# 组成原理课程第一次实报告

# 实验名称：数据运算--补码减法

学号：2310422　姓名：谢小珂　　班次：　1078

## 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、减法器与补码的原理。
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。
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，见图 2.1。

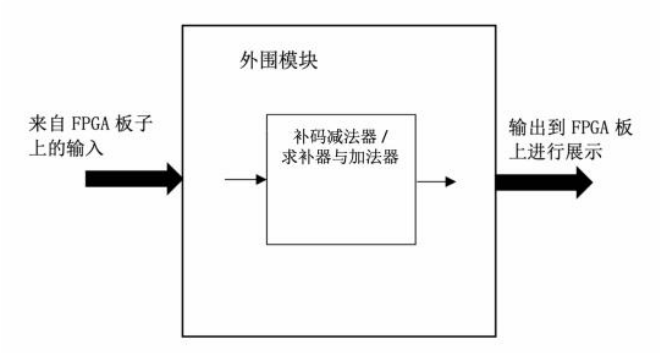
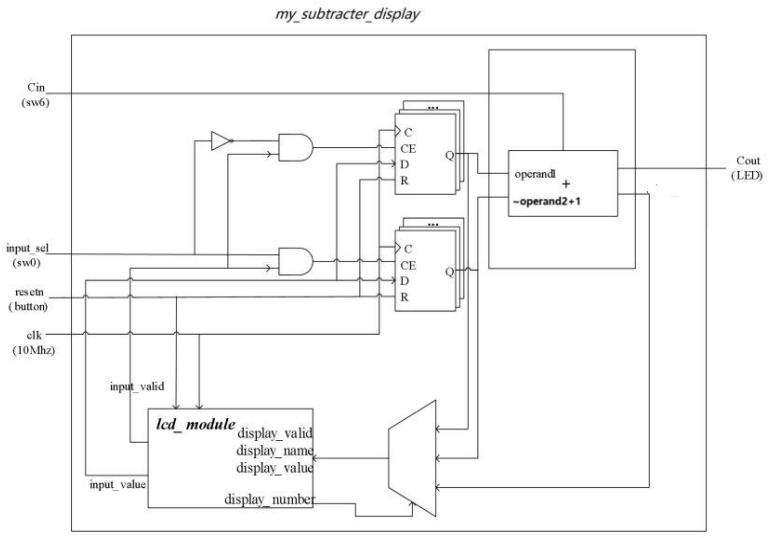
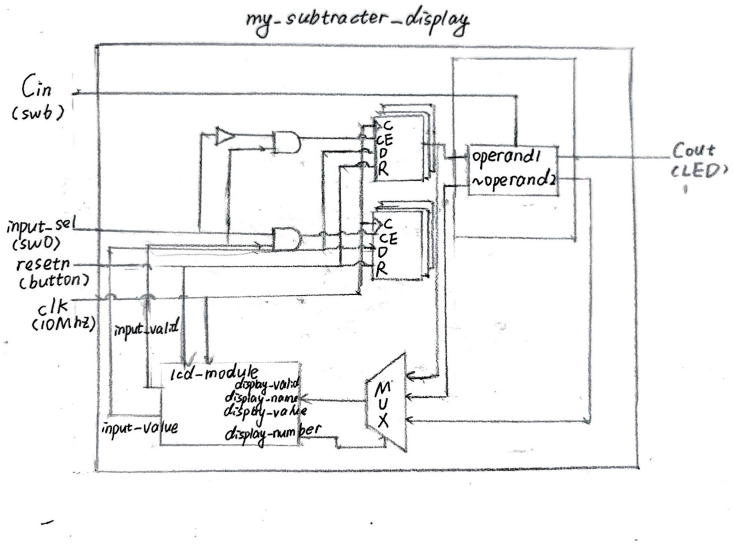



图 2.1 定点减法设计实验的顶层模块大致框图

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




1. 补码减法是通过将减法操作转换为加法操作来实现的。具体来说，op1 - op2 可以转换为 op1 + (-op2)，其中-op2 是 op2 的补码。这种转换简化了硬件设计，因为只需要实现加法器即可。

2. LCD 模块位于左下角，用于显示结果。display_number 是一个多位的二进制数，用于控制多路选择器（MUX）。Input_valid 变量控制使能端（CE），在 Verilog 代码中通过 always 语句进行判断。

3. Cin、input_sel、resetn、clk、input_valid 和 input_value 是输入信号。Cout 是输出信号，通过 LED 灯显示。display_valid、display_name、display_value 和 display_number 是显示屏上显示的值。

5. display_name 和 display_value 是显示屏上实际显示的值。

## 四、实验步骤

### （一）设计减法器

·编写 my_subtractor 文件

```
module my_subtracter(
        input[31:0] A,//被减数
        input[31:0] B,//减数
        input cin,//进位输入
        output cout,//进位输出
        output[31:0] result//差
        );

        wire [31:0] B_complement;//B 的补码
        wire [32:0] Sum;//临时存储的加法结果（包括进位）

        //计算 B 的补码，取反加一
        assign B_complement=~B+1;

        //使用加法器计算 A+（-B)+cin
        // 将 A 和 B_complement 扩展为 33 位，避免位宽不匹配
        assign Sum = {1'b0, A} + {1'b0, B_complement} + {32'b0, cin};

        //输出结果和进位
        assign result=Sum[31:0];
        assign cout=Sum[32];

endmodule
```

1.计算减数 B 的补码 B_complement = ~B + 1，相当于-B。

2.将被减数 A、B 的补码和进位输入 cin 相加，得到 A +（- B） + cin 的结果。

3.结果的低 32 位作为差 result；最高位作为进位输出 cout，表示是否发生借位（A < B 时 cout = 1）。

### （二）仿真验证

·编写 testbench（）文件

```
module testbench();
      reg[31:0] op1;//被减数
      reg[31:0] op2;//减数
      reg cin;//进位输入
      wire[31:0] result;//差
      wire cout;//进位输出

      //实例化被测试模块
      my_subtracter uut(
```

```
            .A(op1),
            .B(op2),
            .cin(cin),
            .cout(cout),
            .result(result)
           );

      initial begin
           //初始化输入信号
        op1 = 32'b0000_0000;
        op2 = 32'b0000_0000;
        cin = 1'b0;
      end

      always #3 op1 = $urandom;
      always #2 op2 = $urandom;
      always #5 cin=~cin;

    // 监视仿真结果
    initial begin
        $monitor("Time: %0t | op1 = %h, op2 = %h, cin = %b | result = %h, cout = %b",
                 $time, op1, op2, cin, result, cout);
    end

endmodule
```

1.op1（被减数）、op2（减数）和 cin（进位输入)初始化为 0。
2.实例化模块:将 op1、op2 和 cin 连接到 my_subtracter 模块的输入，实时计算 result（差）和 cout（进位输出）。
3.使用 $monitor 实时打印仿真时间、输入值（op1、op2、cin）和输出结果（result、cout）。

## （三）实验箱验证

·修改 my_subtracter_display 文件（仅展示修改的代码部分）

```
module my_subtracter_display(

//-----{调用减法模块}begin
    reg  [31:0] sub_operand1;//被减数
    reg  [31:0] sub_operand2;//减数
    wire [31:0] sub_result  ;//差
    wire        sub_borrow;//借位
    my_subtracter subtracter_module(
        .A(sub_operand1),
        .B(sub_operand2),
```

```
        .cin     (sub_cin     ),
        .result  (sub_result  ),
        .cout    (sub_borrow   )
    );

//-----{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
            sub_operand1 <= 32'd0;
        end
        else if (input_valid && !input_sel)
        begin
           sub_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
            sub_operand2 <= 32'd0;
        end
        else if (input_valid && input_sel)
        begin
            sub_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'd1 :
            begin
```

```
                display_valid <= 1'b1;
                display_name  <= "SUB_1";
                display_value <= sub_operand1;
            end
            6'd2 :
            begin
                display_valid <= 1'b1;
                display_name  <= "SUB_2";
                display_value <= sub_operand2;
            end
            6'd3 :
            begin
                display_valid <= 1'b1;
                display_name  <= "RESUL";
                display_value <= sub_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
```

修改说明：

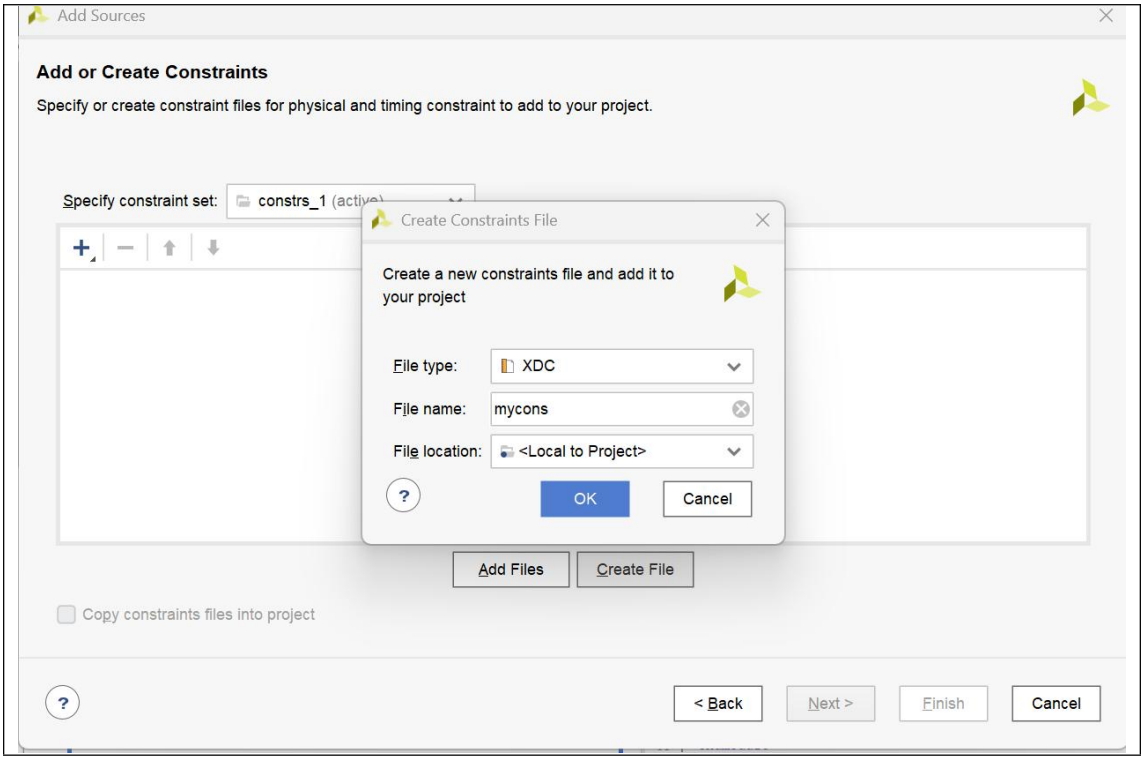
1.减法器模块：将 adder 替换为 subtracter，并调整输入输出信号。

2.输入处理：根据 input_sel 信号选择输入的操作数（被减数或减数）。

3.显示逻辑：将显示名称从 ADD_1 和 ADD_2 改为 SUB_1 和 SUB_2，以反映减法操作。

**·添加约束文件**

在 FPGA 设计中，约束文件（XDC 文件）是上板验证的关键步骤，它定义了外部引脚映射、时序约束以及硬件特性，确保设计能够正确映射到物理硬件上。由于 LCD 触摸屏的引脚绑定是固定的，可以直接使用已有的 XDC 文件，无需重复定义。

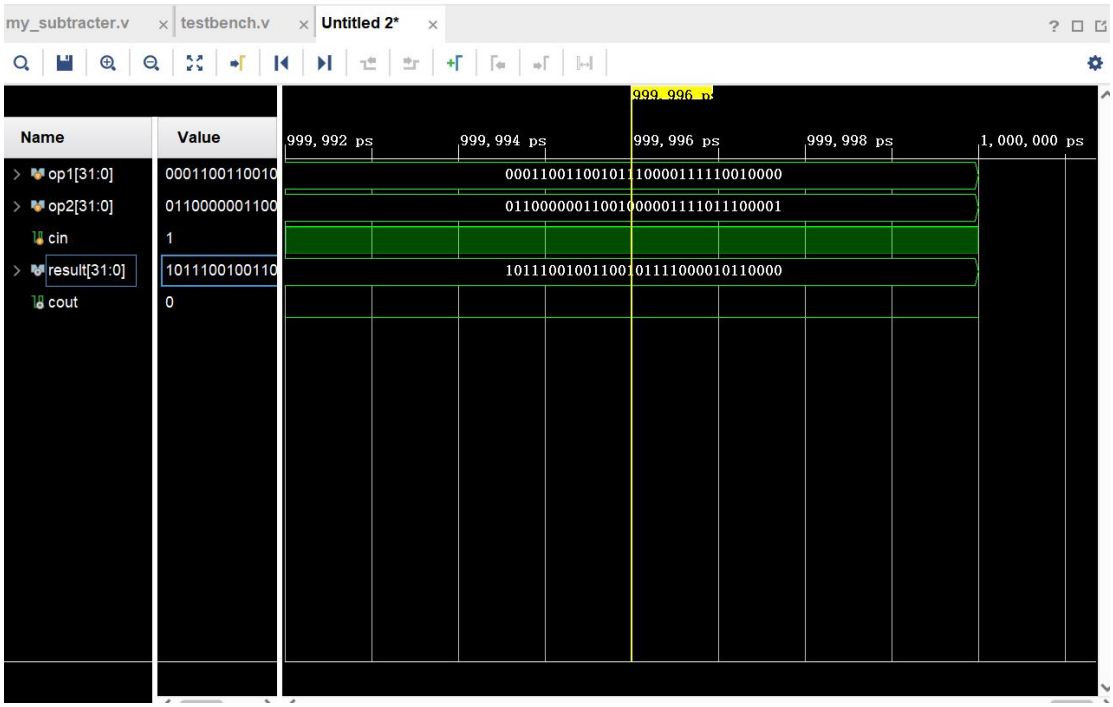



约束文件的作用：

1. 引脚映射：将设计中的信号映射到 FPGA 芯片的具体物理引脚。
2. 时序约束：定义时钟频率、输入输出延迟等，确保设计满足时序要求。
3. 硬件特性配置：设置 I/O 标准、驱动强度、上下拉电阻等硬件特性。

## 五、实验结果分析

・仿真结果 simulation 波形图



以一次结果为例：

被减数 A=0001 1001 1001 0111 0000 1111 1001 0000

减数 B=0110 0000 0110 0100 0001 1110 1110 0001

进位输入 cin=1

差 result=A+~B + 1+cin=1011 1001 0011 0010 1111 0000 1011 0000

进位输出 cout=0

经验证，结果正确。

**·实验箱结果**

**特殊样例：**


图 3.1

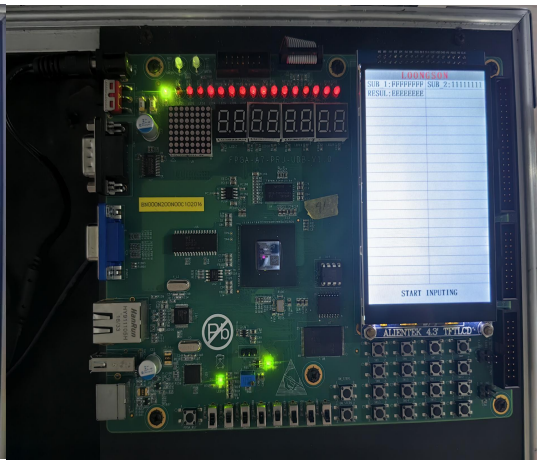



图 3.2

如左图 3. 1 所示：

SUB_1=00000000　SUB_2=11111111　RESUL=EEEEEEEF

**计算过程：**

1>输入值：

A = 32'h00000000 (0)　B = 32'h11111111 (286331153)　cin = 0

2>补码计算：

B_complement = ~B + 1 = 32'hEEEEEEEF (-286331153)

3>加法运算：

Sum = {1'b0, A} + {1'b0, B_complement} + {32'b0, cin}

Sum = 32'h00000000 + 32'hEEEEEEEF + 32'h00000000 = 32'hEEEEEEEF

4>输出结果：

result = 32'hEEEEEEEF

cout = 1'b0 (因为 Sum[32] = 0)

如右图 3. 2 所示

SUB_1=FFFFFFFF　SUB_2=11111111　RESUL=EEEEEEEE

**计算过程：**

1>输入值：

A = 32'hFFFFFFFF　B = 32'h11111111（即 -1）　cin = 0

2>补码运算：

B_complement = ~B + 1 = 32'hEEEEEEEE + 1 = 32'hEEEEEEEF （即 -286331153 的补码表示）

3> 加法运算：

Sum = A + B_complement + cin = 32'hFFFFFFFF + 32'hEEEEEEEF + 32'h00000000

逐位相加：

```
    FFFFFFFF
+ EEEEEEEF
-----------
  1EEEEEEEE
```

结果 Sum 是 33 位的 1EEEEEEEE，其中最高位是进位位。

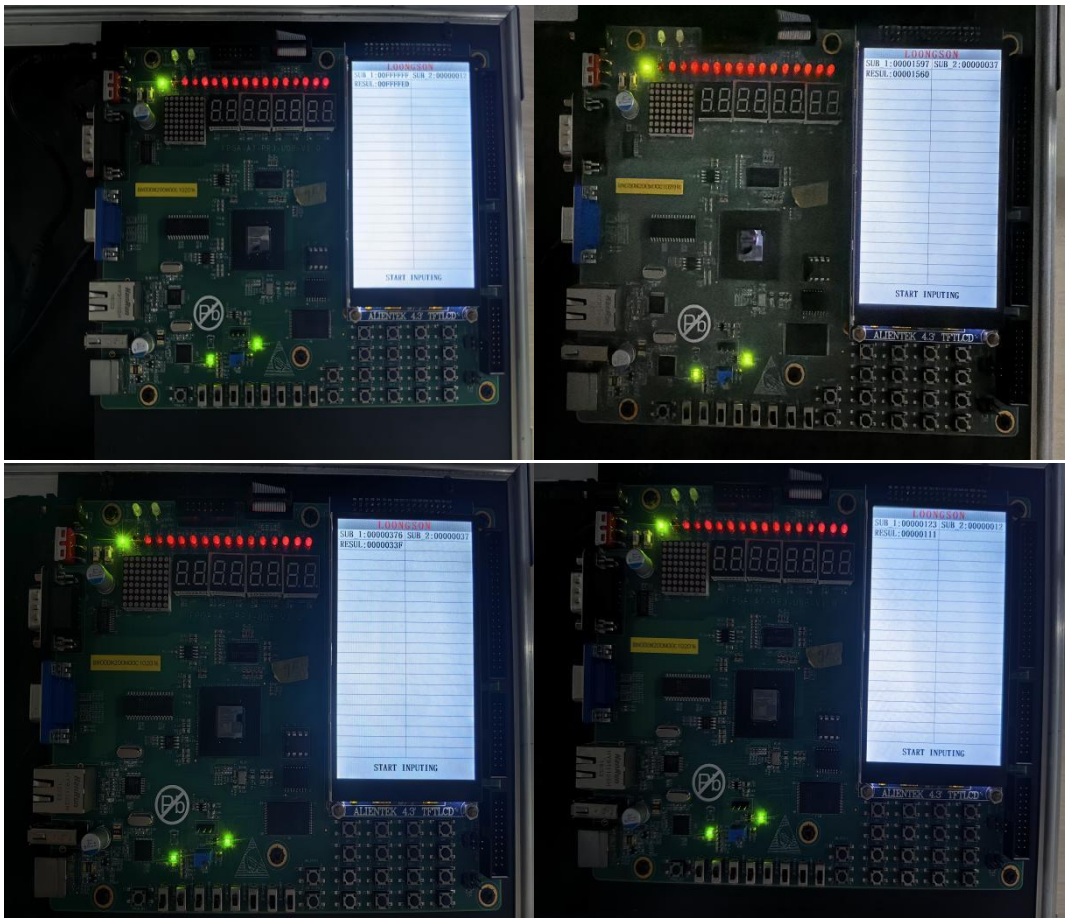
4>输出结果：

result = Sum[31:0] = 32'hEEEEEEEE

cout = Sum[32] = 1'b1 （因为最高位是 1）

**以上样例经验证均正确。**

**其它样例：**




**经验证，均为正确结果。**

## 六、总结感想

1. 本次实验让我更加熟悉了 Verilog 的语法和编程方式，尤其是模块化设计和信号传递的实现。通过编写补码减法器的代码，我掌握了如何利用 Verilog 实现复杂的逻辑运算，并为后续的 CPU 设计实验打下了坚实的基础。

2. 实验过程中，我熟悉了 Vivado 的综合、仿真和 FPGA 部署流程。从代码编写到仿真验证，再到最终的上板调试，我逐步掌握了硬件设计的完整流程。特别是在仿真阶段，通过观察波形图，我能够直观地验证设计的正确性，这对调试和优化代码非常有帮助。

4. 在实验过程中，我遇到了一些问题，例如补码运算的借位信号处理和仿真结果的验证。通过查阅资料，我逐步解决了这些问题，并加深了对计算机加减法运算的理解。

5. 第一次连接实验板并进行调试，虽然过程较为复杂，但最终成功实现功能后，我感到非常有成就感。实验板的调试让我更加直观地理解了硬件设计的实际应用，也让我意识到硬件调试需要耐心和细致。