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。

2、 实验内容说明

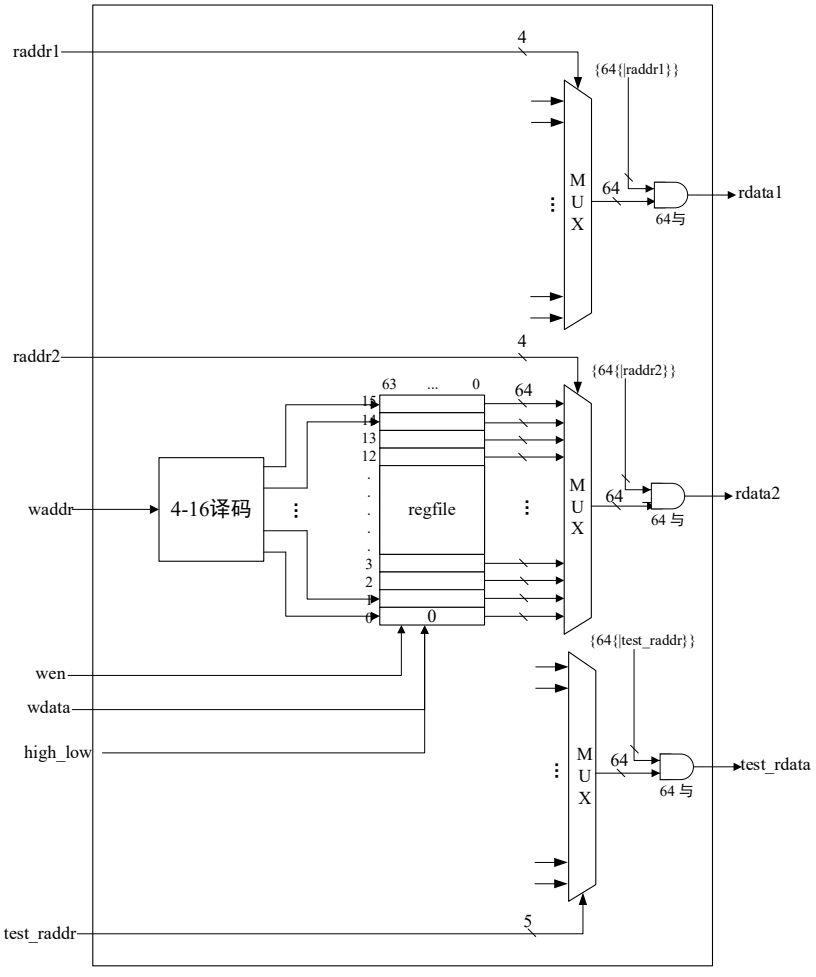
请结合实验指导手册中的实验三（寄存器堆实验）完成对寄存器堆进行 64 位位拓展的改进实验，注意以下几点：

（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。

3、 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     (dk     ),
                .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                (dk                 ),    //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

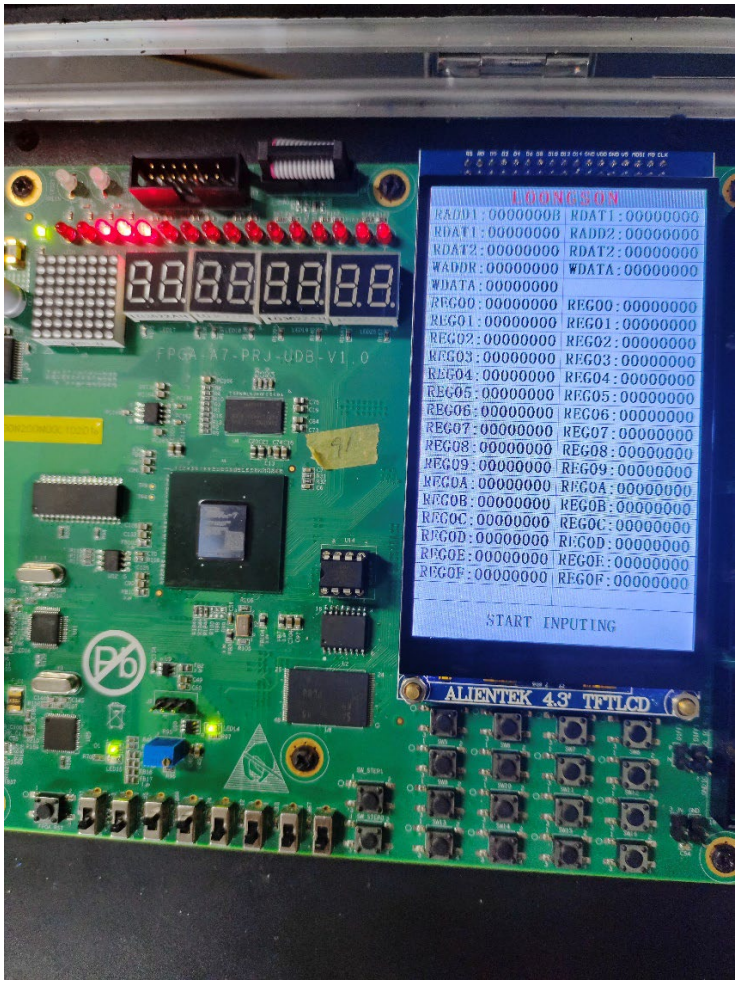
添加 high_low 拨码开关，用来控制高低位的切换。

```
#拨码开关连接，用于输入，依次为sw0, sw1, sw7
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 high_low]
set_property IOSTANDARD LVCMOS33 [get_ports high_low]
```

4、 实验结果分析

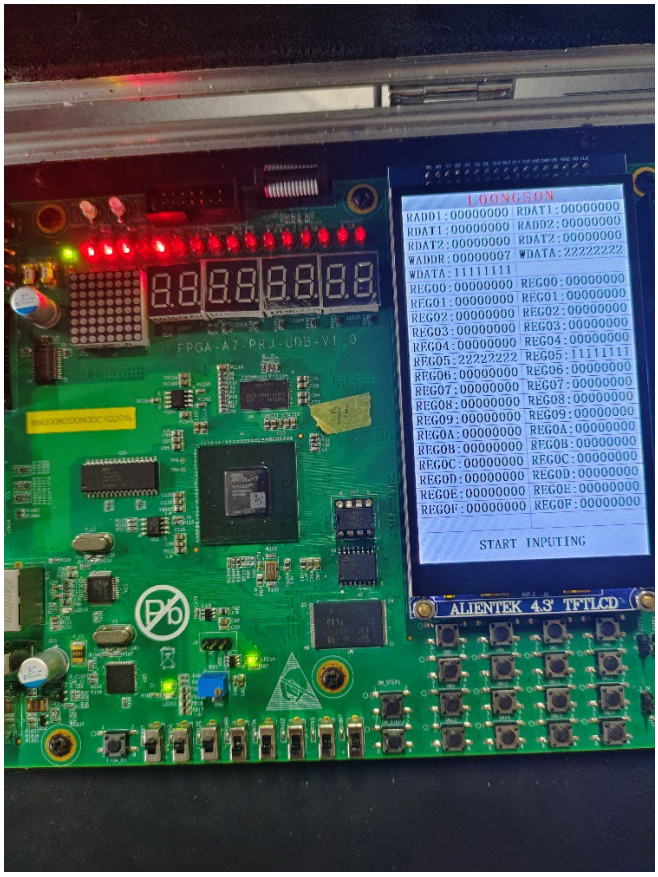
（1）上箱实验

a）原始:



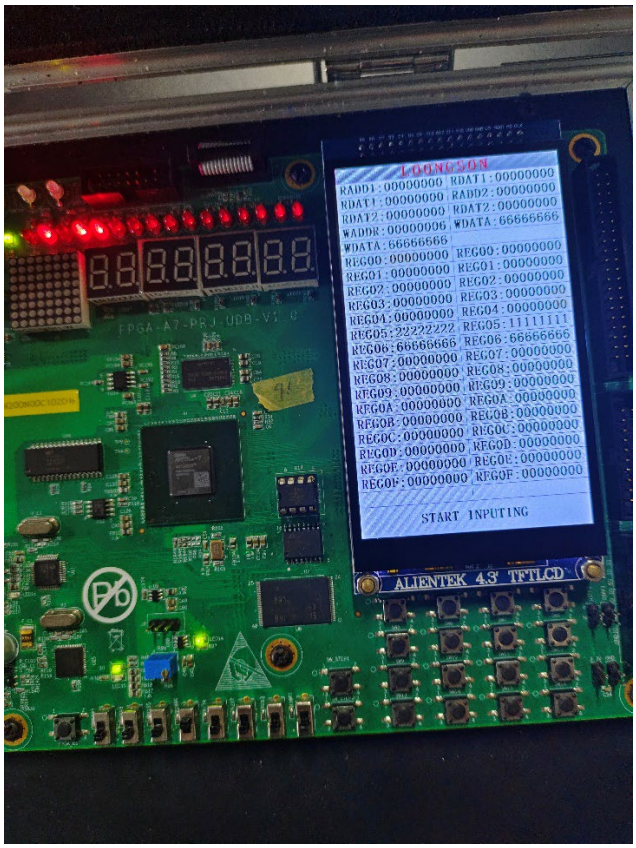
b）写入第一个寄存器:

向寄存器 5 写入 64 位数据 0x2222222211111111



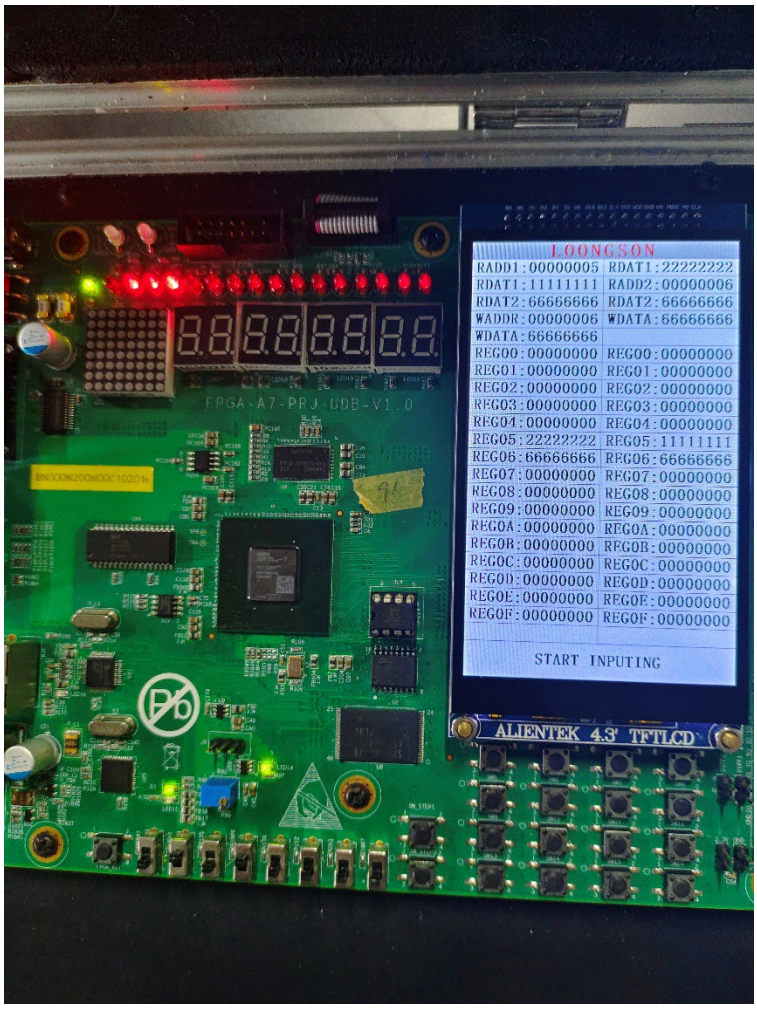
c）写入第二个寄存器:

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！

5、 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。