華中科技大學

单片机课程结课设计

简易数控直流电源设计

院　　系　电子信息与通信学院

专业班级　信卓 2201 班

姓　　名　董浩

学　　号　U202213781

指导教师　肖波

2023 年 9 月 29 日

摘　　要

本设计制作的可调节数控电源系统，可以输出稳定的电压，通过按键进行精度为 0.1V 的调节，并在 LCD1602 屏幕和串口上显示当前电压值和电压设置值。系统由 LCD1602、LM117、ADC（开发板内置）、DAC（开发板内置）、和 STM32F407ZGT6 开发板等部分组成。DAC 输出按键设置的电压，通过 LM117 电压调节电路输出电压，ADC 采集电压信息，将信息通过串口和 LCD1602 输出，同时进行 PID 调节稳定电压。

关键词： 可调电源; 串口调节; STM32F407ZGT6; LM117; LM358

目　　录

1 设计要求

1.1 具体要求

- 可使用按键设置电压；
- PC 机可通过串口向数控电源发送指令，设置电压；
- 数控电源通过串口每 2 秒钟向发送 1 次当前电压值；
- 输出电压范围 0 至 +9.9V，步进 0.1V；
- 最大输出电流：200mA；
- 在 1602 液晶屏上显示当前电压值，以及电压设置值；
- 使用洞洞板焊接核心电路；
- 输出端预留一个电源座或接线柱以便挂接负载电阻。

1.2 设计框图

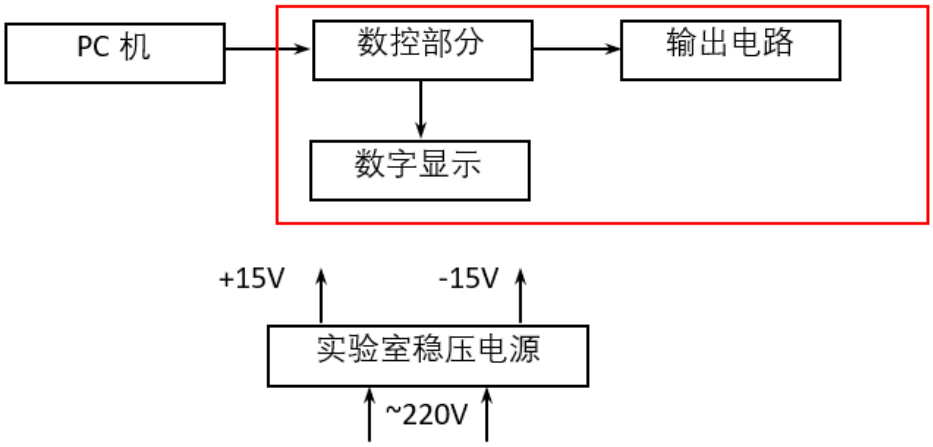

图 1-1 设计框图

2 设计思路

2.1 系统框图

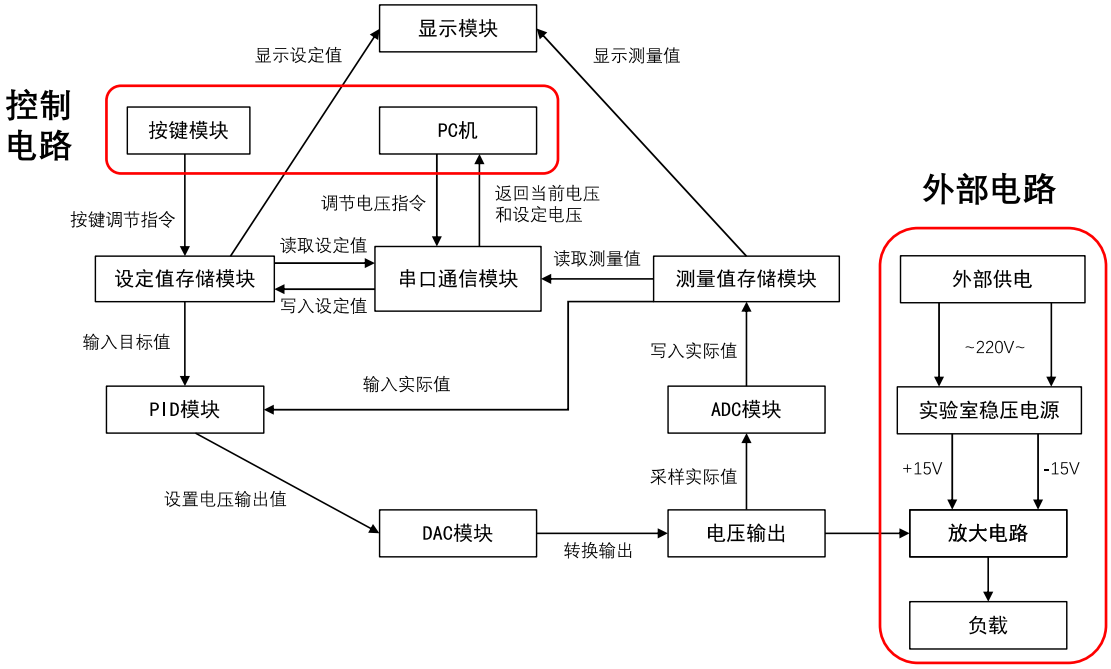

图 2-1 实现框图

2.2 核心原理图

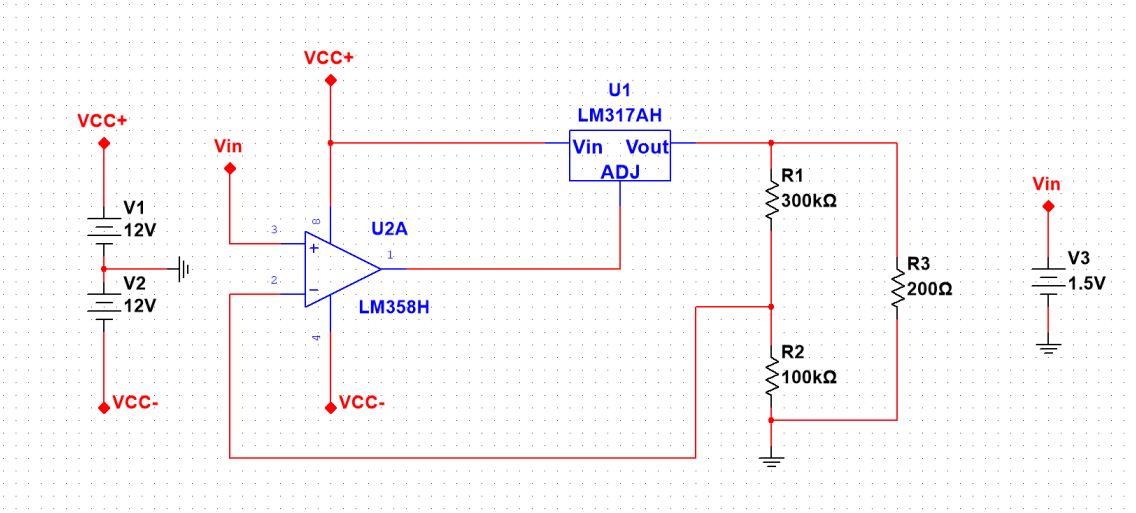

图 2-2 原理图

2.3 方案描述

2.3.1 电路部分

使用 LM358 运算放大器和 LM317 稳压器制作输出电压可变的稳压电源。LM358 使用 ±12V 直流恒压电源供电，LM317 时钟 +12V 直流恒压电源供电，均设置限流 0.50A。使用单片机 DAC 输出的电压作为控制电压，经过 LM358 和 LM317 处理后输出放大后的电压，为负载供电。电源、单片机和放大电路共地。

参数的选择方面，根据 LM317 的特性：

$$V_{OUT} = V_{REF}(1 + \frac{R2}{R1} + I_{ADJ}R2)$$

以及 V_{OUT} 和 ADJ 之间 1.25V 的标准参考电压，结合 LM358 的工作原理，确定 $R1 = 300\Omega,\ R2 = 100\Omega$，从而控制输出电压与输入电压的比值为 4。

2.3.2 单片机部分

设计思路

总体思路为使用按键和串口通信更改存储的设定值，将设定值输入 PID 模块，生成电压输出值。再经过 DAC 模块输出电压，传递给外部电路。同时 ADC 模块对输出电压进行采样，写入测量值存储模块，进而输入到 PID 模块，达到闭环调节的目的。通信方面，PC 机与单片机之间使用 UART 串口通信，串口通信模块读取设定值和测量值，将结果发送给 PC 机，从而在 PC 机上显示。单片机通过接收 PC 机发送的电压调节指令，更改设定值，从而达到使用 PC 机设置电压的目的。

硬件资源分配

本项目使用的硬件资源如下：

- systick：用于 delay 函数的延时
- KEY0、KEY1：用于调节电压值
- TIM3：用于串口定时输出的计时
- EXTI line8 and line9：用于配置按键外部中断
- ADC1_CH5：用于采样电压
- DAC1：用于输出指定电压
- USART1：用于和 PC 机进行串口通信

使用到的外部 IO 如下：

- PA4：DAC1 的输出引脚
- PA5：ADC1_CH5 的输入引脚
- PA9：UART 的 TX 引脚
- PA10：UART 的 RX 引脚
- PB8：开发板上自带的 KEY1
- PB9：开发板上自带的 KEY0
- PD8：连接 LCD 显示屏的 RS 极
- PD9：连接 LCD 显示屏的 RW 极
- PD10：连接 LCD 显示屏的 EN 极
- PF0-10：LCD 显示屏数据输入

2.4 小组分工

本设计由三人合作完成，小组分工如下：

- 董浩：主程序框架、通信指令、ADC、DAC、按键控制、电路仿真、系统组装、电路调试
- 董星星：显示屏模块代码、ADC 采样部分、串口通信
- 雷雨田：PID 算法代码，电路设计、洞洞板焊接、电路调试

3　实现效果

使用 LTspice 进行仿真，设置负载 R 为 200Ω，在直流扫描模式下，对 Vin 进行扫描，起始值 0.1V，结束值 3.3V，步进 0.01V，收集测量值绘图如下：

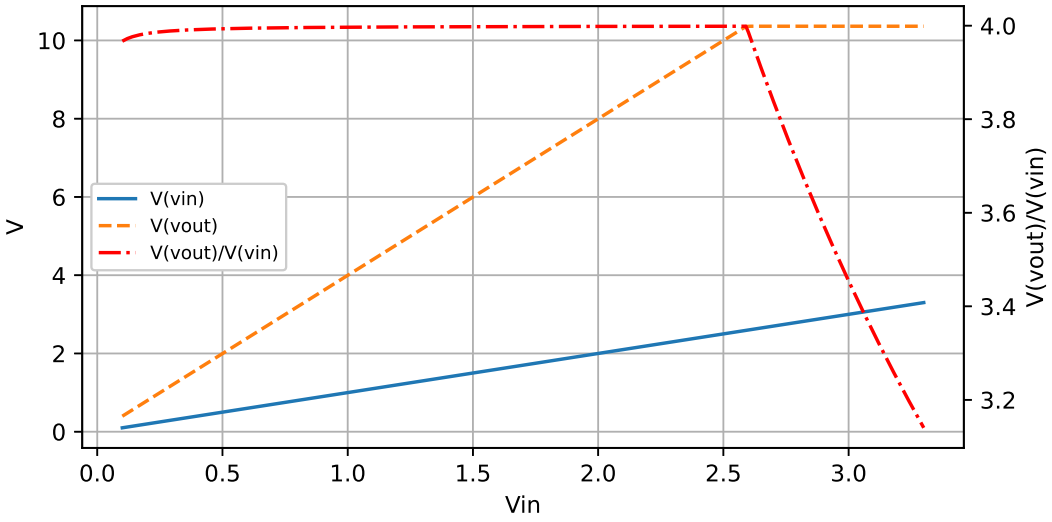

图 3-1　仿真曲线

由图可知，在 200Ω 的负载下，输出最大电压可达到 10V 以上，且输入电压在 0 到 2.5V 的区间内时，输出电压与输入电压的比值为 4，符合设计原理图时的预期。完成仿真后，制作使用洞洞板制作原理图所示电路。洞洞板、LCD1602 和单片机之间使用杜邦线相连，上电测试。系统工作正常，可以自由设定电压，输出指定电压，同时在 LCD 上显示设定电压和实际电压，并传输给 PC 主机，设计通过验收。

4 不足与改进

在实测中，发现电压调节不够迅速，带负载能力不够强以及电压存在 mV 级别的波动等问题，对有关问题进行分析，发现原因以及相应改进措施如下：

对于电压调节不够迅速，原程序在主函数的循环内不断调用 LCD 显示函数来刷新显示屏显示的数值，而显示函数中含有延时函数导致数据刷新和显示不及时。同时 PID 算法部分参数设置的值不够合理，导致电压变化过慢。针对这一问题，改进措施如下：使用定时器定时产生中断，产生中断时将标志位置 1，主程序的循环中检测到该标志位后执行刷新 LCD 命令并将标志位置 0，而从避免主程序不断刷新 LCD 导致电压调节不够迅速。

对于电压存在波动，采用在电路中添加滤波，去耦电容的方式，对输出进行整流，同时稳定电压。

针对带负载能力不够强，测试时发现对于 12Ω 以下的电阻，输出电压的最大值在 10V 以下，暂未得出有效解决方案，考虑更换性能更好的运放或改进电路。对原电路，使用 LTspice 仿真导出数据后绘图如下：

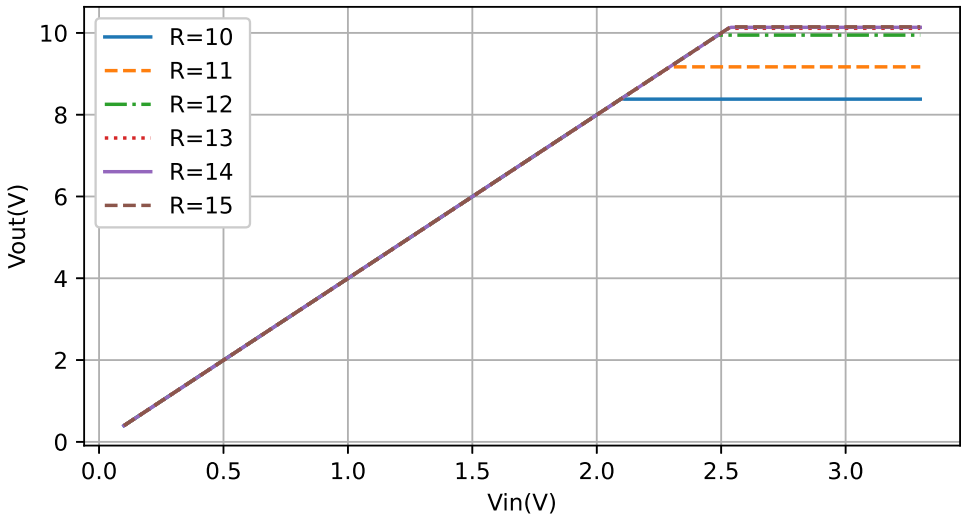

图 4-1 电阻扫描仿真曲线

附录

本次实验验收时用到的原始代码如下：

中断服务函数部分：

```
//配置按键中断服务函数
void EXTI9_5_IRQHandler(void)
{
    if (EXTI_GetITStatus(EXTI_Line9) != RESET) // KEY1 按下
    {
        // LED0 取反
        GPIO_ToggleBits(GPIOF, GPIO_Pin_9);
        target -= 100; // 设定值降低 0.1V
        printf(" 使用按键设定值为%.3fV\r\n", target / 1000.0);
        EXTI_ClearITPendingBit(EXTI_Line9);
    }
    else if (EXTI_GetITStatus(EXTI_Line8) != RESET) // KEY0 按下
    {
        // LED1 取反
        GPIO_ToggleBits(GPIOF, GPIO_Pin_10);
        target += 100; // 设定值增加 0.1V
        printf(" 使用按键设定值为%.3fV\r\n", target / 1000.0);
        EXTI_ClearITPendingBit(EXTI_Line8);
    }
}
// 定时器 3 中断服务函数
// 每隔两秒输出一次值
void TIM3_IRQHandler(void)
{
    u16 adcx;
    if (TIM_GetITStatus(TIM3, TIM_IT_Update) == SET)
```

```
        {
            adcx = Get_Adc_Average(ADC_Channel_5, 20); // 获取通道 5 的转换
            ↪ 值，20 次取平均
            measure = (float)adcx * (3.3 / 4096);          // 获取计算后的带小数的实
            ↪ 际电压值，比如 3.1111
            printf("AD 采样电压为%.3fV\r\n", measure);
            printf(" 测量电压为%.3fV\r\n", 4 * measure);
            printf(" 采样完成\r\n");
        }
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update); // 清除中断标志位
    }
```

控制算法部分：

```
    /*******pid.h*******/
    #ifndef _PID_H
    #define _PID_H

    typedef struct
    {
        double target_value;  // 设定的目标值及需要达到的最终值
        double current_value; // 当前值（可认为外部的反馈值）
        double CAL_value;    // 计算需要输出的值
        double sum_error;    // 累计的偏差值
        double error;        // 误差值
        double last_error;   // 上一次误差值
        double pre_error;    // 上上一次误差值（增量式 pid 中使用）
    } PID;

    double Velocity_FeedbackControl(double Targetvalue, double Currentvalue);
    extern PID pid;
```

```
#endif

/*******pid.c*******/
#include "pid.h"
#define Kp 0.10f
#define Ki 0.25f
#define Kd 0.00f

PID pid;
double output[100];

double Velocity_FeedbackControl(double Targetvalue, double Currentvalue)
{
    pid.last_error=pid.error;
    pid.error=Targetvalue-Currentvalue;
    pid.sum_error=pid.sum_error+(pid.error*Ki);
    pid.CAL_value=(pid.error-pid.last_error)*Kd+pid.error*Kp+pid.sum_error;
    return pid.CAL_value;
}
```

主程序部分：

```
#include "stm32f4xx.h"
#include "delay.h"
#include "LED.h"
#include "key.h"
#include "serial.h"
#include "DAC.h"
#include "time.h"
#include "ADC.h"
#include <string.h>
```

```
#include "pid.h"
#include "LCD.h"
/*
*****Attention***********
****target 使用 4 位整数 ****
例:1234 表示设定电压 1.234V
************************
*/
float measure = 0; // 电压测量值
u16 target = 1100; // 电压设定值
char target_str[5];
char measure_str[5];

int main(void)
{
    // 初始化部分
    Serial_Init(); // 波特率 115200
    delay_init(168);
    LED_Init();                                    // 测试 LED 初始化
    Dac1_Init();                                   // DAC1 输出初始化
    EXTI_Key_Config();                             // 按键 EXTI 中断配置
    Adc_Init();                                    // ADC1_CH5 初始化
    TIM3_Int_Init(20000 - 1, 8400 - 1); // TIM3 初始化，每隔 2 秒自动中断
    LCD_Init();                                    // LED 初始化
    LCD_Clear();                                   // 清屏
    pid.sum_error = 0;

    Dac1_Set_Vol(1100); // 设置 DAC 输出默认电压

    while (1)
```

```
    {
        if (Serial_GetRxFlag() == 1) // 接收到了数据
        {
            Serial_SendArray(Serial_RxPacket, 4);
            target = 1000 * Serial_RxPacket[0] + 100 * Serial_RxPacket[1] +
            ↪  10 * Serial_RxPacket[2] + Serial_RxPacket[3]; // 目标值

            printf(" 使用串口设定值为%.3fV\r\n", target / 1000.0);
        }

        double tg = (double)target;
        Velocity_FeedbackControl(tg, pid.CAL_value);
        u16 tg_ = (u16)pid.CAL_value / 4.0;
        Dac1_Set_Vol(tg_); // 设置电压值为 tg_

        sprintf(target_str, "%.3f", target / 1000.0);
        ↪       // 将 target 转换为字符串，存储到 targrt_str 中
        sprintf(measure_str, "%.3f", 4 * measure);       // 将 measure 转换为字
        ↪  符串，存储到 measure_str 中
        LCD_Display_Str(1, 1, "Target  :");
        LCD_Display_Str(2, 1, "Measure :");

        LCD_Display_Str(1, 12, target_str);
        LCD_Display_Str(2, 12, measure_str);
    }
}
```