**实验报告**

2021年05月18日 成绩：---------

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 刘爱兵 | 学号 | 19114801 | 班级 | 19计科2班 |
| 专业 | 计算机科学与技术 | 课程名称 | 计算机组成原理实验 | 任课教师 | 吴云志 |
| 实验序号 | 03 | 实验名称 | 寄存器堆设计 | | |
| 实验时间 | 2021.05.18 | 机位号 | 18 | 实验设备号 | 21 |

1. **实验目的与要求**
2. 实验目的

学习使用Verilog HDL语言进行时序电路的设计方法；

掌握灵活运用Verilog HDL语言进行各种描述与建模的技巧和方法；

学习寄存器堆的数据传送与读写工作原理，掌握寄存器堆的设计方法。

1. 实验要求
2. 编程实现基本的寄存器堆模块，并通过仿真验证；

编写一个实验验证的顶层模块，调用该寄存器堆模块

2）实验室任务：

配置管脚：见下表

生成\*.bit文件，下载到HDU-XL-01教学开发板中。

完成板级调试。

撰写实验报告。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 信号 | 配置设备管脚 | 功能说明 |
| 输入信号 | 数据信号 | 32个逻辑开关 | 提供读A、读B、写地址 |
| Reset | 1个按钮 | 系统复位 |
| 写入控制信号 | 1个按钮 | 提交数据写入寄存器堆 |
| Addr\_W | 1个按钮 | 将SW末五位提交为写入端口地址 |
| A /B读端口选择 | 1个按钮 | 选择读出A端口还是B端口 |
| Clk | 1个按钮 | 手动时钟输入 |
| Addr\_R | 1个按钮 | 将SW末五位提交为读端口地址 |
| 输出信号 | LED[31:0] | 32个LED灯 | 显示A或者B读取端口数据 |
| 数码管 | 8位数码管 | 最高位显示读取端口状态（A/B），次高位置零；两位显示写入端口地址；两位显示A读端口地址；最低两位显示B读端口地址 |

1. **实验设计与程序代码**
2. 模块设计说明

（描述这个实验的设计方案，分几个模块，各模块的功能，各模块之间的连接关系，可附图）

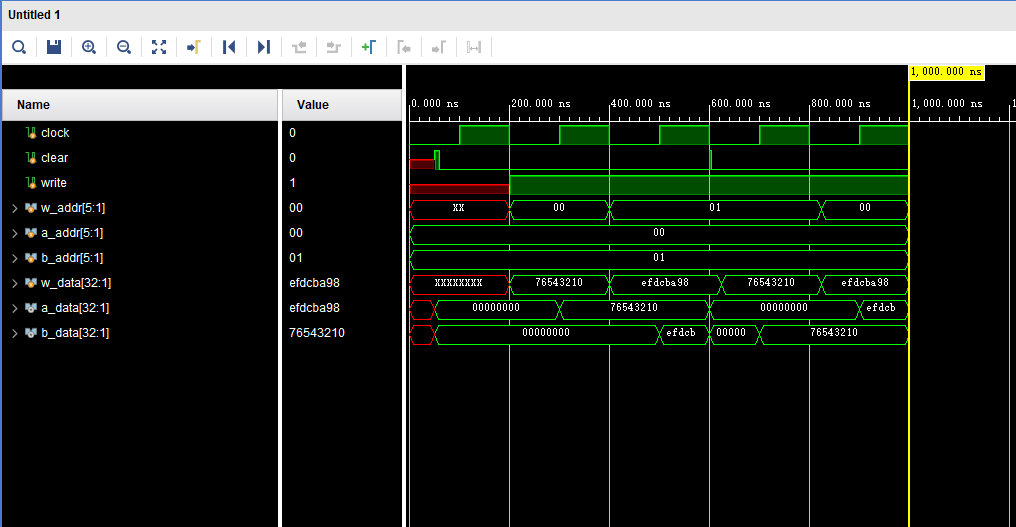


1. 实验程序源代码及注释等

（实验各个模块的代码，包含功能注释）

|  |
| --- |
| （1）**RegFile.v**  `timescale 1ns / 1ps  // Register File，寄存器堆  module RegFile(clk, rst, wrt, w\_addr, w\_data, // 单端口写入  a\_addr, b\_addr, a\_data, b\_data); // A、B 双端口读取  parameter ADDR = 5; // 寄存器堆地址位宽，寻址能力决定最大字数  parameter NUMB = 1 << ADDR; // 寄存器堆字数，如 2^5 共 32 个字  parameter SIZE = 32; // 寄存器堆字长（字的位宽）  reg [SIZE:1] reg\_file [0:NUMB-1]; // NUMB x SIZE 位寄存器堆  integer i; // 用于遍历 NUMB 个字  input clk, rst, wrt; // clock, reset, write：时钟、清零信号、写控制信号  input [ADDR:1] w\_addr; // 写入地址  input [SIZE:1] w\_data; // 写入数据  always @(posedge clk or posedge rst) begin  if (rst) for (i = 0; i < NUMB; i = i+1) reg\_file[i] <= 0; // 异步清零  else if (wrt) reg\_file[w\_addr] <= w\_data; // 时钟上跳且写控制高电平时写入  end  input [ADDR:1] a\_addr, b\_addr; // A、B 两端口读取地址  output [SIZE:1] a\_data, b\_data; // A、B 两端口读出数据  assign a\_data = reg\_file[a\_addr];  assign b\_data = reg\_file[b\_addr];  endmodule // RegFile  （2）**Board.v**  `timescale 1ns / 1ps  // 寄存器堆板极调试  module Board(sw, swb, led, clk, which, seg, enable);  input [1:32] sw;  input [1:6] swb;  reg [5:1] w\_addr, a\_addr, b\_addr;  wire [32:1] a\_data, b\_data;  RegFile RF\_Instance(  .clk(swb[1]),  .rst(swb[2]),  .wrt(swb[3]),  .w\_data(sw),  .w\_addr(w\_addr),  .a\_addr(a\_addr),  .b\_addr(b\_addr),  .a\_data(a\_data),  .b\_data(b\_data));  // SWB4：将 SW 末五位提交至 写入端口地址  always @(posedge swb[4]) w\_addr <= sw[28:32];  // SWB5：通过末位取反切换 A、b 两读取端口 地址提交 及 数据显示，默认为 A 端口  reg [3:0] read\_port = 4'hA; // 4'hA === 4'b1010; 4'hb === 4'b1011  always @(posedge swb[5]) read\_port[0] <= ~read\_port[0];  always @(posedge swb[6]) // SWB6：将 SW 末五位提交至 A 或 b 读取端口地址  if (read\_port === 4'hA) a\_addr <= sw[28:32]; else b\_addr <= sw[28:32];  output reg [1:32] led; // LED：显示 A 或 b 读取端口 读出数据  always @\* led <= (read\_port === 4'hA) ? a\_data : b\_data;  input clk; // 数码管相关  output [2:0] which;  output [7:0] seg;  output reg enable = 1; // 默认开启数码管使能  Display Display\_Instance(.clk(clk), .which(which), .seg(seg),  .data({read\_port, // 最高位显示读取端口状态  7'b0, w\_addr, // 三位显示写入端口地址  3'b0, a\_addr, // 两位显示 A 读端口地址  3'b0, b\_addr})); // 最低两位显示 b 读端口地址  endmodule // Board   1. **实验仿真** 2. 仿真代码（仿真源代码）   **RegFile\_Test.v**  `timescale 1ns / 1ps  // 寄存器堆仿真测试  module RegFile\_Test();  reg clock, clear, write;  reg [5:1] w\_addr, a\_addr, b\_addr;  reg [32:1] w\_data; // input  wire [32:1] a\_data, b\_data; // output  RegFile RF\_Instance( // Instantiate a register file module  .clk(clock),  .rst(clear),  .wrt(write),  .w\_addr(w\_addr),  .w\_data(w\_data),  .a\_addr(a\_addr),  .b\_addr(b\_addr),  .a\_data(a\_data),  .b\_data(b\_data));  always #100 clock = ~clock;  initial begin  clock = 0; a\_addr = 5'b0\_0000; b\_addr = 5'b0\_0001; #50;  clear = 1; #10; clear = 0; #140; // 异步清零  write = 1; w\_addr = 5'b0\_0000; w\_data = 32'h7654\_3210; #200; // A 端口  write = 1; w\_addr = 5'b0\_0001; w\_data = 32'hefdc\_ba98; #200; // B 端口  clear = 1; #5; clear = 0; #20; // 异步清零  write = 1; w\_addr = 5'b0\_0001; w\_data = 32'h7654\_3210; #200; // B 端口  write = 1; w\_addr = 5'b0\_0000; w\_data = 32'hefdc\_ba98; #200; // A 端口  end  endmodule // RegFile\_Test |

1. 仿真波形（运行仿真时波形截图）

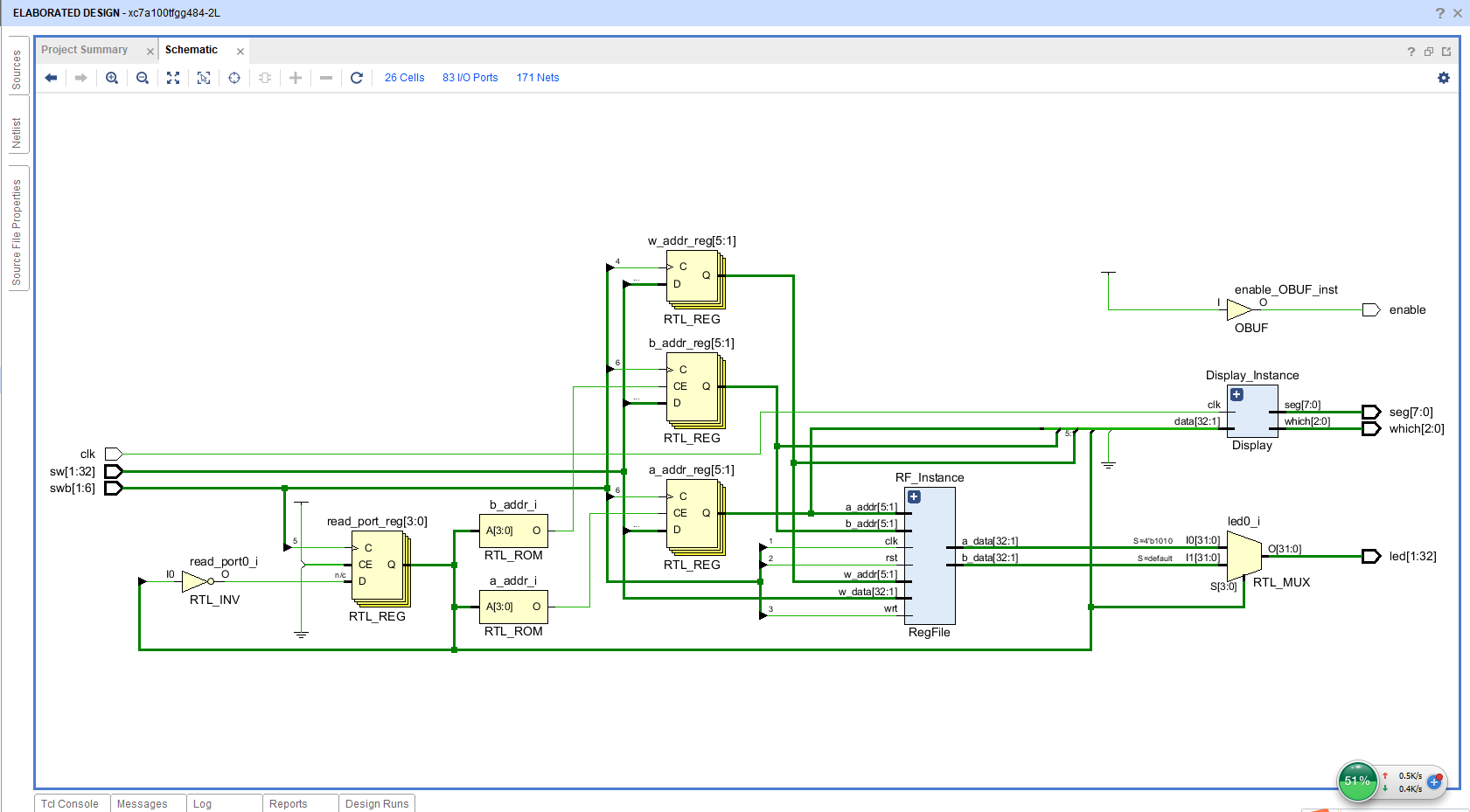


1. 仿真结果分析（对仿真波形进行分析）

32位数据输入、输出都分为两个A、B端口，32位由5个拨号开关控制。

1. **电路图**

（开发工具中显示的电路模块图）



1. **引脚配置**

（引脚约束文件的内容，描述主要配置情况）

# 开启比特流压缩，优化 .bit 文件大小

|  |
| --- |
| set\_property BITSTREAM.GENERAL.COMPRESS TRUE [current\_design]  # Switch，开关  set\_property PULLDOWN true [get\_ports sw]  set\_property IOSTANDARD LVCMOS18 [get\_ports sw]  set\_property PACKAGE\_PIN T3 [get\_ports {sw[1]}]  set\_property PACKAGE\_PIN U3 [get\_ports {sw[2]}]  set\_property PACKAGE\_PIN T4 [get\_ports {sw[3]}]  set\_property PACKAGE\_PIN V3 [get\_ports {sw[4]}]  set\_property PACKAGE\_PIN V4 [get\_ports {sw[5]}]  set\_property PACKAGE\_PIN W4 [get\_ports {sw[6]}]  set\_property PACKAGE\_PIN Y4 [get\_ports {sw[7]}]  set\_property PACKAGE\_PIN Y6 [get\_ports {sw[8]}]  set\_property PACKAGE\_PIN W7 [get\_ports {sw[9]}]  set\_property PACKAGE\_PIN Y8 [get\_ports {sw[10]}]  set\_property PACKAGE\_PIN Y7 [get\_ports {sw[11]}]  set\_property PACKAGE\_PIN T1 [get\_ports {sw[12]}]  set\_property PACKAGE\_PIN U1 [get\_ports {sw[13]}]  set\_property PACKAGE\_PIN U2 [get\_ports {sw[14]}]  set\_property PACKAGE\_PIN W1 [get\_ports {sw[15]}]  set\_property PACKAGE\_PIN W2 [get\_ports {sw[16]}]  set\_property PACKAGE\_PIN Y1 [get\_ports {sw[17]}]  set\_property PACKAGE\_PIN AA1 [get\_ports {sw[18]}]  set\_property PACKAGE\_PIN V2 [get\_ports {sw[19]}]  set\_property PACKAGE\_PIN Y2 [get\_ports {sw[20]}]  set\_property PACKAGE\_PIN AB1 [get\_ports {sw[21]}]  set\_property PACKAGE\_PIN AB2 [get\_ports {sw[22]}]  set\_property PACKAGE\_PIN AB3 [get\_ports {sw[23]}]  set\_property PACKAGE\_PIN AB5 [get\_ports {sw[24]}]  set\_property PACKAGE\_PIN AA6 [get\_ports {sw[25]}]  set\_property PACKAGE\_PIN R2 [get\_ports {sw[26]}]  set\_property PACKAGE\_PIN R3 [get\_ports {sw[27]}]  set\_property PACKAGE\_PIN T6 [get\_ports {sw[28]}]  set\_property PACKAGE\_PIN R6 [get\_ports {sw[29]}]  set\_property PACKAGE\_PIN U7 [get\_ports {sw[30]}]  set\_property PACKAGE\_PIN AB7 [get\_ports {sw[31]}]  set\_property PACKAGE\_PIN AB8 [get\_ports {sw[32]}]  # Switch Button，按钮  set\_property IOSTANDARD LVCMOS18 [get\_ports swb]  set\_property PACKAGE\_PIN R4 [get\_ports {swb[1]}]  set\_property PACKAGE\_PIN AA4 [get\_ports {swb[2]}]  set\_property PACKAGE\_PIN AB6 [get\_ports {swb[3]}]  set\_property PACKAGE\_PIN T5 [get\_ports {swb[4]}]  set\_property PACKAGE\_PIN V8 [get\_ports {swb[5]}]  set\_property PACKAGE\_PIN AA8 [get\_ports {swb[6]}]  # LED  set\_property IOSTANDARD LVCMOS18 [get\_ports led]  set\_property PACKAGE\_PIN R1 [get\_ports {led[1]}]  set\_property PACKAGE\_PIN P2 [get\_ports {led[2]}]  set\_property PACKAGE\_PIN P1 [get\_ports {led[3]}]  set\_property PACKAGE\_PIN N2 [get\_ports {led[4]}]  set\_property PACKAGE\_PIN M1 [get\_ports {led[5]}]  set\_property PACKAGE\_PIN M2 [get\_ports {led[6]}]  set\_property PACKAGE\_PIN L1 [get\_ports {led[7]}]  set\_property PACKAGE\_PIN J2 [get\_ports {led[8]}]  set\_property PACKAGE\_PIN G1 [get\_ports {led[9]}]  set\_property PACKAGE\_PIN E1 [get\_ports {led[10]}]  set\_property PACKAGE\_PIN D2 [get\_ports {led[11]}]  set\_property PACKAGE\_PIN A1 [get\_ports {led[12]}]  set\_property PACKAGE\_PIN L3 [get\_ports {led[13]}]  set\_property PACKAGE\_PIN G3 [get\_ports {led[14]}]  set\_property PACKAGE\_PIN K4 [get\_ports {led[15]}]  set\_property PACKAGE\_PIN G4 [get\_ports {led[16]}]  set\_property PACKAGE\_PIN K1 [get\_ports {led[17]}]  set\_property PACKAGE\_PIN J1 [get\_ports {led[18]}]  set\_property PACKAGE\_PIN H2 [get\_ports {led[19]}]  set\_property PACKAGE\_PIN G2 [get\_ports {led[20]}]  set\_property PACKAGE\_PIN F1 [get\_ports {led[21]}]  set\_property PACKAGE\_PIN E2 [get\_ports {led[22]}]  set\_property PACKAGE\_PIN D1 [get\_ports {led[23]}]  set\_property PACKAGE\_PIN B1 [get\_ports {led[24]}]  set\_property PACKAGE\_PIN B2 [get\_ports {led[25]}]  set\_property PACKAGE\_PIN N3 [get\_ports {led[26]}]  set\_property PACKAGE\_PIN M3 [get\_ports {led[27]}]  set\_property PACKAGE\_PIN K3 [get\_ports {led[28]}]  set\_property PACKAGE\_PIN H3 [get\_ports {led[29]}]  set\_property PACKAGE\_PIN N4 [get\_ports {led[30]}]  set\_property PACKAGE\_PIN L4 [get\_ports {led[31]}]  set\_property PACKAGE\_PIN J4 [get\_ports {led[32]}]  # 数码管相关  set\_property IOSTANDARD LVCMOS18 [get\_ports seg]  set\_property PACKAGE\_PIN H19 [get\_ports {seg[7]}]  set\_property PACKAGE\_PIN G20 [get\_ports {seg[6]}]  set\_property PACKAGE\_PIN J22 [get\_ports {seg[5]}]  set\_property PACKAGE\_PIN K22 [get\_ports {seg[4]}]  set\_property PACKAGE\_PIN K21 [get\_ports {seg[3]}]  set\_property PACKAGE\_PIN H20 [get\_ports {seg[2]}]  set\_property PACKAGE\_PIN H22 [get\_ports {seg[1]}]  set\_property PACKAGE\_PIN J21 [get\_ports {seg[0]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports which]  set\_property PACKAGE\_PIN N22 [get\_ports {which[0]}]  set\_property PACKAGE\_PIN M21 [get\_ports {which[1]}]  set\_property PACKAGE\_PIN M22 [get\_ports {which[2]}]  set\_property -dict {IOSTANDARD LVCMOS18 PACKAGE\_PIN L21} [get\_ports enable]  set\_property -dict {IOSTANDARD LVCMOS18 PACKAGE\_PIN H4} [get\_ports clk]  # [Place 30-574] Poor placement for routing between an IO pin and BUFG.If this  # sub optimal condition is acceptable for this design, you may use the  # CLOCK\_DEDICATED\_ROUTE constraint in the .xdc file to demote this message to a  # WARNING. However, the use of this override is highly discouraged.  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets clk\_IBUF]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets swb\_IBUF[1]]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets swb\_IBUF[2]]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets swb\_IBUF[3]]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets swb\_IBUF[4]]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets swb\_IBUF[5]]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets swb\_IBUF[6]] |

1. **思考与探索**
2. 实验结果记录：

（实验操作的过程及结果记录）

使用5位开关提供读写的寄存器地址；

1位开关提供Write\_Reg信号；指定Write\_Reg=0时执行读操作；=1时执行写操作（只对2位复用开关有效）；

5位选择开关：在写操作时，选择32个指定数据之一作为写入数据。

1个按钮连到Clk，模拟时钟输入；另1个按钮提供Reset；再1个按钮作为读A端口/B端口的选择（可以不用，因为A/B端口地址相同）；

32位LED灯作为读出数据显示

1. 实验结论

（分析实验结果，给出实验结论）

最高位显示读取端口状态，三位显示写入端口地址，两位显示 A 读端口地址，最低两位显示 b 读端口地址。

32个寄存器，每个寄存器可以保存32个二进制位。

1. 问题与解决方案：

（整个实验过程中发生了什么问题？你是如何解决的。）

答：在板上操作时不太明白，经过仔细阅读代码后操作成功。

4、思考与探索

（1）选择8个寄存器执行读写操作，将实验结果记录到表中，结果是否符合预期；如果不符，分析原因。

答：符合。

|  |  |  |
| --- | --- | --- |
| 寄存器地址 | 写入数据 | 读出数据 |
| 10101 | 32’h | 32’h00000000 |
| 10110 | 32’hAAAAAAAA(无clk) | 32’h |
| 10111 | 32’h | 32’h00000000 |
| 11000 | 32’hFFFFFFFFF(无clk脉冲) | 32’h |
| 11001 | 32’h | 32’hFFFFFFFF |
| 11010 | 32’h | 32’h00000000(rst) |
| 11011 | 32’h | 32’h00000000 |
| 11100 | 32’hAABBCCDD | 32’h |

1. 修改基本寄存器堆模块，实现MIPS计算机的寄存器堆，以供后续MIPS CPU的设计使用。

|  |
| --- |
| module REGS(R\_Data\_A,R\_Data\_B,W\_Data,R\_Addr\_A,R\_Addr\_B,W\_Addr,Write\_Reg,rst,clk);  input clk;//写入时钟信号  input rst;//清零信号  input Write\_Reg;//写控制信号  input [4:0]R\_Addr\_A;//A端口读寄存器地址  input [4:0]R\_Addr\_B;//B端口读寄存器地址  input [4:0]W\_Addr;//写寄存器地址  input [31:0]W\_Data;//写入数据  output [31:0]R\_Data\_A;//A端口读出数据  output [31:0]R\_Data\_B;//B端口读出数据    integer i;  reg [31:0] REG\_Files[0:31];  initial  for(i=0;i<32;i=i+1) REG\_Files[i]<=0;  always@(posedge clk or posedge rst)  begin  if(rst)  for(i=0;i<32;i=i+1) REG\_Files[i]<=0;  else  if(Write\_Reg&&W\_Addr!=32'd0) REG\_Files[W\_Addr]<=W\_Data;  end  assign R\_Data\_A=REG\_Files[R\_Addr\_A];  assign R\_Data\_B=REG\_Files[R\_Addr\_B];  endmodule |