组成原理实验课程第1次实验报告

实验名称：数据运算——定点加法

学生姓名：许洋 学号：2313721 班级：张金老师

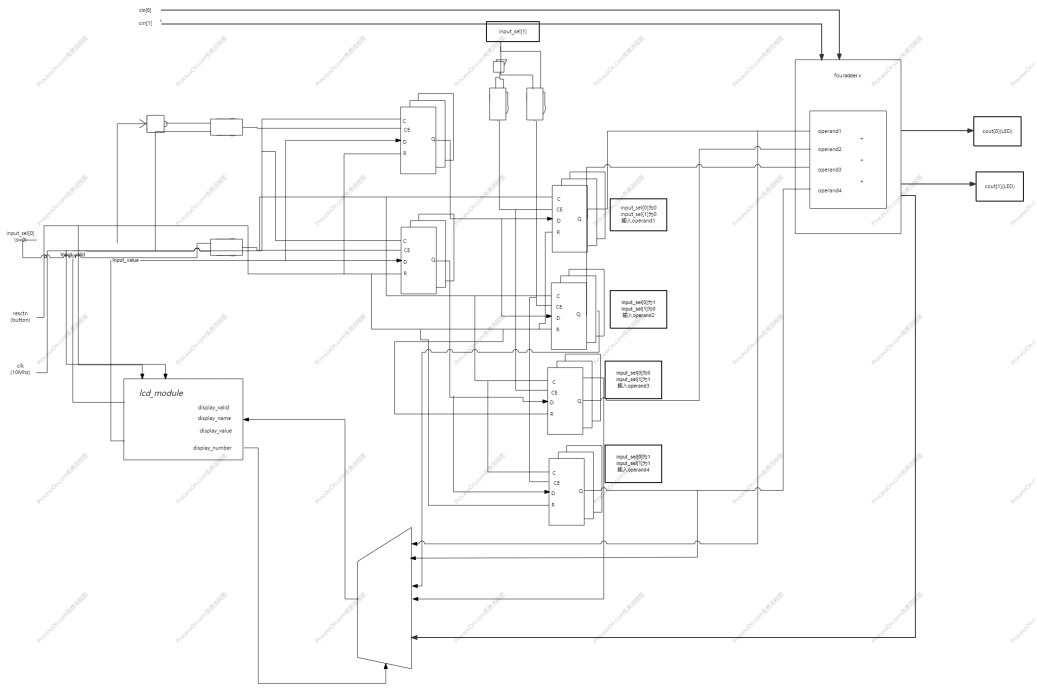
一、实验目的

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。

二、实验内容说明

1. 阅读 LS-CPU-EXB-002 实验箱相关文档，熟悉硬件平台,特别需要掌握利用显示屏观察特定信号的方法。学习软件平台和设计流程。
2. 熟悉计算机中加法器的原理。
3. 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，本次实验的加法器可以使用全加器自己搭建加法模块，也可以在 verilog 中直接使用“+”（系统是自动调用库里加法 IP，且面积时序更优），依据教师要求选择一种方法实现。
4. 根据设计的实验方案，使用 verilog 编写相应代码。
5. 对编写的代码进行仿真，得到正确的波形图。
6. 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，外围模块中调用封装好的触摸屏模块，显示两个加数和加法结果，且需要利用触摸功能输入两个加数。
7. 将编写的代码进行综合布局布线，并下载到实验箱中的 FPGA 板上进行演示。

三、实验原理图



四、实验步骤

1.adder32模块的实现

功能解释

首先，我们需要实现4个32位数的相加操作，原来的文档里实现的是两个32位数进行相加，因此需要扩展被加数，进位和输出。

代码部分，我们首先使用4个32位位宽的输入operand来实现原始值的写入；再将低位进位扩展成2位位宽、高位进位扩展成2位位宽来实现代码内容的修改。关键部分assign的实现，是我们使用4个数进行相加，再加上进位输入。

修改

1.首先，对input的数量进行修改，原来只需要输入两个operand的，现在需要4个，所以我们改成使用operand1~operand4。

```
    input  [31:0] operand1,
    input  [31:0] operand2,
    input  [31:0] operand3,
    input  [31:0] operand4,
```

2.然后，对进位位宽做了修改，原先是进位1位，只需要一个cin；现在由于是四个数相加，所以我们需要两位cin，所以我们将cin改为两位位宽。

```
    input [1:0] cin,
```

3.同理，在修改进位cin后，我们也需要对向高位的进位cout进行修改，由于是四个数相加，则会出现两位向高位的进位，因此需要两位cout，将两个向高位的进位扩展为两位位宽。

```
    output [1:0] cout
```

2.adder_display模块的实现

功能解释

adder_display是本项目的外围模块，该外围模块调用adder32.v，并且调用触摸屏上的模块，以便于在板上获得实验结果。

修改

1.对于语句input input_sel，原来的意思是：sel的值为0，代表输入为加数1(operand1);sel的值为1，代表输入为加数2(operand2)。为了实现4个加数的输入，我们将sel扩展为两位位宽，就能完整表示0-3的数（二进制），就能对应到4个加数。

```
    input [1:0] input_sel,
```

2.对于语句input sw_cin和output led_cout这两句，原先都是实现进位和显示LED灯的，我们在此处都扩展为两位位宽。

```
    input [1:0] sw_cin,
    output [1:0] led_cout,
```

3.然后就是对调用加法模块的修改。我们将两个32位寄存器operand1和operand2的输入，修改为使用四个32位寄存器来输入，分别用两位的cin和cout来控制。

```
     reg  [31:0] adder_operand1;
    reg  [31:0] adder_operand2;
    reg  [31:0] adder_operand3;
    reg  [31:0] adder_operand4;
    wire [1:0] adder_cin;
    wire [31:0] adder_result;
    wire [1:0]  adder_cout;
    adder adder_module(
        .operand1(adder_operand1),
        .operand2(adder_operand2),
        .operand3(adder_operand3),
        .operand4(adder_operand4),
        .cin     (adder_cin     ),
        .result  (adder_result  ),
        .cout    (adder_cout    )
    );
    assign adder_cin = sw_cin;
    assign led_cout  = adder_cout;
```

4.然后是对数的输入的修改，原先的实现方式是，通过input sel的控制，sel输出0就代表是加数1；sel输出1就代表加数是2，再进行分别对应的输出。我们现在需要判定加数1-4，所以使用两位位宽的sel的四种情况来对应4个加数00/01/10/11，通过一个case语句来表现加数的选择。

```
```

```
always @(posedge clk)
    begin
        if (!resetn)
        begin
            adder_operand1 <= 32'd0;
            adder_operand2 <= 32'd0;
            adder_operand3 <= 32'd0;
            adder_operand4 <= 32'd0;
        end
        else if (input_valid)
        begin
            case(input_sel)
                2'b00: adder_operand1 <= input_value;
                2'b01: adder_operand2 <= input_value;
                2'b10: adder_operand3 <= input_value;
                2'b11: adder_operand4 <= input_value;
            endcase
        end
    end
```

5.最后是输出到触摸屏的模块，我们只需要将原来的调用加法的个数从2改到4就可以了。我们增加以下的代码。

```
    6'd3 :
    begin
        display_valid <= 1'b1;
        display_name  <= "ADD_3";
        display_value <= adder_operand3;
    end
    6'd4 :
    begin
        display_valid <= 1'b1;
        display_name  <= "ADD_4";
        display_value <= adder_operand4;
    end
```

3.testbench模块的实现

功能解释

该部分是用于实现功能仿真，以此来检验功能的正确性，在出错的情况下可以准确定位到错误的位置。我们需要将输入激励由2个改到4个，进位信号由1个改到2个就可以了。

修改

1.将输入的寄存器改为了4个，进位输入的寄存器改为了2位，输出的cout改为了2位。

```
    reg [31:0] operand1;
    reg [31:0] operand2;
    reg [31:0] operand3;
    reg [31:0] operand4;
    reg [1:0] cin;
    wire [31:0] result;
    wire [1:0] cout;
```

2.对于uut模块，也是只需要修改输入、进位输入与输出的个数就可以了。

```
    adder32 uut (
        .operand1(operand1),
        .operand2(operand2),
        .operand3(operand3),
        .operand4(operand4),
        .cin(cin),
        .result(result),
        .cout(cout)
    );
```

3.对于开始模拟的版块，我们修改初始输入的个数，从2改为4，这样实现了初始的四输入。同样的，修改cin的位数，修改后期随机生成模拟的变量个数，即可实现模拟仿真的功能。

```
    initial begin

        operand1 = 32'b0;
        operand2 = 32'b0;
        operand3 = 32'b0;
        operand4 = 32'b0;
        cin = 2'b0;

        #100;
    end
    always #10 operand1 = $random;
    always #10 operand2 = $random;
    always #10 operand3 = $random;
    always #10 operand4 = $random;
    always #10 cin = {$random} % 4;
```

4.mycons模块的实现

功能解释

该文件是一个约束文件，功能是添加引脚绑定，使实验箱的引脚与我们的功能联系起来。

修改

1.修改对应的输出LED灯，写两句，分别对应相应的引脚。

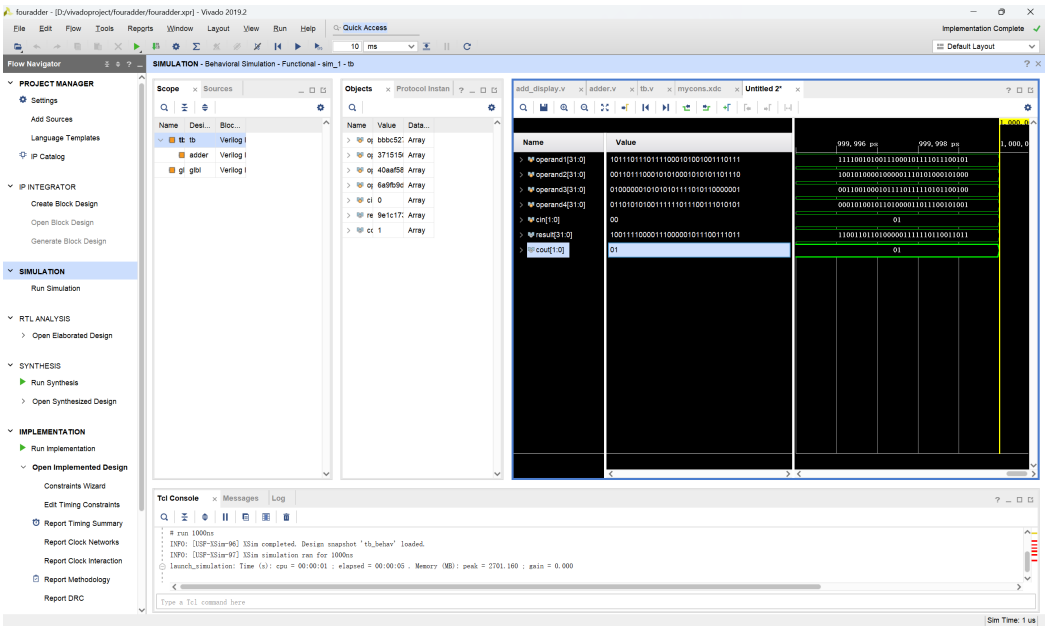
```
set_property PACKAGE_PIN H7   [get_ports {led_cout[0]}]
set_property PACKAGE_PIN D5   [get_ports {led_cout[1]}]
```

2.修改对应的sel和cin的引脚，分别对应到实验箱的1-4开关，通过开关的切换就可以实现加数的选择与进位的输入。后面的对于IOSTANDARD的修改也是同理，就不再说明了。

```
set_property PACKAGE_PIN AC21 [get_ports {input_sel[0]}]
set_property PACKAGE_PIN AD24 [get_ports {input_sel[1]}]
set_property PACKAGE_PIN AC22 [get_ports {sw_cin[0]}]
set_property PACKAGE_PIN AC23 [get_ports {sw_cin[1]}]
```

五、实验结果分析

1.仿真验证



以上是我们的仿真验证文件随机生成的波形图，使用计算器验证一下结果是否正确。

1011101110111100010100100111011 1+0011011100010101000101010110111 0+0100000001010101111010110000001+0110101010011111101110011101010 1=0001100111100001110000010111001110 11

结果正确。

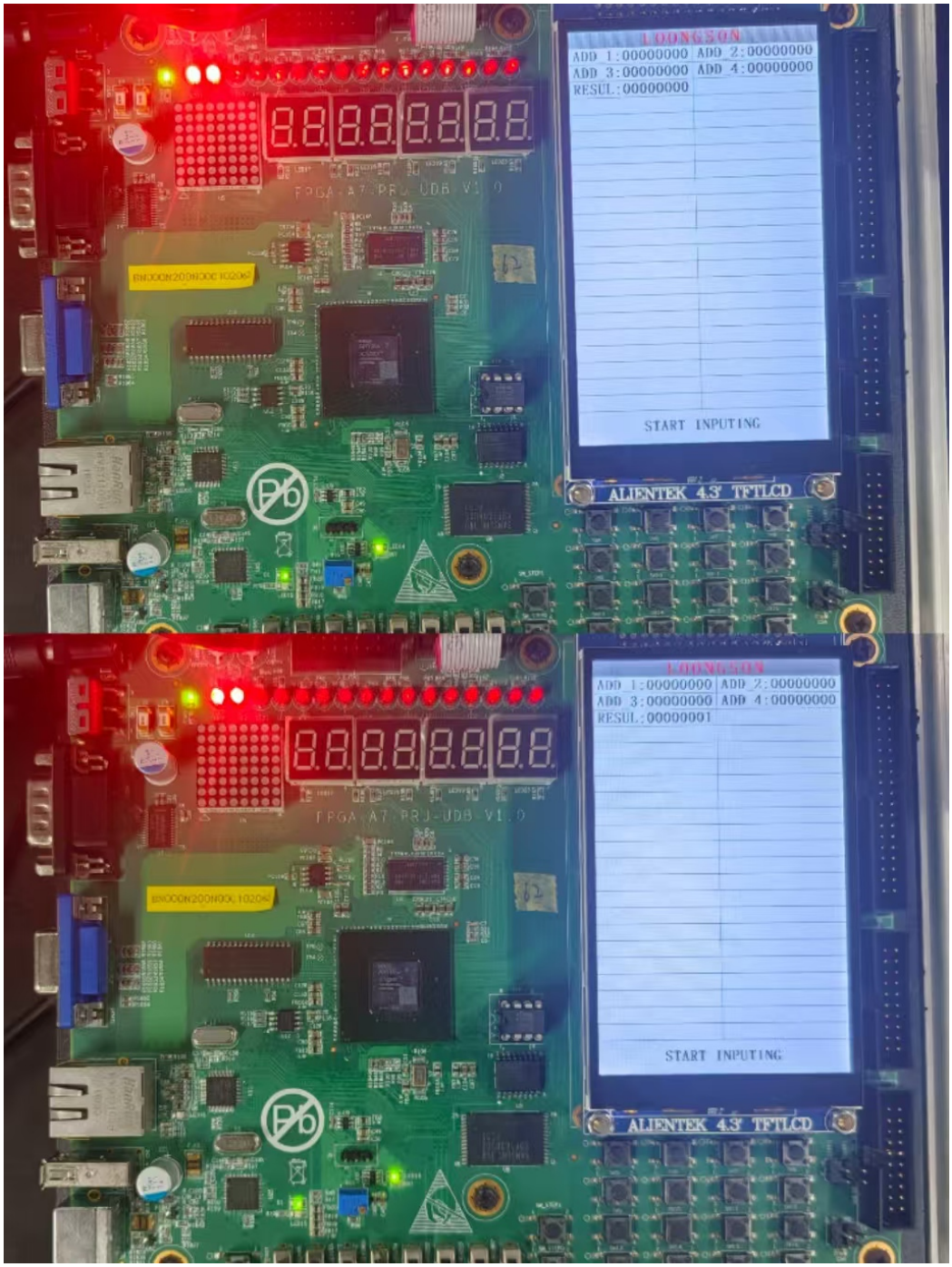
2.上箱验证

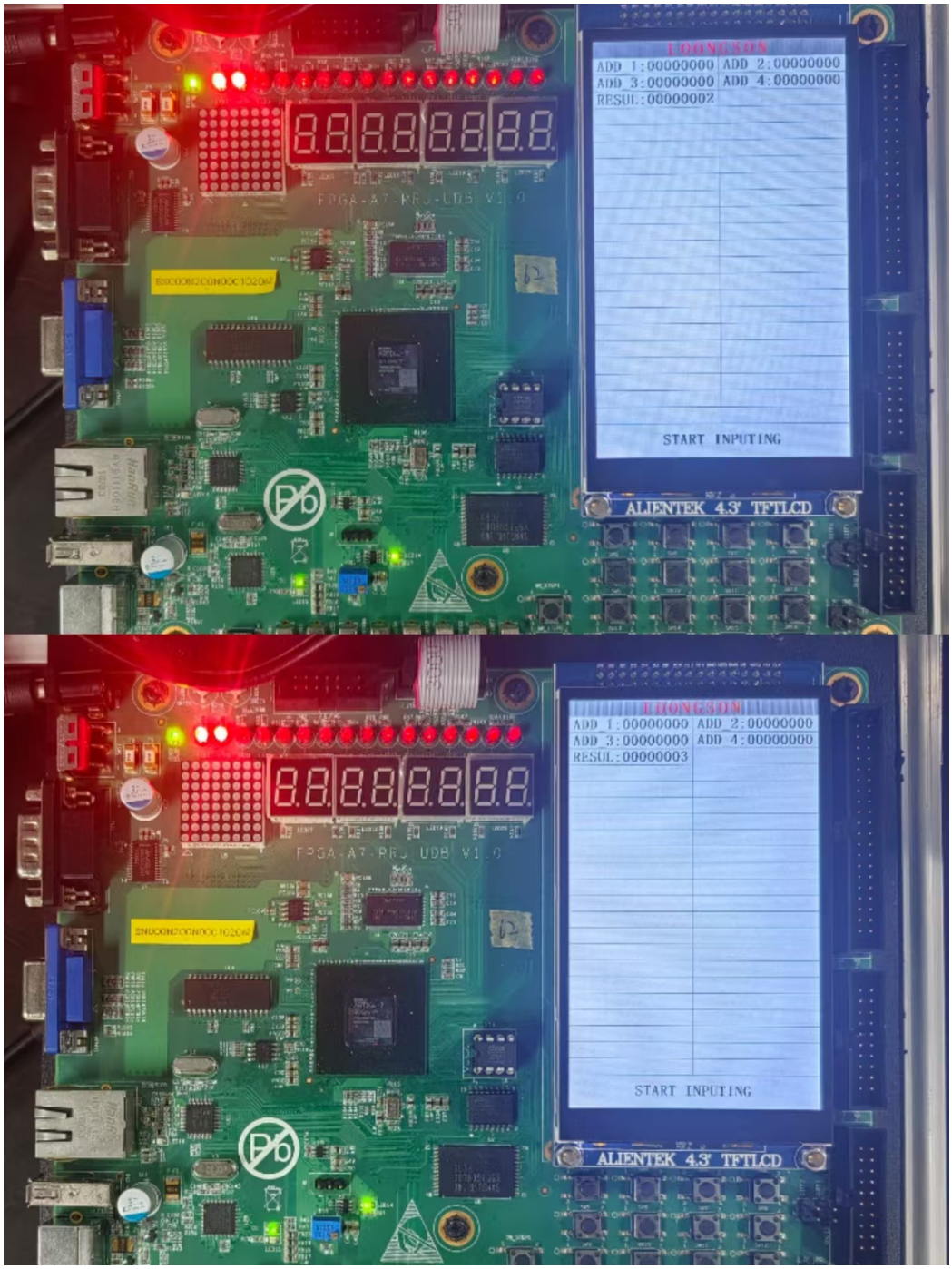
在实验箱上，从左边开始的第一个和第二个开关是控制加数选择的；第三个和第四个开关是手动输入进位的，我们分别进行上箱操作，得出以下结果。

(1)输入进位情况

我们调整开关3，4的情况来观察输入进位是否有效。

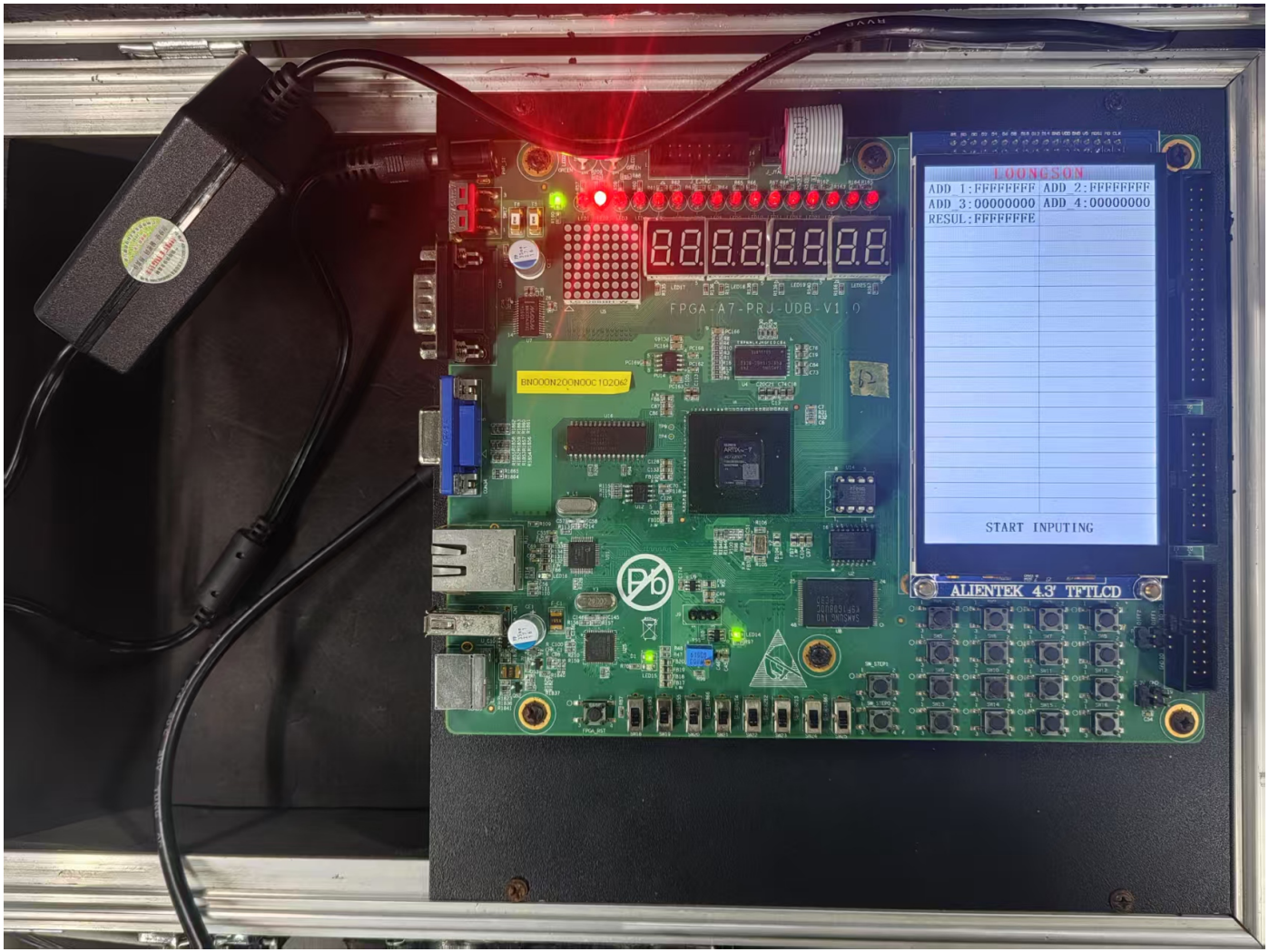
下面四张图片展示了进位情况。可以看到通过开关拨动展示了输入进位的4种情况。

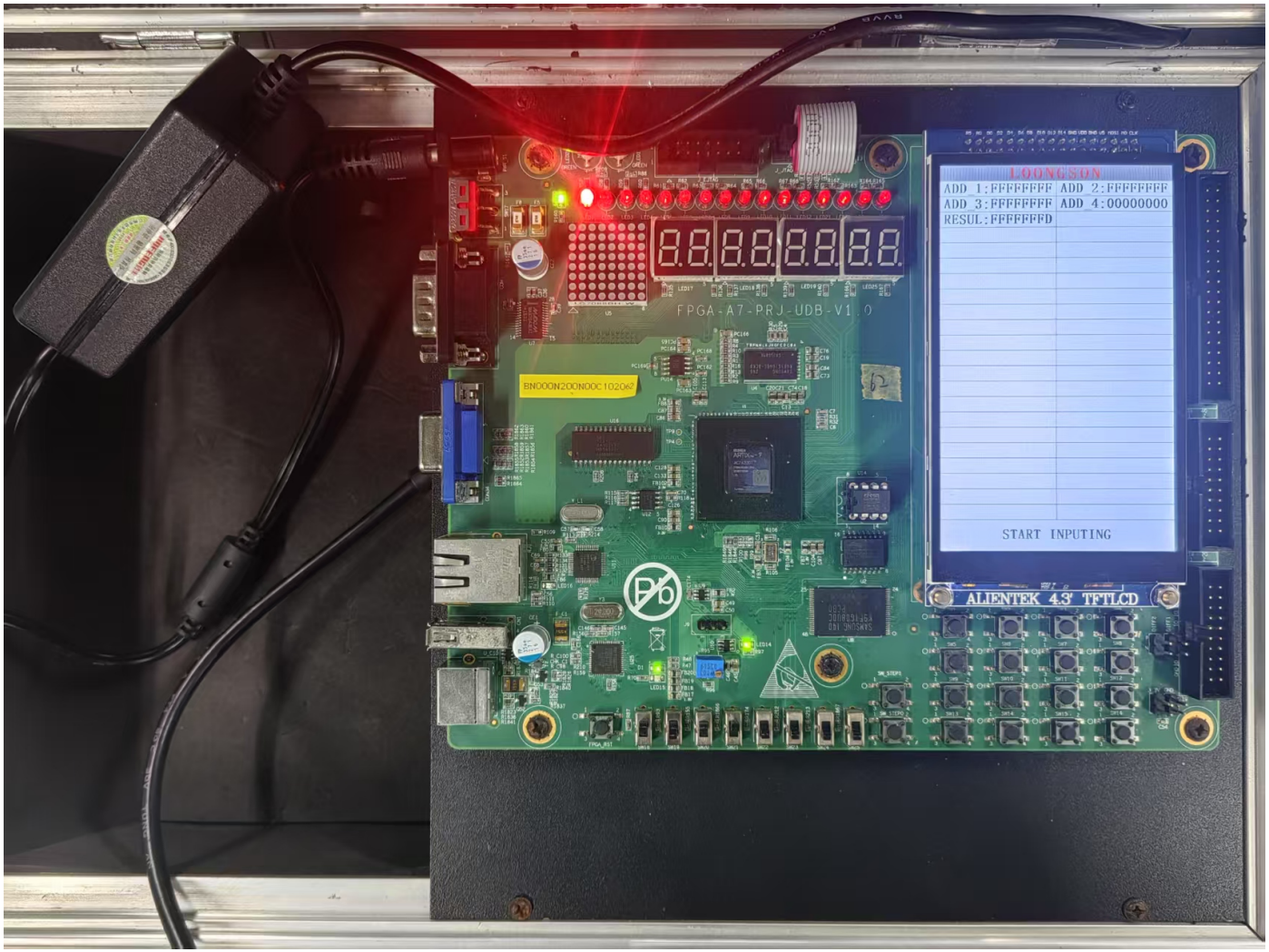


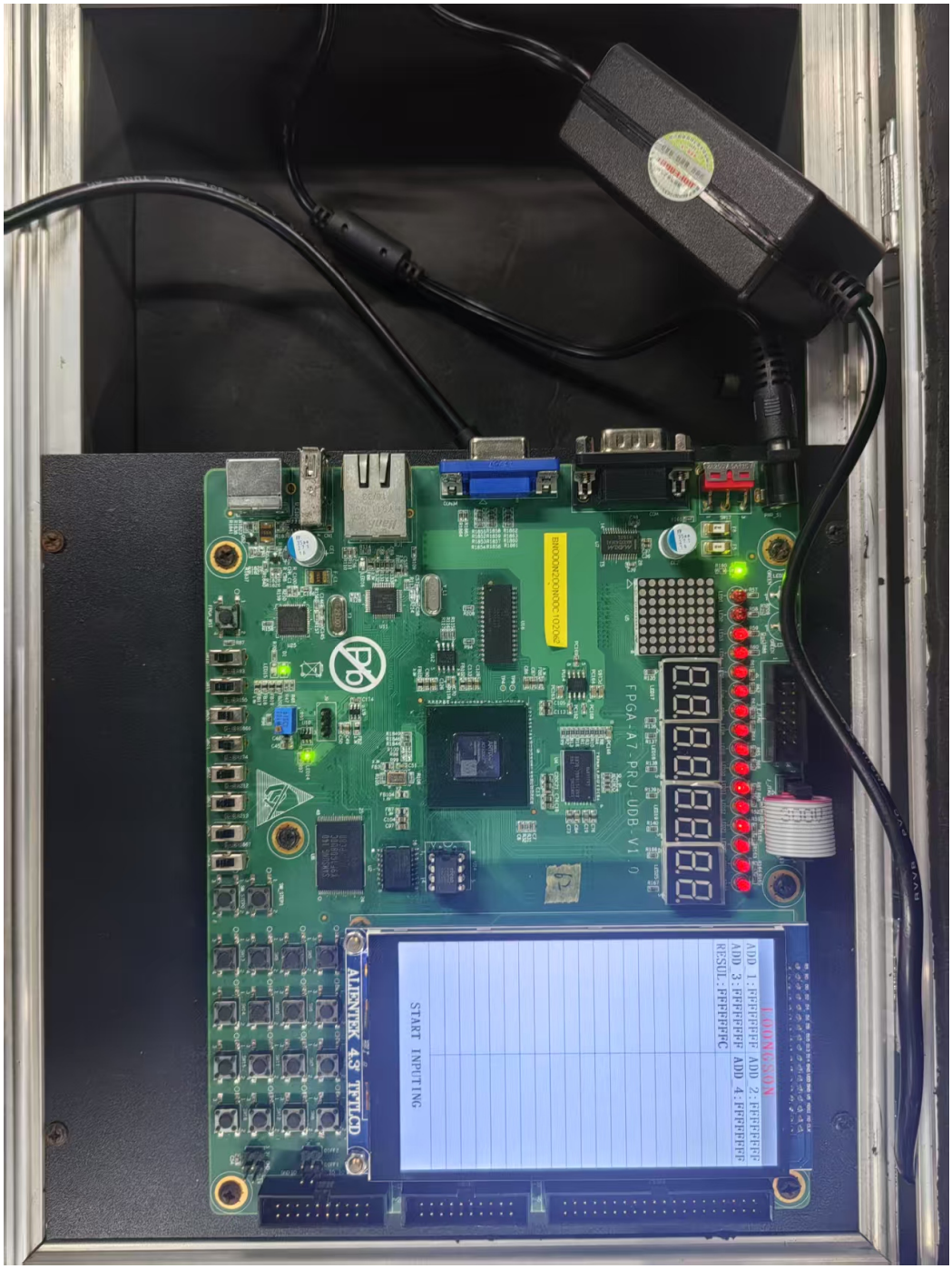


(2)输出进位情况

通过对4个加数进行不同的赋值观察led灯的亮和灭来观察输出进位情况。可以看到有四种不同的情况（有一种情况在（1）的图中）。


LOONGSON
ADD_1:FFFFFFFF
ADD_2:FFFFFFFF
ADD_3:00000000
ADD_4:00000000
RESUL:FFFFFFFE
START INPUTING
ALIENTEK 4.3' TFTLCD
FPGA-A7-PRJ-UDB-V1.0


LOONGSON
ADD_1:FFFFFFFF
ADD_2:FFFFFFFF
ADD_3:FFFFFFFF
ADD_4:00000000
RESUL:FFFFFFFD
START INPUTING
ALIENTEK 4.3' TFTLCD
FPGA-A7-PRJ-UDB-V1.0



六、总结感想

1. 通过这次实验，我学会了如何在vivado上创建一个新的项目，学会了如何去调试，如何编译运行，如何做仿真，如何将工程与实验箱联系起来，入门了verilog语言。
2. 了解了加法运算的基本原理，并在上机课上得到了实现，对理论课的知识理解地更加透彻了。
3. 对于vivado的三类基本文件——设计文件、约束文件、仿真文件有了初步的了解，学会了如何在项目中创建这些文件或者是导入这些文件。
4. 了解了外围模块文件的作用，在实验报告前面的整体流程中，adder_display就是起到了一个外围的作用，在内部调用了adder32文件，并直接能够调用函数adder32，类似于C++中的类与对象的原理。
5. 我对vivado的内容有了更深入了解，希望在接下来的实验当中能收获更多知识。