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	李涛老师
学生姓名	孙蕗	学号	2112060	指导老师	董前琨
实验地点	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

（1） 实验设备

1） 装有 Xilinx Vivado 的计算机一台。

2） LS-CPU-EXB-002 教学系统实验箱一套。

（2） 实验任务

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，本次实验的加法器可以使用全加器自己搭建加法模块，也可以在 verilog 中直接使用“+”（系统是自动调用库里加法 IP，且面积时序更优），依据教师要求选择一种方法实现。

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。外围模块中需调用封装好的触摸屏模块，显示两个加数和加法结果，且需要利用触摸功能输入两个加数。

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。

8） 不要在前三格显示。

（3） 实验要求

1） 了解软硬件平台；

2） 掌握定点加法的工作原理；

3） 确定定点加法的输入输出端口设计；

4） 在课前画好设计框图或实验原理图；

5） 如果对 FPGA 板了解的话，可确定设计中与 FPGA 板上交互的接口，画出包含外围模块的整体设计框图，即补充完善图 2.1。

（4） 实验实施：

1） 确认定点加法的设计框图的正确性；

2） 编写 verilog 代码；

3） 对该模块进行仿真，得出正确的波形，截图作为实验报告结果一项的材料；

4） 完成调用定点加法模块的外围模块的设计，并编写代码；

5） 对代码进行综合布局布线下载到实验箱里 FPGA 板上，进行上板验证。

（5） 实验检查：

1） 完成上板验证后，让指导老师或助教进行检查，进行现场演示，可对演示结果进

行拍照作为实验报告结果一项的材料。

（6） 实验报告的撰写：

1） 实验结束后，需按照规定的格式完成实验报告的撰写。

（7） 本次实验

1） 设计两个模块，8 位加法器和 32 位加法器，其中 32 位加法器通过调用 8 位加法器实现。

2） 针对 32 位加法器进行仿真验证

3） 针对 32 位加法器进行上实验箱验证

4） 要求不从前三格输出

3、 实验原理图

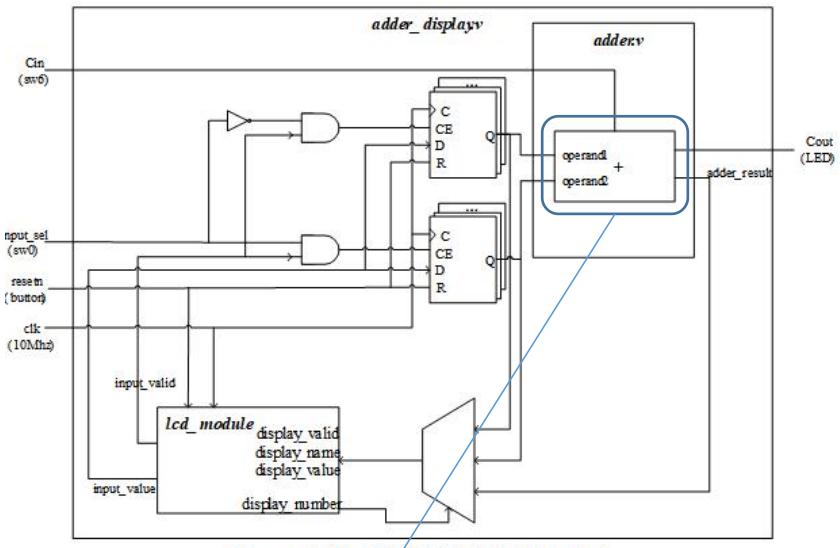
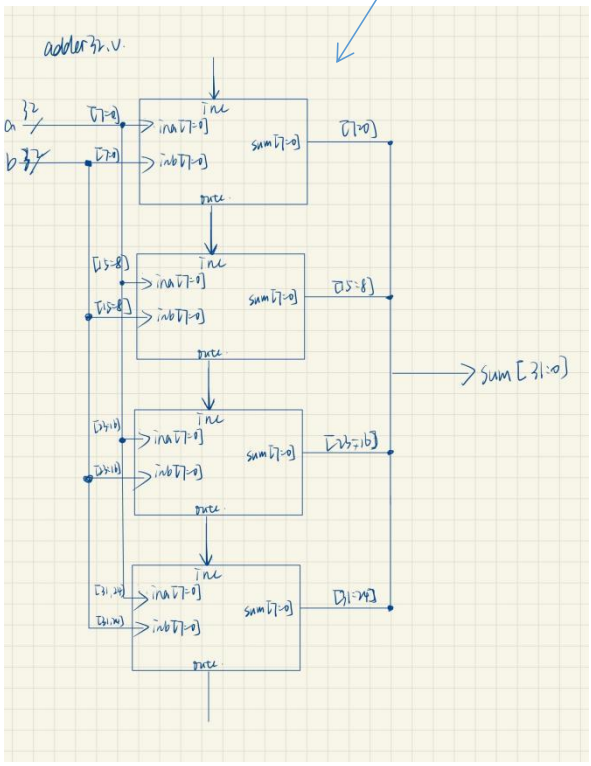


图 2.40 定点加法参考设计的顶层模块框图



4、 实验步骤

（1） 编写 adder8 模块，8 位全加器

```
module adder8(
    input [7:0] ina,//一个八位的数字输入
    input [7:0] inb,//一个八位的数字输入
    input inc,//一个进位输入
    output [7:0] sum,//一个八位的加法结果
    output outc//一个向高位的进位
    );
    assign {outc,sum}= ina+inb+inc;//8 位全加器
endmodule
```

有 2 个 8 位数的输入和 1 个进位输入，产生 1 个 8 位的加法和结果和 1 个向高位的进位。

（2） 编写 8 位全加器功能仿真 testbench

```
module testbench1;
reg[7:0] op1,op2;
reg op;
wire[7:0] sum;
wire flag;
adder8 uut(op1,op2,op,sum,flag);//op1 是第一个 8 位数，op2 是第二个 8 位数，op 是进位，sum 是 8 位加法运算和，flag 是进位标志
initial//可类比 C++的构造函数
begin
    op1=8'b0;op2=8'b0;op=1'b0;
end
always #3 op1=$random%9'b1_0000_0000;//always 语句对后面的语句敏感，#3 代表延长 3 个时间单位，合用意味着每隔 3 个时间单位，…。32 位数，取模的作用是取后 8 位
always #5 op2=$random%9'b1_0000_0000;//每隔 5 个时间单位，得到一个 8 位随机数
always #7 op=$random%2'b1_0;
endmodule
```

（3） 编写 adder32 模块，32 位全加器，32 位加法器通过调用 8 位加法器实现。

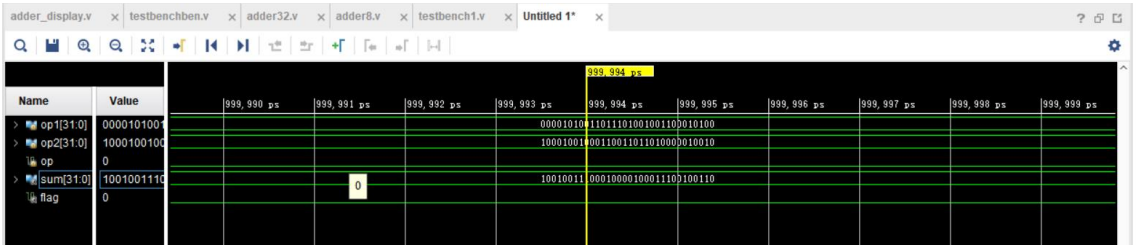
```
module adder32(
    input [31:0] ina,//32 位数输入
    input [31:0] inb,//32 位数输入
    input inc,//进位输入
    output [31:0] sum,//一个 32 位加法结果
    output outc,//一个 32 位加法向高位的进位
    wire [2:0] c//每一步的进位
    );
    adder8 u1 (.ina(ina[7:0]),.inb(inb[7:0]),.inc(inc),.outc(c[0]),.sum(sum[7:0]));//前 8 位数加法，分别取两个输入的前 8 位数参与运算，进位是进位输入 inc，输出的是前 8 位运算的加法结果和进位输出，进位输出记录在 c[0]里，加法运算结果记录在 sum 的前 8 位
    adder8 u2 (.ina(ina[15:8]),.inb(inb[15:8]),.inc(c[0]),.outc(c[1]),.sum(sum[15:8]));//第 9-16 位数加法，分别取两个输入的第 9-16 位数参与运算，进位是 c[0]，输出的是第 9-16 位运算的加法结果和进位输出，进位输出记录在 c[1]里，加法运算结果记录在 sum 的第 9-16 位
```

```
    adder8 u3 (.ina(ina[23:16]),.inb(inb[23:16]),.inc(c[1]),.outc(c[2]),.sum(sum[24:16]));//第17-24位数加法，分别取两个输入的第17-24位数参与运算，进位是c[1]，输出的是第17-24位运算的加法结果和进位输出，进位输出记录在c[2]里，加法运算结果记录在sum的第17-24位
    adder8 u4 (.ina(ina[31:24]),.inb(inb[31:24]),.inc(c[2]),.outc(outc),.sum(sum[31:24]));//第25-32位数加法，分别取两个输入的第25-32位数参与运算，进位是c[2]，输出的是第25-32位运算的加法结果和进位输出，进位输出记录在outc里，加法运算结果记录在sum的第25-32位
endmodule
```

（4） 32位全加器功能仿真testbench

```
module testbenchben;
reg[31:0] op1,op2;//两个32位数输入
reg op;//进位输入
wire[31:0] sum;//一个32位加法结果
wire flag;//一个32位加法向高位的进位
adder32 uut(op1,op2,op,sum,flag);//op1是第一个32位数，op2是第二个32位数，op是进位，sum是32位加法运算和，flag是进位标志
initial//可类比C++的构造函数
begin
    op1 = 0;
    op2 = 0;
    op = 0;
// Wait 100 ns for global reset to finish
    #100;
// Add stimulus here
end
    always #10 op1 = $random;  //$random为系统任务，产生一个随机的32位数
    always #10 op2 = $random;  //#10 表示等待10个单位时间(10ns)，即每过10ns，赋值一个随机的32位数
    always #10 op = {$random} % 2; //加了拼接符，{$random}产生一个非负数，除2取余得到0或1
endmodule
```

（5） adder32，32位全加器，仿真验证，调成二进制数便于观察



调成二进制数便于检验，第一个32位数op1是0000 1010 0110 1110 1001 0011 0001 0100，第二个32位数op2是1000 1001 0001 1001 1011 0100 0001 0010，进位op是0，32位运算结果sum是1001 0011 1000 1000 0100 0111 0010 0110，进位输出flag是0，结果是正确的。

代码编辑和功能仿真都已完成，认为功能基本正确，后续流程就是上板验证了。

（6） 添加外围展示模块，adder_diaplay 文件。对 adder_display 文件进行修改,只展示修改部分的代码。

```
//-----{调用加法模块}begin
    reg    [31:0] adder32_ina;
    reg    [31:0] adder32_inb;
    wire          adder32_inc;
    wire [31:0]   adder32_sum;
    wire          adder32_outc;
    adder32 adder32_module
    (
        .ina(adder32_ina),
        .inb(adder32_inb),
        .inc(adder32_inc),
        .sum(adder32_sum),
        .outc(adder32_outc)
    );
    assign adder32_inc = sw_cin;
    assign led_cout    = adder32_outc;
//-----{调用加法模块}end
```

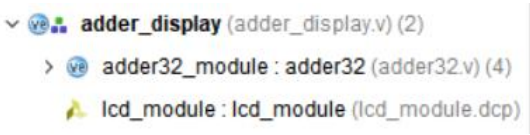
```
//-----{从触摸屏获取输入}begin
    //根据实际需要输入的数修改此小节，
//建议对每一个数的输入，编写单独一个 always 块
    //当 input_sel 为 0 时，表示输入数为加数 1，即 adder32_ina
    always @(posedge clk)
    begin
        if (!resetn)
        begin
            adder32_ina <= 32'd0;
        end
        else if (input_valid && !input_sel)
        begin
            adder32_ina <= input_value;
        end
    end

    //当 input_sel 为 1 时，表示输入数为加数 2，即 adder32_inb
    always @(posedge clk)
    begin
        if (!resetn)
        begin
            adder32_inb <= 32'd0;
        end
```

```
            else if (input_valid && input_sel)
            begin
                adder32_inb <= 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'd6 ://从第 6 格开始输出
            begin
                display_valid <= 1'b1;
                display_name    <= "ADD_1";
                display_value <= adder32_ina;
            end
            6'd7 ://第 7 格
            begin
                display_valid <= 1'b1;
                display_name    <= "ADD_2";
                display_value <= adder32_inb;
            end
            6'd8 ://第 8 格
            begin
                display_valid <= 1'b1;
                display_name    <= "RESUL";
                display_value <= adder32_sum;
            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---------------------//
```

（7） 添加 lcd_module.dcp

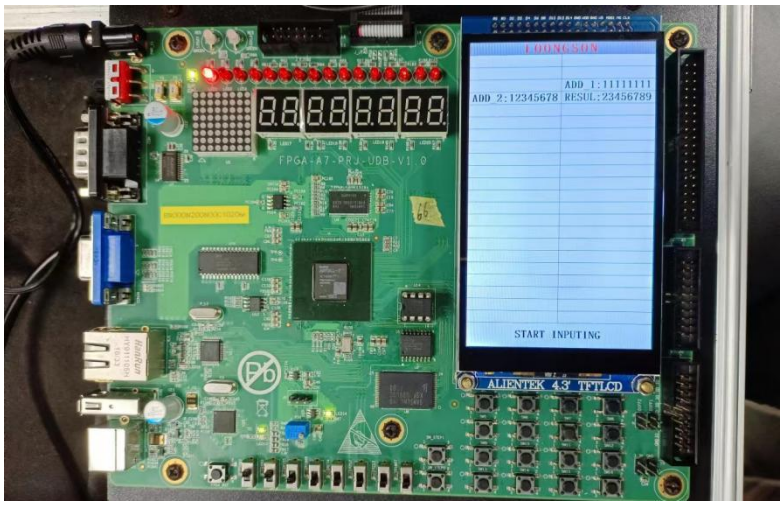


LS-CPU-EXB-002 配套资源设计时，将 lcd_module 模块封装为一个黑盒的网表文件，只需要调用即可。

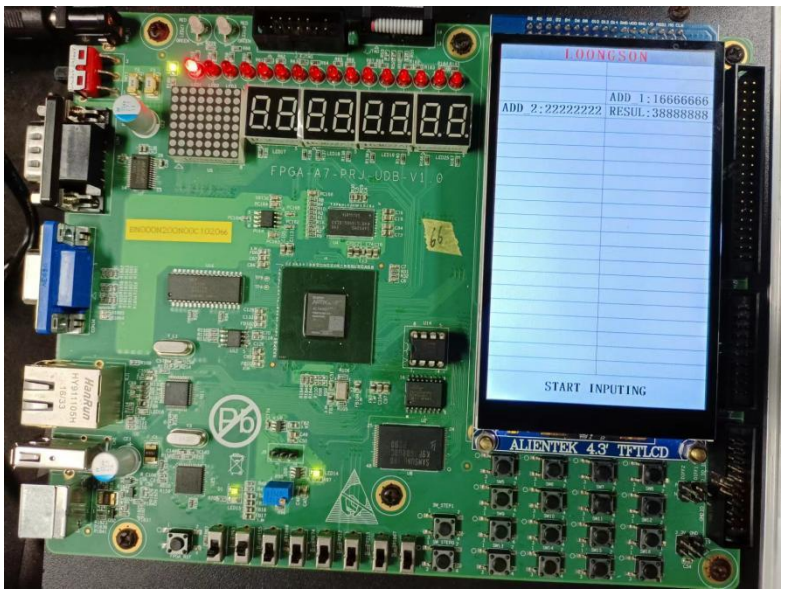
（8） 添加约束文件 adder.xdc

约束文件后缀名为.xdc，用“Add or create constraints”添加或创建约束文件。

5、 实验结果分析



第一个输入为 1111 1111，第二个输入为 1234 5678，进位标志 0，输出结果为 2345 6789。



第一个输入为 1666 6666，第二个输入为 2222 2222，进位标志 0，输出结果为 3888 8888。

6、 总结感想

本次实验主要是学习使用 verilog 语言的语法和结构，理解计算机硬件的基本构建块，并学习使用 vivado 软件。先正确编写 8 位全加器并验证功能，再之后使用 4 个 8 位全加器

构建成一个 32 位全加器，实现更高位数的加法，这样可以减少后续检查错误需要消耗的时间。通过不断地测试和调试，最终实现了 32 位全加器的功能。verilog 的每个模块都应该有清晰的输入和输出端口，并且应该有良好的模块化设计,学会了如何使用仿真工具来验证代码的正确性，确保代码可以按照预期运行。通过将 8 位全加器组合成 32 位全加器，我理解了如何将更复杂的电路设计分解为更简单的模块，并将它们组合成更大的系统.