摘 要

随着社会经济的发展，人民生活水平的提高，家庭供热的要求变的越来越高，在现代社会，供热系统朝着集中、智能化的方向发展。一个良好的供热系统应该具备节能、高效、安全的特点。因此，一个优秀的系统应该具备温度可设定、自动调整、高稳定、快速转换的优秀性质。对于这样一个供热系统，良好的数据采集和滤波是必不可少的，如果数据采集不够快速，那么这个系统就会有滞后性，不利于快速转换，也不利于稳定，如果数据采集误差很大，那么系统就会产生很大误差，而现实生活中也有很多干扰，我们应该根据干扰和噪声选择滤波算法，经典的滤波有十种，各有优点，根据具体情况选择。当然，对于这种控制系统，PID算法是重中之重，调节好PID参数之后，才可能拥有良好的调节性能。

以上，我们设计一个系统，由msp430f5529单片机、lcd12864、ads1118,、dac数模转换、按键这些主要组件组成的系统。由ads读取模拟量，单片机实现软件滤波和PID控制，通过lcd显示当前的气温室温和相关参数。通过按键修改温度设定等等。

**关键词：**温度控制**;** 单片机; PID算法; 滤波

**Abstract**

With the development of social economy and the improvement of people's living standard, the demand for household heating is becoming higher and higher. In modern society, the heating system is developing towards centralization and intelligence.A good heating system should have the characteristics of energy saving, high efficiency and safety.Therefore, an excellent system should have the excellent properties of temperature setting, automatic adjustment, high stability and quick conversion.For such a heating system, data acquisition and filtering is indispensable, if data acquisition fast enough, then the system will have lag, is not conducive to rapid transformation, is not conducive to stability, if data acquisition error is large, the system will generate a large error, and also has a lot of interference in real life, we should choose according to interference and noise filtering algorithm, the classical filter has ten kinds, each have advantages, according to the specific circumstances.Of course, for this kind of control system, PID algorithm is the top priority, after adjusting the PID parameters, may have good tuning performance.

Above, we design a system composed of msp430f5529 single-chip microcomputer, lcd12864, ads1118, dac digital-to-analog conversion, key and other main components.Ads reads the analog quantity, SCM realizes software filtering and PID control, and displays the current temperature, room temperature and related parameters through LCD.Changing the temperature setting by pressing the button and so on.

**Key words**: temperature control;Single chip microcomputer;PID algorithm.The filter

# 绪论

1. 前言

目前，温度控制系统被应用到生产和生活的方方面面。温度控制已经在工业、医疗、环境监测、家庭等诸多领域有所应用。同时，以单片机为载体的温度控制系统在电子产品中也越来越多。

现在有很多电子产品中时不时会用到温度检测和温度控制功能。而温度检测装置的种类不尽相同，比如常见的有温度检测模块的有电饭煲、微波炉、空调、电冰箱等等常规家电，用于温度控制和温度惊醒。采用msp430单片机对温度进行一定的控制，不仅具有控制方便、灵活性高、组态简单、价格低廉等优点。而且可以大幅度提高被控温度的技术指标和性能，从而能够大大提高产品质量和降低产品的成本。单片机由于其功能强大，体积小，可靠性高，价格低廉和开发周期非常短的优点，成为自动化和各个领域中广泛应用的器件，在日常生活中也逐渐成为必不可少的器件，它所发发挥的作用越来越大。因此，单片机对温度控制问题是我们生活中经常会碰到的一个问题

本论文根据上述问题，针对家庭集中供暖的问题，设计了以msp430f5529为核心的温度控制系统，ads1118为采集器，阀门电动机为执行器，根据要求来设计完成温度控制要求，设计过程比较顺利，单元电路设计，元器件的选择方面也比较考究。

温度控制系统的现状：

我经过网上的查询，翻阅相关的书籍，浏览相关文献，了解到目前以单片机为核心的温度控制系统还有很多，并且他们的方案比较灵活，应用面也比较广，可用于工业上的加热炉、热处理炉、反应炉。在日常生活中的应用也比较广泛，比如热水器、室温控制、温室大棚的温度控制。以上提到的温度控制的产品，根据其使用技术、技术指标、系统组成、功能特点选出其中具有代表性的几种如下：

1. 大棚温度控制系统

该设计通过结合土壤湿度、光强、温度等等条件进行综合温度控制，通过科学的控温，从而有利于农作物的生长，提高作物的产量，增加农夫的经济收益。该系统的最大优点是可以同时检测多个大棚的温度，从而进行最优温度控制。该系统内有单片机、串口通信、温度传感器和上位机（计算机）组成。计算机主要就是用于总控和监测，显示温度、报警和控制等功能；温度传感器对大棚内温度进行测量、显示；对单片机编程，使之能读取温度传感器的温度值，并且把读取到的温度传感器的值通过串口通信传递给上位机；串口通信模块的作用就是上位机和单片机之间的数据交互。

2.电烤箱温度控制系统

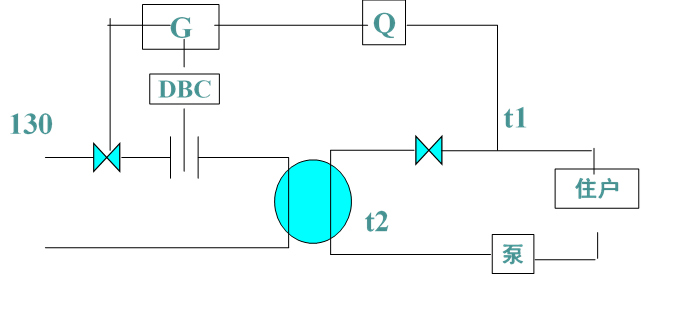
这个方案采用的是美国德州仪器公司生产的FLASH型超低耗16位单片机msp430f123作为核心器件，利用热电偶检测系统温度，用集成温度传感器AD590作为温度测量器件，利用该单片机内部的比较器完成高精度的AD信号采样，根据温度的变化情况，通过单片机编写闭环算法，从而实现温度的测量和温度的自动控制功能，他的测量范围比较低，大约在0-250之间，有高精度和响应速度较快的特点。

3.小型热水锅炉的温度控制系统

该设计方案可以解决北方冬季分散取暖时采用人工烧水供热，耗煤量大，浪费人力，温度变化大的问题。设计方案的硬件方面采用的是MCS-51系列单片机，扩展程序储存器2732，AD590温度监测元件测量环境温度和供水温度，ADC0809进行模数转换，同向驱动器7407、光电耦合器及9103的功放完成对电视的控制。软件方面建立了供暖系统数学模型，本系统硬件电路简单，软件程序易于实现，它可用于一台或多台小型取暖热水锅炉的温度控制，可使居室温度基本恒定，节煤，节电，省人力

1. 温度控制方案

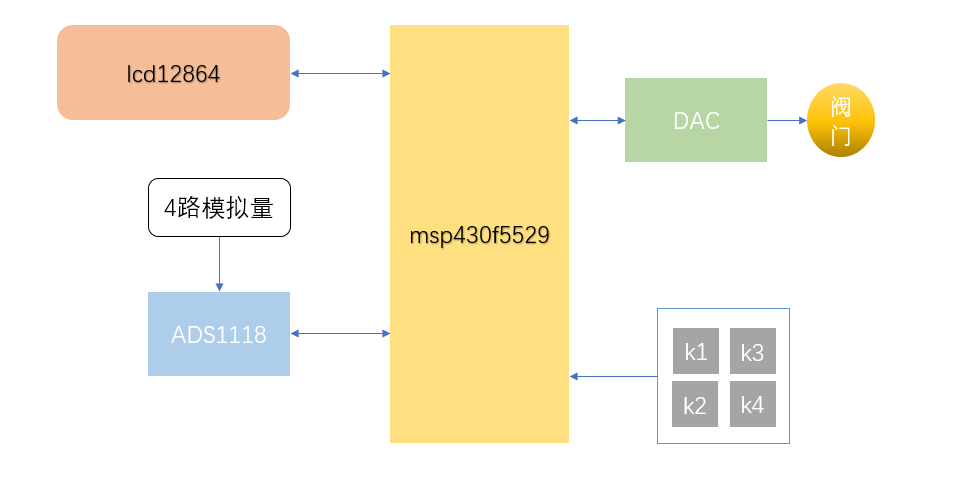
根据本设计的要求，总共有4路模拟量输入，分别为气温t1、室温t2、阀门开度g和管道压力d，通过ADS读取模拟量，msp430f5529经过软件滤波和转换得到合适的值，然后将值传入内部PID算法，PID算法会根据设定自动调整阀门达到温度控制的效果，一般时候，当室温远远低于设定值的时候（一般为10摄氏度左右），PID算法会根据室温和气温的温度差，直接用P算法（微分和积分的系数都为0）快速调整，当误差小于一定的范围内时，在加入微分和积分进行调整。



# 总体硬件设计

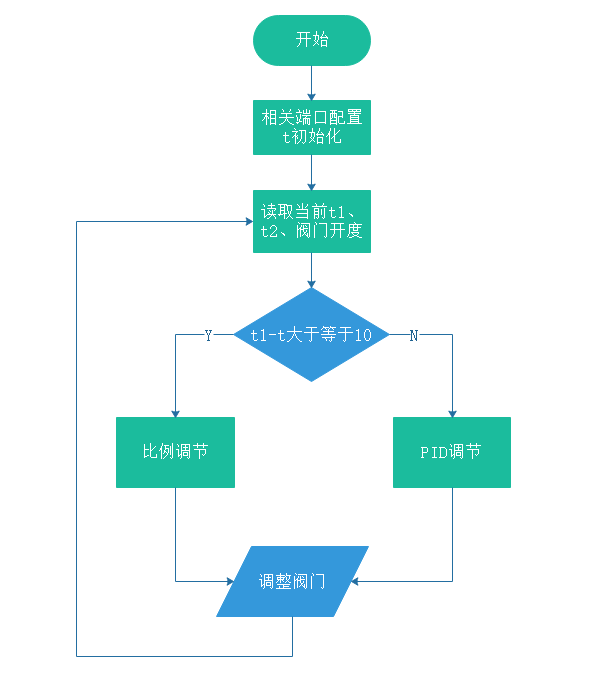
温度控制系统的主要硬件部分主要包括以下几个部分：msp430单片机、ads1118、DAC、lcd12864、独立按键、温度检测模块、步进电机

该系统通过温度模块检测室外温度t1室内温度t2，并且有温度设定值t，当室内温度t2与设定温度t温差较大时，msp430直接用比例控制快速调整，驱动步进电机（当做阀门开度），加大管道流速，使之t2快速提高，进入一定误差范围后，开启PID控制，此时主要影响室温的是气温t1，所以主要比较的是t1和t2，根据t1的变化来调整开度。Lcd上会显示当前的室外气温t1，室温t2，当前的温度设定值，阀门的开度，由于管道热水流和阀门以及室温和气温变化实际的实现较难，所以我们在msp430内部模，给阀门开度设一个值，手动设定气温。



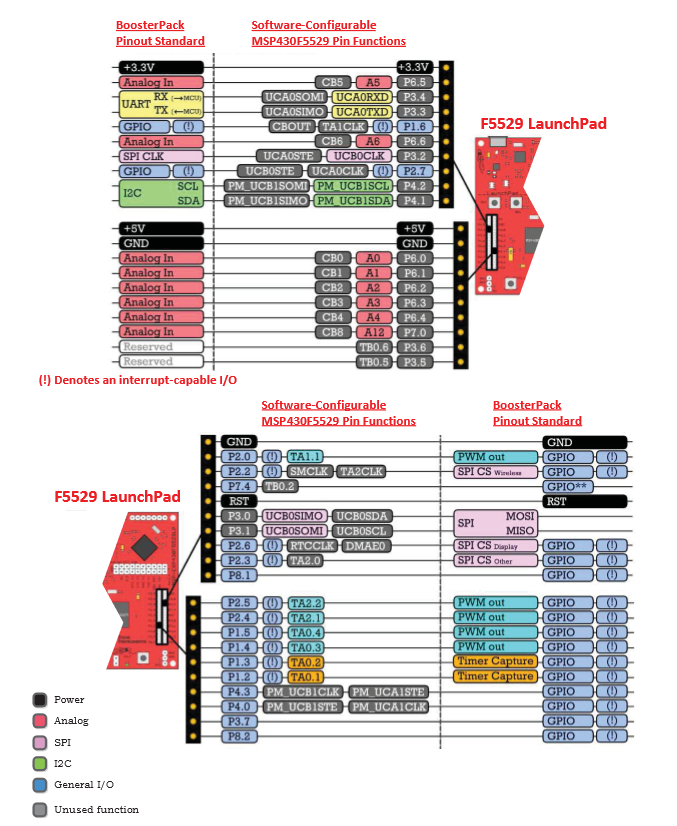
# 第三章 总体软件设计

该温控系统的核心PID控制，因此软件需要围绕PID进行编写，单片机首先进行初始化，初始化端口，初始化温度设定值t，然后检测t1和t2，进入PID控制，循环PID控制和检测t1、t2，我们来讨论集中重要的部分。



**1.msp430f5529的端口配置**

Msp430f5529端口简介：



我们这里选用了一些端口用于和外围器件的连接，用于和lcd通信的端口为P3.1-P3.7(用于并行数据口)、P6.6(用于RS/CS)、P6.5（用于RW/SID）、P6.4（用于EN/SCLK）。16位ADC模块P2.4(DIN)、P2.5(Dout)、P2.6（CS）、P2.7(SCLK)；独立按键P1.0（s1）、P4.7（S3）、P1.1（S4）、P2.1（S5）。

所以对于端口的初始化如下：

**LCD12862端口配置：**

/\*12864基础应用指令\*/

#define CLEAR\_SCREEN 0x01 //清屏指令：清屏且AC值为00H

#define AC\_INIT 0x02 //将AC设置为00H。且游标移到原点位置

#define CURSE\_ADD 0x06 //设定游标移到方向及图像整体移动方向（默认游标右移，图像整体不动）

#define FUN\_MODE 0x30 //工作模式：8位基本指令集

#define DISPLAY\_ON 0x0c //显示开,显示游标，且游标位置反白

#define DISPLAY\_OFF 0x08 //显示关

#define CURSE\_DIR 0x14 //游标向右移动:AC=AC+1

#define SET\_CG\_AC 0x40 //设置AC，范围为：00H~3FH

#define SET\_DD\_AC 0x80

/\*12864扩展应用指令\*/

#define SP\_MODE 0x34 // 8位数据，基本指令操作

#define GRAPON 0x36 // 扩充指令，图形显示开

#define gdbas 0x80

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 12864 画图、打点

//

#define LCD\_DataIn P3DIR=0x00 //数据口方向设置为输入

#define LCD\_DataOut P3DIR=0xff //数据口方向设置为输出

#define LCD2MCU\_Data P3IN

#define MCU2LCD\_Data P3OUT

#ifdef P\_eight

#define LCD\_CMDOut P8DIR |= (BIT1+BIT2);P7DIR |=BIT0 //P3口的低三位设置为输出

#define LCD\_RS\_H P8OUT|=BIT1 //8.1高

#define LCD\_RS\_L P8OUT&=~BIT1 //P8.1低

#define LCD\_RW\_H P8OUT|=BIT2 //P8.2高

#define LCD\_RW\_L P8OUT&=~BIT2 //P8.2低

#define LCD\_EN\_H P7OUT|=BIT0 //7.0高

#define LCD\_EN\_L P7OUT&=~BIT0 //P7.0

#endif

#ifdef P\_six

#define LCD\_CMDOut P6DIR|=0x70 //P6口 0111 0000三位设置为输出

#define LCD\_RS\_H P6OUT|=BIT6 //P6.6高

#define LCD\_RS\_L P6OUT&=~BIT6 //P6.6低

#define LCD\_RW\_H P6OUT|=BIT5 //P6.5高

#define LCD\_RW\_L P6OUT&=~BIT5 //P6.5低

#define LCD\_EN\_H P6OUT|=BIT4 //P6.4高

#define LCD\_EN\_L P6OUT&=~BIT4 //P6.4低

#endif

**ADS1118端口配置：**

#define SCLK\_H P4OUT |= BIT0

#define SCLK\_L P4OUT &= ~BIT0

#define CS\_H P4OUT |= BIT1

#define CS\_L P4OUT &= ~BIT1

#define DataIN\_H P4OUT |= BIT2

#define DataIN\_L P4OUT &= ~BIT2

#define uint unsigned int

#define uchar unsigned char

void ADS1118\_Init(void)

{

P4DIR |= BIT0;

P4DIR |= BIT1;

P4DIR |= BIT2; //输出，单片机写数据

P4DIR &=~ BIT3; //输入，单片机读数据

}

**2.msp430f5529软件滤波：**

我们在AD采集中经常要用到滤波，不同的情况下需要使用不同的滤波，它们各有优点，下面是十种经典的滤波方法的程序的优缺点：

## 1、限副滤波

　　A、方法：

　　根据经验判断，确定两次采样允许的最大偏差值（设为A）

　　每次检测到新值时判断：

　　如果本次值与上次值之差《=A，则本次值有效

　　如果本次值与上次值之差》A，则本次值无效，放弃本次值，用上次值代替本次值

　　B、优点：

　　能有效克服因偶然因素引起的脉冲干扰

　　C、缺点

　　无法抑制那种周期性的干扰

　　平滑度差

　　程序：

　　/\* A值可根据实际情况调整

　　value为有效值，new\_value为当前采样值

　　滤波程序返回有效的实际值\*/

　　#define A 10

　　char value;

　　char filter（）

　　{

　　char new\_value;

　　new\_value = get\_ad（）;

　　if （ （ new\_value - value 》 A ） || （ value - new\_value 》 A ） ）

　　return value;

　　else

　　return new\_value;

　　}

## 2、中位值滤波法

　　A、方法：

　　连续采样N次（N取奇数），把N次采样值按大小排列 ，取中间值为本次有效值

　　B、优点：

　　能有效克服因偶然因素引起的波动干扰，对温度、液位的变化缓慢的被测参数有良好的滤波效果

　　C、缺点：

　　对流量、速度等快速变化的参数不宜

　　程序：

　　/\* N值可根据实际情况调整

　　排序采用冒泡法\*/

　　#define N 11

　　char filter（）

　　{

　　charvalue\_buf［N］;

　　char count，i，j，temp;

　　for （ count=0;count《N;count++）

　　{

　　value\_buf［count］ = get\_ad（）;

　　delay（）;

　　}

　　for（j=0;j《N-1;j++）

　　{

　　for（i=0;i《N-j－1;i++）

　　{

　　if （ value\_buf》value\_buf［i+1］ ）

　　{

　　temp =value\_buf;

　　value\_buf = value\_buf［i+1］;

　　value\_buf［i+1］ = temp;

　　}

　　}

　　}

　　returnvalue\_buf［（N-1）/2］;

　　}

## 3、算术平均滤波法

　　A、方法：

　　连续取N个采样值进行算术平均运算

　　N值较大时：信号平滑度较高，但灵敏度较低

　　N值较小时：信号平滑度较低，但灵敏度较高

　　N值的选取：一般流量，N=12；压力：N=4

　　B、优点：

　　适用于对一般具有随机干扰的信号进行滤波

　　这样信号的特点是有一个平均值，信号在某一数值范围附近上下波动

　　C、缺点：

　　对于测量速度较慢或要求数据计算速度较快的实时控制不适用

　　比较浪费RAM

　　程序：

　　#define N 12

　　char filter（）

　　{

　　int sum = 0;

　　for （ count=0;count《N;count++）

　　{

　　sum + =get\_ad（）;

　　delay（）;

　　}

　　return （char）（sum/N）;

　　}

## 4、递推平均滤波法（又称滑动平均滤波法） （FIR前身）

　　A、方法：

　　把连续取N个采样值看成一个队列

　　队列的长度固定为N

　　每次采样到一个新数据放入队尾，并扔掉原来队首的一次数据。（先进先出原则）

　　把队列中的N个数据进行算术平均运算，就可获得新的滤波结果

　　N值的选取：流量，N=12；压力：N=4；液面，N=4~12；温度，N=1~4

　　B、优点：

　　对周期性干扰有良好的抑制作用，平滑度高

　　适用于高频振荡的系统

　　C、缺点：

　　灵敏度低

　　对偶然出现的脉冲性干扰的抑制作用较差

　　不易消除由于脉冲干扰所引起的采样值偏差

　　不适用于脉冲干扰比较严重的场合

　　比较浪费RAM

　　程序：

　　#define N 12

　　char value\_buf［N］;

　　char i=0;

　　char filter（）

　　{

　　char count;

　　int sum=0;

　　value\_buf［i++］ = get\_ad（）;

　　if （ i == N ） i = 0;

　　for （ count=0;count《N，count++）

　　sum+ = value\_buf［count］;

　　return （char）（sum/N）;

　　}

#p#副标题#e#

## 5、中位值平均滤波法（又称防脉冲干扰平均滤波法）

　　A、方法：

　　相当于“中位值滤波法”+“算术平均滤波法”

　　连续采样N个数据，去掉一个最大值和一个最小值

　　然后计算N-2个数据的算术平均值

　　N值的选取：3~14

　　B、优点：

　　融合了两种滤波法的优点

　　对于偶然出现的脉冲性干扰，可消除由于脉冲干扰所引起的采样值偏差

　　C、缺点：

　　测量速度较慢，和算术平均滤波法一样

　　比较浪费RAM

　　程序：

　　#define N 12

　　char filter（）

　　{

　　char count，i，j;

　　char value\_buf［N］;

　　int sum=0;

　　for （count=0;count《N;count++）

　　{

　　value\_buf［count］ = get\_ad（）;

　　delay（）;

　　}

　　for （j=0;j《N-1;j++）

　　{

　　for （i=0;i《N-j-1;i++）

　　{

　　if （ value\_buf》value\_buf［i+1］ ）

　　{

　　temp = value\_buf;

　　value\_buf = value\_buf［i+1］;

　　value\_buf［i+1］ = temp;

　　}

　　}

　　}

　　for（count=1;count《N-1;count++）

　　sum +=value［count］;

　　return（char）（sum/（N-2））;

　　}

## 6、限幅平均滤波法

　　A、方法：

　　相当于“限幅滤波法”+“递推平均滤波法”

　　每次采样到的新数据先进行限幅处理，

　　再送入队列进行递推平均滤波处理

　　B、优点：

　　融合了两种滤波法的优点 对于偶然出现的脉冲性干扰，可消除由于脉冲干扰所引起的采样值偏差

　　C、缺点：

　　比较浪费RAM

　　程序略 参考子程序1、3

## 7、一阶滞后滤波法

　　A、方法：

　　取a=0~1

　　本次滤波结果=（1-a）\*本次采样值+a\*上次滤波结果

　　B、优点：

　　对周期性干扰具有良好的抑制作用 适用于波动频率较高的场合

　　C、缺点：

　　相位滞后，灵敏度低 滞后程度取决于a值大小 不能消除滤波频率高于采样频率的1/2的干扰信号

　　程序：

　　/\* 为加快程序处理速度假定基数为100，a=0~100 \*/

　　#define a 50

　　char value;

　　char filter（）

　　{

　　char new\_value;

　　new\_value = get\_ad（）;

　　return （（100-a）\*value + a\*new\_value）;

　　}

## 8、加权递推平均滤波法

　　A、方法：

　　是对递推平均滤波法的改进，即不同时刻的数据加以不同的权

　　通常是，越接近现时刻的数据，权取得越大。

　　给予新采样值的权系数越大，则灵敏度越高，但信号平滑度越低

　　B、优点：

　　适用于有较大纯滞后时间常数的对象和采样周期较短的系统

　　C、缺点：

　　对于纯滞后时间常数较小，采样周期较长，变化缓慢的信号 不能迅速反应系统当前所受干扰的严重程度，滤波效果差

　　程序：

　　/\* coe数组为加权系数表，存在程序存储区。\*/

　　#define N 12

　　char code coe［N］ ={1，2，3，4，5，6，7，8，9，10，11，12};

　　char code sum\_coe = 1+2+3+4+5+6+7+8+9+10+11+12;

　　char filter（）

　　{

　　char count;

　　char value\_buf［N］;

　　int sum=0;

　　for （count=0，count《N;count++）

　　{

　　value\_buf［count］ = get\_ad（）;

　　delay（）;

　　}

　　for （count=0，count《N;count++）

　　sum += value\_buf［count］\*coe［count］;

　　return （char）（sum/sum\_coe）;

　　}

## 9、消抖滤波法

　　A、方法：

　　设置一个滤波计数器

　　将每次采样值与当前有效值比较：

　　如果采样值＝当前有效值，则计数器清零

　　如果采样值《》当前有效值，则计数器+1，并判断计数器是否》=上限N（溢出）

　　如果计数器溢出，则将本次值替换当前有效值，并清计数器

　　B、优点：

　　对于变化缓慢的被测参数有较好的滤波效果，

　　可避免在临界值附近控制器的反复开/关跳动或显示器上数值抖动

　　C、缺点：

　　对于快速变化的参数不宜

　　如果在计数器溢出的那一次采样到的值恰好是干扰值，则会将干扰值当作有效值导入系统

　　程序：

　　#define N 12

　　char filter（）

　　{

　　char count=0;

　　char new\_value;

　　new\_value =get\_ad（）;

　　while （value！=new\_value）

　　{

　　count++;

　　if （count》=N）return new\_value;

　　delay（）;

　　new\_value =get\_ad（）;

　　}

　　return value;

　　}

## 10、限幅消抖滤波法

　　A、方法：

　　相当于“限幅滤波法”+“消抖滤波法”

　　先限幅，后消抖

　　B、优点：

　　继承了“限幅”和“消抖”的优点

　　改进了“消抖滤波法”中的某些缺陷，避免将干扰值导入系统

　　C、缺点：

　　对于快速变化的参数不宜

　　程序略 参考子程序1、9

我们的系统并不是一个有周期性扰动的系统，扰动是随机的。鉴于我们主要是检测温度，所以我们选择中位值滤波法。

软件实现PID算法：

* **PID控制器的数字化**

　　PID控制器是工业过程控制中广泛采用的一种控制器，其中，P、I、D分别为比例（Proportion）、积分（Integral）、微分（Differential）的简写；将偏差的比例、积分和微分通过线性组合构成控制量，用该控制量对受控对象进行控制，称为**PID算法**。

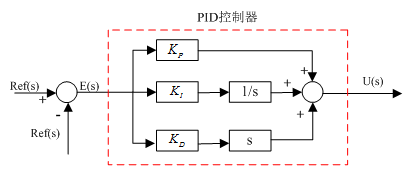
　　为了用软件实现PID算法，需将PID控制器离散化。

1. **整体思路**

****

　　2. **方框图**

　　PID控制器的**方框图**如图所示：



　　3. **拉氏域的表达式**

　　根据方框图，可写出PID控制器对应的**传递函数：**

https://images2015.cnblogs.com/blog/892317/201602/892317-20160207204638663-367974908.png

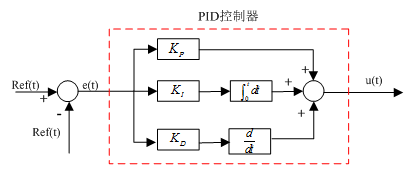
**（**其中，Kp为比例系数，ki为积分系数，Kd为微分系数**）**

　　4. **时域的表达式**

　　在分析时，通常借助于拉氏空间，例如判断系统的稳定性与相对稳定性；而现在我们关心的是时域里的问题，因此对上式进行**拉普拉斯逆变换**，得到**时域里的表达式：**

https://images2015.cnblogs.com/blog/892317/201602/892317-20160207204928507-1257742530.png

       其对应的**结构框图**如图所示：



　　5. **差分方程**

　　该时域里的表达式不便于编程处理，因此需对该式进行**离散化处理**，从而得到可编程实现的差分方程，**分析过程**如下：

**（说明：PID离散化的实质为微积分的离散化（数值化处理），由于这个推导过程很多教材上都有介绍，因而略去推导过程，只给出最终表达式，程序的算法就是基于此表达式而写的）**

**数字PID控制器的增量式算法：**

https://images2015.cnblogs.com/blog/892317/201602/892317-20160208222227589-1071929136.png

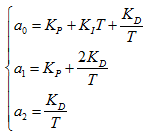
**（**其中，T为步长，即采样周期（**由微控制器的定时器确定**）**）**

**记**u(kT)=u(k)，便得到**PID控制器增量式算法的差分方程：**

https://images2015.cnblogs.com/blog/892317/201602/892317-20160208222621886-1419547327.png

　　这样就可编程实现了**（**或许有人会问，为什么差分方程就可编程实现呢？这是因为解差分方程的一般解法就是**迭代法**，而迭代法只需初值跟通项公式，这在计算机编程中很容易实现**）**

　　为使编程方便，可引入中间变量，定义如下：



　　则，**PID控制器增量式算法的差分方程变为：**

https://images2015.cnblogs.com/blog/892317/201602/892317-20160208231255245-1866540755.png

**说明：**

　　 （1）在PID增量式算法中只需对输出u(t)作**限幅处理**；

　　 （2）当微分系数 Kd=0 时，PID控制器就成了PI控制器**（在编写PID程序时默认使其为PI调节器）**；

　　   　　当积分系数 Ki=0 时，PID控制器就成了PD控制器。

=======================================================================================

* **基于微控制器的算法实现**

　　我写的数字PID程序如图所示（**在最后的附件部分**），有两套代码，一套是**直接函数调用（C/C++通用）**，另一套是**使用函数指针进行函数调用（仅适用于C）**，现从两个方面对该程序做讲解：

**（一）程序风格**

程序采用了**模块化编程**的思想，这样做的**目的**是**增强代码的可移植性及程序的可读性**。

　　程序被**拆分**成三个模块：

　　一个是PID的头文件’PID.h’：主要是定义算法实现有关的数据类型；

　　一个是PID的源文件’PID.c’：主要是定义算法实现的函数；

　　一个是主函数文件’amain.c’：PID程序的使用方法，即在主程序中做相应的初始化工作，在中断服务程序中进行PID的计算。

**说明：读这个程序时可能有点困难，不过这属情理之中的事，毕竟刚接触这种风格的童鞋不太能理解这种风格的产生（为什么这么做）及用意（这么做的好处）；我的建议是：在理解算法的原理后，根据自己的编程风格尝试着写一下，然后再跟这套程序对比着来理解，推敲一下别人为什么要这么做；当熟悉了整个流程后，你才能体会这种程序风格的优势，再将这种编程风格慢慢转化为自己的编程风格。**

**（二）程序中涉及的C语法讲解**

　　这里，我只讲述**为什么要采用这些语法**以及**采用这些语法所带来的好处**，至于细枝末节的问题，就请各位童鞋自行查阅有关资料，顺带给大家推荐一本不错的C语言教材：**C Primer Plus**，毕竟学习的兴趣浓度跟书籍的编排也有关。

**1. 条件编译指令**

第一处：#ifndef PID\_H语句

　　使用该语句的**目的**是避免造成把重复定义语句（如，结构体类型定义）添加到工程中，而使得编译出错

**说明：**其实也可不用#ifndef语句，因为每个定义的变量都具有特定的物理含义，不会造成重复定义现象。

　　第二处：#if (PID\_DEBUG) 语句

　　使用该语句的**目的**是**实现功能切换**（**注意了：**是在校正PID参数后手动切换，通过改变宏定义语句#define PID\_DEBUG 1中的宏体实现），具体请看程序清单。

**2. 结构体及结构体指针**

使用结构体类型的**好处**：可为实现某一功能的各变量进行“打包”处理

　　使用结构体指针的**好处**：通过传址调用，对方便对结构体变量本身进行操作

**3. typedef数据类型定义**

使用typedef数据类型定义的**好处**是方便**跨平台进行代码移植**操作；但由于教材的缘故，造成很多童鞋都停留在表面层次上的理解**(typedef 数据类型 别名)**，因而此处作重点讲解。

**我的理解：任何一个typedef声明中的标识符不再是一个变量，而是代表一个数据类型，其表示的数据类型为正常变量声明（去掉typedef）的那个标识符的数据类型。**

　　理解起来可能有点困难，现结合实例来讲解：

**[例1]**

　　typedef int Myint;

**分析：**

**第一步：正常变量声明（去掉typedef）**

　　　　int Myint;

　　该语句表示定义一个int型变量Myint**（**这里，Myint为变量名**）**；

**第二步：整体分析**

　　　　typedef int Myint;

　　该语句表示定义一个Myint类型**（**此时，Myint为数据类型标识符**）**，其具体所表示的类型：int型；

**应用：**

　　　　Myint a; //声明整型变量a

**[例2]**

　　typedef struct

　　{

　　 //省略成员

　　}PID;

**分析：**

**第一步：正常变量声明（去掉typedef）**

　　　　struct

　　　　{

　　　　 //省略成员

　　　　}PID;

　　该语句表示定义一个结构体变量PID**（**这里，PID为变量名**）**；

**第二步：整体分析**

　　　　typedef struct

　　　　{

　　　　//省略成员

　　　　}PID;

　　该语句表示定义一个PID类型**（**此时，PID为数据类型标识符**）**，其具体所表示的类型：结构体类型，且其具有的成员同结构体变量PID**（**这里，PID为变量名**）**；

**应用：**

　　PID ASR; //定义结构体变量ASR

**[例3]**

　　typedef void (\*PFun)(int );

**分析：**

**第一步：正常变量声明（去掉typedef）**

　　　　void (\*PFun)(int );

　　该语句表示定义一个函数指针PFun**（**这里，PFun为变量名**）**；

**第二步：整体分析**

　　　　typedef void (\*PFun)(int );

　　该语句表示定义一个PFun类型**（**此时，PFun为数据类型标识符**）**，其具体所表示的类型：函数指针类型，且其指向形参为int型，无返回值的一类函数；

**应用：**

　　　　PFun pf; //定义函数指针pf

**说明：**typedef的用法与宏定义#define的用法类似，但又有**区别**，体现在以下两点：

**(a)  typedef是对数据类型的定义，而#define是对数值的定义；**

**(b)  typedef由编译器解释，而#define由预处理器执行。**

**4. 空形参函数和形参带(void)函数**

　　这是在C/C++中相当**容易混淆**的地方，因此这里重点介绍一下，若是这个知识点没搞懂，那么这个程序你就无法看懂为什么会如此定义函数指针及利用函数指针来进行函数调用。

**void本身就是一种数据类型（空类型），把void作为形参时，表示这个函数不需要参数。**

**在C++中，空形参表与新参为void是等价的，这是C++中明确规定的；但在C中则是两回事：C中的空形参表仅表示函数的形参个数和类型不确定，并非没有参数，这会暂时挂起编译器的类型检查机制，从而造成类型安全隐患，所以在C中欲表示函数无形参时，最好用void，此时编译器将进行函数参数类型验证。**

**[例]**

[复制代码](javascript:void(0);)

void pid\_calc(int); //函数声明

void (\*calc\_1)(int); //函数指针声明

void (\*calc\_2)(); //函数指针声明

void main()

{

//将函数的入口地址赋给函数指针

calc\_1=pid\_calc; //C编译通过；C++编译通过

calc\_2=pid\_calc; //C编译通过；C++编译失败

}

[复制代码](javascript:void(0);)

**5. 函数指针及其函数调用**

　　函数调用，除了直接调用”函数名(实参)”这种语法外，还可通过函数指针来实现，两者并无区别，但为了**代码的紧凑性及美观性**，建议大家使用函数指针来进行函数调用。

　　在我放出的两套代码中，一套是**直接函数调用**（**C/C++通用**），另一套是**使用函数指针进行函数调用**（**仅适用于C**），大家可体会这两种用法的区别。

**6. 数据类型转换**

　　C语言中的数据类型分为自动类型转换与强制类型转换

　　(1) 自动类型转换**(由编译器完成)**

**（**自动转换的适用场合及其转换规则，请读者查阅有关资料**）**

　　(2) 强制类型转换**（**通过类型转换运算实现**）**

　　在本程序中，即可以将自定义函数的函数名pid\_calc**（函数名代表对应函数的入口地址）**直接赋值给函数指针calc，也可将自定义函数的函数名pid\_calc先强制类型转换**（**转换为函数指针**）**后，再赋值给函数指针calc；这两种方式虽说能达到同样的效果，但其所反映的思想却有所不同。

　　现把代码截取出来，方便大家对比：

[复制代码](javascript:void(0);)

void pid\_calc(PID \*p); //函数声明

void (\*calc)(); //函数指针：指向PID计算函数

void main()

{

//将函数的入口地址赋给指针变量

calc=(void (\*)(unsigned long))pid\_calc; //编译通过（强制类型转换）

calc=pid\_calc; //编译通过

}

[复制代码](javascript:void(0);)

**7. 代码换行问题**

为了**代码的美观**及**调试方便**，需涉及到**代码换行问题**。

　　在本程序的宏定义语句中使用了”\”，这是宏定义中连接上下行的连接符，表示该宏定义还未结束。

[复制代码](javascript:void(0);)

//定义PID控制器的初始值

#define PID\_DEFAULTS {0,0, \

0,0,0, \

0.0002, \

0,0,0, \

0,0,0, \

0,0,0,0, \

(void (\*)(unsigned long))pid\_calc}

[复制代码](javascript:void(0);)

=======================================================================================

**附件一：直接函数调用（C/C++通用）**

**PID.h文件**

[复制代码](javascript:void(0);)

//===================================================

//PID.h

//===================================================

#ifndef PID\_H

#define PID\_H

//定义PID计算用到的结构体类型

typedef struct

{

float Ref; 　　　　　　　　//输入：系统待调节量的给定值

float Fdb; 　　　　　　　　//输入：系统待调节量的反馈值

//PID控制器部分

float Kp; 　　　　　　　　 //参数：比例系数

float Ki; 　　　　　　　　 //参数：积分系数

float Kd; 　　　　　　　　 //参数：微分系数

float T; 　　　　　　　　 //参数：离散化系统的采样周期

float a0; 　　　　　　　　 //变量：a0

float a1; 　　　　　　　　 //变量: a1

float a2; 　　　　　　　　 //变量: a2

float Err; 　　　　　　　 //变量：当前的偏差e(k)

float Err\_1; 　　　 //历史：前一步的偏差e(k-1)

float Err\_2; 　　　　 //历史：前前一步的偏差e(k-2)

float Out; 　　　　　　 //输出：PID控制器的输出u(k)

float Out\_1; 　　 //历史：PID控制器前一步的输出u(k-1)

float OutMax; 　　　　 //参数：PID控制器的最大输出

float OutMin; 　　　　 //参数：PID控制器的最小输出

}PID;

//定义PID控制器的初始值

#define PID\_DEFAULTS {0,0, \

0,0,0, \

0.0002, \

0,0,0, \

0,0,0, \

0,0,0,0}

//条件编译的判别条件

#define PID\_DEBUG 1

//函数声明

void pid\_calc(PID \*p);

#endif

//===================================================

//End of file.

//===================================================

[复制代码](javascript:void(0);)

**PID.c文件**

[复制代码](javascript:void(0);)

//===================================================

//PID.c

//===================================================

#include "PID.h"

//===================函数定义========================

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*说 明：

\* （1）PID控制器默认为PI调节器

\* （2）使用了条件编译进行功能切换：节省计算时间

\* 在校正PID参数时，使用宏定义将PID\_DEBUG设为1；

\* 当参数校正完成后，使用宏定义将PID\_DEBUG设为0，同时，在初始化时

\* 直接为p->a0、p->a1、p->a2赋值

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void pid\_calc(PID \*p)

{

//使用条件编译进行功能切换

#if (PID\_DEBUG)

float a0,a1,a2;

//计算中间变量a0、a1、a2

a0=p->Kp+p->Ki\*p->T+p->Kd/p->T;

a1=p->Kp+2\*p->Kd/p->T;

a2=p->Kd/p->T;

//计算PID控制器的输出

p->Out=p->Out\_1+a0\*p->Err-a1\*p->Err\_1+a2\*p->Err\_2;

#else

//计算PID控制器的输出

p->Out=p->Out\_1+p->a0\*p->Err-p->a1\*p->Err\_1+p->a2\*p->Err\_2;

#endif

//输出限幅

if(p->Out>p->OutMax)

p->Out=p->OutMax;

if(p->Out<p->OutMin)

p->Out=p->OutMin;

//为下步计算做准备

p->Out\_1=p->Out;

p->Err\_2=p->Err\_1;

p->Err\_1=p->Err;

}

//===================================================

//End of file.

//===================================================

[复制代码](javascript:void(0);)

**amain.c主函数文件**

[复制代码](javascript:void(0);)

//===================================================

//amain.c

//===================================================

//将用户定义的头文件包含进来

#include "PID.h"

//=============宏定义=====================

#define T0 　　0.0002 　　　　　　//离散化采样周期，单位s

//============全局变量========================

//定义PID控制器对应的结构体变量

PID ASR=PID\_DEFAULTS; 　　　　　　//速度PI调节器ASR

//定义PID控制器的参数及输出限幅值

float SpeedKp=2,SpeedKi=1,SpeedLimit=10; 　//速度PI调节器ASR

//===============主程序=======================

void main()

{

//初始化PID控制器

ASR.Kp=SpeedKp;

ASR.Ki=SpeedKi;

ASR.T=T0;

ASR.OutMax=SpeedLimit;

ASR.OutMin=-SpeedLimit;

}

//============中断服务程序====================

interrupt void T1UFINT\_ISR(void)

{

//转速调节ASR

ASR.Ref=input1; 　　　　　　　　//速度给定

ASR.Fdb=input2; 　　　　　　　　//速度反馈

ASR.Err=ASR.Ref-ASR.Fdb; 　　 //偏差

pid\_calc(&ASR); 　　　　　　　　//函数调用：启动PID计算

output=ASR.Out; 　　　　　　　　//读取PID控制器的输出

}

//===================================================

//End of file.

//===================================================

[复制代码](javascript:void(0);)

=======================================================================================

**附件二：使用函数指针进行函数调用（仅适用于C）**

**PID.h文件**

[复制代码](javascript:void(0);)

//===================================================

//PID.h

//===================================================

#ifndef PID\_H

#define PID\_H

//定义PID计算用到的结构体类型

typedef struct

{

float Ref; 　　　　　　//输入：系统待调节量的给定值

float Fdb; 　　　　　　//输入：系统待调节量的反馈值

//PID控制器部分

float Kp; 　　　　　　 //参数：比例系数

float Ki; 　　　　　　 //参数：积分系数

float Kd; 　　　　　　 //参数：微分系数

float T; 　　　　　　 //参数：离散化系统的采样周期

float a0; 　　　　　　 //变量：a0

float a1; 　　　　　　 //变量: a1

float a2; 　　　　　　 //变量: a2

float Err; 　　　　　　//变量：当前的偏差e(k)

float Err\_1; 　　　　　//历史：前一步的偏差e(k-1)

float Err\_2; 　　 //历史：前前一步的偏差e(k-2)

float Out; 　　　　　　//输出：PID控制器的输出u(k)

float Out\_1; 　　 //历史：PID控制器前一步的输出u(k-1)

float OutMax; 　　　　//参数：PID控制器的最大输出

float OutMin; 　　　　//参数：PID控制器的最小输出

void (\*calc)(); 　　　//函数指针：指向PID计算函数

}PID;

//定义PID控制器的初始值

#define PID\_DEFAULTS {0,0, \

0,0,0, \

0.0002, \

0,0,0, \

0,0,0, \

0,0,0,0, \

(void (\*)(unsigned long))pid\_calc} //加与不加强制类型转换都没影响

//条件编译的判别条件

#define PID\_DEBUG 1

//函数声明

void pid\_calc(PID \*p);

#endif

//===================================================

//End of file.

//===================================================

[复制代码](javascript:void(0);)

**PID.c文件**

[复制代码](javascript:void(0);)

//===================================================

//PID.c

//===================================================

#include "PID.h"

//===================函数定义========================

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*说 明：

\* （1）PID控制器默认为PI调节器

\* （2）使用了条件编译进行功能切换：节省计算时间

\* 在校正PID参数时，使用宏定义将PID\_DEBUG设为1；

\* 当参数校正完成后，使用宏定义将PID\_DEBUG设为0，同时，在初始化时

\* 直接为p->a0、p->a1、p->a2赋值

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void pid\_calc(PID \*p)

{

//使用条件编译进行功能切换

#if (PID\_DEBUG)

float a0,a1,a2;

//计算中间变量a0、a1、a2

a0=p->Kp+p->Ki\*p->T+p->Kd/p->T;

a1=p->Kp+2\*p->Kd/p->T;

a2=p->Kd/p->T;

//计算PID控制器的输出

p->Out=p->Out\_1+a0\*p->Err-a1\*p->Err\_1+a2\*p->Err\_2;

#else

//计算PID控制器的输出

p->Out=p->Out\_1+p->a0\*p->Err-p->a1\*p->Err\_1+p->a2\*p->Err\_2;

#endif

//输出限幅

if(p->Out>p->OutMax)

p->Out=p->OutMax;

if(p->Out<p->OutMin)

p->Out=p->OutMin;

//为下步计算做准备

p->Out\_1=p->Out;

p->Err\_2=p->Err\_1;

p->Err\_1=p->Err;

}

//===================================================

//End of file.

//===================================================

[复制代码](javascript:void(0);)

**amain.c主函数文件**

[复制代码](javascript:void(0);)

//===================================================

//amain.c

//===================================================

//将用户定义的头文件包含进来

#include "PID.h"

//=============宏定义=====================

#define T0 　　0.0002 　　　　//离散化采样周期，单位s

//============全局变量========================

//定义PID控制器对应的结构体变量

PID ASR=PID\_DEFAULTS; 　　　　//速度PI调节器ASR

//定义PID控制器的参数及输出限幅值

float SpeedKp=2,SpeedKi=1,SpeedLimit=10; 　　//速度PI调节器ASR

//===============主程序=======================

void main()

{

//初始化PID控制器

ASR.Kp=SpeedKp;

ASR.Ki=SpeedKi;

ASR.T=T0;

ASR.OutMax=SpeedLimit;

ASR.OutMin=-SpeedLimit;

}

//============中断服务程序====================

interrupt void T1UFINT\_ISR(void)

{

//转速调节ASR

ASR.Ref=input1; 　　　　　　　　//速度给定

ASR.Fdb=input2; 　　　　　　　　//速度反馈

ASR.Err=ASR.Ref-ASR.Fdb; 　　 //偏差

ASR.calc(&ASR); 　　　　　　　　//函数调用：启动PID计算

output=ASR.Out; 　　　　　　　　//读取PID控制器的输出

}

//===================================================

//End of file.

//===================================================

[复制代码](javascript:void(0);)