**组成原理实验课程第 3 次实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | 寄存器堆实现 | | | 班级 | 李涛 |
| 学生姓名 | 胡博浩 | 学号 | 2212998 | 指导老师 | 董前琨 |
| 实验地点 | 津南实验楼A308 | | 实验时间 | 2024.4.25 | |

1. **实验目的**

（1）熟悉并掌握MIPS计算机中寄存器堆的原理和设计方法。

（2）初步了解MIPS指令结构和源操作数/目的操作数的概念。

（3）熟悉并运用verilog语言进行电路设计。

（4）为后续设计cpu的实验打下基础。

1. **实验内容说明**

请结合实验指导手册中的实验三（寄存器堆实验）完成对寄存器堆进行64位位拓展的改进实验，注意以下几点：

（1）原始的32个32位寄存器堆，需要修改成16个64位的寄存器堆，注意地址和位宽变化。

（2）在display模块，注意读出数据和写入的数据都应是64位，lcd屏上的格子需要调整分配。此外，input\_sel等相关信号注意位宽是否调整。

（3）本次实验没有仿真，直接上试验箱验证，实验报告中注意对实验结果的介绍，分析和总结需要详细说明。

**实验原理图**



将原始的32个32位寄存器堆，修改成了16个64位的寄存器堆。为了控制16个寄存器的读取，将地址位数变为4位。由于是将两个32位寄存器合并为一个64位寄存器，因此添加high\_low信号来控制高低位的切换。

1. **实验步骤**

（1）regfile.v

a）将读取数据的改为64位，将寄存器数量改为16个，并修改读端口1、2和调试端口的逻辑，将原来32个改为16个。

b）将读取地址的raddr、waddr、test\_addr通通改为4位。

module regfile(

input clk,

input wen,

input [3 :0] raddr1,

input [3 :0] raddr2,

input [3 :0] waddr,

input [63:0] wdata,

output reg [63:0] rdata1,

output reg [63:0] rdata2,

input [3 :0] test\_addr,

output reg [63:0] test\_data

);

reg [63:0] rf[15:0];

always @(posedge clk)

begin

if (wen)

begin

rf[waddr] <= wdata;

end

end

//读端口1

always @(\*)

begin

case (raddr1)

5'd1 : rdata1 <= rf[1 ];

5'd2 : rdata1 <= rf[2 ];

5'd3 : rdata1 <= rf[3 ];

5'd4 : rdata1 <= rf[4 ];

5'd5 : rdata1 <= rf[5 ];

5'd6 : rdata1 <= rf[6 ];

5'd7 : rdata1 <= rf[7 ];

5'd8 : rdata1 <= rf[8 ];

5'd9 : rdata1 <= rf[9 ];

5'd10: rdata1 <= rf[10];

5'd11: rdata1 <= rf[11];

5'd12: rdata1 <= rf[12];

5'd13: rdata1 <= rf[13];

5'd14: rdata1 <= rf[14];

5'd15: rdata1 <= rf[15];

default : rdata1 <= 64'd0;

endcase

end

//读端口2

always @(\*)

begin

case (raddr2)

5'd1 : rdata2 <= rf[1 ];

5'd2 : rdata2 <= rf[2 ];

5'd3 : rdata2 <= rf[3 ];

5'd4 : rdata2 <= rf[4 ];

5'd5 : rdata2 <= rf[5 ];

5'd6 : rdata2 <= rf[6 ];

5'd7 : rdata2 <= rf[7 ];

5'd8 : rdata2 <= rf[8 ];

5'd9 : rdata2 <= rf[9 ];

5'd10: rdata2 <= rf[10];

5'd11: rdata2 <= rf[11];

5'd12: rdata2 <= rf[12];

5'd13: rdata2 <= rf[13];

5'd14: rdata2 <= rf[14];

5'd15: rdata2 <= rf[15];

default : rdata2 <= 64'd0;

endcase

end

//调试端口，读出寄存器值显示在触摸屏上

always @(\*)

begin

case (test\_addr)

5'd1 : test\_data <= rf[1 ];

5'd2 : test\_data <= rf[2 ];

5'd3 : test\_data <= rf[3 ];

5'd4 : test\_data <= rf[4 ];

5'd5 : test\_data <= rf[5 ];

5'd6 : test\_data <= rf[6 ];

5'd7 : test\_data <= rf[7 ];

5'd8 : test\_data <= rf[8 ];

5'd9 : test\_data <= rf[9 ];

5'd10: test\_data <= rf[10];

5'd11: test\_data <= rf[11];

5'd12: test\_data <= rf[12];

5'd13: test\_data <= rf[13];

5'd14: test\_data <= rf[14];

5'd15: test\_data <= rf[15];

default : test\_data <= 64'd0;

endcase

end

endmodule

（2）regfile\_display.v

a）添加high\_low信号，控制高低位的输入

b）对读取地址的修改为4位，读取数据的修改为64位

c）在写入数据的时候，通过high\_low来控制高低位的输入

d）在读取数据和显示数据的时候，将64位数据分为两个32位数据、并列排放在一行内

module regfile\_display(

//时钟与复位信号

input clk,

input resetn, //后缀"n"代表低电平有效

//拨码开关，用于产生写使能和选择输入数

input wen,

input [1:0] input\_sel,

//0为低32位，1为高32位

input high\_low,

//led灯，用于指示写使能信号，和正在输入什么数据

output led\_wen,

output led\_waddr, //指示输入写地址

output led\_wdata, //指示输入写数据

output led\_raddr1, //指示输入读地址1

output led\_raddr2, //指示输入读地址2

//触摸屏相关接口，不需要更改

output lcd\_rst,

output lcd\_cs,

output lcd\_rs,

output lcd\_wr,

output lcd\_rd,

inout[15:0] lcd\_data\_io,

output lcd\_bl\_ctr,

inout ct\_int,

inout ct\_sda,

output ct\_scl,

output ct\_rstn

);

//-----{LED显示}begin

assign led\_wen = wen;

assign led\_raddr1 = (input\_sel==2'd0);

assign led\_raddr2 = (input\_sel==2'd1);

assign led\_waddr = (input\_sel==2'd2);

assign led\_wdata = (input\_sel==2'd3);

//-----{LED显示}end

//-----{调用寄存器堆模块}begin

//寄存器堆多增加一个读端口，用于在触摸屏上显示16个寄存器值

wire [63:0] test\_data;

wire [3 :0] test\_addr;

reg [3 :0] raddr1;

reg [3 :0] raddr2;

reg [3 :0] waddr;

reg [63:0] wdata;

wire [63:0] rdata1;

wire [63:0] rdata2;

regfile rf\_module(

.clk (clk ),

.wen (wen ),

.raddr1(raddr1),

.raddr2(raddr2),

.waddr (waddr ),

.wdata (wdata ),

.rdata1(rdata1),

.rdata2(rdata2),

.test\_addr(test\_addr),

.test\_data(test\_data)

);

//-----{调用寄存器堆模块}end

//---------------------{调用触摸屏模块}begin--------------------//

//-----{实例化触摸屏}begin

//此小节不需要更改

reg display\_valid;

reg [39:0] display\_name;

reg [31:0] display\_value;

wire [5 :0] display\_number;

wire input\_valid;

wire [31:0] input\_value;

lcd\_module lcd\_module(

.clk (clk ), //10Mhz

.resetn (resetn ),

//调用触摸屏的接口

.display\_valid (display\_valid ),

.display\_name (display\_name ),

.display\_value (display\_value ),

.display\_number (display\_number),

.input\_valid (input\_valid ),

.input\_value (input\_value ),

//lcd触摸屏相关接口，不需要更改

.lcd\_rst (lcd\_rst ),

.lcd\_cs (lcd\_cs ),

.lcd\_rs (lcd\_rs ),

.lcd\_wr (lcd\_wr ),

.lcd\_rd (lcd\_rd ),

.lcd\_data\_io (lcd\_data\_io ),

.lcd\_bl\_ctr (lcd\_bl\_ctr ),

.ct\_int (ct\_int ),

.ct\_sda (ct\_sda ),

.ct\_scl (ct\_scl ),

.ct\_rstn (ct\_rstn )

);

//-----{实例化触摸屏}end

//-----{从触摸屏获取输入}begin

//根据实际需要输入的数修改此小节，

//建议对每一个数的输入，编写单独一个always块

//16个寄存器显示在10~41号的显示块，故读地址为（display\_number-1）

assign test\_addr = (display\_number-6'd11)/2;

//当input\_sel为2'b00时，表示输入数为读地址1，即raddr1

always @(posedge clk)

begin

if (!resetn)

begin

raddr1 <= 4'd0;

end

else if (input\_valid && input\_sel==2'd0)

begin

raddr1 <= input\_value[3:0];

end

end

//当input\_sel为2'b01时，表示输入数为读地址2，即raddr2

always @(posedge clk)

begin

if (!resetn)

begin

raddr2 <= 4'd0;

end

else if (input\_valid && input\_sel==2'd1)

begin

raddr2 <= input\_value[3:0];

end

end

//当input\_sel为2'b10时，表示输入数为写地址，即waddr

always @(posedge clk)

begin

if (!resetn)

begin

waddr <= 4'd0;

end

else if (input\_valid && input\_sel==2'd2)

begin

waddr <= input\_value[3:0];

end

end

//当input\_sel为2'b11时，表示输入数为写数据，即wdata

always @(posedge clk)

begin

if (!resetn)

begin

wdata <= 64'd0;

end

else if (input\_valid && input\_sel==2'd3)

begin

if(high\_low==0)

wdata[31:0] <= input\_value;

else

wdata[63:32] <= input\_value;

end

end

//-----{从触摸屏获取输入}end

//-----{输出到触摸屏显示}begin

//根据需要显示的数修改此小节，

//触摸屏上共有44块显示区域，可显示44组32位数据

//44块显示区域从1开始编号，编号为1~44，

always @(posedge clk)

begin

if (display\_number >=6'd11 && display\_number <=6'd42 )

begin //块号11~42显示16个通用寄存器的值

display\_valid <= 1'b1;

if (display\_number%2==1)

begin

display\_name[39:16] <= "REG";

display\_name[15: 8] <= {4'b0011,4'b0000};

display\_name[7 : 0] <= {4'b0011,test\_addr[3:0]};

display\_value <= test\_data[63:32];

end

else

begin

display\_value <= test\_data[31:0];

end

end

else

begin

case(display\_number)

6'd1 : //显示读端口1的地址

begin

display\_valid <= 1'b1;

display\_name <= "RADD1";

display\_value <= raddr1;

end

6'd2 : //显示读端口1读出的数据

begin

display\_valid <= 1'b1;

display\_name <= "RDAT1";

display\_value <= rdata1[63:32];

end

6'd3 : //显示读端口1读出的数据

begin

display\_valid <= 1'b1;

display\_value <= rdata1[31:0];

end

6'd4 : //显示读端口2的地址

begin

display\_valid <= 1'b1;

display\_name <= "RADD2";

display\_value <= raddr2;

end

6'd5 : //显示读端口2读出的数据

begin

display\_valid <= 1'b1;

display\_name <= "RDAT2";

display\_value <= rdata2[63:32];

end

6'd6 : //显示读端口2读出的数据

begin

display\_valid <= 1'b1;

display\_value <= rdata2[31:0];

end

6'd7 : //显示写端口的地址

begin

display\_valid <= 1'b1;

display\_name <= "WADDR";

display\_value <= waddr;

end

6'd8 : //显示写端口写入的数据

begin

display\_valid <= 1'b1;

display\_name <= "WDATA";

display\_value <= wdata[63:32];

end

6'd9 : //显示写端口写入的数据

begin

display\_valid <= 1'b1;

display\_value <= wdata[31:0];

end

default :

begin

display\_valid <= 1'b0;

display\_name <= 40'd0;

display\_value <= 64'd0;

end

endcase

end

end

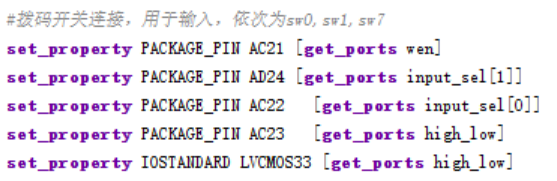
//-----{输出到触摸屏显示}end

//----------------------{调用触摸屏模块}end---------------------//

endmodule

（3）regfile.xdc

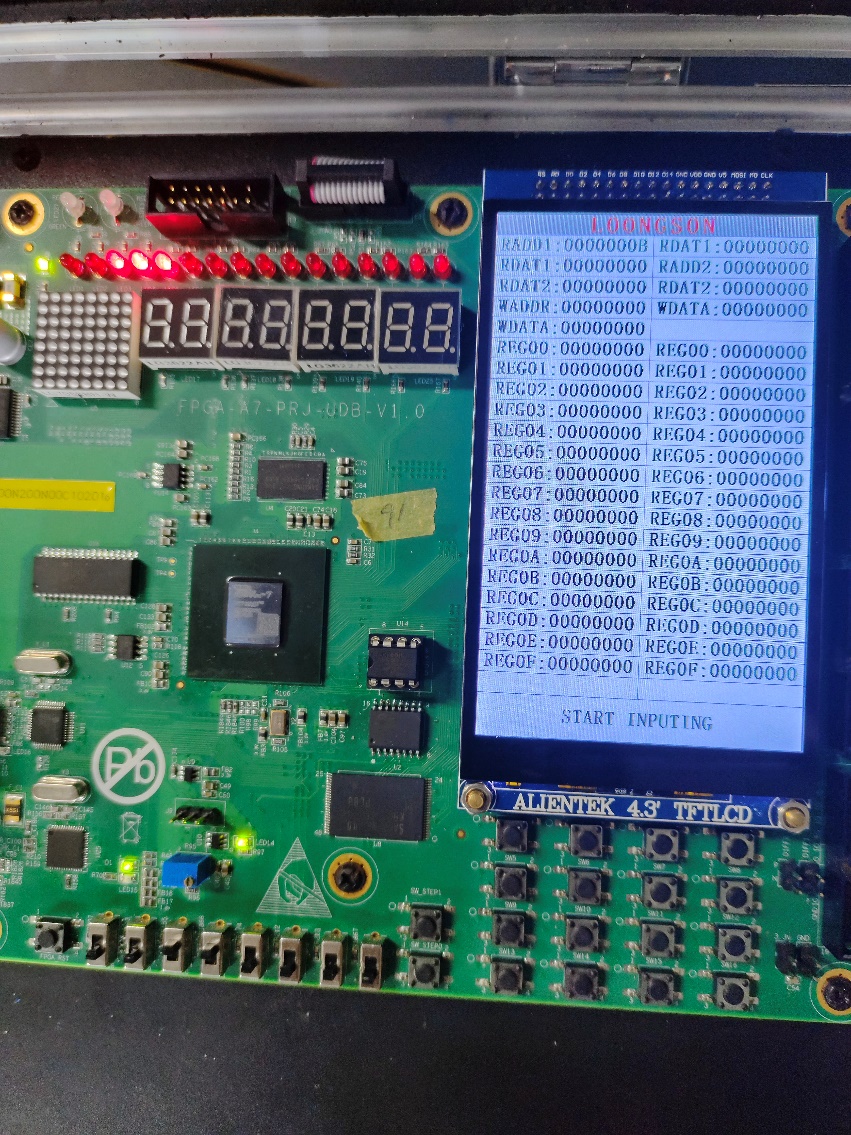
添加high\_low拨码开关，用来控制高低位的切换。



1. **实验结果分析**

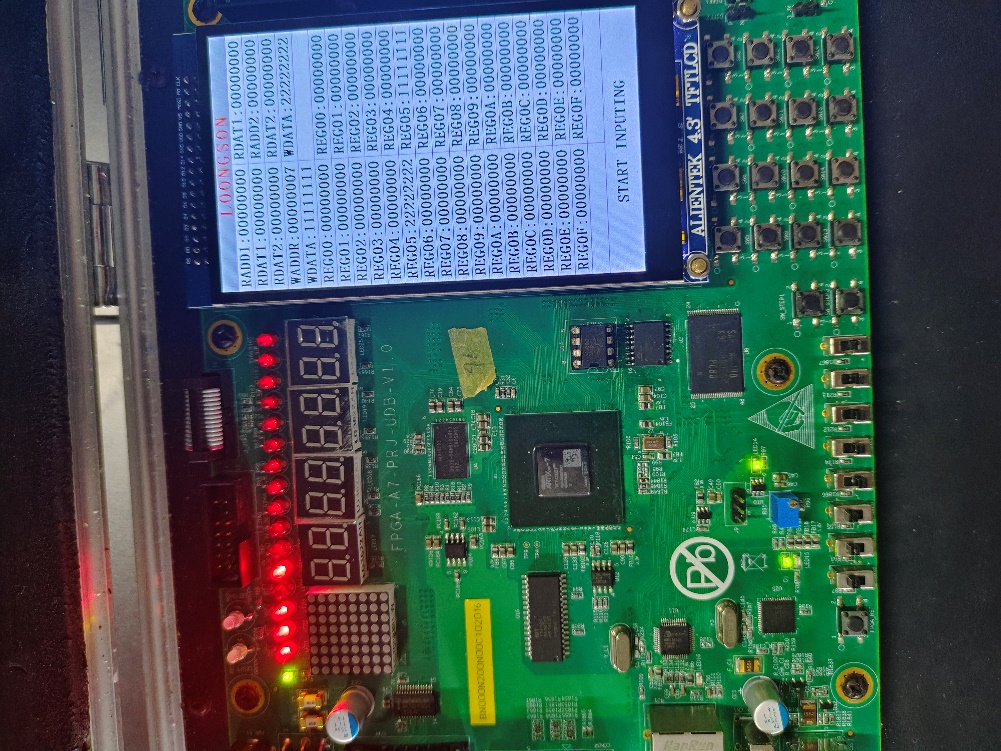
（1）上箱实验

a）原始：



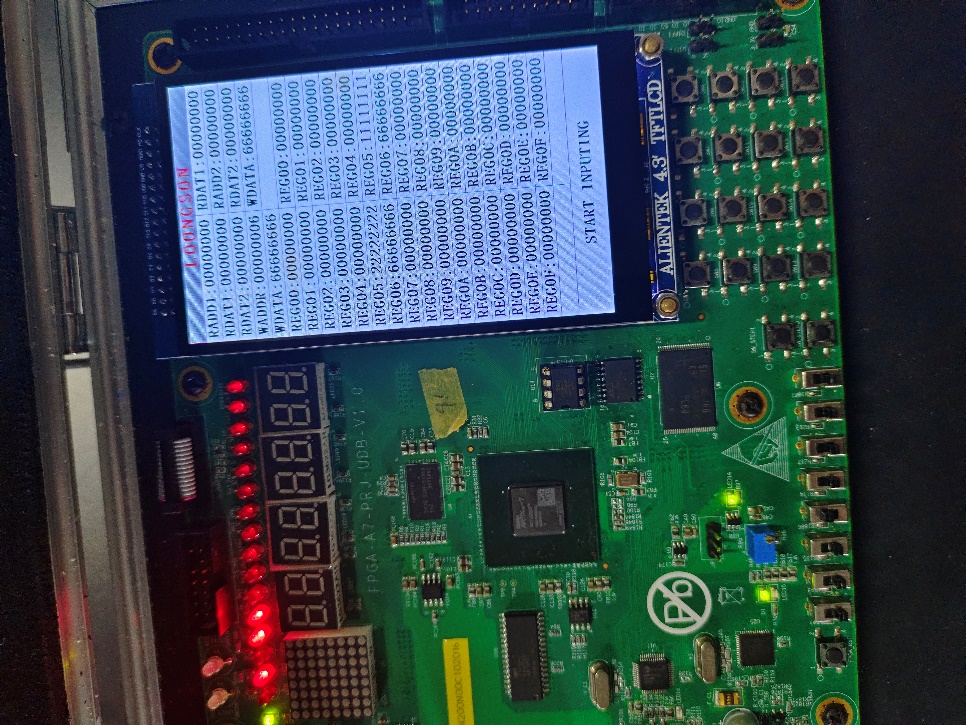
b）写入第一个寄存器：

向寄存器5写入64位数据0x2222222211111111



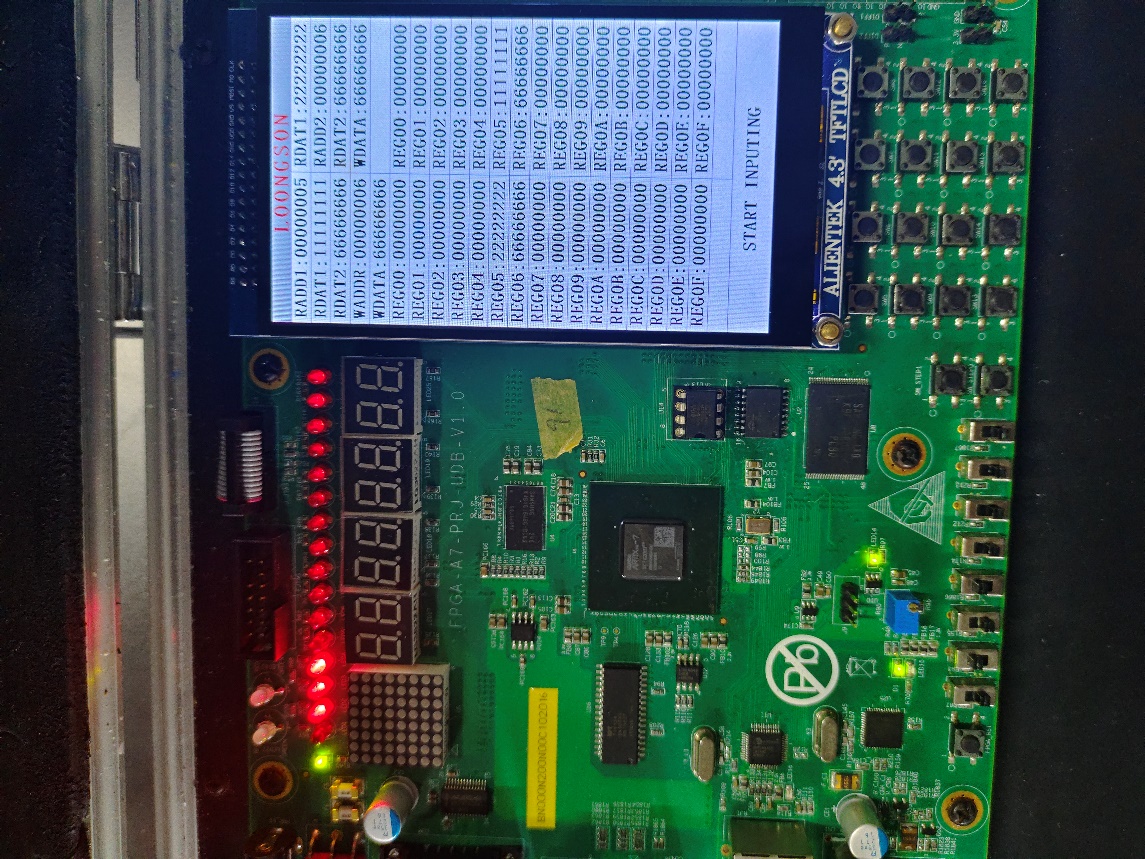
c）写入第二个寄存器：

向寄存器6写入64位数据6666666666666666



d）读取寄存器：

将5号寄存器的数据读入到radd1中，将6号寄存器的数据读入到radd2中。



**由图可知，寄存器堆的写入和读取功能正常，实验成功！**

1. **总结感想**

这次对寄存器堆进行64位位拓展的改进实验，让我收获颇丰。

首先，通过这次实验，我深入理解了寄存器堆的设计原理以及如何进行位宽拓展。从32位寄存器堆拓展到64位，不仅仅是简单地加倍，还需要考虑地址映射、数据位宽调整等细节，这提高了我对数字电路设计的抽象能力和逻辑思维能力。

其次，实验中我需要对相关信号和模块进行调整，以确保数据的正确读写和显示。这锻炼了我的Verilog编程能力和硬件设计能力，让我更加熟悉了Vivado工具的使用方法。

另外，我学习了利用visio画原理图，这帮我梳理了实验的逻辑、对整个改进过程进行了整理归纳。

最重要的是，通过实验结果的分析和总结，我发现了改进后的寄存器堆能够正常工作，并且能够正确地读写64位数据。这增强了我的信心，让我相信自己在硬件设计领域的能力不断提升。