组成原理实验课程第_1_次实验报告

实验名称	加法器改进			班级	李涛
学生姓名	阿斯雅	学号	2210737	指导老师	董前琨
实验地点	实验楼 A 区 308		实验时间	3/21 中午	

1、 实验目的

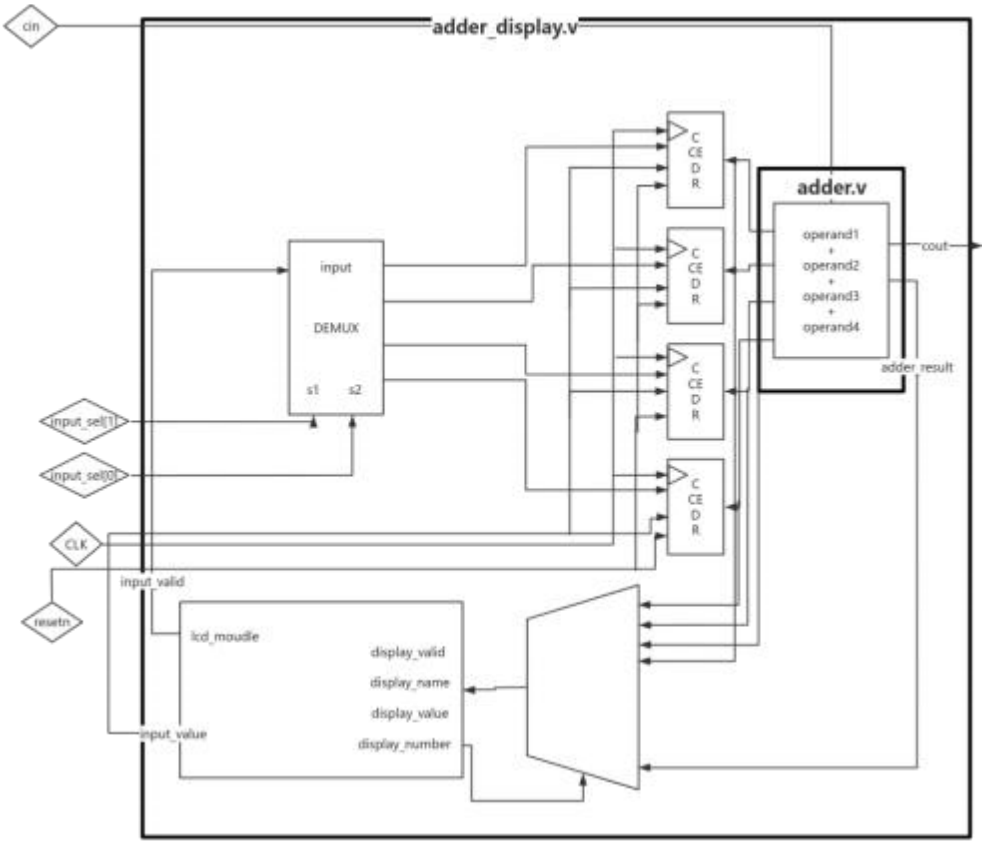
1. 熟悉 LS-CPU-EXB-002 实验箱和软件平台。
2. 掌握利用该实验箱各项功能开发组成原理和体系结构实验的方法。
3. 理解并掌握加法器的原理和设计。
4. 熟悉并运用 verilog 语言进行电路设计。
5. 为后续设计 cpu 的实验打下基础。

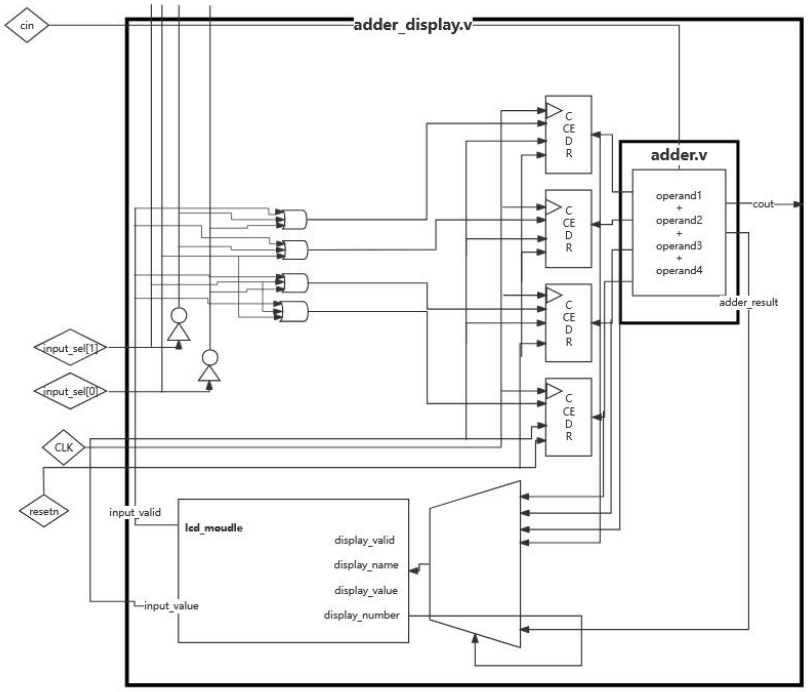
2、 实验内容说明

结合实验指导手册中的实验一完成功能改进， 实现一个能做 4 个 32 位数的加法的加法器。

3、 实验原理图及所说明

3.1、实验图





3.2、说明

从上面实验原理图我们可以知道，四个加数分别对应四个D触发器，而D触发器的输入是我们从触摸屏也就是lcd_moudle模块上输入的input_value值。当我们从触摸屏上点击‘OK’时，input_valid会相应的变成高电平。选择我们是要输入哪一个加数是通过片选信号input_sel来完成的，因为有四个加数，所以要有两个片选信号来表示四种状态，也就

是00,01,10,和11。当我们把两位片选信号和input_valid输入给一个多路分配器时就可以完成把input_valid的高电平根据输入的片选信号的不同输出到不同的位置上，相应的其他三路的输出就会变为0。这样我们就可以实现把input_value给到不同的触发器。而四个触发器的输出是连接到了加法模块上，而在加法模块里，是真正的实现了四个加数和输入进位的加法。当计算完之后，四个加数和最后结果会返回到触摸屏上，根据不同的display_number来到不同格子上来显示。

4、 实验步骤

4.1、修改 adder.v 加法器模块

我们知道在改进之前的加法器是能实现两个 32 位数 op1,op2 和一位进位数 cin 的加法，而改进后我们要实现四个 32 位数和两位进位的加法，所以在改进前的基础上我们要新定义两个32位数op3和 op4，也要把一位进位 cin 改变为两位进位。同时，因为四个数的加法可以产生两位进位， 所以我们也要把输出的进位 cout 改变为两位进位。而最后的结果就为{cout,result}=op1+op2+op3+op4+cin。如下图所示

```
module adder(
    input  [31:0] operand1,
    input  [31:0] operand2,
    input  [31:0] operand3,
    input  [31:0] operand4,
    input   [1:0]      cin,
    output [31:0] result,
    output  [1:0]      cout

    );
    assign {cout,result} = operand1 + operand2 + operand3+ operand4+ cin;

endmodule
```

4.2、 修改 testbench.v 模拟模块

在改进之前，模拟模块中，输入的寄存器只有两个数 op1 和 op2,输入的进位和输出的进位也只有一位，所以在改进之后， 我们要新

定义两个输入寄存器 op3 和 op4，同时也要把 cin 和 cout 改为两位。然后在初始化阶段， 把 op1，op2，op3，op4 和 cin 都设为 0，接着每过 10 个时间单位， 就随机赋值 op1，op2，op3，op4 和 cin，但因为 cin 是两位二进制数， 所以赋值 cin 的时候我们可以把随机值模 4 之后再赋给 cin。如下图所示。

```
module testbench;

    // Inputs
    reg [31:0] operand1;
    reg [31:0] operand2;
    reg [31:0] operand3;
    reg [31:0] operand4;
    reg [1:0]cin;

    // Outputs
    wire [31:0] result;
    wire [1:0]cout;
    // Instantiate the Unit Under Test (UUT)
    adder uut (
        .operand1(operand1),
        .operand2(operand2),
        .operand3(operand3),
        .operand4(operand4),
        .cin(cin),
        .result(result),
        .cout(cout)
    );

    initial begin
        // Initialize Inputs
        operand1 = 0;
        operand2 = 0;
        operand3 = 0;
        operand4 = 0;
        cin = 0;
        // Wait 100 ns for global re
        #100;
        // Add stimulus here
    end
    always #10 operand1 = $random;
    always #10 operand2 = $random;
    always #10 operand3 = $random;
    always #10 operand4 = $random;
    always #10 cin = {$random} % 4;
endmodule
```

4.3、修改 adder_display.v 触摸屏调用模块

在触摸屏调用模块我们首先要改的就是拨码开关，因为在原先的

加法器里面因为只有两个数，所以我们只需要一位的拨码开关就可以表示两种状态，即 0 和 1。但改进之后， 我们需要通过拨码开关来选择四个寄存器的输入。所以我们需要两个拨码开关来表示四种状态， 即 00,01,10 和 11。所以我们要把 input_sel 改成两位的。同样的我们也要把用于输入进位 sw_cin 的拨码开关和用于显示输出进位的 led 灯 led_cout 也要设成两位的。 如下图所示。

```
//拨码开关，用于选择输入
input [1:0]input_sel, //
input [1:0]sw_cin,

//led灯，用于显示cout
output [1:0]led_cout,
```

接着我们需要修改从触摸屏获得输入的代码部分，因为在改进之前我们需要输入的数只有两个，所以我们只需要判断 input_sel 是否等于零即可， 但现在 input_sel 有四种状态， 所以我们需要一个个判断， 即当 input_sel 等于 0 的时候， 我们是要输入 op1,等于 1 的时候是要输入 op2 以此类推。接着我们需要修改触摸屏需要显示的那部分代码，因为在原先修改之前， 我们只需要三个格子就能表示 op1，op2 和 result，但现在我们需要五个格子来显示 op1，op2，op3，op4 和result。所以在 case 语句里面，我们根据不同的 display_number，在不同的格子上显示五个数。 如下图所示。

```
    always @(posedge clk)
    begin
        if (!resetn)
        begin
            adder_operand1 <= 32'd0;
        end
        else if (input_valid && input_sel==0)
        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==1)
        begin
            adder_operand2 <= input_value;
        end
    end
```

```
always @(posedge clk)
  begin
      if (!resetn)
      begin
          adder_operand3 <= 32'd0;
      end
      else if (input_valid && input_sel==2)
      begin
          adder_operand3 <= input_value;
      end
  end

  //op4
   always @(posedge clk)
     begin
         if (!resetn)
         begin
             adder_operand4 <= 32'd0;
         end
         else if (input_valid && input_sel==3)
         begin
             adder_operand4 <= input_value;
         end
     end
```

```
always @(posedge clk)
begin
    case(display_number)
        6'd1 :
        begin
            display_valid <= 1'b1;
            display_name  <= "ADD_1";
            display_value <= adder_operand1;
        end
        6'd2 :
        begin
            display_valid <= 1'b1;
            display_name  <= "ADD_2";
            display_value <= adder_operand2;
        end
        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
          6'd5 :
         begin
             display_valid <= 1'b1;
             display_name  <= "RESULT";
             display_value <= adder_result;
          end


        default :
        begin
            display_valid <= 1'b0;
            display_name  <= 40'd0;
            display_value <= 32'd0;
        end
    endcase
end
```

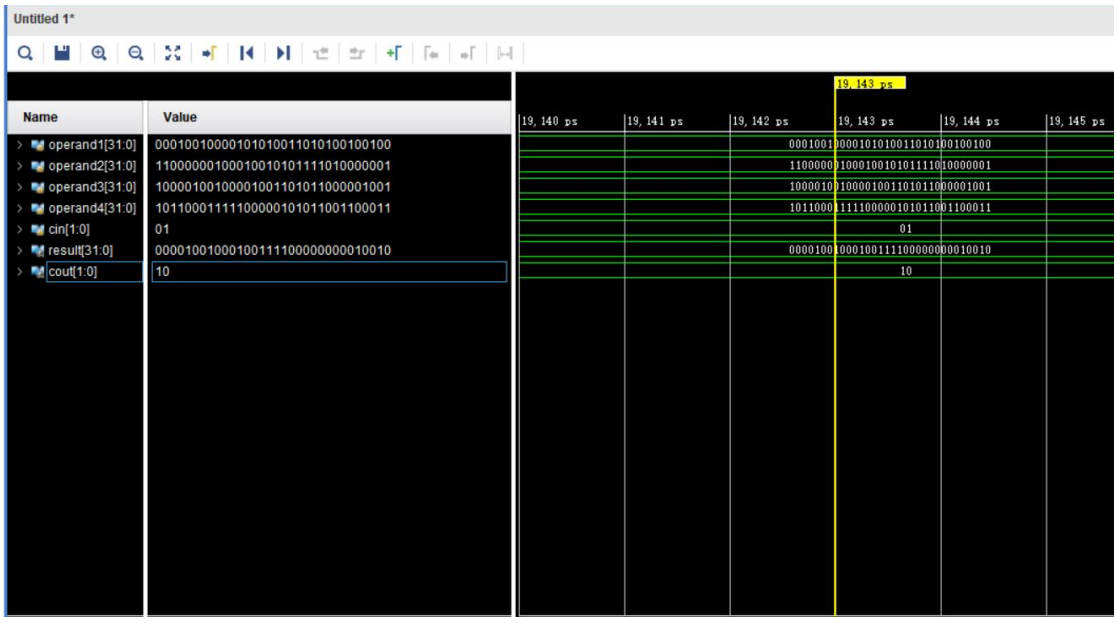
4.4、修改约束文件

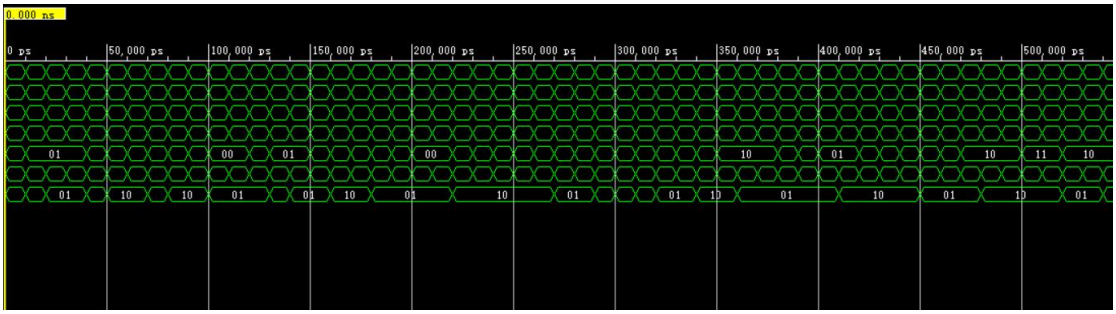
约束文件就是将顶层模块的输入输出端口与 FPGA 板上的 IO 接口引脚绑定。但因为原先 input_sel，sw_cin，led_cout 只需要一位，而现在每个都需要两位，所以我们要在原来的单个拨码开关的绑定的基础上新增几个拨码开关的绑定，但要注意的是我们要以数组的形式表示不同的拨码开关，如 input_sel[0]和 input_sel[1]。我们可以根据引脚对应关系，新增几个不同的拨码开关。 最后我们也不能忘了给 I/O 端口统一设置为 LVCMOS33。如下图所示。

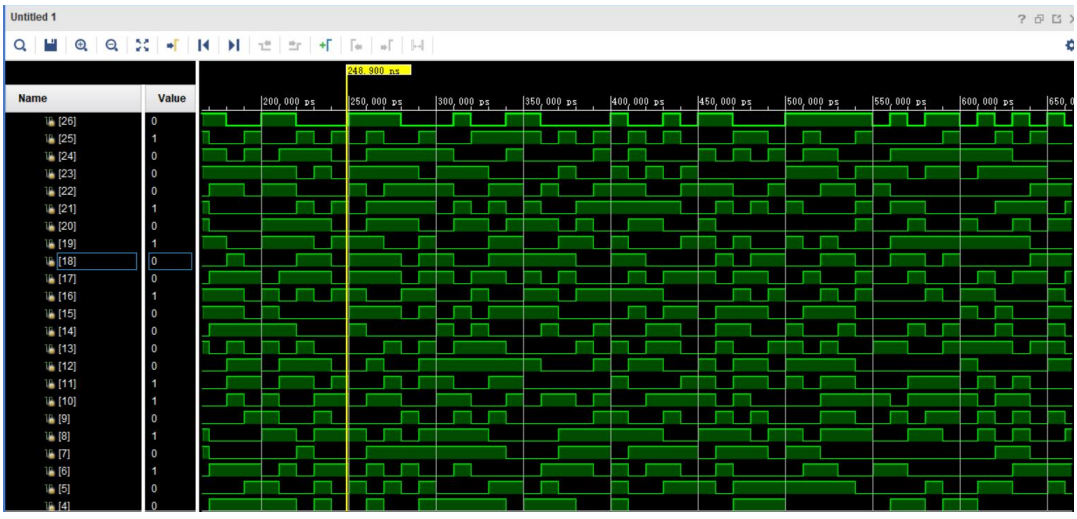
```
set_property PACKAGE_PIN AC19 [get_ports clk]
set_property PACKAGE_PIN H7   [get_ports {led_cout[0]}]
set_property PACKAGE_PIN AC21 [get_ports {input_sel[0]}]
set_property PACKAGE_PIN AD24 [get_ports {sw_cin[0]}]
set_property PACKAGE_PIN Y3   [get_ports resetn]
set_property PACKAGE_PIN D5   [get_ports {led_cout[1]}]
set_property PACKAGE_PIN AC22 [get_ports {input_sel[1]}]
set_property PACKAGE_PIN AC23 [get_ports {sw_cin[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports {led_cout[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports  {input_sel[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw_cin[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports resetn]
set_property IOSTANDARD LVCMOS33 [get_ports {led_cout[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {input_sel[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw_cin[1]}]
```

5、 实验结果分析

5.1、仿真结果

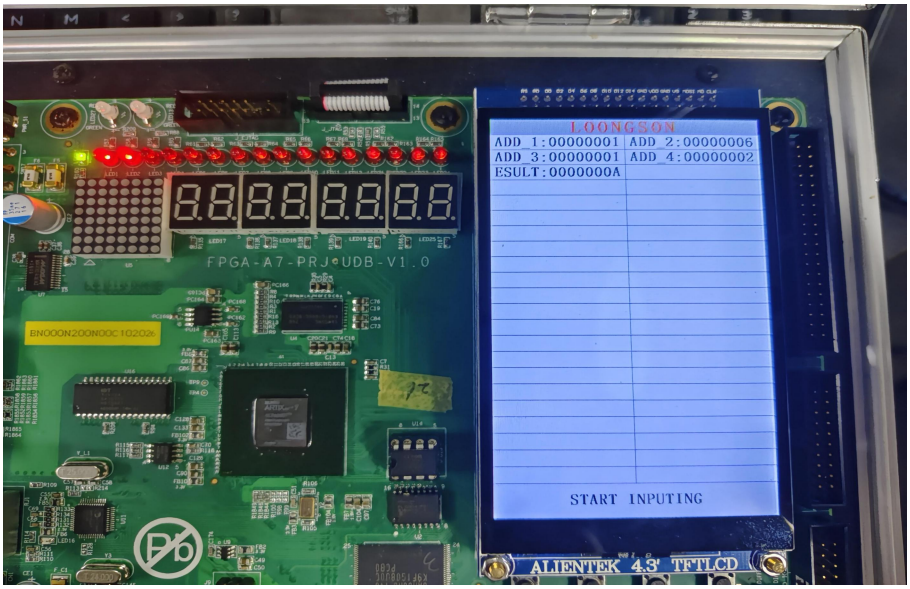




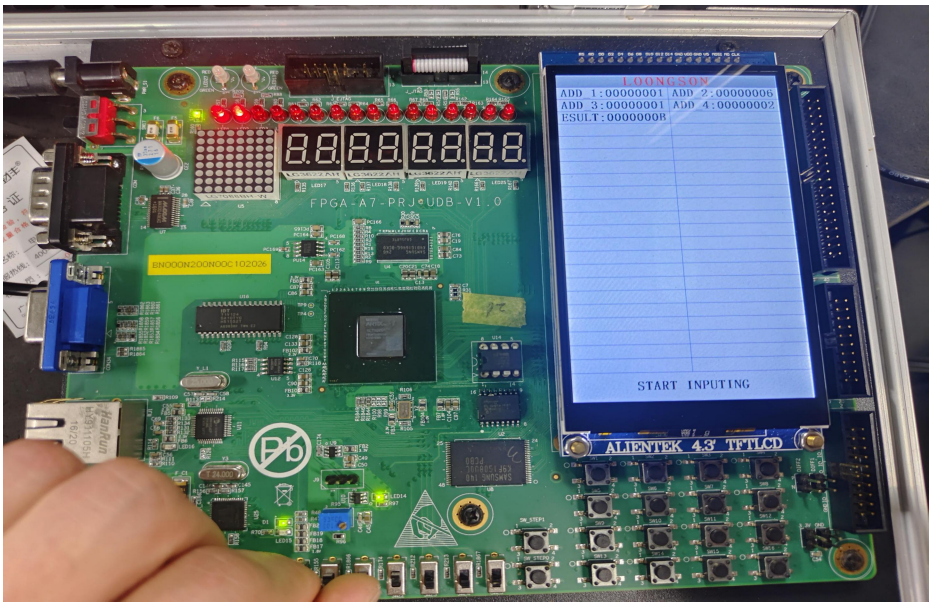


我们可以从波形图中看出每过一万个皮秒，也就是十纳秒，每一位就会改变一次值。这也符合我们在testbench模块里面写的代码，每过10个时钟单元，程序就会随机初始化四个加数和进位。

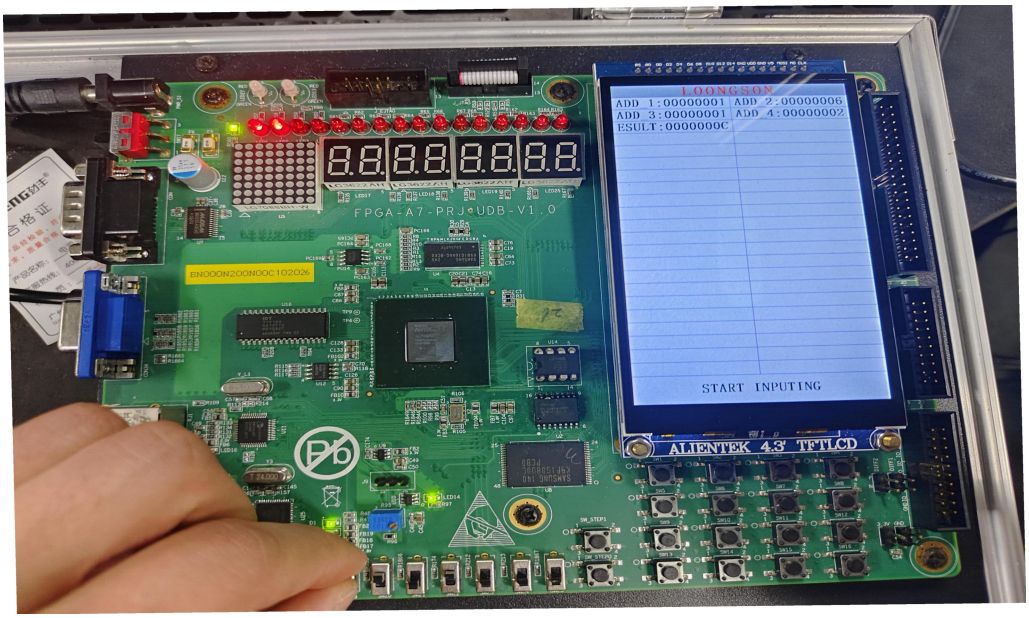
5.2、实验箱结果



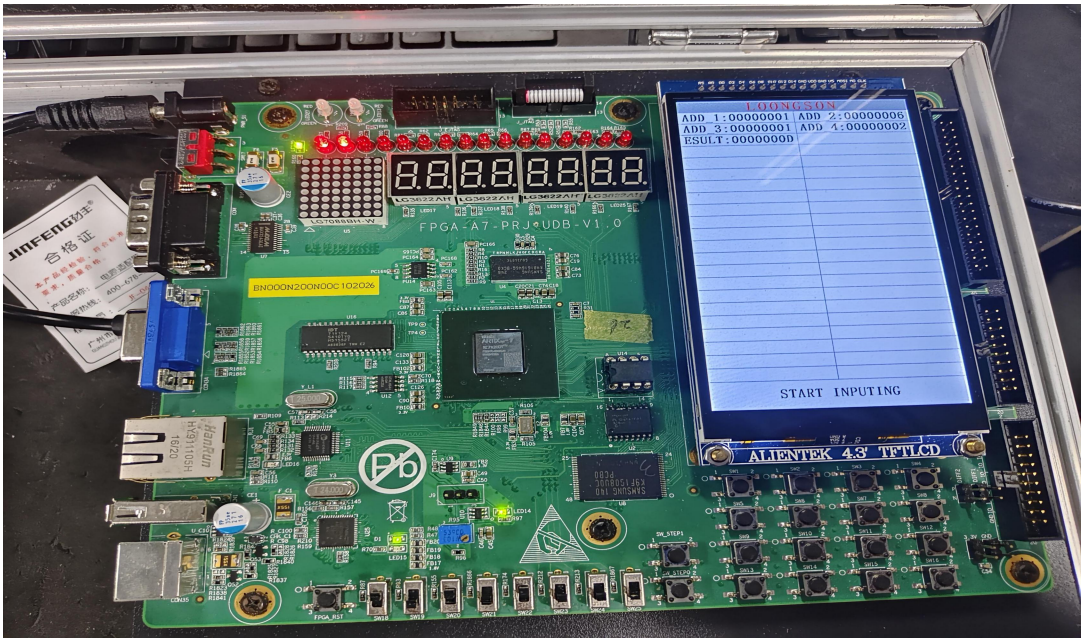
（cin==0）



（cin==1）



（cin==2）



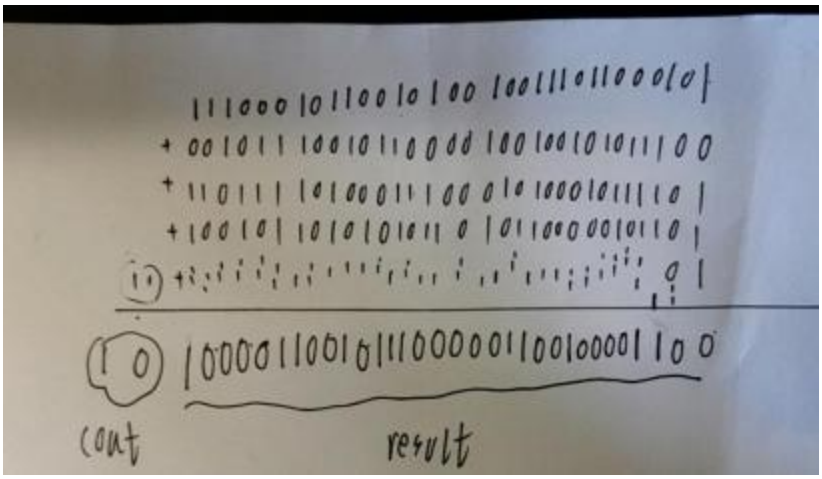
（cin==3）

随后我在实验箱烧录了程序，从触摸屏上输入了四个加数分别为：1，1，2和6。而进位则通过拨码开关来控制。当分别控制进位为0,1,2,和3的时候，我们可以通过显示屏上的显示可以验证结果是否正确。

5.3、结果分析

在仿真模拟中，从上到下的七个数分别为 op1，op2，op3，op4，

cin，result 和 cout。我们可以根据上面五个数来验证程序的正确性。



可以看出结果和仿真模拟是一样的，所以程序没有问题。

6、 总结感想

通过本次实验， 我首先是接触到了 Verilog 语言， 一开始是不太能看懂的，但慢慢发现只有认真看代码，能看出来有好多地方是跟我们学的高级语言是大差不离的。 然后在老师的帮助和查阅资料后， 成功的读懂并修改了本次实验代码。

其次， 我知道了该怎么把代码烧录到实验箱上运行，原来是先是通过一个约束文件， 把 I/O 端口和实验箱上的开关绑定在一起，然后再把代码文件转换成二进制文件再用 usb 接口烧录到实验箱上。

但我觉得最重要的是，通过本次实验， 锻炼了自己的实操能力。因为修改代码和烧录代码是我们自己独立完成的， 在此过程中我们遇到了许多困难，但我们老师不在身边的情况下，我们同学之间

通过相互帮助， 即使所用时间很长，但最终成功的完成了任务。我也相信， 在接下来的实验实操中，我们肯定会越来越得心应手，很出色的完成任务。