官方店铺：https://shop111407010.taobao.com

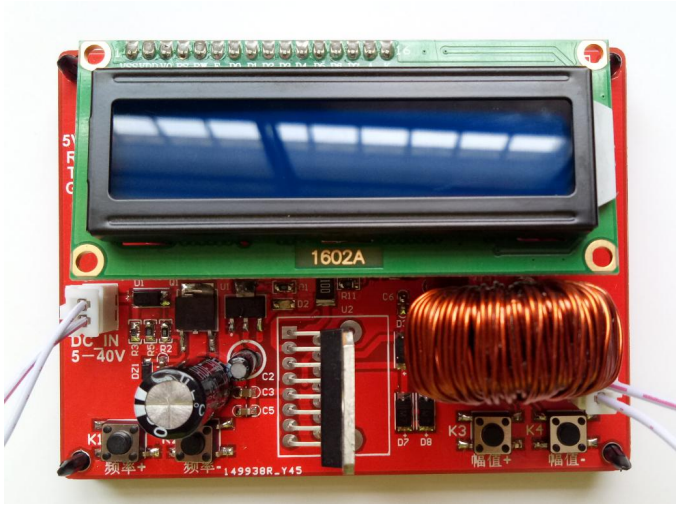
设计单片机 SPWM 逆变电路，由软件部分和硬件电路组成。

一．设计硬件电路：

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

二．编写软件

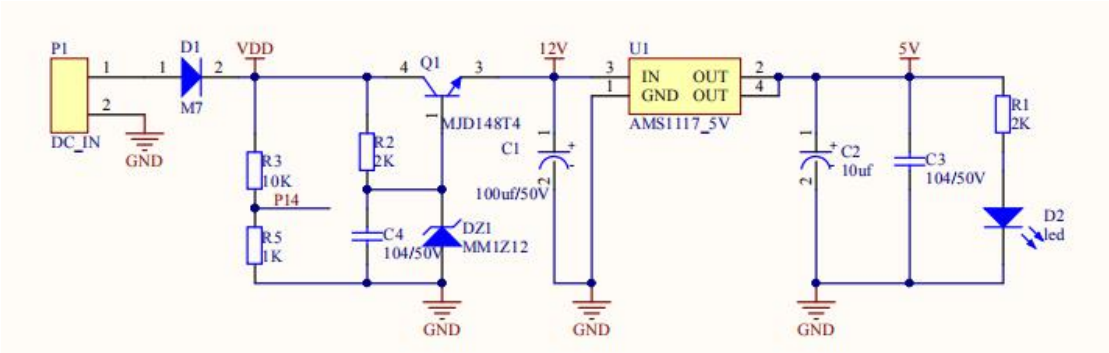
1. SPWM 脉宽数据生成函数。
2. 正弦波输出频率设置函数。
3. PWM 输出函数。
4. 按键功能。
5. 串口数据解析。
6. 液晶显示。
7. 输入电压、输出电流采集。
8. 过流保护。



以下分别对应各部分讲解：

一．设计硬件电路

1.电源&2.输入电压采集



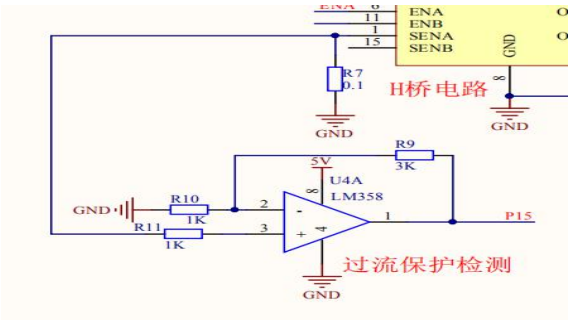
上图为本设计的电源电路，区中 D1 为二极管，其目的为防止正负极电源反接，由于二极管的正向导通特性，用户反接电源时亦不会烧坏后级电路。

图中 R3、R5 组成分压电路，由单片机 AD 采集 IO 口 P14 采集 AD 值，并由计算公式电压=11*5000*(AD/1024)mV;（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）。

图中 R2、C4、DZ1、Q1 组成线性稳压器，DZ1 为 12V 的稳压管，此电路的目的为将 VDD 稳压后为 12V 输出，此 12V 给 L298 提供驱动电源。关于此线性稳压电路原理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 L298 数字电路部分提供电源，D2 为电源指示灯。

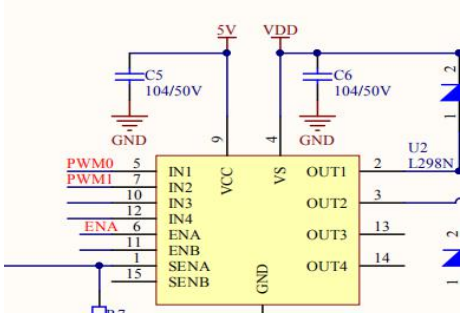
3. 输出电流采集



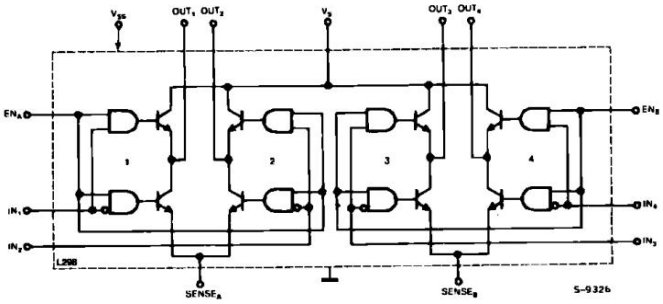
图中 L298 的 SENA 端口为其内部 H 桥电路的低端，我们将 R7 采样电阻串入其中，则全桥中的电流将会在此电阻上分压，此电压需要经过运放放大后送到单片机 AD 口检测。从电流采集的公式为：

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 4 倍送 AD 检测 2A 电流时,采样电阻分压为 0.2V,放大 4 倍为 0.8V. 对应 AD 值=(0.8/5)*1024= 163.84=164;

4. 驱动电路



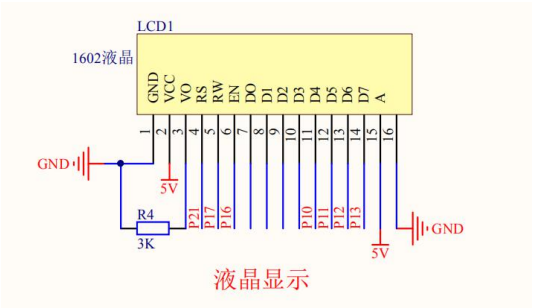
由于 L298 将 H 桥驱动电路集成到其芯片内布，所以只需要将单片机的两路 SPWM 直接和 L298 相连即可，ENA 是 L298 输出使能控制，高电平使能输出有效。



L298 内部集成电路

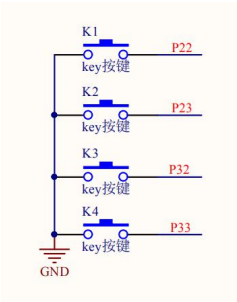
L298 将驱动电路（驱动 mos 管所必须的电平转换）集成在其内部，故我们电路无需再添加驱动电路。

5. 显示



显示采用 LCD1602 液晶，由于单片机 IO 口数量紧张，故使用 4 个 IO 来驱动，共使用 7 个 IO 来完成液晶驱动控制。R4 位液晶对比度调节电阻，3K 阻值为合适阻值。

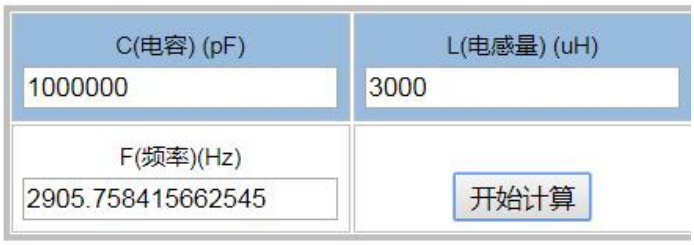
6. 按键



按键检测采用检测低电平有效，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控制。

7. 输出滤波

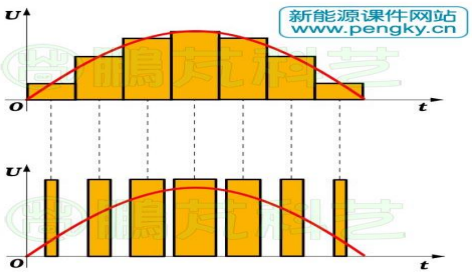
全桥逆变输出经过 LC 低通滤波器，即电感和电容组成的低通滤波器。在此我们仅以滤除基波以外的任意波为主要目的来说明，通过 LC 滤波电路截止频率公式：
应用公式: F(频率) = 1 / (2 * Pi * Sqrt(L*C))



https://tool.520101.com/dianlu/lcdianlu/
我们使用的电容=105=1uf=1000000pf，电感=3mH=3000uH。

二．编写软件
1.SPWM 脉宽数据生成函数。

首先我们要明白为什么要用 SPWM 波来驱动全桥电路就可以经过滤波即可得到正弦波，SPWM 波其实是一种按照正弦函数规律变化的 PWM 波。
为什么是 SPWM 波驱动？
面积等效原理转换
把直流电转换成正弦波交流电是根据根据面积等效原理，在下图的上图中的正弦半波（红线）分成 n 等份，把正弦半波看成是由 n 个彼此相连的矩形脉冲组成的波形，为简单清晰，划分为 7 等份。7 个脉冲的幅值按正弦规律变化，每个脉冲面积与相对应的正弦波部分面积相同，这一连续脉冲就等效正弦波。



如果把上述脉冲序列改为相同数量的等幅而不等宽的矩形脉冲（图 1 下图），脉冲中心位置不变，并且使该矩形脉冲面积和上图对应的矩形脉冲相同，得到下图所示的脉冲序列，脉冲宽度按正弦波规律变化，这就是 PWM 波形。根据面积等效原理，PWM 波形和正弦半波是等效的，图中红线就是该序列波形的平均值。

对于正弦波的负半周，也可以用同样的方法得到 PWM 波形。这种脉冲的宽度按正弦规律变化而和正弦波等效的 PWM 波形，也称 SPWM 波形。要改变等效输出的正弦波的幅值时，只需按照同一比例系数改变上述各脉冲的宽度即可。

```
//获取不同点数的正弦波数据
//point: 半周期内的取样点数
//maxnum: 半周期内对应PWM输出最大值
void getSinTab(uchar point,uint maxnum)
{
    uchar i=0;
    float x;   //弧度
    float jiao;//角度 分度角
    jiao=180.000/point;
    maxnum=maxnum*Para; //1v系数 *输出峰值电压= 峰值pwm
    if(maxnum>255)
       maxnum=255;
  for(i=0;i<point;i++)
    {
      x=jiao*i;     //得到角度值
      x=x*0.01744; //角度转弧度  弧度=角度*（π/180）
      pwm[i]=maxnum*sin(x)+0.5;//+0.5 对得到的的数据进行四舍五入

    }
}
```

在程序中，我们使用的是 8 位宽度的 PWM，半个正弦波周期采样点数为 50 个点，故我们采用的是单极性 SPWM，即当 SPWM 输出半周期完毕后，全桥中正在工作的一组开关管（例如：左桥臂上管和右桥臂下管）会关闭，然后另一组开始导通工作。故将 180 度细分为 50 个点中则每点对应的角度值为 jiao=180.000/point。对应输出正弦波的最大幅值的 PWM 脉宽为 255，则 Para=255/(Voltage/10)， Para 为输入电压细分出输出 1V 对应的 pwm 脉宽系数，（Voltage/10）为输入电压（单位：V）。 则设置峰值输出电压对应的峰值 pwm=1v 脉宽系数 *输出峰值电压=Para*maxnum。由于脉宽最大为 255，所以要对所得脉宽做限制处理，if(maxnum>255)　maxnum=255;接下来就由每个点对应的角度值来计算出对应的脉宽数据了：

```
for(i=0;i<point;i++)
  {
    x=jiao*i;      //得到角度值
    x=x*0.01744; //角度转弧度   弧度=角度*（π/180）
    pwm[i]=maxnum*sin(x)+0.5;//+0.5 对得到的的数据进行四舍五入
```

}

通过 void getSinTab(uchar point,uint maxnum)该函数，我们只需要改变其输入参数 maxnum 大小即可改变输出幅值大小。

2. 正弦波输出频率设置函数

首先我们知道一个正弦波输出周期包含了 100 个点，那么我们让这 100 个点在 20ms 内输出，那么对应一个正弦波输出的频率就是 50hz。这里我们开启定时器 0 中断服务函数来更新 SPWM 输出点，那么 50hz 正弦波输出对应的定时器 0 中断时间为 20/100=200us。

定时器 0 定时时间计算方法：

设定时器进中断时间为 cycle。

正弦波输出频率=1/(cycle*一周期正弦波点数*10 的负六次方)=1/(cycle×10 负四次方)，由此得出 cycle=10000/频率；

设定时器计数值为 timeinit，timeinit=cycle/（定时器计一个数所需的时间）=cycle/(1000000/11059200)；注：定时器采用 1T 模式计数，指令周期=系统晶振周期。

定时器初值即可得出：

timerH=(65536-timeinit)/256;

timerL=(65536-timeinit)%256;

综上所得：

```
//timeinit/1000 为定时器定时时间
//由于定时器最小定时限制，最低频率可设置为2hz
void getSinPlv(uint plv)
{
    float cycle=0.0000;
    unsigned  int timeinit=0;
    cycle=10000/plv;
    timeinit=cycle*11.0592+0.5; //得到的数据四舍五入
    timerH=(65536-timeinit)/256;
    timerL=(65536-timeinit)%256;

}
```

通过 void getSinPlv(uint plv)，改变输入参数 plv 大小即可改变输出正弦波频率。

3. PWM 输出函数

PWM 由单片机内部的 PCA 模块计数器产生，该 PCA 含有一个 CL 计数器，和 CCAP0L，CCAP0H 寄存器以及 CCAP1L，CCAP1H 寄存器，当 CL 的计数值小于 CCAP1L 值时，对应 PWM 输出口 P35 输出低，当 CL 的计数值等于或大于 CCAP1L 值时，对应 PWM 输出口 P35 输出高。当 CL 的值由 FF 溢出变为 0 时，CCAP1H 的值装载到 CCAP1L 中，这样就实现了无干扰更新了 PWM。（注：CCAP0L 和 CCAP0H 为 PWM0 寄存器，CCAP1L 和 CCAP1H 为 PWM1 寄存器）

PWM 输出的频率即载波频率，其频率是由寄存器 CMOD 决定：

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	-	-	-	-	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。

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。

CPS1	CPS0	选择PCA/PWM时钟源输入
0	0	0，系统时钟，SYSclk/12
0	1	1，系统时钟，SYSclk/2
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
1	1	3，ECI/P3.4脚输入的外部时钟（最大速率＝SYSclk/2）

我们设置寄存器为 CMOD=0x03;即 PWM 计时器采用 1/2 系统时钟，那么一周期 PWM 的频率等于 (SYSclk/2)/256=21.6Khz。

CCAPM0、CCAPM1 分别为 PWM0 和 PWM1 模式控制寄存器：

CCAPMn：PCA模块n(n=0, 1, 2, 3)的比较/捕获寄存器

SFR name	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCAPMn	name	-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn

PCA模块工作模式设定（CCAPMn寄存器，n = 0,1,2,3）

-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	模块功能
	0	0	0	0	0	0	0	无此操作
	X	1	0	0	0	0	X	16位捕获模式，由CCPn的上升沿触发
	X	0	1	0	0	0	X	16位捕获模式，由CCPn的下降沿触发
	X	1	1	0	0	0	X	16位捕获模式，由CCPn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
	1	0	0	0	0	1	0	8位PWM

这里我们设置　CCAPM0=0x42;//8 位 PWM 输出
　　CCAPM1=0x42;//8 位 PWM 输出

CR 为 PWM 计数器的运行开关，CR=1 时开始运行。

综合以上，我们得出 PWM 初始化函数：

```
void init_pca(void)//pca计数器初始化函数
{
CMOD=0x03;//  1/2系统时钟
CCON=0x00; //中断标志清零
CCAPM0=0x42;//8位PWM输出
CCAPM1=0x42;///8位PWM输出
CL=0x00;//清零pca计数器
CH=0x00;
CCAP0L=pwm[0];     //初始化spwm输出的占空比
CCAP0H=pwm[0];
CCAP1L=pwm[0];     //初始化spwm输出的占空比
CCAP1H=pwm[0];
CR=1;//运行pca计数器
}
```

4. 按键功能

按键的目的用来调节输出频率及幅值，之前我们已经将改变正弦波输出频率和幅值的函数封装好，只需改变其函数的输入参数即可改变对应的输出：

```
void keyscan()
 {
  if(!plvadd)
   {
       TimeCount2=0;
       while(!plvadd);
       if(PlvTemp<200)
          PlvTemp++;
           display();  //液晶显示
   }
  if(!plvmin)
   {
       TimeCount2=0;
       while(!plvmin);
       if(PlvTemp>2)
          PlvTemp--;
           display();  //液晶显示
   }
```

上图中函数为频率调节部分，通过按键可以改变 PlvTemp 的值大小，然后只要将 PlvTemp 更新到对应的 getSinPlv(PlvTemp)函数中即可。

```
    if(!voladd)
     {
       TimeCount2=0;
       while(!voladd);
       if(VolTemp<Voltage/10)
          VolTemp++;
           display();  //液晶显示
     }
    if(!volmin)
     {
       TimeCount2=0;
       while(!volmin);
       if(VolTemp>0)
          VolTemp--;
           display();  //液晶显示

     }
```

上图中函数为幅值调节部分，通过按键可以改变 VolTemp 的值大小，然后只需将 VolTemp 的值更新到对应的 getSinTab(50,VolTemp)函数中即可。

5. 串口数据解析

此模块设计有串口控制输出频率和幅值功能，故需要将串口中断打开来接收收到的控制指令。串口波特率设置为 9600，无校验，停止位 1 位，由定时器 1 提供波特率发生器，需要打开串口接收中断，具体配置初始化如下：

```
void UartInit(void)        //9600bps@11.0592MHz
{
    PCON &= 0x7F;        //波特率不倍速
    SCON = 0x50;         //8位数据,可变波特率
    AUXR |= 0x40;        //定时器1时钟为Fosc,即1T
    AUXR &= 0xFE;        //串口1选择定时器1为波特率发生器
    TMOD &= 0x0F;        //清除定时器1模式位
    TMOD |= 0x20;        //设定定时器1为8位自动重装方式
    TL1 = 0xDC;      //设定定时初值
    TH1 = 0xDC;      //设定定时器重装值
    ET1 = 0;         //禁止定时器1中断
    TR1 = 1;         //启动定时器1
    REN=1;         //允许串口接收
    ES=1;          //打开串口中断
}
```

我们定义的串口控制指令格式为：

第一字节	第二字节	第三字节
0xaa	0x01：控制频率 0x02：控制幅值	频率：0-200hz 幅值：0-40V

我们的上位机控制即是按照以上格式编写，如果想要使用串口调试助手直接发送指令控制，请按照以上格式发送即可控制。

下图为串口中断接收函数：

```
void serial()interrupt 4          //串口中断服务函数
  {
   static uchar j;
   ES=0;                  //禁止中断
   if(!RI);
   RI=0;               //清楚接收完毕标志
   date[j]=SBUF;
   if(date[0]==0xaa)   //判断接收到的第一位为上位机所发数据
      j++;
  else
      j=0;

   if(j==4)          //接收传感器所发数据9字节
     {
     j=0; recFlag=1;
     }
   ES=1;            //允许串口中断

  }
```

下图为接收到串口数据后的数据解析函数：

```
//接收串口数据解析
void getSetTemp()
{
 if(recFlag==1)
   {
      recFlag=0;
     if(date[1]==1) //修改频率
       {
        if(date[2]>200)
           date[2]=200;
           PlvTemp=date[2];
       }
        if(date[1]==2) //修改幅值
       {
           if(date[2]>Voltage)
              date[2]=Voltage;
          VolTemp=date[2];
       }
   }
```

recFlag 为接收到一组数据完成的标志符号。

同样的，VolTemp 和 PlvTemp 分别为幅值和频率调节值，只要更新到对应设置函数即可。

6. 液晶显示

液晶显示的驱动函数这里不再赘述，这里主要说明液晶显示的参数。

Input：输入电压值。　on 或 off 为输出状态指示，on 为正常输出，off 代表输出过流保护，已关闭输出。
Fr：输出频率值，此值为按键调节设置值。　AC：输出正向峰值电压值，此值为按键所设置值。
具体显示函数：

```
void display()
{
        WriteLcdInstr(0x80+7);
        WriteLcdData(LCD1602_Table[Voltage/100]); //显示输入电压值
        WriteLcdData(LCD1602_Table[Voltage%100/10]);
        WriteLcdData('.');
        WriteLcdData(LCD1602_Table[Voltage%10]);
        WriteLcdInstr(0x80+13);
    if(ShortFlag==1)
        {
            WriteLcdData('o'); //状态显示
            WriteLcdData('f');
            WriteLcdData('f');
        }
        else
        {
            WriteLcdData('o');
            WriteLcdData('n');
            WriteLcdData(' ');
        }
    WriteLcdInstr(0x80+0x40+3);
    WriteLcdData(LCD1602_Table[PlvTemp/100]);  //输出频率显示
    WriteLcdData(LCD1602_Table[PlvTemp%100/10]);
    WriteLcdData(LCD1602_Table[PlvTemp%10]);
    WriteLcdInstr(0x80+0x40+12);
    WriteLcdData(LCD1602_Table[VolTemp/100]);  //输出幅值显示
    WriteLcdData(LCD1602_Table[VolTemp%100/10]);
    WriteLcdData(LCD1602_Table[VolTemp%10]);
}
```

液晶显示在系统中是 1 秒钟刷新显示一次，通过定时器计数变量来判断：

```
        keyscan();  //按键检测
        if(TimeCount1>1)//1秒刷新一次
        {
            TimeCount1=0;
            display();  //液晶显示
        }
```

7. 输入电压、输出电流采集

输入电压、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。

在正弦波输出回路中串联一只 0.1 欧姆的功率电阻，电阻分所得信号过小，经 LM358 运放放大后再由单片机 AD 检测即可。

AD 使用前要初始化：

```
void InitADC()
{
    P1ASF = 0x30;                    //Open channels ADC function  4,5通道
    ADC_DATA = 0;                     //Clear previous result

    ADC_CONTR = ADC_POWER | ADC_SPEEDHH;
    Delayms(2);

}
```

分别将 P14、P15 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	C5H	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：

```
/*----------------------------
Get ADC result
----------------------------*/
WORD GetADCResult(BYTE ch)
{
    WORD ADC_Result;
    ADC_CONTR = ADC_POWER | ADC_SPEEDHH | ch | ADC_START;
    _nop_();                          //Must wait before inquiry
    _nop_();
    _nop_();
    _nop_();
    while (!(ADC_CONTR & ADC_FLAG));//Wait complete flag
    ADC_CONTR &= ~ADC_FLAG;          //Close ADC
    ADC_Result=ADC_DATA;
    ADC_Result=ADC_DATA<<2|ADC_LOW2;
    return ADC_Result;                  //Return ADC result
}
```

从上图函数中可以看到 ADC_SPEEDHH 是代表一个最高 AD 采集速度的值，已在文件中对其做出定义：　#define ADC_SPEEDHH 0x60　　//70 clocks

代表利用 70 个时钟周期可以进行一次 AD 转换，也就是大约 70us 可以转换一次。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：

```
        }
            for(i=0;i<16;i++)
            ADC_Process(ADC_CHANNEL4);
            Voltage=ADCResult>>1;     //电压=11*5000*(AD/1024)mV; //采集输入电压值
            if((Voltage-_voltage>10)||(_voltage-Voltage>10))
             {
               _voltage=Voltage;
               Para=255/(Voltage/10);  //由输入电压细分出输出1V 对应的pwm脉宽
               getSinTab(50,VolTemp);
             }
```

这里进行了 16 次连续的 AD 采集，目的为进行数字滤波（多次采集值之和求平均值），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。if((Voltage-_voltage>10)||(_voltage-Voltage>10)) ，这样一个判断的目的是当输入电压改变时，我们要相应的改变 1V 系数即 Para，然后对输出做出更新即 getSinTab(50,VolTemp);，因为我们的目的是当输入电压改变时保持输出电压不变。

接下来我们看输出电流采集：

```
        /*
        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4倍 送AD检测
        2A电流时,采样电阻分压为0.2V,放大4倍为0.8V. 对应AD值= (0.8/5)*1024= 163.84=164;
        */
            for(i=0;i<16;i++)
            ADC_Process(ADC_CHANNEL5); //过流值采集
            Current=ADCResult;
```

这里进行了 16 次连续的 AD 采集，目的为进行数字滤波（多次采集值之和求平均值），以确保所得数值的稳定。电流值的计算在硬件设计部分已介绍，这里不再赘述。

8. 过流保护

过流保护即为保护功率器件 L298 来设置的，当单片机 AD 采集到的 AD 值达到过流值 2A 后，即关断输出，从而达到保护全桥的目的。

过流保护后，我们设置了定时，过流后保护后，5 秒之后，单片机会再次尝试打开输出，如果还是过流则立刻关闭输出进行保护。5 秒后再次尝试打开，依次进行下去，这样就达成一种类似人们打嗝一样，反反复复。

接下来我们看具体实现：

```
        if(Current>164)//电流大于2A保护，停止输出
        {
            TimeCount=0;
            ShortFlag=1; //过流标识
            ENA=0; //关闭L298,即关闭了H桥
        }
        //采样电阻分压值为0.048/4=0.012V 对应电流为0.012/0.1=0.12A
else    if(Current<10)//电流小于0.12A时   输出
        {
             if(TimeCount>5) //5s
             {
                TimeCount=0;
                ShortFlag=0; //过流标志清除
                ENA=1;//开启输出
             }
        }
```

从上图函数可以看到，过流值 2A 时对应的 Ad 值为 164，ENA=0 即关闭输出使能。TimeCount 是一个以 1 秒为基数的计数变量，TimeCount>5 即代表 5 秒时间到。TimeCount 是在 PCA 作为定时器的定时中断服务函数中进行计时。PCA 计数器初始化配置如下：

```
void Pca_Timer()
{
    CCON = 0;                       //Initial PCA control register
                                    //PCA timer stop running
                                    //Clear CF flag
                                    //Clear all module interrupt flag
    CL = 0;                         //Reset PCA base timer
    CH = 0;
    CMOD=0x02;//  1/2系统时钟
    value = T100Hz;
    CCAP2L = value;
    CCAP2H = value >> 8;            //Initial PCA module-0
    value += T100Hz;
    CCAPM2 = 0x49;                  //PCA module-0 work in 16-bit timer mode and enable

    CR = 1;                         //PCA timer start run
    EPCAI = 1;
    EA = 1;
    cnt = 0;
```

此处配置与 PWM 模式配置的区别就在于，此处 PCA 模式为定时器模式，我们将其定时中断间配置为 10ms,PCA 定时频率=PCA 时钟/PCA 最大计数值=11059200/(11059200/2/100)=100hz =10ms。

PCA 定时器中断服务函数如下：

```
void PCA_isr() interrupt 6 using 1      //10ms
{
    CCF2 = 0;                       //Clear interrupt flag
    CCAP2L = value;
    CCAP2H = value >> 8;            //Update compare value
    value += T100Hz;
    if (cnt-- == 0) //1s
    {   led=~led; //调试用
        cnt = 100;                  //Count 100 times
        TimeCount1++;
        if((VolTemp!=_voltemp)||(PlvTemp!=_plvtemp))
        TimeCount2++;
            if(ShortFlag==1)//过流
               TimeCount++;
    }
}
```

```
if((VolTemp!=_voltemp)||(PlvTemp!=_plvtemp))
TimeCount2++;
```

这一句的判断作用：当检测到设置量 VolTemp 或 PlvTemp 发生改变时（由按键或串口设置的改变），TimeCount2 在累加，如果此时还有按键按下那么 TimeCount2 会在按键检测处被清零。

```
 {
  if(!plvadd)
   {
       TimeCount2=0;
       while(!plvadd);
       if(PlvTemp<200)
          PlvTemp++;
           display();  //液晶显示
   }
  if(!plvmin)
   {
       TimeCount2=0;
       while(!plvmin);
       if(PlvTemp>2)
          PlvTemp--;
           display();  //液晶显示
   }
```

其目的就是只有按键按下后 2S 内还没有按键操作时才会进行频率和幅值的更新：

```
        }
        if(TimeCount2>2)     //按键无操作2s后更新频率或幅值
        {
            TimeCount2=0;
            getSinPlv(PlvTemp);
            getSinTab(50,VolTemp);
            _voltemp=VolTemp;
            _plvtemp=PlvTemp;
        }
```

好了，至此硬件和软件的设计均已讲解完毕，感谢大家的支持！如有疑问请联系我们的技术支持 qq：2067054198。