组成原理实验课程第_三_次实验报告

实验名称	寄存器堆的实现			班级	李涛
学生姓名	齐明杰	学号	2113997	指导老师	董前琨
实验地点	津南实验楼 A306		实验时间	2023.4.18	

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

i. 复现寄存器堆实验：

1. 学习 MIPS 计算机中寄存器堆的设计及原理，如：有多少个寄存器，有无特殊设置的寄存器，mips 指令如何去索引寄存器的等。

2. 自行设计本次实验的方案，画出结构框图，详细标出输入输出端口，本次实验建议设计为异步读同步写的寄存器堆，即读寄存器不需要时钟控制，但写寄存器需时钟控制。

3. 根据设计的实验方案，使用 verilog 编写相应代码。

4. 对编写的代码进行仿真，得到正确的波形图。

5. 将以上设计作为一个单独的模块，设计一个外围模块去调用该模块，外围模块中需调用封装好的 LCD 触摸屏模块，显示寄存器堆的读写端口地址和数据，最好能扫描出所有寄存器的值显示在 LCD 触摸屏上，并且需要利用触摸功能输入寄存器堆的读写地址和写数据。

ii. 对原实验的写操作进行改进：

将原有的寄存器堆的写操作进行改进，使用 4 位 wen 控制信号，对应写入 wdata 的四个字节，如：

0011: 写低 16 位，即后 2 个字节

0001: 写低 8 位，即最后 1 个字节

1100: 写高 16 位，即前 2 个字节

依此类推…

iii. 对原实验的读操作进行改进：

将原有的寄存器堆的读操作进行改进，使用 2 位 ren 控制信号，控制读出数据的高 16 位和低 16 位.注意寄存器堆的两个读端口同时控制。

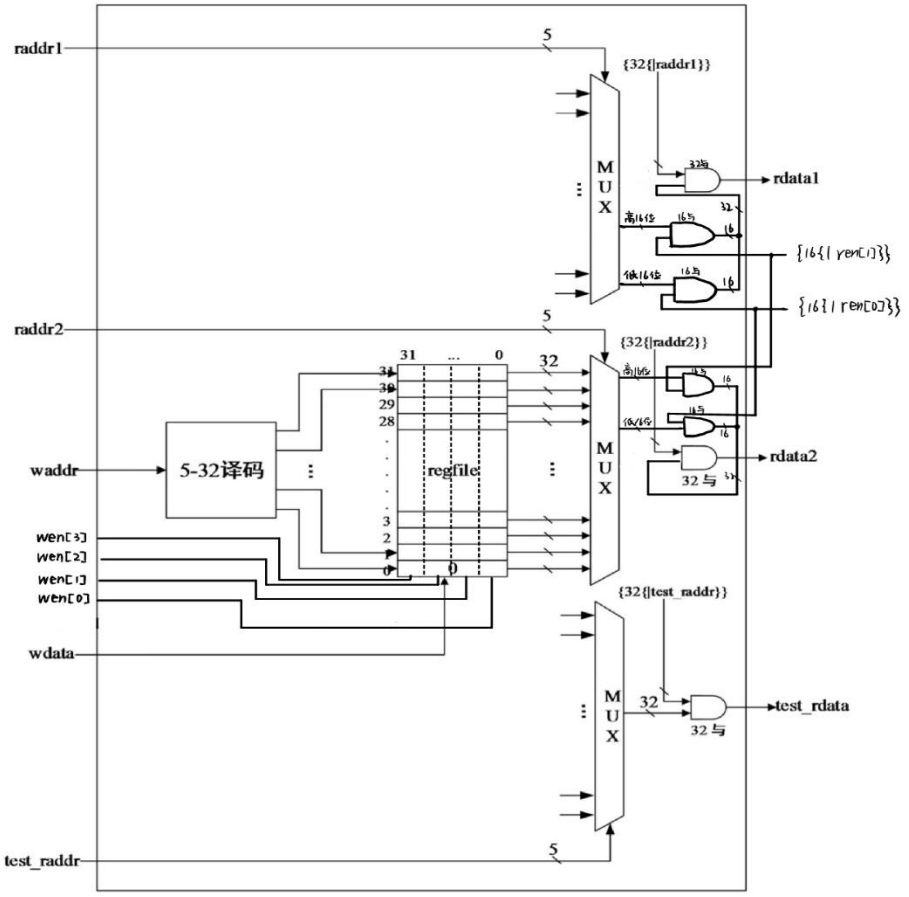
01：读低 16 位

10：读高 16 位

11：读 32 位，即整个数据

改进实验需要使用全部 8 个拨码开关。

3、 实验原理图



图中，写使能信号 wen[3:0]依次连接到寄存器堆的四个字节，分别对应各个字节的写入操作。读使能信号 ren[1]拓展为 16 位，和寄存器的高 16 位做与操作，ren[0]拓展为 16 位，和寄存器的低 16 位做与操作，二者拼接成 32 位的数，最终和拓展后的 raddr 做与操作(即若 raddr=0 则 rdata 必为 0)

4、 实验步骤

项目代码(修改处用红色标记):

(1) regfile.v

```
`timescale 1ns / 1ps
//*************************************************************************
//    > 文件名: regfile.v
//    > 描述   : 寄存器堆模块，同步写，异步读
//    > 作者   : LOONGSON
//    > 日期   : 2016-04-14
//*************************************************************************
```

```
module regfile(
    input           clk,
    input   [3 :0]  wen,//写使能，决定四个字节的输出与否
    input   [1 :0]  ren,//读使能，决定读高 16 位还是低 16 位还是全部 32 位
```

注：为了能控制四个字节的写入，我们把写使能分为 4 个值，wen[3],wen[2],wen[1],wen[0]分别对应数据从高到低四个字节，若 wen[i]为 1 则写入该字节到指定地址。同理为了分别读高低 16 位数据，我们将读使能分成 ren[1],ren[0]，分别对应高 16 位，低 16 位。

```
    input   [4 :0] raddr1,
    input   [4 :0] raddr2,
    input   [4 :0] waddr,
    input   [31:0] wdata,
    output reg [31:0] rdata1,
    output reg [31:0] rdata2,
    input   [4 :0] test_addr,
    output reg [31:0] test_data
    );
    reg [31:0] rf[31:0];

    // three ported register file
    // read two ports combinationally
    // write third port on rising edge of clock
    // register 0 hardwired to 0

    //最高字节
    always @(posedge clk)
    begin
        if (wen[3])
        begin
            rf[waddr][31:24] <= wdata[31:24];
        end
```

注：若 wen[3]为 1 则写入数据的第一个字节到指定寄存器对应字节中

```
    end
    //次高字节
    always @(posedge clk)
    begin
        if (wen[2])
        begin
            rf[waddr][23:16] <= wdata[23:16];
        end
```

注：若 wen[2]为 1 则写入数据的第二个字节到指定寄存器对应字节中

```
    end
    //次低字节
    always @(posedge clk)
    begin
```

```
            if (wen[1])
            begin
                rf[waddr][15:8] <= wdata[15:8];
            end
```
注：若 wen[1]为 1 则写入数据的第三个字节到指定寄存器对应字节中
```
        end
        //最低字节
        always @(posedge clk)
        begin
            if (wen[0])
            begin
                rf[waddr][7:0] <= wdata[7:0];
            end
```
注：若 wen[0]为 1 则写入数据的第四个字节到指定寄存器对应字节中
```
        end

        //读端口 1
        always @(*)
        begin
            case (raddr1)
                5'd1 : rdata1 <= {rf[1 ][31:16] & {16{ren[1]}},rf[1 ][15:0] &{16{ren[0]}}};
                5'd2 : rdata1 <= {rf[2 ][31:16] & {16{ren[1]}},rf[2 ][15:0] &{16{ren[0]}}};
                5'd3 : rdata1 <= {rf[3 ][31:16] & {16{ren[1]}},rf[3 ][15:0] &{16{ren[0]}}};
                5'd4 : rdata1 <= {rf[4 ][31:16] & {16{ren[1]}},rf[4 ][15:0] &{16{ren[0]}}};
                5'd5 : rdata1 <= {rf[5 ][31:16] & {16{ren[1]}},rf[5 ][15:0] &{16{ren[0]}}};
                5'd6 : rdata1 <= {rf[6 ][31:16] & {16{ren[1]}},rf[6 ][15:0] &{16{ren[0]}}};
                5'd7 : rdata1 <= {rf[7 ][31:16] & {16{ren[1]}},rf[7 ][15:0] &{16{ren[0]}}};
                5'd8 : rdata1 <= {rf[8 ][31:16] & {16{ren[1]}},rf[8 ][15:0] &{16{ren[0]}}};
                5'd9 : rdata1 <= {rf[9 ][31:16] & {16{ren[1]}},rf[9 ][15:0] &{16{ren[0]}}};
                5'd10: rdata1 <= {rf[10 ][31:16] & {16{ren[1]}},rf[10 ][15:0] &{16{ren[0]}}};
                5'd11: rdata1 <= {rf[11 ][31:16] & {16{ren[1]}},rf[11 ][15:0] &{16{ren[0]}}};
                5'd12: rdata1 <= {rf[12 ][31:16] & {16{ren[1]}},rf[12 ][15:0] &{16{ren[0]}}};
                5'd13: rdata1 <= {rf[13 ][31:16] & {16{ren[1]}},rf[13 ][15:0] &{16{ren[0]}}};
                5'd14: rdata1 <= {rf[14 ][31:16] & {16{ren[1]}},rf[14 ][15:0] &{16{ren[0]}}};
                5'd15: rdata1 <= {rf[15 ][31:16] & {16{ren[1]}},rf[15 ][15:0] &{16{ren[0]}}};
                5'd16: rdata1 <= {rf[16 ][31:16] & {16{ren[1]}},rf[16 ][15:0] &{16{ren[0]}}};
                5'd17: rdata1 <= {rf[17 ][31:16] & {16{ren[1]}},rf[17 ][15:0] &{16{ren[0]}}};
                5'd18: rdata1 <= {rf[18 ][31:16] & {16{ren[1]}},rf[18 ][15:0] &{16{ren[0]}}};
                5'd19: rdata1 <= {rf[19 ][31:16] & {16{ren[1]}},rf[19 ][15:0] &{16{ren[0]}}};
                5'd20: rdata1 <= {rf[20 ][31:16] & {16{ren[1]}},rf[20 ][15:0] &{16{ren[0]}}};
                5'd21: rdata1 <= {rf[21 ][31:16] & {16{ren[1]}},rf[21 ][15:0] &{16{ren[0]}}};
                5'd22: rdata1 <= {rf[22 ][31:16] & {16{ren[1]}},rf[22 ][15:0] &{16{ren[0]}}};
                5'd23: rdata1 <= {rf[23 ][31:16] & {16{ren[1]}},rf[23 ][15:0] &{16{ren[0]}}};
                5'd24: rdata1 <= {rf[24 ][31:16] & {16{ren[1]}},rf[24 ][15:0] &{16{ren[0]}}};
```

```
            5'd25: rdata1 <= {rf[25 ][31:16] & {16{ren[1]}},rf[25 ][15:0] &{16{ren[0]}}};
            5'd26: rdata1 <= {rf[26 ][31:16] & {16{ren[1]}},rf[26 ][15:0] &{16{ren[0]}}};
            5'd27: rdata1 <= {rf[27 ][31:16] & {16{ren[1]}},rf[27 ][15:0] &{16{ren[0]}}};
            5'd28: rdata1 <= {rf[28 ][31:16] & {16{ren[1]}},rf[28 ][15:0] &{16{ren[0]}}};
            5'd29: rdata1 <= {rf[29 ][31:16] & {16{ren[1]}},rf[29 ][15:0] &{16{ren[0]}}};
            5'd30: rdata1 <= {rf[30 ][31:16] & {16{ren[1]}},rf[30 ][15:0] &{16{ren[0]}}};
            5'd31: rdata1 <= {rf[31 ][31:16] & {16{ren[1]}},rf[31 ][15:0] &{16{ren[0]}}};
```

注：若 ren[1]为 1，则使 16 位 1 和待读寄存器的高 16 位做“与”操作，即获取高 16 位的值，若 ren[1]为 0 则获取 16 位 0；

若 ren[0]为 1，则使 16 位 1 和待读寄存器的低 16 位做“与”操作，即获取低 16 位的值；若 ren[0]为 0 则获取 16 位 0；

再将二者拼接，即可达到任意读取高 16 位，低 16 位或全读取的目的。

```
            default : rdata1 <= 32'd0;
        endcase
    end
    //读端口 2
    always @(*)
    begin
        case (raddr2)
            5'd1 : rdata2 <= {rf[1 ][31:16] & {16{ren[1]}},rf[1 ][15:0] &{16{ren[0]}}};
            5'd2 : rdata2 <= {rf[2 ][31:16] & {16{ren[1]}},rf[2 ][15:0] &{16{ren[0]}}};
            5'd3 : rdata2 <= {rf[3 ][31:16] & {16{ren[1]}},rf[3 ][15:0] &{16{ren[0]}}};
            5'd4 : rdata2 <= {rf[4 ][31:16] & {16{ren[1]}},rf[4 ][15:0] &{16{ren[0]}}};
            5'd5 : rdata2 <= {rf[5 ][31:16] & {16{ren[1]}},rf[5 ][15:0] &{16{ren[0]}}};
            5'd6 : rdata2 <= {rf[6 ][31:16] & {16{ren[1]}},rf[6 ][15:0] &{16{ren[0]}}};
            5'd7 : rdata2 <= {rf[7 ][31:16] & {16{ren[1]}},rf[7 ][15:0] &{16{ren[0]}}};
            5'd8 : rdata2 <= {rf[8 ][31:16] & {16{ren[1]}},rf[8 ][15:0] &{16{ren[0]}}};
            5'd9 : rdata2 <= {rf[9 ][31:16] & {16{ren[1]}},rf[9 ][15:0] &{16{ren[0]}}};
            5'd10: rdata2 <= {rf[10 ][31:16] & {16{ren[1]}},rf[10 ][15:0] &{16{ren[0]}}};
            5'd11: rdata2 <= {rf[11 ][31:16] & {16{ren[1]}},rf[11 ][15:0] &{16{ren[0]}}};
            5'd12: rdata2 <= {rf[12 ][31:16] & {16{ren[1]}},rf[12 ][15:0] &{16{ren[0]}}};
            5'd13: rdata2 <= {rf[13 ][31:16] & {16{ren[1]}},rf[13 ][15:0] &{16{ren[0]}}};
            5'd14: rdata2 <= {rf[14 ][31:16] & {16{ren[1]}},rf[14 ][15:0] &{16{ren[0]}}};
            5'd15: rdata2 <= {rf[15 ][31:16] & {16{ren[1]}},rf[15 ][15:0] &{16{ren[0]}}};
            5'd16: rdata2 <= {rf[16 ][31:16] & {16{ren[1]}},rf[16 ][15:0] &{16{ren[0]}}};
            5'd17: rdata2 <= {rf[17 ][31:16] & {16{ren[1]}},rf[17 ][15:0] &{16{ren[0]}}};
            5'd18: rdata2 <= {rf[18 ][31:16] & {16{ren[1]}},rf[18 ][15:0] &{16{ren[0]}}};
            5'd19: rdata2 <= {rf[19 ][31:16] & {16{ren[1]}},rf[19 ][15:0] &{16{ren[0]}}};
            5'd20: rdata2 <= {rf[20 ][31:16] & {16{ren[1]}},rf[20 ][15:0] &{16{ren[0]}}};
            5'd21: rdata2 <= {rf[21 ][31:16] & {16{ren[1]}},rf[21 ][15:0] &{16{ren[0]}}};
            5'd22: rdata2 <= {rf[22 ][31:16] & {16{ren[1]}},rf[22 ][15:0] &{16{ren[0]}}};
            5'd23: rdata2 <= {rf[23 ][31:16] & {16{ren[1]}},rf[23 ][15:0] &{16{ren[0]}}};
            5'd24: rdata2 <= {rf[24 ][31:16] & {16{ren[1]}},rf[24 ][15:0] &{16{ren[0]}}};
            5'd25: rdata2 <= {rf[25 ][31:16] & {16{ren[1]}},rf[25 ][15:0] &{16{ren[0]}}};
```

```
            5'd26: rdata2 <= {rf[26 ][31:16] & {16{ren[1]}},rf[26 ][15:0] &{16{ren[0]}}};
            5'd27: rdata2 <= {rf[27 ][31:16] & {16{ren[1]}},rf[27 ][15:0] &{16{ren[0]}}};
            5'd28: rdata2 <= {rf[28 ][31:16] & {16{ren[1]}},rf[28 ][15:0] &{16{ren[0]}}};
            5'd29: rdata2 <= {rf[29 ][31:16] & {16{ren[1]}},rf[29 ][15:0] &{16{ren[0]}}};
            5'd30: rdata2 <= {rf[30 ][31:16] & {16{ren[1]}},rf[30 ][15:0] &{16{ren[0]}}};
            5'd31: rdata2 <= {rf[31 ][31:16] & {16{ren[1]}},rf[31 ][15:0] &{16{ren[0]}}};
```

注：与上面同理

```
            default : rdata1 <= 32'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];
            5'd16: test_data <= rf[16];
            5'd17: test_data <= rf[17];
            5'd18: test_data <= rf[18];
            5'd19: test_data <= rf[19];
            5'd20: test_data <= rf[20];
            5'd21: test_data <= rf[21];
            5'd22: test_data <= rf[22];
            5'd23: test_data <= rf[23];
            5'd24: test_data <= rf[24];
            5'd25: test_data <= rf[25];
            5'd26: test_data <= rf[26];
            5'd27: test_data <= rf[27];
            5'd28: test_data <= rf[28];
            5'd29: test_data <= rf[29];
            5'd30: test_data <= rf[30];
```

```
                5'd31: test_data <= rf[31];
                default : test_data <= 32'd0;
            endcase
        end
    endmodule
```

(2) regfile_display.v

```
    //************************************************************************
    //    > 文件名: regfile_display.v
    //    > 描述   : 寄存器堆显示模块，调用 FPGA 板上的 IO 接口和触摸屏
    //    > 作者   : LOONGSON
    //    > 日期   : 2016-04-14
    //************************************************************************
    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 [3:0]wen,
        input [1:0]ren,
```

注：四个写使能拨码开关和两个读使能拨码开关

```
        input [1:0] input_sel,

        //led 灯，用于指示写使能信号，和正在输入什么数据
        output led_wen0,
        output led_wen1,
        output led_wen2,
        output led_wen3,
        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
        output led_ren0,
        output led_ren1,
```

注：修改 led 灯与写使能，读使能有关的输出信号表示

```
        //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_wen0     = wen[0];
        assign led_wen1     = wen[1];
        assign led_wen2     = wen[2];
        assign led_wen3     = wen[3];
        assign led_ren0     = ren[0];
        assign led_ren1     = ren[1];
```

注：将 led 修改后的输出信号与变量绑定

```
        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 32 个寄存器值
        wire [31:0] test_data;
        wire [4 :0] test_addr;

        reg   [4 :0] raddr1;
        reg   [4 :0] raddr2;
        reg   [4 :0] waddr;
        reg   [31:0] wdata;
        wire [31:0] rdata1;
        wire [31:0] rdata2;
        regfile rf_module(
            .clk    (clk    ),
            .wen    (wen    ),
            .ren    (ren    ),
            .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
    //32 个寄存器显示在 7~38 号的显示块，故读地址为（display_number-1）
    assign test_addr = display_number-5'd7;
    //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 <= 5'd0;
    end
    else if (input_valid &&   input_sel==2'd0)
    begin
        raddr1 <= input_value[4:0];
    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 <= 5'd0;
    end
    else if (input_valid && input_sel==2'd1)
    begin
        raddr2 <= input_value[4:0];
    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   <= 5'd0;
    end
    else if (input_valid && input_sel==2'd2)
    begin
        waddr   <= input_value[4:0];
    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   <= 32'd0;
```

```
            end
            else if (input_valid && input_sel==2'd3)
            begin
                wdata   <= 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'd6 && display_number <6'd39 )
            begin //块号 7~38 显示 32 个通用寄存器的值
                display_valid <= 1'b1;
                display_name[39:16] <= "REG";
                display_name[15: 8] <= {4'b0011,3'b000,test_addr[4]};
                display_name[7 : 0] <= {4'b0011,test_addr[3:0]};
                display_value           <= test_data;
              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;
                    end
                    6'd3 : //显示读端口 2 的地址
                    begin
                        display_valid <= 1'b1;
                        display_name   <= "RADD2";
                        display_value <= raddr2;
                    end
                    6'd4 : //显示读端口 2 读出的数据
```

```
                    begin
                        display_valid <= 1'b1;
                        display_name   <= "RDAT2";
                        display_value <= rdata2;
                    end
                    6'd5 : //显示写端口的地址
                    begin
                        display_valid <= 1'b1;
                        display_name   <= "WADDR";
                        display_value <= waddr;
                    end
                    6'd6 : //显示写端口写入的数据
                    begin
                        display_valid <= 1'b1;
                        display_name   <= "WDATA";
                        display_value <= wdata;
                    end
                    default :
                    begin
                        display_valid <= 1'b0;
                        display_name   <= 40'd0;
                        display_value <= 32'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

```
#时钟信号连接
set_property PACKAGE_PIN AC19 [get_ports clk]

#脉冲开关，用于输入作为复位信号，低电平有效
set_property PACKAGE_PIN Y3 [get_ports resetn]

#led 灯连接，用于输出
set_property PACKAGE_PIN H7 [get_ports led_wen0]
set_property PACKAGE_PIN F7 [get_ports led_wen1]
set_property PACKAGE_PIN G8 [get_ports led_wen2]
set_property PACKAGE_PIN H8 [get_ports led_wen3]
set_property PACKAGE_PIN J8 [get_ports led_ren0]
set_property PACKAGE_PIN J23 [get_ports led_ren1]
```

注：修改与写使能、读使能有关的 led 接口

```
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]

#拨码开关连接，用于输入，依次为 sw0,sw1,sw7
set_property PACKAGE_PIN AC21 [get_ports input_sel[1]]
set_property PACKAGE_PIN AD24 [get_ports input_sel[0]]
set_property PACKAGE_PIN AC22 [get_ports wen[3]]
set_property PACKAGE_PIN AC23 [get_ports wen[2]]
set_property PACKAGE_PIN AB6  [get_ports wen[1]]
set_property PACKAGE_PIN W6   [get_ports wen[0]]
set_property PACKAGE_PIN AA7  [get_ports ren[1]]
set_property PACKAGE_PIN Y6   [get_ports ren[0]]
```

注：修改拨码开关接口，使用了全部 8 个拨码开关，从左到右：选择输入数*2，写使能*4，读使能*2

```
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports resetn]
set_property IOSTANDARD LVCMOS33 [get_ports led_wen0]
set_property IOSTANDARD LVCMOS33 [get_ports led_wen1]
set_property IOSTANDARD LVCMOS33 [get_ports led_wen2]
set_property IOSTANDARD LVCMOS33 [get_ports led_wen3]
set_property IOSTANDARD LVCMOS33 [get_ports led_ren0]
set_property IOSTANDARD LVCMOS33 [get_ports led_ren1]
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[0]]
set_property IOSTANDARD LVCMOS33 [get_ports wen[1]]
set_property IOSTANDARD LVCMOS33 [get_ports wen[2]]
set_property IOSTANDARD LVCMOS33 [get_ports wen[3]]
set_property IOSTANDARD LVCMOS33 [get_ports ren[0]]
set_property IOSTANDARD LVCMOS33 [get_ports ren[1]]
set_property IOSTANDARD LVCMOS33 [get_ports input_sel[1]]
set_property IOSTANDARD LVCMOS33 [get_ports input_sel[0]]

#触摸屏引脚连接
set_property PACKAGE_PIN J25 [get_ports lcd_rst]
set_property PACKAGE_PIN H18 [get_ports lcd_cs]
set_property PACKAGE_PIN K16 [get_ports lcd_rs]
set_property PACKAGE_PIN L8 [get_ports lcd_wr]
set_property PACKAGE_PIN K8 [get_ports lcd_rd]
set_property PACKAGE_PIN J15 [get_ports lcd_bl_ctr]
```

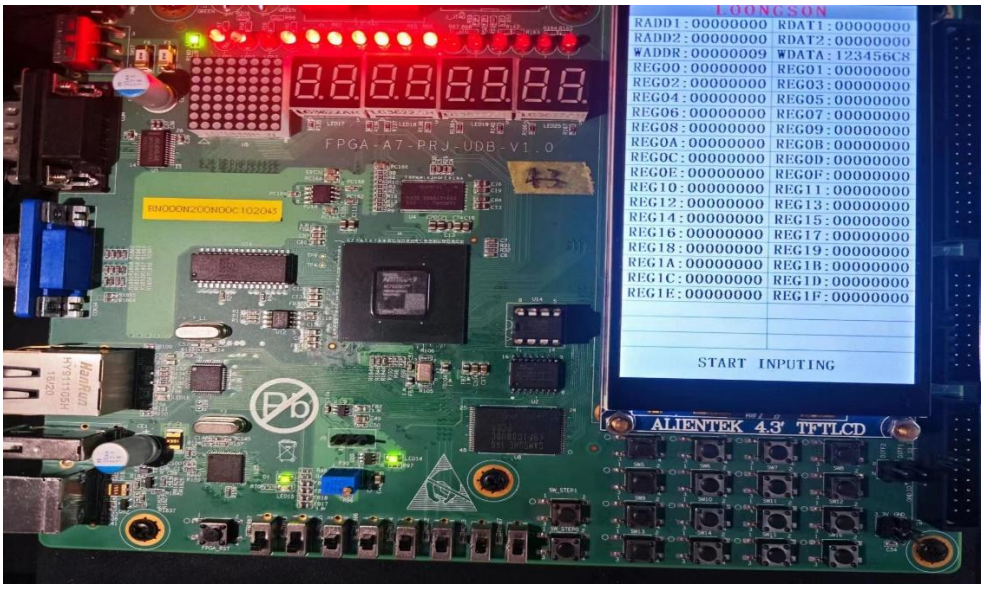
```
set_property PACKAGE_PIN H9 [get_ports {lcd_data_io[0]}]
set_property PACKAGE_PIN K17 [get_ports {lcd_data_io[1]}]
set_property PACKAGE_PIN J20 [get_ports {lcd_data_io[2]}]
set_property PACKAGE_PIN M17 [get_ports {lcd_data_io[3]}]
set_property PACKAGE_PIN L17 [get_ports {lcd_data_io[4]}]
set_property PACKAGE_PIN L18 [get_ports {lcd_data_io[5]}]
set_property PACKAGE_PIN L15 [get_ports {lcd_data_io[6]}]
set_property PACKAGE_PIN M15 [get_ports {lcd_data_io[7]}]
set_property PACKAGE_PIN M16 [get_ports {lcd_data_io[8]}]
set_property PACKAGE_PIN L14 [get_ports {lcd_data_io[9]}]
set_property PACKAGE_PIN M14 [get_ports {lcd_data_io[10]}]
set_property PACKAGE_PIN F22 [get_ports {lcd_data_io[11]}]
set_property PACKAGE_PIN G22 [get_ports {lcd_data_io[12]}]
set_property PACKAGE_PIN G21 [get_ports {lcd_data_io[13]}]
set_property PACKAGE_PIN H24 [get_ports {lcd_data_io[14]}]
set_property PACKAGE_PIN J16 [get_ports {lcd_data_io[15]}]
set_property PACKAGE_PIN L19 [get_ports ct_int]
set_property PACKAGE_PIN J24 [get_ports ct_sda]
set_property PACKAGE_PIN H21 [get_ports ct_scl]
set_property PACKAGE_PIN G24 [get_ports ct_rstn]

set_property IOSTANDARD LVCMOS33 [get_ports lcd_rst]
set_property IOSTANDARD LVCMOS33 [get_ports lcd_cs]
set_property IOSTANDARD LVCMOS33 [get_ports lcd_rs]
set_property IOSTANDARD LVCMOS33 [get_ports lcd_wr]
set_property IOSTANDARD LVCMOS33 [get_ports lcd_rd]
set_property IOSTANDARD LVCMOS33 [get_ports lcd_bl_ctr]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports ct_int]
```

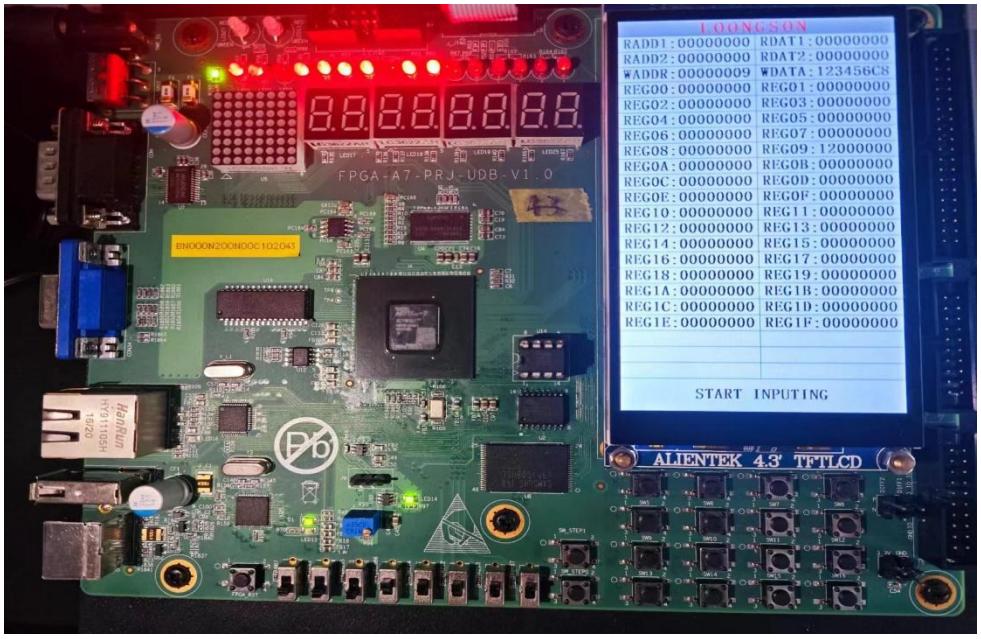
```
set_property IOSTANDARD LVCMOS33 [get_ports ct_sda]
set_property IOSTANDARD LVCMOS33 [get_ports ct_scl]
set_property IOSTANDARD LVCMOS33 [get_ports ct_rstn]
```

5、 实验结果分析

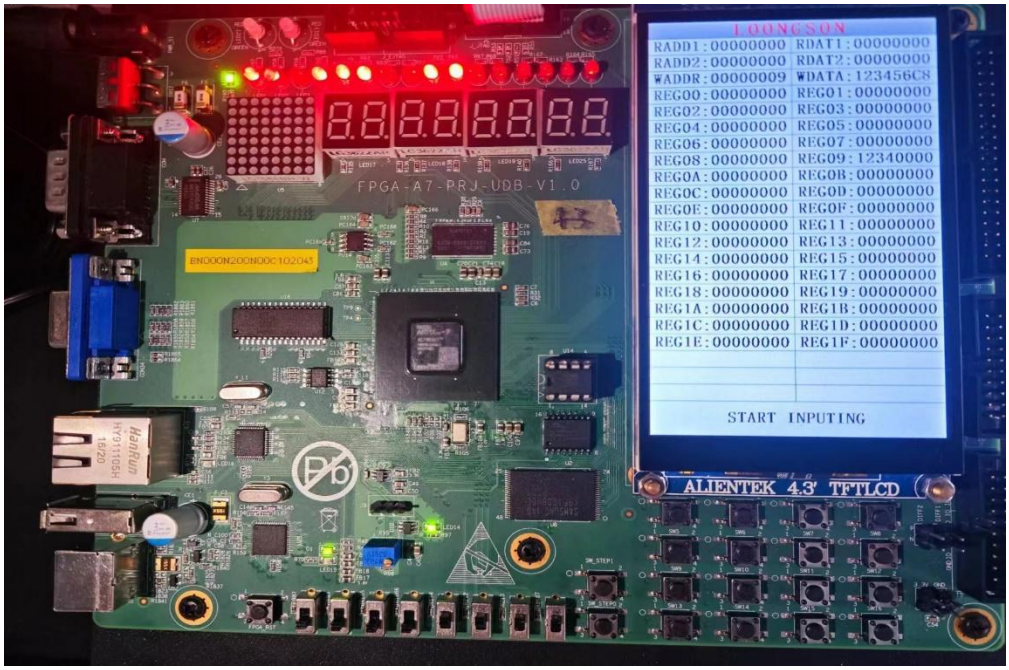
总述：8 个拨码开关从左到右依次控制：input_sel[1], input_sel[0], wen[3], wen[2], wen[1], wen[0], ren[1], ren[0]，据此完成如下各图步骤：



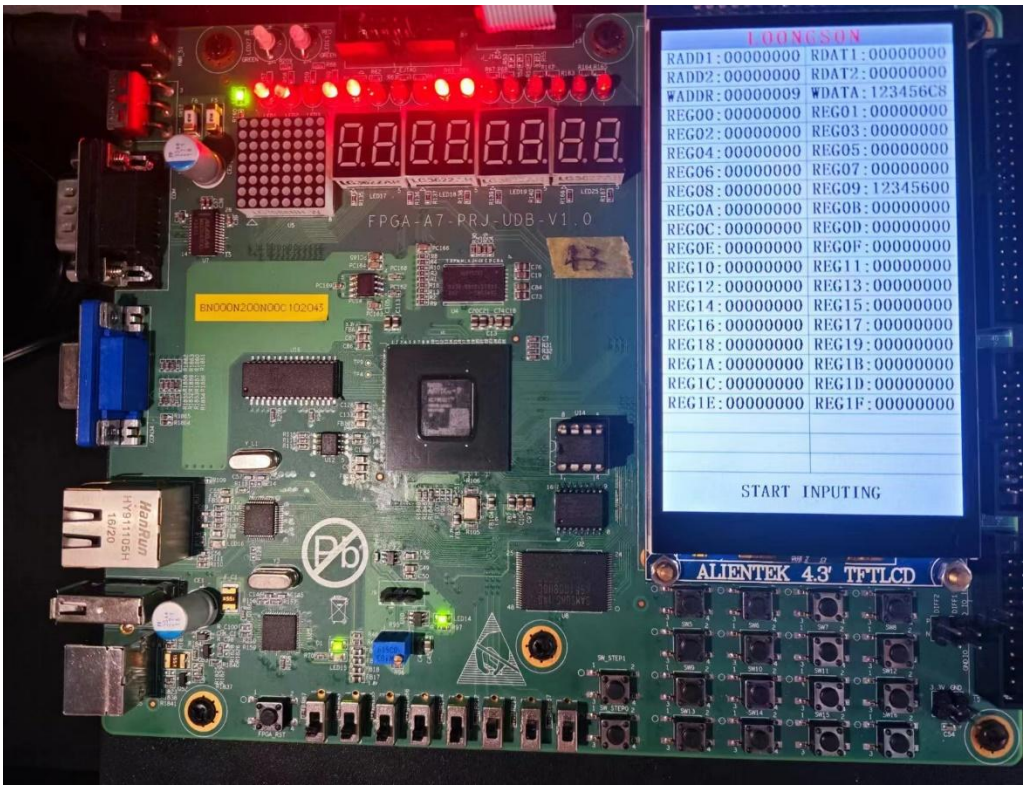
分析：首先测试写使能开关，分别写入 WADDR = 0x9，WDATA = 0x123456C8，但把四个写使能 wen[3:0]均拨为 0，此时 REG09 保持 0，无写入。



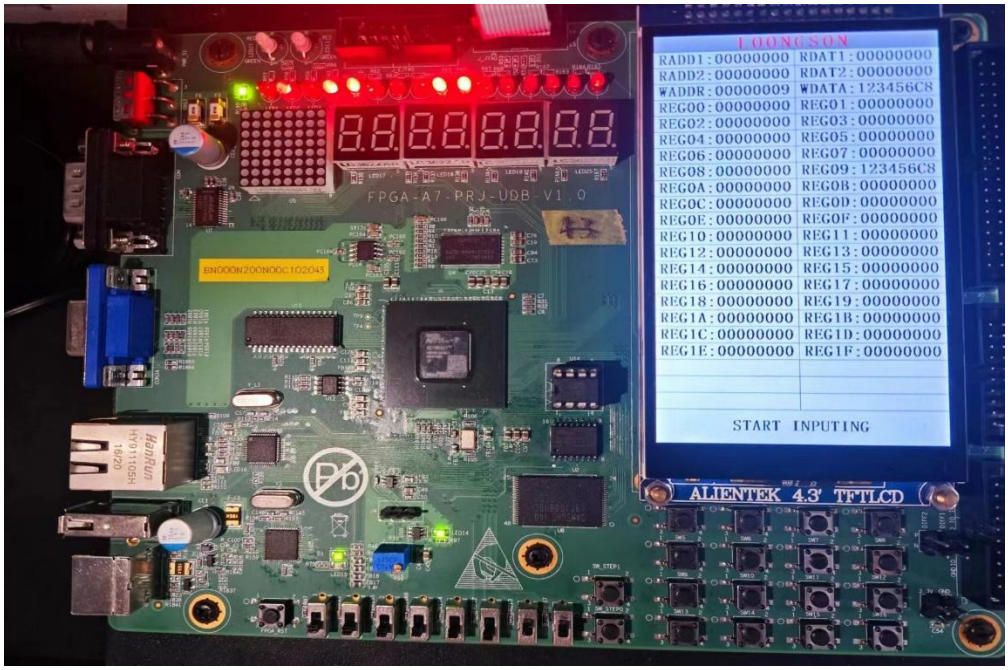
分析：将 wen[3]拨为 1，此时写入第一个字节，REG09 = 0x12000000



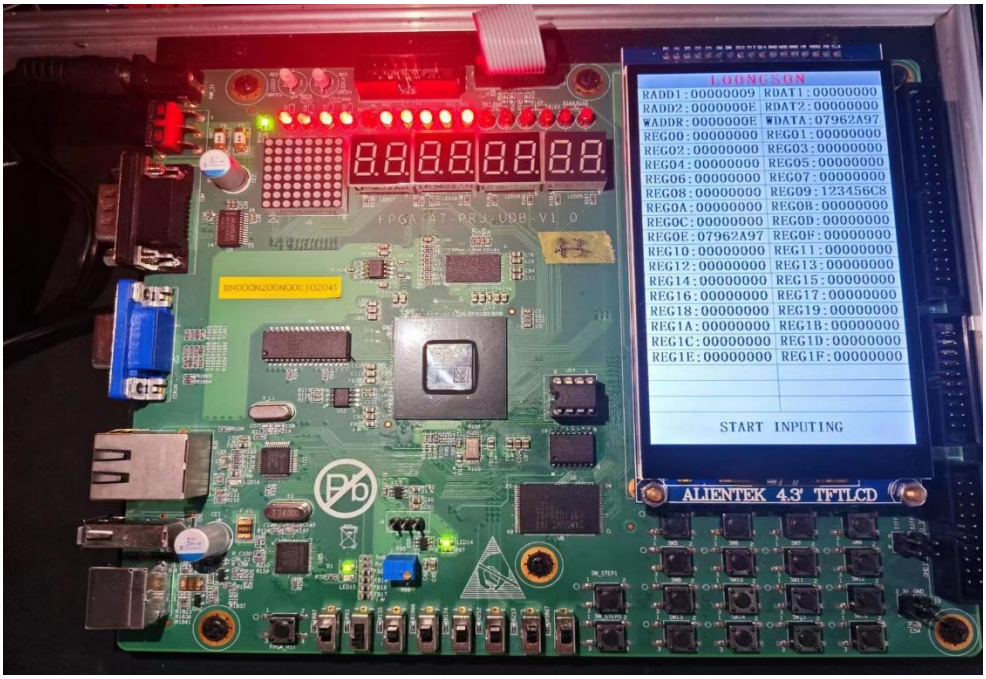
分析：将 wen[2]拨为 1，此时写入第一、二个字节，REG09 = 0x12340000



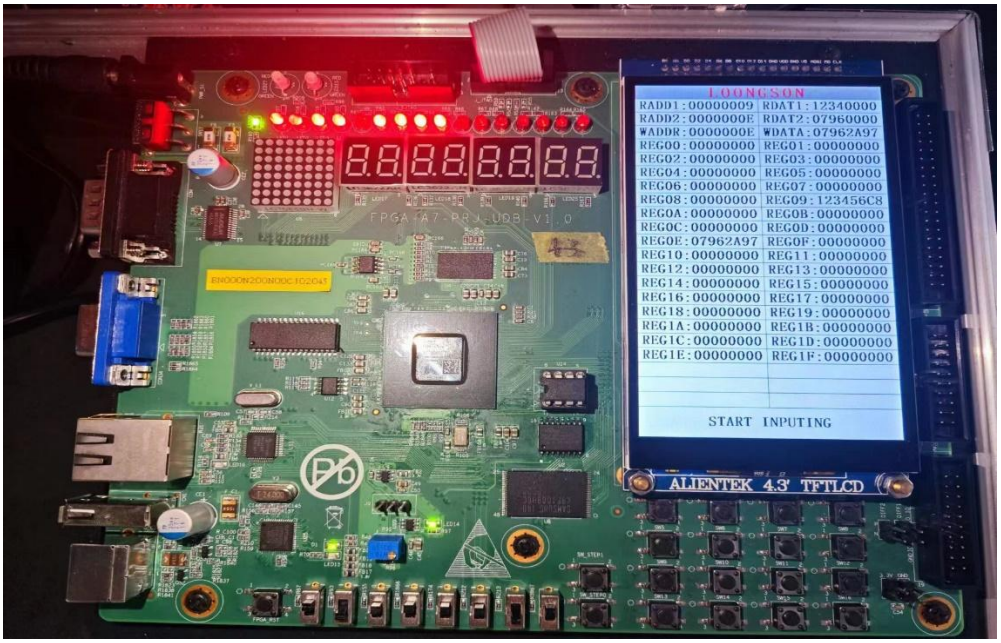
分析：将 wen[1]拨为 1，此时写入第一、二、三个字节，REG09 = 0x12345600



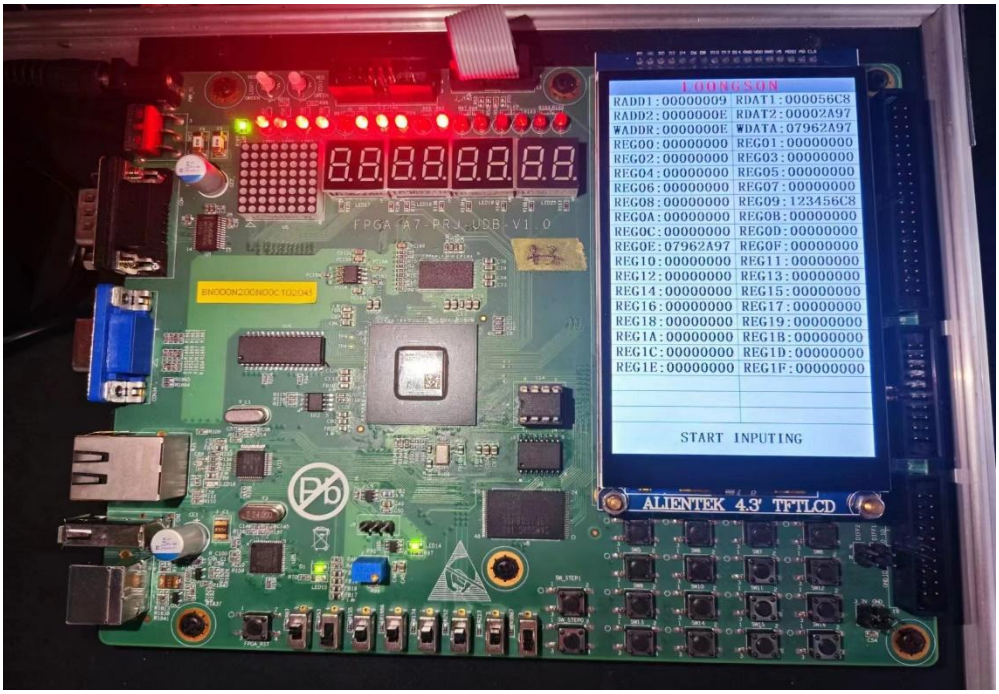
分析：将 wen[0]拨为 1，此时写入第一、二、三、四个字节，REG09 = 0x123456C8



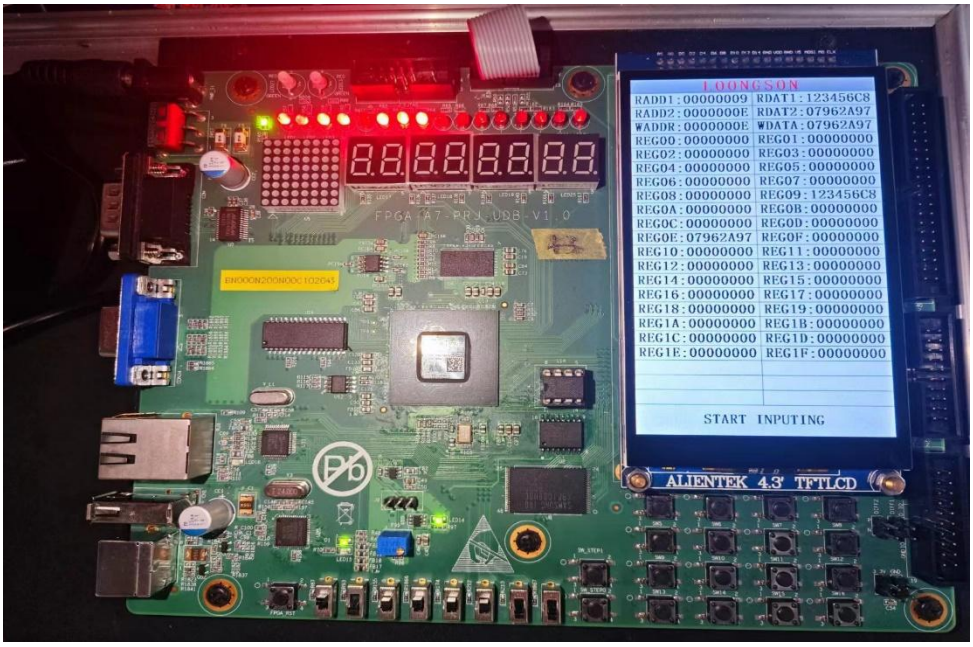
分析：接下来测试读使能开关，分别输入 RADD1 = 0x9, RADD2 = 0xE，两个读使能拨码均拨为 0，此时未读取数据，RDAT1 = RDAT2 = 0



分析：切换读使能拨码开关使 ren[1] = 1，ren[0] = 0，此时 RDAT1 = 0x12340000, RDAT2 = 0x07960000，均为高 16 位数据。



分析：切换读使能拨码开关使 ren[1] = 1，ren[0] = 0，此时 RDAT1 = 0x56C8，RDAT2 = 0x2A97，均为低 16 位数据。



分析：切换读使能拨码开关使 ren[1] = 1，ren[0] = 1，此时 RDAT1 = 0x123456C8, RDAT2 = 0x07962A97，均为 32 位数据。

6、 总结感想

通过本次实验，我理解了寄存器堆的实现原理，掌握了通过拨码开关来控制对寄存器堆的读写。同时，由于实验要求利用全部 8 个拨码开关，我必须对 xdc 文件进行修改，因此学习了有关部分的改写方法。另外，四个字节的选择写入也需要一定的技巧，我利用与操作和拼接最终完成了相关内容。

经过本次寄存器堆的实验，我也对上箱实验有了进一步的认知，对实验箱的结构和组成有了更深入的了解，更加熟练对于 verilog 语句的编写、改写。