组成原理课程第__一__次实验报告

实验名称：定点加法

学号：__2012679__ 姓名：__王娇妹__ 班次：__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

复现两个 32 位数的定点加法实验，并根据老师要求进行改进。

改进一：在原始 32 位加法器的基础上，实现 32 位补码减法器。

1、 输入的两个操作数各自为补码，输出的差也是补码。

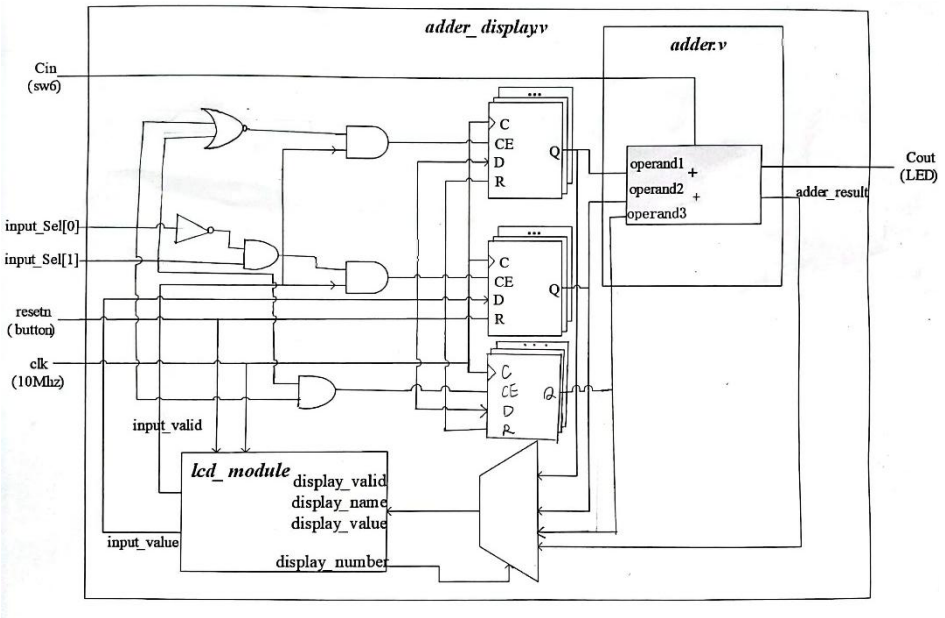
2、 cout 不再是进位标记，而是减法溢出标记，cin 信号不用。

3、 波形仿真。

改进二：在原始 32 位加法器的基础上，实现三个 32 位数的加法器。

1、 波形仿真。

三、实验原理图



三位数的加法，设置当 input_sel 为两位。

当 input_sel 为 00 时，!input_sel[0] && !input_sel[1]表示输入数为加数 1，即 operand1

当 input_sel 为 10 时，!input_sel[0] && input_sel[1]表示输入数为加数 2，即 operand2

当 input_sel 为 11 时，input_sel[0] && input_sel[1]表示输入数为加数 3，即 operand3

四、实验步骤

改进一：补码减法器

```
module adder(
    input   [31:0] operand1,
    input   [31:0] operand2,
```

```
    output [31:0] result,
    output          cout
    );
    assign result = operand1 + ~operand2 + 1 ;
    assign cout =
(~result[31]&operand1[31]&~operand2[31])|(result[31]&~operand1[31]&operand2[31]);
  endmodule
```

补码的运算规则是，$[x]_{补} - [y]_{补} = [x]_{补} + [-y]_{补}$，可以将补码减法转换为补码加法。

$[-y]_{补} = [y]_{补} + 2^{-n}$ ，即$[-y]_{补}$等于$[y]_{补}$取反，末位加 1。

所以$[x]_{补} - [y]_{补} = [x]_{补} + \sim[y]_{补} + 1$

当运算结果超出机器字长所能表示的数值范围，这种情况叫做溢出。

补码加减法时，将符号位和数值部分一起参加运算。

operand1[31]、operand2[31]分别是两个操作数的符号位，result[31]为运算结果符号位。

当 operand1[31] = 1，operand2[31] = 0（正数减负数）,而 result[31] = 1（结果为负）时,发生负溢出；

当 operand1[31] = 0，operand2[31] = 1（负数减正数）,而 result[31] = 0（结果为正）时，发生正溢出。

改进二：三位数加法器

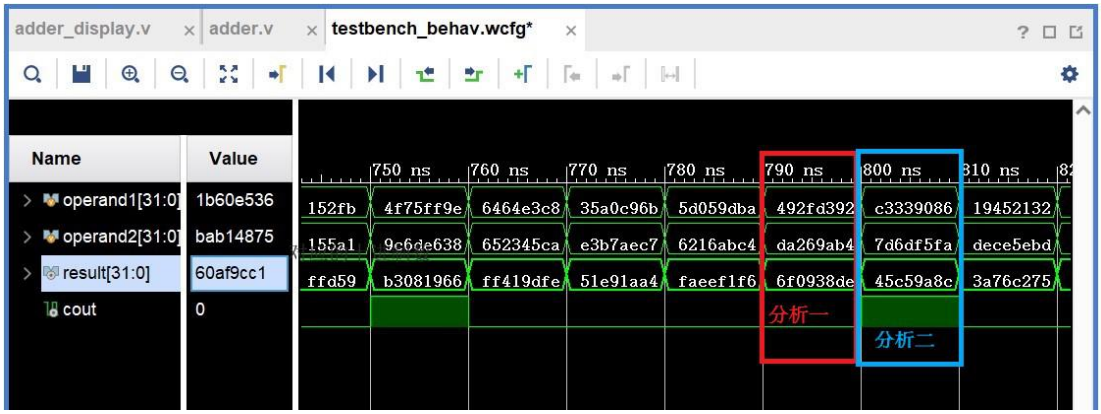
```
    module adder(
        input   [31:0] operand1,
        input   [31:0] operand2,
        input   [31:0] operand3,
        input          cin,
        output [31:0] result,
        output         cout
        );
        assign {cout,result} = operand1 + operand2 + operand3 + cin;
      endmodule

//修改，当 input_sel 为 00 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] && !input_sel[1]))
    begin
        adder_operand1 <= input_value;
    end
end
//修改，当 input_sel 为 10 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[0] && input_sel[1]))
    begin
        adder_operand2 <= input_value;
    end
end
//修改，当 input_sel 为 11 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[0] && input_sel[1]))
    begin
        adder_operand3 <= input_value;
    end
end
```

五、实验结果分析

改进一：补码减法器



分析一（无溢出）：

输入：

operand1 = 492fd392

二进制数 0100 1001 0010 1111 1101 0011 1001 0010，正数对应的十进制数为 1227871122。

operand2 = da269ab4

二进制数 1010 0101 1101 1001 0110 0101 0100 1100，负数对应的十进制数为-635004236。

输出：

result = 6f0938de

二进制数 0110 1111 0000 1001 0011 1000 1101 1110，正数

对应的十进制数为 1862875358。

cout = 0，说明没有溢出

1227871122 - （-635004236） = 1862875358，没有溢出，仿真结果正确。

分析二（有溢出）：

输入：

operand1 = c3339086

二进制：1011 1100 1100 1100 0110 1111 0111 1010，负数

对应的十进制数-1020030842

operand2 = 7d6df5fa

二进制数 0111 1101 0110 1101 1111 0101 1111 1010，正数

对应的十进制数 2104358394。

输出：

result = 45c59a8c

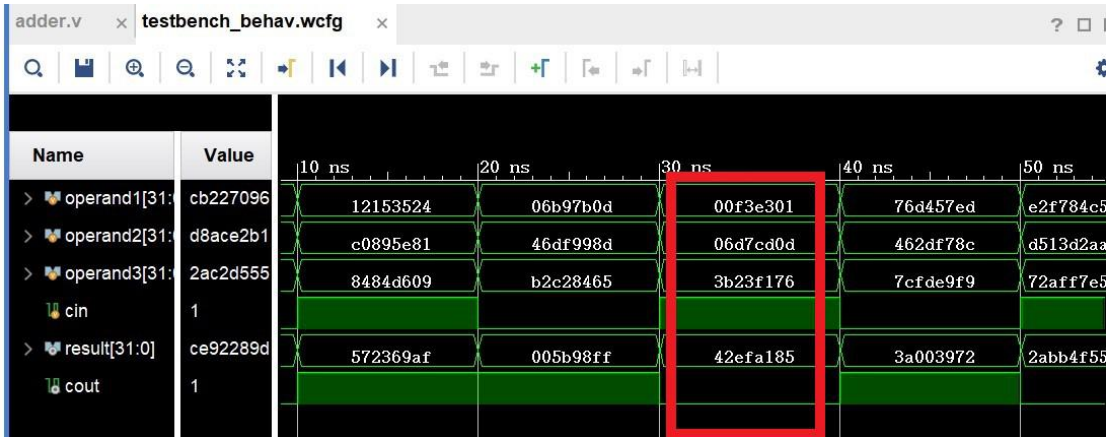
正数，二进制数 0100 0101 1100 0101 1001 1010 1000 1100

对应的十进制数 1170578060。

cout = 1，说明有溢出

负数减正数，结果却为正数，说明发生了溢出，cout 为 1，说明仿真结果正确。

改进二：三位数加法器



输入：

operand1 = 00f3e301

operand2 = 06d7cd0d

operand3 = 3b23f176

cin = 1

输出：

result = 42efa185，cout = 0

00f3e301 + 06d7cd0d + 3b23f176 + 1 = 42efa185，说明仿真结果正确。

六、总结感想

在进行三位数加法的改进测试时，我遇到了无论操作数是多少，仿真时都没有进位的问题。经过检查代码发现是因为在“assign {cout,result} = operand1 + operand2 + operand3 + cin;”这一步多加了一对大括号将等号右边括了起来，导致 cout 一直为零。

通过此次实验，我对补码加减法的运算规则、溢出判断等有了更深刻的理解。