**寄存器堆与存储器及其应⽤**

李卓 pb19000064

**实验目的**

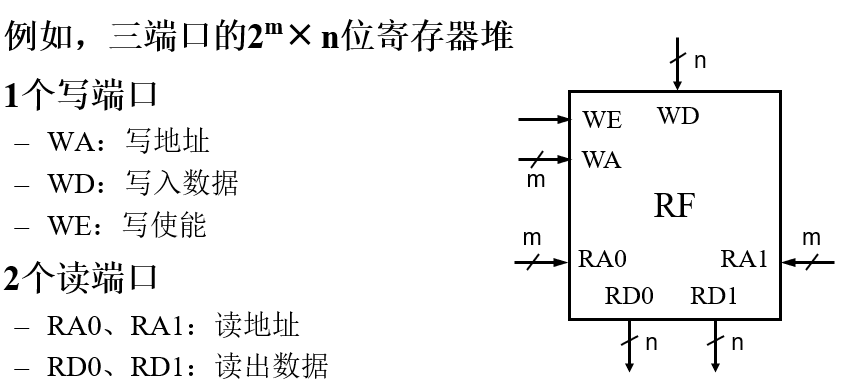
掌握寄存器堆(Register File)和存储器的功能 、时序及其应⽤

熟练掌握数据通路和控制器的设计和描述⽅法

**实验原理**

1. 寄存器堆

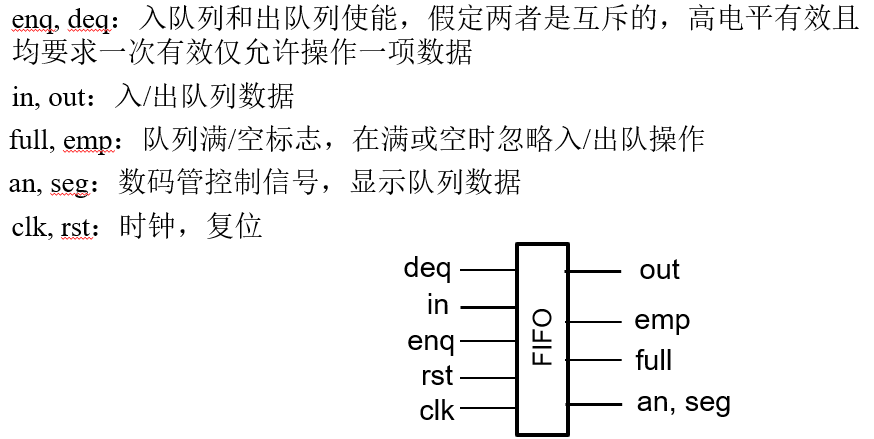
行为方式参数化描述32 X WIDTH寄存器堆

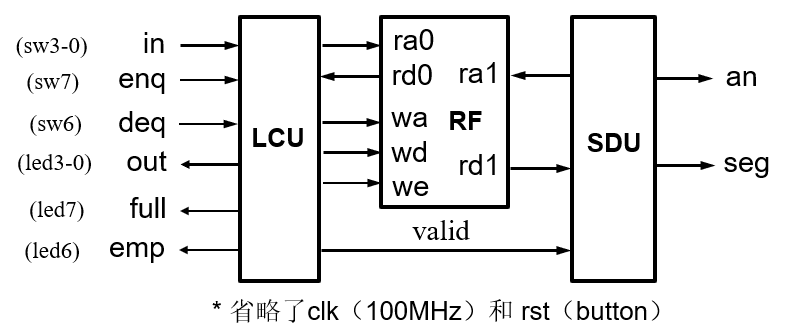


1. RAM存储器

利⽤ IP 核中的 Block Memory Generator 以及 Distributed Memory Generator，⽣成 16×8 位块式存储器和 16×8 位分布式存储器。

1. 利用寄存器堆实现FIFO队列

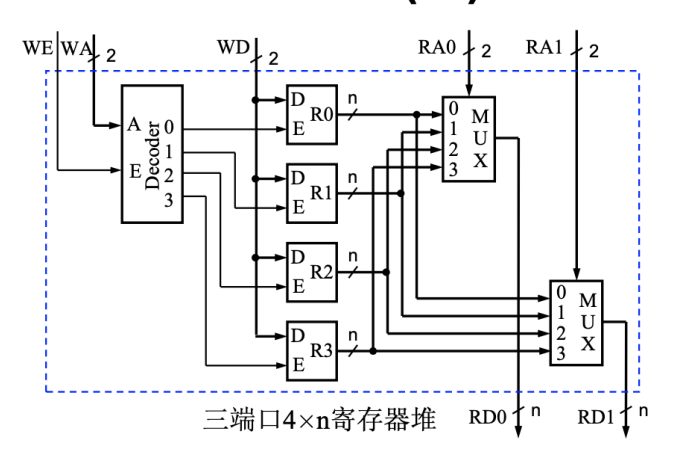




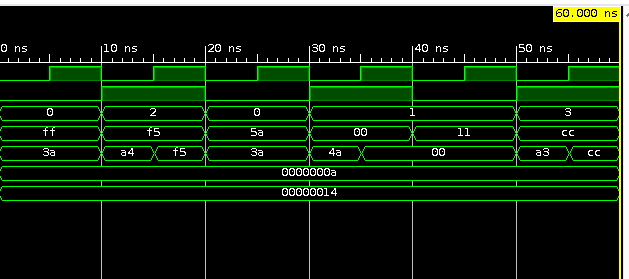
**实验过程**

1. 寄存器堆

数据通路如下：

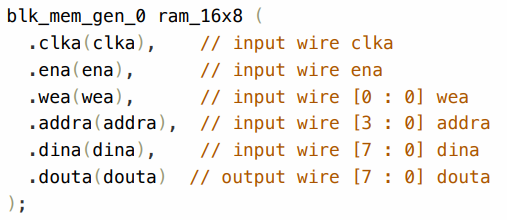


仿真如下



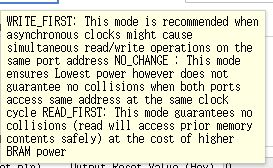
1. RAM存储器

例化端口



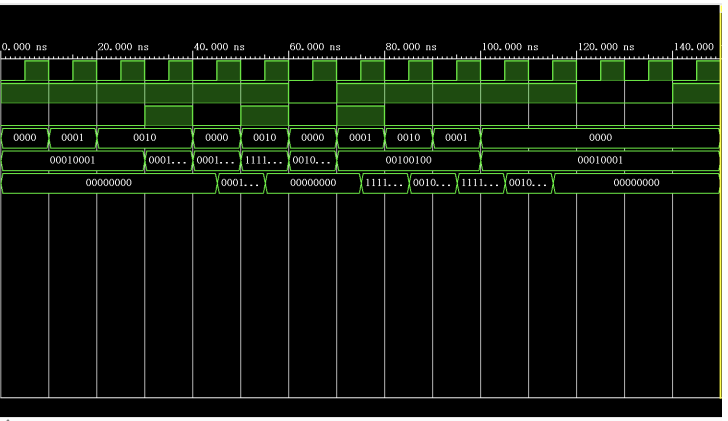
块式存储器同步，分布式则是异步

分布式各个模式定义如下



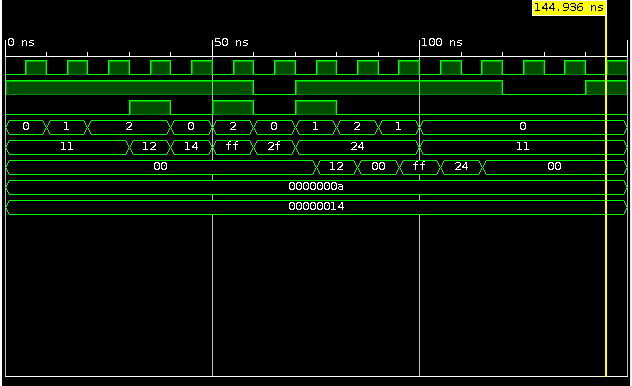
write first：当写数据时读出的是写进去的数据 no change：写数据时不读数据显示出的数据保持不变 read first: 当写数据时读出的是正在读的原来数据

块式仿真

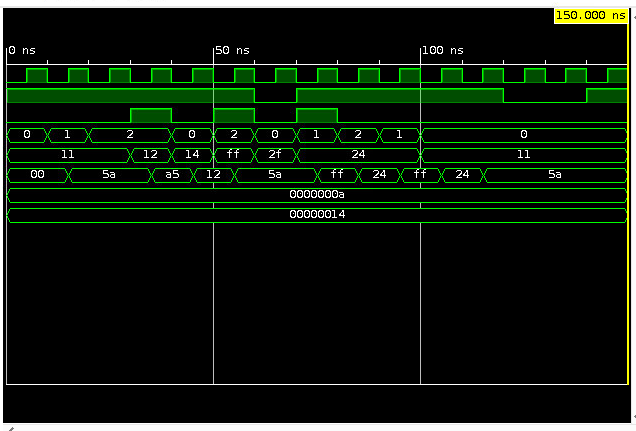


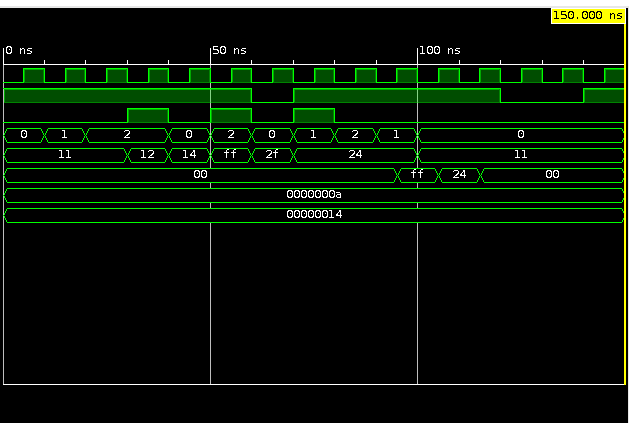
分布式仿真:

read first:



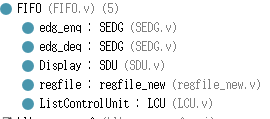
write first:

  
no change:



1. FIFO队列

文件结构：



sedg 判断 enq 和deq是否被按下。在使 能信号的⼀次有效持续期间，仅允许最多⼊队或出队一个数据

LCU是队列控制单元

用head指针和tail指针标记队列的头和尾，count寄存器存储队列中数据数目，用来判断队满（full）和队空(emp)。valid数组记录每个地址是否有数据

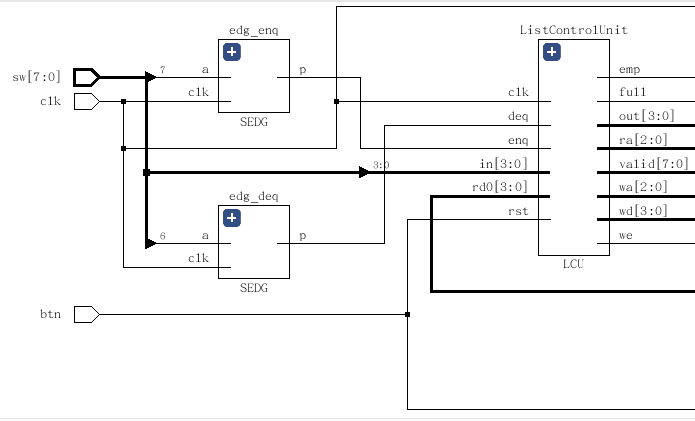
⼊队列使能(enq)有效时，将输⼊数据(in)加⼊队尾（tail），队尾后移并修改valid；出队列使能(deq) 有效时，将队列头数据输出(out), 队头(head)后移并修改valid。当队列满时不能执⾏⼊队操作，队列空时不能进⾏出队操作。

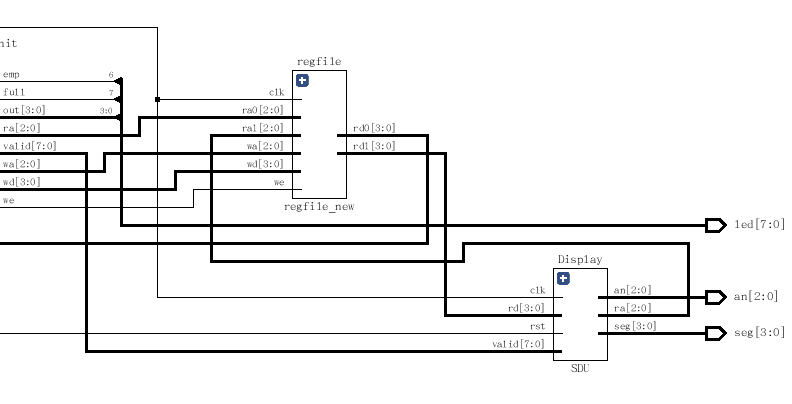
regfile 是已完成的寄存器堆

SDU是数码管显示单元

an和ra同步，循环遍历寄存器的所有地址，如果该地址有数据（valid该位有效），就读取数据将其输出到seg端口，显示在数码管上；如果没有数据，就什么都不做跳过该地址，这样在数码管上对应的位置不亮

数据通路：





**实验代码**

SEDG.v

*module SEDG(*

*input a,*

*input clk,*

*output reg s,*

*output wire p*

*);*

*reg st,pt;*

*always@(posedge clk)begin*

*if(a)begin*

*st <= 1;*

*end*

*else begin*

*st <= 0;*

*end*

*end*

*always@(posedge clk)begin*

*if(st)begin*

*s <= 1;*

*end*

*else begin*

*s <= 0;*

*end*

*end*

*always@(posedge clk)begin*

*if(s)begin*

*pt <= 1;*

*end*

*else begin*

*pt <= 0;*

*end*

*end*

*assign p = (~pt) & s;*

*endmodule*

LCU.v

*module LCU(*

*input [3:0]in,*

*input enq,*

*input deq,*

*input clk,*

*input [3:0]rd0,*

*input rst,*

*output reg [3:0]out,*

*output wire full,*

*output wire emp,*

*output wire [2:0]ra,*

*output wire [2:0]wa,*

*output wire [3:0]wd,*

*output wire we,*

*output reg [7:0] valid*

*);*

*reg [2:0] head,tail;//poninter to head and tail*

*reg [3:0] count;*

*assign we = enq&(~full);*

*assign ra = head;*

*assign wa = tail;*

*assign wd = in;*

*always @( posedge clk )*

*begin*

*if(rst)begin*

*valid <= 8'b0;*

*head <= 3'b0;*

*tail <= 3'b0;*

*count <= 4'd0;*

*out <= 4'd0;*

*end*

*else if(enq)begin*

*if(~full)begin*

*valid[tail] <= 1;*

*tail <= tail + 3'd1;*

*count <= count + 1;*

*end else begin end*

*end*

*else if(deq)begin*

*if(~emp)begin*

*valid[head] <= 0;*

*head <= head + 3'd1;*

*out <= rd0;*

*count <= count - 1;*

*end else begin end*

*end else begin end*

*end*

*assign full = (count == 4'h8)?1:0;*

*assign emp = (count == 4'd0)?1:0;*

*endmodule*

SDU.v

*module SDU(*

*input rst,*

*input clk,*

*input [3:0]rd,//read data*

*input [7:0]valid,*

*output wire [2:0]ra,//read address*

*output wire [2:0]an,//segment address*

*output wire [3:0]seg//segment data*

*);*

*reg [23:0]count;*

*wire [3:0]x0;*

*reg [2:0] an\_reg;*

*assign x0 = 4'h0;*

*assign an = ra;*

*assign ra = an\_reg;*

*always@(posedge clk)*

*begin*

*if(valid[count[15:13]])begin*

*an\_reg <= count[15:13];*

*end else begin end*

*end*

*always@(posedge clk)*

*begin*

*if(rst)begin*

*count <= 24'd0;*

*end else begin*

*count <= count + 1;*

*end*

*end*

*assign seg=valid[ra]?rd:x0;*

*endmodule*

FIFO.v register\_file.v

*module FIFO(*

*input [7:0]sw,*

*input btn,*

*input clk,*

*output wire [7:0]led,*

*output wire [2:0]an,*

*output wire [3:0]seg*

*);*

*wire [2:0]ra0, ra1, wa;*

*wire [3:0]rd0, rd1, wd;*

*wire we;*

*wire enq\_edge;*

*wire deq\_edge;*

*wire [7:0]valid;*

*SEDG edg\_enq(sw[7],clk, ,enq\_edge);*

*SEDG edg\_deq(sw[6],clk, ,deq\_edge);*

*SDU Display(btn, clk, rd1, valid, ra1, an, seg);*

*regfile\_new regfile(clk, ra0, rd0, ra1, rd1, wa, we, wd);*

*LCU ListControlUnit(sw[3:0], enq\_edge, deq\_edge, clk, rd0, btn, led[3:0], led[7], led[6], ra0, wa, wd, we, valid);*

*endmodule*

*module register\_file*

*#( parameter WIDTH = 32 ) (*

*input clk, //clock*

*input [ 4: 0 ] ra0, //read address 0*

*output [WIDTH - 1: 0] rd0, //read data 0*

*input [4: 0] ra1, //read address 1*

*output [WIDTH - 1: 0] rd1, //read data 1*

*input [4: 0] wa, //write address*

*input we, //write enable*

*input [WIDTH - 1: 0] wd //write data*

*);*

*reg [WIDTH - 1: 0] mem[0: WIDTH - 1];*

*assign rd0 = mem[ra0];*

*assign rd1 = mem[ra1];*

*always @( posedge clk )*

*begin*

*if ( we )*

*begin*

*mem[ wa ] <= wd;*

*end*

*end*

*endmodule*