**组成原理实验课程第 三 次实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | 寄存器堆实验 | | | 班级 | 李涛老师 |
| 学生姓名 | 张洋 | 学号 | 2111460 | 指导老师 | 董前琨老师 |
| 实验地点 | 津南实验楼A306 | | 实验时间 | 4月18日19：00-20：30 | |

1. **实验目的**
2. 熟悉并掌握MIPS计算机中寄存器堆的原理和设计方法。
3. 初步了解MIPS指令结构和源操作数/目的操作数的概念。
4. 熟悉并运用verilog语言进行电路设计。
5. 为后续设计cpu的实验打下基础。
6. **实验内容说明**
7. 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。
8. 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。
9. 本次实验建议寄存器堆设计为1个写端口和2个读端口，后续CPU实验用到的寄存器堆需要1个写端口和2个读端口。
10. 根据设计的实验方案，使用verilog编写相应代码。
11. 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，见图1。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。

寄存器堆模块

外围模块

来自FPGA板子上的输入

输出到FPGA板上进行展示

图1 寄存器堆设计实验的顶层模块大致框图

1. 将编写的代码进行综合布局布线，并下载到实验箱中的FPGA板子上进行演示。
2. 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，比如wen为“1011”时，写入第4、2、1三个字节。
3. 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。
4. **实验原理图**



1. **实验步骤**

1.理解寄存器堆的原理图。

2.编写verilog代码实现要求的功能。

3.完成调用寄存器堆模块的外围模块的设计，并编写代码；

4.对代码进行综合布局布线下载到实验箱里FPGA板上，进行上板验证。

**改进代码（改进处标红）**

**regfile.v**

module regfile(

input clk,

input [3 :0] wen, //增加4位wen控制信号

input [1 :0] ren, //增加2位ren控制信号

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

//4位wen控制信号，对应写入wdata的四个字节，比如wen为“1011”时，写入第4、2、1三个字节。

always @(posedge clk)

begin

if (wen[0]) //判断wdata[7:0]是否写入

begin

rf[waddr][7:0] <= wdata[7:0];

end

if (wen[1]) //判断wadta[15:8]是否写入

begin

rf[waddr][15:8] <= wdata[15:8];

end

if (wen[2]) //判断wdata[23:16]是否写入

begin

rf[waddr][23:16] <= wdata[23:16];

end

if (wen[3]) //判断wdata[31:24]是否写入

begin

rf[waddr][31:24] <= wdata[31:24];

end

end

//读操作，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。

always @ (posedge clk)

begin

if (ren == 2'b00) begin //ren等于00时表示不读取寄存器中的值采用，采用阻塞赋值方式，不会读出当前寄存器中的值。

rdata1 = rf[raddr1];

rdata2 = rf[raddr2];

end

else if (ren == 2'b01) begin //ren等于01时表示读取寄存器中的低十六位。

rdata1[15:0] <= rf[raddr1][15:0];

rdata2[15:0] <= rf[raddr2][15:0];

end

else if (ren == 2'b10) begin //ren等于10时表示读取寄存器中的高十六位。

rdata1[31:16] <= rf[raddr1][31:16];

rdata2[31:16] <= rf[raddr2][31:16];

end

else if (ren == 2'b11) begin ren等于11时表示读取寄存器中的全部32数，采用非阻塞赋值方式，可以读出当前寄存器中的值。

rdata1 <= rf[raddr1];

rdata2 <= rf[raddr2];

end

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;

**regfile\_display.v**

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,

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\_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

**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\_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]

#拨码开关连接，用于输入，依次为sw0,sw1……sw7

set\_property PACKAGE\_PIN AC21 [get\_ports wen[3]]

set\_property PACKAGE\_PIN AD24 [get\_ports wen[2]]

set\_property PACKAGE\_PIN AC22 [get\_ports wen[1]]

set\_property PACKAGE\_PIN AC23 [get\_ports wen[0]]

set\_property PACKAGE\_PIN AB6 [get\_ports ren[1]]

set\_property PACKAGE\_PIN W6 [get\_ports ren[0]]

set\_property PACKAGE\_PIN AA7 [get\_ports input\_sel[1]]

set\_property PACKAGE\_PIN Y6 [get\_ports input\_sel[0]]

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 ren]

set\_property IOSTANDARD LVCMOS33 [get\_ports input\_sel[1]]

set\_property IOSTANDARD LVCMOS33 [get\_ports input\_sel[0]]

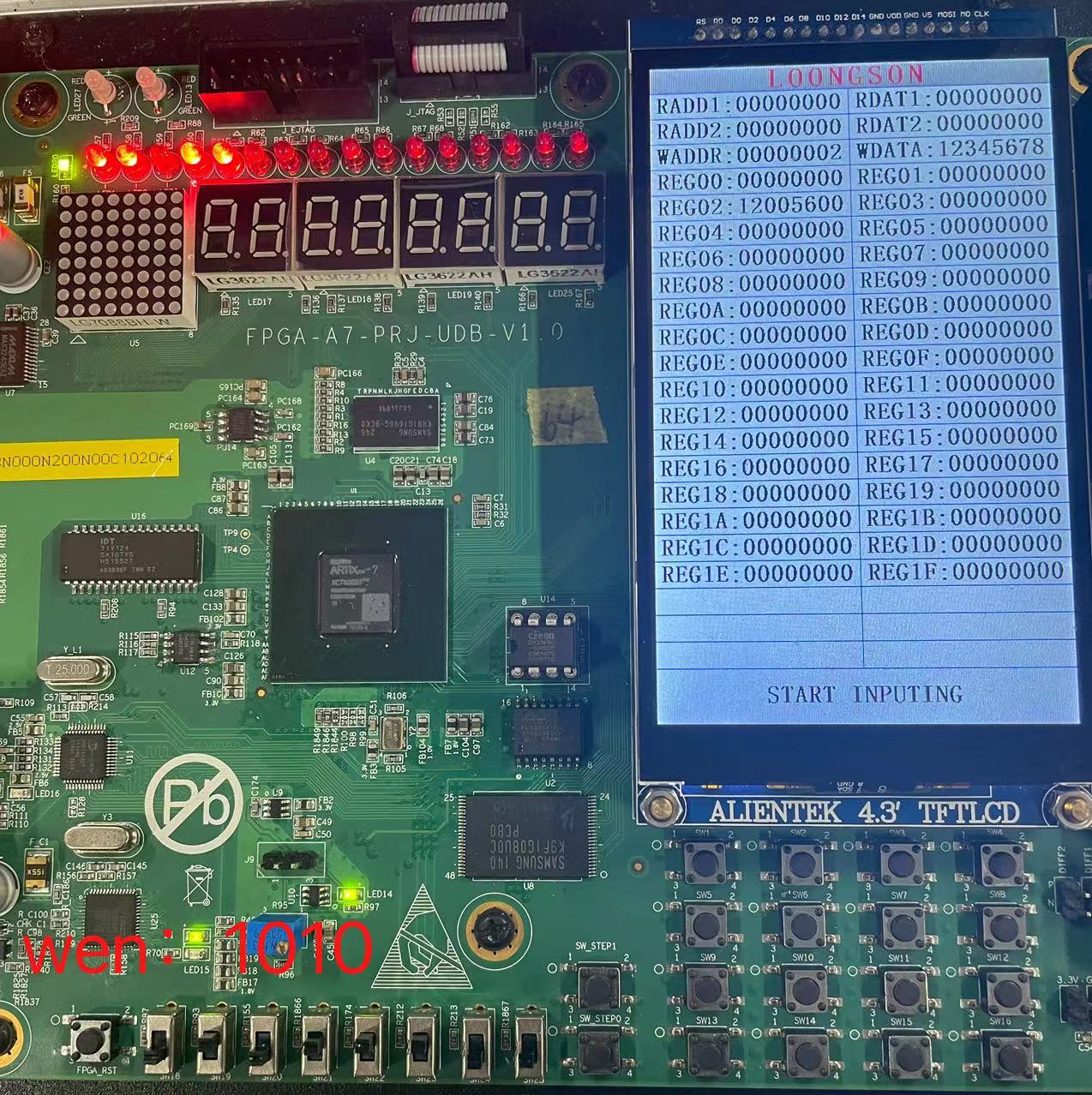
1. **实验结果分析**
2. 不同位置拨码开关表示的内容：

sw0表示wen[3]，sw1表示wen[2]，sw2表示wen[1]，sw3表示wen[0]

sw4表示ren[1]，sw5表示ren[0]，sw6表示input\_sel[1]，sw7表示input\_sel[0]

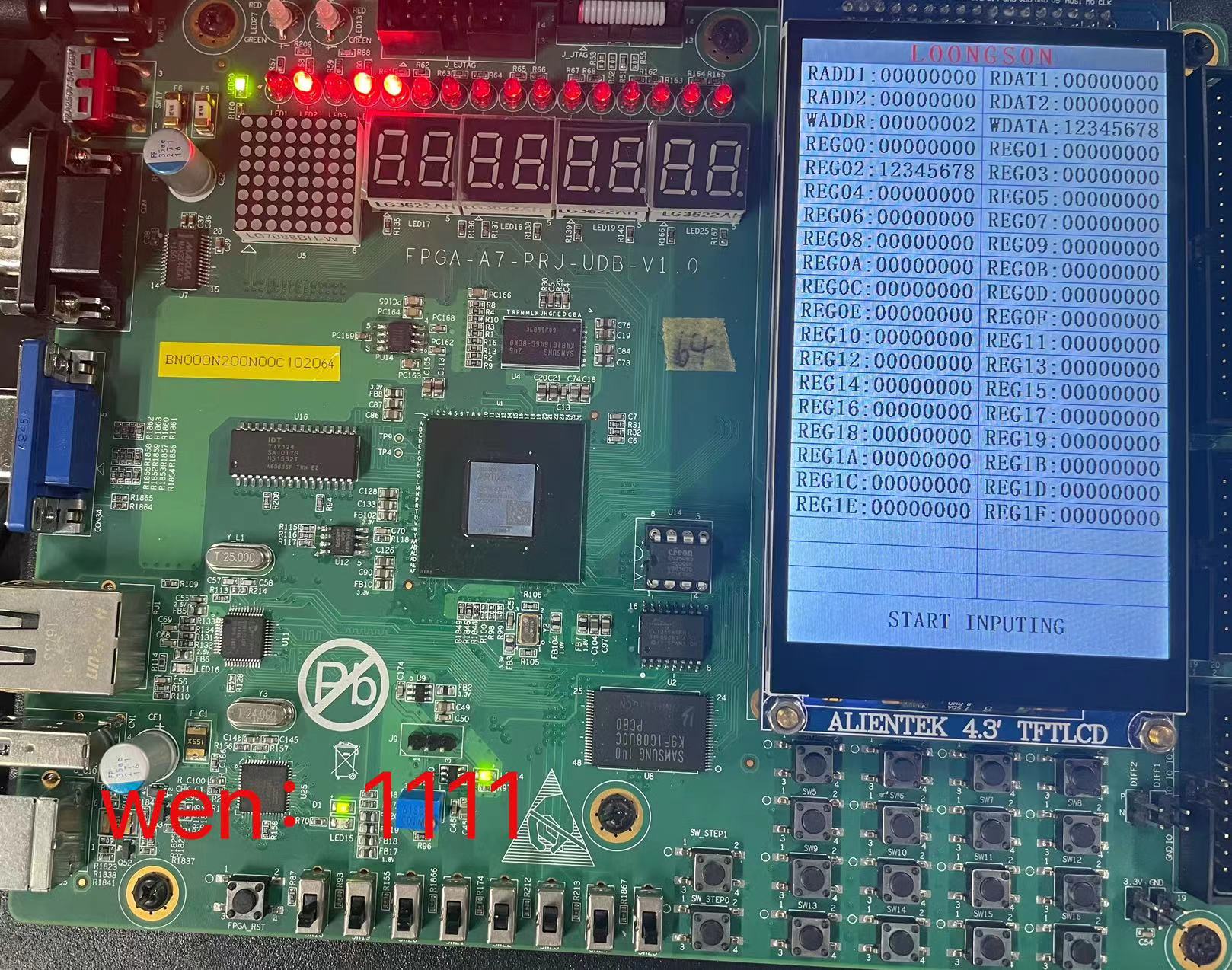


1. input\_sel为10时输入WADDR为00000002，input\_sel为11时输入WDATA为12345678，同时设置wen为1010，意思是把WDATA中的31-24和15：8位写入2号寄存器。



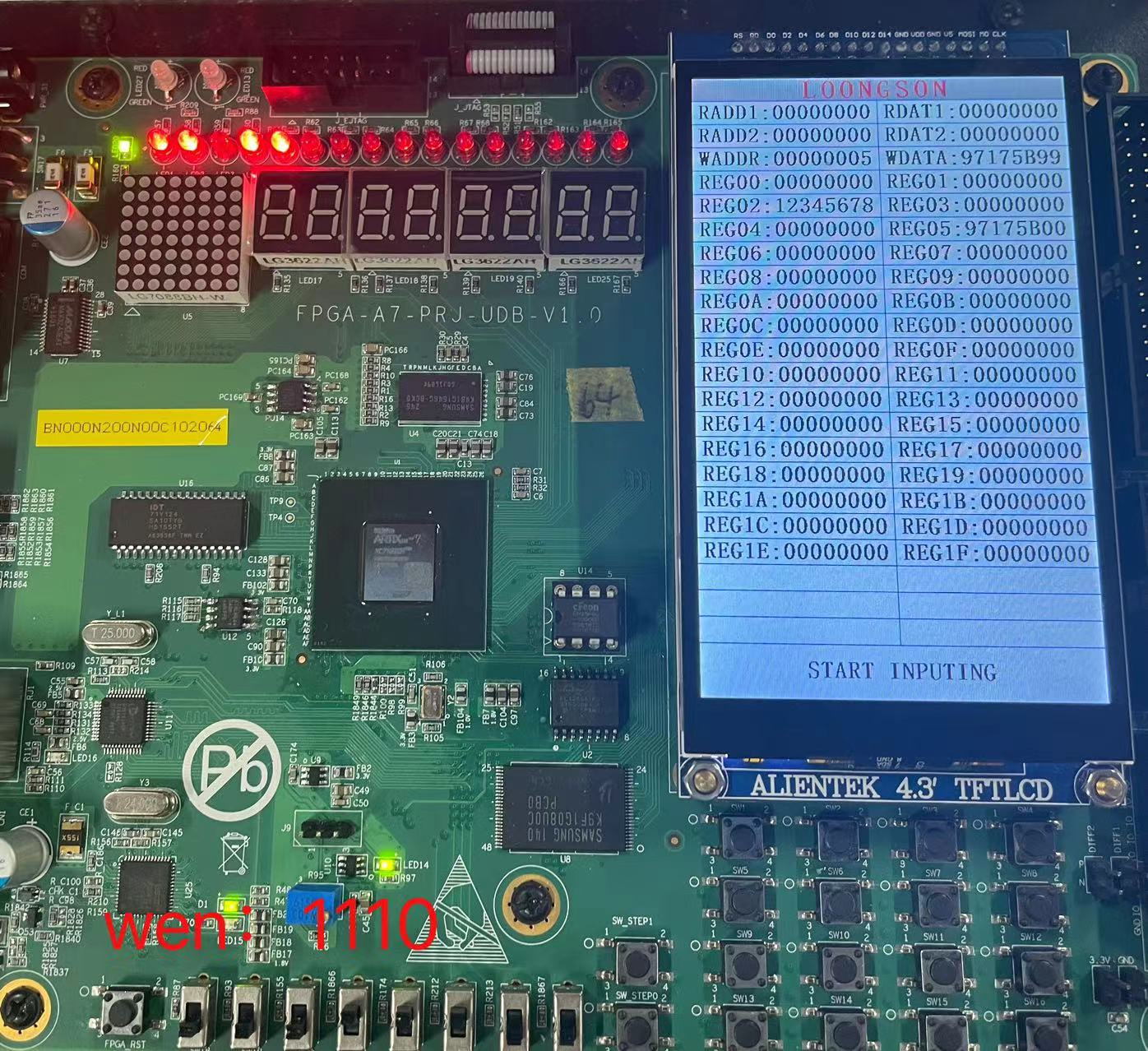
2号寄存器中的值为12005600，显示的内容正确。

（3）再（2）的基础上把wen设置为1111，意思是把12345678全部写入2号寄存器。



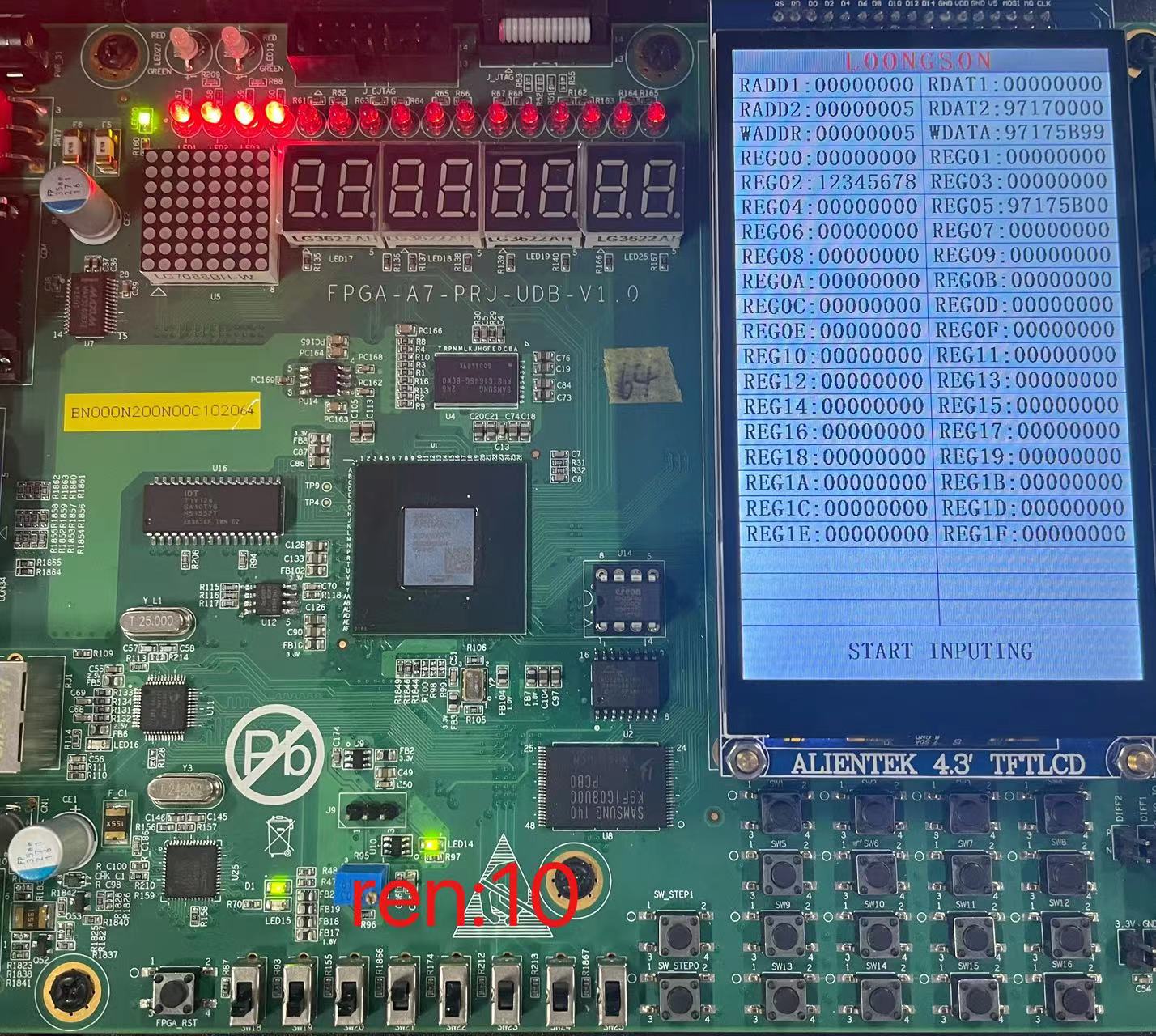
2号寄存器中的值位12345678，显示的内容正确。

（4）input\_sel为10时输入WADDR为00000005，input\_sel为11时输入WDATA为97175B99，同时设置wen为1110，意思是把WDATA中除了最低8位的值全部写入5号寄存器。



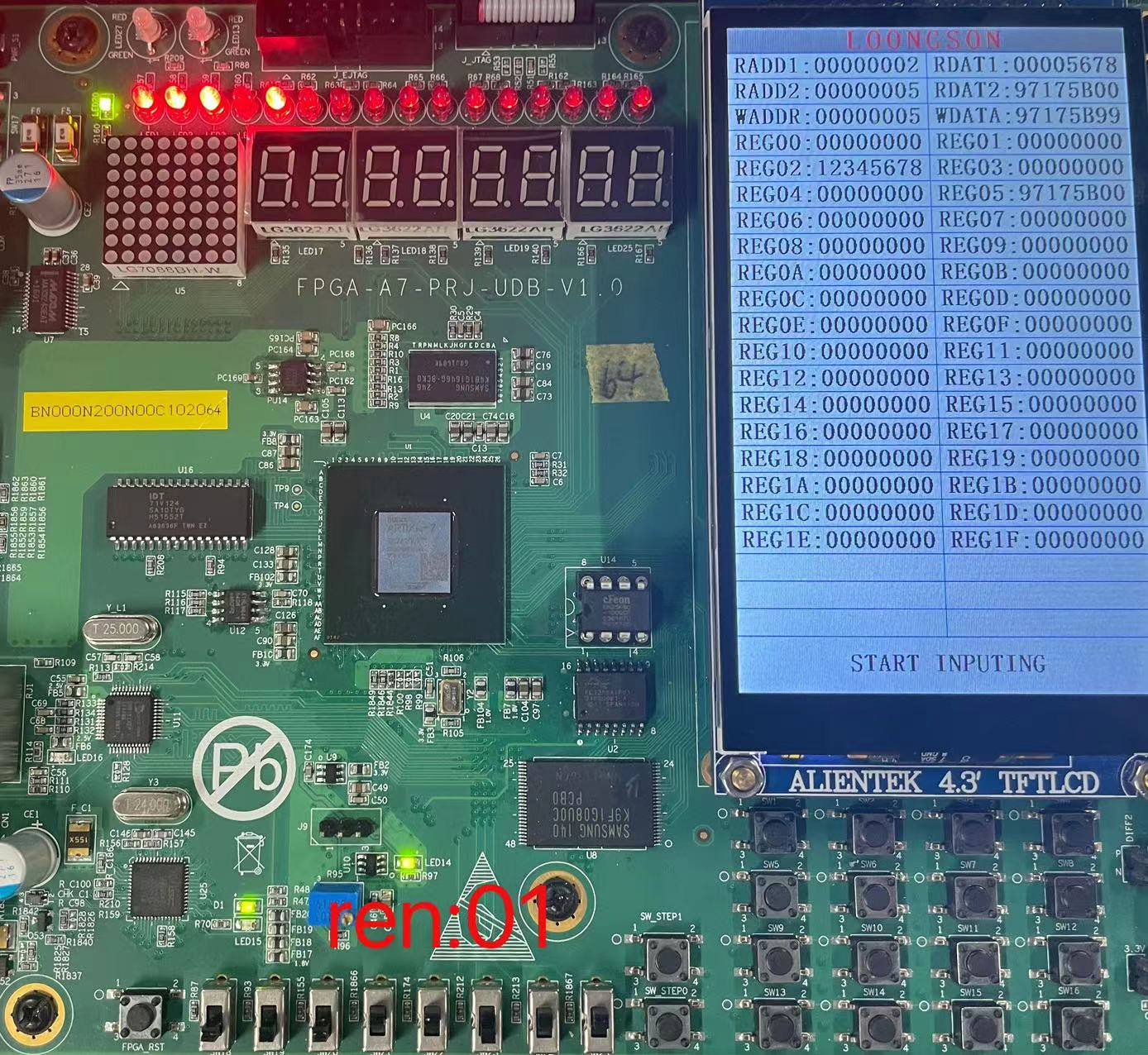
5号寄存器中的值位97175B00，显示的内容正确。

（5）input\_sel为01时输入RADD2为00000005，同时设置ren为10，意思是读取5号寄存器的高十六位。



5号寄存器中存储的值为97175B00，读取的内容为高十六位的9717，显示的内容正确。

（6）input\_sel为00时输入RADD1为00000002，同时设置ren为01，意思是读取2号寄存器的低十六位，并且由于寄存器堆的两个读端口同时控制，此时也会读取RADD2中寄存器的低十六位。



2号寄存器的值为12345678，可以看到RDAT1中读取的值为00005678，为低十六位，同时RDAT2中的值变为97175B00，也读取了RADD2中寄存器低十六位的值，显示的内容正确。

1. **总结感想**

此次实验让我更深入地理解了计算机体系结构中寄存器读取和写入的原理。

通过本次实验，我深入理解了寄存器堆的基本原理和设计。我学会了如何编写带有读写控制信号的寄存器堆，这使得我的设计更加灵活和可控。我还学会了如何在设计中使用模块化和层次化思想，这使得我的代码更加易于维护和扩展。

此外，在编写代码的过程中我通过查阅资料对于verilog语言更加熟悉了，比如我对于赋值操作=和<=分别在什么时候使用有了更加深入的了解，还对一些控制语句的使用更加熟练了。

总之，寄存器堆实验让我深入了解了计算机体系结构和数字电路设计的原理，为我以后的学习和工作奠定了坚实的基础。