实验报告

实验名称：Lab3_寄存器堆与计数器

学生姓名： 田宏宇 学号：PB17111573

实验日期： 2019 年 4 月 12 日

一、实验目的

1、熟练Vivado和N4的设计实现流程

2、模块化、层次化、参数化设计方法

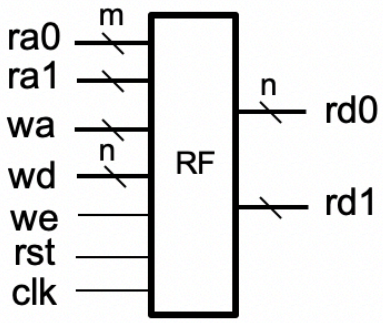
3、寄存器堆和计数器的描述方法

二、实验内容

1. 寄存器堆（Register File）

ra0, rd0; ra1, rd1：2个异步读端口

wa, wd, we：1个同步写端口

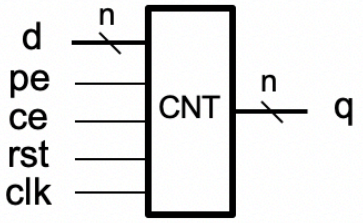


2. 计数器（Counter）

ce：计数使能，1: q=q+1

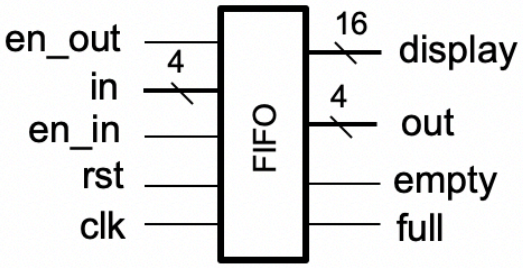
pe：同步装数使能，1: q=d

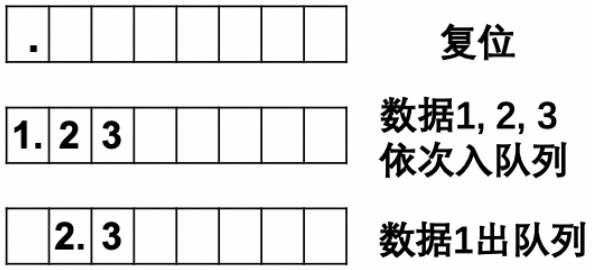
rst：异步清零，1: q=0



3. 最大长度为8的FIFO循环队列：用寄存器堆和适当逻辑实现

- **en_out, en_in**：出/入队列使能，一次有效仅允许操作一项数据
- **out, in**：出/入队列数据
- **full, empty**：队列空/满，空/满时忽略出/入队操作
- **display**：8个数码管的控制信号，显示队列状态





三、实验设计

1.寄存器堆：

定义2^m个n位的寄存器，在每次时钟沿到来的时将wd写入对应wa地址的寄存器，将rd0和rd1定义为wire型变量，通过assign赋值读取地址ra0和ra1对应数据，实现异步读。

3.FIFO循环队列：

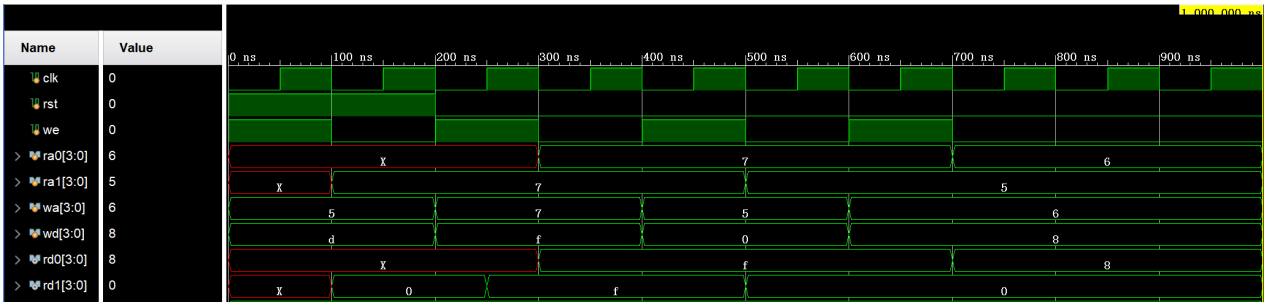
定义8个4位的寄存器，实现一个异步读端口和一个同步写端口，定义写指针pw和读指针pr，当队列非满时，可以入队，pw移动到下一个位置；当队列非空时，可以出队，pr移动到下一个位置。当pr==pw时，若刚进行完入队操作，则队列已满full=1；若刚进行完出队操作，则队列已空empty=1；将八个寄存器中的数据通过读端口实时显示在八个七段数码管上。

四、实验过程

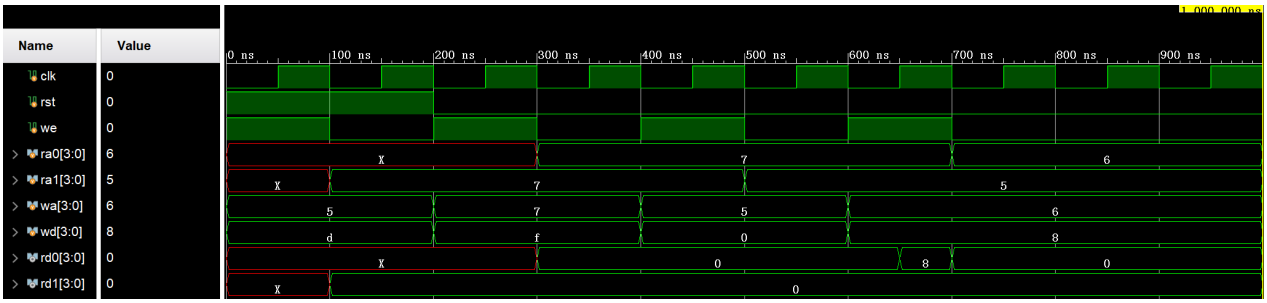
1.寄存器堆

仿真结果

（1）

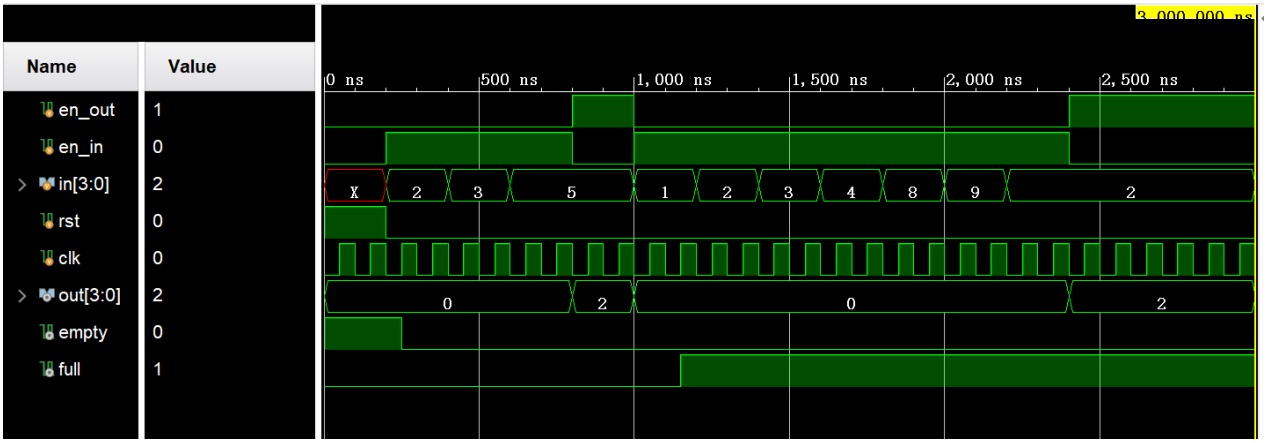


（2）



2.FIFO

第一次失败仿真



数据输出错误，不符合预想入队出队操作。

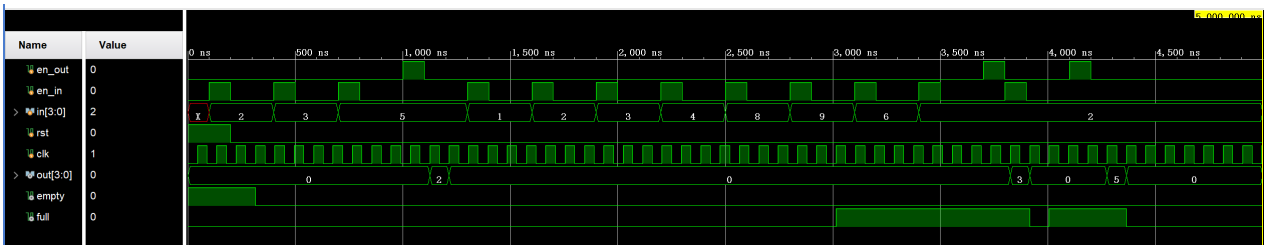
分析原因：在每次en-in和en-out为1时，时钟clk已经过了多个周期，因此在每次使能入队或出队时，实际上进行了不止一次的入出队操作。

解决方法：将en-in和en-out电平信号转换为上升沿信号

```
reg[2:0] delay1;
reg[2:0] delay2;
always @ ( posedge clk  )
delay1 <= { delay1[1:0], en_out} ;
wire pos_enout = delay1[1] && ( ~delay1[2] );
always @ ( posedge clk  )
delay2 <= { delay2[1:0], en_in} ;
wire pos_enin = delay2[1] && ( ~delay2[2] );
```

将pos-enin和pos-enout信号作为出入队操作的使能信号，成功解决了问题。

仿真成功



五、实验总结

在这次实验中，我掌握了具有两个异步读端口和一个同步写端口寄存器堆的实现，同时对FIFO循环队列有了比较深刻的认识，即通过读和写两个地址控制入队和出队的操作，同时在实验中复习的七段数码管显示的原理，对组成原理实验模块化、层次化、参数化设计方法进行了亲身实践，提高了我的独立思考和设计能力。

附：代码

寄存器堆代码

```
module Regfile( clk, rst, we, ra0,ra1,wa,wd,rd0,rd1);

parameter m = 4;
parameter n = 4;
parameter regnum = 1<<m;

input clk;
input rst;
input we;
input [m-1:0]ra0;
input [m-1:0]ra1;
input [m-1:0]wa;
input [n-1:0]wd;
output wire [n-1:0]rd0;
output wire [n-1:0]rd1;
reg [n-1:0] regf [regnum-1:0];
integer i;
initial
  for(i=0;i<regnum;i=i+1)
    regf[i]<=0;
always@(posedge clk or posedge rst)
begin
  if(rst)
    for(i=0;i<regnum;i=i+1)
        regf[i]<=0;
  else
    if(we)
    begin

      regf[wa]<=wd;
    end
end
assign rd0 = regf[ra0];
assign rd1 = regf[ra1];
endmodule
```

寄存器堆仿真代码

```
module regfile_test();

parameter m = 4;
parameter n = 4;

reg clk;
reg rst;
reg we;

reg [m-1:0]ra0;
reg [m-1:0]ra1;
reg [m-1:0]wa;
reg [n-1:0]wd;
wire [n-1:0]rd0;
wire [n-1:0]rd1;

Regfile A1 (
  .clk(clk),
  .rst(rst),
  .we(we),
  .ra0(ra0),
  .ra1(ra1),
  .wa(wa),
  .wd(wd),
  .rd0(rd0),
  .rd1(rd1)
);
initial clk=0;
initial
begin
rst=1;
#200 rst=0;
end
always #50 clk=~clk;
initial
begin
we=1;
wa=4'b0101;
wd=4'b1101;
#100
we=0;
ra1=4'b0111;
#100
we=1;
wa=4'b0111;
```

```
wd=4'b1111;
#100
we=0;
ra0=4'b0111;
#100
we=1;
wa=4'b0101;
wd=4'b0000;
#100
we=0;
ra1=4'b0101;
#100
we=1;
wa=4'b0110;
wd=4'b1000;
#100
we=0;
ra0=4'b0110;
end

endmodule
```

FIFO代码

```
module FIFO(en_out, en_in, in, rst, clk, out, empty, full, seg, dp, an);
input en_out;
input en_in;
input [3:0] in;
input rst;
input clk;
output wire [3:0] out;
output reg empty;
output reg full;
reg [2:0]pw;
reg [2:0]pr;
reg [3:0] regf [7:0];
integer i;
reg [2:0]j;
wire clk_out1;
wire f50hz;
clk_wiz_0 A3 (.clk_out1(clk_out1), .clk_in1(clk));
reg[2:0] delay1;
reg[2:0] delay2;
always @ ( posedge clk  )
```

```
delay1 <= { delay1[1:0], en_out} ;
wire pos_enout = delay1[1] && ( ~delay1[2] );
always @ ( posedge clk  )
delay2 <= { delay2[1:0], en_in} ;
wire pos_enin = delay2[1] && ( ~delay2[2] );

always@(posedge clk or posedge rst)
begin
if(rst)
begin
for(i=0;i<8;i=i+1)
regf[i]<=4'hF;
full<=0;
empty<=1;
pw<=3'h0;
pr<=3'h0;
end
else
begin
if( pos_enin)
if(!full==1)
begin
regf[pw]<=in;
pw<=(pw+1)%8;
if(pw==pr)
full<=1;
empty<=0;
end
if(pos_enout)
if(!empty==1)
begin
regf[pr]<=4'hF;
add(pr,pr);
if(pr==pw)
empty<=1;
full<=0;
end
end
end
assign out=(pos_enout)?regf[pr]:0;
task add;
input [2:0]a;
output [2:0]b;
reg [2:0]temp;
begin
temp=a+1;
b=temp%8;
end
endtask
```

```
div A5 (.clk_out1(clk_out1),  .f50hz(f50hz));
output reg [6:0] seg;
output reg dp;
output reg [7:0] an;
wire [6:0] segf [8:0];
seg_ctrl A6(regf[0],segf[0]);
seg_ctrl A7(regf[1],segf[1]);
seg_ctrl A8(regf[2],segf[2]);
seg_ctrl A16(regf[3],segf[3]);
seg_ctrl A26(regf[4],segf[4]);
seg_ctrl A36(regf[5],segf[5]);
seg_ctrl A46(regf[6],segf[6]);
seg_ctrl A56(regf[7],segf[7]);
always@(posedge f50hz)

if(rst)
j<=0;
else
add(j,j);
always@(posedge f50hz)
begin
case(j)
3'h0:an<=8'b01111111;
3'h1:an<=8'b10111111;
3'h2:an<=8'b11011111;
3'h3:an<=8'b11101111;
3'h4:an<=8'b11110111;
3'h5:an<=8'b11111011;
3'h6:an<=8'b11111101;
3'h7:an<=8'b11111110;
default: an<=8'b00000000;
endcase
seg<=segf[j];
if(j==pr)
dp<=0;
else
dp<=1;
end
endmodule

module div(clk_out1,f50hz);
input clk_out1;
output reg f50hz;
reg [12:0] j;
always@(posedge clk_out1)
begin
if(j==999)
begin
```

```
j<=0;
f50hz<=~f50hz;
end
else
j<=j+1;
end
endmodule
module seg_ctrl(x,seg);
input [3:0]x;
output reg [6:0]seg;
always@(x)
case(x)
4'h0:seg<=7'b0000001;
4'h1:seg<=7'b1001111;
4'h2:seg<=7'b0010010;
4'h3:seg<=7'b0000110;
4'h4:seg<=7'b1001100;
4'h5:seg<=7'b0100100;
4'h6:seg<=7'b0100000;
4'h7:seg<=7'b0001111;
4'h8:seg<=7'b0000000;
4'h9:seg<=7'b0000100;
default: seg<=7'b1111111;
endcase
endmodule
```