目录

[版本历史 １](#_Toc12611)

[1. 程序文件详解 ３](#_Toc16836)

[2. 系统程序配置和解析 ３](#_Toc4247)

[2.1系统时钟配置（SYS） ４](#_Toc12109)

[2.2引脚配置（GPIO） ６](#_Toc19723)

[2.3定时器配置（TIM） ７](#_Toc29665)

[2.4通讯配置（I2C & USART） ８](#_Toc30237)

[2.5采样配置（ADC & DMA） ９](#_Toc26209)

[2.6看门狗配置（IWDG） １０](#_Toc24347)

[2.7中断配置（NVIC） １１](#_Toc19725)

[3. 用户程序 １３](#_Toc7366)

[3.1用户程序框架 １３](#_Toc15130)

[3.2按键扫描和LCD显示程序（AnalysisUart2） １３](#_Toc26902)

[3.2.1按键指令集 １４](#_Toc28652)

[3.2.2窗口函数运用（WINPROCPTR） １５](#_Toc26108)

[3.3任务轮询程序（TimerServer） １６](#_Toc27977)

[3.3.1喂狗及泄放判断程序（TimerProc\_Run） １８](#_Toc16213)

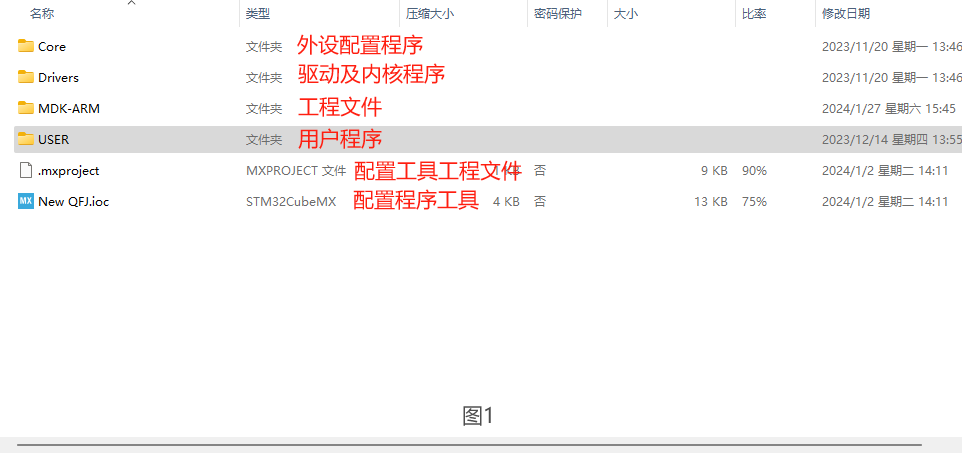
[3.3.2数据、图标的显示及判断程序（TimerProc\_Temp） １８](#_Toc25603)

[3.3.3采样处理程序（TimerProc\_Pres） １８](#_Toc28021)

[3.3.4注气PWM控制及PID算法（TimerProc\_PWM） ２０](#_Toc21211)

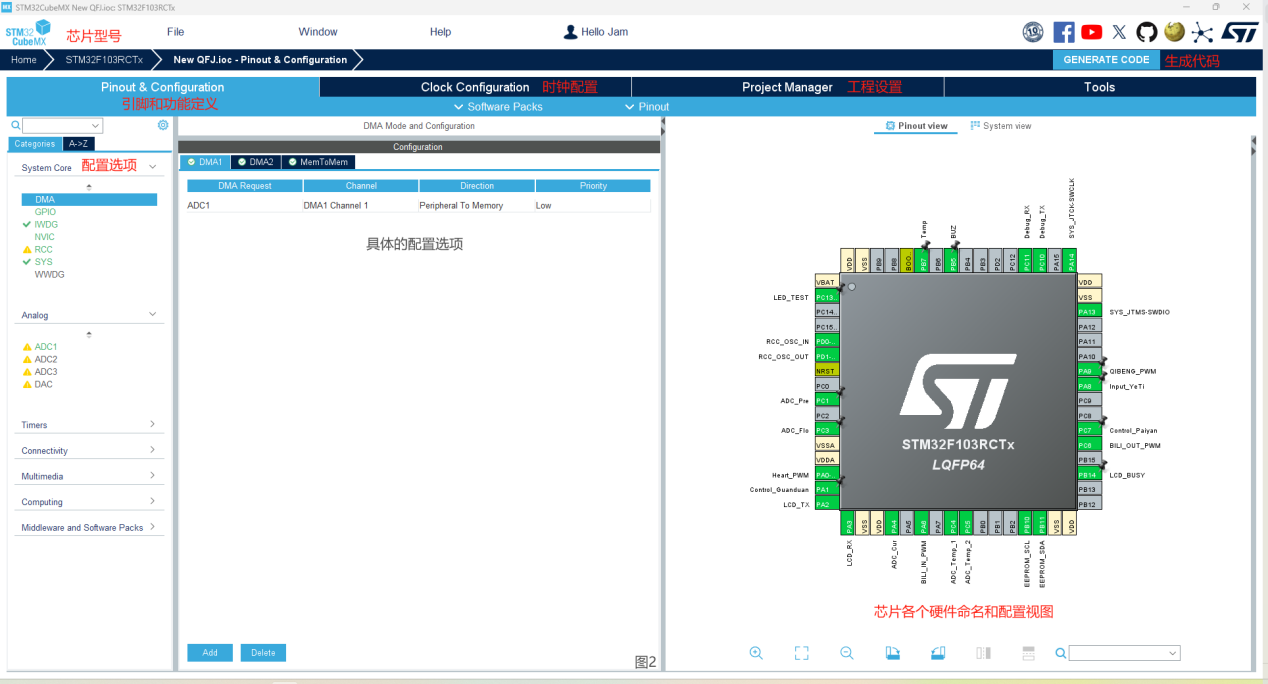
[3.3.5温度PWM控制及PID算法（TimerProcT\_PWM） ２１](#_Toc9447)

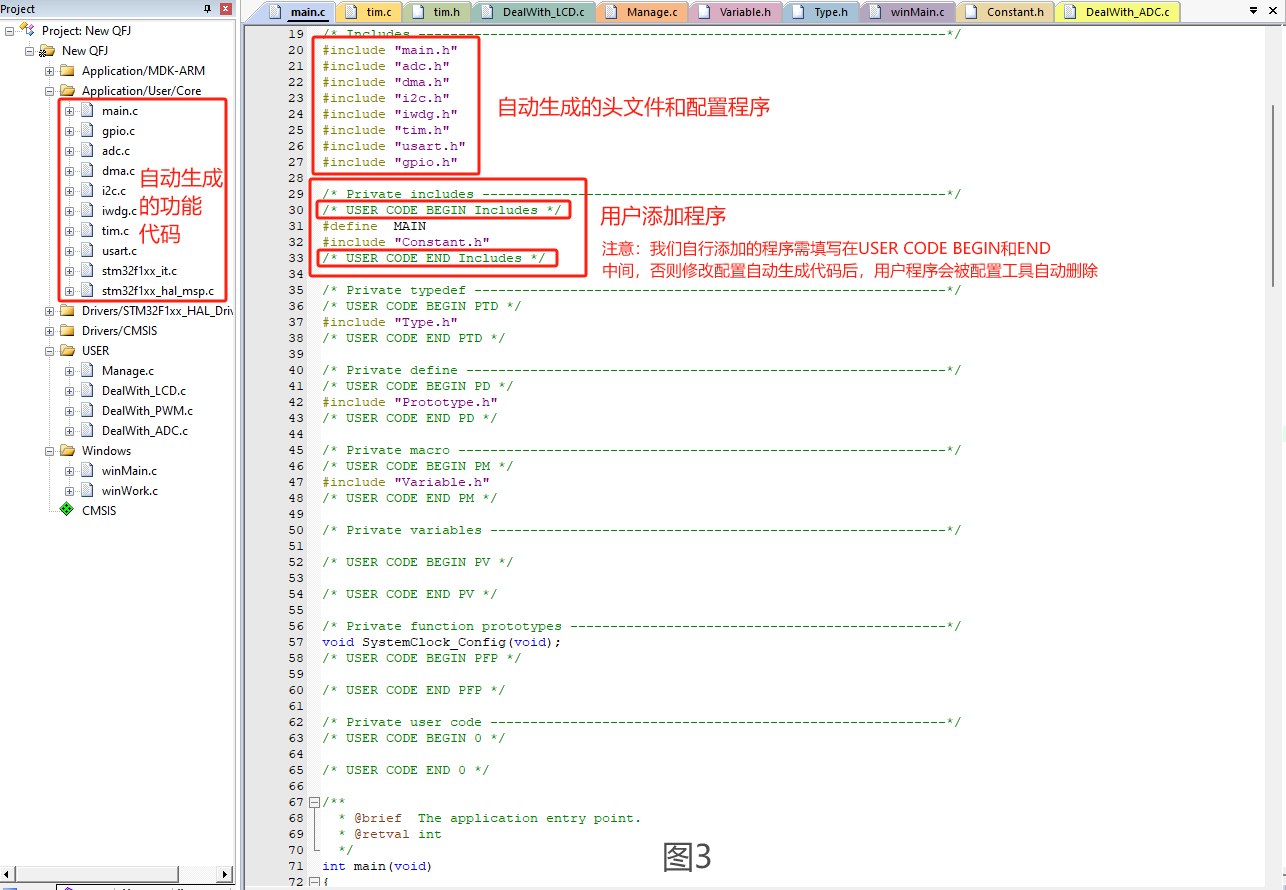
# 程序文件详解



如图1，软件文件夹中包含以上几类：外设配置程序文件夹（非必要勿要手动修改），驱动及内核程序文件夹（勿要手动修改），工程文件文件夹，用户程序文件夹，配置工具工程文件，配置程序工具。其中我们主要的核心程序和算法，都在用户程序文件中，平常的修改也尽可能在此文件夹中的.c和.h文件中进行。除了以上程序，配置程序工具为，STM32官方提供的32 位 ARM Cortex STM32 微控制器的图形工具**STM32CubeMX**，它允许对 STM32 微控制器和微处理器进行非常简单的配置，生成初始化 C 代码，还可以通过一步一步的操作为 Arm Cortex-M 内核或 Arm Cortex-A core 生成部分 Linux 设备树。目前是用的CubeMX为6.9.1版本，低版本可能会存在兼容性等问题。

# 系统程序配置和解析

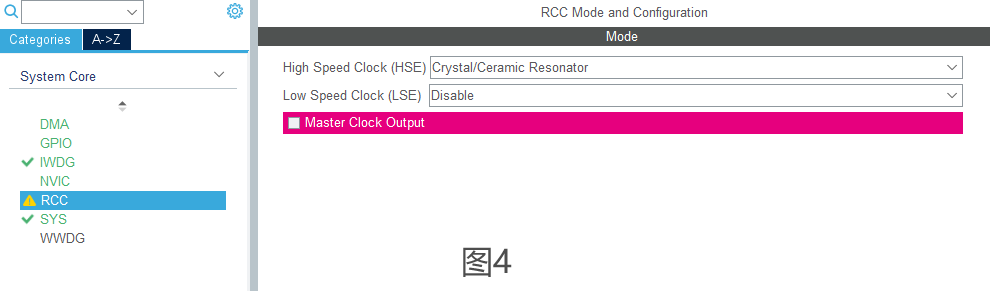


如图2 ，以上是STM32CubeMX的配置界面，包含引脚和功能配置、时钟配置和工程配置，可以通过左边和配置选项和右边的芯片配置图，具体配置软件的各个功能，详细的操作指南可见官网STM32CubeMX中文操作手册，配置完成后，点击右上角生成代码按钮，就可以生成如下代码（如图3）。由于是软件自动生成，所以我们的用户程序需写在特定位置，以防配置更新后，被工具自动删除。

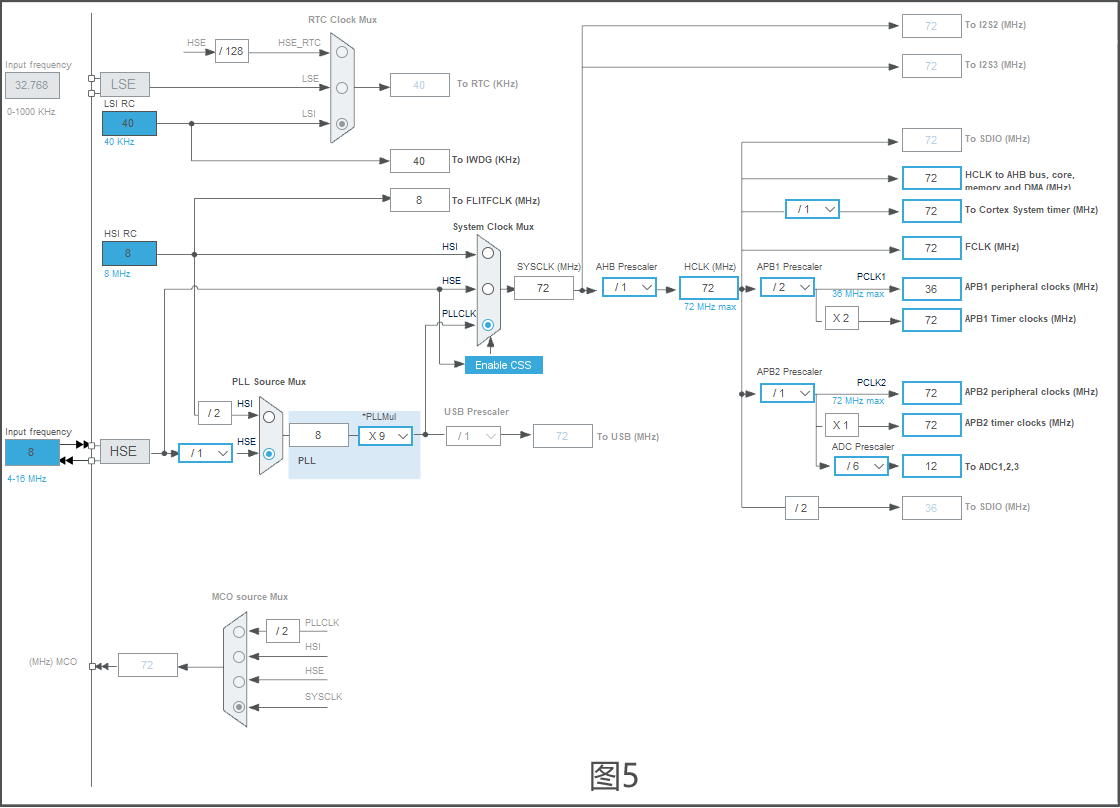
## 2.1系统时钟配置（SYS）

首先我们要知道，芯片需要一个频率来进行工作，通常选用的是晶振来提供工作频率，原理图上显示我们是使用的外部时钟源，这里我们使用的是HSE时钟源（High Speed Extern--高速外部时钟），输入频率为8MHz，所以图5的input frequency为8M。由图5可知，系统时钟的来源有三个，分别是PLLCLK、HSE、HSI。PLL是预分频器，我们勾选的是PLLCLK，因为可以让我们的输出频率达到最大，可以直接设置系统时钟为72MHz，其他外设时钟配置也如图5。

RCC配置：

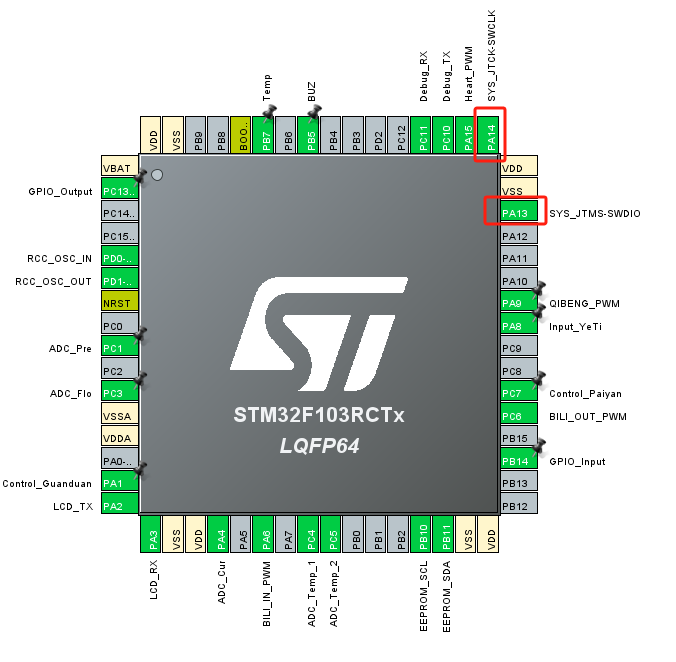
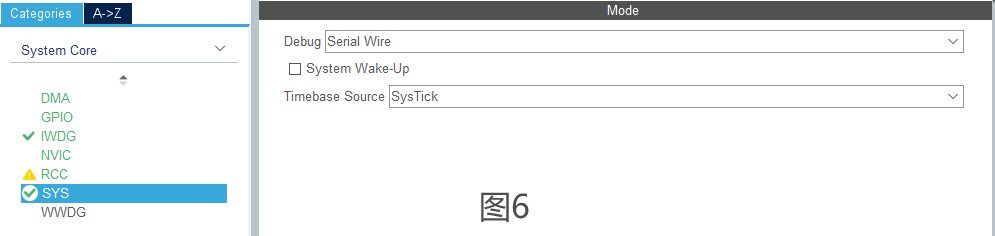


时钟树配置：



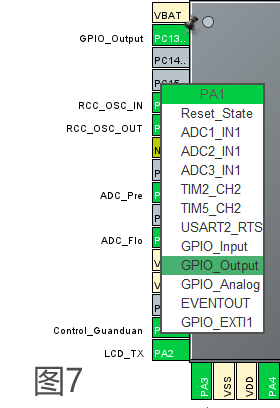
SYS配置：

主要是配置芯片的调试方式和时基，如图6，调试端口采用SW调试端口，采用同步串行协议：SWCLK引脚为从主机到从机的时钟信号；SWDIO为双向数据信号。Timebase Source： 一般指HAL的时基，即HAL库中的uwTick，用于实现HAL\_Delay()以及作为各种timeout时钟的基准。一般情况下直接选择默认的Sys Tick（嘀嗒定时器）来维护SYS Timebase Source即可，即直接放在SysTick\_Handler()中断服务函数中，也就是HAL\_IncTick()函数。



## 2.2引脚配置（GPIO）

点击CUBEMX左侧你想要配置的引脚，以配置PA1为例，即可出现GPIO的配置界面，在此只介绍通用的GPIO配置，即模拟的GPIO，再简单来说只配置GPIO为输入或者输出，对应GPIO\_Input和GPIO\_Output。若想要配置PA0为输出模式，则在上图界面中点击GPIO\_Output即可配置引脚为输出，同时在界面上也会有相应的显示(如图7);



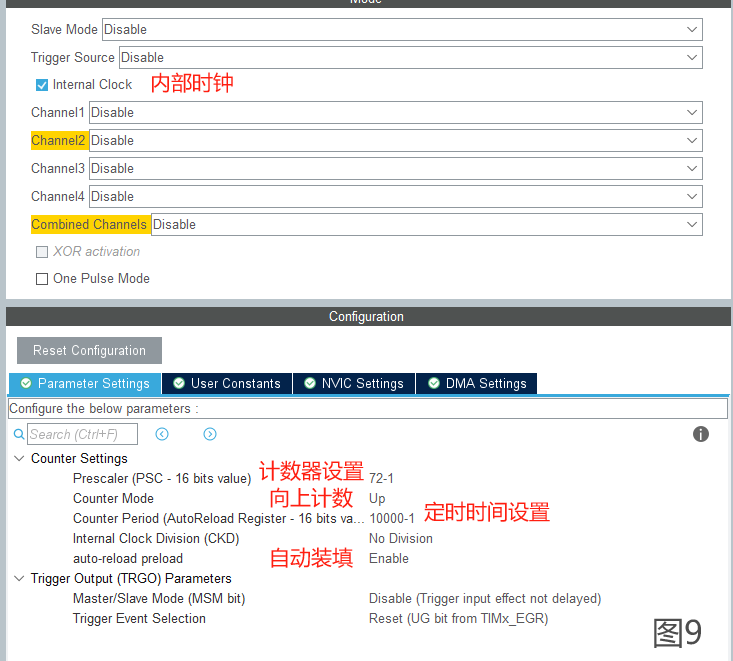
此时只是仅仅配置了GPIO口为输出模式，STM32CUBEMX还提供许多其他参数以供配置。点击System Core下的GPIO选项，从右侧窗口找到PA1这个IO口并点击一下。点击之后窗口界面会显示PA1的详细配置参数（输出），如图8所示；以此类推配置好相应的通用IO口。



## 2.3定时器配置（TIM）

根据原理图的引脚定义，我们程序中主要使用了基础定时器TIM4及中断，TIM1\_CH2定时器1通道2输出PWM，TIM2\_CH1定时器2通道1输出PWM，TIM3\_CH1定时器3通道1输出PWM和TIM8\_CH1定时器8通道1输出PWM；并根据需求设置默认的定时时间，PWM频率和PWM的占空比。

基础定时器TIM4的配置（如图9）：



根据之前配置的系统时钟可知，系统主频率为72MHz，主要需要确定的参数为定时时间和计数器设置，这两个变量具有如下关系式：

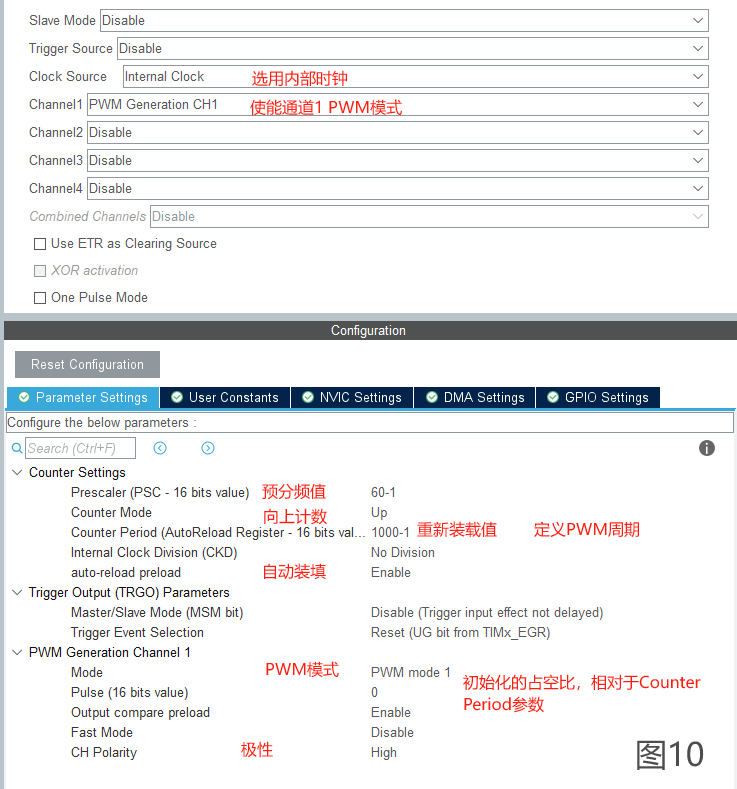
定时周期计算：

　定时器频率为 f = 72M / Prescaler / Period = 72000 000 / 72 /10000= 100Hz；

　定时时间T = 1 / f 则： 1s /100Hz = 0.01s = 2 = 10ms。

所以TIM4定时为10ms，由于实际使用中需要使用中断计数，所以需要使能中断，并使能计数器的自动装填，以达到10ms计数一次的目的。

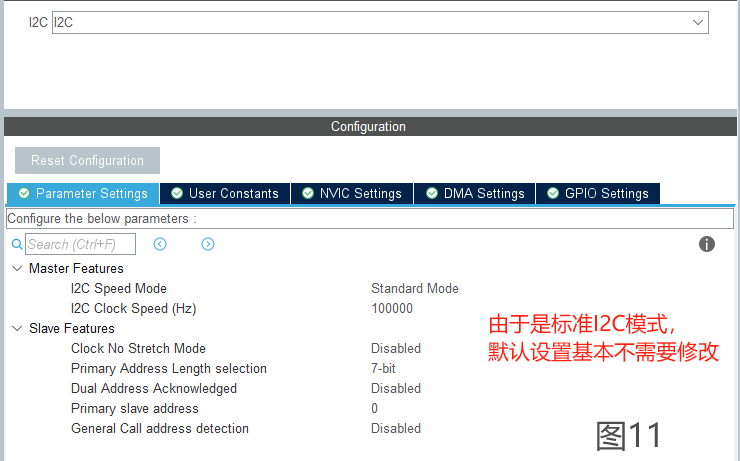
同理定时器PWM的配置，需要在此基础上，再增加PWM配置（如图10）：



由上述定时周期公式计算可知，PWM周期为1200Hz，初始化的占空比默认为0%，同理根据频率要求配置完其他PWM口。

## 2.4通讯配置（I2C & USART）

根据原理图的引脚定义，确认I2C和UART的具体映射口，以确保使能争取的I2C和UART，例如原理图上使用的PB10和PB11作为I2C通讯接口，参考规格书的引脚复用和映射表，其对应I2C2，所以如图11使能I2C2。



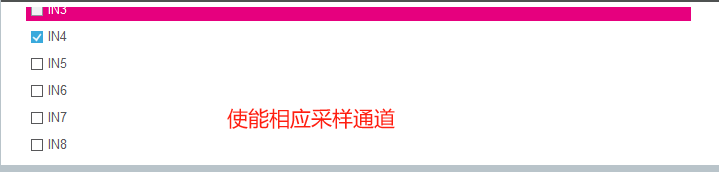
同样的UART也是类似配置，只不过需要确认波特率等参数和使能中断（如图12）。



其中模式分为一下几种：

## 2.5采样配置（ADC & DMA）

根据原理图和规格书，确认好采样引脚所对应的采样通道，然后使能相应通道，配置ADC采样参数，具体如下：





配置完相应的采样通道后，由于我们是多通道采样，所以数据量和处理时间也会很大，为了节省相应内核资源，采用DMA的方式作为采样和存储，配置如下，只需添加，并填写好相应的资源即可。



## 2.6看门狗配置（IWDG）

在由单片机构成的微型计算机系统中单片机的工作常常会受到来自外界电磁场的干扰，造成程序的跑飞，而陷入死循环；或者因为用户配置代码出现BUG，导致芯片无法正常工作，出于对单片机运行状态进行实时监测的考虑，便产生了一种专门用于监测单片机程序运行状态的模块或者芯片，俗称“看门狗”(watchdog)

简单说：看门狗的本质就是定时计数器，计数器使能之后一直在累加 而喂狗就是重新写入计数器的值，时计数器重新累加，

如果在一定时间内没有接收到喂狗信号（表示MCU已经挂了），便实现处理器的自动复位重启（发送复位信号）

STM32内置两个看门狗，提供了更高的安全性、时间的精确性和使用的灵活性。两个看门狗设备（独立看门狗、窗口看门狗）可以用来检测和解决由软件错误引起的故障。当计数器达到给定的超时值时，触发一个中断（仅适用窗口看门狗）或者产生系统复位。

独立看门狗（IWDG)由专用的低速时钟（LSI）驱动（40kHz），即使主时钟发生故障它仍有效。独立看门狗适合应用于需要看门狗作为一个在主程序之外 能够完全独立工作，并且对时间精度要求低的场合。

窗口看门狗由从APB1时钟（36MHz）分频后得到时钟驱动。通过可配置的时间窗口来检测应用程序非正常的过迟或过早操作。 窗口看门狗最适合那些要求看门狗在精确计时窗口起作用的程序。

所以独立看门狗更加适合我们的程序需求，而且由原理图可知，硬件电路上也预留了硬件看门狗电路，如果觉得软件看门狗不稳定或者不够用，可启用硬件看门狗，我们这为了节省成本就使用了芯片内部自带的独立看门狗了，具体配置见图14。



超出（溢出）时间计算：

 Tout=((4×2^PRER) ×RLR)/LSI时钟频率 = 32\*2500/40000 = 2s

所以当程序持续2s没有喂狗的话，程序将会自动重启，看门狗时间设置这么长也是考虑到I2C读写EEROM的delay时间。

## 2.7中断配置（NVIC）

顾名思义，所谓中断函数就是当它满足一定的条件时就会暂停主函数的执行内容，转而去执行中断函数，是软件中常用的事件处理方式，由于此项目中未有IO口用于外部中断，所以只需对定时器串口等外设，进行中断使能和优先级选择，具体如下：

这里的几处选项解释一下：

最上面的Priority group是选择优先级分组，STM32（更确切地说是contex-M3内核的单片机）使用4个bit位来分组，这里可以设置用几个bit来区分抢占优先级和响应优先级；



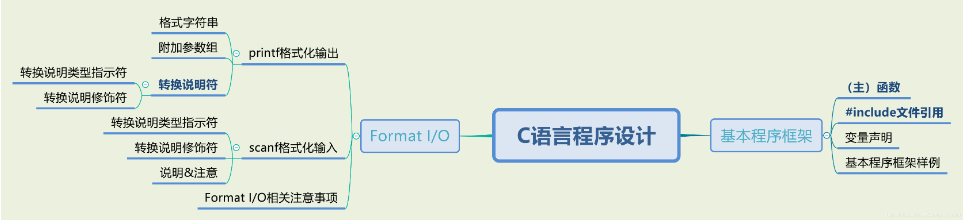
比如图中选择的是4bit用来区分抢占优先级、1bit用来区分响应优先级，则抢占优先级可以选择为0~15，响应优先级可以选择为0~1；数值越小则优先级越高；

抢占优先级高的可以中断低优先级的中断，中断嵌套的形式实现；

抢占优先级相同时，有多个中断同时发生，则先运行响应优先级高的中断。

# 用户程序

## 3.1用户程序框架

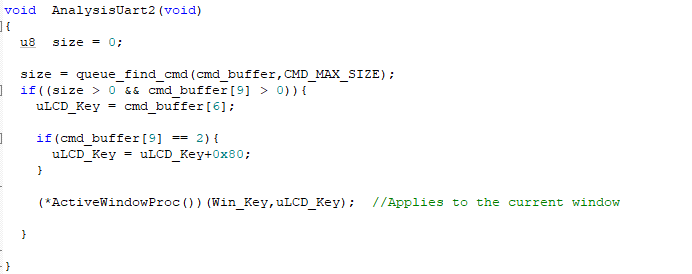


以上是程序设计时的基本框架，我们的主程序也基本套用此类框架，但是再次基础上有所改进，并引入任务轮询、函数指针数组和结构体定义等概念。

主程序由于是STM32CubeMX自动生成，在此之前在.c文件的开头，都#include好了相应的文件引用和系统初始化程序；主函数采用的是int main(void){}；再嵌套whil(1)的程序架构，因此我们可以在while函数中循环我们的LCD程序和任务程序，已检测和控制系统运行；而在whil之前，我们需要初始化相应的外设、读取EEROM、时钟和装置任务轮询，清空和初始化关键数据和数组。

## 3.2按键扫描和LCD显示程序（AnalysisUart2）

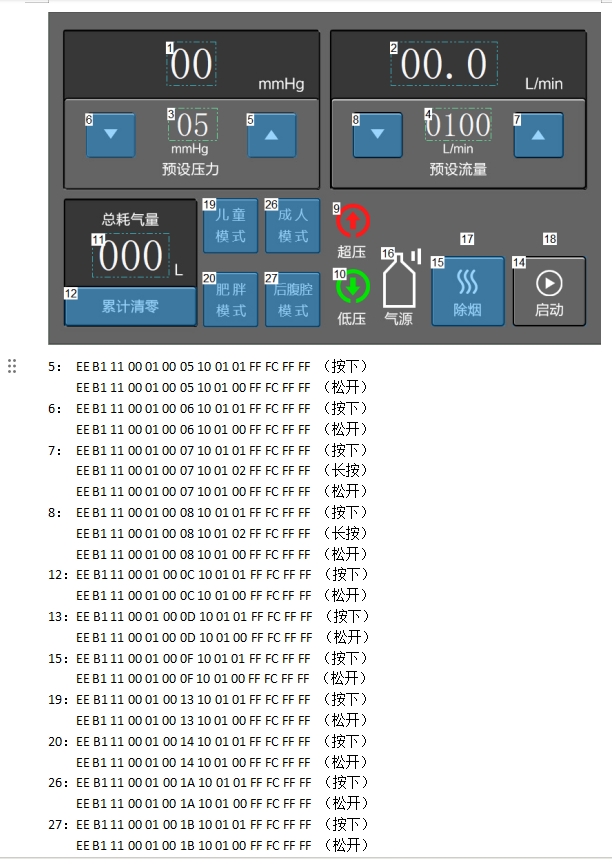
具体程序如下：



局部变量size为串口接受到指令集长度，如果不为0，则表示主机接收到了从机（显示屏）发来的指令，这类指令大部分以按键为主；

而数组中的cmd\_buffer[6]，cmd\_buffer[9]，则分别代表按键序号和按键状态，所以判断cmd\_buffer[9] > 0及有按键按下时，会指向函数指针，并运行相应功能，为1表示短按，为2则表示长按，所以cmd\_buffer[9] == 2时，对变量uLCD\_Key进行处理，方便之后DispatchKey程序中对变量uLCD\_Key的分类和对用功能程序执行。

### 3.2.1按键指令集



此指令集为最新指令集，有多种模式选择，如果屏幕为老程序，则没有26（成人模式）和27（后腹腔模式）按键，19和20也默认为之前的低压模式和标准模式。

### 3.2.2窗口函数运用（WINPROCPTR）

在函数理解上，最为复杂的即为AnalysisUart2函数中，最后一行的一函数指针(\*ActiveWindowProc())(Win\_Key,uLCD\_Key); ，其主要组成分为：

1. typedef void ( \* WINPROCPTR)(u8,u8); 结构体定义
2. WINPROCPTR ActiveWindowProc(void) 函数指向

{

return (OpenWndCollection[hActiveWindow]->WindowProc);

}

1. WINDOW \* OpenWndCollection[15];

第一点很好理解，typedef用法定义，总结一下就是: 任何声明变量的语句前面加上typedef之后，原来是变量的都变成一种类型，不管这个声明中的标识符号出现在中间还是最后。所以ActiveWindowProc也被定义成了一个指针函数，根据第二点中的return函数，这个(\*ActiveWindowProc())(Win\_Key,uLCD\_Key);最终的指向为WindowProc(u8 Type,u8 nRecive)这个函数。

但是这其实只理解对了部分，最终return的是存放在OpenWndCollection数组里的WindowProc函数，可以试下全局搜索，winMain.c中有WindowProc函数winWork.c中也有WindowProc函数，其.c中使用static静态声明，使其只在此文件中运行，不会导致其重复编译或者函数重名，所以运行的是哪个WindowProc函数，我们就要详细剖析OpenWndCollection数组和一系列的window函数，即为本章重点的窗口函数。

先看结构体WINDOW的定义：

typedef struct tagWINDOW {

WINPROCPTR WindowProc;

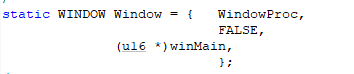
u8 Active;

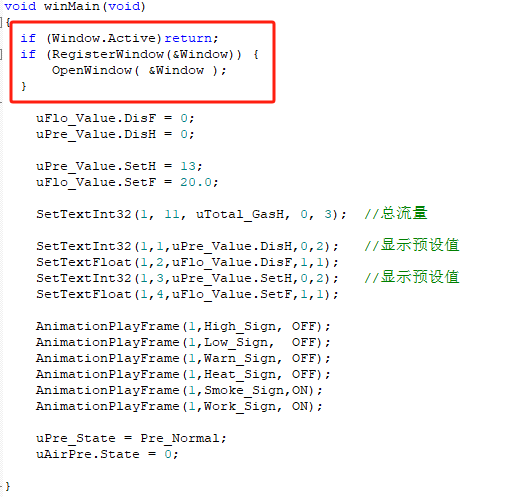
u16\* Address;

} WINDOW;

所以第三点中的WINDOW \* OpenWndCollection[15];其实就是定义了一个结构体指针数组，并且数组里面的元素都是结构体的地址，简单来说OpenWndCollection数组最终可以指向15个WindowProc函数，Active变量和Address地址。这样剖析完后，这个结构体指针数组的功能其实就很明了了，大概率是用于界面显示和转换用的，可对应不同界面不同按键，做出相应的功能。

那具体数组变量hActiveWindow如何增减，怎么确定窗口是否激活，我们就要转换到窗口变化程序winMain和winWork中；这两个.c中基本都采用的静态static定义结构体和函数。我们可以以winMain的定义为例，静态结构体声明里的WindowProc即为指向函数，FALSE为变量Active的值，Address内为函数winMain的地址。



那是为什么Active的默认值为啥要给FALSE，这就跟以下程序有关了：

还是以winMain函数为例，起始的判断，就是为了判断窗口的Active变量是否为TURE，即窗口是否激活，如果未激活，则return不运行下面的显示程序。那什么时候激活变量hActiveWindow又什么时候加减呢，这里就包含了两个非常关键的函数RegisterWindow和OpenWindow。

顾名思义，RegisterWindow函数用于注册串口，OpenWindow函数用于激活窗口，所以简单来说，运行RegisterWindow函数，表明hActiveWindow++,结构体Window的值就会赋值给OpenWndCollection[hActiveWindow]，运行winMain程序后数组从0加加为1，创建OpenWndCollection[1]的结构体指针数组，指向winMain.c内对应的WindowProc；运行OpenWindow函数，表明激活Active，直接赋值为TURE。同理有创建就比然有关闭，道理类似，这里就不做详解。

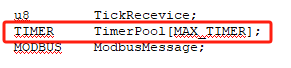
所以举一反三的看待这个结构体指针数组程序，winMain其实实现了开机气腹未工作时的功能合集和屏幕显示，而winWork实现了正常气腹工作时的功能合集和屏幕显示。

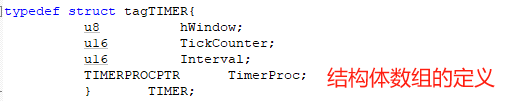
## 3.3任务轮询程序（TimerServer）

1.2介绍的AnalysisUart2函数，主要是用于LCD显示和按键扫描，所以本章就详细介绍下功能程序的架构和实现。

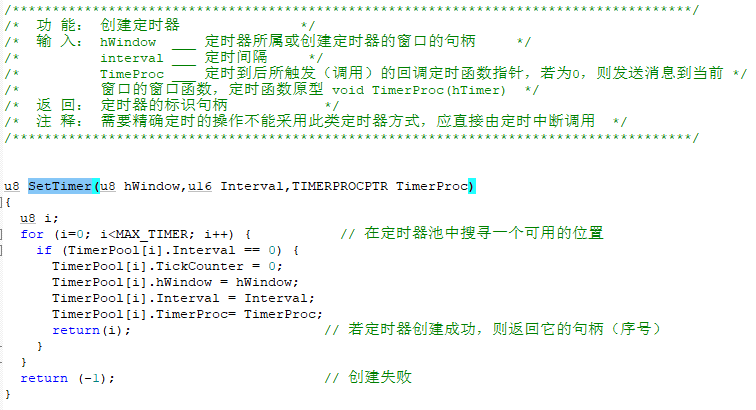
在具体了解各个功能程序之前，我们的先了解下TimerServer这一函数的工作原理，我们也同1.2一样，看组成，然后逐步剖析。







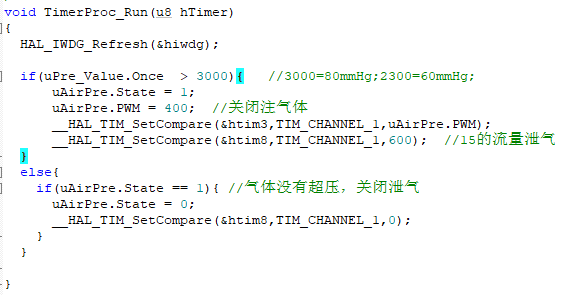
有了1.2的基础，我们理解起TimerServer函数应该更加简单，TIMER结构体定义了TimerPool[MAX\_TIMER]数组，每个数组里的元素都包含有TIMER结构体里的变量和指针函数。之后就是根据程序确定各个变量的作用，那么这其中不得不提到TimerServer函数运行前的装载函数SetTimer，具体如下：



如程序标注的一样，TimeProc为触发后的回调函数，interval为定时间隔，及程序的触发时间，hWindow我们暂未用的，不作赘述，最后的TickCounter即是我们定时的计数值。总结来说，TimerServer函数就起到了个任务轮询的作用，相应定时时间到后，触发相应任务程序函数，实现相应功能。

### 3.3.1喂狗及泄放判断程序（TimerProc\_Run）

根据TimerServer的函数介绍，我们再看TimerProc\_Run的功能，首先根据uTime\_Run = SetTimer(0, 50, TimerProc\_Run); 的定义，我们可以确认，TimerProc\_Run 50次计数周期执行一次，也就相当于500ms执行一次，根据我们之前IWDG的设置，这个喂狗时间是符合要求的。再看TimerProc\_Run函数的源程序，除了喂狗功能外，还有判断系统是否超压、泄气的功能程序。



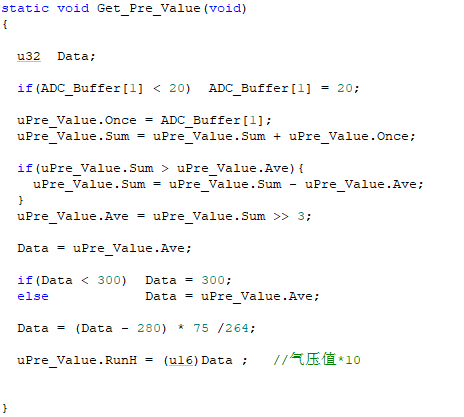
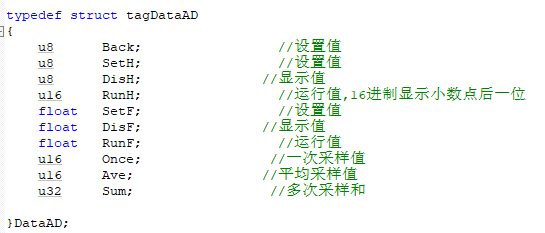
### 3.3.2数据、图标的显示及判断程序（TimerProc\_Temp）

同样的根据uTime\_Temp = SetTimer(0, 200,TimerProc\_Temp); 的定义，可以确认TimerProc\_Temp函数2s执行一次。由于逻辑和判断都较为简单，这里就不具体展开详解，具体可参考程序；主要其功能，用于压力、流量和总流量的数据显示，超压、低压的图标显示及判断。

### 3.3.3采样处理程序（TimerProc\_Pres）

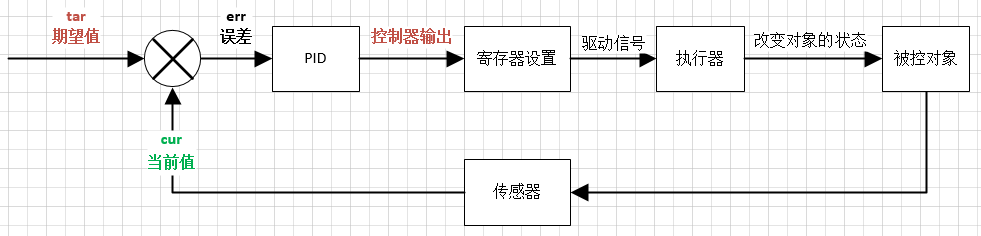
同样的根据uTime\_Pres= SetTimer(0, 10, TimerProc\_Pres); 的定义，可知TimerProc\_Pres函数100ms执行一次。其包含了系统所有的数据采样和处理函数，包括压力采样，流量采样，气源压力采样和气腹管两NTC的采样温度。具体的DMA采样电压转换程序，因为较为基础，我们就直接跳过了，我们这直接看数据处理程序，以压力传感器为例。

可以看到，程序中数组ADC\_Buffer即为DMA转换后的电压采样数据，可根据之前ADC的配置排序，对应其引脚和功能，像ADC\_Buffer[1]即为压力传感器采样数据。采样数据通过累加求平均的方式，得到最后的Data，最终通过规格书上的电压转换公式，带入方程，得到最终的气压值uPre\_Value.RunH。

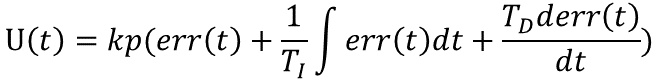


流量、气源和温度的采样转换也是同样的道理，其中需要注意的是，温度和流量的采样，程序中用了分段函数，根据数据局部标定，数据来源更加准确；压力数据由于特殊数据处理的原因，采样过程中放大了10，后续显示中会缩小10倍后显示。还有特别要注意的是，程序中这种ADC求和做平均的方式，不同于以往认知中的加权平均，这类的求平均，需要一定的数据量，而且有一定的滤波和抗扰效果，具体可用excel表格模拟下；而且其中移位的多少，也决定了数据需要稳定的时间。

### 3.3.4注气PWM控制及PID算法（TimerProc\_PWM）



PID控制应该算是应用非常广泛的控制算法了，PID(proportion integration differentiation)其实就是指比例，积分，微分控制，公式如下图，总的来说，当得到系统的输出后，将输出经过比例，积分，微分3种运算方式，叠加到输入中，从而控制系统的行为。



将比例、积分、微分三种调节规律结合在一起， 只要三项作用的强度配合适当，既能快速调节，又能消除余差，可得到满意的控制效果。

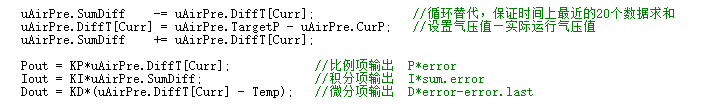
特点：

* PID 控制作用中，比例作用是基础控制；微分作用是 用于加快系统控制速度；积分作用是用于消除静差。
* 只要比例、积分、微分三种控制规律强度配合适当， 既能快速调节，又能消除余差，可得到满意控制效果。
* Kp 较小时，系统对微分和积分环节的引入较为敏感，积分会引起超调，微分可能会引起振荡，而振荡剧烈的时候超调也会增加。
* Kp 增大时，积分环节由于滞后产生的超调逐渐减小，此时如果想要继续减少超调可以适当引入微分环节。继续增大 Kp 系统可能会不太稳定，因此在增加 Kp 的同时引入 Kd 减小超调，可以保证在 Kp 不是很大的情况下也能取得较好的稳态特性和动态性能。
* Kp 较小时，积分环节不宜过大，Kp 较大时积分环节也不宜过小（否则调节时间会非常地长），当使用分段PID ，在恰当的条件下分离积分，可以取得更好的控制效果。原因在于在稳态误差即将满足要求时，消除了系统的滞后。因此系统超调会明显减少。

在理解完PID算法后，我在看程序，首先在进入PID之前，程序设置了注气时间和停止注气时间，在初始的工作下，注气9.5s 停气0.5s，一个循环完成后，改为注气4.5s 停气0.5s。

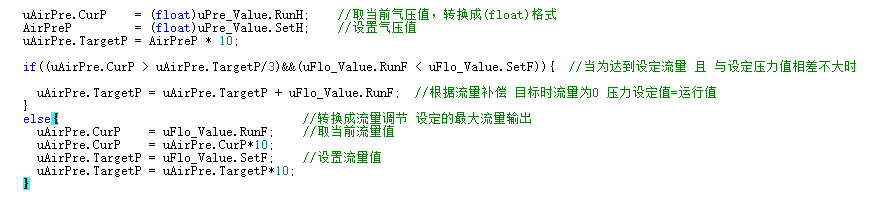
根据之前的公式，PID的输出值可以分为比例输出：

* Pout =比例系数\*误差值= KP\*Error
* Iout=积分系数\*误差值之和=KI\*Sum.Error
* Dout=微分系数\*误差的导数=KD\*(Error-Error.last)

所以按照此逻辑，程序如下：

然后设置好比例输出的上下限，防止数据振荡，同时也限定好PID的调节上下限和参数的调节范围，防止死区，至此PID算法的核心，我们已经了解。

但是这里还有个重点，就是设定的选择，源程序如下图。当压力很低，例如刚启动的时候，我们需要将流量拉满，快速注气，所以这个时候就将PID的设定值转换成了流量值，以达到流量快速上升的目的；相同如果流量大于设定值，我们也需要将PID设定值转化成流量，以达到控制流量的目的。如果由于外部原因，当流量采样值小于设定值，这个时候，我们需要判断压力是否达到设定值，所以将压力和流量相加作为PID的设定值，这样做的目的，就是当流量为0时，压力设定值=运行值，并且由于流量参数的加入，提供了一定的超调量，以抵消静压压力小于动态压力的问题。



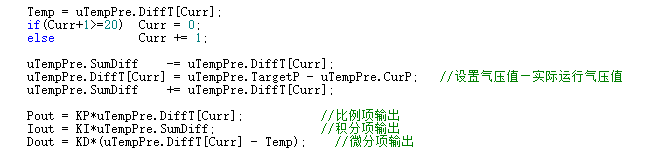
### 3.3.5温度PWM控制及PID算法（TimerProcT\_PWM）

有了上面注气PWM的理解，相比之下，温度PID就更加简单。首先在进去PID之前，还是先看下PID控制程序的运行状态，具体的运行地址在winWork的RefreshState函数内，相当于当气腹机开始工作时，加热就已经开始进行了。

同上的PID理论，PID的输出值可以分为比例输出：

* Pout =比例系数\*误差值= KP\*Error
* Iout=积分系数\*误差值之和=KI\*Sum.Error
* Dout=微分系数\*误差的导数=KD\*(Error-Error.last)

所以按照此逻辑，程序如下：



需要注意的是，由于我们气腹管有两个NTC，所以我们的PID设定值也可以根据需求切换。