组成原理课程第 三 次实报告

实验名称：寄存器堆实现

学号： 2213041 姓名： 李雅帆 班次： 李涛老师

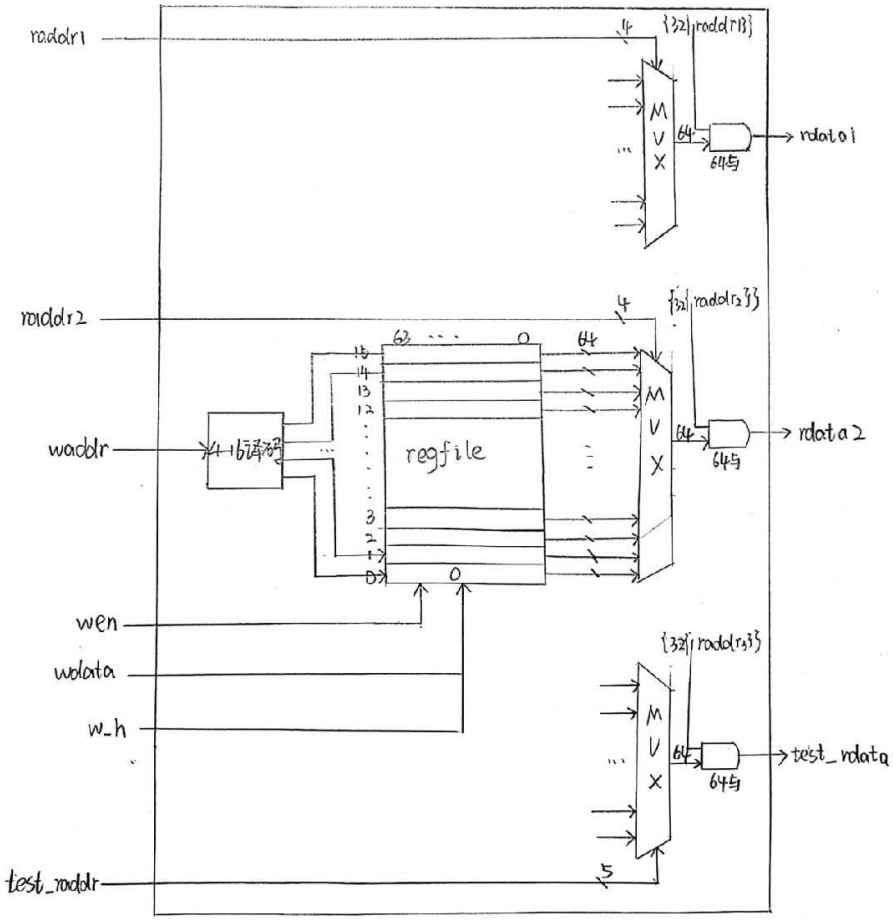
一、实验目的

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。

二、实验内容说明

将原始的 32 个 32 位寄存器堆，修改成 16 个 64 位的寄存器堆，注意地址和位宽变化。要注意在 display 模块，写入和读出的数据都应该是 64 位，lcd 屏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。

三、实验原理图



四、实验步骤

1．在 regfile.v 文件中

①数据宽度修改：

输入数据总线宽度为 input [63:0] wdata，输出数据总线宽度为 output reg [63:0] rdata1, rdata2, test_data。

```
//input       [63:0] wdata,
input       [63 :0] wdata,
// output reg [63:0] rdata1,
output reg [63:0] rdata1,
//output reg [63:0] rdata2,
output reg [63:0] rdata2,
// input       [4 :0] test_addr,
input       [6 :0] test_addr,
//output reg [63:0] test_data
 output reg [63:0] test_data
);
```

②寄存器文件数组修改：

在第一个版本中，寄存器文件数组声明为 reg [63:0] rf[15:0]，有 16 个寄存器。

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

③Case 语句修改：

case 语句被修改以匹配新的数据宽度和寄存器宽度，对于每个寄存器的访问使用了 64 位宽度的寄存器。

④默认值修改：

默认值也随之被调整以匹配新的数据宽度，默认值修改为 64'd0。

```
//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];
5'd16: rdata1 <= rf[16];
5'd17: rdata1 <= rf[17];
5'd18: rdata1 <= rf[18];
5'd19: rdata1 <= rf[19];
5'd20: rdata1 <= rf[20];
5'd21: rdata1 <= rf[21];
5'd22: rdata1 <= rf[22];
5'd23: rdata1 <= rf[23];
5'd24: rdata1 <= rf[24];
5'd25: rdata1 <= rf[25];
5'd26: rdata1 <= rf[26];
5'd27: rdata1 <= rf[27];
5'd28: rdata1 <= rf[28];
5'd29: rdata1 <= rf[29];
5'd30: rdata1 <= rf[30];
5'd31: rdata1 <= rf[31];*/
4'd1 : rdata1 <= rf[1 ];
4'd2 : rdata1 <= rf[2 ];
4'd3 : rdata1 <= rf[3 ];
4'd4 : rdata1 <= rf[4 ];
4'd5 : rdata1 <= rf[5 ];
4'd6 : rdata1 <= rf[6 ];
4'd7 : rdata1 <= rf[7 ];
4'd8 : rdata1 <= rf[8 ];
4'd9 : rdata1 <= rf[9 ];
4'd10: rdata1 <= rf[10];
4'd11: rdata1 <= rf[11];
4'd12: rdata1 <= rf[12];
4'd13: rdata1 <= rf[13];
```

```
            4'd14: rdata1 <= rf[14];
            4'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];*/
            4'd1 : rdata2 <= rf[1 ];
            4'd2 : rdata2 <= rf[2 ];
            4'd3 : rdata2 <= rf[3 ];
            4'd4 : rdata2 <= rf[4 ];
            4'd5 : rdata2 <= rf[5 ];
            4'd6 : rdata2 <= rf[6 ];
            4'd7 : rdata2 <= rf[7 ];
            4'd8 : rdata2 <= rf[8 ];
            4'd9 : rdata2 <= rf[9 ];
            4'd10: rdata2 <= rf[10];
            4'd11: rdata2 <= rf[11];
            4'd12: rdata2 <= rf[12];
            4'd13: rdata2 <= rf[13];
            4'd14: rdata2 <= rf[14];
            4'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];*/
            ////////////////////////////////
            /*7'd1 : test_data <= rf[1 ];
            7'd2 : test_data <= rf[2 ];
            7'd3 : test_data <= rf[3 ];
            7'd4 : test_data <= rf[4 ];
            7'd5 : test_data <= rf[5 ];
            7'd6 : test_data <= rf[6 ];
            7'd7 : test_data <= rf[7 ];
            7'd8 : test_data <= rf[8 ];
            7'd9 : test_data <= rf[9 ];
            7'd10: test_data <= rf[10];
            7'd11: test_data <= rf[11];
            7'd12: test_data <= rf[12];
            7'd13: test_data <= rf[13];
            7'd14: test_data <= rf[14];
            7'd15: test_data <= rf[15];*/
            7'd0 : test_data <= rf[0 ];
```

```
                7'd1 : test_data <= rf[0 ];
                7'd2 : test_data <= rf[1 ];
                7'd3 : test_data <= rf[1 ];
                7'd4 : test_data <= rf[2 ];
                7'd5 : test_data <= rf[2 ];
                7'd6 : test_data <= rf[3 ];
                7'd7 : test_data <= rf[3 ];
                7'd8 : test_data <= rf[4 ];
                7'd9 : test_data <= rf[4 ];
                7'd10: test_data <= rf[5 ];
                7'd11: test_data <= rf[5 ];
                7'd12: test_data <= rf[6 ];
                7'd13: test_data <= rf[6 ];
                7'd14: test_data <= rf[7 ];
                7'd15: test_data <= rf[7 ];
                7'd16 : test_data <= rf[8 ];
                7'd17 : test_data <= rf[8 ];
                7'd18 : test_data <= rf[9 ];
                7'd19 : test_data <= rf[9 ];
                7'd20 : test_data <= rf[10 ];
                7'd21 : test_data <= rf[10 ];
                7'd22 : test_data <= rf[11 ];
                7'd23: test_data <= rf[11 ];
                7'd24: test_data <= rf[12 ];
                7'd25: test_data <= rf[12];
                7'd26: test_data <= rf[13];
                7'd27: test_data <= rf[13];
                7'd28: test_data <= rf[14];
                7'd29: test_data <= rf[14];
                7'd30: test_data <= rf[15];
                7'd31: test_data <= rf[15];
                default : test_data <= 64'd0;
            endcase
        end
    endmodule
```

2.在 regfile_display.v 文件中

①输入选择信号 (input_sel) 修改：

input_sel 的宽度被扩展为 3 位，范围为 input [2:0] input_sel。这样的修改使得可以支持更多的输入选择。

```
    //拨码开关，用于产生写使能和选择输入数
    input wen,
    //input [2:0] input_sel,
      input [2:0] input_sel,
```

②寄存器和数据宽度修改：

寄存器数组 rf 的宽度被修改为 reg [3:0] rf[15:0];，每个寄存器的宽度为 64 位。同时，测试数据的宽度也从 wire [31:0] test_data; 修改为 wire [63:0] test_data;。

③读写地址宽度修改：

读写地址的宽度被修改为 4 位，范围为 reg [3:0] raddr1, raddr2, waddr;。使得读写地址范围减小，但是保持了足够的范围来访问寄存器数组。

④写数据宽度修改：

写数据的宽度被修改为 64 位，范围为 reg [63:0] wdata;。

```
//wire [63:0] test_data;
    wire [63:0] test_data;
   // wire [4 :0] test_addr;
   wire [6 :0] test_addr;

    //reg  [4 :0] raddr1;
    reg  [3 :0] raddr1;
  //  reg  [4 :0] raddr2;
     reg  [3:0] raddr2;
   // reg  [4 :0] waddr;
   reg  [3 :0] waddr;

   // reg  [63:0] wdata;
    reg  [63:0] wdata;
    //wire [63:0] rdata1;
    wire [63:0] rdata1;
    //wire [63:0] rdata2;
    wire [63:0] rdata2;
```

3.在 regfile.xdc 文件中

①新增输入端口：

添加了一个新的输入端口 input_sel[2]，并将其对应的引脚设置为 AD24。

②修改输入端口数量：

在原有的输入端口 input_sel[1] 和 input_sel[0] 后面新增了一个输入端口

input_sel[2]。

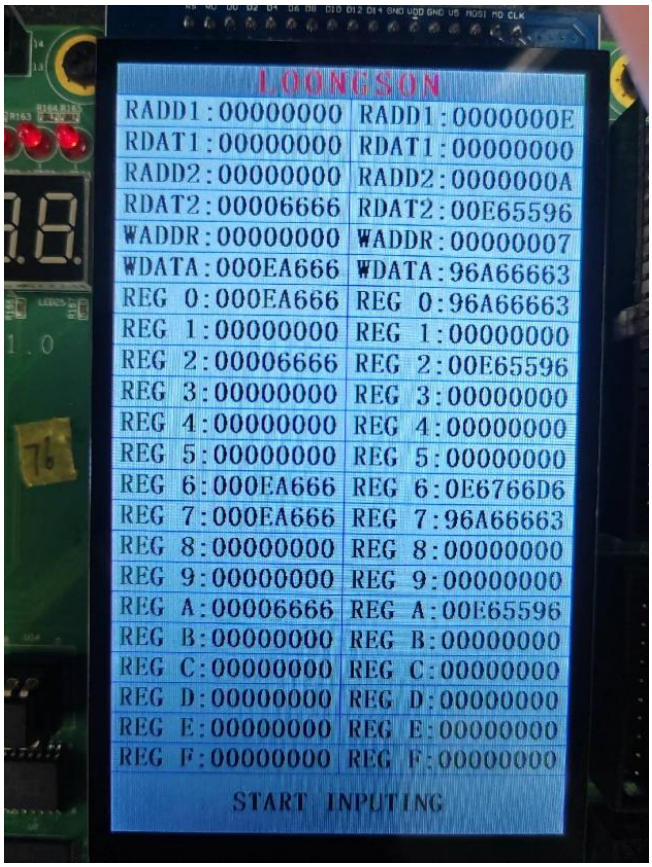
③更新 IO 标准设置：

对新增的输入端口 input_sel[2] 进行了 IO 标准设置。

```
#led灯连接，用于输出
set_property PACKAGE_PIN H7 [get_ports led_wen]
set_property PACKAGE_PIN D5 [get_ports led_waddr]
set_property PACKAGE_PIN A3 [get_ports led_wdata]
set_property PACKAGE_PIN A5 [get_ports led_raddr1]
set_property PACKAGE_PIN A4 [get_ports led_raddr2]
```

四、实验结果分析

由实验箱的结果图可以看出，所有的数据为全部写入。



五、总结感想

这次的实验让我更深入地了解了 MIPS 计算机中寄存器堆的原理和设计方法。

通过将原始的 32 个 32 位寄存器堆修改成 16 个 64 位的寄存器堆，我不仅仔细考虑了地址和位宽的变化，还深入理解了如何在 Verilog 中进行电路设计，以及如何适应不同位宽的数据操作。特别是在调整 display 模块和 lcd 屏的格子分配时，我更加深入地理解了数据的存储和展示方式对电路设计的影响。