计算机组成原理 实验报告

姓名： 杨佳熹 学号：PB17000050 实验日期：2019-3-28

一、实验题目：

Lab03 寄存器堆与计数器

二、实验目的：

设计一个寄存器堆实现普通的读写，并且利用寄存器堆实现简易的队列，并把队列内容在数码管显示出来。

三、实验平台：

ISE / Vivado（暂不支持其他 Verilog HDL 开发环境的检查）

四、实验过程：

寄存器堆：此部分需要两个异步读端口和一个同步写端口，为实现同步操作，利用 posedge 和时钟进入 always 改变寄存器的值。同时用行为级实现读取操作。

代码：

```
module regfile(ra0,rd0,ra1,rd1,wa,wd,we,clk,rst);
parameter m=2;
parameter n=2;
input [m:0] ra0,ra1,wa;
input clk,rst,we;
input [n:0]wd;
output [n:0] rd0,rd1;
reg [m:0] i;
reg flag;
reg [n:0] register [n:0];
always @(posedge clk or posedge rst)
begin
    if(rst)
        begin
            flag=0;
            i=0;
        end
```

```
    else if(i<n)
        begin
            i=i+1;
            flag=0;
        end
     else
        flag=1;
end
always @(posedge clk)
begin
    if(flag==0)
        begin
            register[i]=0;
        end
    else
        begin
            if(we&&wa)
                begin
                    register[wa]=wd;
                end
        end
end
assign rd0= (ra0)? register[ra0]:0;
assign rd1= (ra1)? register[ra1]:0;
endmodule
```

Fifo：此部分利用寄存器堆，改变寄存器中的值，并用上学期用到的分屏显示，用 ip 制造出 5Mhz 的信号，利用视觉暂留原理显示出队列。

代码：

```
module seg_ctrl(
input [3:0]x,
input [7:0]sel,
output [7:0]an,
```

```
output reg [6:0]seg
);
assign an =sel;
always @(x)
begin
case(x)
4'b0000:seg=7'b0000001;
4'b0001:seg=7'b1001111;
4'b0010:seg=7'b0010010;
4'b0011:seg=7'b0000110;
4'b0100:seg=7'b1001100;
4'b0101:seg=7'b0100100;
4'b0110:seg=7'b0100000;
4'b0111:seg=7'b0001111;
4'b1000:seg=7'b0000000;
4'b1001:seg=7'b0000100;
4'b1010:seg=7'b1110111;
default:seg=7'b0110000;
endcase
end
endmodule
module
fifo(en_out,en_in,in,rst,clk,full,empty,out,i1,j1,an,seg,dp);
//tem0,tem1,tem2,tem3,tem4,tem5,tem6,tem7,i1,j1,
input en_out,en_in,rst,clk;
input [3:0]in ;
output reg full,empty;
output reg [3:0] out;
output reg [3:0] i1,j1;
output  [7:0]   an;
output  [6:0]   seg;
output reg dp;
```

```
reg flag;
reg [3:0] i,j,count;
reg [3:0]queue[7:0];
reg tag;
reg     [7:0]   seg_sel;
reg     [3:0]   seg_din;
reg     [31:0]  cnt,cnt5;
wire    clk_5m;
wire    rst_n;
reg fl;
reg inflag,outflag;
clk_wiz_0   clk_wiz_0(
.clk_in1    (clk),
.clk_out1   (clk_5m),
.reset      (rst),
.locked     (rst_n)
);
always @ (posedge clk_5m or posedge rst)
begin
    if(rst)
    begin
        i=0;
        j=0;
        tag=0;
        out=0;
        for(count=0;count<8;count=count+1)
        begin
            queue[count]=10;
        end
        inflag=1;
        outflag=1;
    end
```

```
    else if(en_in&&!full&&inflag)
        begin
        inflag=0;
        queue[i]=in;
        i=(i==7)? 0:i+1;
        tag=1;
        end
    else if(en_out&&!empty&&outflag)
        begin
        outflag=0;
        out=queue[j];
        queue[j]=10;
        j=(j==7)? 0:j+1;
        tag=0;
        end
    else if(!en_in || !en_out)
    begin
        if(!en_in)
        inflag=1;
        if(!en_out)
        outflag=1;
    end
end
//always @ (posedge clk)
//begin
//    if(flag==0)
//        queue[count]=10;
//    else
//    begin

//        j1=j;
//        i1=i;
```

```
//        tag=0;
//    end
//end

always @ (posedge clk_5m)
begin
    empty=0;
    full=0;
    if(i==j)
    begin
        if(tag==0)
        empty=1;
        if(tag==1)
        full=1;
     end
     i1=i;
     j1=j;
end
//always @ (posedge clk)
//begin
//    //f1=0;
//    if(en_in)
//    begin
//       // f1=1;
//        queue[i]=in;
//        i=i+1;
//    end
//    else if(en_out)
//    begin
//        //f1=1;
//        out=queue[j];
//        queue[j]=10;
```

```
//        j=j+1;
//    end
//end

//always @ (i or j)
//begin
//    if(i==8)
//        i=0;
//    else if (j==8)
//        j=0;
//end

//always @ ( posedge clk)
//begin
//    if(!full&&flag&&en_in)
//    begin
//    queue[i]=in;
//    tag=1;
//    if(i+1>7)
//        i=0;
//    else
//        i=i+1;
//    end
//    if(!empty&&flag&&en_out)
//    begin
//        out=queue[j];
//        queue[j]=10;
//        tag=0;
//        if(j+1>7)
//            j=0;
//        else
//            j=j+1;
```

```
//    end
//end

//always @ (posedge clk)
//begin
//    if(!empty&&flag&&en_out)
//    begin
//        out=queue[j];
//        queue[j]=10;
//        tag=0;
//        if(j+1>7)
//            j=0;
//        else
//            j=j+1;
//    end
//end



seg_ctrl  seg_ctrl(
 .x              (seg_din),
 .sel            (seg_sel),
 .an             (an),
 .seg            (seg)
 );

always@(posedge clk_5m or negedge rst_n)
 begin
    if(~rst_n)
        cnt <= 32'h0;
    else if(cnt<32'd5000000)
        cnt <= cnt + 1;
```

```
    else
        cnt <= 32'h0;
 end

always@(posedge clk_5m or negedge rst_n)
begin
    if(~rst_n)
    begin
        seg_sel <= 8'b0;
        seg_din <= 10;
    end
    else case(cnt[15:13])
        3'b000:
        begin
            seg_sel <= 8'b0111_1111;
            seg_din <= queue[0];
            if(j1==0)
                dp=0;
            else
                dp=1;
        end
        3'b001:
        begin
            seg_sel <= 8'b1011_1111;
            seg_din <= queue[1];
            if(j1==1)
                dp=0;
            else
                dp=1;
        end
        3'b010:
        begin
```

```
        seg_sel <= 8'b1101_1111;
        seg_din <= queue[2];
        if(j1==2)
            dp=0;
        else
            dp=1;
    end
    3'b011:
    begin
        seg_sel <= 8'b1110_1111;
        seg_din <= queue[3];
        if(j1==3)
            dp=0;
        else
            dp=1;
    end
    3'b100:
    begin
        seg_sel <= 8'b1111_0111;
        seg_din <= queue[4];
        if(j1==4)
            dp=0;
        else
            dp=1;
    end
    3'b101:
    begin
        seg_sel <= 8'b1111_1011;
        seg_din <= queue[5];
        if(j1==5)
            dp=0;
        else
```

```
                dp=1;
            end
            3'b110:
            begin
                seg_sel <= 8'b1111_1101;
                seg_din <= queue[6];
                if(j1==6)
                    dp=0;
                else
                    dp=1;
            end
            3'b111:
            begin
                seg_sel <= 8'b1111_1110;
                seg_din <= queue[7];
                if(j1==7)
                    dp=0;
                else
                    dp=1;
            end
            endcase
        end
        endmodule
```

五、实验结果：

寄存器堆：

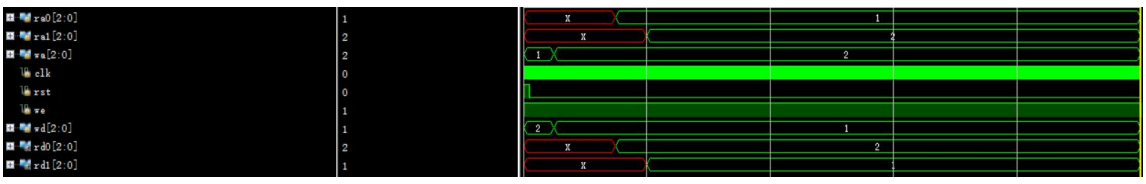

图 1：寄存器堆仿真结果

仿真中时钟频率设置为 0.02，并出示给一个 rst 时钟沿清零。始终进行读取操作结果如上。

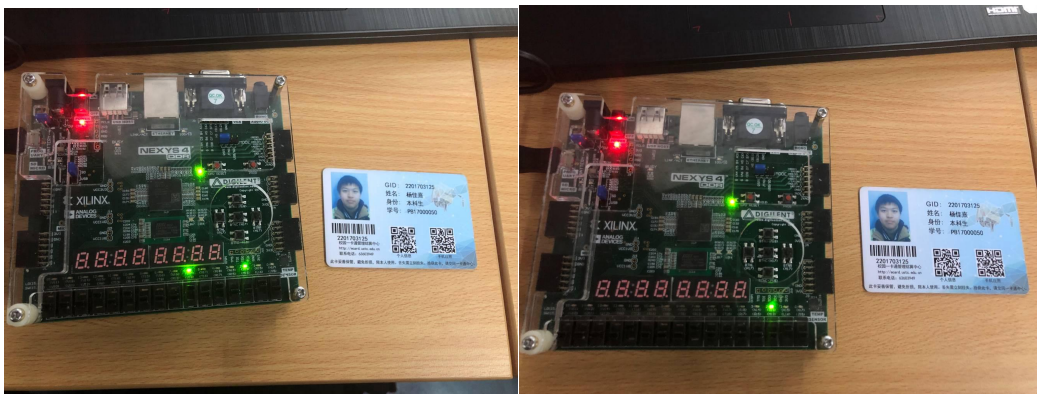

图 2：寄存器堆下载结果

图为 1 地址为四 2 地址也为四的下载结果。

Fifo：

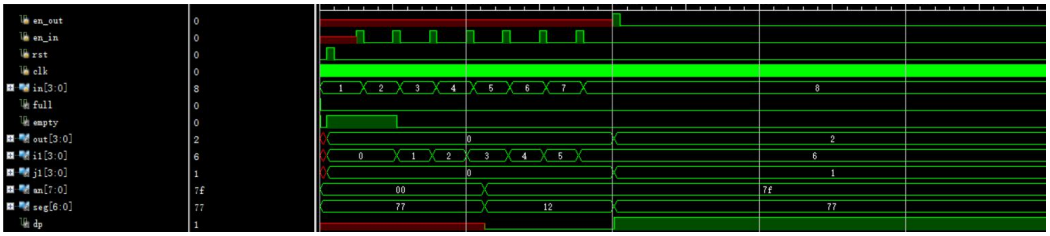

图 3：fifo 仿真结果

此部分为数码管输出，不好在仿真结果中显示，结果会在下载结果中显示。仿真只能看到 out 的结果。

图 4：fifo 下载结果

图 4 为初始状态，队列为空点指示队头，后为入队列一些数字并队头出队列小数点后移。

六、心得体会：

本次实验内容主要关于寄存器堆，第一个实验就是利用 verilog 编写一个普通的寄存器堆，为以后的写处理器的实验先做了一个铺垫。第二个实验为 fifo 是寄存器堆的一个小应用，基于上学期数据结构的队列，设出头尾指针并实现基本的出入队列操作，就能够完成。

本次实验遇到的问题大体上有两个，第一个是多次调用的问题，一开始以为设置 en_in 和 en_out 的上升沿进入 always 来对队列进行出入操作就看可以实现一次只能出入一次的要求，但是下载后就会出现问题。所以改进方法是两个操作

放入一个 always 内，并设置 flag 保证使能有效只能入入队列一次，具体操作为当使能为 1 时 flag 为 1 时进行出入队列操作并且设 flag=0，当使能失效时再将 flag 复原保证第二次使能还有效。第二个问题时钟调用问题，同时使用 ip 输出的时间和初始的时间会有问题，最后索性都改成 5m 的时间，问题就解决了