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			班级	李涛
学生姓名	胡博浩	学号	2212998	指导老师	董前琨
实验地点	津南实验楼 A308		实验时间	2024.3.21	

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

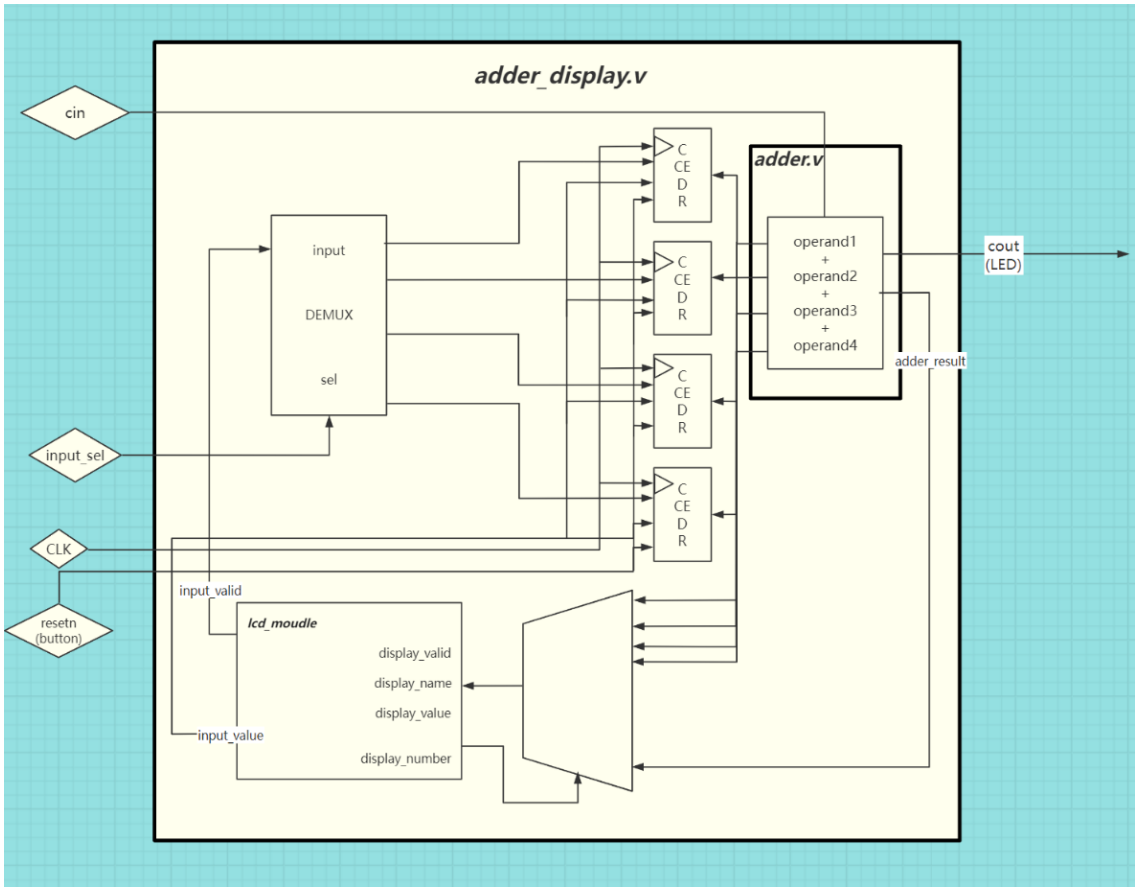
请结合实验指导手册中的实验一（加法器实验）完成功能改进，实现一个能做 4 个 32 位数的加法的加法器，注意以下几点：

（1）除了修改 adder 模块，display、testbench 和约束文件都有修改，注意加法器进位，四个数加法会出现 2 位的进位。

（2）实验报告中需要补充原理图，并对原理图进行解释说明。原理图参照图 2.40 进行修改，建议使用 visio 画图（别的画图软件也可，不会画图的可以手绘然后照片放报告里面）。

（3）实验报告中需要有仿真结果（波形截图），并针对图中的数据解释说明（某一时刻哪些数相加，进位情况如何，结果如何，是否验证的模块的正确性），还需要有实验箱上箱验证的照片，同样，针对照片中的数据也需要解释说明。

3、 实验原理图



原理图展示了一个模块化的设计，主模块为 adder_display，它调用了两个子模块：adder 用于加法运算，lcd_module 用于显示。在 adder_display 模块的输入端，cin 是加法器的进位输入，而 input_sel 是一个选择器信号，用于选择输入到加法器的操作数。这些输入信号通过拨码开关进行输入，允许用户手动设置操作数。加法器的输出端连接到了一个名为 cout 的信号线上，代表加法运算的进位输出。此外，cout 还连接到了一个 LED 灯上，用于显示加法运算的进位结果。当进位发生时，LED 灯会亮起，提供直观的视觉反馈。此外，图中还包含了一个 DEMUX 模块，它是一个数据选择器，用于根据 sel 信号选择并传递不同的操作数到加法器。通过调整 sel 信号的值，可以选择 operand1、operand2、operand3 或 operand4 中的任意一个作为加法器的输入。

综上所述，通过调用子模块、设置输入信号和连接显示器，该设计能够实现加法运算结果的显示功能。同时，通过拨码开关和选择器信号，用户可以灵活地选择和配置输入操作数以及观察运算结果。

4、 实验步骤

（1）加法器 adder32.v

```
module adder32(
    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
```

a) 代码修改：添加两个加数(operand3,operand4)，将进位输入(cin)和进位输出(cout)修改为 2 位，求和计算也相应改变。

b) 功能：计算四个 32 位输入与一个进位输入的和，并输出求和结果以及进位输出。

（2）加法器展示模块 adder_display.v

```
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 [1:0]input_sel,
    //0:输入为加数 1(add_operand1);1:输入为加数 2(add_operand2);2:输入为加数 3(add_operand3);1:输入为加数 4(add_operand4);
    input [1:0]sw_cin,
    //led 灯，用于显示 cout
    output [1:0] 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;
        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;
        adder32 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;
//-----{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==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
        //当 input_sel 为 2 时，表示输入数为加数 3，即 operand3
        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
            //当 input_sel 为 3 时，表示输入数为加数 4，即 operand4
            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
//-----{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'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    <= "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
```

a) 代码修改：修改部分用红色标注。大致分为四部分

①选择信号、**sw_cin**、**led_out**：均修改为两位

②调用加法模块的部分：匹配加法器的修改

③从触摸屏获取输入：添加加数 **3** 和 **4** 的部分

④输出到触摸屏显示：**1,2,3,4** 为加数、**5** 为结果 **result**

b) 功能：根据选择信号，输入不同的加数，并调用加法模块得到结果，同时实时显示各个数。

（3）仿真 tb.v

```
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)
    adder32 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 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 operand3 = $random;
    always #10 operand4 = $random;
    always #10 cin = {$random} % 4; //加了拼接符，{$random}产生一个非负数，除3取余得到0或1或2或3
endmodule
```

a) 代码修改：添加了两个寄存器（operand3,operand4），将进位输入和输出修改为两位，添加了两个随机数，并修改 cin 的生成范围。

b) 功能：可以实现四个 32 为数的加法器实验的仿真模拟

（4）约束文件 mycons.xdc

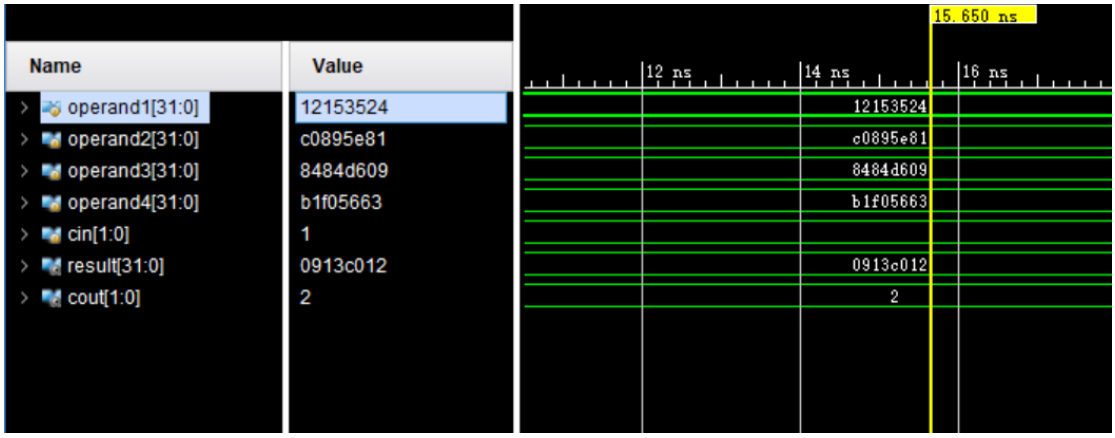
```
set_property PACKAGE_PIN AC19 [get_ports clk]
set_property PACKAGE_PIN H7   [get_ports led_cout[1]]
set_property PACKAGE_PIN D5   [get_ports led_cout[0]]

set_property PACKAGE_PIN Y3   [get_ports resetn]
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 sw_cin[1]]
set_property PACKAGE_PIN AC23 [get_ports sw_cin[0]]
```

a） 代码修改：led 输出绑定两个，选择信号匹配两个拨码开关，进位输入用两个拨码开关实现。

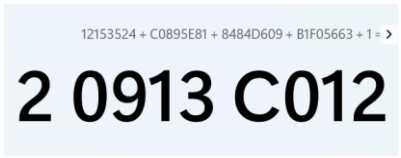
b） 功能：实现了实验箱上 led 灯、拨码开关、触摸屏的连接，程序因而可以在实验箱上运行。

5、 实验结果分析

（1）仿真实验



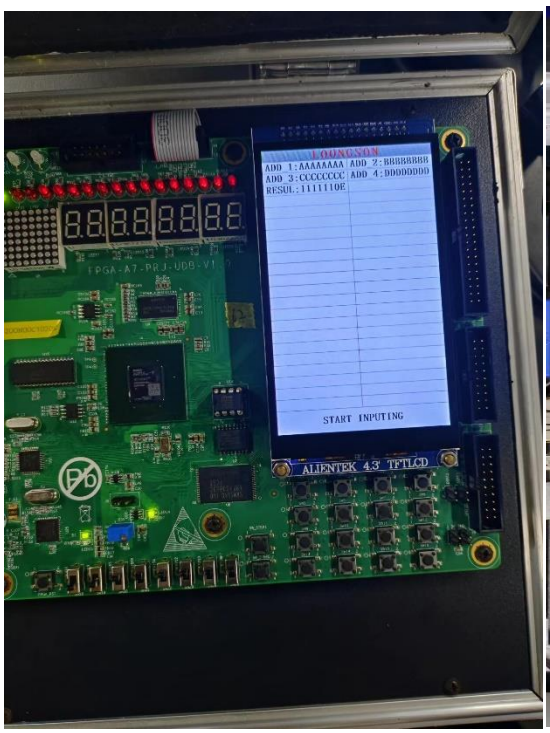
0x12153524+0xc0895e81+0x8484d609+b1f05663+1=0x0913c012(进位为 2)



计算正确，说明仿真模拟实验成功。

（2）上箱实验

说明输入没有问题



加数为 AAAAAAAA, BBBBBBBB, CCCCCCCC, DDDDDDDD, 进位输入分别为 0, 1, 2, 3

显然结果无误，实验成功

6、 总结感想

这次实验让我深入了解了在 CPU 内部运算中的一些重要操作，特别是对加法器进行修改和调整。通过这个过程，我进一步掌握了使用 Verilog 语言进行硬件描述和编程的基本技能。在调整显示屏显示位置的过程中，我遇到了一些问题，但通过仔细研究代码并修正错误，最终成功解决了问题。这次实验也让我更加熟悉了实验箱的布局结构和使用方法，

提高了我的实验技能。

总的来说，这次实验对我来说是一次很好的学习和实践机会，让我在 Verilog 语言的应用和硬件实验方面都有了进一步的提升。我相信这些经验和技能将对我的未来学习和工作有很大的帮助。