首先将默认100Mhz时钟分频，用于分时复用。

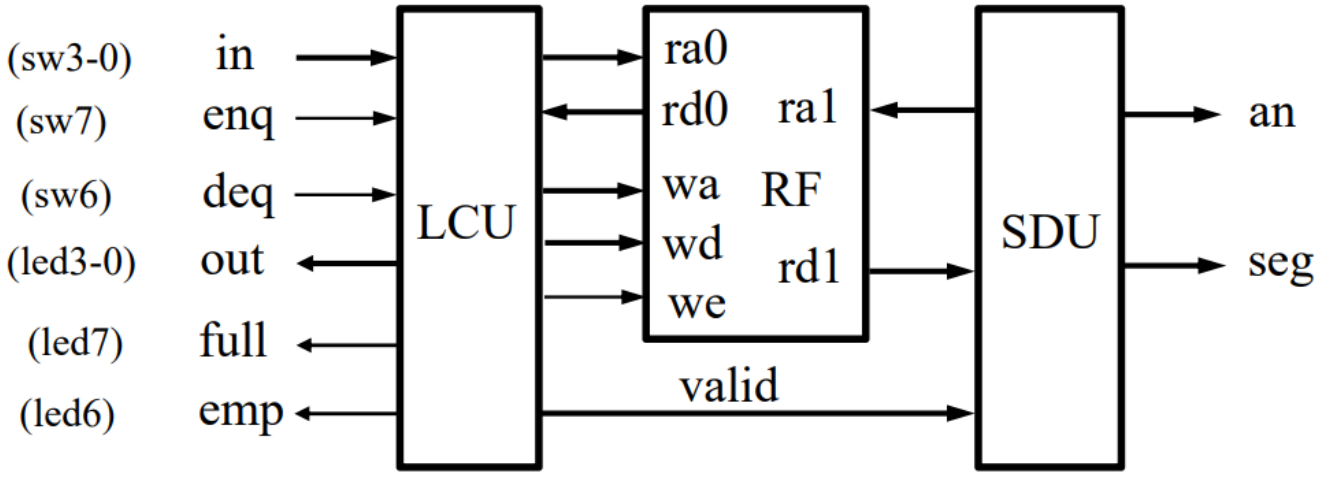
其次根据数据地址ards和队头地址head的差值，决定数码管编号an。

最后将传入的数据data赋给数码管seg。

此外，要注意valid的限制。

· 顶层模块FIFO

采用PPT的数据通路，如图：



此外，还需要增加一条从LCU到SDU的线传输head，以及clk和rst接口。

按照数据通路连线，代码如下：

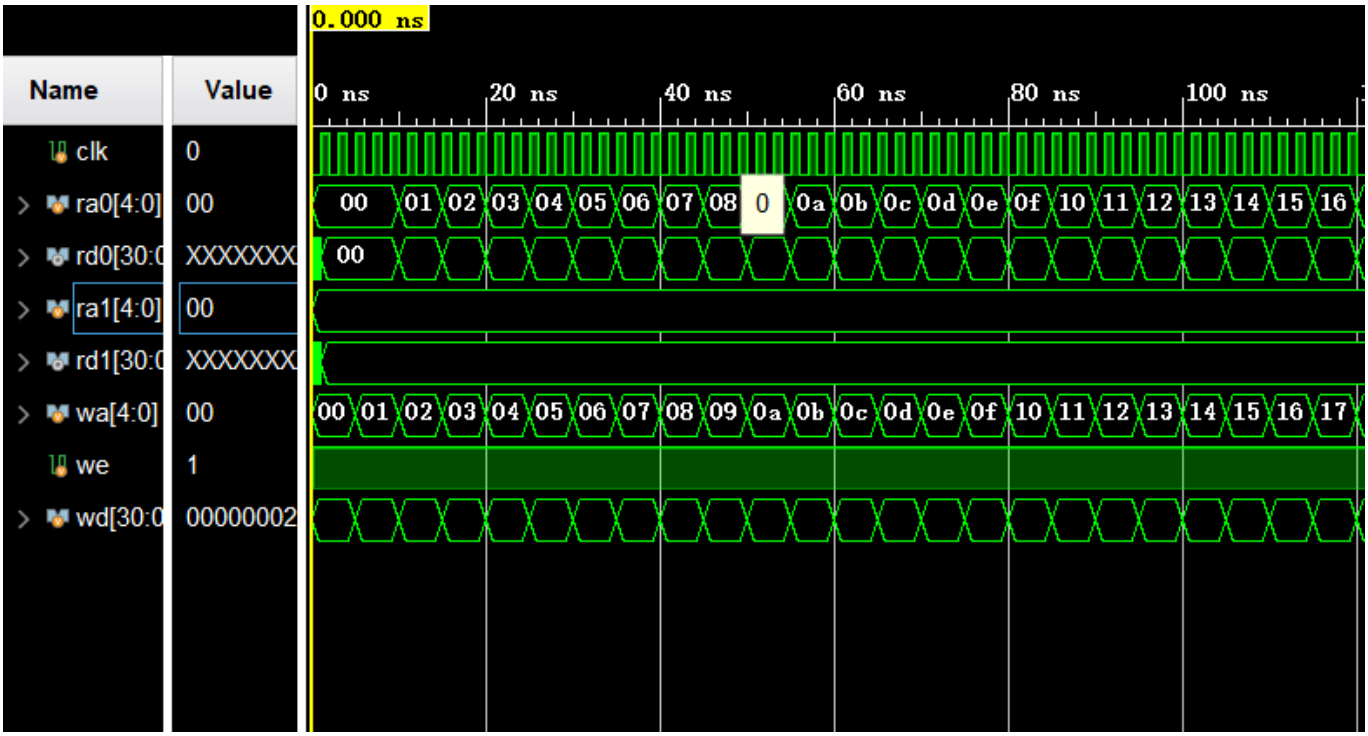
```
module FIFO(
    input clk,
    input rst,
    input enq,
    input deq,
    input [3:0] in,
```

```
    output [3:0] out,
    output empty,
    output full,
    output [2:0] an,
    output [3:0] seg
    );

    wire enq_edge;
    wire deq_edge;
    getEdge edgeEnq(
        .clk(clk),
        .in(enq),
        .p(enq_edge)
    );
    getEdge edgeDeq(
        .clk(clk),
        .in(deq),
        .p(deq_edge)
    );

    wire we;
    wire [2:0] ra0, wa, ra1;
    wire [3:0] rd0, wd, rd1;
    wire [7:0] valid;
    wire [2:0] head;

    LCU lcu(
        .clk(clk),
        .rst(rst),
        .in(in),
        .rd(rd0),
        .enq_edge(enq_edge),
        .deq_edge(deq_edge),
        .full(full),
        .empty(empty),
        .out(out),
        .ra(ra0),
        .wa(wa),
        .wd(wd),
        .we(we),
        .valid(valid),
        .head(head)
    );
    RF#(4) regFile(
        .clk(clk),
        .ra0(ra0),
        .ra1(ra1),
        .rd0(rd0),
        .rd1(rd1),
        .wa(wa),
        .wd(wd),
        .we(we)
    );
    SDU sdu(
```

```
        .clk(clk),
        .valid(valid),
        .data(rd1),
        .adrs(ra1),
        .an(an),
        .seg(seg),
        .head(head)
    );
endmodule
```

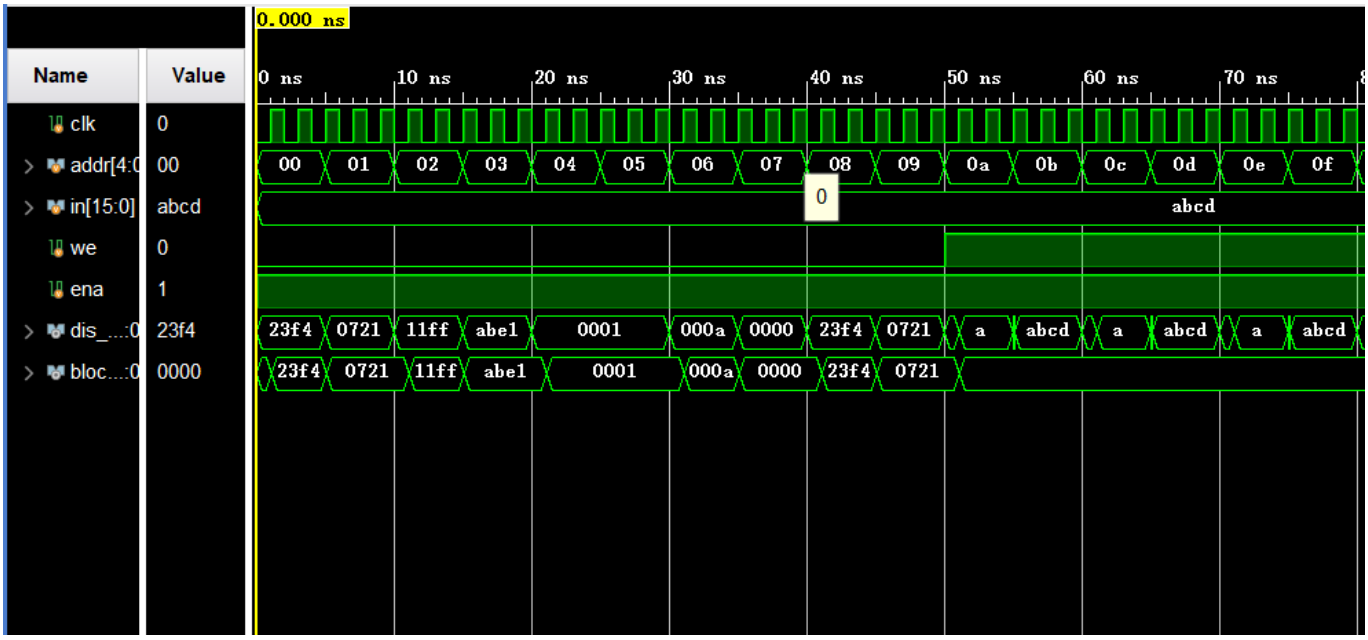
仿真结果与分析

· RF仿真



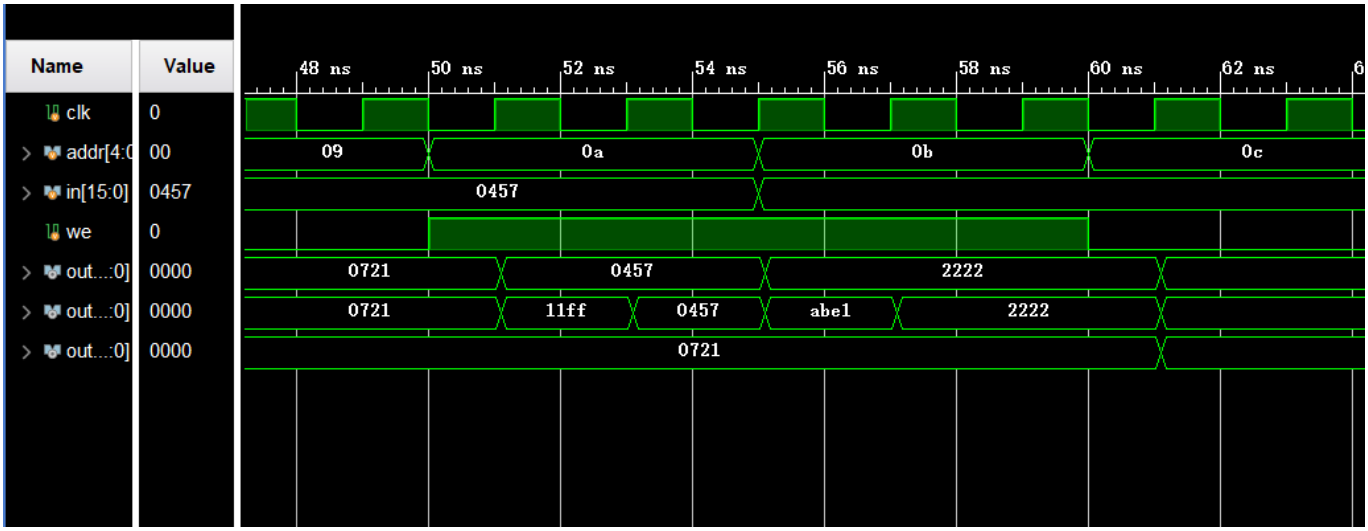
如图，向第x个寄存器写入x+2，随后读取，结果正确。尝试向第0个寄存器写入2，读取结果依然是0。

· DRAM和BRAM仿真



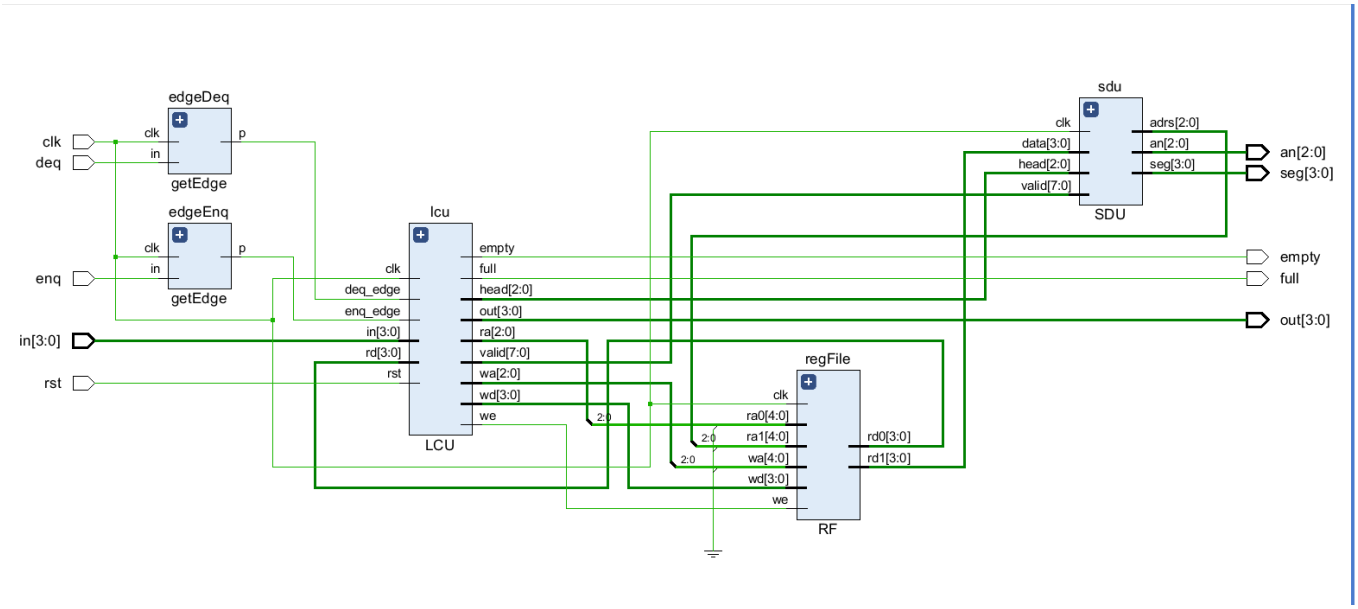
如图，DRAM立刻读取，BRAM等到时钟上升沿读取。写入都在时钟上升沿进行。

· BRAM不同模式仿真



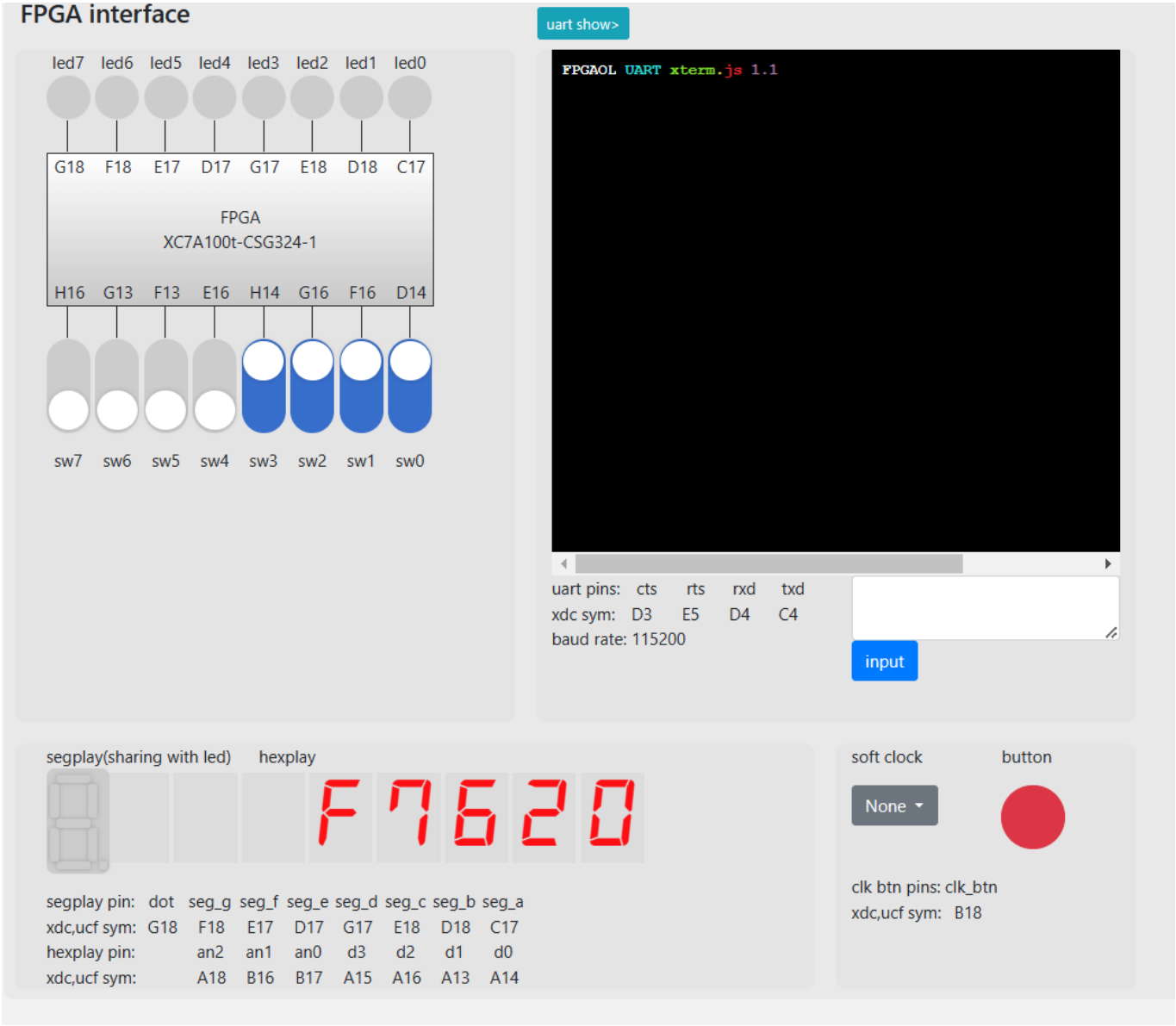
如图，在we无效时没有区别，在we有效时，write first模式会先显示写入的数据，read first会先显示读取到的数据，no change保持不变。

RTL分析



如图，与设计的数据通路基本一致，添加了取边沿模块。

测试结果



如图，第0个寄存器值总为0，其余寄存器按FIFO模式实现队列。

总结

- 本次实验实现了寄存器堆，应用它实现了一个FIFO队列，实验难度适中。
- 写寄存器堆时，对we无效时的行为理解有误，但是测试并没有反映出这个问题，以后需要更全面的考虑测试数据。