实验报告：四位二进制数的可控加法/减法FPGA设计

详细叙述设计方案和设计过程

设计方案

使用Vivado2017编程以及相应开发板，实现了基本的二进制加减法计算器的实现。

基本功能：输入两个四位二进制数，按下运算按钮后，以用数码管的方式显示运算的结果。

基本操作：

1. 数字输入操作：采用两个微动开关，分别代表数字0与数字1的输入按键，输入逻辑与普通计算器相同：逐个输入，即按下数字按钮，先前已经输入的数字会向前推进一位。
2. 运算操作：在输入完成第一个数字之后，摁下代表加法或者减法的按键，便可以开始输入第二个数字，在输入完第二个数字后，摁下代表运算（count）的按键，便可以在数码管上显示计算结果。
3. 清零操作：在一次计算完成后，或者是输入了错误的数字，需要摁下代表清零（clr）的按键，计算器会立即重置，开始等待第一个数字的输入。

显示：

1. 在摁下数字输入按键时，后四位数码管会实时显示已经输入的数字。如果输入的数字超出了四位，则会继续向前推进，只取后四位进行运算。
2. 在摁下运算（count）按键时，后四位数码管会立刻显示计算结果。如果运算结果大于或小于四位二进制数能够显示的最大范围（即0-15），会在最后一个数码管显示E。

设计过程

> 基本方法：采用迭代的方法，逐步加入功能。

功能实现的简要描述

计算器是存在三种状态的状态机：

1. 输入第一个数字的状态：状态变量state = 1

> 在这个状态下输入数字，能存入number1

2. 输入第二个数字的状态：状态变量state = 0

> 在这个状态下输入数字，能存入number1

3. 显示结果的状态

> 在这个状态下显示运算结果

状态切换

- 从任意状态切换至状态一：使用clr按键

> 将number1和number2清零，state=1

- 从状态一切换至状态二：使用加法按键add或减法按键sub

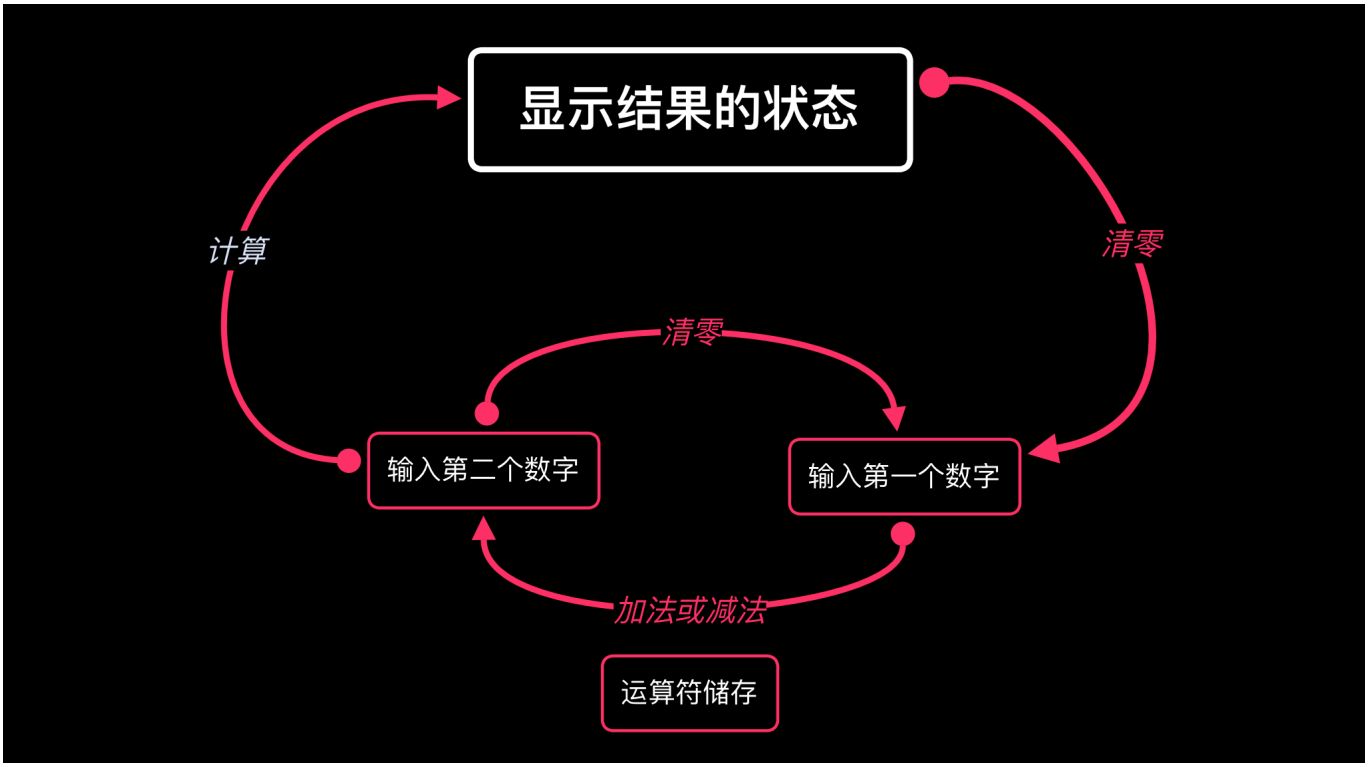
> 将运算法则储存在func变量中

- 从状态二切换至显示结果的状态：使用计算按键count

> 根据func变量的值，对number1和number2进行加法或减法

详细设计过程间代码注释

画出完整的顶层逻辑设计图



写出每一个文本文件的源程序

> binaryAdder.v 程序文件

```
module D0_display(
    input [3:0] D0_bits ,
    input [3:0] D0_NUM ,
    output reg [6:0] D0_a_to_g ,
    output [3:0] D0_led_bits
    );

    assign D0_led_bits = D0_bits ;

    always @(*)
    begin
        case(D0_NUM)
        0:D0_a_to_g=7'b1111110;
        1:D0_a_to_g=7'b0110000;
```

```
            'hE: D0_a_to_g=7'b1001111;
            default: D0_a_to_g=7'b0000000;
            endcase
        end
    endmodule

    module Count_FFFF(
        input clk,
        input clr, //清零按键
        input one, //按键1
        input zero, //按键0
        input add, //按键加
        input sub, //按键减
        input count, //按键计算
        output [6:0] a_to_g , //数码管显示内容
        output [3:0] led_bits //
        );
        reg [3:0] num ;//数码管单次显示的字符
        reg [35:0] ckl_cnt; //时钟
        reg [3:0] t_led_bits ;//选择led的显示位
        reg [3:0] number1 = 0; //储存数字一
        reg [3:0] number2 = 0; //储存数字二
        reg [3:0] value = 0; //数码管上实时显示的数值
        reg state = 1; //状态（为一输入第一个数字，为零输入第二个数字
        reg func = 0; //运算法则（为一进行加法，为零进行减法
        reg flag = 1; //是否已经储存这一次输入的标志

        always@(posedge clk)
        begin
            ckl_cnt = ckl_cnt + 1 ;//

            if(state)//如果是状态值是1，等待第一个数字的输入
            begin
                //如果按键1被按下
                if(one)
                begin
                    if(flag)//标志为一，说明这一次输入还没有储存
                    begin
                        number1[3] = number1[2];
                        number1[2] = number1[1];
                        number1[1] = number1[0];
                        number1[0] = 1; //将已经储存的数字向前移动一位
                        value = number1;
                        flag = 0;//储存完毕，将标志置零
                    end
                end
                //如果按键0被按下
                else if(zero)
                begin
                    if(flag)
                    begin
                        number1[3] = number1[2];
                        number1[2] = number1[1];
                        number1[1] = number1[0];
```

```
                number1[0] = 0;
                value = number1;
                flag = 0;
             end
        end
        //两个按键都是弹起状态时，将标志置1，可以进行下一次输入
        else flag = 1;
    end

        else//同理，状态是0，等待第二个数字的输入
    begin
        if(one)
        begin
            if(flag)
            begin
                number2[3] = number2[2];
                number2[2] = number2[1];
                number2[1] = number2[0];
                number2[0] = 1;
                value = number2;
                flag = 0;
            end
        end
        else if(zero)
        begin
            if(flag)
            begin
                number2[3] = number2[2];
                number2[2] = number2[1];
                number2[1] = number2[0];
                number2[0] = 0;
                value = number2;
                flag = 0;
             end
        end
        else flag = 1;
    end

    if(add)begin state = 0;func = 1;end
    //如果加法按键被按下，运算变量func置1，状态置0等待输入第二个数字
    else if(sub)begin state = 0; func = 0; end
    //如果加减法按键被按下，运算变量func置0，状态置0等待输入第二个数字
    else if(count)
    //如果运算按钮按下
        begin
        if (func) value = number1 + number2; //运算变量func为1，进行加法
        else value = number1 - number2;//否则进行减法
    end
    else if(clr) //如果清零按钮按下
    begin
        state = 1; //等待输入第一个数字
        value = 0; //数值置零
        number1 = 0;
        number2 = 0;
```

```
        end
    end

    always@(*) //将结果数字的四位分别显示
        case(ckl_cnt[15:14])
        0:begin num <= value[0]; t_led_bits <= 4'b0001;end
        1:begin num <= value[1];t_led_bits <= 4'b0010;end
        2:begin num <= value[2];t_led_bits <= 4'b0100;end
        3:begin num <= value[3];t_led_bits <= 4'b1000;end
        endcase

    D0_display
myD0_display(.D0_bits(t_led_bits),.D0_NUM(num),.D0_a_to_g(a_to_g),.D0_led_
bits(led_bits)) ;
endmodule
```

ee.xdc 引脚定义文件

```
set_property PACKAGE_PIN D4 [get_ports {a_to_g[6]}]
set_property PACKAGE_PIN E3 [get_ports {a_to_g[5]}]
set_property PACKAGE_PIN D3 [get_ports {a_to_g[4]}]
set_property PACKAGE_PIN F4 [get_ports {a_to_g[3]}]
set_property PACKAGE_PIN F3 [get_ports {a_to_g[2]}]
set_property PACKAGE_PIN E2 [get_ports {a_to_g[1]}]
set_property PACKAGE_PIN D2 [get_ports {a_to_g[0]}]
set_property PACKAGE_PIN G1 [get_ports {led_bits[3]}]
set_property PACKAGE_PIN F1 [get_ports {led_bits[2]}]
set_property PACKAGE_PIN E1 [get_ports {led_bits[1]}]
set_property PACKAGE_PIN G6 [get_ports {led_bits[0]}]
set_property PACKAGE_PIN P17 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports {a_to_g[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_to_g[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_to_g[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_to_g[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_to_g[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_to_g[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a_to_g[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_bits[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_bits[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_bits[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_bits[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clr]
set_property PACKAGE_PIN V1 [get_ports one]
set_property PACKAGE_PIN R17 [get_ports sub]
set_property IOSTANDARD LVCMOS33 [get_ports add]
set_property IOSTANDARD LVCMOS33 [get_ports count]
set_property IOSTANDARD LVCMOS33 [get_ports one]
set_property IOSTANDARD LVCMOS33 [get_ports sub]
set_property IOSTANDARD LVCMOS33 [get_ports zero]
set_property PACKAGE_PIN U4 [get_ports add]
set_property PACKAGE_PIN R15 [get_ports count]
```

```
set_property PACKAGE_PIN R11 [get_ports zero]
set_property PACKAGE_PIN R1 [get_ports clr]
```

详细叙述调试中所碰到的问题及解决方案

困难一：在分配clear变量和count变量的引脚时出现错误

```
[Place 30-876] Port 'clr'  is assigned to PACKAGE_PIN 'R15'
which can only be used as the N side of a differential clock input.
Please use the following constraint(s) to pass this DRC check:
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clr_IBUF}]
```

```
[Place 30-574] Poor placement for routing between an IO pin and BUFG.
If this sub optimal condition is acceptable for this design,
you may use the CLOCK_DEDICATED_ROUTE constraint in the .xdc file to
demote this message to a WARNING.
However, the use of this override is highly discouraged.
These examples can be used directly in the .xdc file to override this
clock rule.
        < set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets switch_IBUF]
>

        switch_IBUF_inst (IBUF.O) is locked to IOB_X0Y49
         and switch_IBUF_BUFG_inst (BUFG.I) is provisionally placed by
clockplacer on BUFGCTRL_X0Y16
```

问题解决：不可以将clr等变量放在always@的参数列表里，统一把函数定义在时钟函数里

困难二：遇到出错syntax error near else

原因排查：对begin -- end使用情况不够熟悉，导致if -- else语句出错。

verilog 中 begin -- end 用法就是一个模块的起始和结束的标记 在 verilog 中，begin -- end 就是一个模块 （相当于C语言的程序块）的起始和结束的标记。 非常类似于C语言中的大括号: {} 例如：每一个 always 模块，都需要有 begin -- end 来确定起始和结束。

```
always @(*)
begin
    ......
end
```

除了always语句外，分支语句if~else、case，以及每一个模块都需要 begin -- end 来确定起始和结束位置。

试验后的心得

在使用开发板和vivado进行FPGA进行开发时，因为此前的新生杯比赛，我有一定的arduino的开发经验，开发的思路是大致相同的。

二者的相同之处在于都需要结合硬件进行编程，不仅要考虑代码本身语法的正确性，还需要符合硬件运行的规律。

二者的最大不同之处，也就是我进行FPGA开发时的最大感受是，arduino的硬件连接是比较隐式的，而FPGA与硬件结合的更为紧密，在语法上也有了更多要求，有对信号上升沿，下降沿的判断，有以二进制变量为基础的总线型变量和寄存器变量，能感觉到FPGA的开发更加接近机器语言，而arduino或者是c语言则更为抽象。

学习不同层次的计算机语言对我们理解计算机的工作细节，理解程序的运作都有巨大的作用。经过这次的开发实践，我不仅有了更多硬件开发的经验，也对计算机语言的运作有了更深刻的理解，我相信这对未来我的学习工作会有很大的促进作用。