

组成原理课程第一次实报告

实验名称：数据运算：定点加法

学号：2312631 姓名：林盛森 班次：0981 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。

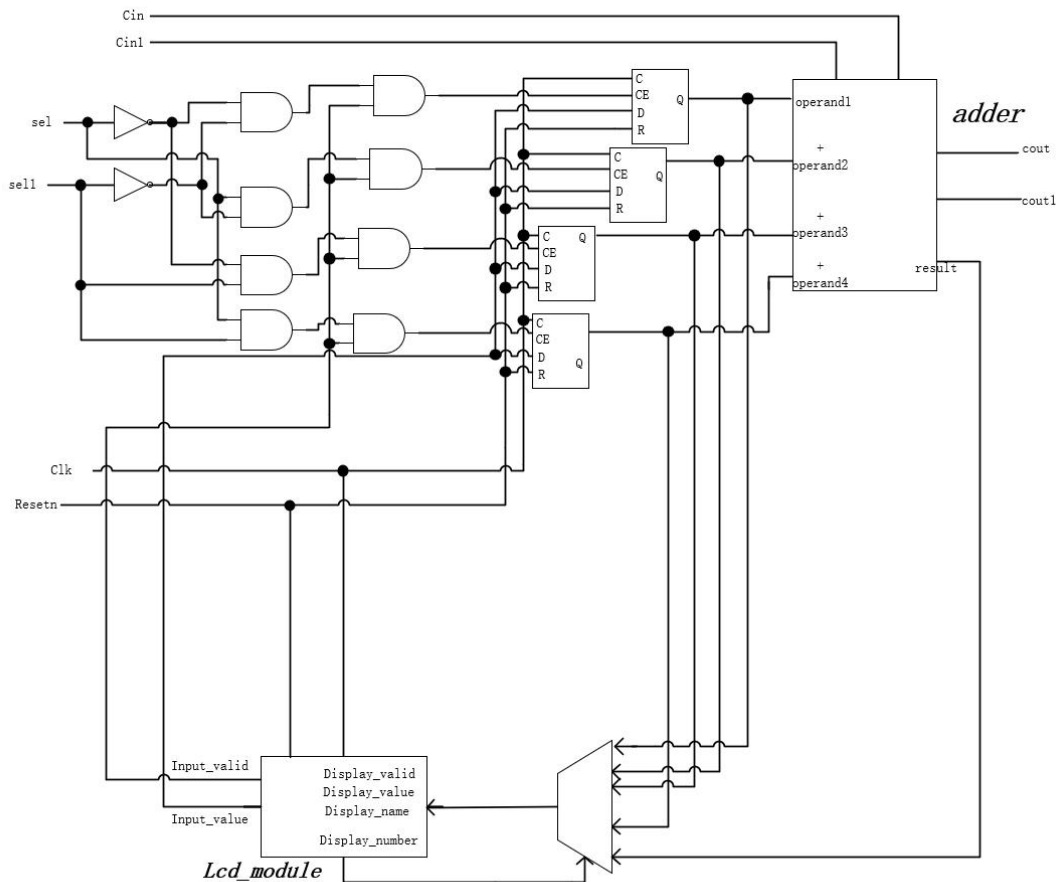
二、 实验内容说明

1. 参考实验手册了解两个数的定点加法器的电路设计，学会使用 vivado、Verilog 语言的基本使用，并学会如何对电路进行实现以及实验箱的使用；
2. 在实验手册给出的两位定点数字加法的基础上，实现四位定点数字加法，并在实验箱上体现出来。

三、 实验原理图

由于现需要四个定点数相加，需要相加三次，一个进位 cin 可能会发生溢出，为了防止进位溢出，因此需要两个 cin，分别为 cin 和 cin1，相应的也需要两个 cout，分别为 cout 和 cout1，两个 sel 来选择四个数字，并在 input_valid,sel,sel1 的三元逻辑上进行设计电路，原理图如下图所示；

Adder_display



四、 实验步骤

1.adder 模块:

根据原理部分说明，首先需要四个 32 位位宽的 operand 来实现定点加数的输入，并且需要两个 cin，两个 cout，分别为 cin，cin1，cout，cout1，以及 assign 部分是四个数相加以及把 cin1 作为进位的高位，cin 作为进位的低位，以下为修改后的代码；

```
module adder(
    input [31:0] operand1,
    input [31:0] operand2,
    input [31:0] operand3,
    input [31:0] operand4,
    input cin,
    input cin1,
    output [31:0] result,
    output cout,
    output cout1
);
    assign {cout1,cout,result} = operand1 + operand2 + operand3 + operand4 + cin + 2 * cin1;
```

2.adder_display 模块:

(1)由于现在显示屏上须显示四个加数和结果,所以需要两个 sel 位,分别位 sel 和 sel1,其中 sel 为低位, sel1 为高位,这样两位二进制数就可以表示 0~3 四种数值,从而来对应四个加数, 0~3 依次对应 operand1~4;

```
//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 input_sel1, //二进制0~3: 实现四位加数的输入
```

(2) 由于低位进位端 cin 和高位进位端 cout 的拓展,相应的进位输入也要变成两个 sw_cin,sw_cin1,进位输出的显示也需要两个 led_cout,led_cout 和 led_cout1;

```
input sw_cin,
input sw_cin1,

//led灯,用于显示cout
output led_cout,
output led_cout1,
```

(3) 相应的,由于 adder_display 中调用了 adder,所以相应的调用代码逻辑我们也需要进行修改;

```
//-----{调用加法模块}begin
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_cin1;
wire [31:0] adder_result ;
wire      adder_cout;
wire      adder_cout1;
```

首先是对这些变量(或参数)进行修改,增加了两个存储加数的寄存器,并且分别引入两个 cin 和 cout, 实现两位输入进位和两位输出进位;

```
adder adder_module(
    .operand1(adder_operand1),
    .operand2(adder_operand2),
    .operand3(adder_operand3),
    .operand4(adder_operand4),
    .cin      (adder_cin      ),
    .cin1     (adder_cin1     ),
    .result   (adder_result   ),
    .cout     (adder_cout     ),
    .cout1    (adder_cout1    )
);
assign adder_cin = sw_cin;
assign adder_cin1 = sw_cin1;
assign led_cout  = adder_cout;
assign led_cout1 = adder_cout1;
```

接着对函数内部的一些参数也进行相应的修改,增加对应数量;

(4) 接着是加数的输入部分，由于我们从两个加数拓展到了四个加数，而加数分别输入的逻辑是用 `sel` 这个标志来判断的，两个加数是，`sel` 为 0 则为第一个加数，为 1 则为输入第二个加数，所以当我们拓展到四个加数时，引入了 `sel1`，则 `sel=0`，`sel1=0` 代表了第一个加数，相应的，`sel1=0`，`sel=1` 代表了第二个加数，`sel1=1`，`sel=0` 代表了第三个加数，`sel1=1`，`sel=1` 代表了第四个加数，所以我们需要对这段逻辑进行修改；

```
always @(posedge clk)
begin
    if (!resetn)
    begin
        adder_operand1 <= 32'd0;
    end
    else if (input_valid && !input_sel && !input_sel1)
    begin
        adder_operand1 <= input_value;
    end
end

always @(posedge clk)
begin
    if (!resetn)
    begin
        adder_operand2 <= 32'd0;
    end
    else if (input_valid && input_sel && !input_sel1)
    begin
        adder_operand2 <= input_value;
    end
end

always @(posedge clk)
begin
    if (!resetn)
    begin
        adder_operand3 <= 32'd0;
    end
    else if (input_valid && !input_sel && input_sel1)
    begin
        adder_operand3 <= input_value;
    end
end

always @(posedge clk)
begin
    if (!resetn)
    begin
        adder_operand4 <= 32'd0;
    end
    else if (input_valid && input_sel && input_sel1)
    begin
        adder_operand4 <= input_value;
    end
end
```

(5) 然后是对在触摸屏上显示加数结果等逻辑的修改，需要增加显示第三、四个加数的代码逻辑；

```
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
```

3. testbench 模块

(1) 首先还是对输入、低位进位输入、高位进位输出的修改；

```
// Inputs
reg [31:0] operand1;
reg [31:0] operand2;
reg [31:0] operand3;
reg [31:0] operand4;
reg cin;
reg cin1;
```

```
// Outputs
wire [31:0] result;
wire cout;
wire cout1;
```

(2) 然后是创建测试对象的实例这一部分，也是修改相应的输入、进位输入、输出；

```
adder uut (
    .operand1(operand1),
    .operand2(operand2),
    .operand3(operand3),
    .operand4(operand4),
    .cin(cin),
    .cin1(cin1),
    .result(result),
    .cout(cout),
    .cout1(cout1)
);
```

(3) 接着是输入、进位输入的初始化修改；

```
initial begin
    // Initialize Inputs
    operand1 = 0;
    operand2 = 0;
    operand3 = 0;
    operand4 = 0;
    cin = 0;
    cin1 = 0;
    // Wait 100 ns for global reset to finish
    #100;
    // Add stimulus here
end
```

(4) 最后修改随机生成变量的个数，增加第三、四个数和低位进位的高位端的随机输入；

```
always #10 operand1 = $random; // $1
always #10 operand2 = $random; // #1
always #10 operand3 = $random;
always #10 operand4 = $random;
always #10 cin = {$random} % 2; //加
always #10 cin1 = {$random} % 2;
```

4. Constraints 模块

(1) 对 package_pin 部分修改，包括 cout, cin, sel; cout 对应的这两句分别对应了相

应的输出 led 灯，sel 和 cin 的这几行分别对应了实验箱上的开关：

```
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_cout1]
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_sel1]
set_property PACKAGE_PIN AD24 [get_ports sw_cin]
set_property PACKAGE_PIN AC23 [get_ports sw_cin1]
```

(2) 对 iostandard 也作相同的修改：

```
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports led_cout]
set_property IOSTANDARD LVCMOS33 [get_ports led_cout1]
set_property IOSTANDARD LVCMOS33 [get_ports resetn]
set_property IOSTANDARD LVCMOS33 [get_ports input_sel]
set_property IOSTANDARD LVCMOS33 [get_ports input_sel1]
set_property IOSTANDARD LVCMOS33 [get_ports sw_cin]
set_property IOSTANDARD LVCMOS33 [get_ports sw_cin1]
```

五、 实验结果分析

1. 仿真结果



如图所示，四个输入分别为 faf32ef5, 7a87aff5, aeeacc5d, ca481294, 输出为 eeadbddb, 最后一位相加 $5+5+d+4=27$, 所以向高位进 1, 余 11, 即 b; 其次:

$f+f+5+9+1=45$, 进 2, 余 13, 即 d;

$e+f+c+2+2=45$, 进 2, 余 d;

$2+a+c+1+2=27$, 进 1, 余 b;

$3+7+a+8+1=29$, 进 1, 余 d;

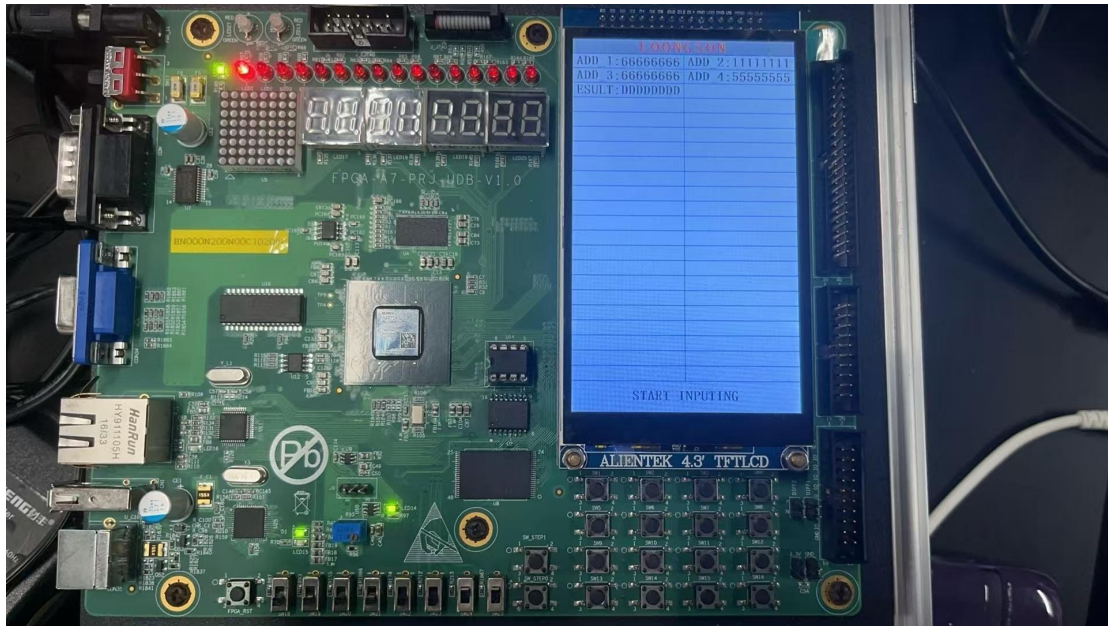
$f+8+e+4+1=42$, 进 2, 余 a;

$a+a+e+a+2=46$, 进 2, 余 e;

$f+7+a+c+2=46$, 进 2, 余 e;

由上述计算核实, 说明仿真结果正确, 设计是合理的;

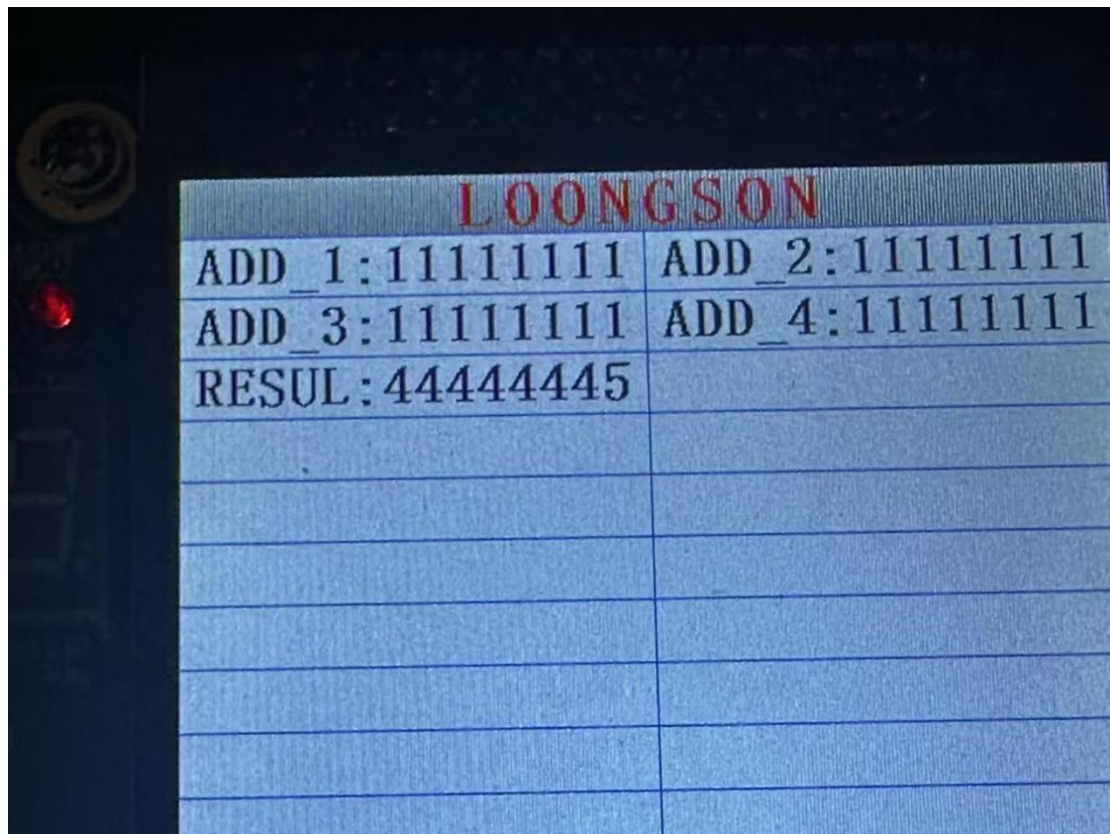
2. 实验箱运行结果



(1) 进位为 00:

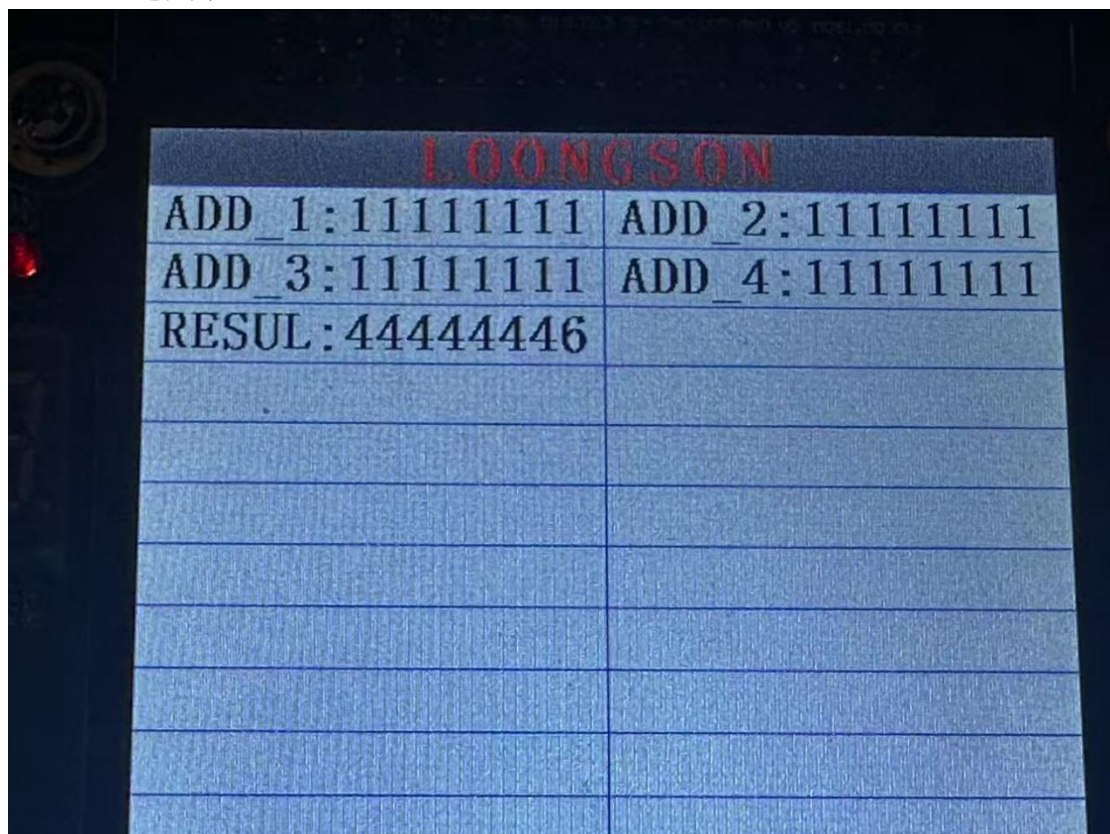


(2) 进位为 01:



LOONGSON	
ADD_1:11111111	ADD_2:11111111
ADD_3:11111111	ADD_4:11111111
RESUL:44444445	

(3) 进位为 10:



LOONGSON	
ADD_1:11111111	ADD_2:11111111
ADD_3:11111111	ADD_4:11111111
RESUL:44444446	

(4) 进位为 11:

LOONGSON	
ADD_1:11111111	ADD_2:11111111
ADD_3:11111111	ADD_4:11111111
RESUL:44444447	

六、总结感想

- 1.学会了 verilog 语言的基本使用方法，与 c++类似，较易理解；
- 2.了解了如何设计一个加法器，应该有哪些模块，以及这些模块之间如何去调用；
- 3.学会如何利用仿真测试去验证程序的正确性；
- 4.对于 vivado 中的设计文件，仿真文件，约束文件的有了更深刻的了解，以及其中不同文件的作用以及文件与文件间的关系；
- 5.学会了实验箱的使用方法，以及要通过约束文件去对应程序中的输入输出与对应的引脚开关.