浙江大學

《EDA技术应用》实验报告

项目名称：四位 LED 加减法计数器

姓　　名：

学　　号：

学　　院：生物医学工程与仪器科学学院

专　　业：生物医学工程

指导老师：黄添添、裘利坚

2026 年 4 月 20 日

浙江大学 实验报告

课程名称：EDA 技术应用　实验类型：EDA 实验　指导老师：黄添添、裘利坚

实验名称：四位 LED 加减法计数器

姓　　名：　　学　　号：　　专　　业：生物医学工程

实验地点：玉泉教六 204　日　　期：2026 年 4 月 20 日

目　录

一、实验目的

(1) 了解 Quartus II 开发环境，熟悉 Verilog 语言基本语法
(2) 实现 Led 灯表示四位计数器按键加法功能
(3) 修改程序实现 Led 灯表示四位计数器按键减法功能
(4) 修改程序实现 Led 灯表示十进制四位计数器减法功能

二、实验内容

根据实验资料中提供的 Dip_Led 工程，学习实现按键加法和减法功能。修改程序实现 Led 灯表示十进制四位计数器减法功能。

三、主要仪器设备

Quartus ii 13.1, DE2-115 开发板。

四、操作方法和实验步骤

这部分内容将说明每一部分代码的实现逻辑，并在这部分的末尾展现整体的代码。

4.1 外部异步信号同步化处理

实际物理按键（PBSwitch）和拨码开关（DIPSwitch）的电平翻转是随机发生的，与开发板内部 50MHz 的系统时钟并不完全对齐。如果直接将异步信号引入核心逻辑，极易产生亚稳态问题。因此，代码中采用了双寄存器打拍的方法（`PBSwitch_flop1` 与 `PBSwitch_flop2`）。这种设计能够有效地将外部异步信号同步到本地时钟域中，不仅提高了系统的稳定性，还为后续的边沿检测提供了当前状态和历史状态的对比依据。

```
always @ ( posedge clk or negedge rst ) begin
        if ( !rst ) begin
            PBSwitch_flop1 <= 1'b1;            // 复位时按键默认处于未按下状态（高电平）
            PBSwitch_flop2 <= 1'b1;            // 历史状态同步复位
            DIPSwitch_flop1 <= 2'b00;          // 复位时拨码开关状态寄存器清零
            DIPSwitch_flop2 <= 2'b00;          // 历史状态同步复位
        end
        else begin
            // 消除亚稳态，将外部异步按键信号同步到系统时钟域
            PBSwitch_flop1 <= PBSwitch;        // 采样当前物理按键电平
            PBSwitch_flop2 <= PBSwitch_flop1; // 寄存上一周期的采样值

            // 将外部两位拨码开关信号同步到系统时钟域
            DIPSwitch_flop1 <= DIPSwitch;      // 采样当前物理拨码开关电平
            DIPSwitch_flop2 <= DIPSwitch_flop1; // 寄存上一周期的采样值
        end
    end
```

4.2 按键去抖设计

机械按键在闭合与断开的瞬间不可避免地会产生毫秒级的机械抖动。本设计采用软件去抖逻辑。通过对比 `PBSwitch_flop1` 与 `PBSwitch_flop2`，精准捕捉按键释放的瞬间。一旦检测到电平跳变，立刻为 count 寄存器赋初值 `16'h61A8`（十进制 25000）。在 50MHz 的时钟驱动下，这相当于启动了一个约 0.5ms 的延时屏蔽窗口。在倒计时归零前，系统将忽略高频的电平抖动，确保按键状态的稳定。

```
always @ ( posedge clk or negedge rst ) begin
        if ( !rst ) begin
            count <= 16'h0;                         // 复位时，延时计数器清零
        end
        else if ( PBSwitch_flop1 ) begin            // 检测到当前按键处于未按下状态（高电平）
            if ( !PBSwitch_flop2 ) begin            // 且上一周期按键处于按下状态，说明发生了按键释放的边沿跳变
                count <= 16'h61A8;                  // 触发消抖延时，重置倒计时器为25000（约0.5ms延迟）
            end
            else if ( count == 16'h0 ) begin        // 如果延时已经结束（减至0）
                count <= count;                     // 维持0值，避免非预期的计数器下溢出
            end
            else begin                              // 如果延时仍在进行中
                count <= count - 1'b1;              // 延时计数器每个时钟周期递减1
            end
        end
        else begin                                  // 如果按键处于按下稳定状态
            count <= 16'h0;                         // 保持计数器处于清零复位状态
        end
    end
```

4.3 边沿检测与单周期有效脉冲提取

为了防止按键长按导致计数器连续多次更新，系统必须将宽幅的按键电平信号转换为窄带脉冲。代码通过判断延时计数器 count 减至 `16'h0001` 的那一特定时钟周期，输出一个仅维持 20ns（单时钟周期）的高电平信号 `PBSwitch_valid`。核心业务逻辑仅在这个极短的触发沿到来时才执行一次运算。

```
always @ ( posedge clk or negedge rst ) begin
        if ( !rst ) begin
            PBSwitch_valid <= 1'b0;                 // 复位时，有效脉冲标志位清零
        end
        else if ( count == 16'h0001) begin          // 当延时倒计时器刚好减至1的特定时钟周期
            PBSwitch_valid <= 1'b1;                 // 产生一个单时钟周期的高电平脉冲，表示一次有效按键动作
        end
        else begin                                  // 其他所有周期状态下
            PBSwitch_valid <= 1'b0;                 // 强制有效脉冲标志位保持低电平，防止多重触发
        end
    end
```

4.4 计数器的更新加减法实现

计数器更新需要在 PBSwitch_valid 有效时更新计数器，并且通过两个拨码开关（DIPSwitch 0 和 DIPSwitch 1）的组合，实现了四种不同工作模式的灵活切换。

开关 0（DIPSwitch[0]）负责切换进制： 状态为 0 时代表十六进制，状态为 1 时代表十进制。

开关 1（DIPSwitch[1]）负责切换加减： 状态为 0 时执行加法，状态为 1 时执行减法。

通过这两个开关的排列组合，刚好构成了以下四种具体的工作情况：十六进制加法、十六进制减法、十进制加法、十进制减法

在十进制模式下我加入了对非法数字的排错处理。如果在十六进制模式下将数字加到了 12，此时突然拨动开关切入十进制模式，12 对十进制而言是乱码。代码逻辑会立刻识别出这个大于 9 的非法状态，只要按下按键，就会自动将其纠正回合法的 0 或 9，确保系统运行稳定。

```
always @ ( posedge clk or negedge rst) begin
        if ( !rst ) begin
            led_reg <= 4'b0000;                 // 异步复位，计数值初始化为0，所有LED熄灭
        end
        else if ( PBSwitch_valid ) begin        // 仅在接收到有效的单周期按键脉冲时更新状态

            // 判断拨码开关第0位：决定进制模式
            if ( DIPSwitch_flop2[0] == 1'b0 ) begin

                // 模式1：十六进制计数模式
                // 判断拨码开关第1位：决定加法或减法
                if ( DIPSwitch_flop2[1] == 1'b0 ) begin
                    led_reg <= led_reg + 1'b1; // 执行二进制加法逻辑，依靠寄存器自然溢出实现循环
                end
                else begin
                    led_reg <= led_reg - 1'b1; // 执行二进制减法逻辑，依靠寄存器自然下溢实现循环
                end

            end
            else begin

                //  模式2：十进制计数模式
                // 判断拨码开关第1位：决定加法或减法
                if ( DIPSwitch_flop2[1] == 1'b0 ) begin
                    // 十进制加法逻辑
                    // 边界保护：若当前值达最大值9，或因模式切换出现非法BCD码（>9），则强制归零
                    if ( led_reg >= 4'd9 ) begin
                        led_reg <= 4'd0;
                    end
                    else begin
                        led_reg <= led_reg + 1'b1; // 正常十进制加1
                    end
                end
                else begin
                    // 十进制减法逻辑
                    // 边界保护：若当前值为0，或因模式切换出现非法BCD码（>9），则强制跳转至最大值9
                    if ( led_reg == 4'd0 || led_reg > 4'd9 ) begin
                        led_reg <= 4'd9;
                    end
                    else begin
                        led_reg <= led_reg - 1'b1; // 正常十进制减1
                    end
                end

            end // 进制模式选择结束

        end // 脉冲有效判断结束
    end // always块结束
```

4.5 完整代码

```
module Dip_Led(
    input clk,                      // 50MHz系统时钟输入
```

```
    input rst,                      // 异步复位信号输入，低电平有效
    input PBSwitch,                 // 物理按键输入，用于触发计数脉冲
    input [1:0] DIPSwitch,          // 两位拨码开关输入：[0]位控制进制，[1]位控制加减法
    output [3:0] led                // 四位LED输出，用于直观显示当前计数值
);

    // 按键信号同步与打拍寄存器
    reg PBSwitch_flop1;             // 存储当前时钟周期的按键状态
    reg PBSwitch_flop2;             // 存储上一时钟周期的按键状态
    reg PBSwitch_valid;             // 单周期有效按键脉冲输出标志位

    // 拨码开关信号同步与打拍寄存器,判断拨码开关第0位决定进制模式,判断拨码开关第1位决定加法或减法
    reg [1:0] DIPSwitch_flop1;      // 存储当前时钟周期的拨码开关状态
    reg [1:0] DIPSwitch_flop2;      // 存储上一时钟周期的拨码开关状态

    // 消抖与业务逻辑寄存器
    reg [15:0] count;               // 16位宽延时计数器，用于按键消抖计时
    reg [3:0] led_reg;              // 4位核心业务寄存器，存储实际的计数值

    // 将内部寄存器状态直接映射到外部物理引脚
    assign led[3] = led_reg[3];   // 连接最高位LED
    assign led[2] = led_reg[2];   // 连接次高位LED
    assign led[1] = led_reg[1];   // 连接次低位LED
    assign led[0] = led_reg[0];   // 连接最低位LED


    // 模块一：外部异步信号同步化处理

    always @ ( posedge clk or negedge rst ) begin
        if ( !rst ) begin
            PBSwitch_flop1 <= 1'b1;             // 复位时按键默认处于未按下状态（高电平）
            PBSwitch_flop2 <= 1'b1;             // 历史状态同步复位
            DIPSwitch_flop1 <= 2'b00;           // 复位时拨码开关状态寄存器清零
            DIPSwitch_flop2 <= 2'b00;           // 历史状态同步复位
        end
        else begin
            // 消除亚稳态，将外部异步按键信号同步到系统时钟域
            PBSwitch_flop1 <= PBSwitch;         // 采样当前物理按键电平
            PBSwitch_flop2 <= PBSwitch_flop1; // 寄存上一周期的采样值

            // 将外部两位拨码开关信号同步到系统时钟域
            DIPSwitch_flop1 <= DIPSwitch;       // 采样当前物理拨码开关电平
            DIPSwitch_flop2 <= DIPSwitch_flop1; // 寄存上一周期的采样值
        end
    end


    // 模块二：按键消抖延时控制

    always @ ( posedge clk or negedge rst ) begin
        if ( !rst ) begin
            count <= 16'h0;                     // 复位时，延时计数器清零
        end
        else if ( PBSwitch_flop1 ) begin        // 检测到当前按键处于未按下状态（高电平）
            if ( !PBSwitch_flop2 ) begin        // 且上一周期按键处于按下状态，说明发生了按
键释放的边沿跳变
                count <= 16'h61A8;              // 触发消抖延时，重置倒计时器为25000（约
0.5ms延迟）
            end
            else if ( count == 16'h0 ) begin  // 如果延时已经结束（减至0）
                count <= count;                 // 维持0值，避免非预期的计数器下溢出
            end
            else begin                          // 如果延时仍在进行中
                count <= count - 1'b1;          // 延时计数器每个时钟周期递减1
```

```
65              end
66          end
67          else begin                                  // 如果按键处于按下稳定状态
68              count <= 16'h0;                         // 保持计数器处于清零复位状态
69          end
70      end
71
72
73      // 模块三：边沿检测与单周期有效脉冲提取
74
75      always @ ( posedge clk or negedge rst ) begin
76          if ( !rst ) begin
77              PBSwitch_valid <= 1'b0;                 // 复位时，有效脉冲标志位清零
78          end
79          else if ( count == 16'h0001) begin          // 当延时倒计时器刚好减至1的特定时钟周
    期
80              PBSwitch_valid <= 1'b1;                 // 产生一个单时钟周期的高电平脉冲，表示一次
    有效按键动作
81          end
82          else begin                                  // 其他所有周期状态下
83              PBSwitch_valid <= 1'b0;                 // 强制有效脉冲标志位保持低电平，防止多重触
    发
84          end
85      end
86
87
88      // 模块四：算术逻辑运算与状态控制（多模式支持）
89
90      always @ ( posedge clk or negedge rst) begin
91          if ( !rst ) begin
92              led_reg <= 4'b0000;                     // 异步复位，计数值初始化为0，所有LED熄灭
93          end
94          else if ( PBSwitch_valid ) begin            // 仅在接收到有效的单周期按键脉冲时更新状态
95
96              // 判断拨码开关第0位：决定进制模式
97              if ( DIPSwitch_flop2[0] == 1'b0 ) begin
98
99                  // 模式1：十六进制计数模式
100                 // 判断拨码开关第1位：决定加法或减法
101                 if ( DIPSwitch_flop2[1] == 1'b0 ) begin
                        led_reg <= led_reg + 1'b1; // 执行二进制加法逻辑，依靠寄存器自然溢出
    实现循环
103                 end
104                 else begin
                        led_reg <= led_reg - 1'b1; // 执行二进制减法逻辑，依靠寄存器自然下溢
    实现循环
106                 end
107
108             end
109             else begin
110
111                 // 模式2：十进制计数模式
112                 // 判断拨码开关第1位：决定加法或减法
113                 if ( DIPSwitch_flop2[1] == 1'b0 ) begin
114                     // 十进制加法逻辑
115                     // 边界保护：若当前值达最大值9，或因模式切换出现非法BCD码（>9），则强制归零
116                     if ( led_reg >= 4'd9 ) begin
117                         led_reg <= 4'd0;
118                     end
119                     else begin
120                         led_reg <= led_reg + 1'b1; // 正常十进制加1
121                     end
122                 end
```

```
                    else begin
                        // 十进制减法逻辑
                        // 边界保护：若当前值为0，或因模式切换出现非法BCD码（>9），则强制跳转至最大值9
                        if ( led_reg == 4'd0 || led_reg > 4'd9 ) begin
                            led_reg <= 4'd9;
                        end
                        else begin
                            led_reg <= led_reg - 1'b1; // 正常十进制减1
                        end
                    end

                end // 进制模式选择结束

            end // 脉冲有效判断结束
        end // always块结束

endmodule
```

五、实验结果与分析

将代码编译、经过 pin planner 之后烧录到开发板上能够实现实验要求的结果。在实际操作中，通过拨动分配好的两个物理拨码开关（SW1,SW0），系统能够完美实现四种状态的切换，代码功能正确完整。

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair
DIPSwitch[1]	Input	PIN_AC28	5	B5_N2	PIN_AC28	2.5 V (default)		8mA (default)		
DIPSwitch[0]	Input	PIN_AB28	5	B5_N1	PIN_AB28	2.5 V (default)		8mA (default)		
PBSwitch	Input	PIN_M21	6	B6_N1	PIN_M21	2.5 V (default)		8mA (default)		
clk	Input	PIN_Y2	2	B2_N0	PIN_Y2	3.3-V LVTTL		8mA (default)		
led[3]	Output	PIN_E24	7	B7_N1	PIN_E24	2.5 V (default)		8mA (default)	2 (default)	
led[2]	Output	PIN_E25	7	B7_N1	PIN_E25	2.5 V (default)		8mA (default)	2 (default)	
led[1]	Output	PIN_E22	7	B7_N0	PIN_E22	2.5 V (default)		8mA (default)	2 (default)	
led[0]	Output	PIN_E21	7	B7_N0	PIN_E21	2.5 V (default)		8mA (default)	2 (default)	
rst	Input	PIN_M23	6	B6_N2	PIN_M23	2.5 V (default)		8mA (default)		

图 5.1 – pin planner

5.1 十六进制加法验证

当两个拨码开关均拨至低电平时，系统进入十六进制加法模式。初始状态下四个 LED 灯全灭（0000）。每按一次按键，LED 灯代表的二进制数值加 1。当连续按键使得四个 LED 灯全亮（达到最大值 1111，即 15）时，再次按下按键，四个 LED 灯瞬间全灭（归零为 0000），开始新的加法循环。

5.2 十六进制减法验证

保持控制进制的开关 0 为低电平，将控制加减法的开关 1 拨至高电平。系统进入十六进制减法模式。当 LED 状态为全灭（0000）时，按下按键，四个 LED 灯同时亮起（跳变为 1111，即 15），随后每按一次按键，数值正常递减。

5.3 十进制加法验证

将开关 0 拨至高电平（切换为十进制），开关 1 拨回低电平（加法）。按键每按一次，LED 数值加 1。当灯光状态显示为 1001（即十进制的 9）时，再次按下按键，LED 变为（0000）。

5.4 十进制减法验证

将两个开关均拨至高电平。在 LED 全灭（0000）的状态下按下按键，LED 灯光状态瞬间跳变为 1001（即十进制的最大有效值 9）。随后每次按键，数值均在 9 到 0 之间正常递减循环。

具体开发板运行结果的视频内容无法在报告中展示，实验内容已经交给助教验收。如下图片仅作开发板运行后的展示。

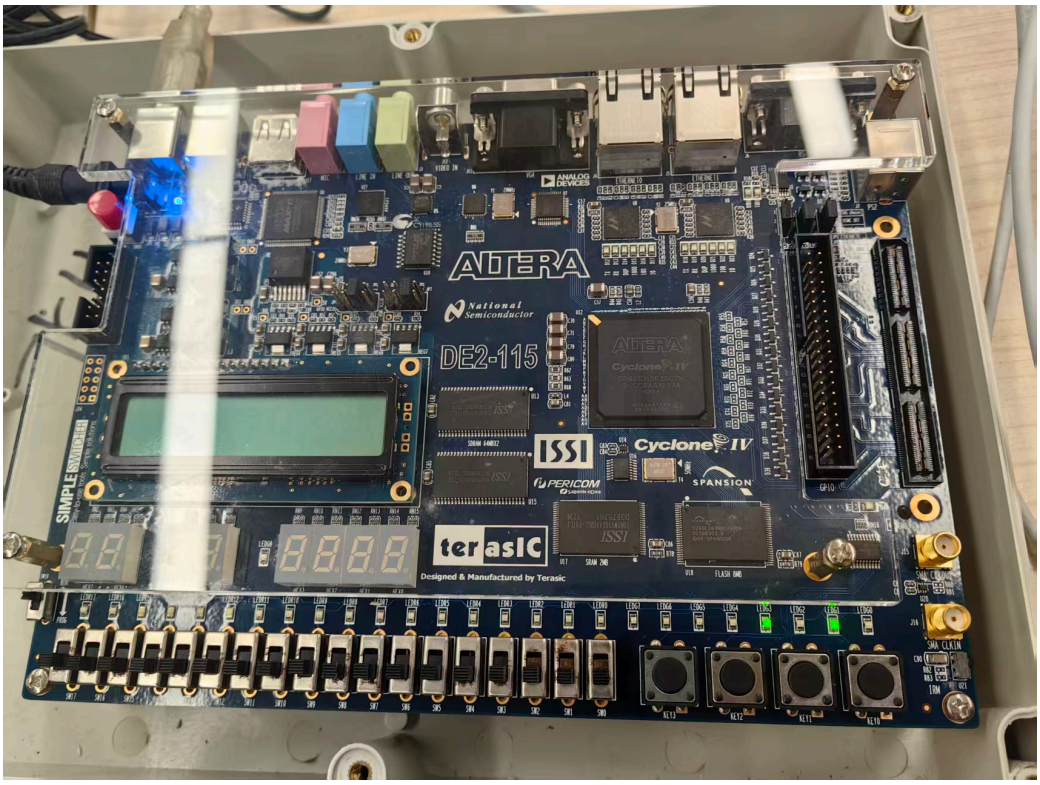

图 5.2 – 开发板运行结果

六、讨论与心得

从最初实验一的简单模块调用，到本次独立设计并完善一个包含信号同步、按键消抖、边缘检测和多模式状态控制的完整硬件工程，我深刻体会到了硬件描述语言（Verilog）与传统纯软件编程的本质区别。

本次实验的核心难点之一在于处理物理按键的机械抖动。在理想的纯逻辑仿真中是不存在抖动概念的，但在将代码映射到真实的 DE2-115 开发板上时，如果不加入由 16 位计数器构成的延时屏蔽窗口，按键弹片的微小弹跳就会导致数值的多重触发。此处使用的是软件去抖方法，这在大规模按键去抖的时候比较合适。当按键规模不大时，也可以选择硬件的方式去抖，比如使用 RS 锁存去抖。

在本次实验开始时，为了分别完成实验要求中“十六进制减法”和“十进制减法”的独立要求，我编写了两个完全分离的程序，每次验证不同的功能都需要重新全编译并向开发板烧录一次文件件。后来经过助教老师的指导，通过引入开发板上的两个拨码开关作为输入状态选择信号，我将多套算术逻辑整合进同一个主控模块中，仅用一套代码就实现了一个多功能计数器。这个过程让我深刻理解了硬件设计中资源复用的理念，即通过改变控制信号的组合，就能让同一块物理电路展现出截然不同的功能。

并且在更改代码过程中，我进一步思考并解决了跨模式切换可能带来的非法数值问题。如果在十六进制加法模式下计数到了 10 到 15 之间的数值，此时用户若直接拨动开关进入十进制模式，就会产生非法的 BCD 码，导致系统后续的加减法逻辑发生错乱。为此，我在十进制的加减法判断分支中加入了严密的边界校验（遇到大于 9 的数值则在下一次按键时强制归零或跳回最大合法值 9）。

总而言之，本次实验不仅锻炼了我的 Verilog 代码编写与 Debug 能力，更重要的是帮我完成了从软件顺序执行到硬件并行与时序控制的思维跨越。