组成原理实验课程第 3 次实报告

实验名称	寄存器堆改进			班级	李涛
学生姓名	阿斯雅	学号	2210737	指导老师	董前琨
实验地点	实验楼 A 区 308		实验时间	4/18 中午	

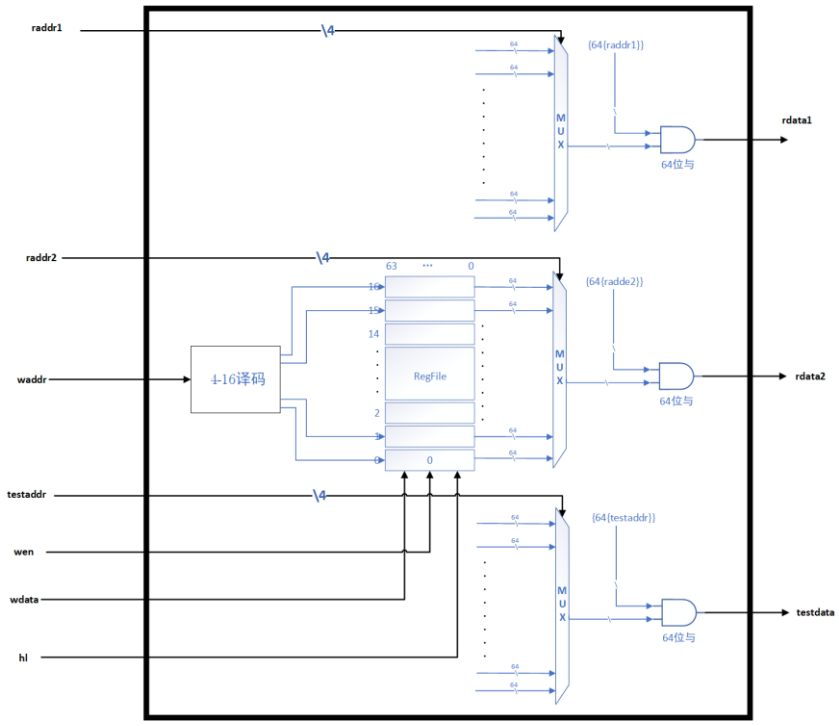
一、 实验目的

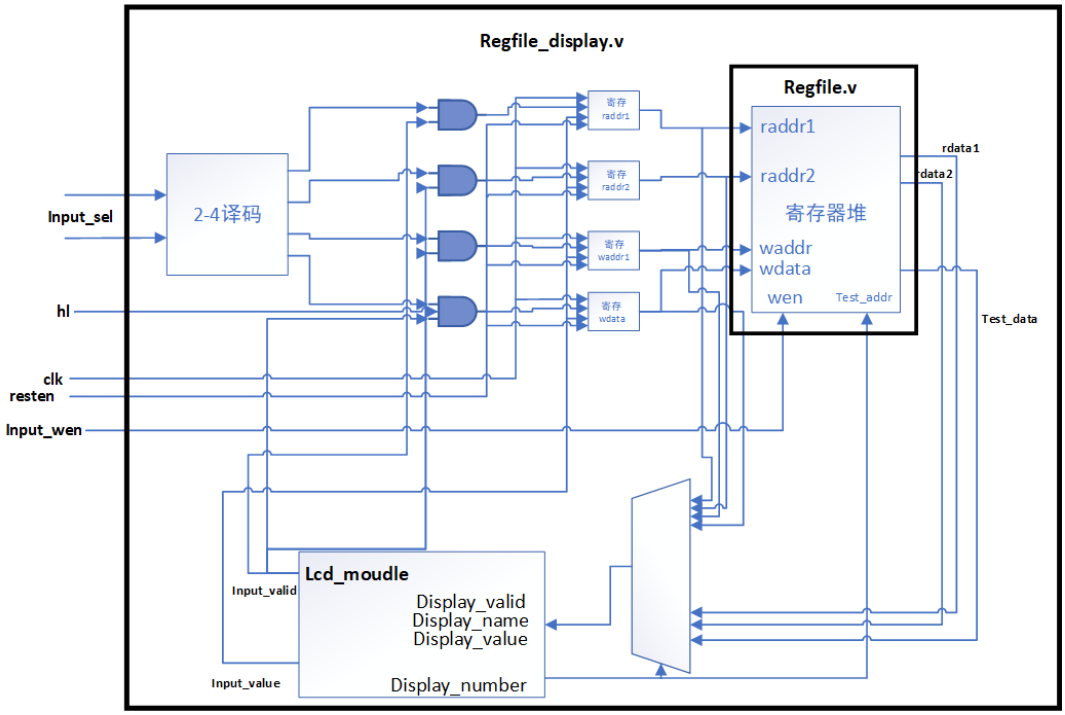
1. 熟悉并掌握 MIPS 计算机中寄存器堆的原理和设计方法。
2. 初步了解 MIPS 指令结构和源操作数/目的操作数的概念。
3. 熟悉并运用 verilog 语言进行电路设计。
4. 为后续设计 CPU 实验打下基础

二、 实验内容说明

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：将原始的 32 个 32 位寄存器堆，修改成 16 个 64 位的寄存器堆。

三、实验原理图





四、 实验步骤

4.1、分析

首先，我们知道原先的 32 个寄存器保存在一个名为 rf 的数组中，因此，我们需要将其元素个数和元素位宽改为 16 个和 64 位。随后，由于寄存器个数改为了 16 个，因此需要将查询地址的位宽也相应修改。原先在 32 个寄存器下地址的位宽是 5 位，所以现在需要将其修改为 4 位，以便可以查询到全部寄存器。接着，我们需要对 case 语句做相应的调整，现在只需考虑 16 种情况，其中包含了 0 号寄存器的特殊情况。

修改完上述基础设计模块后，我们需要对显示文件 display 模块做相应的调整。首先，显示屏上的一个格子最多能显示 32 位数据，因此，如何显示 64 位数据成为了一个关键问题。我想到可以采用拼接的方式解决这个问题，即将 64 位数据拆分为高 32 位和低 32 位，然后分别显示在连续的两个格子中。为了在连续的两个格子中显示相同的数据内容，我们可以将当前的 test_addr 右移一位，这样每两个格子才会取到下一个数。连续的两个格子可以通过它们的编号的奇偶数来区分，奇数格子显示高位，偶数格子显示低位。解决了 display_value 之后，我们还需要处理 display_name，因为 16 个数字可以用一个字符来表示，所以在 “REG” 之后我们直接接上原先 display_name 的最后一个字符。我们可以在当前 display_name 的最后一个字符上显示辨别高低位的字符，如果是高位则显示 “H”，否则显示 “L”。

修改完 display 模块后，我们还需要解决一个关键问题，即除了显示屏能显示的最大位数是 32 位之外，我们从显示屏上输入的 input_value 的最大位数也是 32 位。因此，要输入 64 位数据，需要分两次输入，第一次输入高 32 位，第二次输入低 32 位。为了解决这个问题，我新增了一个片选信号 hl，用来区分当前输入的是高位还是低位。当拨码开关的值为 0 时，表示输入的是低位，否则是高 32 位。

通过以上修改和解决关键问题，我们就完成了对设计模块的修改。其他的小问题可以在实验过程中逐步调整。

4.2、修改代码

修改 Regfile.v 文件

```
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 @(*)
  begin
      case (raddr1)
          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
```

```
    always @(*)
    begin
        case (test_addr)
            4'd1 : test_data <= rf[1 ];
            4'd2 : test_data <= rf[2 ];
            4'd3 : test_data <= rf[3 ];
            4'd4 : test_data <= rf[4 ];
            4'd5 : test_data <= rf[5 ];
            4'd6 : test_data <= rf[6 ];
            4'd7 : test_data <= rf[7 ];
            4'd8 : test_data <= rf[8 ];
            4'd9 : test_data <= rf[9 ];
            4'd10: test_data <= rf[10];
            4'd11: test_data <= rf[11];
            4'd12: test_data <= rf[12];
            4'd13: test_data <= rf[13];
            4'd14: test_data <= rf[14];
            4'd15: test_data <= rf[15];

            default : test_data <= 63'd0;
        endcase
    end
endmodule
```

修改 display.v 文件

```
module regfile_display(
    //?????λ???
    input clk,
    input resetn,     //???"n"??????????Ч

    //?????????????±?????????????
    input wen,
    input [1:0] input_sel,
    input hl,

    //led?????????±??????????????????????????
    output led_wen,
    output led_waddr,     //??????±???
    output led_wdata,     //??????±????
    output led_raddr1,    //??????????1
    output led_raddr2,    //??????????2

    //????????????????????
    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
```

```
assign test_addr = (display_number-5'd10)>>1;
//??input_sel?2'b00??????????????????1????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

//??input_sel?2'b01??????????????????2????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
```

```
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

//??input_sel?2'b11????????????? ? ???????wdata
always @(posedge clk)
begin
    if (!resetn)
    begin
        wdata  <= 64'd0;
    end
    else if (input_valid && input_sel==2'd3&&hl==0)
    begin
        wdata[63:32]  <= input_value;
    end
    else if(input_valid && input_sel==2'd3&&hl==1)
    begin
     wdata[31:0] <= input_value;
     end

end
```

```
always @(posedge clk)
begin
    if (display_number >6'd9 && display_number <6'd42&&display_number%2==0 )
    begin //????7~38????32?????u???????
        display_valid <= 1'b1;
        display_name[39:16] <= "REG";
        display_name[15: 8] <= {4'b0011,test_addr[3:0]};
        display_name[7 : 0] <= "H";
        display_value <= test_data[63:32];
      end
    else if(display_number >6'd9 && display_number <6'd42&&display_number%2==1)
    begin //????7~38????32?????u???????
              display_valid <= 1'b1;
              display_name[39:16] <= "REG";
              display_name[15: 8] <= {4'b0011,test_addr[3:0]};
              display_name[7 : 0] <= "L";
              display_value <= test_data[31:0];
     end
    else
    begin
        case(display_number)
            6'd1 : //????????1????
            begin
                display_valid <= 1'b1;
                display_name  <= "RADD1";
                display_value <= raddr1;
            end
```

```
begin
    display_valid <= 1'b1;
    display_name  <= "RDAT1H";
    display_value <= rdata1[63:32];
end

6'd3 : //????????2????
begin
    display_valid <= 1'b1;
    display_name  <= "RDAT1L";
    display_value <= rdata1[31:0];
end
6'd4 : //????????2???????????
begin
    display_valid <= 1'b1;
    display_name  <= "RADD2";
    display_value <= raddr2;
end
6'd5 : //???¤??????
begin
    display_valid <= 1'b1;
    display_name  <= "RDAT2H";
    display_value <= rdata2[63:32];
end
```

```
            7'd7:
            begin
                display_valid <= 1'b1;
                display_name  <= "WADDR";
                display_value <= waddr;
            end
            8'd8:
            begin
                display_valid <= 1'b1;
                display_name  <= "WDATAH";
                display_value <= wdata[63:32];
            end
            9'd9:
            begin
                display_valid <= 1'b1;
                display_name  <= "WDATAL";
                display_value <= wdata[31:0];
            end
            default :
            begin
                display_valid <= 1'b0;
                display_name  <= 40'd0;
                display_value <= 32'd0;
            end
        endcase
    end
end
```

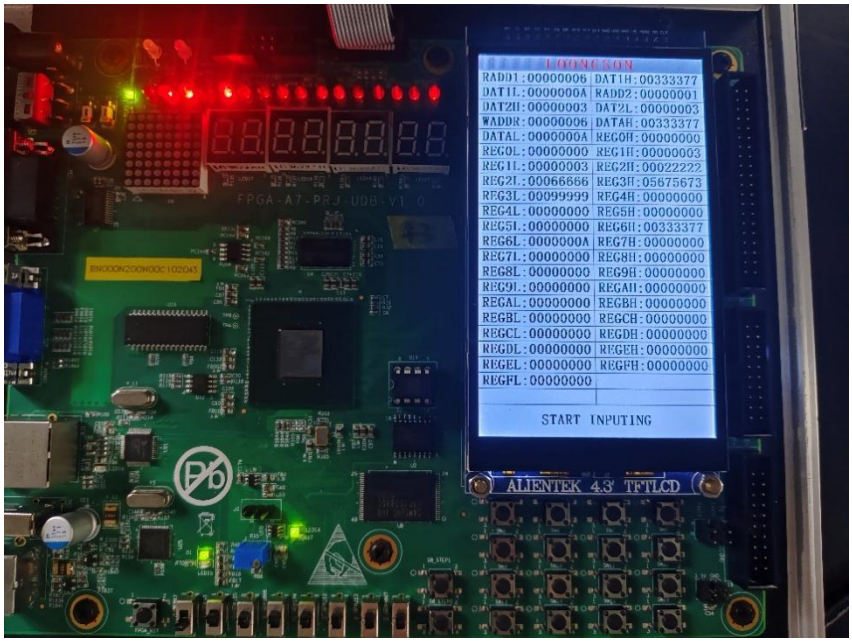
修改约束文件

```
set_property PACKAGE_PIN AC21 [get_ports wen]
set_property PACKAGE_PIN AD24 [get_ports input_sel[1]]
set_property PACKAGE_PIN AC22   [get_ports input_sel[0]]
set_property PACKAGE_PIN AC23   [get_ports hl]

set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports resetn]
set_property IOSTANDARD LVCMOS33 [get_ports led_wen]
set_property IOSTANDARD LVCMOS33 [get_ports led_raddr1]
set_property IOSTANDARD LVCMOS33 [get_ports led_raddr2]
set_property IOSTANDARD LVCMOS33 [get_ports led_waddr]
set_property IOSTANDARD LVCMOS33 [get_ports led_wdata]
set_property IOSTANDARD LVCMOS33 [get_ports wen]
set_property IOSTANDARD LVCMOS33 [get_ports input_sel[1]]
set_property IOSTANDARD LVCMOS33 [get_ports input_sel[0]]
set_property IOSTANDARD LVCMOS33 [get_ports hl]
```

五、实验结果分析

上实验箱后的结果



根据实验箱的结果，首先我向 6 号寄存器的高位写入了 0x33377，然后给其低位写入了 0xA。接着，我向 1 号寄存器的高位写入了 0x3，低位写入了 0x3。然后，我设置了 RADD1 为 6，RADD2 为 1，并查看了 6 号寄存器和 1 号寄存器中的值。通过观察，可以发现 DATA1H、DATA1L 以及 DATA2H、DATA2L 都显示了正确的值。

六、总结感想

通过本次改进寄存器堆的实验，我首先通过复现代码是了解到了寄存器堆的实现原理，是用 case 语句来控制每次取到的寄存器号和操作数，而在改进代码时遇到了好多的问题，比如要怎么保证显示屏显示的时候连续两个格子显示的是同一个寄存器的数据，在显示屏只能输入 32 位数据的情况下，怎么输入 64 位数据等等问题。

对第一个问题，我通过右移运算和取模运算巧妙的解决了，而对第二个问题，我就不得不新增加一个片选信号来控制现在输入的低位还是高位，总结来说，本次实验很大的提升了我对 verilog 语言的认识，也增强了我的 verilog 编程能力。