

浙江大学

本科实验报告

数控恒流源设计

课程名称：	电子电路设计实验
姓 名：	箫宇 etc
学 院：	信息与工程学院
系：	
专 业：	信息工程
学 号：	
指导老师：	王子立、金向东、龚淑君

2023 年 7 月 2 日

浙江大学实验报告

专业： 信息工程
姓名： 箫宇 etc
学号：
日期： 2023 年 7 月 2 日
地点： 东 4-227

课程名称： 电子电路设计实验 指导老师： 王子立、金向东、龚淑君 成绩：
实验名称： 数控恒流源设计 实验类型： 设计实验 同组学生姓名：

一 系统方案

1. 系统总体框图

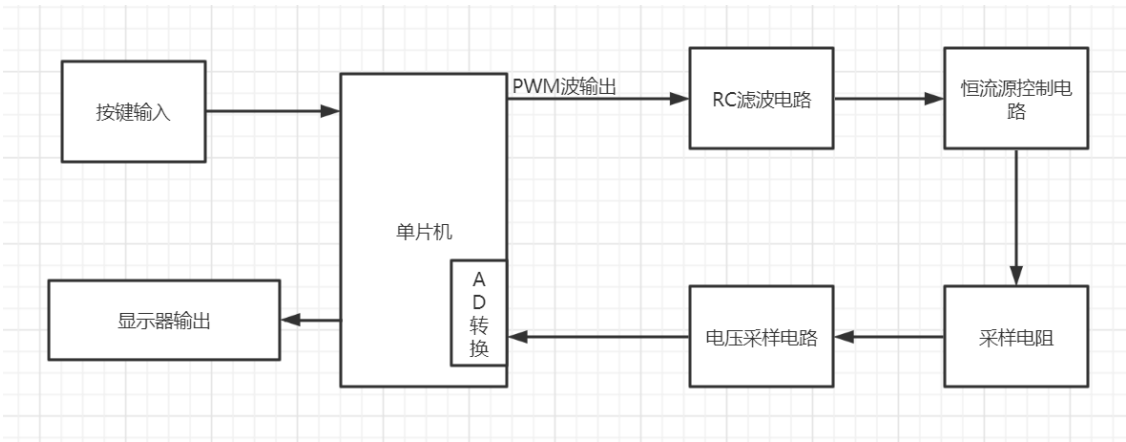


图 1: 系统总体框图

2. 方案比较与选择

2.1 采用开关电源的恒流源

采用开关电源的恒流源电路如图 1 所示。当电源电压降低或负载电阻 R_1 降低时，采样电阻 R_S 上的电压也将减少，则 SG3524 的 12、13 管脚输出方波的占空比增大，从而 BG1 导通时间变长，使电压 U_0 回升到原来的稳定值。BG1 关断后，储能元件 L_1 、 E_2 、 E_3 、 E_4 保证负载上的电压不变。当输入电源电压增大或负载电阻值增大引起 U_0 增大时，原理与前类似，电路通过反馈系统使 U_0 下降到原来的稳定值，从而达到稳定负载电流 I_1 的目的。

优点：开关电源的功率器件工作在开关状态，功率损耗小，效率高。与之相配套的散热器体积大大减小，同时脉冲变压器体积比工频变压器小了很多。因此采用开关电源的恒流源具有效率高、体积小、重量轻等优点。

缺点：开关电源的控制电路结构复杂，输出纹波较大，在有限的时间内实现比较困难。

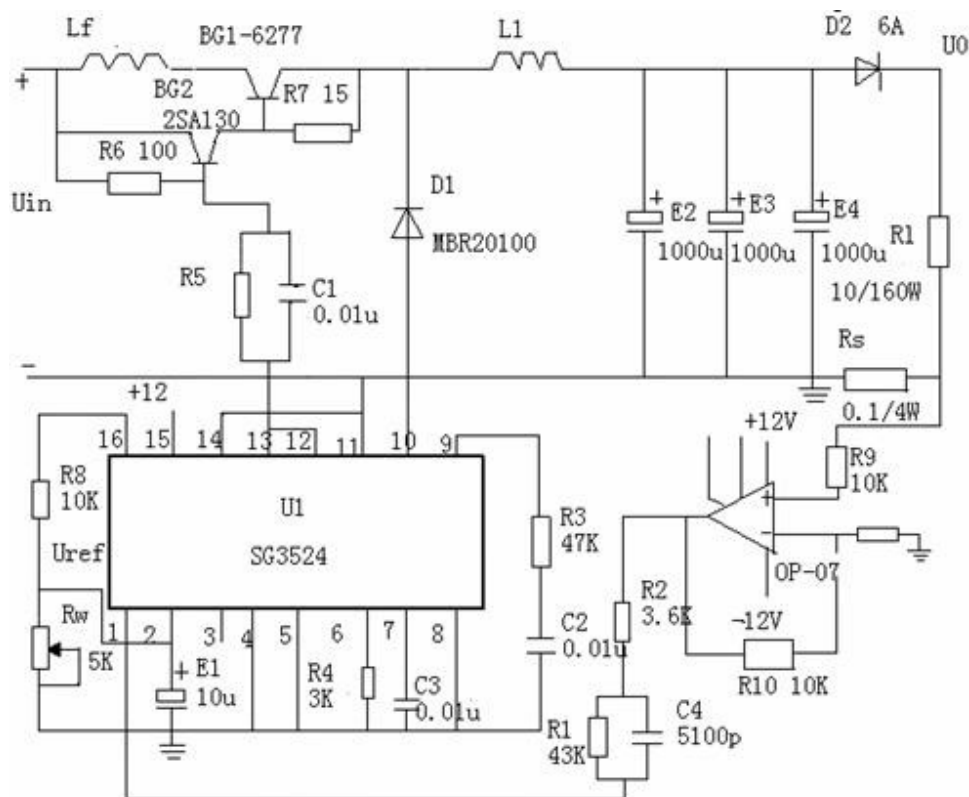


图 2: 采用开关电源的恒流源

2.2 采用集成稳压器构成的开关恒流源

系统电路构成如图 2 所示。MC7805 为三端固定式集成稳压器，调节 R_W ，可以改变电流的大小，其输出电流为： $I_L = (U_{OUT}/R_W) + I_4$ ，式中 I_4 为 MC7805 的静态电流，小于 10mA。当 R_W 较小即输出电流较大时，可以忽略，当负载电阻 R_L 变化时，MC7805 改变自身压差来维持通过负载的电流不变。

优点：该方案结构简单，可靠性高 缺点：无法实现数控。

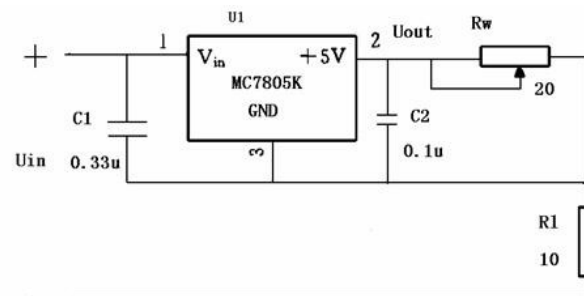


图 3: 采用集成稳压器构成的开关恒流源

3. 采用单片机控制的恒流源

该方案恒流源电路由 N 沟道的 MOSFET、高精度运算放大器、采样电阻等组成, 其电路原理图如图 3 所示。利用功率 MOSFET 的恒流特性, 再加上电流反馈电路, 使得该电路的精度很高。

该电流源电路可以结合单片机构成数控电流源。通过键盘预置电流值, 单片机输出相应的数字信号给 D/A 转换器, D/A 转换器输出的模拟信号送到运算放大器, 控制主电路电流大小。实际输出的电流再通过采样电阻采样变成电压信号, A/D 转换后将信号反馈到单片机中。单片机将反馈信号与预置值比较, 根据两者间的差值调整输出信号大小。这样就形成了反馈调节, 提高输出电流的精度。本方案可实现题目要求, 当负载在一定范围内变化时具有良好的稳定性, 而且精度较高。

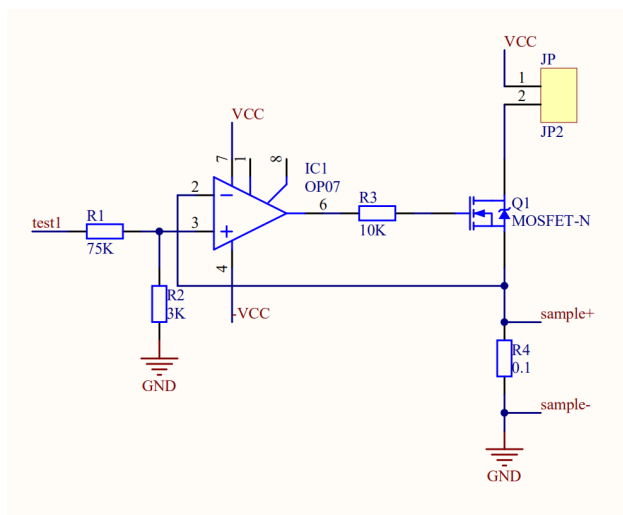


图 4: 采用单片机控制的恒流源

4. 方案描述

该数控恒流源电路主要包括按键输入、LCD 显示屏输出、单片机、RC 滤波电路、压控恒流源电路、电压采样电路几个模块。

系统工作原理为: 通过按键对电流值 $S_CURRENT$ 进行预置, 并显示在 LCD 显示屏上, 单片机根据预置的电流值输出一定占空比的 PWM 波, 通过 RC 滤波电路后输出直流信号。直流信号输送至压控恒流源模块, 通过电阻分压后输送至大功率场效应管 IRF640, 产生相应的电流值。场效应管的漏极电流即为恒流源的实际输出电流。场效应管的漏极电流近似于源极电流, 在电压采样模块, 场效应管的源极电流经过采样电阻后转化为电压信号, 经过差分放大电路后输送至单片机, 单片机采集此信号, 作出相应的调整处理后输出显示在 LCD 显示屏上, 作为电流源的自测表的输出值 $N_CURRENT$, 实际的输出电流通过电流表测量得到。

二 理论分析与计算

1. 理论分析

我们组的电路主要分为 3 个部分, 恒流控制电路, 采样放大电路, 单片机电路, 这里主要对恒流控制电路和采样放大电路进行介绍分析。

恒流控制电路主要用于控制一个恒定的电流进行输出, 首先根据你设定的值控制单片机输出一个占空比确定的 pwm 波, pwm 波经过 RC 滤波变为一个恒定的电压, 也就是经过 pwm 输出电路, 此时这个确定的电压值在经过分压进入恒流控制电路, 这里进行分压的原因主要由于我们所取的采样电阻比较小, 因此必须控制输入的电压值, 经过确定分压比之后的电压再进入运算放大器, 根据理想运放的特性我们可以指导, 2、3 两脚的电压值近似相等, 而 2 脚又与采样电阻相连, 因此此时完整的电压进入采样电阻, 采样电阻另一端又接地, 因此此时采样电阻上的电流即为分压后的电压除以采样电阻的阻值, 而根据 mos 管的特性, 运算放大器和 mosfet 相连出来的电流可以近似忽略, 因此采样电阻上的电流值与我们的输出接口 JP JP2 上的电流值相等, 而据此可知, 我们可以控制 pwm 波的占空比来线性地控制电流。

而电压采样电路主要起到一个放大反馈的作用, 采样电阻上的电压进入我们的电压采样电路, 而根据电压采样电路上的电阻阻值, 我们可以确定一个恒定的放大比, 在这个电路中, 我们把 R7、R8 的值设定为相等, R9、R10 的值也设定为相等, 最后得到的放大比例即为两者之比, 放大后的电压后输出到 AD 输入电路, 此时对放大后的电压进行 RC 滤波, 之后将这个滤波后的电压输入到单片机的 Adin, 单片机根据输入进来的 Adin 可以进行数模转化, 而数模转化比较后则可以对单片机的 pwm 占空比进行一个调节, 以此来达到一个反馈调节的功能。

至于电源模块, LCD 电路我们则是根据老师所给的单片机学习板而设计, 两者相吻合以此来保证正确性, 我们也适当删减了一些单片机学习板上一些没有必要的电路。

恒流控制电路和电压采样电路我们主要参考了老师放在群里的资料。

1.1 理论计算

电路中主要需要计算的是恒流控制电路和电压采样电路的一些电阻值。

这里我先对恒流控制电路上的分压比进行计算, 一开始在不知道采样电阻只能是 0.1 欧姆之前, 我们是按照 0.15 欧姆来进行计算的, 不过此处我就根据 0.15 欧姆的思路给出 0.1 欧姆的计算思路, 首先是分压比调控, 因为 pwm 波的占空比只能在 256 个值之内进行调节, 因此我们取了一个比较好的占空比也即为 200, 因此我们设定它的满量程也即是实验要求 20mA 到 1000mA, 满量程对应此, 1000mA 在采样电阻对应 0.15V, 而 pwm 最大的电压量为 5V, 因此 $5V \cdot k \cdot 200 / 256 = 0.15$, 据此我们可以得到 $k = 0.0384$, 而根据标称电阻值, 我们得到了两个电阻值是比较符合的, 3K 欧姆和 75K 欧姆, $3 / 75 + 3$ 我们可以得到也为 0.384, 因此这就是我们在横流控制电路上的计算。

然后我们对于电压采样电路上的电阻值进行计算, 此处我们采用了相同的电阻值进行控制, 也就是 $R7 = R8 = 3K$, 而 $R9 = R10 = 75K$, 而根据运放的特性我们可以得到放大的比例为 $R9 / R7 = 25$, 因此我们对于采样电阻上的电压放大了 25 倍。之后的一些电容值我们根据老师所给的资料以及老师之后的建议我们进行了调节, 最后所得到的电容值如之前的电路图所示。

此时我们已经可以完整的量程范围 (20, 1000)mA, 而采样电阻上的电压值的范围为 (3, 150)mV, 再到分压前的电压为 (78, 3900)mV, 最后根据此得到占空比 (0.0156, 0.78), 对应应有占空比的分子 (4, 200)。

而在最后电路板的焊接过程中, 我们又进行了调节, 采样电阻改为了 0.1 欧姆, 而在实验室的柜子中 75K 欧姆的电阻是不存在的, 因此我们采取了 100K 欧姆的电阻进行了代替, 当然根据我们电路板的测试中, 在 Adin 电路和 pwm 输入电路中, 本身就存在一定的压降, 因此我们最后还是根据实际测出来 pwm 值, AD 值, 以及实际电流值, 来对我们程序内部的参数来进行设计。

三 电路与程序设计

1. 硬件电路设计

1.1 各模块电路

(1) 总体电路

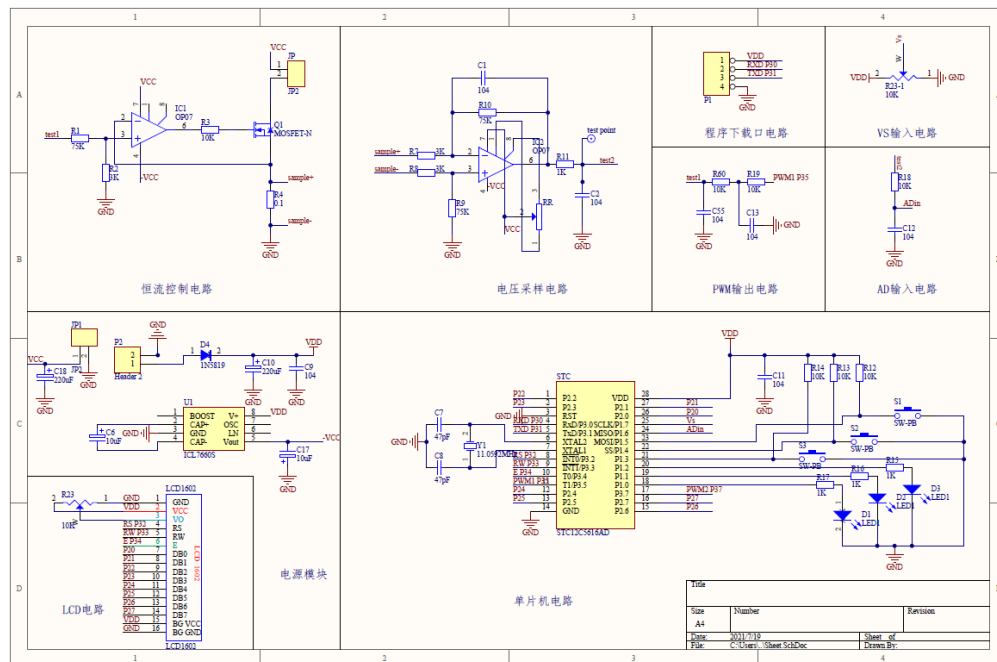


图 5: 总体电路

(2) 压控恒流源模块

压控恒流源是该系统的主要部分，通过电压来控制输出电流的变化。该压控恒流源电路由分压电阻 R_1 、 R_2 ，运算放大器 IC_1 ，大功率场效应管 Q_1 ，采样电阻 R_4 ，电流输出模块 JP_2 等组成。压控恒流源电路的工作原理为：输入 RC 滤波电路得到的直流信号，经过分压后得到 U_3 输入运算放大器，控制输出对应的电流 I_o 。运放采用 OP-07 作为电压跟随器，由于 $U_2=U_3$ ，场效应管 $I_d=I_s$ ，可以得到 $I_o=I_s=U_2/R_{53}=U_3/R_{53}$ 。从 $I_o=U_3/R_{53}$ 可以看出，电路输入电压 U_3 控制输出电流 I_o ，即 I_o 不随 R_L 的变化而变化，从而实现压控恒流。

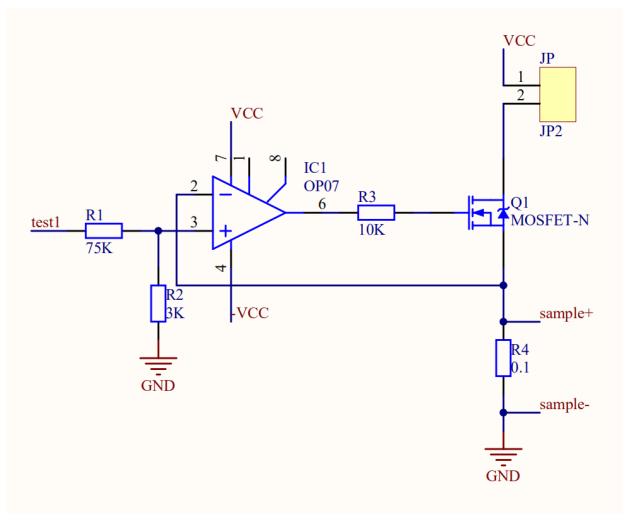


图 6: 压控恒流源模块

(3) 电压采样模块

电压采样模块是反馈部分。该电压采样模块采用差分放大电路，由运算放大器 IC2、若干电阻电容组成。电路原理图如图 7 所示。

其工作原理为：对采样电阻 R4 上的电压进行采样，经放大后输入单片机，由单片机内部的 ADC 模块转换为数字信号作为反馈，PID 调整输出的 PWM 波，实现负反馈。

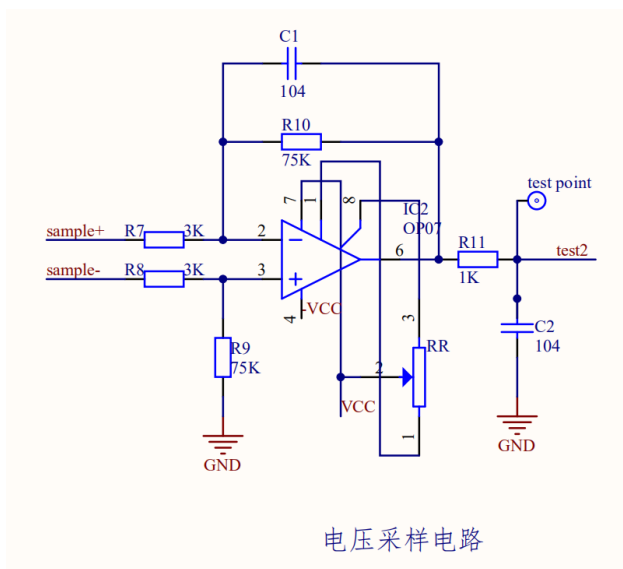


图 7: 电压采样模块

(4) 按键输入模块

该模块采用三个按键输入，按键 1 控制改变数字的位，按键 2 控制数字 +1，按键 3 为确认输入，将设定的数字输入单片机。按键采用防抖动设计。

(5) 单片机以及 A/D 转换模块

单片机为本系统的主控制器,本次设计采用型号为 STC1C5410AD 的单片机,指令周期短,工作速率快,功耗低,具有丰富的片上资源,可以在线下载,易于调试。

单片机自带的 A/D 转换模块为 10 位，取高八位为有效位，转换模块将输入的采样电压转换成数字信号反馈给单片机，单片机将此反馈信号与预置值比较，根据两者间的差值调整输出信号大小。这样就形成了反馈调节，提高输出电流的精度。同时，将采样回来的电压经过单片机处理传送到 LCD，可以显示当前的实际电流值。

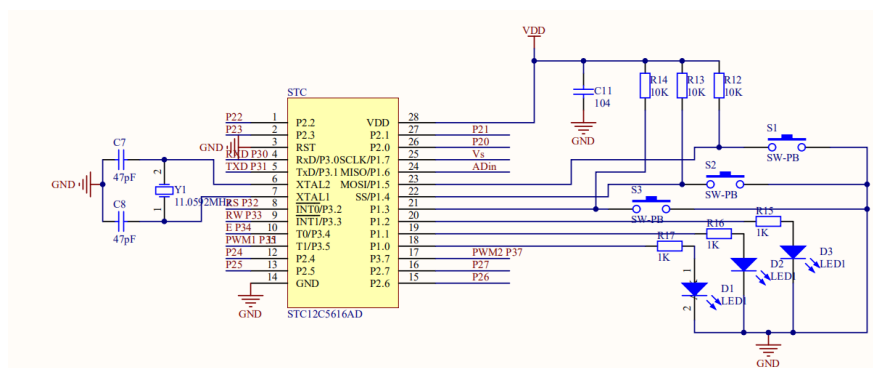


图 8: 单片机电路

(6) LCD 显示模块

LCD 显示屏型号为 LCD1602，第一行显示设定电流值 S_CURRENT，第二行显示单片机测量值 N_CURRENT。

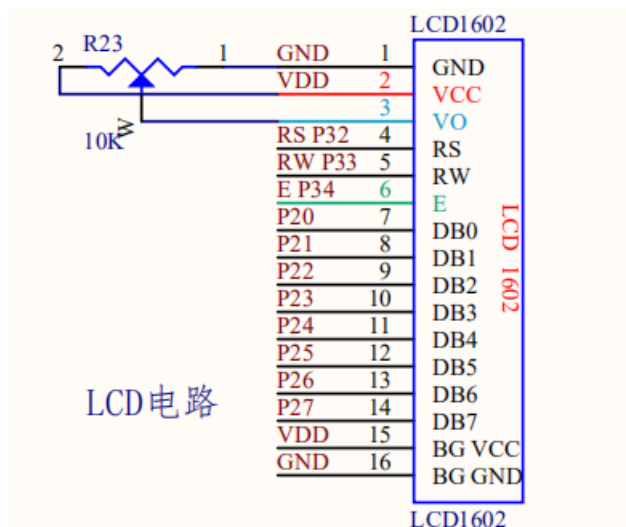


图 9: LCD 电路

(7) 电源模块

电源模块由学生电源提供 12V 直流，由 ICL7660S 生成 -5V 的电压。

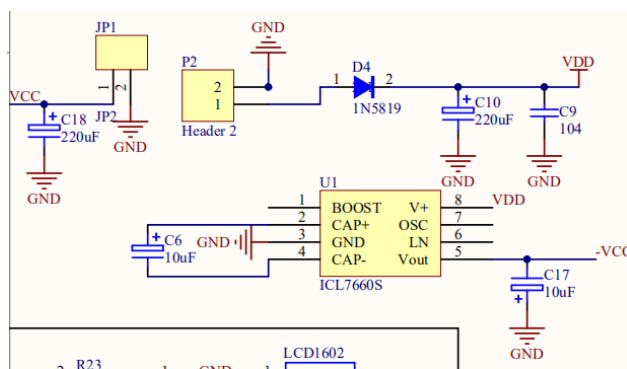


图 10: 电源模块

1.2 PCB 设计

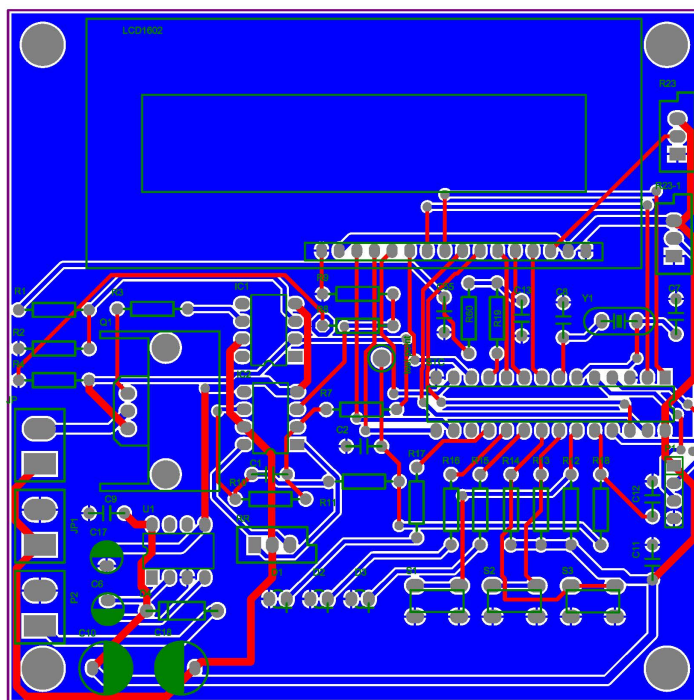


图 11: PCB 设计

2. 软件设计

2.1 流程图

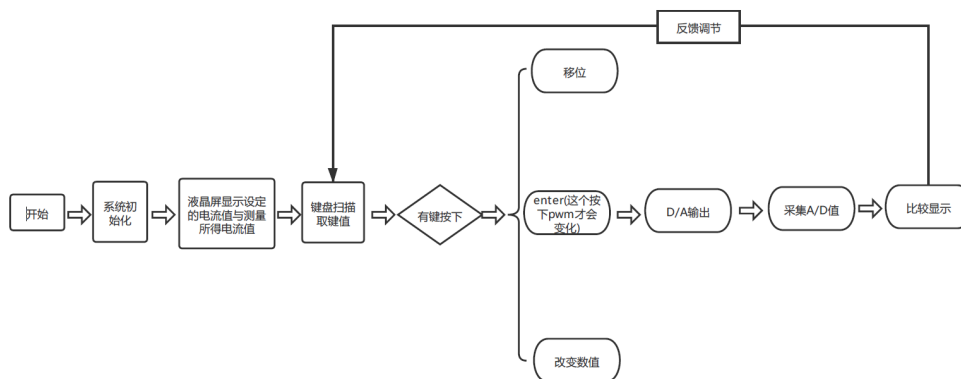


图 12: 流程图

2.2 模块核心代码

Listing 1: ADC 输入及转换为实际电流

```
1 void ADC_Scan(){
2     // 开启ADC，通道六扫描
3     ADC_CONTR = ADC_POWER | ADC_SPEEDLL | 6 | ADC_START;
4     _nop(); //Must wait before inquiry
5     _nop();
6     _nop();
7     _nop();
8     while (!(ADC_CONTR & ADC_FLAG)); //Wait complete flag
9     ADC_CONTR &= ~ADC_FLAG; //Close ADC
10    // 将10位转换结果存入Adc_resuit变量
11    Now_Current = ADC_DATA*4+(ADC_LOW2 && 0x03);
12
13    // 将电流转换为实际电流值
14    Now_Current = Now_Current/0.65;
15
16 }
```

Listing 2: 按键操作及 PWM 控制

```
void    Key_treat()
2      {
        if (Key3_press_flag)
4        {
            //将预设值传入PWM
6            result = Set[3]*1000 + Set[2]*100 + Set[1]*10 + Set
              [0];
            //将按键输入结果转化为PWM控制值
8            result = 255 - result/gain;
            Key3_press_flag = 0;
10        }

        if (Key2_press_flag)
12        {
            switch (target)
14            {
                {
16                case 0: Set[3-target] = (Set[3-target] + 1)%2;
                    break;

18                default: Set[3-target] = (Set[3-target] + 1)%10;
                    break;
20                }
            }
22        Key2_press_flag = 0;
        }

24        if (Key1_press_flag)
26        {
            target = (target+1)%4;
28            Key1_press_flag = 0;
        }
30    }
```

四 测试方案与测试结果

1. 测试步骤

- (1) 单片机 PWM 波输出占空比设置为 50%，改变负载，测试恒流控制模块是否能达到恒流控制效果
- (2) 断开 PWM 波反馈回路，测量单片机 ADC 输入，得到输出电流与 ADC 转换结果之间的关系，校准参数
- (3) 连接 PWM 反馈回路，设定不同的电流值，计算得到 PWM 占空比与电流之间的关系，调整代码

2. 测试结果

2.1 电流范围与步进调节

设定值 (mA)	20	30	40	50	500	1000
测定值 (mA)	19.7	29.4	39.1	48.4	498.5	1003.1
显示值 (mA)	20	26	38	44	493	998

2.2 负载电阻改变电流变化测量

设定 500mA		
接入电阻	第一次测量	第二次测量
0 欧姆	498.5mA	500.1mA
7.5 欧姆	493.1mA	495.5mA

2.3 纹波电流测量

接入 7.5Ω 电阻，将示波器接在电阻两端，得到纹波电流为 2mA

3. 数据处理

设定值 (mA)	20	30	40	50	500	1000
测定值 (mA)	19.7	29.4	39.1	48.4	498.5	1003.1
显示值 (mA)	20	26	38	44	493	998
设定与测量差值	0.3	0.6	0.9	1.6	1.5	-3.1
测量与显示差值	-0.3	3.4	1.1	4.4	5.5	5.1

设定 500mA		
接入电阻	第一次测量	第二次测量
0 欧姆	498.5	500.1
7.5 欧姆	493.1	495.5
变化值	5.4	4.6

4. 数据分析

4.1 电流范围与步进

由上述表格可知, 我们能够通过设定不同的值使输出电流在 20mA 1000mA 的范围内变化, 步进至少能达到 10mA 的精度, 达到了基础部分对步进的要求, 同时电流可调范围达到了提升要求部分。

4.2 输出电流与给定值偏差

由上述表格可知, 设定不同的输出电流时, 测量电流与设定值之间的最大差值出现在设定 1000mA 时, 为 3.1mA, 远远小于要求中的设定值的 $1\%+10\text{mA}$

4.3 显示输出与电流差值

由上述表格可知, 显示值与测量差值之间的误差最大发生在设定电流为 500mA 时, 差值为 5.5mA, 要求允许的最大差值为 $(1000 \times 1\% + 3)\text{mA} = 13\text{mA}$, 满足要求

4.4 改变负载电阻

我们将电流输出设定为 500mA, 由表三可知, 空载时输出电流与接入负载后输出电流变化值在 2mA 左右, 而要求中的变化值为 3mA, 符合系统设计要求

4.5 文波系统

测量出的文波电流为 2mA, 达到了基础要求中的 5mA, 但是没有达到提升部分中的 1mA 要求

五 完整代码

最后附上完整代码

Listing 3: 完整代码

```
1 #include <reg52.h>
  #include <STC12C.h>
3 #include <intrins.h>
  #include <stdio.h>
5 #include <math.h>

7
  // 变量类型标识的宏定义
9 #define Uchar unsigned char
  #define Uint unsigned int
11 #define uchar unsigned char
  #define uint unsigned int
13
  #define gain 4.57718
15 #define DataPort P2      // 数据端口
  #define Busy      0x80
17
  // 接口定义
19 sbit    Led1 = P1^0;
    sbit    Led2 = P1^1;
21 sbit    Led3 = P1^2;

23 sbit    Key1 = P1^5;
    sbit    Key2 = P1^4;
25 sbit    Key3 = P1^3;

27 // 控制引脚定义, 不同的连接必须修改的部分
    sbit RS=P3^2;
29 sbit RW=P3^3;
    sbit EN=P3^4;
31
    #define rs RS
33 #define rw RW
    #define en EN
35
  //ADC 相关变量
37 #define ADC_POWER    0x80          //ADC power control bit
    #define ADC_FLAG    0x10          //ADC complete flag
```

```
39 #define ADC_START    0x08           //ADC start control bit
   #define ADC_SPEEDLL 0x00           //420 clocks
41
   /*按键相关变量*/
43 bit Key1_press_flag;    //Key1按下标志
   bit Key2_press_flag;    //Key2按下标志
45 bit Key3_press_flag;    //Key3按下标志

47 bit Key1_back;         //按键的上一状态
   bit Key2_back;
49 bit Key3_back;

51 //电流预设值修改显示相关
   int result;
53 int Set[4] = {0,0,0,0}; //电流预设值
   int target; //选择修改位
55
   //ADC输出控制PWM变量
57 Uint PWM_Ctrl = 0;
   Uint Now_Current = 0;
59
   //LCD显存
61 uchar Dsp_buf[16];
   uchar Dsp_buf2[16];
63
   //定时计数器
65 uint Timer_Count;

67 void InitADC();
   void ADC_Scan();
69 void Key_Scan();
   void Key_treat();
71 void LcdWriteData( char dataW );
   void LcdWriteCommand( Uchar CMD );
73 void Lcd_init( void );
   void Disp_XY( char posx,char posy);
75 void DispOneChar(Uchar x,Uchar y,Uchar Wdata);
   void Display (void);
77 void Pwm_init(void);
   void Pwm_treat();
79 void system_init(void);
   void timer0_int (void);
81 void Delay(uint y);
```

```
83  /*****延时函数*****/
    // 延时y ms
85  void Delay(uint y)
    {
87      uint x;
      while (y--)
89      {
          x = 1000;
91      while (x--);
      }
93  }

95  /*****系统初始化*****/
    void system_init(void)
97  {
        // 定义端口输入输出, p1^0-P1^2为推挽输出, p1^3-P1^7为输入, 设置PxM0,
        PxM1,
99      P1M0 = 0xf8;          // #11111000b
      P1M1 = 0x07;          // #00000111b
101
        // T0, 模式设定
103      TR0 = 0;             // 停止计数
      ET0 = 1;             // 允许中断
105      PT0 = 1;             // 高优先级中断
      TMOD = 0x01;         // #00000,0001, 16位定时模式
107
      TH0 = 0;
109      TL0 = 0;
      TR0 = 1;             // 开始运行
111
      P1 &= ~0x07;         // 清除LED
113  }

115  void timer0_int (void) interrupt 1
    {
117      TH0 = 0xfc;
      TL0 = 0x67;          // 1ms 定时中断
119
      Key_Scan();
121
      ++Timer_Count;
123
```



```
    if ((Timer_Count & 0x3f) == 1){
125        ADC_Scan();
    }
127
    if (Timer_Count > 500)
129    {
        Timer_Count = 0;
131        Pwm_treat();
    }
133 }

135
// 按键相关函数
137
/***** 键盘扫描函数 *****/
139 void    Key_Scan()
{
141     if (!Key1)          // 开机键扫描, 检测下降沿
    {
143         if(Key1_back)
        {
145             Key1_press_flag = 1;
        }
147     }

149     if (!Key2)          // 开机键扫描
    {
151         if(Key2_back)
        {
153             Key2_press_flag = 1;
        }
155     }

157     if (!Key3)          // 开机键扫描
    {
159         if(Key3_back)
        {
161             Key3_press_flag = 1;
        }
163     }

165     Key1_back = Key1;
    Key2_back = Key2;
```

```
167     Key3_back = Key2;

169 }

171 /***** 按键处理 *****/
void    Key_treat()
173 {
    if (Key3_press_flag)
175     {
        // 将预设值传入PWM
177         result = Set[3]*1000 + Set[2]*100 + Set[1]*10 + Set[0];
        // 将按键输入结果转化为PWM控制值
179         result = 255 - result/gain;
        Key3_press_flag = 0;
181     }

183     if (Key2_press_flag)
    {
185         switch (target)
        {
187             case 0: Set[3-target] = (Set[3-target] + 1)%2;
                        break;

189             default: Set[3-target] = (Set[3-target] + 1)%10;
                        break;
191         }
        Key2_press_flag = 0;
193     }

195     if (Key1_press_flag)
    {
197         target = (target+1)%4;
        Key1_press_flag = 0;
199     }

201 }

203 // 初始化ADC寄存器
205 void InitADC(){
    ADC_DATA = 0; // 将ADC结果寄存器清零
207     ADC_CONTR = ADC_POWER | ADC_SPEEDLL;
    Delay(2);
209 }
```

```
211 void ADC_Scan(){
    //开启ADC, 通道六扫描
213     ADC_CONTR = ADC_POWER | ADC_SPEEDLL | 6 | ADC_START;
    _nop(); //Must wait before inquiry
215     _nop();
    _nop();
217     _nop();
    while (!(ADC_CONTR & ADC_FLAG)); //Wait complete flag
219     ADC_CONTR &= ~ADC_FLAG; //Close ADC
    //将10位转换结果存入Adc_resuit变量
221     Now_Current = ADC_DATA*4+(ADC_LOW2 && 0x03);

223     //将电流转换为实际电流值
    Now_Current = Now_Current/0.65;
225
227 }

/*=====
229 显示光标定位
=====*/
231 void Disp_XY( char posx,char posy)
{
233     uchar temp;

235     temp = posx %0x40; // & 0x07;
    posy &= 0x01;
237     if ( posy==1) {temp |= 0x40;}
    temp |= 0x80;
239     LcdWriteCommand(temp);
241 }

/*=====
243 按指定位置显示数出一个字符
=====*/
245 void DispOneChar(Uchar x,Uchar y,Uchar Wdata) {

247     Disp_XY( x, y ); // 定位显示地址
    LcdWriteData( Wdata ); // 写字符
249 }

251 /*=====
    初始化程序, 必须按照产品资料介绍的初始化过程进行
```

```
253  =====*/
void Lcd_init( void )
255  {
    LcdWriteCommand(0x38);          // 显示模式设置(以后均检测忙信号)
257    Delay(2);
    LcdWriteCommand(0x0f);          // 开显示屏, 管光标, 光标不闪烁
259    Delay(2);
    LcdWriteCommand(0x06);          // 显示地址递增
261    Delay(2);
    LcdWriteCommand(0x01);          // 显示清屏
263    Delay(2);
    LcdWriteCommand(0x02);  // 光标
265  }

267  /*=====
    写控制字符子程序: E=1 RS=0 RW=0
269  =====*/
void LcdWriteCommand(Uchar CMD)
271  {
    rs=0;          //选命令寄存器
273    rw=0;          //写
    DataPort=CMD;  //写命令
275    Delay(1);      //延时, 让1602准备数据
    en=1;          //使能信号变化, 写入数据
277    en=0;
    }

279  /*=====
    当前位置写字符子程序: E =1 RS=1 RW=0
    =====*/
283  void LcdWriteData( Uchar Data )
    {
285    rs=1;          //选数据寄存器
    rw=0;          //写
287    DataPort=Data; //写命令
    Delay(1);      //延时, 让1602准备数据
289    en=1;          //使能信号变化, 写入数据
    en=0;
291  }

293  /*****显示函数*****/
void Display (void)
295  {
```

```
int n,m =0;
297 //将设定值输入显存
sprintf(Dsp_buf,"S_Currute:%1d%1d%1d%1dmA" , Set[3] , Set[2] , Set[1]
    ,Set[0]);
299 //将AD采样值输入显存
sprintf(Dsp_buf2,"N_Currute:%04dmA" ,Now_Current);
301 LcdWriteCommand(0x80);
for(m=0;m<16;m++)
303 {
    LcdWriteData(Dsp_buf[m]);
305 }
LcdWriteCommand(0x80+0x40);
307 for(n=0;n<16;n++)
{
309     LcdWriteData(Dsp_buf2[n]);
}
311 }

313 void Pwm_init(void)
{
315     CCON = 0;                //Initial PCA control register
                                //PCA timer stop running
317                                //Clear CF flag
                                //Clear all module interrupt flag
319     CL = 0;                  //Reset PCA base timer
    CH = 0;
321     CMOD = 0x02;            //00000010B //Set PCA timer clock source as Fosc/2
                                //Disable PCA timer overflow interrupt
323
    CCAP1H = CCAP1L = result;    //PWM1 port output 0% duty cycle
                                square wave
325     CCAPM1 = 0x42;          //PCA module-1 work in 8-bit PWM mode
                                and no PCA interrupt

327     CR = 1;                //PCA timer start run
}
329
void Pwm_treat()
331 {
    CCAP1H = CCAP1L = result+2;
333
}
335
```

```

/*****主函数*****/
337 void main(void)
{
339     system_init();
    Delay(100);
341     Lcd_init();
    InitADC();
343     Pwm_init();

345     EA = 1;

347     while (1)
    {
349         Key_treat();
        Delay(50);
351         Display();
        Disp_XY(target+10,0);
353     }
}
```