组成原理课程第 一 次实报告

实验名称：加法器改进实现 4 个 32 位数字相加

学号： 2213041 姓名： 李雅帆 班次： 李涛老师

一、实验目的

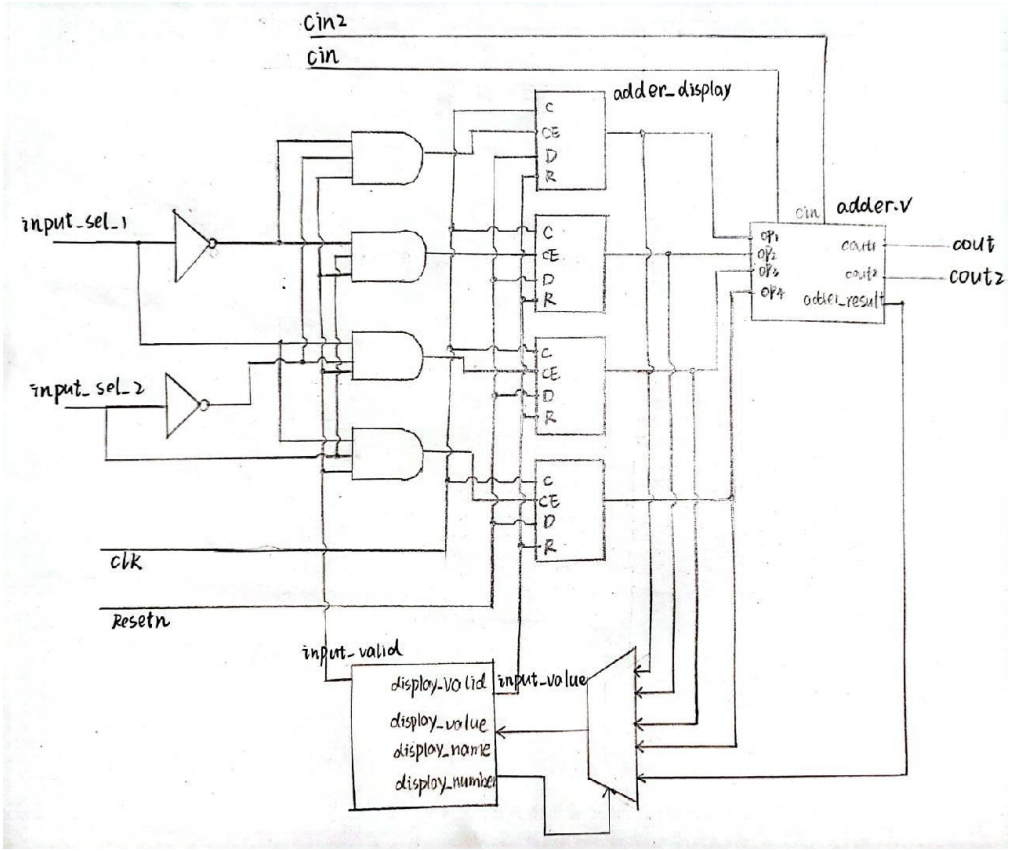
1. 熟悉 LS-CPU-EXB-002 实验箱和软件平台。
2. 掌握利用该实验箱各项功能开发组成原理和体系结构实验的方法。
3. 理解并掌握加法器的原理和设计。
4. 熟悉并运用 verilog 语言进行电路设计。
5. 为后续设计 cpu 的实验打下基础。

二、实验内容说明

在两个 32 位加法器的基础上进行功能改进，修改各个模块，用四个 32 位寄存器实现加法运算，对修改后的代码进行仿真，得到正确的波形图，并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

用四个加法器和选择器进行实现，输入输出已在图中各个端口标出。



四、实验步骤

1.在 **adder3.v** 中将加法器拓展到四个输入。

```
input    [31:0] operand1,
input    [31:0] operand2,
   input    [31:0] operand3,
   input    [31:0] operand4,
   input                cin,
   input                cin_2,
   output [31:0] result,
   output               cout,
   output               cout_2
      );
assign {cout,cout_2,result} = operand1 + operand2 + operand3 + operand4 + cin + cin_2;
```

2.在 **adder_display.v** 中将输入输出乘 2，寄存器设置为 4 个。将对应的输入输出端口记录好，和 adder 顺序对应，四个输入用两个拨码开关的二进制进行表示。

```
     input input_sel,
    input input_sel_2,
    input sw_cin,
    input sw_cin_2,
    output led_cout,
    output led_cout_2,

    reg    [31:0] adder_operand1;
    reg    [31:0] adder_operand2;
    reg    [31:0] adder_operand3;
    reg    [31:0] adder_operand4;
    wire               adder_cin;
    wire               adder_cin_2;
    wire [31:0] adder_result    ;
    wire               adder_cout;
     wire               adder_cout_2;
    adder adder_module(
          .operand1(adder_operand1),
          .operand2(adder_operand2),
          .operand3(adder_operand3),
          .operand4(adder_operand4),
          .cin         (adder_cin         ),
          .cin_2       (adder_cin_2       ),
          .result    (adder_result    ),
          .cout        (adder_cout        ),
          .cout_2      (adder_cout_2      )
```

```
);
assign adder_cin = sw_cin;
assign adder_cin_2 = sw_cin_2;
assign led_cout    = adder_cout;
 assign led_cout_2    = adder_cout_2;
always @(posedge clk)
begin
      if (!resetn)
      begin
            adder_operand1 <= 32'd0;
      end
      else if (input_valid && !input_sel && !input_sel_2)//1
      begin
            adder_operand1 <= input_value;
      end
end

//当 input_sel 为 1,input_sel_2=0 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 && !input_sel_2) //2
      begin
            adder_operand2 <= input_value;
      end
end
//当 input_sel 为 0,input_sel_2=1 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 && input_sel_2) //3
      begin
            adder_operand3 <= input_value;
      end
end
  //当 input_sel 为 1,input_sel_2=1 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 && input_sel_2) //4
    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  <= "RESUL";
            display_value <= adder_result;
        end
        default :
```

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

3.在 **mycons.xdc** 中，对照表格，将每一个端口分配一个拨码开关，包括新增端口。

```
set_property PACKAGE_PIN AC19 [get_ports clk]
set_property PACKAGE_PIN H7      [get_ports led_cout]
set_property PACKAGE_PIN D5      [get_ports led_cout_2]
set_property PACKAGE_PIN Y3      [get_ports resetn]
set_property PACKAGE_PIN AC21 [get_ports input_sel]
set_property PACKAGE_PIN AC22 [get_ports input_sel_2]
set_property PACKAGE_PIN AC23 [get_ports sw_cin]
set_property PACKAGE_PIN AD24 [get_ports sw_cin_2]

set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports led_cout]
set_property IOSTANDARD LVCMOS33 [get_ports led_cout_2]
set_property IOSTANDARD LVCMOS33 [get_ports resetn]
set_property IOSTANDARD LVCMOS33 [get_ports input_sel]
set_property IOSTANDARD LVCMOS33 [get_ports input_sel_2]
set_property IOSTANDARD LVCMOS33 [get_ports sw_cin]
set_property IOSTANDARD LVCMOS33 [get_ports sw_cin_2]
```

4.在 **testbenc.v** 中，对每个数进行刷新，对应端口。

```
    // Inputs
    reg [31:0] operand1;
    reg [31:0] operand2;
    reg [31:0] operand3;
    reg [31:0] operand4;
    reg cin;
    reg cin_2;
    // Outputs
    wire [31:0] result;
```

```
    wire cout;
    wire cout_2;
    // Instantiate the Unit Under Test (UUT)
    adder uut (
        .operand1(operand1),
        .operand2(operand2),
        .operand3(operand3),
        .operand4(operand4),
        .cin(cin),
        .cin_2(cin_2),
        .result(result),
        .cout(cout),
        .cout_2(cout_2)
    );
    initial begin
        // Initialize Inputs
        operand1 = 0;
        operand2 = 0;
        operand3 = 0;
        operand4 = 0;

        cin = 0;
        cin_2=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 = $random; //$random 为系统任务，产生一个随机的 32 位数

always #10 operand4 = $random;

always #10 cin = {$random} % 2; //加了拼接符，{$random}产生一个非负数，除 2 取余得到 0 或 1

always #10 cin_2 = {$random} % 2;

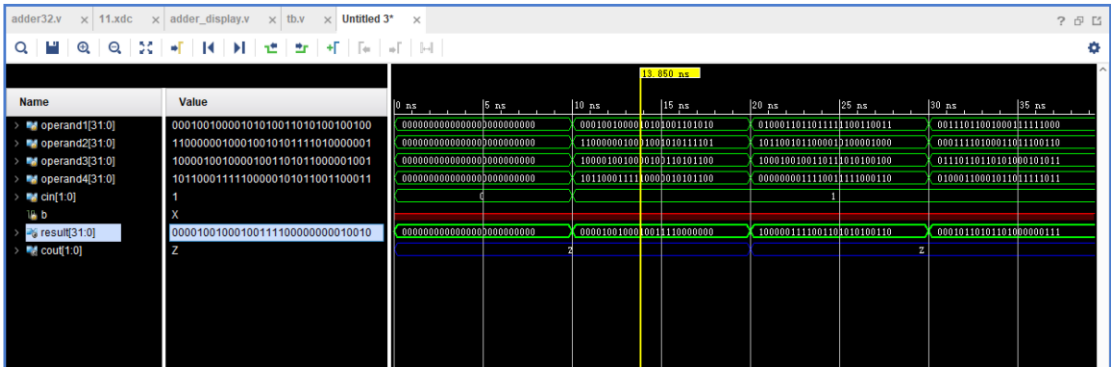
endmodule

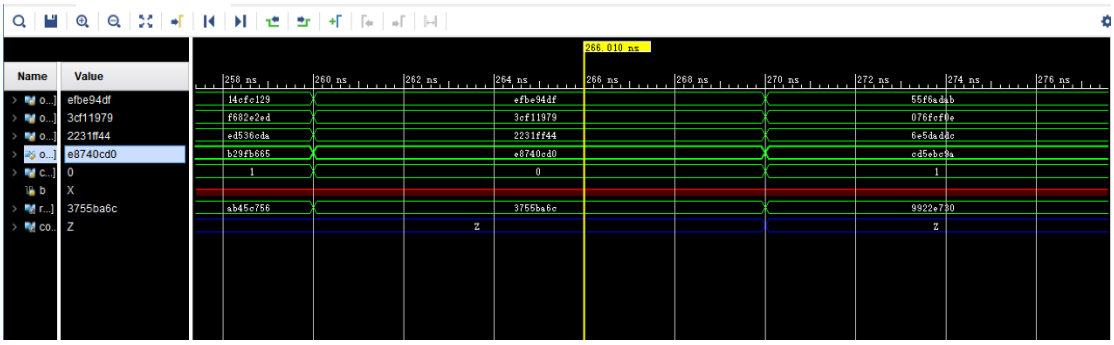
五、实验结果分析

1.仿真结果

仿真结果的波形图如下所示。

图 2，图 3 是四个数相加的结果验证，通过验证可得，波形图中某一时刻四个数相加得到了正确的结果，如果发生两次进位，也会被记录。





2.实验箱结果截图

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，控制两个选择拨码开关，用二进制控制输入的寄存器，然后在屏幕上输入四组数据并计算。

通过验证可以发现得到了正确的计算结果。

六、总结感想

通过实验，我对 LS-CPU-EXB-002 实验箱和软件平台有了更深入的了解,掌握了在 Vivado 中创建一个新项目的方法，并且掌握了利用该实验箱各项功能进行组成原理和体系结构实验的方法。

在修改和运用两个 32 位加法器的基础上，我对加法器的原理和设计有了更深入的理解，并且学会了如何运用 verilog 语言进行电路设计。

通过实验仿真，我成功得到了正确的波形图，并且进行了综合布局布线，并下载到了实验箱中的 FPGA 板上进行演示。

通过这次实验，我为后续设计 cpu 的实验打下了坚实的基础。