组成原理实验课程第一次实验报告

实验名称：数据运算：定点加法　班级：李涛老师　学生姓名：陆皓喆　学号：2211044

指导老师：董前琨　　实验地点：实验楼A306　实验时间：2024.03.14

一、实验目的

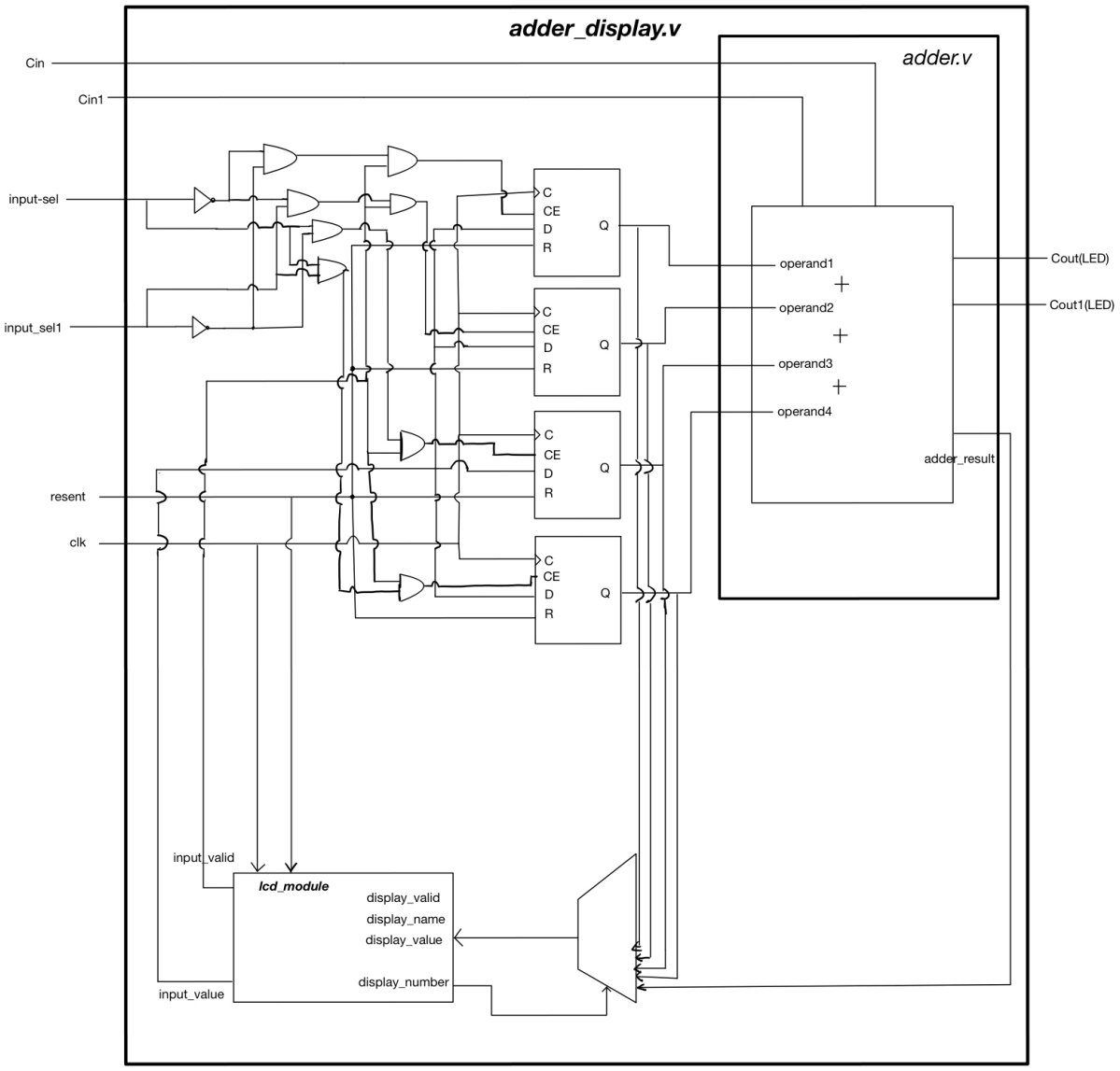
1. 熟悉LS-CPU-EXB-002 实验箱和软件平台。
2. 掌握利用该实验箱各项功能开发组成原理和体系结构实验的方法。
3. 理解并掌握加法器的原理和设计。
4. 熟悉并运用verilog 语言进行电路设计。
5. 为后续设计cpu的实验打下基础。

二、实验内容说明

1. 初步接触 Verilog语言， 使用 vivado软件来进行电路的设计，学会如何建立源文件、设计外围模块、对电路进行综合和实现等功能，了解基本的电路上箱验证的方法。
2. 结合实验指导手册中的实验一（加法器实验）完成功能改进，实现一个能完成4个32位数的加法的加法器。

三、实验原理图

我们根据实验要求的分析，得出了最终的实验流程，如下图所示。



我们使用四个`operand`来进行叠加，可以发现4个32位数叠加，需要两位进位`cin`和`cin1`，分别对应实验箱上的2和4端口。同样，输出端我们也需要两位`cout`和`cout1`来进行表示。对于输入端口的选择，我们同样使用两个端口`sel`和`sel1`来确定输出端口，分别对应了实验箱上的1和3端口。`input_sel`和`input_sel1`用于选择指定输入数，两个二进制的数合起来可以表示4个含义，用于指定4个输入数。`adder.v`模块用于实现四个数的相加，而外围模块`adder_display.v`则用于串联整个工程。

四、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。

```
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]
```

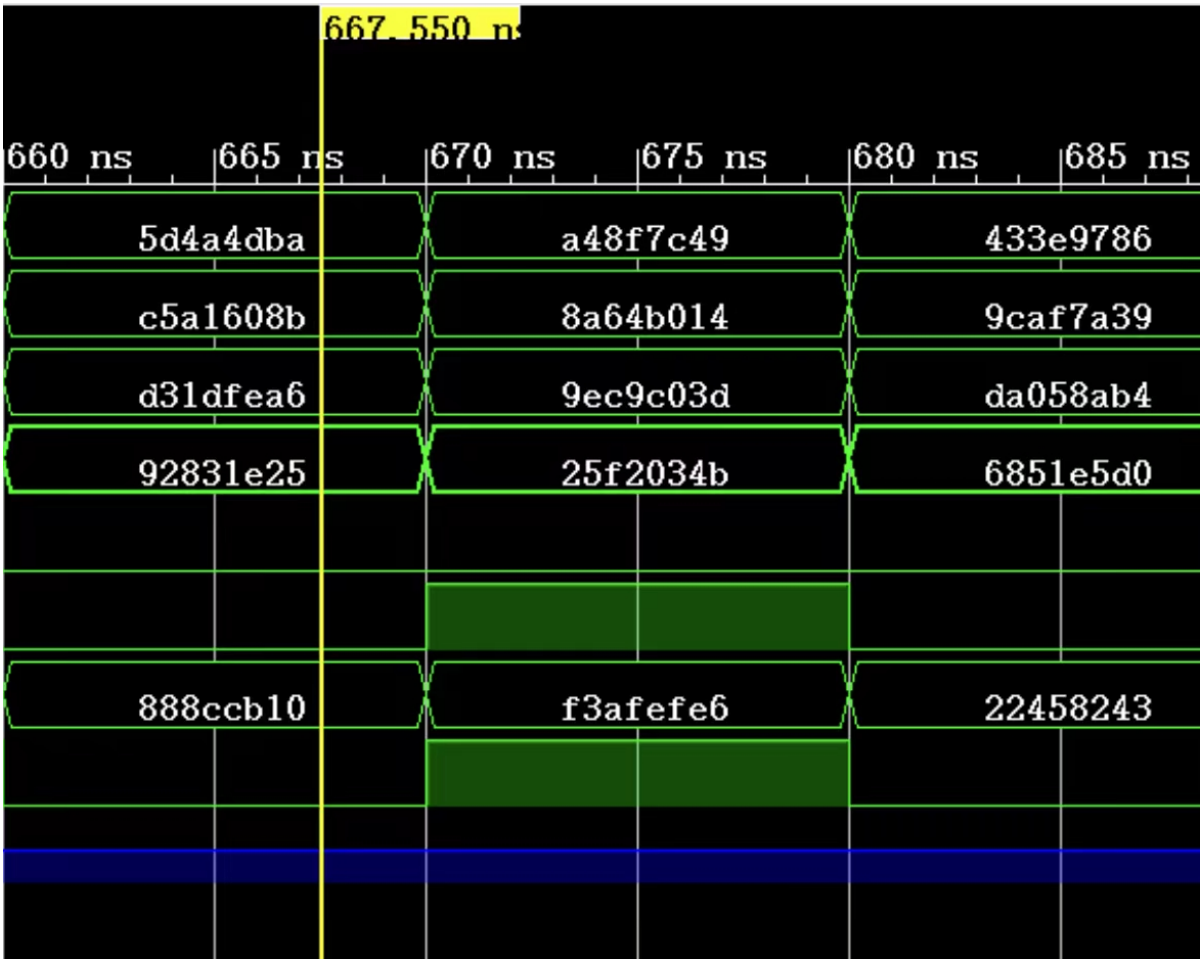
2.修改对应的 `sel` 和 `cin` 的引脚，分别对应到实验箱的1-4开关，通过开关的切换就可以实现加数的选择与进位的输入。后面的对于 `IOSTANDARD` 的修改也是同理，就不再说明了。

```
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]
```

五、实验结果分析

1.仿真验证

我们通过vivado中的仿真模拟来检验自己的程序是否正确地实现了功能。以下是我的仿真结果。



以上是我们的仿真验证文件随机生成的波形图，我们使用十六进制计算器来验证一下结果是否正确。

我们观察第一列的数据，首先最低位相加，a+b+6+5=32，进位2，余0；

b+8+a+2+2=33，进位2，余1；

d+0+e+e+2=43，进位2，余b；

4+6+f+1+2=28，进位1，余c；

a+1+d+3+1=28，进位1，余c；

4+a+1+8+1=24，进位1，余8；

d+5+3+2+1=24，进位1，余8；

5+c+d+9+1=40，进位2，余8。

经过计算，我们发现第一列的四个16进制数相加确实是888ccb10，通过以上的仿真，说明我们的模块设计是正确的。

2.上箱验证

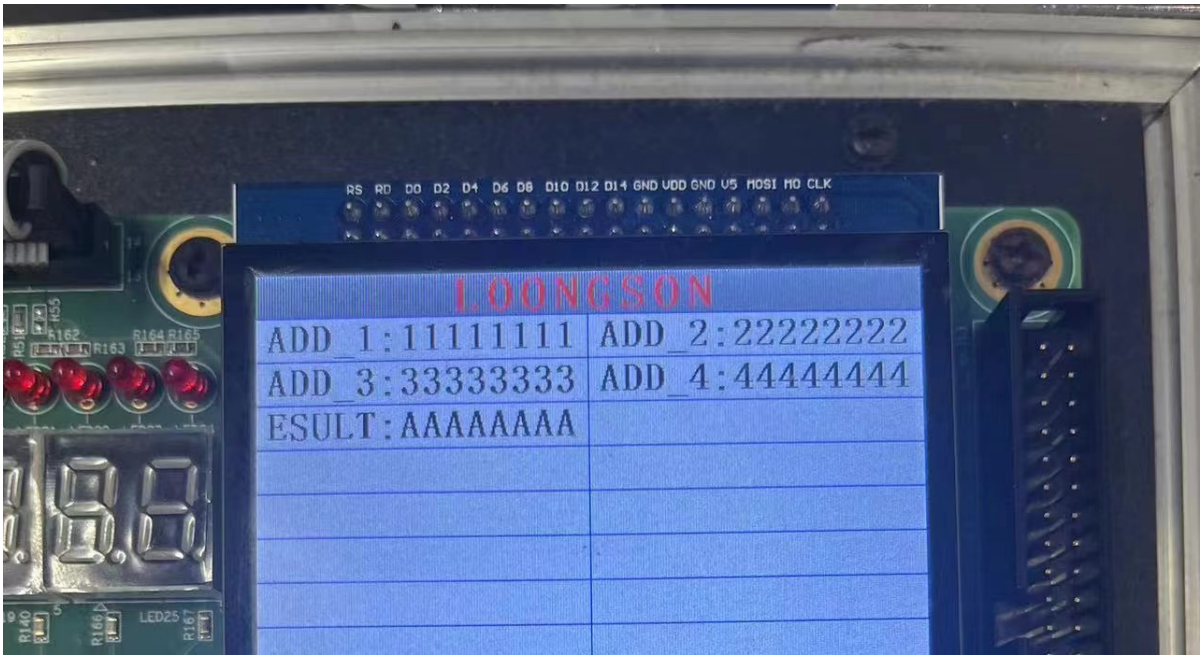
在实验箱上，从左边开始的第一个和第三个开关是控制加数选择的；第二个和第四个开关是手动输入进位的，我们分别进行上箱操作，得出以下结果。

(1)无进位

我们将`cin`和`cin1`调到0的位置进行测试。

$$11111111 + 22222222 + 33333333 + 44444444 = AAAAAAAA$$

发现测试结果无进位，且输出正确。

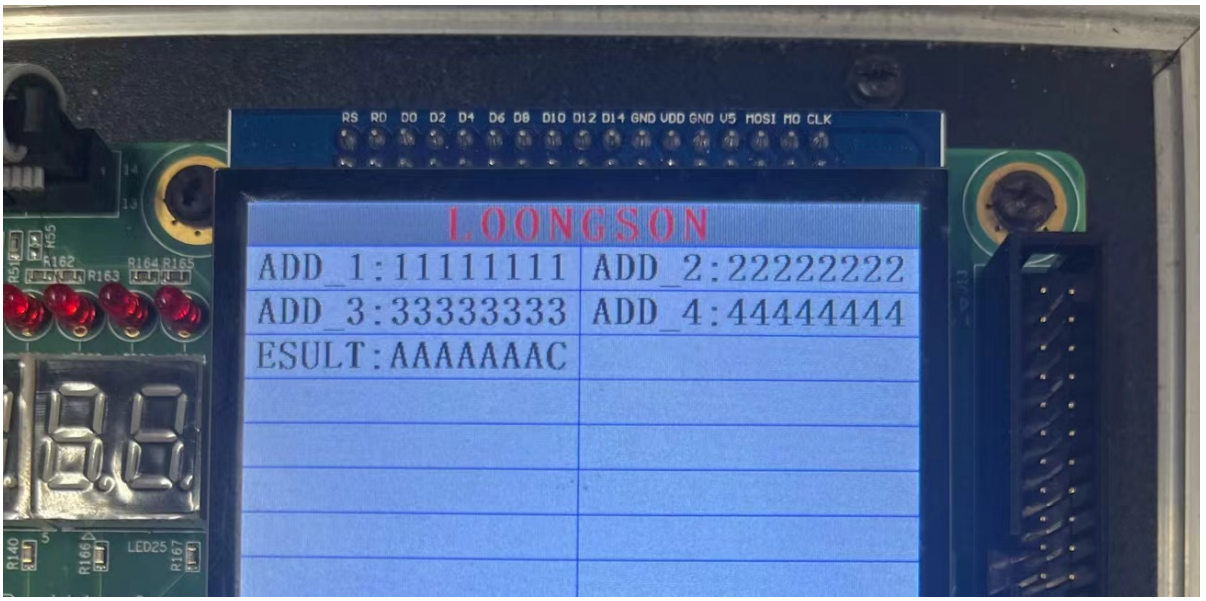


(2)有进位

我们将`cin1`拨到1，进行测试。

$$11111111 + 22222222 + 33333333 + 44444444 + 1 \times 2 = AAAAAAAC$$

结果比原先多出2，结果正确。



以上的结果说明我们的模块设计正确，实验完成！

六、总结感想

1. 通过这次实验，我学会了如何在vivado上创建一个新的项目，学会了如何去调试，如何编译运行，如何做仿真，如何将工程与实验箱联系起来，入门了verilog语言。
2. 了解了加法运算的基本原理，并在上机课上得到了实现，对理论课的知识理解地更加透彻了。
3. 对于vivado的三类基本文件——设计文件、约束文件、仿真文件有了初步的了解，学会了如何在项目中创建这些文件或者是导入这些文件。
4. 了解了外围模块文件的作用，在实验报告前面的整体流程中，`adder_display`就是起到了一个外围的作用，在内部调用了`adder32`文件，并直接能够调用函数`adder32`，类似于C++中的类与对象的原理。
5. 考虑到4个数相加的进位可能性，我尝试了将原来的1位进位都修改为了两位进位，并在项目中进行修改与实践，使我对于项目代码有了更深入的了解。
6. 第一个实验原理并不是很难，文件的数量与关系也不复杂，但是刚入门时确实遇到了很多困难，比如说项目不知道如何运行，不知道如何接箱子，不知道怎么调节进位与调节输入数等等。经过两次课的实操，我已经大致对vivado的内容有了基本的了解，希望在接下来的实验当中能收获更多知识。