**实验报告**

2019 年 4 月 16 日 成绩：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 於文卓 | 学号 | 17061833 | 班级 | 17052317 |
| 专业 | 计算机科学与技术 | | 课程名称 | 计算机组成原理课程设计 | |
| 任课老师 | 冯建文 | 指导老师 | 冯建文 | 机位号 |  |
| 实验序号 | 4 | 实验名称 | 寄存器堆设计实验 | | |
| 实验时间 | 4月16日 | 实验地点 | 1-225 | 实验设备号 | #9 |
| **一、实验目的与要求** | | | | | |
| 1. 实验目的： 2. 学习使用Verilog HDL语言进行时序电路的设计方法 3. 掌握运用Verilog HDL进行行为描述与建模的技巧和方法 4. 学习多端口寄存器堆的数据传送与读写工作原理，掌握多端口寄存器堆的设计方法 5. 实验要求： 6. 设计一个32\*32位的三端口寄存器，含有32个寄存器，每个寄存器32位。 7. 三端口寄存器中，包含两个读端口和一个写端口，可同时读两个数据并写一个数据。 8. 读出的数据使用数码管显示 | | | | | |
| **二、实验设计与程序代码** | | | | | |
| 1. 模块设计说明 2. 设计register\_file模块，主要输入为5位R\_Addr\_A读地址A，R\_Addr\_B读地址B，W\_Addr写地址，32位的写入数据W\_Data和Write\_reg信号。输出为读到的A和B的数据 3. 构建数码管模块，使得A和B的数据显示在数码管上 4. 构建顶层模块，调用寄存器堆模块和数码管模块。由于开关的限制，在顶层模块中用三位的choiceWriteData选择八组数据送入寄存器堆中。 5. 实验程序源代码及注释等 6. 寄存器堆模块代码  |  | | --- | | `timescale 1ns / 1ps  module register\_file(  input [4:0]R\_Addr\_A,//读地址A 0-32  input [4:0]R\_Addr\_B,//读地址B 0-32  input [4:0]W\_Addr,//写地址 0-32  input [31:0]W\_Data,//写数据 32位的数据  input Write\_Reg,//写信号  input clk,  input reset\_,  output [31:0]R\_Data\_A,//读到A数据  output [31:0]R\_Data\_B//读到B数据  );  integer i;  reg [31:0]REG\_Files[0:31];//32个寄存器堆，每个32位  assign R\_Data\_A = REG\_Files[R\_Addr\_A];//读操作A  assign R\_Data\_B = REG\_Files[R\_Addr\_B];//读操作B    //写操作  always@(posedge clk or negedge reset\_)  begin  if(!reset\_)  begin  for(i=0;i<31;i=i+1)  REG\_Files[i] <= 32'h0000\_0000;  end  else  begin  if(Write\_Reg)  REG\_Files[W\_Addr] <= W\_Data;  end  end  endmodule |  1. 数码管模块代码  |  | | --- | | module scan(    input clk,//100MHZ  input rst\_,//reset  input [31:0]key,//通过拨动按钮控制段选  output reg [7:0]wei,//在always里赋值 需要用到reg  output reg [7:0]duan  );  wire clk\_4ms;//分频过后2ms一个方波(改过了 是2ms了)  reg [2:0]wei\_count;//位码计数器，控制4个数码管  reg [7:0]segcode[0:15] = {//用数组的方式定义 左表示位数 右表示个数  8'b0000\_0011,//0  8'b1001\_1111,//1  8'b0010\_0101,//2  8'b0000\_1101,//3  8'b1001\_1001,//4  8'b0100\_1001,//5  8'b0100\_0001,//6  8'b0001\_1111,//7  8'b0000\_0001,//8  8'b0000\_1001,//9  ~8'b11101110,//A  ~8'b00111110,//b  ~8'b10011100,//C  ~8'b01111010,//d  ~8'b10011110,//E  8'b01110001//F  };  div f1(rst\_,clk,clk\_4ms);//分频器模块  wire clk\_slow;//一秒钟两次  div\_slow f2(rst\_,clk,clk\_slow);  always @(posedge clk\_4ms or negedge rst\_)//每一个上升沿都使得计数器+1 做到连续扫描数码管  begin  if(!rst\_)  begin  wei\_count<=3'b00;  end  else  begin  wei\_count<=wei\_count+1'b1;  if(wei\_count ==3'b111)  begin  wei\_count<=3'b000;  end  end    end  reg [7:0] pos\_1,pos\_2,pos\_3,pos\_4,pos\_5,pos\_6,pos\_7,pos\_8;  //用开关控制数码管显示  always @(posedge clk\_4ms or negedge rst\_)  begin  if(!rst\_)  begin  pos\_1 <= 4'b0000;  pos\_2 <= 4'b0000;  pos\_3 <= 4'b0000;  pos\_4 <= 4'b0000;  pos\_5 <= 4'b0000;  pos\_6 <= 4'b0000;  pos\_7 <= 4'b0000;  pos\_8 <= 4'b0000;  end  else  begin  pos\_1 <= key[3:0];//if(pos\_1 > 4'd9) pos\_1<=4'b0000;  pos\_2 <= key[7:4];//if(pos\_2 > 4'd9) pos\_2<=4'b0000;  pos\_3 <= key[11:8];//if(pos\_3 > 4'd9) pos\_3<=4'b0000;  pos\_4 <= key[15:12];//if(pos\_4 > 4'd9) pos\_4<=4'b0000;  pos\_5 <= key[19:16];  pos\_6 <= key[23:20];  pos\_7 <= key[27:24];  pos\_8 <= key[31:28];  end  end  always @(\*)//数码管扫描进程 想想为什么是\* ？？？  begin  case(wei\_count)  3'b000: begin wei<=8'b11111110; duan<=segcode[pos\_1];end  3'b001: begin wei<=8'b11111101; duan<=segcode[pos\_2];end  3'b010: begin wei<=8'b11111011; duan<=segcode[pos\_3];end  3'b011: begin wei<=8'b11110111; duan<=segcode[pos\_4];end  3'b100: begin wei<=8'b11101111; duan<=segcode[pos\_5];end  3'b101: begin wei<=8'b11011111; duan<=segcode[pos\_6];end  3'b110: begin wei<=8'b10111111; duan<=segcode[pos\_7];end  3'b111: begin wei<=8'b01111111; duan<=segcode[pos\_8];end  default:begin wei<=8'b11111111;end  endcase  end    endmodule |  1. 顶层模块  |  | | --- | | `timescale 1ns / 1ps  //////////////////////////////////////////////////////////////////////////////////  // Company:  // Engineer:  //  // Create Date: 21:01:22 04/15/2019  // Design Name:  // Module Name: top  // Project Name:  // Target Devices:  // Tool versions:  // Description:  //  // Dependencies:  //  // Revision:  // Revision 0.01 - File Created  // Additional Comments:  //  //////////////////////////////////////////////////////////////////////////////////  module top(  input [4:0]R\_Addr\_A,//读地址A 0-32  input [4:0]R\_Addr\_B,//读地址B 0-32  input [4:0]W\_Addr,//写地址 0-32  input [2:0]choiceWriteData,//选择写数据    input Out\_Reg,  input Write\_Reg,//写信号    input clk,//寄存器堆的信号  input scan\_clk,//数码管的信号  input reset\_,  output [7:0]wei,  output [7:0]duan  );  reg [31:0]W\_Data;  wire [31:0]Data\_A;  wire [31:0]Data\_B;  always@(\*)  begin  case(choiceWriteData)  3'b000:W\_Data = 32'hABCD\_EFFF;  3'b001:W\_Data = 32'h1111\_1111;  3'b010:W\_Data = 32'h1234\_5678;  3'b011:W\_Data = 32'h0000\_0001;  3'b100:W\_Data = 32'h0000\_1234;  3'b101:W\_Data = 32'h0000\_FFFF;  3'b110:W\_Data = 32'hFFFF\_1234;  3'b111:W\_Data = 32'hFFFF\_FFFF;  endcase  end  wire [31:0]Data;  register\_file mode1(R\_Addr\_A,R\_Addr\_B,W\_Addr,W\_Data,Write\_Reg,clk,reset\_,Data\_A,Data\_B);  assign Data = Out\_Reg?Data\_A:Data\_B;  //Data = Data\_A;  scan mode2(scan\_clk,reset\_,Data,wei,duan);  endmodule | | | | | | |
| **三、实验仿真** | | | | | |
| 1. 仿真代码  |  | | --- | | `timescale 1ns / 1ps  ////////////////////////////////////////////////////////////////////////////////  // Company:  // Engineer:  //  // Create Date: 14:21:09 04/16/2019  // Design Name: register\_file  // Module Name: D:/fpga/shiyan4/test.v  // Project Name: shiyan4  // Target Device:  // Tool versions:  // Description:  //  // Verilog Test Fixture created by ISE for module: register\_file  //  // Dependencies:  //  // Revision:  // Revision 0.01 - File Created  // Additional Comments:  //  ////////////////////////////////////////////////////////////////////////////////  module test;  // Inputs  reg [4:0] R\_Addr\_A;  reg [4:0] R\_Addr\_B;  reg [4:0] W\_Addr;  reg [31:0] W\_Data;  reg Write\_Reg;  reg clk;  reg reset\_;  // Outputs  wire [31:0] R\_Data\_A;  wire [31:0] R\_Data\_B;  // Instantiate the Unit Under Test (UUT)  register\_file uut (  .R\_Addr\_A(R\_Addr\_A),  .R\_Addr\_B(R\_Addr\_B),  .W\_Addr(W\_Addr),  .W\_Data(W\_Data),  .Write\_Reg(Write\_Reg),  .clk(clk),  .reset\_(reset\_),  .R\_Data\_A(R\_Data\_A),  .R\_Data\_B(R\_Data\_B)  );  initial begin  // Initialize Inputs  R\_Addr\_A = 0;  R\_Addr\_B = 0;  W\_Addr = 0;  W\_Data = 0;  Write\_Reg = 0;  clk = 0;  reset\_ = 0;  end  // Wait 100 ns for global reset to finish  always #10 clk=~clk;  initial  begin  // Add stimulus here    #20;//不写 读取AB的数据  R\_Addr\_A = 32'h0000\_0001;  R\_Addr\_B = 32'h0000\_0002;  W\_Addr = 32'h0000\_0001;  W\_Data=32'hABCD\_EFFF;  Write\_Reg=0;  reset\_=1;  #20;//往A的地址写数据  R\_Addr\_A = 32'h0000\_0001;  R\_Addr\_B = 32'h0000\_0002;  W\_Addr = 32'h0000\_0001;  W\_Data=32'hABCD\_EFFF;  Write\_Reg=1;  reset\_=1;  #20;//往B的地方写数据  R\_Addr\_A = 32'h0000\_0001;  R\_Addr\_B = 32'h0000\_0002;  W\_Addr = 32'h0000\_0002;  W\_Data=32'hAAAA\_AAAA;  Write\_Reg=1;  reset\_=1;  #20;//读A和B的内容  R\_Addr\_A = 32'h0000\_0001;  R\_Addr\_B = 32'h0000\_0002;  W\_Addr = 32'h0000\_0002;  W\_Data=32'h1111\_1111;  Write\_Reg=0;  reset\_=1;  #20;  reset\_=0;  end  endmodule |  1. 仿真波形      1. 仿真结果分析 2. A端口读$1地址的数据，B端口读$2的数据，由于write\_reg信号为0，所以AB端口读出来的数据均为零 3. W端口往$1地址写数据，数据为32’h abcd\_efff，write\_reg置为1，一个上跳沿后，可以读到A读到的数据为abcd\_efff, B读到的数据依然为0 4. W端口往$2地址写数据，数据为32’h aaaa\_aaaa，write\_reg置为1，一个上跳沿后，可以读到A读到的数据为abcd\_efff, B读到的数据变为32’haaaa\_aaaa。 5. 当rest\_为0时，异步置零，数据A和数据B均变为零   仿真结果符合要求，仿真结果正确 | | | | | |
| **四、电路图** | | | | | |
|  | | | | | |
| **五、引脚配置（约束文件）** | | | | | |
| //如果边沿输入信号，还要在";"前加上：| CLOCK\_DEDICATED\_ROUTE = FALSE  //譬如: NET "clk" LOC = N17 | IOSTANDARD = LVCMOS18 | CLOCK\_DEDICATED\_ROUTE = FALSE; //BT\_S[7]  //HCS-A01板卡上的实际开关  NET "scan\_clk" LOC = E3 |IOSTANDARD = LVCMOS18;  NET "clk" LOC = P18 |IOSTANDARD = LVCMOS18 |CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "reset\_" LOC = N17 |IOSTANDARD = LVCMOS18 |CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "R\_Addr\_A[4]" LOC = V5 | IOSTANDARD = LVCMOS18; //R\_Addr\_A[4]  NET "R\_Addr\_A[3]" LOC = T4 | IOSTANDARD = LVCMOS18; //R\_Addr\_A[3]  NET "R\_Addr\_A[2]" LOC = V6 | IOSTANDARD = LVCMOS18; //R\_Addr\_A[2]  NET "R\_Addr\_A[1]" LOC = T5 | IOSTANDARD = LVCMOS18; //R\_Addr\_A[1]  NET "R\_Addr\_A[0]" LOC = T6 | IOSTANDARD = LVCMOS18; //R\_Addr\_A[0]  NET "R\_Addr\_B[4]" LOC = V7 | IOSTANDARD = LVCMOS18; //R\_Addr\_B[4]  NET "R\_Addr\_B[3]" LOC = R8 | IOSTANDARD = LVCMOS18; //R\_Addr\_B[3]  NET "R\_Addr\_B[2]" LOC = U9 | IOSTANDARD = LVCMOS18; //R\_Addr\_B[2]  NET "R\_Addr\_B[1]" LOC = T9 | IOSTANDARD = LVCMOS18; //R\_Addr\_B[1]  NET "R\_Addr\_B[0]" LOC = V10 | IOSTANDARD = LVCMOS18; //R\_Addr\_B[0]  NET "W\_Addr[4]" LOC = R10 | IOSTANDARD = LVCMOS18; //W\_Addr[4]  NET "W\_Addr[3]" LOC = U11 | IOSTANDARD = LVCMOS18; //W\_Addr[3]  NET "W\_Addr[2]" LOC = R11 | IOSTANDARD = LVCMOS18; //W\_Addr[2]  NET "W\_Addr[1]" LOC = U12 | IOSTANDARD = LVCMOS18; //W\_Addr[1]  NET "W\_Addr[0]" LOC = T13 | IOSTANDARD = LVCMOS18; //W\_Addr[0]  NET "choiceWriteData[0]" LOC = V14 | IOSTANDARD = LVCMOS18; //choiceWriteData[0]  NET "choiceWriteData[1]" LOC = T14 | IOSTANDARD = LVCMOS18; //choiceWriteData[1]  NET "choiceWriteData[2]" LOC = V15 | IOSTANDARD = LVCMOS18; //choiceWriteData[1]  NET "Write\_Reg" LOC = R15 | IOSTANDARD = LVCMOS18; //Write\_Reg  NET "Out\_Reg" LOC = U16 | IOSTANDARD = LVCMOS18; //Out\_Reg  NET "wei[7]" LOC = C9 |IOSTANDARD = LVCMOS18;  NET "wei[6]" LOC = C10 |IOSTANDARD = LVCMOS18;  NET "wei[5]" LOC = D10 |IOSTANDARD = LVCMOS18;  NET "wei[4]" LOC = C11 |IOSTANDARD = LVCMOS18;  NET "wei[3]" LOC = M17 |IOSTANDARD = LVCMOS18;  NET "wei[2]" LOC = J14 |IOSTANDARD = LVCMOS18;  NET "wei[1]" LOC = K13 |IOSTANDARD = LVCMOS18;  NET "wei[0]" LOC = P14 |IOSTANDARD = LVCMOS18;  NET "duan[7]" LOC = F14 |IOSTANDARD = LVCMOS18;  NET "duan[6]" LOC = N14 |IOSTANDARD = LVCMOS18;  NET "duan[5]" LOC = J13 |IOSTANDARD = LVCMOS18;  NET "duan[4]" LOC = G13 |IOSTANDARD = LVCMOS18;  NET "duan[3]" LOC = F13 |IOSTANDARD = LVCMOS18;  NET "duan[2]" LOC = G14 |IOSTANDARD = LVCMOS18;  NET "duan[1]" LOC = M13 |IOSTANDARD = LVCMOS18;  NET "duan[0]" LOC = H14 |IOSTANDARD = LVCMOS18; | | | | | |
| **六、思考与探索** | | | | | |
| 1. 实验结果记录：  |  |  |  | | --- | --- | --- | | 寄存器地址$ | 写入数据 | 读出数据 | | $0 | 32’h1111\_1111 | 32’h1111\_1111 | | $4 | 32’h1234\_1234 | 32’h1234\_1234 | | $7 | 32’abcd\_abcd | 32’abcd\_abcd | | $12 | 32’haaaa\_bbbb | 32’haaaa\_bbbb | | $20 | 32’hbbaa\_bacd | 32’hbbaa\_bacd | | $21 | 32’hffff\_ffff | 32’hffff\_ffff | | $29 | 32’h0000\_ffff | 32’h0000\_ffff | | $31 | 32’heeee\_ffff | 32’heeee\_ffff |   当读地址和写地址相同，Write\_reg置为1，一个上跳沿后，数码管可以显示出W\_data的数据   1. 实验结论：   实验结果也符合预期，说明实验结果正确   1. 问题与解决方案： 2. 寄存器堆的时钟信号接在了E3管脚上。不应该直接由cpu提供时钟信号，clk由按键控制。 3. 按键往下按是0，开关往下拨是1，容易记反 4. 思考题：   (2)实现MIPS计算机的寄存器   |  | | --- | | always@(posedge clk or negedge reset\_)  begin  if(!reset\_)  begin  for(i=0;i<31;i=i+1)  REG\_Files[i] <= 32'h0000\_0000;  end  else  begin  if(Write\_Reg)  begin  if(W\_Addr == 0)  begin  REG\_Files[W\_Addr] <= 32'h0000\_0000;  end  else  REG\_Files[W\_Addr] <= W\_Data;  end  end    end |   我的做法是，在写操作的时候多加一个判断，如果是往$0地址写入，写入的数据永远都是0  （3）写操作需要在**写信号为1的同时来一个上跳沿信号**才能写入，而读信号不需要时钟信号，**只要读取的地址数据改变了**，立马可以读到。电路上，如果读写地址为同一个，当按下时钟的按钮产生一个上跳沿使得数据写入地址后，**数码管立刻可以显示写入的数据** | | | | | |