官方店铺：https://shop527463029.taobao.com

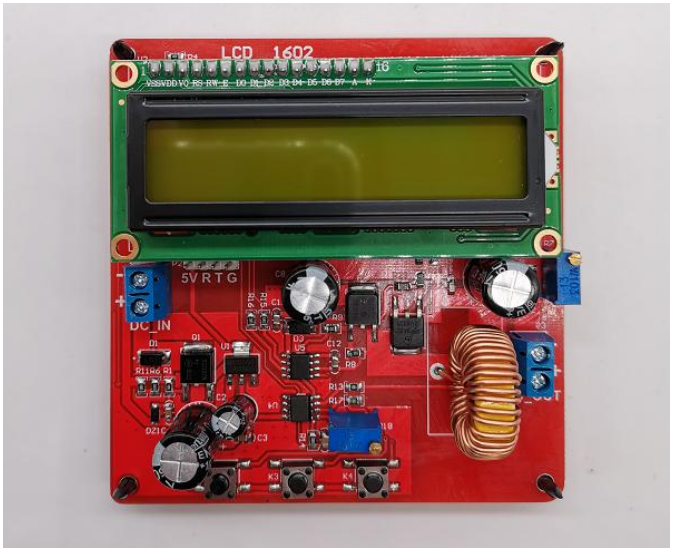
单片机 Buck 电路设计《由软件部分和硬件电路组成》

一．硬件电路设计

1. 供电电源。
2. 输入电压采集。
3. 输出电流采集（过流保护）。
4. Buck 驱动电路。
5. 液晶显示。
6. 按键检测。
7. 输出滤波参数。
8. 输出电压采集。

二．软件编写

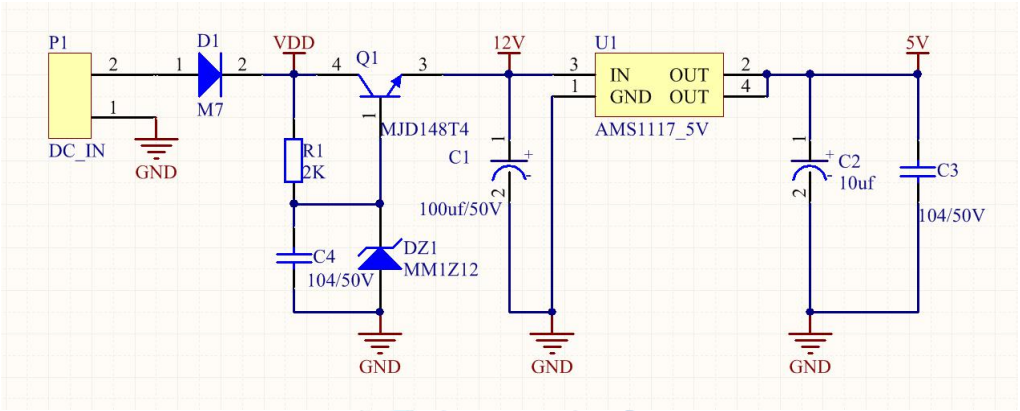
1. PWM 输出配置函数。
2. 按键功能。
3. 液晶显示。
4. 输入电压、输出电压、输出电流采集。
5. 输入欠压、输入过压、输出过流保护。
6. Pid 恒压算法控制。



以下分别对应各部分讲解：

一．硬件电路设计

1.供电电源

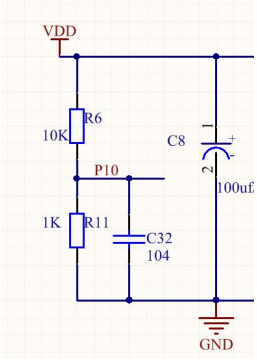


上图为本设计的电源电路，其中 D1 为普通单向整流二极管，其目的为防止用户将输入电源正负极接反，由于二极管的单向导通特性，用户反接电源正负极时后级电路不会工作，从而避免后级电路因反接电源而损坏。

图中 R1、C4、DZ1、Q1 组成线性稳压器，DZ1 为 12V 的稳压管，一部分电流经 R1 流过 DZ1 稳压二极管，那么三极管 Q1 的基极电压为 12V，那么三极管就会处于导通状态，这样三极管的发射极就会是小于基极 0.7V 左右的电压，也就是 11.3V 的电压。这部分电路的目的是将 VDD 稳压后为约 12V 输出，这里的 12V 给后级的 MOS 管驱动 IC（IR2104）提供电源。关于此线性稳压电路原理详细讲解请参考我们博客 https://blog.csdn.net/ssss992/article/details/82252834。

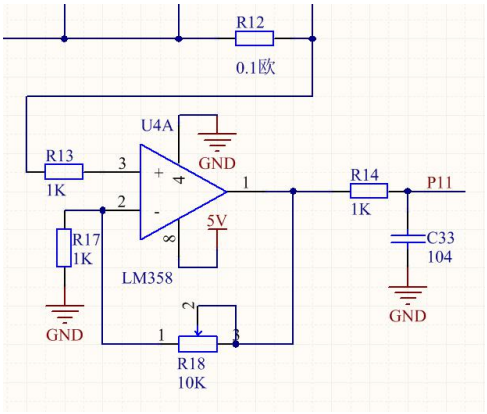
图中 U1 为 12V 降为 5V 作用，5V 给单片机和运放电路部分提供电源。

2. 输入电压采集



图中 R6、R11 组成输入分压电路，由单片机 AD 采集 IO 口 P10 采集 AD 值，并由计算公式：输入电压=11*5000*(AD/1024)mV≈54*ADmV;（注：5000 代表单片机供电电压 5000mv，11 代表需要将采集到的电压值放大 11 倍方为输入电压值，AD 代表采集到的 AD 值，1024 代表单片机的 AD 位数，即 10 位 AD 对应满值为 1024）。

3. 输出电流采集



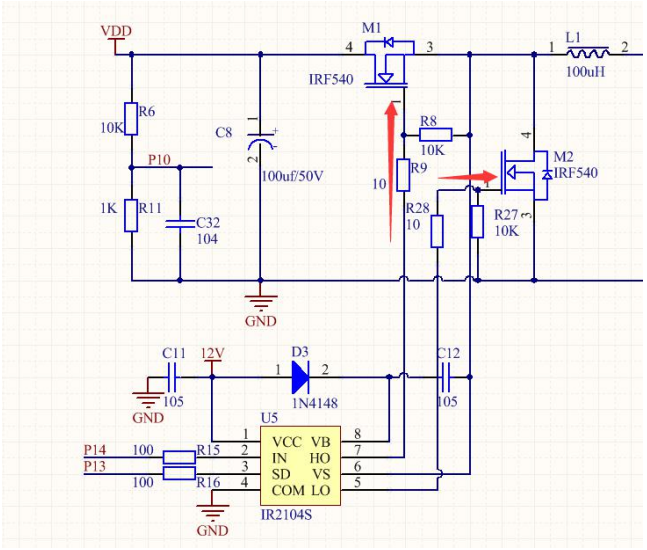
我们将 R12 采样电阻串入输出负极回路中，那么输出的电流将会在采样电阻上分压，这个电压需要经过运放放大后送到单片机 AD 口检测。输出电流采集的公式为：

输出电流=（(AD/1024*5000)/10）/0.1 =4.8*AD(mA) ；

过流值计算：设置过流值为 2A. 电流采样电阻为 0.1 欧姆,经放大电路放大 10 倍送 AD 检测 2A 电流时,采样电阻分压为 0.2V,放大 10 倍为 2V. 对应 AD 值= (2/5)*1024= 409.6=410;

当测量到电流与实际测量值偏差较大时，可调节 R18 电位器进行校准。

4. 驱动电路



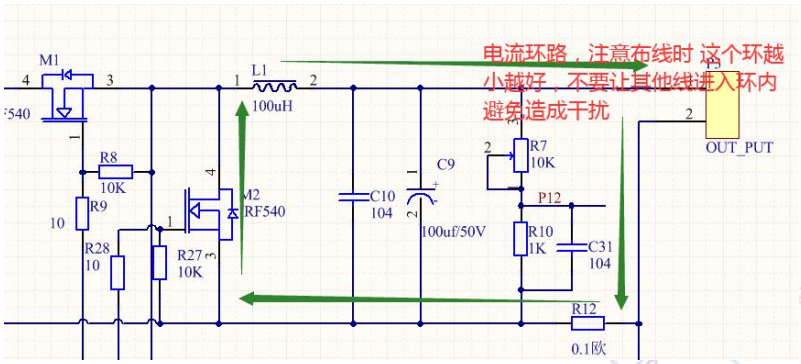
本电路的开关拓扑采用 Buck 方式，Buck 变换器是一种降压式非隔离开关电源，当开关

管导通时，输入电源通过电感给输出供电，同时电感存储能量；当开关管关断时，电感通过续流二极管给输出供电；如此反复即可维持输出产生一个恒定的电压。

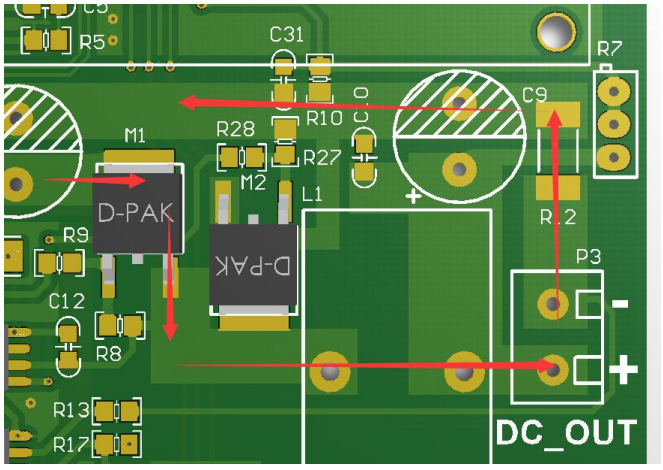
这里我们采用 M2（N-Mos 管）来替代传统的二极管构成同步整流方式。传统的非同步整流驱动方式中使用的是二极管，M1 是 N 沟道 Mos 管，GS（栅源）电压至少大于 10V 方可满足打开状态，由于 M1 的 S 极为输出端，不是常规接地，那么 G 极驱动就属于浮动驱动，也就是我们需要保证 G 极电压减去 S 极电压大于 10V。所以采用专用高低端 Mos 驱动 IC（IR2104）来驱动，该 IC 可以在 M1 导通后，与负载构成低阻抗回路后，由 12V 电源通过 D3 向 C12 充电，从而抬高 HO 输出的电压，即在 VS 端端电压的基础上抬高 12V，这样就能保证无论输出电压多高，M1 的 G 极电压始终比 S 极电压大 12V，即足可以保证 M1 导通。

那么我们要注意关键一点，即只有在输出与负载构成低阻抗回路时，才能保证 M1 的正常导通。因为低阻抗时才能满足快速充电以抬高驱动电压，这就说明当采用二极管作为整流器件时，如果想要输出电压正常，就必须在后级带负载，而且负载要足够大（这里负载越小即输出电流越小），也就是传统的二极管整流模式驱动时是不能空载工作的。

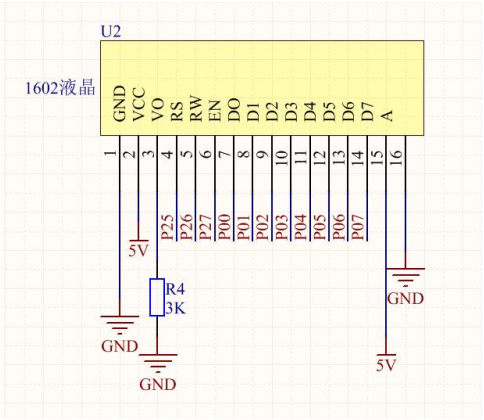
我们将传统的整流二极管更换为 N-Mos 管后，那么在每个高端 Mos 关闭时，这个代替二极管的 N-Mos 管将被导通，VS 端通过 M2 对地导通，使 C12 电容得以充电，从而使 HO 正常输出。所以采用 N-Mos 管替代二极管完美解决了输出空载正常输出问题，并降低了由于二极管自身导通压降带来的损耗。



注意电路板 PCB 布局中，由于电感电流续流的作用构成的电流环路电流随着开关频率变化较大，电流的变化产生较强的电磁干扰，故 PCB 布线时尤其注意该环路的面积一定要最小，且环路内尽可能不要有其他线路，尤其信号采样线路要尽可能远离环路，避免造成干扰。以下为本开发板的环路布线布局，开发者可以参考，虽然按照这样的布局较为符合理论设计，但图中的 M2 的控制端线路尽可能在环路外部最好，这也是开发板性能再提升需改进之处。

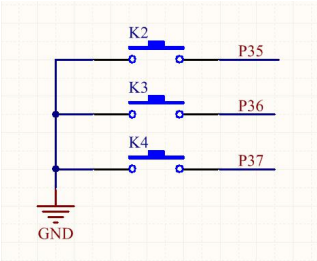


5. 显示



显示部分采用 LCD1602 液晶，利用单片机 P0 口作为数据口，另 3 个 IO 来完成液晶驱动片选等控制。R4 为液晶对比度调节电阻，经对比，3K 阻值为合适阻值。

6. 按键检测

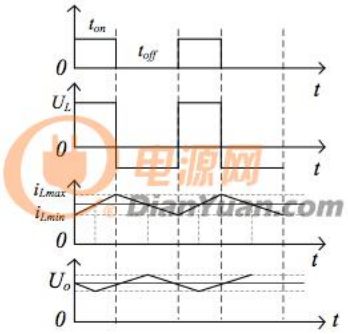


按键检测采用检测低电平有效功能，故直接将按键一端接地即可，按键按下后单片机检测到低电平进行相应功能控制。

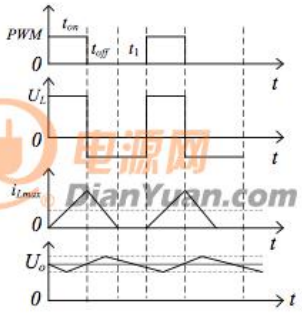
7. 输出滤波参数

Buck 降压型开关电路如果在后级不加滤波电路的情况下，那么输出为高低变化的脉冲波电压。使用合适的 LC 滤波器可将脉冲波平滑成无纹波的直流输出，其值等于脉冲电压的平均值。滤波器采用电感和电容组成低通滤波器，这样通过调节驱动开关管的占空比，可以控制经过滤波后的电压值。

需要特别注意一点的是电感的工作模式问题，电感电流在开关管的导通和关闭周期内的电流波形如下图时，即开关管导通时，电感电流呈上曲线，开关管闭合时，电感电流呈下降曲线，然后随着开关管的变化周而复始，这种电流变化模式我们称之为 CCM（continuous current mode：连续电流模式）工作模式，即工作周期中电感的电流从未降为 0。在 CCM 模式下，占空比不随负载电流改变而改变（相当于零输出阻抗），输出电压 Vo=占空比*输入电压。



当电感值比较小，或输出负载电阻比较大时（输出电流较小），当电感电流下降至 0 时，开关周期还未开始，则电感电流会保持一段时间为 0 电流，如下图波形。即处于闲置状态，那么这种状态称之为 DCM（Discontinuous conduction mode：不连续工作模式）模式 。在 DCM 模式下，占空比随负载电流减小而改变（相当于有输出阻抗）。所以工作在 DCM 模式下，电路控制比较困难。因此我们在选择电感参数的时候，要将电感值设计为符合工作在 CCM 模式状态下的范围内。



因此电感的选择应保证直流输出电流为最小规定电流（通常约为额定负载电流的 10%，即 0.1Ion，Ion 为额定输出电流）时，电感电流也保持连续。电感的电流的变化量 di=imax-imin（imax 即电感电流的最大值，imin 即最小值），因为当直流电流等于电感电流峰峰值一半时，进入不连续工作模式，则

Iomin =0.1Ion=（imax-imin）/2，即 di=（imax-imin）=0.2Ion

另外根据电感和电压以及和电流变化率的关系（电感的伏安特性）

电感两端电压=电感值*电流的变化率 → VL=L*（di/Ton）

VL： 电感两端电压 L：电感值 di：电流变化量 Ton：开关管导通时间

→ di=VL*Ton/L=（Vi-Vo）*Ton/L

Vi:输入电压 Vo：输出电压

→ L=（Vi-Vo）*Ton/di =(Vi-Vo)*Ton /(0.2Ion)

又由于 Ton=Vo*T/Vi;（T:开关管开关时间）。

→ L=(5（Vi-Vo）*Vo*T)/（Vi*Ion）

到此，电感的计算公式已经推导出来，接下来我们将设计目标参数列出如下：
开关频率：22Khz;
输入电压：20V;
输出电压： 5V;
输出电流： 3A;
在 CCM 模式下且负载电流 10%额定负载时，纹波小于 100mV;
假设最小负载电流是 10%的额定输出电流，即输出电流为 3*0.1=0.3A 时。

L=（5（Vi-Vo）*Vo*T)/（Vi*Ion）=（5*（20-5）*5*45*10^{-6}）/(20*5)≈ 168uh;
这里根据成品电感感值种类，我们采用电感值为 150uh。

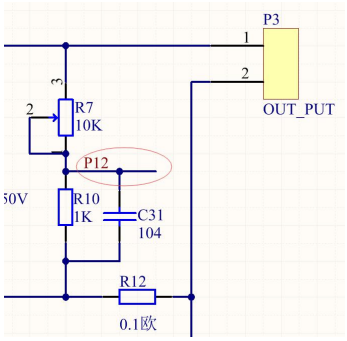
根据 di=（imax-imin）=0.2Ion 得出：电感电流的峰峰值为 0.2Ion=0.6A。
假设输出纹波电压大部分分量由滤波电容的 ESR（等效串联电阻 Ro）产生，那么可以选择电容器使 ESR 满足纹波电压的要求。

设纹波电压的峰峰值 Vrr==0.1V,则需要 ESR 为 Ro=Vrr/di= 0.1/(imax-imin)≈0.17 欧。

根据典型 ESR 与电容值的关系式：RoCo=50*10^{-6} 。
可得：Co=50*10^{-6} /0.17 ≈ 294uf。

根据以上关系可知 ESR 越小（电容越大），di 一定时，则纹波会减小。那么我们选择用 470uf 的电容，则完全满足纹波要求。

8. 输出电压采集



输出电压采集同样采用电阻分压方式采集，C31 电容为滤波抗干扰作用。图中 R7、R10 组成输入分压电路，由单片机 AD 采集 IO 口 P12 采集 AD 值，并由计算公式：输出电压=(R7+1)*5000*(AD/1024)mV;（注：5000 代表单片机供电电压 5000mv，(R7+1)代表需要将采集到的电压值放大(R7+1)倍方为输出电压值，AD 代表采集到的 AD 值，1024 代表单片机的 AD 位数，即 10 位 AD 对应满值为 1024）。这里将 R7 用电位器作为分压电阻，正是因为在 pid 稳压算法比较时要直接用到采集的 AD 值有关，目的是可以通过调节电位器达到理论的分压比例，具体的原因可详看软件部分<按键设置>。

二．编写软件

1.PWM 输出配置函数

要驱动 Buck 中开关管，利用单片机的一路 PCA 计数器的 PWM 输出模式来驱动。单片机采集输出电压与用户设置电压进行比较反馈，控制 pwm 占空比相应变化，理论上占空比越大，输出电压越高。开关管的开关频率由 pwm 的频率决定，考虑到开关效率及经验设置 pwm 输出频率为 22khz（过高则开关损耗越大，过低则滤波器体积越大），具体的程序编写设置如下：

```
/************************************************************
函数说明：  PWM 初始化
************************************************************/
void init_pwm()
{
    CCON=0;                 //pca 控制寄存器复位
    CL=0;                   //pca 计数器低位清零
    CH=0;                   //pca 计数器高位清零
    CMOD=0X02;              //0X08:43KHZ // 0X02,22KHZ // 0X0A,10KHZ
    CCAPM1=0x42;            //设置为 8 位 PWM 输出，无中断
    CCAP1L=CCAP1H=255; //初始化 pwm 输出的占空比
    CR=1;                   //开始计数运行
}
```

查看单片机数据手册中中关于 CCON 寄存器的功能描述，可知各位设置为 0 即可。

2. PCA控制寄存器CCON

PCA控制寄存器的格式如下：

CCON：PCA控制控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	name	CF	CR	-	-	-	-	CCF1	CCF0

CF：PCA计数器阵列溢出标志位。当PCA计数器溢出时，CF由硬件置位。如果CMOD寄存器的ECF位置位， 则CF标志可用来产生中断。CF位可通过硬件或软件置位， 但只可通过软件清零。

CR：PCA计数器阵列运行控制位。该位通过软件置位， 用来起动PCA计数器阵列计数。该位通过软件清零， 用来关闭PCA计数器。

CCF1：PCA模块1中断标志。当出现匹配或捕获时该位由硬件置位。该位必须通过软件清零。

CCF0：PCA模块0中断标志。当出现匹配或捕获时该位由硬件置位。该位必须通过软件清零。

CL 和 CH 是 PCA 的计数器的低高位，我们将其设置初始值为 0，一旦开启 pca 计数，那么该寄存器则会开始累计增加，即 CR=1 时。

4. PCA的16位计数器 — 低8位CL和高8位CH

CL和CH地址分别为E9H和F9H，复位值均为00H，用于保存PCA的装载值。

寄存器 CMOD 控制了 pwm 输出频率，即手册描述如下：

1. PCA工作模式寄存器CMOD

PCA工作模式寄存器的格式如下：

CMOD：PCA工作模式寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	name	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF

CIDL：空闲模式下是否停止PCA计数的控制位。
当CIDL=0时，空闲模式下PCA计数器继续工作；
当CIDL=1时，空闲模式下PCA计数器停止工作。

CPS2、CPS1、CPS0：PCA计数脉冲源选择控制位。PCA计数脉冲选择如下表所示。

CPS2	CPS1	CPS0	选择PCA/PWM时钟源输入
0	0	0	0，系统时钟/12，SYSclk/12
0	0	1	1，系统时钟/2，SYSclk/2
0	1	0	2，定时器0的溢出脉冲。由于定时器0可以工作在1T模式，所以可以达到计一个时钟就溢出，从而达到最高频率CPU工作时钟SYSclk。通过改变定时器0的溢出率，可以实现可调频率的PWM输出
0	1	1	3，ECI/P1.2(或P4.1)脚输入的外部时钟（最大速率＝SYSclk/2）
1	0	0	4，系统时钟，SYSclk
1	0	1	5，系统时钟/4，SYSclk/4
1	1	0	6，系统时钟/6，SYSclk/6
1	1	1	7，系统时钟/8，SYSclk/8

例如，CPS2/CPS1/CPS0 = 1/0/0时，PCA/PWM的时钟源是SYSclk，不用定时器0，PWM的频率为SYSclk/256

我们将 CMOD 设置为 0x02，即 pwm 的频率为 SYSclk/256/2≈22Khz。

寄存器 CCAPM1 控制 pwm 输出模式及是否开启 pwm 中断，手册描述如下：

PCA模块的工作模式设定表如下表所列：

PCA模块工作模式设定（CCAPMn寄存器，n＝0,1）

-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	模块功能
	0	0	0	0	0	0	0	无此操作
	1	0	0	0	0	1	0	8位PWM, 无中断
	1	1	0	0	0	1	1	8位PWM输出，由低变高可产生中断
	1	0	1	0	0	1	1	8位PWM输出，由高变低可产生中断
	1	1	1	0	0	1	1	8位PWM输出，由低变高或者由高变低均可产生中断
	X	1	0	0	0	0	X	16位捕获模式，由CCPn/PCAn的上升沿触发
	X	0	1	0	0	0	X	16位捕获模式，由CCPn/PCAn的下降沿触发
	X	1	1	0	0	0	X	16位捕获模式 由CCPn/PCAn的跳变触发
	1	0	0	1	0	0	X	16位软件定时器
	1	0	0	1	1	0	X	16位高速输出

360 南通国芯微电子有限公司 总机：0513-5501 2928 / 2929 / 2966 传真：0513-5501 2969 / 2956 / 2947

我们将 CCAPM1 设置为 0x42，即二进制 01000010，即设置为 8 位 PWM 输出，无中断。

寄存器 CCAP1L 和 CCAP1H 分别是用于和 PCA 的计数器相比较从而控制 pwm 输出占空比的寄存器，即改变 CCAP1L 和 CCAP1H 的值即可改变 pwm 占空比。寄存器手册说明如下：

5. PCA捕捉/比较寄存器 — CCAPnL（低位字节）和CCAPnH（高位字节）

当PCA模块用于捕获或比较时，它们用于保存各个模块的16位捕捉计数值；当PCA模块用于PWM模式时，它们用来控制输出的占空比。其中，n=0、1，分别对应模块0和模块1。复位值均为00H。它们对应的地址分别为：

CCAP0L — EAH、CCAP0H — FAH：模块0的捕捉/比较寄存器。

CCAP1L — EBH、CCAP1H — FBH：模块1的捕捉/比较寄存器。

2. 按键功能函数

按键用来设置输出电压的大小，最小步进值为 0.1V。K_ADD 为设置增加电压按键功能，K_MIN 为设置减少电压功能，K_Start 为控制电压输出开关功能。加 delay(30);函数为用户按键延时消抖动设计。

```
/*********************************************************
函数说明：按键延时防抖动
*********************************************************/
void key_scan()
{
   if(!K_ADD)
    {
      delay(30);
       if(!K_ADD)
       {
            if(U2<(InVoltage*0.8))     //最大输出电压为输入电压的 80%
            {
                 U2+=1;
                  Out_voltage0_temp=Out_voltage0_temp+2;
                  if(Out_voltage0_temp>1023)
                       Out_voltage0_temp=1023;
            }
       }
    }
    else if(!K_MIN)
    {
       delay(30);
      if(!K_MIN)
      {
            if(Out_voltage0_temp<2)
            Out_voltage0_temp=2;
            Out_voltage0_temp=Out_voltage0_temp-2;
            if(U2>0)
                 U2--;
      }
```

```
        }
        else if(!K_Start)
        {
              delay(50);
         if(!K_Start)
           {
              if(!lock)
              {

                    hz_lcdDis(0,11,"Stop!");
                    SD_1=0;lock=1;
              }
              else
              {

                   hz_lcdDis(0,11,"Start");
                   SD_1=1;              lock=0;
              }

           }
        }
    }
```

函数中 if(U2<(InVoltage*0.8)意为设置输出最大电压值时输入电压的 80%，这里是限制作用，防止过大出错。U2 为液晶显示输出电压值，这里是放大十倍的，便于加减运算。Out_voltage0_temp 是用于 pid 运算的 AD 比较值，采集到的输出电压端的分压值 AD 为 Out0_voltage，我们在按键设置函数中将设置 AD 比较值的步进为 2，说明 Out_voltage0_temp 每增加 2，即代表设置输出电压值加 0.1V，同理 Out_voltage0_temp 每减少 2，即设置输出电压减小 0.1V。根据输出电压计算公式：输出电压=(R7+1)*5000*(AD/1024)mV; (R7+1)=10.24。即电位器 R7 的应调节为=9.24k，由于 9.24 为特殊阻值，故选用电位器。

Lock 为输出控制标志，SD_1 为驱动 ic 控制端，SD_1=1，即高电平时驱动 IC 输出有效。SD_1=0；即 SD_1 为低电平时驱动 ic 输出无效，均输出低电平。

3. 液晶显示函数

液晶显示的驱动函数这里不再赘述，具体驱动函数可参考 LCD1602 液晶驱动说明等网络文档资料，这里主要说明液晶显示的参数。

IN：输入电压值（单片机测量得出）。 Start 或 Stop 为输出状态指示，Start 为正常输出，Stop 代表关闭输出。" OCP "为过流保护状态，" OVP "为输入过压保护状态，" UVP "为输入欠压状态。

OU：输出电压值，此值为按键调节设置值。 I： 输出电流值（单片机测量得出）。

具体显示函数：

```
/*参数显示*/
num_lcdDis(0,3,InVoltage/10,2);     //显示输入电压
num_lcdDis(0,6,InVoltage%10,1);     //显示输入电压
num_lcdDis(1,3,U2/10,2);            //显示输出电压
num_lcdDis(1,6,U2%10,1);            //显示输出电压
num_lcdDis(1,11,OutCurrent/1000,1);   //显示输出电流
num_lcdDis(1,13,OutCurrent/10%100,2);//显示输出电流
/*状态判断显示*/
if(OverCurrent==1)//过流保护
{
    OverCurrent=0;
    SD_1=0;
    lock=1;
    hz_lcdDis(0,11," OCP ");
}
if(InVoltage>400)//过压保护
{
    SD_1=0;
    lock=1;
    hz_lcdDis(0,11," OVP ");
}
else if(InVoltage<150)//欠压保护
{
    SD_1=0;
    lock=1;
    hz_lcdDis(0,11," UVP ");
}
```

4. 输入电压、输出电压、输出电流采集

输入电压、输出电压、输出电流的采集均采用单片机内部的 AD（10 位）来检测，输入电压的检测采用电阻分压（由于输入电压超过单片机自身供电电压，所以必须采用电阻分压形式），测得分压值后，按照电阻比例推算出输入电压值，同理采集输出电压也采用输出电阻分压方式。

输出电流的采集，将 0.1 欧姆的功率电阻串入输出回路的地端。电流采样电阻分压所得信号过小，不利于单片机有效测量（0.1 欧姆的电阻流过电流为 3A 时，电阻分压才为 0.3V，单片机 AD 测量范围为 0-5V，可见没有有效发挥单片机的测量量程），那么经 LM358 运放放

大电路将小信号电压进行放大后后再由单片机 AD 检测即可。

AD 使用前要初始化：

```
/*********************************************************
函数说明： AD 初始化函数
*********************************************************/
void InitADC()
{
    P1M1=0x07; //设置 P1 口的 P10、P11、P12 为高阻输入模式
    P1M0=0x00; //设置 P1 口的 P10、P11、P12 为高阻输入模式
    P1ASF=0X07;//选择 ADC 通道 0 1 2   //0000 0111
    ADC_RES=0; //A/D 转换结果高 8 位 清零
    ADC_CONTR=ADC_POWER | ADC_SPEEDLL | ADC_START;
     //配置 ADC 电源开关、采集速度、开始采集
}
```

分别将 P10、P11、P12 通道的 AD 开关打开。

ADC_CONTR：ADC控制寄存器

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	name	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0

CHS2/CHS1/CHS0：模拟输入通道选择，CHS2/CHS1/CHS0

CHS2	CHS1	CHS0	Analog Channel Select（模拟输入通道选择）
0	0	0	选择 P1.0 作为A/D输入来用
0	0	1	选择 P1.1 作为A/D输入来用
0	1	0	选择 P1.2 作为A/D输入来用
0	1	1	选择 P1.3 作为A/D输入来用
1	0	0	选择 P1.4 作为A/D输入来用
1	0	1	选择 P1.5 作为A/D输入来用
1	1	0	选择 P1.6 作为A/D输入来用
1	1	1	选择 P1.7 作为A/D输入来用

对应采集 AD 值函数：

```
/*---------------------------
获取 ADC 值
---------------------------*/
WORD GetADCResult(BYTE ch)
{
    WORD ADC_Result;
    ADC_CONTR = ADC_POWER | ADC_SPEEDL | ch | ADC_START;
    _nop_();                          //延时等待
    _nop_();
    _nop_();
    _nop_();
```

```
    while (!(ADC_CONTR & ADC_FLAG));//等待转换完成标志
    ADC_CONTR &= ~ADC_FLAG;              //清除完成标志
    ADC_Result=ADC_RES;                  //得到高 8 位数据
    ADC_Result=ADC_Result<<2|ADC_RESL; //加上低 2 位数据
    return ADC_Result;                   //返回结果
}
```

从上图函数中可以看到 ADC_SPEEDL 是代表一个 AD 采集速度的值，已在文件中对其做出定义：#define ADC_SPEEDL 0x20 //360 clocks

速度配置寄存器说明如下：

SPEED1，SPEED0：模数转换器转换速度控制位

SPEED1	SPEED0	A/D转换所需时间
1	1	90个时钟周期转换一次，CPU工作频率21MHz时，A/D转换速度约250KHz
1	0	180个时钟周期转换一次
0	1	360个时钟周期转换一次
0	0	540个时钟周期转换一次

STC12C5A60S2系列单片机的A/D转换模块说使用的时钟是内部R/C振荡器所产生的系统时钟，不使用时钟分频寄存器CLK_DIV对系统时钟分频后所产生的供给CPU工作所使用的时钟.

好处：

这样可以让ADC用较高的频率工作，提高A/D 的转换速度

这样可以让CPU用较低的频率工作，降低系统的功耗

代表利用 360 个时钟周期可以进行一次 AD 转换，也就转换速率为 62.5K。此函数有输入参数 ch，即选择要采集的 AD 通道。

接下来看采集输入电压：

```
/*************************************************************
函数说明： FIFO 滤波函数
*************************************************************/
#define FILTER_N 32
int filter_buf[FILTER_N + 1];
int Filter1() {
  int i;
  int filter_sum = 0;
  filter_buf[FILTER_N] = InVoltage_ad_temp;     //  InVoltage_ad_temp 由定时器中断获取
  for(i = 0; i < FILTER_N; i++) {
    filter_buf[i] = filter_buf[i + 1]; // 所有数据左移，低位仍掉
    filter_sum += filter_buf[i];
  }
  return (int)(filter_sum>>5);//(filter_sum / FILTER_N);
}
InVoltage=Filter1()*54;
```

这里 Filter1()是先进先出的滤波函数，进行了 32 次 AD 采集值的求和求平均值，目的为通过数字滤波，以确保电压数值的稳定，输入电压采集的公式推算我们在硬件部分已经介绍，这里不再赘述。

输出电压采集，由于没有使用采集的输出电压进行显示，仅用到了测量输出电压，该通道 AD 值如下：

```
Out0_voltage=GetADCResult(2);//输出电压采集通道 AD 值
```

输出电流采集：

```
int filter_buf2[FILTER_N + 1];
int Filter2() {
  int i;
  int filter_sum = 0;
  filter_buf2[FILTER_N] = Current_ad_temp;//   Current_ad_temp  由定时器中断获取
  for(i = 0; i < FILTER_N; i++) {
    filter_buf2[i] = filter_buf2[i + 1]; //  所有数据左移，低位仍掉
    filter_sum += filter_buf2[i];
  }
  return (int)(filter_sum>>5);//(filter_sum / FILTER_N);
}
OutCurrent=Filter2()*4.87;   //mA
```

这里 Filter2()同样是先进先出的滤波函数，进行了 32 次 AD 采集值的求和求平均值，目的为通过数字滤波，以确保电流数值的稳定，输出电流采集的公式推算我们在硬件部分已经介绍，这里不再赘述。

5. 输入欠压、输入过压、输出过流保护

输入欠压保护，在主循环函数中实现，检测输入电压小于 15V 时为欠压状态，150 为放大 10 倍的电压值。为什么要欠压保护，由于电源输入端是采用线性稳压电路，对输入电压要求在 12V 以上，这里模块按照 15V 为最低工作电压。具体处理函数如下：

```
else if(InVoltage<150)//欠压保护
{
    SD_1=0;              //关闭驱动 IC 输出
    lock=1;              //锁定状态
    hz_lcdDis(0,11," UVP "); //显示
}
```

输入过压保护，目的为保护电源输入的稳压器件，参照稳压电路中的三极管手册，这里将输入过压阈值设为 30V，即检测到输入电压大于 30V 时则关闭输出。具体处理函数如下：

```
if(InVoltage>300)//过压保护
{
    SD_1=0;              //关闭驱动 IC 输出
    lock=1;              //锁定状态
    hz_lcdDis(0,11," OVP "); //显示
}
```

过流保护即为保护功率器件开关管设置的，当单片机的采集电流通道 AD 值达到过流值 2A 的时候，即关断输出，从而达到保护元件目的。

过流检测功能在定时器中断函数中扫描，当检测到电流大于 2A（对应 AD 值为），关闭

驱动 IC 输出，过流标志置位，锁定状态标志，锁定后 PID 不在运算，输出电压 0V，起到保护作用。具体处理函数如下：

```
if(Current_ad_temp>410)//过流了 >2A        2000mA/4.87=410;
{
OverCurrent=1;       //过流标志，用于显示
SD_1=0;              //关闭驱动 ic 输出
lock=1;              //锁定状态
}
```

主循环判断过流标志，显示“OCP”。

```
if(OverCurrent==1)//过流保护
{
    OverCurrent=0;
    SD_1=0;
    lock=1;
    hz_lcdDis(0,11," OCP ");
}
```

6. Pid 恒压算法控制

数字电源进行稳压输出的核心技术是通过输出反馈，单片机进行占空比控制，这里的反馈指的是单片机采集输出电压，然后与设定的目标输出电压相比较，通过一定的算法处理后来改变 pwm 占空比的输出。这里的算法可以是比较简单的比较判断输出，例如检测到输出值大于设定值时，减小占空比，反之则增大占空比。这种方法在一些对响应速度要求不高或纹波要求不高的情况下可以使用，如果要求更快的稳压控制以及较低的纹波电压，那么使用 PID 调节调节才是更适合的且常用的算法，下面是两种类型的 PID 算法 ，位置式和增量式。

- **位置式 PI：**

$$u(k) = K_p e(k) + K_i T_{sam} \sum_{i=1}^{k} e(i) = K_p e(k) + u_i(k) = K_p e(k) + K_i T_{sam} e(k) + u_i(k-1)$$

式中 T_{sam} ——采样周期。

可以看出，比例部分只与当前的偏差有关，而积分部分则是系统过去所有偏差的累积。位置式 PI 调节器的结构清晰，P 和 I 两部分作用分明，参数调整简单明了。

但直观上看，要计算第 k 拍的输出值 $u(k)$ ，需要存储 $e(0)\ldots e(k)$ 等每一拍的偏差，当很大时，则占用很大的内存空间，并且需要花费很多时间去计算，这是目前书籍及网络上普遍认为的位置式 PI 的缺点。

然而在具体编程操作中，可在每一拍对积分部分进行累积，再加上当前拍的比例部分，即为当前 的输出，根本不需要大量的内存空间；另外由于输出有可能超过允许值，因此需要对输出进行限幅，而当输出限幅的时候，积分累加部分也应同时进行限幅，以防输出不变而积分项继续累加，也即所谓的积分饱和过深。

- **增量式 PI：**

由位置式 PI 的式子可知，PI 调节器的第（k-1）拍（也即上一拍）输出为

$$u(k-1)=K_pe(k-1)+K_iT_{sam}\sum_{i=1}^{k-1}e(i)$$

两式相减，可得出 PI 调节器输出增量

$$\Delta u(k)=u(k)-u(k-1)=K_p[e(k)-e(k-1)]+K_iT_{sam}e(k)$$

上式仅仅为增量，只需要当前的和上一拍的偏差即可得出结果，不需要存储每一拍的偏差，因此占内存空间小，这也是普遍认为的增量式的优点。

然而很多场合下需要的往往不只增量，还有上一拍的输出值，于是可知增量式 PI 调节器算法为

$$u(k)=u(k-1)+\Delta u(k)=u(k-2)+\Delta u(k-1)+\Delta u(k)=u(0)+\Delta u(1)+\ldots+\Delta u(k)$$

由于 $u(0)=0$，在具体编程操作中，对每一拍的 $\Delta u(k)$ 进行累积，即为 PI 调节器的输出；同样地，为了避免超过允许值，仅需对输出限幅即可。

对应代码如下：

```
/*******************增量式 PID 控制
**********************************************************
一切的一切，目的就是：以最快的速度到目标点，然后稳住！
有位大师（韩京清）这么说 PID，I 是控制过去，P 是控制现在，D 是控制未来！
函数功能：增量 PI 控制器
入口参数：测量值，目标值
返回  值：  PWM
根据增量式离散 PID 公式
pwm+=Kp[e（k）-e(k-1)]+Ki*e(k)+Kd[e(k)-2e(k-1)+e(k-2)]
e(k)代表本次偏差
e(k-1)代表上一次的偏差  以此类推
pwm 代表实际输出：增量进行累积即为 PI 调节器输出
在我们的控制闭环系统里面，只使用 PI 控制，
不使用微分 D 作用（微分主要用于对输出惯性比较大的控制系统）
pwm+=Kp[e（k）-e(k-1)]+Ki*e(k)
    Target:        目标值
    Encoder:       测量值
    Bias:          当前偏差
    Last_bias:     上次偏差
    Kp:            比例系数
    Ki:            积分系数
    Pwm:           输出值
**************************************************************************/
int Incremental_PI (int Encoder,int Target)
{
```

```
    Bias=Target-Encoder;                    //求出速度偏差，由测量值减去目标值。
    Pwm+= Kp*(Bias-Last_bias)+Ki*Bias;   //使用增量 PI 控制器求出  PWM。
    Last_bias=Bias;                 //保存上一次偏差
    return Pwm>>5;              //（这里做移位处理的目的是将输出值缩小后使用）
}
```

定时器中断函数中采集输入电压、输出电流的 AD 值，通过 AD 值计算输入电压及输出电流的函数放在主循环执行，目的是不在定时器中断中处理耗时较多的函数，避免定时中断函数处理用时超时，影响 pid 运算。定时中断函数中同时采集输出电压的 AD 值，我们要将 PID 运算函数放入定时器中断函数中定时执行，将采集到的 AD 值作为输入参数送入 Incremental_PI 函数中运算。以下为定时器中断函数实现代码：

```
/************************************************************
函数说明：  Timer0 中断
实现功能：电压反馈 PID 运算控制输出 pwm。    采集输出电压 AD、输出电流 AD、输入电压 AD。
************************************************************/
void Timer0() interrupt 1       //5ms  进入执行一次
{
    int speed=0;
    TL0 = 0x00;          //设置定时初值
    TH0 = 0xEE;          //设置定时初值

    GetOutCurrent(); //采集输出电流 AD 值
    GetInvoltage();       //采集输入电压 AD 值
     Out0_voltage=GetADCResult(2);//输出电压采集通道 AD 值
    if(Current_ad_temp>410)//过流了 >2A        2000mA/4.87=410;
    {
    OverCurrent=1;       //过流标志，用于显示
    SD_1=0;                 //关闭驱动 ic 输出
    lock=1;                 //锁定状态
    }
    if(!lock) //非锁定状态：正常输出运行
    {
        speed=Incremental_PI(Out0_voltage,Out_voltage0_temp);
        if(speed>230) //  限制最大占空比  90%       255*90%
        speed=230;
        if(speed<1)
        speed=1;
        PWM_temp_0=speed;
        CCAP1H=CCAP1L=255-PWM_temp_0;   //这是值指的是 pwm 的低电平时间
    }
    else   //输出锁定状态：
     {
       PWM_temp_0=10; //占空比固定
```

```
            CCAP1H=CCAP1L=255-PWM_temp_0;     //这是值 pwm 的低电平时间
            Pwm=0; Last_bias=0; //pid 参数清零
          }

    }
```

下面我们介绍如下进行 pid 参数整定，一般步骤如下

a.确定比例系数 Kp

确定比例系数 Kp 时，首先去掉 PID 的积分项和微分项，一般是令 Ki=0、Kd=0，使 PID 为纯比例调节。输入设定为系统允许的最大值的 60%～70%，即目标电压先固定设置为 19V（输入电压为 30V，占空比限制输出为 90%，则理论输出电压小于 27V，那么按照 27V 的 70%为固定目标电压即 19V 来调试），由 0 逐渐加大比例增益 Kp（注意：一般 pid 调节中 Kp Ki Kd 均采用浮点小数，这样能保证调节精度，但是受限于 51 单片机处理浮点数耗时较长，这里将浮点数增大为整数，同样可以在保证调节精度的条件下提高单片机执行 pid 运算的速度，只需在 pid 输出结果进行相应缩小即可），直至系统出现振荡；再反过来，从此时的比例系数 Kp 逐渐减小，直至系统振荡消失，记录此时的比例系数 Kp，设定 PID 的比例系数 Kp 为当前值的 60%～70%。比例系数 Kp 调试完成。

b.确定积分时间常数 Ki

比例系数 Kp 确定后，设定一个较大的积分时间常数 Ki 的初值，然后逐渐减小 Ki，直至系统出现振荡，之后在反过来，逐渐加大 Ki，直至系统振荡消失。记录此时的 Ki，设定 PID 的积分时间常数 Ki 为当前值的 150%~180%。积分时间常数 Ki 调试完成。

c.确定微分时间常数 Kd

微分时间常数 Td 一般不用设定，为 0 即可。若要设定，与确定 Kp 和 Ki 的方法相同，取不振荡时的 30%。

d.系统空载、带载联调，再对 PID 参数进行微调，直至满足要求。

由以上步骤，实际调试得出的 Kp=1，Ki=5;

到此，关于 STC12C5A 单片机 Buck 电源板的软硬件设计已全部讲解完毕，在此希望能帮助到大家，如有疑问，欢迎咨询技术 qq: 2067054198。

2020-02-12

新乡科美电子技术有限公司