**组成原理课程第 1 次实报告**

**实验名称：32位加法器**

学号： 2313721 姓名： 许洋 班次： 张金老师

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。

1. 实验内容说明

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。

1. 实验原理图

（画图并简要说明）

1. 实验步骤

（**1.adder32模块的实现**

**功能解释**

​ 首先，我们需要实现4个32位数的相加操作，原来的文档里实现的是两个32位数进行相加，因此只需要一个进位和一个输出，但是当出现四个数相加时，我们就需要用两位数的输入与输出来实现了。

​ 代码部分，我们首先使用4个32位位宽的输入operand来实现原始值的写入；再分别使用两个进位输入cin和cin1、两个高位进位输出cout和cout1来实现代码内容的修改。关键部分assign的实现，是我们使用4个数进行相加，再加上两个我们的进位输入。

**代码部分**

​ 以下是我修改后的代码部分。

module adder32(

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 {cout,result} = operand1 + operand2 +operand3+operand4+ cin+2\*cin1;

endmodule

**修改**

​ 跟原来的代码相比，我做了以下的修改。

​ 1.首先，我对input的数量进行了修改，原来只需要输入两个operand的，现在需要4个，所以我们改成使用operand1~operand4。

input [31:0] operand1,

input [31:0] operand2,

input [31:0] operand3,

input [31:0] operand4,

​ 2.然后，我还对进位的位数做了修改，原先是进位1位，只需要一个cin；现在由于是四个数相加，所以我们需要两位cin，为了做区分，我们将这两个进位分别命名为cin和cin1。

input cin,

input cin1,

​ 3.同理，在修改进位cin后，我们也需要对向高位的进位cout进行修改，由于是四个数相加，则会出现两位向高位的进位，因此需要两位cout，为了区分，我们将两个向高位的进位分别命名为cout和cout1。

output cout,

output cout1

​ 4.最后则是对进位计算方式的修改。原来的assign为assign {cout,result} = operand1 + operand2 + cin;现在我们将其修改为assign {cout,result} = operand1 + operand2 +operand3+operand4+ cin+2\*cin1;发现我们的cout和result变成了由4个operand和两位进位输入来控制，其中cin1为进位高位，所以在计算时需要乘上2，来表示高位的进位。

assign {cout,result} = operand1 + operand2 +operand3+operand4+ cin+2\*cin1;

**2.adder\_display模块的实现**

**功能解释**

​ adder\_display是本项目的外围模块，该外围模块调用adder32.v，并且调用触摸屏上的模块，以便于在板上获得实验结果。

**代码部分**

​ 下面是我修改后的代码部分。

module adder\_display(

input clk,

input resetn,

input input\_sel,

input input\_sel1,

input sw\_cin,

input sw\_cin1,

output led\_cout,

output led\_cout1,

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

);

reg [31:0] adder\_operand1;

reg [31:0] adder\_operand2;

reg [31:0] adder\_operand3;

reg [31:0] adder\_operand4;

wire adder\_cin;

wire addr\_cin1;

wire [31:0] adder\_result ;

wire adder\_cout;

wire adder\_cout1;

adder32 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;

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 ),

.resetn (resetn ),

.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\_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 )

);

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

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 <= "RESULT";

display\_value <= adder\_result;

end

default :

begin

display\_valid <= 1'b0;

display\_name <= 40'd0;

display\_value <= 32'd0;

end

endcase

end

endmodule

**修改**

​ 跟原来的代码相比，我修改了以下这些部分。

​ 1.对于语句input input\_sel，原来的意思是：sel的值为0，代表输入为加数 1(operand1);sel的值为1，代表输入为加数2(operand2)。为了实现4个加数的输入，我们设置两个数sel和sel1，sel代表高位、sel1代表低位，两个数分别取0和1，就能完整表示0-3的数（二进制），就能对应到4个加数。

input input\_sel,

input input\_sel1,

​ 2.对于语句input sw\_cin和output led\_cout这两句，原先都是实现进位和显示LED灯的，我们在此处都加成两个，用cin和cin1、cout和cout1来实现两位进位的输入与高位进位的显示。

input sw\_cin,

input sw\_cin1,

output led\_cout,

output led\_cout1,

​ 3.然后就是对调用加法模块的修改。可以看到原来的调用，只使用了两个32位寄存器operand1和operand2的输入，只使用了一个adder\_cin和一个adder\_cout，在此处我们修改为：使用四个32位寄存器来输入，分别用两个cin和cout。对于加法模块的调用，我们跟上面的一样，也是只需要对operand的数量与cin、cout的数量进行修改就可以了。对于最后的与实验箱链接的部分，也是从一句变成两句。

reg [31:0] adder\_operand1;

reg [31:0] adder\_operand2;

reg [31:0] adder\_operand3;

reg [31:0] adder\_operand4;

wire adder\_cin;

wire addr\_cin1;

wire [31:0] adder\_result ;

wire adder\_cout;

wire adder\_cout1;

adder32 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;

​ 4.然后是对数的输入的修改，原先的实现方式是，通过input sel的控制，sel输出0就代表是加数1；sel输出1就代表加数是2，再进行分别对应的输出。我们现在需要判定加数1-4，所以我么使用前面命名的sel和sel1来实现。两个二进制数，代表了00/01/10/11四种可能，我们需要将其进行一一对应。

​ 按照原来的方法我们编写语句，当sel和sel1都为0时，代表输出加数1；当sel为0，sel1为1时，代表输出加数2；当sel为1，sel1为0时，代表输出加数3；当sel和sel1都为1时，代表输出加数4。

​ 我们对应的编写代码即可，使用二进制数之间的与或非逻辑运算来实现判断过程。我们拿add1来举例子，当输入的valid和！sel和！sel1都为1时，即sel和sel1都为0时，选择加数1，说明sel与sel1的取值成功确定了对应的加数位置。

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

​ 5.最后是输出到触摸屏的模块，我们只需要将原来的调用加法的个数从2改到4就可以了。我们增加以下的代码。

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

**3.testbench模块的实现**

**功能解释**

​ 该部分是用于实现功能仿真，以此来检验功能的正确性，在出错的情况下可以准确定位到错误的位置。我们需要将输入激励由2个改到4个，进位信号由1个改到2个就可以了。

**代码部分**

​ 下面是我修改后的代码部分。

module testbench;

reg [31:0] operand1;

reg [31:0] operand2;

reg [31:0] operand3;

reg [31:0] operand4;

reg cin;

reg cin1;

wire [31:0] result;

wire cout;

wire cout1;

adder32 uut (

.operand1(operand1),

.operand2(operand2),

.operand3(operand3),

.operand4(operand4),

.cin(cin),

.cin1(cin1),

.result(result),

.cout(cout),

.cout1(cout1)

);

initial begin

operand1 = 0;

operand2 = 0;

operand3 = 0;

operand4 = 0;

cin = 0;

cin1= 0;

#100;

end

always #10 operand1 = $random;

always #10 operand2 = $random;

always #10 operand3 = $random;

always #10 operand4 = $random;

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

always #10 cin1 = {$random} % 2;

endmodule

**修改**

​ 1.将输入的寄存器改为了4个，进位输入的寄存器改为了2个，输出的cout改为了2个。

reg [31:0] operand1;

reg [31:0] operand2;

reg [31:0] operand3;

reg [31:0] operand4;

reg cin;

reg cin1;

wire [31:0] result;

wire cout;

wire cout1;

​ 2.对于uut模块，也是只需要修改输入、进位输入与输出的个数就可以了。

adder32 uut (

.operand1(operand1),

.operand2(operand2),

.operand3(operand3),

.operand4(operand4),

.cin(cin),

.cin1(cin1),

.result(result),

.cout(cout),

.cout1(cout1)

);

​ 3.对于开始模拟的版块，我们修改初始输入的个数，从2改为4，这样实现了初始的四输入。同样的，修改cin的个数，修改后期随机生成模拟的变量个数，即可实现模拟仿真的功能。

initial begin

operand1 = 0;

operand2 = 0;

operand3 = 0;

operand4 = 0;

cin = 0;

cin1= 0;

#100;

end

always #10 operand1 = $random;

always #10 operand2 = $random;

always #10 operand3 = $random;

always #10 operand4 = $random;

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

always #10 cin1 = {$random} % 2;

**4.mycons模块的实现**

**功能解释**

​ 该文件是一个约束文件，功能是添加引脚绑定，使实验箱的引脚与我们的功能联系起来。

**代码部分**

​ 下面是我修改后的代码部分。

解释

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]

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]

#lcd

set\_property PACKAGE\_PIN J25 [get\_ports lcd\_rst]

set\_property PACKAGE\_PIN H18 [get\_ports lcd\_cs]

set\_property PACKAGE\_PIN K16 [get\_ports lcd\_rs]

set\_property PACKAGE\_PIN L8 [get\_ports lcd\_wr]

set\_property PACKAGE\_PIN K8 [get\_ports lcd\_rd]

set\_property PACKAGE\_PIN J15 [get\_ports lcd\_bl\_ctr]

set\_property PACKAGE\_PIN H9 [get\_ports {lcd\_data\_io[0]}]

set\_property PACKAGE\_PIN K17 [get\_ports {lcd\_data\_io[1]}]

set\_property PACKAGE\_PIN J20 [get\_ports {lcd\_data\_io[2]}]

set\_property PACKAGE\_PIN M17 [get\_ports {lcd\_data\_io[3]}]

set\_property PACKAGE\_PIN L17 [get\_ports {lcd\_data\_io[4]}]

set\_property PACKAGE\_PIN L18 [get\_ports {lcd\_data\_io[5]}]

set\_property PACKAGE\_PIN L15 [get\_ports {lcd\_data\_io[6]}]

set\_property PACKAGE\_PIN M15 [get\_ports {lcd\_data\_io[7]}]

set\_property PACKAGE\_PIN M16 [get\_ports {lcd\_data\_io[8]}]

set\_property PACKAGE\_PIN L14 [get\_ports {lcd\_data\_io[9]}]

set\_property PACKAGE\_PIN M14 [get\_ports {lcd\_data\_io[10]}]

set\_property PACKAGE\_PIN F22 [get\_ports {lcd\_data\_io[11]}]

set\_property PACKAGE\_PIN G22 [get\_ports {lcd\_data\_io[12]}]

set\_property PACKAGE\_PIN G21 [get\_ports {lcd\_data\_io[13]}]

set\_property PACKAGE\_PIN H24 [get\_ports {lcd\_data\_io[14]}]

set\_property PACKAGE\_PIN J16 [get\_ports {lcd\_data\_io[15]}]

set\_property PACKAGE\_PIN L19 [get\_ports ct\_int]

set\_property PACKAGE\_PIN J24 [get\_ports ct\_sda]

set\_property PACKAGE\_PIN H21 [get\_ports ct\_scl]

set\_property PACKAGE\_PIN G24 [get\_ports ct\_rstn]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_rst]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_cs]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_rs]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_wr]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_rd]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_bl\_ctr]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[8]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[9]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[10]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[11]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[12]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[13]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[14]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[15]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports ct\_int]

set\_property IOSTANDARD LVCMOS33 [get\_ports ct\_sda]

set\_property IOSTANDARD LVCMOS33 [get\_ports ct\_scl]

set\_property IOSTANDARD LVCMOS33 [get\_ports ct\_rstn]

**修改**

​ 1.修改对应的输出LED灯，写两句，分别对应相应的引脚。

set\_property PACKAGE\_PIN H7 [get\_ports led\_cout]

set\_property PACKAGE\_PIN D5 [get\_ports led\_cout1]

​ 2.修改对应的sel和cin的引脚，分别对应到实验箱的1-4开关，通过开关的切换就可以实现加数的选择与进位的输入。后面的对于IOSTANDARD的修改也是同理，就不再说明了。

解释

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]

实验结果分析

（仿真结果截图或者实验箱运行结果拍照，注意需要对实验结果进行分析，输入是什么，输出是什么，结果是什么，是否验证了正确性）

1. 总结感想

（说说本次实验的总结感想）

注意：实验报告提交的文件名为“组原第X次实验\_学号\_姓名\_班级.pdf”请务必提交pdf文件，word文件偶尔会有排版问题，班级用理论课老师姓名表示，分别是李涛老师/张金老师