组成原理实验课程第_1_次实报告

实验名称	数据运算：定点加法			班级	李涛
学生姓名	齐明杰	学号	2113997	指导老师	董前琨
实验地点	津南实验楼 A306		实验时间	2023.3.21	

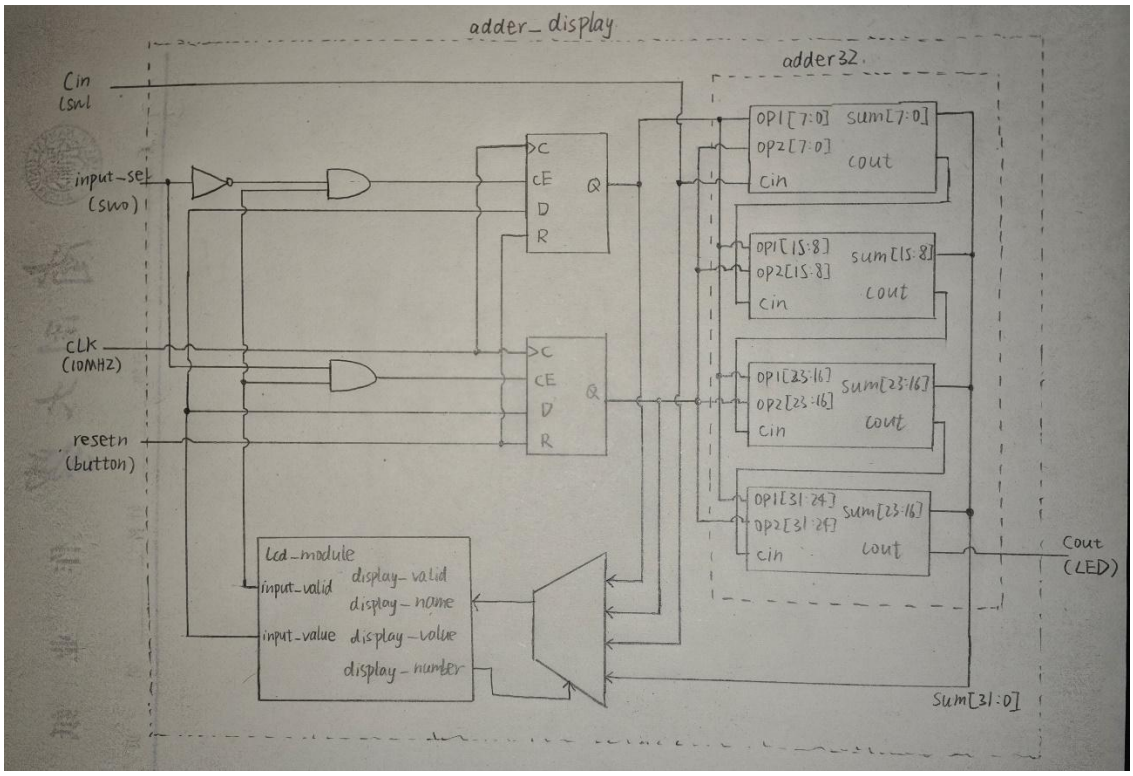
1、 实验目的

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

在教材的实验基础之上，进行改进实验。改进实验中，要求利用 Xilinx Vivado 平台和 LS-CPU-EXB-002 实验箱，设计 adder8 模块，实现 8 位加法器，并进行综合、实现和仿真，进行验证，然后级联 4 个 adder8 加法器，实现 32 位加法器，并上板验证电路设计的正确性。通过 LCD 触摸屏实现加数的输入，通过一个拨码开关切换不同加数的输入，通过另一个拨码开关实现进位的输入，并将其相加求和的结果通过 LCD 触摸屏中的指定区域显示，将进位结果通过一个 LED 灯表示，并最终完成验证。

3、 实验原理图



注：我们分别对 0-7、8-15、16-23、24-31 位进行对应的加法，即用四个 adder8 来实现 32 位加法器，同时将每部分的 cout（进位输出）作为下一部分加法器的 cin（进位输入），以此 0-7 位加法器的进位输入通过人为控制，24-31 位加法器若有进位，则溢出，通常情况无进位，这将通过指示灯的亮（无进位）灭（有进位）来体现。

4、 实验步骤

1.创建项目工程

启动 vivado 软件，在菜单栏点击“File”->“New Project”，出现新建工程向导，选择“Next”，输入工程名称，选择工程的文件位置，然后选择“Next”。选择“RTL Project”，勾选“Do not specify sources at this time”，点击“Next”： 在筛选器的“family”选择“Artix 7”，“package”选择“fbg676”，在筛选得到的型号里面选择“xc7a200tfbg676-2”，然后选择“Next”，点击“Finish”。

2.新建源文件和展示模块

在“Project Manager”下点击“Add sources”，选择“Add or create design sources”；点击“Next”，创建文件，点击“Create File”，依次添加 adder8.v, adder32.v 并在其中写入代码后保存。同时新建外围模块 adder_display.v，在其中写入代码之后保存。

3.连接实验箱

将功能代码进行综合和布局布线后下载到 FPGA 板上运行，在板上检查运行的正确性。 首先需要添加引脚绑定的约束文件。创建约束文件 adder.xdc，并在其中写入引脚绑定代码后保存。在打开 FPGA 实验板，并将下载线与电脑相连后，打开电源，完成 bit 文件的烧写。之后可以通过拨码开关，来控制输入不同的加数；可以通过 LCD 触摸屏上的小键盘，来输入不同的加数完成加法操作。

项目代码：

1. adder8.v

```
module adder8(
    input   [7:0] operand1,
    input   [7:0] operand2,
    input         cin,
    output [7:0] result,
    output        cout
    );
    assign {cout,result} = operand1 + operand2 + cin;
endmodule
```

注：实验提供的模板 adder.v 中是 32 位，本处改为 8 位，其他不变。通过计算两个八位输入(operand1,operand2)与一个进位输入(cin)的加和，来输出求和结果(result)以及进位输出(cout)。

2. adder32.v

```
module adder32(
    input[31:0] operand1,
    input[31:0] operand2,
    input cin,
    output[31:0] result,
    output cout
    );
    wire cin1,cin2,cin3;
    adder8 a(operand1[7:0],operand2[7:0],cin,result[7:0],cin1);
    adder8 b(operand1[15:8],operand2[15:8],cin1,result[15:8],cin2);
    adder8 c(operand1[23:16],operand2[23:16],cin2,result[23:16],cin3);
```

```
    adder8 d(operand1[31:24],operand2[31:24],cin3,result[31:24],cout);
endmodule
```

注：在 **adder32.v** 中，我将原始的 **32** 位加法器分解为四个 **8** 位加法器（**adder8**），并按字节进行操作。改动之处主要包括以下几点：

1.将一个 **32** 位加法器替换为四个 **8** 位加法器（**adder8**）。

2.将操作数（**operand1** 和 **operand2**）分解为 **8** 位宽度的字节，作为各个 **8** 位加法器的输入。

3.使用中间变量（**cin1, cin2, cin3**）来传递各个 **8** 位加法器之间的进位信息。

4.用四个 **8** 位加法器（**a**、**b**、**c**、**d**）来实现逐字节相加的功能，最后通过 **cout** 输出进位。

据此我将一个 **32** 位加法器分解为了四个 **8** 位加法器。

3. adder_display.v

```
//************************************************************************
****
//    > 文件名: adder_display.v
//    > 描述   ：加法器显示模块，调用 FPGA 板上的 IO 接口和触摸屏
//    > 作者   : LOONGSON
//    > 日期   : 2016-04-14
//************************************************************************
****
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 input_sel, //0:输入为加数 1(add_operand1);1:输入为加数 2(add_operand2)
    input sw_cin,

    //led 灯，用于显示 cout
    output 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;
    wire          adder_cin;
    wire [31:0] adder_result    ;
    wire          adder_cout;
    adder32 adder_module(
        .operand1(adder_operand1),
        .operand2(adder_operand2),
        .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)
        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)
        begin
            adder_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'd5 :
            begin
                display_valid <= 1'b1;
                display_name    <= "ADD_1";
                display_value <= adder_operand1;
            end
            6'd6 :
            begin
                display_valid <= 1'b1;
                display_name    <= "ADD_2";
                display_value <= adder_operand2;
            end
            6'd7 :
            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
```

注： 1. 改动后的文件使用 adder32 模块，而非原来的 adder

2. 更改了输出到触摸屏显示的编号。在原始代码中，显示编号为 1、2、3；而在修改后的代码中，显示编号更改为 5、6、7，显示的数值将在触摸屏上的编号 5、6、7 处显示。

4. testbench.v

```
module testbench();
reg[31:0] op1,op2;
reg op;
wire[31:0] sum;
wire flag;
adder32 uut(op1,op2,op,sum,flag);
```

```
initial begin
    op1 = 32'b0;op2 = 32'b0;op = 1'b0;
end
always #3 op1 = $random % 33'b1_0000_0000_0000_0000_0000_0000_0000_0000;
always #5 op2 = $random % 33'b1_0000_0000_0000_0000_0000_0000_0000_0000;
always #7 op = $random % 2'b1_0;
endmodule
```

注：

1. 我修改了输入和输出信号的名称，从 operand1, operand2, cin, result, cout 改为 op1, op2, op, sum, flag。这不会影响功能，只是使用了不同的命名

2. 将原来的 adder 实例替换为 adder32，意味着测试中使用的是我前面编写的使用 4 个 8 位加法器的 32 位加法器。

3. 修改了 initial 块，将初始值设置语句简化为一行。

4. 修改了 always 块中的时间间隔和操作数生成方式：

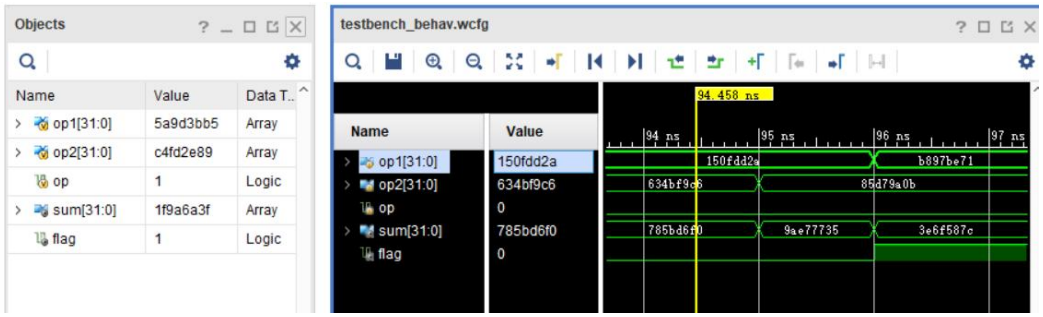
(1)将每 10ns 生成新值的 operand1 和 operand2 更改为每 3ns 和 5ns 分别为 op1 和 op2 生成新值。

(2)使用取余操作 % 33'b1_0000_0000_0000_0000_0000_0000_0000_0000 来限制 op1 和 op2 的最大值，使它们小于 2^32。

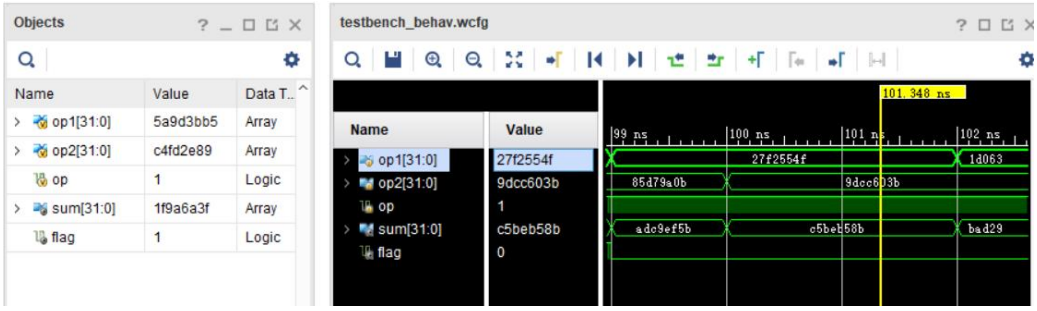
(3)将每 10ns 为 cin 生成新值的语句更改为每 7ns 为 op 生成新值。

5、 实验结果分析

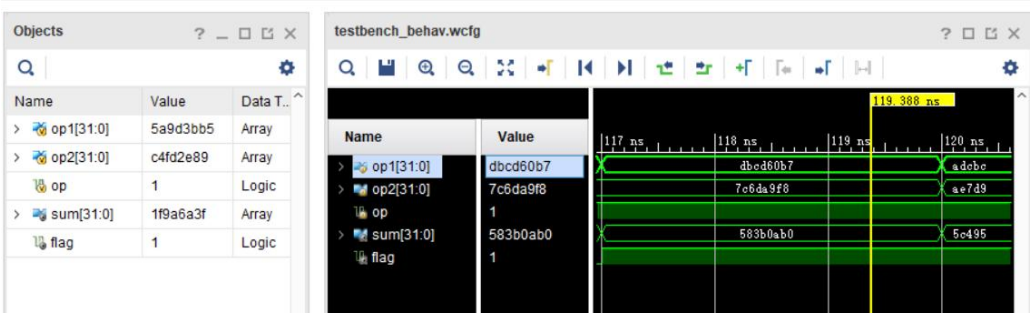
（一） 仿真验证



注： **0x150fdd2a + 0x634bf9c6 + 0 = 0x785bd6f0 (无进位)**



注： **0x27f2554f + 0x9dcc603b + 1 = 0xc5beb58b (无进位)**



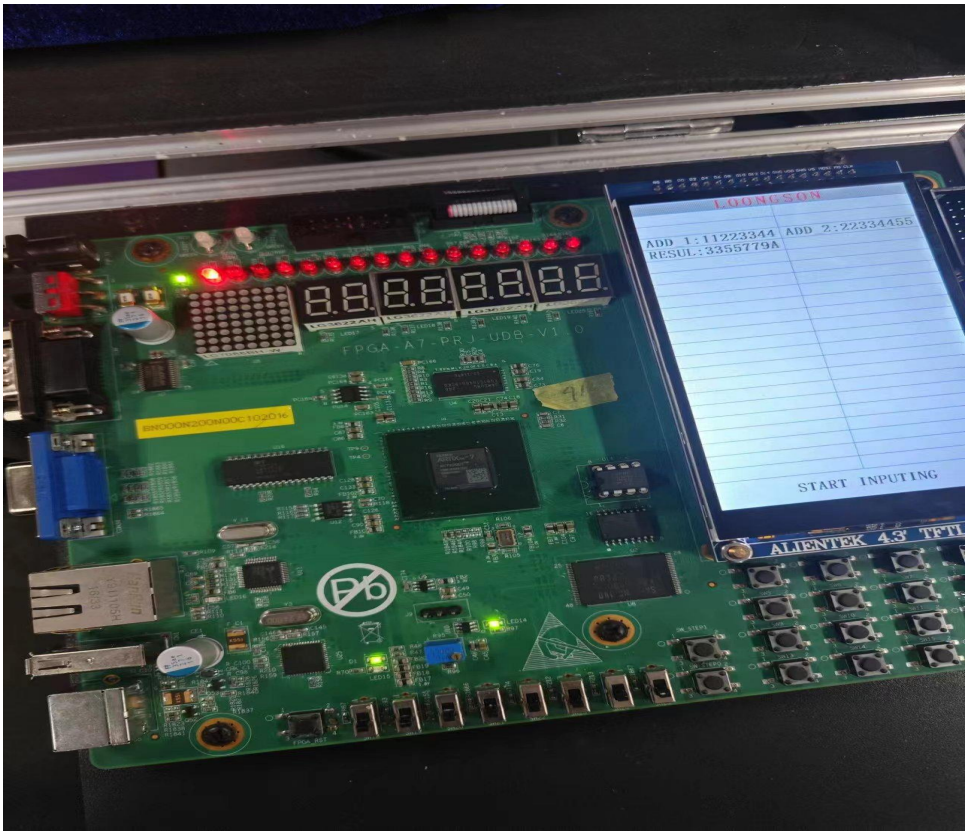
注：0xdbcd60b7 + 0x7c6da9f8 + 1 = 0x583b0ab0　(有进位)

故以上仿真波形可说明 32 位(4 个 8 位)加法器的正确性(包括有进位，无进位情况)。

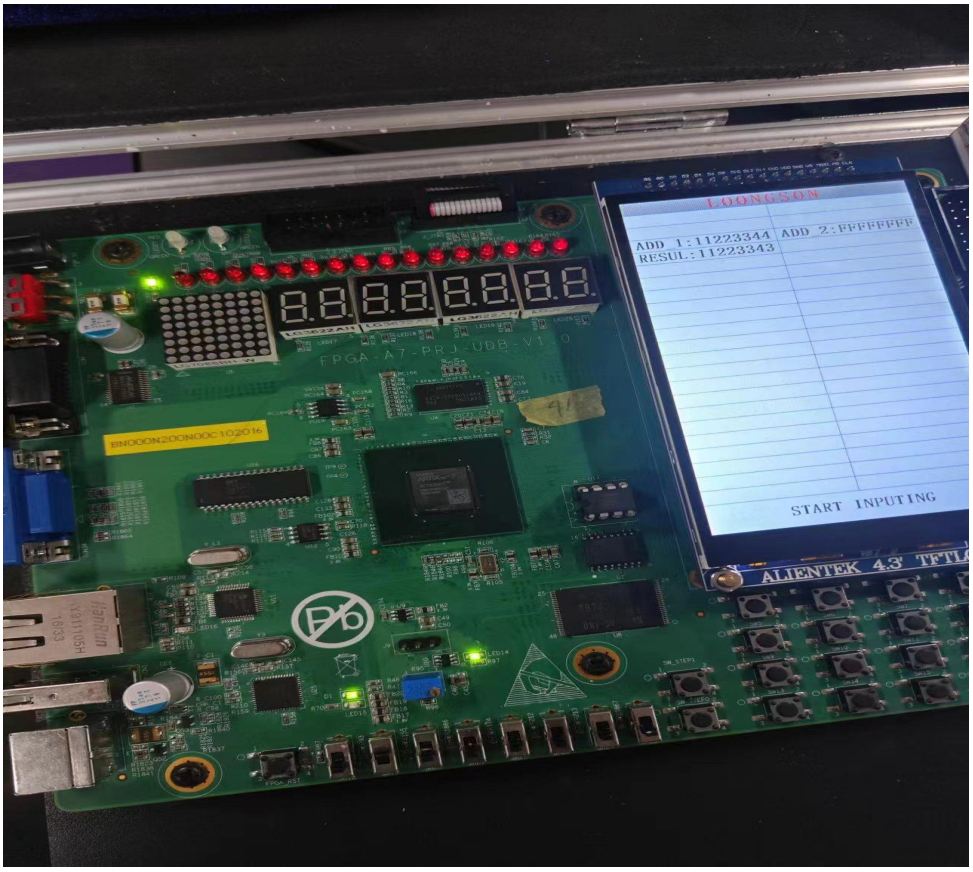
（二）上箱验证

将进位拨码开关拨至“0”状态，输入第一个加数 ADD_1=0x11223344，第二个加数 ADD_2=0x22334455，结果 RESUL=0x33557799，如下图(我将两个加数和结果分别显示在 5、6、7 格)：

此时，将进位拨码开关拨至“1”状态，可见答案比刚才多 1，即 RESUL=0x33557799+1=0x3355779A,如下图：



将进位拨码开关拨向“0”状态，并重新输入 ADD_2=0xFFFFFFFF, ADD_1=0x11223344，则 RESULT=0x11223343，由于加数过大最高位会发生进位，故指示灯熄灭，如下图：



6、 总结感想

本次实验中，我们需要调整显示屏显示的位置，即不能显示在 1、2、3.因此，我仔细研究了 adder_display.v 的代码，找到了修改显示位置的地方。

由于 adder_display.v 中模块之前是 adder module，我写的模块是 adder32，所以一开始一直报错，好在我很快找到了该错误并修正。

另外，本次实验也让我更熟悉了实验箱的使用，了解了实验箱的布局结构