**组成原理实验课程第 一 次实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | 加法器实验 | | | 班级 | 李涛老师 |
| 学生姓名 | 张洋 | 学号 | 2111460 | 指导老师 | 董老师 |
| 实验地点 | 津南实验楼A306 | | 实验时间 | 3月21日 18：30 – 20：30 | |

1. **实验目的**

1.熟悉LS-CPU-EXB-002实验箱和软件平台。

2.掌握利用该实验箱各项功能开发组成原理和体系结构实验的方法。

3.理解并掌握加法器的原理和设计。

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。

1. **实验内容说明**

在教材的实验基础之上，进行改进实验。改进实验中，要求设计两个模块：8位加法器和32位加法器，其中32位加法器通过调用8位加法器实现。利用Xilinx Vivado平台进行仿真验证，验证无误后用实验箱进行操作，通过 LCD触摸屏实现两个加数的输入，通过拨码开关实现一位来自低位进位的输入，并将其相加求和的结果通过 LCD 触摸屏中的指定区域显示，将进位结果通过LED灯表示，完成验证。

1. **实验原理图**



从图中可以看到，外围模块 adder\_display 内部调用了adder和lcd\_module 模块。adder模块中调用四个8位加法器模块，在外部除了时钟和复位信号外，还有cin和input\_sel通过拨码开关输入，以及cout输出到 led 灯上。

1. **实验步骤**

（1）新建工程

启动 vivado 软件，在菜单栏点击“File”->“New Project”，出现新建工程向导，选择“Next”，输入工程名称，选择工程的文件位置，然后选择“Next”。选择“RTL Project”，勾选“Do not specify sources at this time”，点击“Next”： 在筛选器的“family”选择“Artix 7”，“package”选择“fbg676”，在筛选得到的型号里面选择“xc7a200tfbg676-2”，然后选择“Next”，点击“Finish”。

（2）添加源文件

在“Project Manager”下点击“Add sources”，选择“Add or create design sources”；点击“Next”，创建文件，点击“Add Files”，选择 adder.v，点击“OK”，并在其中写入代码后保存。

（3）添加展示外围模块

在本例中，按照实验要求还需要一个外围模块，该外围模块调用 adder.v，且要调用触摸屏模块的以便在板上实验结果。新建外围模块 adder\_display.v，在其中写入代码之后保存。

（4）上板验证

将功能代码进行综合和布局布线后下载到 FPGA 板上运行，在板上检查运行的正确性。首先需要添加引脚绑定的约束文件。创建约束文件 adder.xdc，并在其中写入引脚绑定代码后保存。主要包括时钟与复位信号的引脚连接，led 灯和拨码开关的引脚连接，以及 LCD 触摸屏引脚的连接。之后进行综合、布局布线、产生可烧写文件。打开 FPGA 实验板，并将下载线与电脑相连后，打开电源，完成 bit 文件的烧写。之后就是对于实验板的操作，可以通过拨码开关，来控制输入不同的加数；可以通过 LCD 触摸屏上的小键盘，来输入不同的加数；可以通过一个拨码开关来控制来自低位的进位。

**改进代码（改进处标红）：**

**1.adder\_8.v //adder\_8模块为自定义的8位加法器模块，实现两个八位二进制数的输入和一个低位进位的输入，产生一个8位的结果和对下一个位置的高位进位输出。**

module adder\_8(

input [7:0] op1,

input [7:0] op2,

input op,

output [7:0] sum,

output flag

);

assign{flag , sum} = op1 + op2 + op;

endmodule

**2.adder\_32.v //adder\_32模块为自定义的32位加法器模块，通过调用4个8位加法器实现。声明第一个8位加法器u1，对0-7位的数字和低位进位相加，产生0-7位的结果和一个高位进位，高位进位用作第二个8位加法器的低位进位。声明第二个8位加法器u2，对8-15位的数字和低位进位相加，产生8-15位的结果和一个高位进位，高位进位用作第三个8位加法器的低位进位。声明第三个8位加法器u3，对16-23位的数字和低位进位相加，产生16-23位的结果和一个高位进位，高位进位用作第四个8位加法器的低位进位。声明第四个8位加法器u4，对24-31位的数字和低位进位相加，产生24-31位的结果和一个高位进位flag。**

module adder\_32(

input [31:0] op1,

input [31:0] op2,

input op,

output[31:0] sum,

output flag

);

wire [2:0] c;

//0-7位二进制相加

adder\_8 u1(.op1(op1[7:0]), .op2(op2[7:0]), .op(op), .sum(sum[7:0]), .flag(c[0]));

//8-15位二进制数相加

adder\_8 u2(.op1(op1[15:8]), .op2(op2[15:8]), .op(c[0]), .sum(sum[15:8]), .flag(c[1]));

//16-23位二进制数相加

adder\_8 u3(.op1(op1[23:16]),.op2(op2[23:16]),.op(c[1]),.sum(sum[23:16]),.flag(c[2]));

//24-31位二进制数相加

adder\_8 u4(.op1(op1[31:24]),.op2(op2[31:24]),.op(c[2]),.sum(sum[31:24]),.flag(flag));

endmodule

**3.adder\_display.v //添加展示外围模块**

module adder\_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，用于选择输入数和产生cin

input input\_sel, //0:输入为加数1(add\_operand1);1:输入为加数2(add\_operand2)

input sw\_cin,

//led灯，用于显示cout

output led\_cout,

//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

);

//-----{调用加法模块}begin

reg [31:0] adder\_operand1;

reg [31:0] adder\_operand2;

wire adder\_cin;

wire [31:0] adder\_result ;

wire adder\_cout;

adder\_32 adder\_module(

.op1(adder\_operand1),

.op2(adder\_operand2),

.op(adder\_cin),

.sum(adder\_result),

.flag(adder\_cout)

);

assign adder\_cin = sw\_cin;

assign led\_cout = adder\_cout;

//-----{调用加法模块}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

//当input\_sel为0时，表示输入数为加数1，即operand1

always @(posedge clk)

begin

if (!resetn)

begin

adder\_operand1 <= 32'd0;

end

else if (input\_valid && !input\_sel)

begin

adder\_operand1 <= input\_value;

end

end

//当input\_sel为1时，表示输入数为加数2，即operand2

always @(posedge clk)

begin

if (!resetn)

begin

adder\_operand2 <= 32'd0;

end

else if (input\_valid && input\_sel)

begin

adder\_operand2 <= 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

case(display\_number)

6'd4 : //lcd显示位置从1，2，3改为4，5，6

begin

display\_valid <= 1'b1;

display\_name <= "ADD\_1";

display\_value <= adder\_operand1;

end

6'd5 :

begin

display\_valid <= 1'b1;

display\_name <= "ADD\_2";

display\_value <= adder\_operand2;

end

6'd6 :

begin

display\_valid <= 1'b1;

display\_name <= "RESUL";

display\_value <= adder\_result;

end

default :

begin

display\_valid <= 1'b0;

display\_name <= 40'd0;

display\_value <= 32'd0;

end

endcase

end

//-----{输出到触摸屏显示}end

//----------------------{调用触摸屏模块}end---------------------//

Endmodule

**4.testbench.v //在进行功能仿真时，需要先建立一个 testbench(测试平台)。产生输入激励，送入到要测试的功能模块里，然后读出功能模块的执行结果，与预期的结果进行比较，以此验证功能模块的正确性。在本题中，需要产生的输入激励就是 2 个加数和 1 个低位进位信号，在该激励输入到加法功能模块中后，会输出加法结果和向高位的进位信号。仿真的过程中会产生波形文件，可以通过观察波形文件确定功能的正确性，在出错的情况下可以定位错误位置。**

module testbench;

// Inputs

reg [31:0] operand1;

reg [31:0] operand2;

reg cin;

// Outputs

wire [31:0] result;

wire cout;

// Instantiate the Unit Under Test (UUT)

adder\_32 uut(operand1,operand2,cin,result,cout);

initial begin

// Initialize Inputs

operand1 = 0;

operand2 = 0;

cin = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

end

always #10 operand1 = $random; //$random为系统任务，产生一个随机的32位数

always #10 operand2 = $random; //#10 表示等待10个单位时间(10ns)，即每过10ns，赋值一个随机的32位数

always #10 cin = {$random} % 2; //加了拼接符，{$random}产生一个非负数，除2取余得到0或1

endmodule

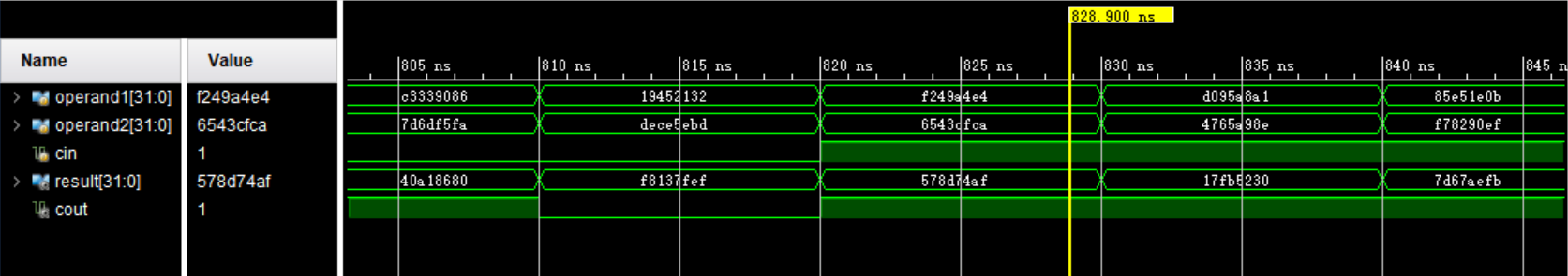
**5.adder.xdc** **//在有了板上验证机制后，需要添加引脚绑定的约束文件。所谓约束文件就是将顶层模块(本例中为 adder\_display)的输入输出端口与 FPGA 板上的 IO 接口引脚绑定，以完成在板上的输入输出。**

（相较于源代码没有更改故没有粘贴到实验报告中）

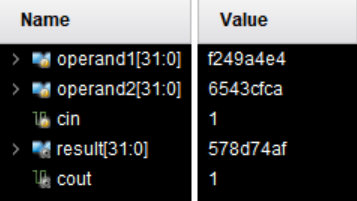
1. **实验结果分析**

（仿真结果截图或者实验箱运行结果拍照，注意需要对实验结果进行分析，输入是什么，输出是什么，结果是什么，是否验证了正确性）

仿真：

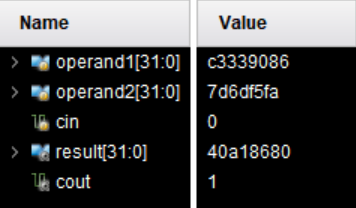


验证：

 f249a4e4 + 6543cfca + 1 = 1578D74AE

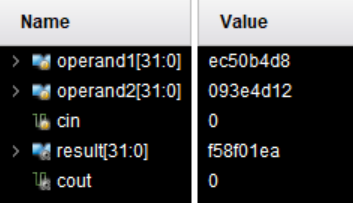
第一个加数：f249a4e4；第二个加数：6543cfca；低位进位：1

结果输出：578D74AE；高位进位：1

 c3339086 + 7d6df5fa = 140A18680

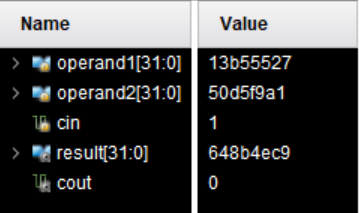
第一个加数：c3339086；第二个加数：7d6df5fa；低位进位：0

结果输出：40A18680；高位进位：1

 ec50b4d8 + 093e4d12 = f58f01ea

第一个加数：ec50b4d8；第二个加数：093e4d12；低位进位：0

结果输出：f58f01ea；高位进位：0

 13b55527 + 50d5f9a1 + 1 = 648b4ec9

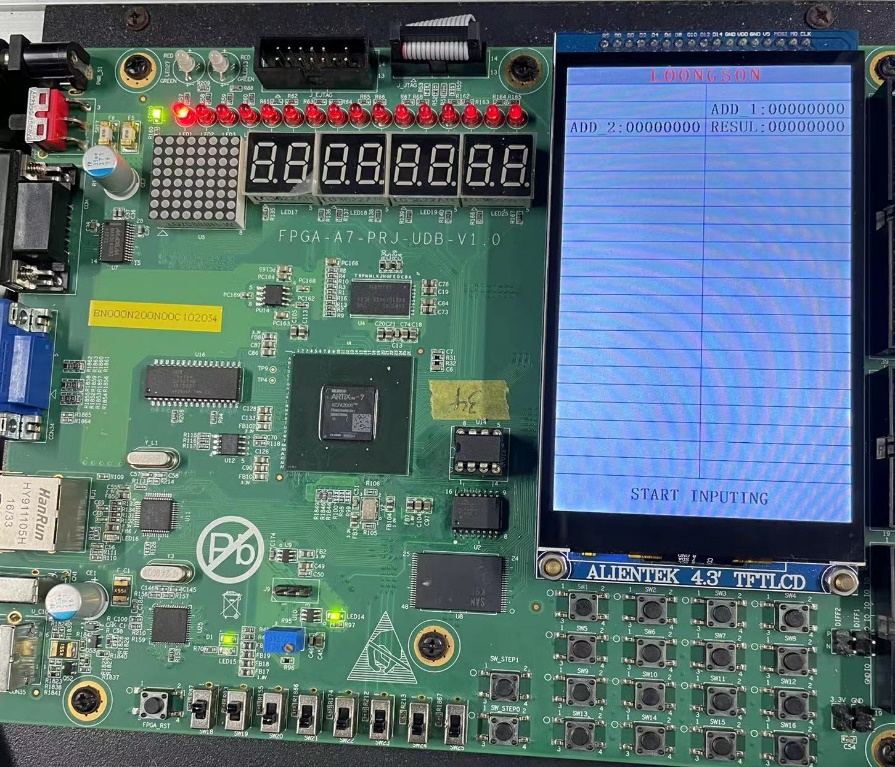
第一个加数：13b55527；第二个加数：50d5f9a1；低位进位：1

结果输出：648b4ec9；高位进位：0

结果正确。通过观察波形可以看到加法功能模块随机测试并没有出错，我们认为功能趋于稳定，可以认为是正确的。至此，代码编辑和功能仿真都已完成，认为功能基本正确，后续流程就是上板验证了。

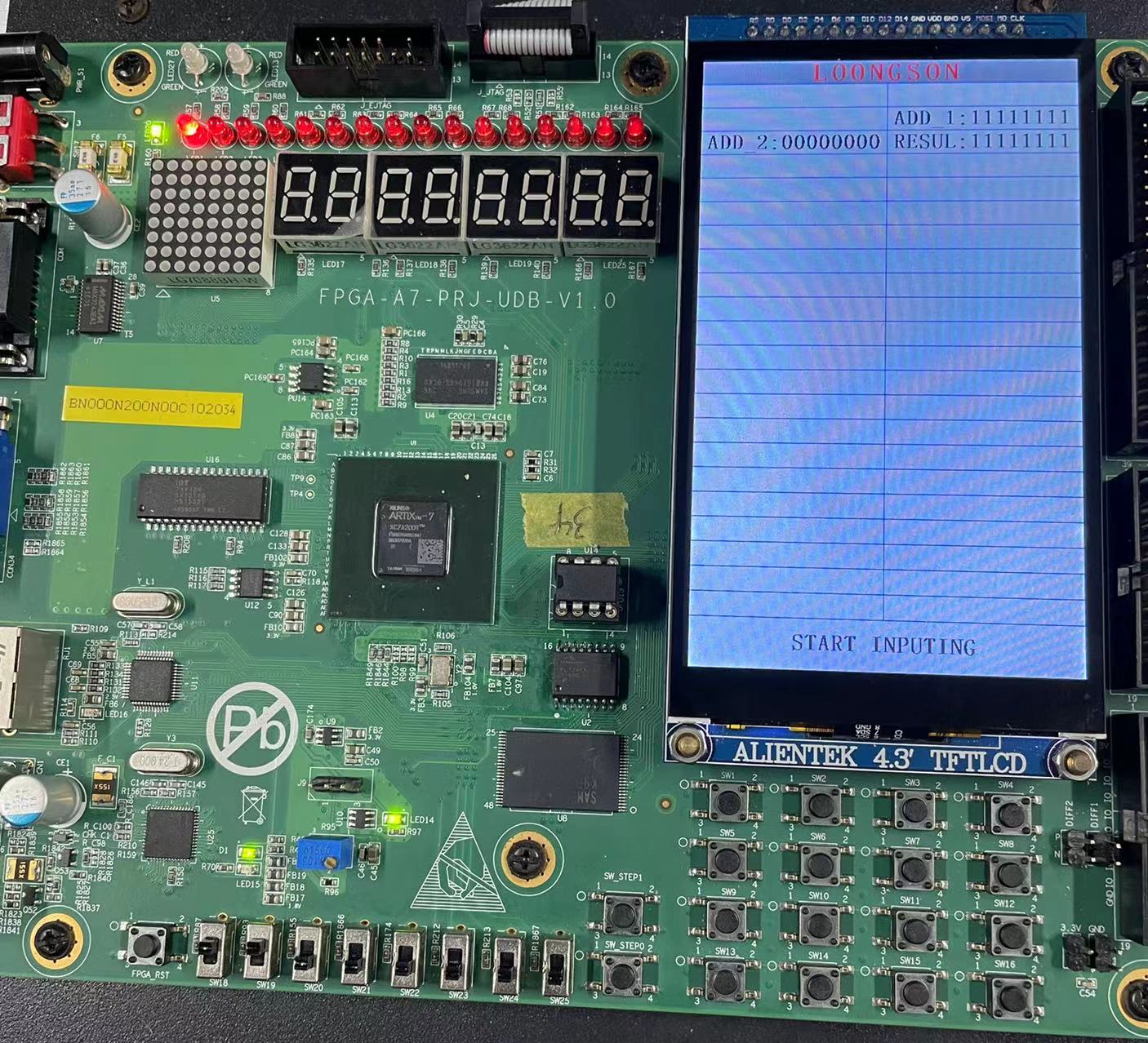
在实验箱上验证：

1. 输入前



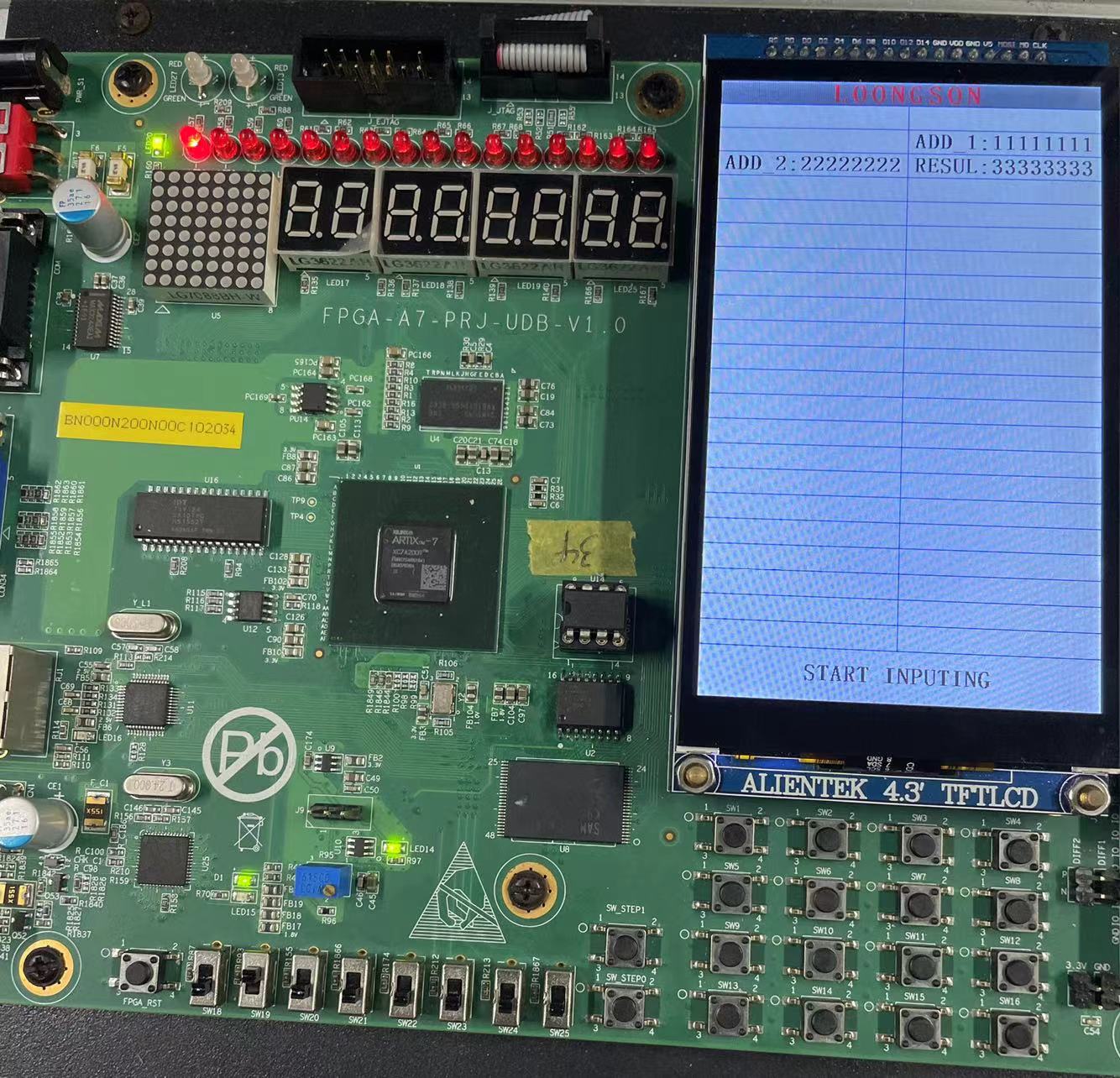
ADD\_1:00000000 ADD\_2:00000000 RESUL:00000000 低位进位:0 高位进位:0（灯亮）

1. 输入第一个加数



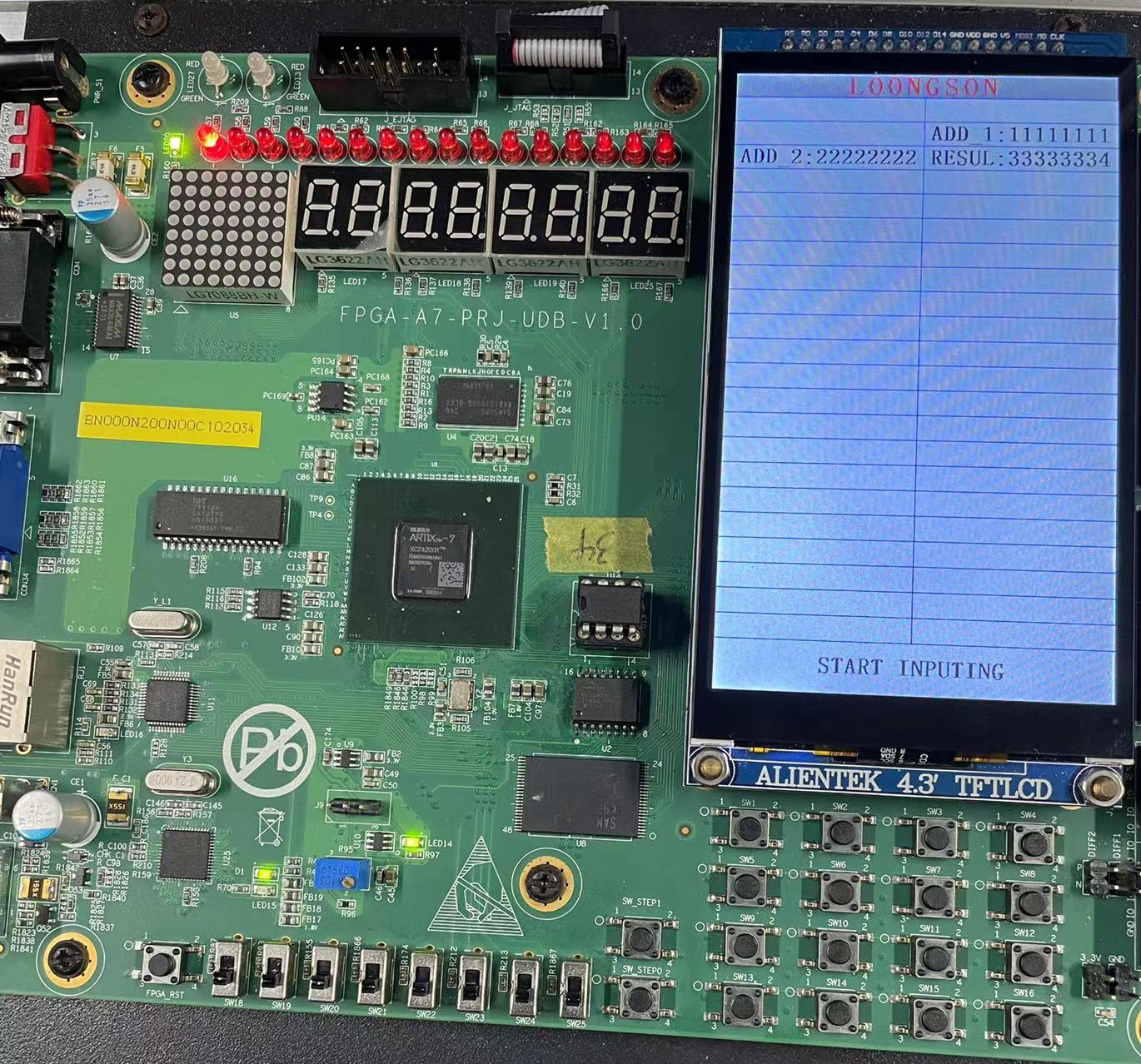
ADD\_1:11111111 ADD\_2:00000000 RESUL:11111111 低位进位:0 高位进位:0（灯亮）

1. 输入第二个加数



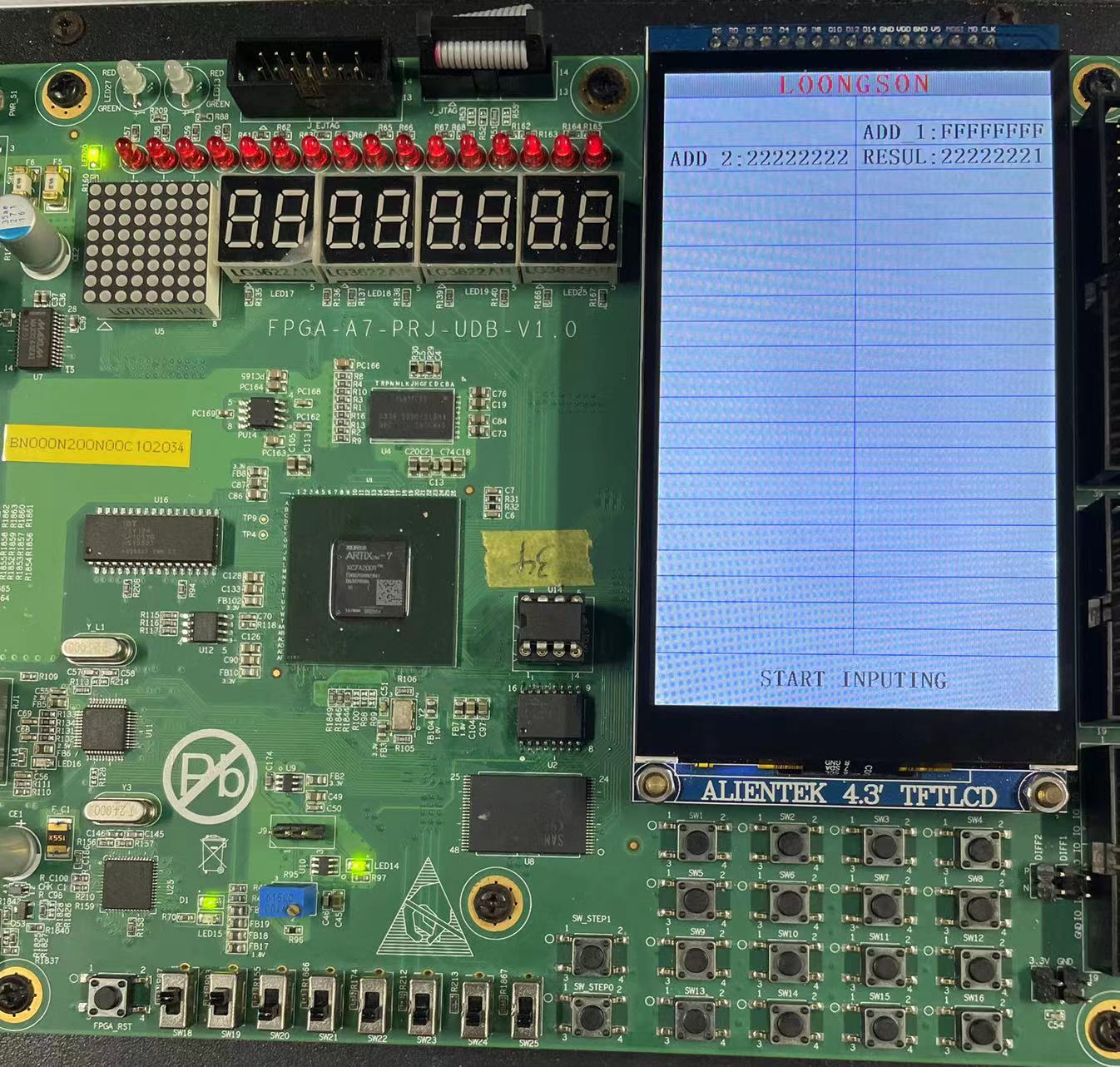
ADD\_1:11111111 ADD\_2:22222222 RESUL:33333333 低位进位:0 高位进位:0（灯亮）

1. 拨动低位进位的拨码开关



ADD\_1:11111111 ADD\_2:22222222 RESUL:33333334 低位进位:1 高位进位:0（灯亮）

1. 更改第一个加数验证显示高位进位的LED灯的亮灭



ADD\_1:11111111 ADD\_2:FFFFFFFF RESUL:11111110 低位进位:0 高位进位:1（灯灭）

1. **总结感想**

在实验中，我们使用了4个八位加法器来实现32位加法器。整个实验过程中，我学到了许多有关数字电路和计算机组成原理的知识，并且对于数字电路和逻辑设计方面的知识有了更深刻的理解。

首先，实验的过程非常有趣。我们需要对八位加法器的原理和功能进行深入的理解，并且需要将它们集成到一个更大的电路中，以实现32位加法器。在实验的初期，我遇到了一些挑战。我对于Verilog语言不太熟悉，在做实验前花费了大量时间去了解Verilog语言，也会出现代码错误导致不能正确输出结果。然而，这些错误让我更加了解数字电路的工作原理，因为我需要仔细检查每个部分，以找出错误的原因。

此外，这个实验还提醒我数字电路设计的可扩展性。通过将多个八位加法器组合在一起，我们可以构建一个更大的电路，以实现更高级别的计算。这个实验让我意识到，在数字电路设计中，我们可以使用类似的方法来构建更复杂的电路和系统以满足更高级别的计算需求。

总之，这个实验让我对数字电路，计算机组成原理和逻辑设计方面的知识有了更深入的理解。