STM32F072-M0内核操作手册

**作者:向仔州**

安装STM32CUBEMX软件…………………………………………………………………………………..2

用STM32CubeMX生成代码……………………………………………………………………………….7

设置keil的下载功能………………………………………………………………………………………….13

GPIO管脚输出操作…………………………………………………………………………………………….16

GPIO管脚输入操作…………………………………………………………………………………………….18

STM32F0中断操作……………………………………………………………………………………………..20

Systick内核定时器做延时函数…………………………………………………………………………..26

USART串口通讯………………………………………………………………………………………………….30

串口映射成printf实现……………………………………………………………………………………….35

CAN总线实现……………………………………………………………………………………………………..36

CAN波特率档位统计，1M，500K，125K………………………………………………………...46

TIM1定时器使用………………………………………………………………………………………………..47

用定时器2做PWM输出…………………………………………………………………………………..50

ADC单通道轮询问电压采集……………………………………………………………………………...53

ADC单通道DMA循环采集………………………………………………………………………………..58

ADC多通道轮询采集………………………………………………………………………………………….60

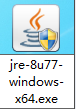
ADC多通道DMA采集………………………………………………………………………………………..61

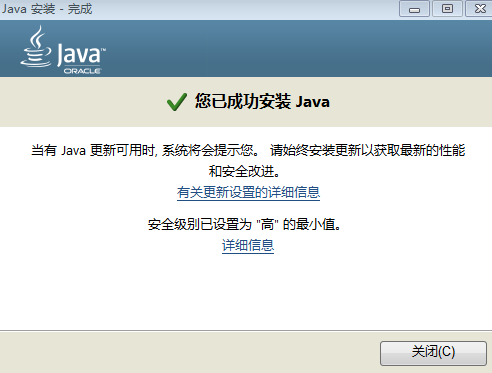
**安装STM32CUBEMX软件**

这个软件就是用图形来生成代码，很方便

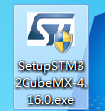
安装cubemx之前要安装JAVA运行环境

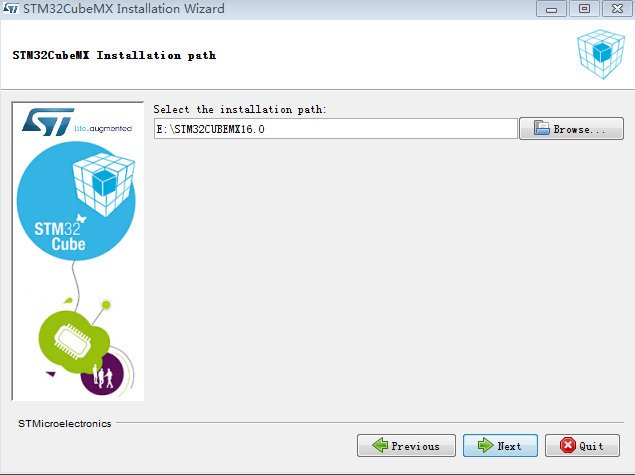
去orcal官网下载JAVA的jre包运行环境

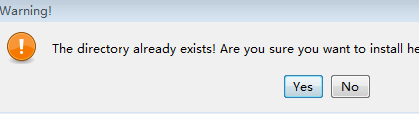
我电脑上64位的，所以就安装这个，直接打开软件点击安装就是了

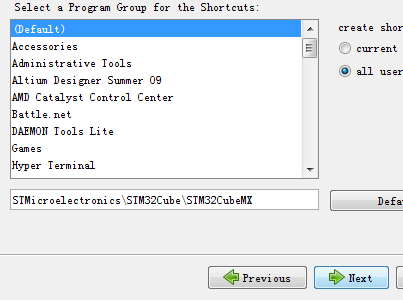
JAVA运行环境安装完成

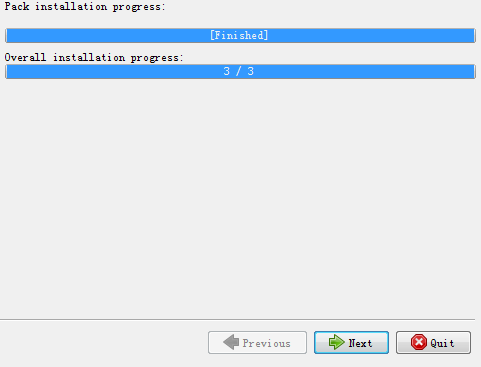
安装cubemx软件

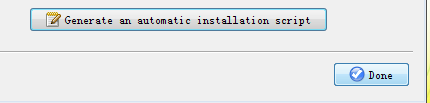
我这里的版本是16.0，点击我允许协议，然后点击下一步

选择软件安装的路径

这里选择yes

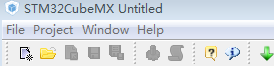
什么都不要管直接点击next

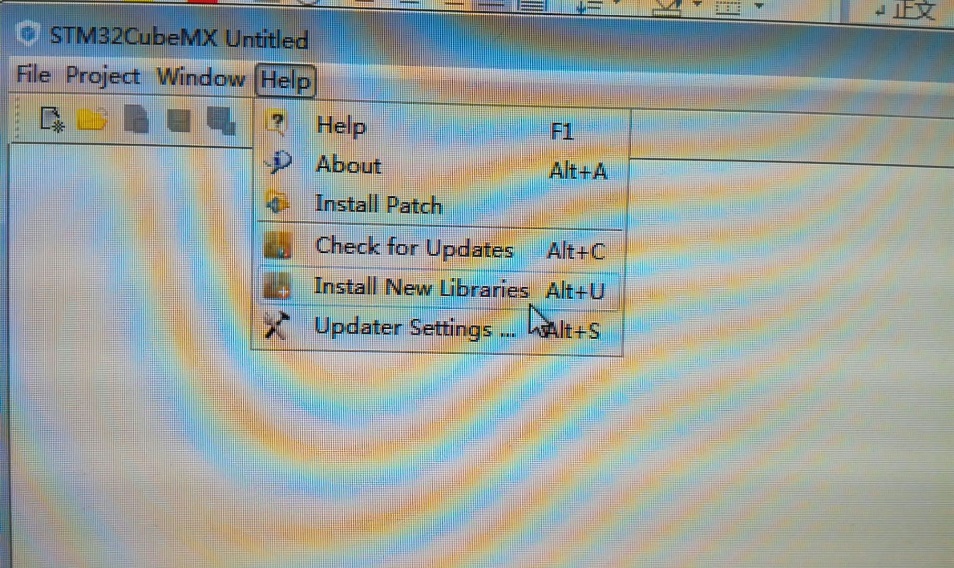
这就是安装完成，点击下一步

点击done

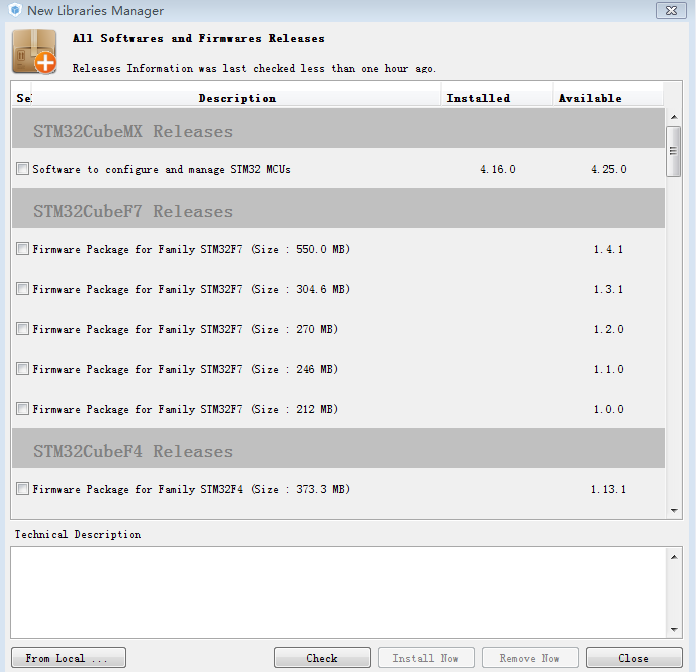
这个图标就是STM32CubeMX软件

打开CubeMX软件

点击help



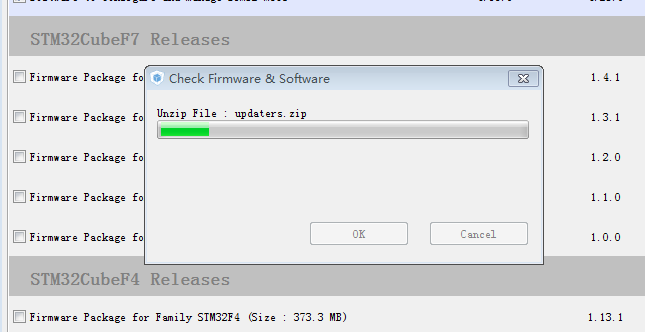
如果你软件不是最新版本可以在这里选择更新



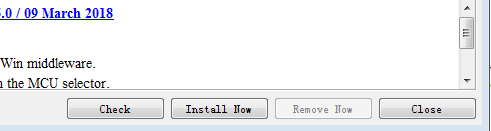
然后点击check检查是否能更新，记住电脑一定要上网

如果要更新到4.25我们勾选这个

这里提示官方有4.25的版本了，但是我们现在安装的是4.16的版本

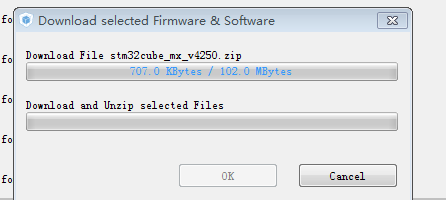


显示绿色就是表示可以更新

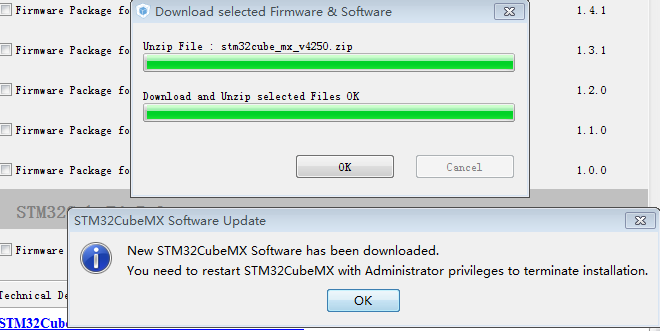


点击Install Now更新

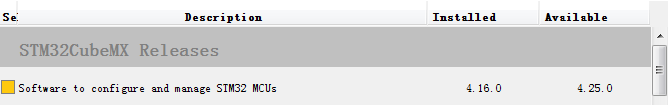
既然可以更新我们就再次选择这个



现在开始下载更新了

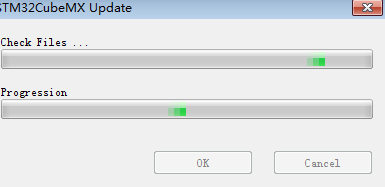


显示OK就是更新完成，点击OK



再次查看版本，发现没有更新

这个时候我们要关闭软件，其实更新脚本已经下载进来了只是我们要重启软件才能完全更新

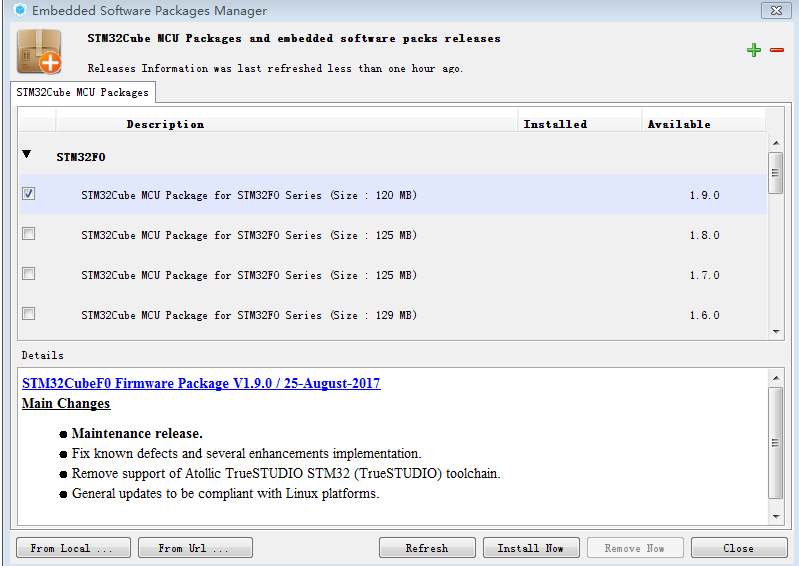
 

打开软件后发现在自动更新

重新打开软件，一定要以管理员身份运行软件

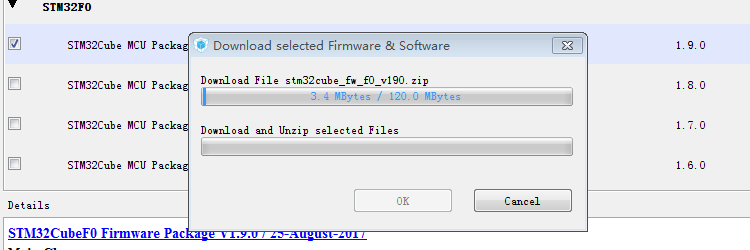
然后安装插件，我们要选择和我们使用的芯片型号配对的插件版本我这里使用的是STM32F0

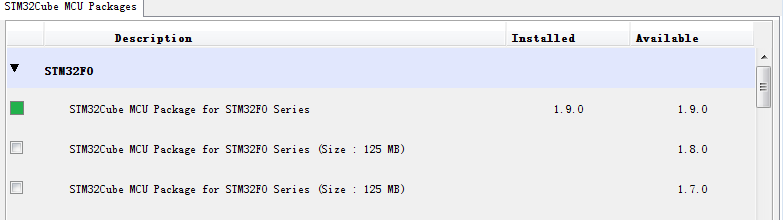
所以我们最新的f0版本插件



点击安装

选择最新版本的f0插件



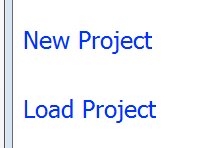


插件安装完成

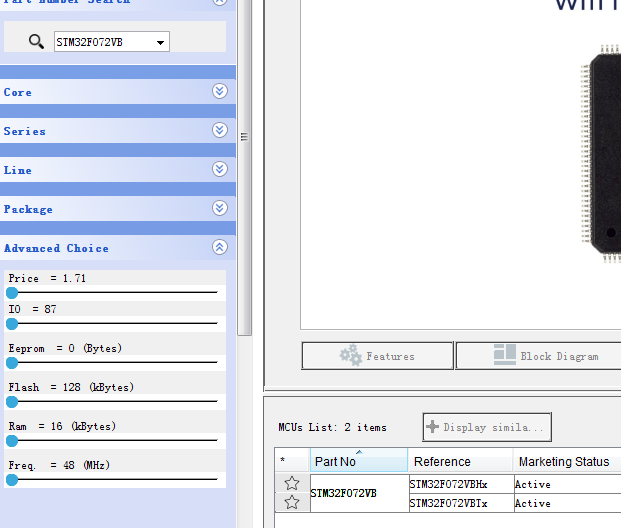
然后关闭整个软件，一定要关闭整个软件，然后再次打开软件就可以正常使用了。

**用STM32CubeMX生成代码**

打开CubeMX



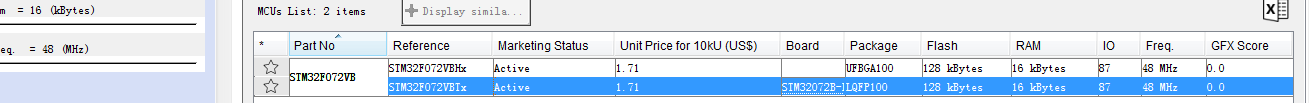
点击New Project



这里是你需要芯片的CPU速度，flash大小，ram大小，你可以拖动圆圈选择

我用的是100脚的CPU我双击这个

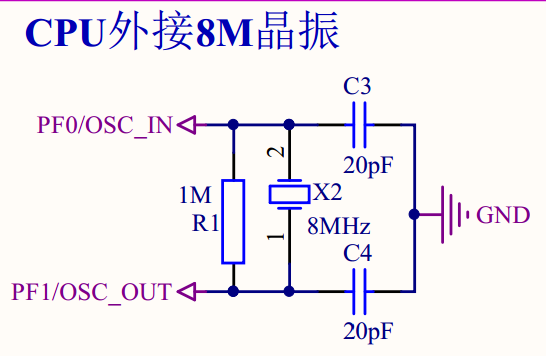
输入你要的型号



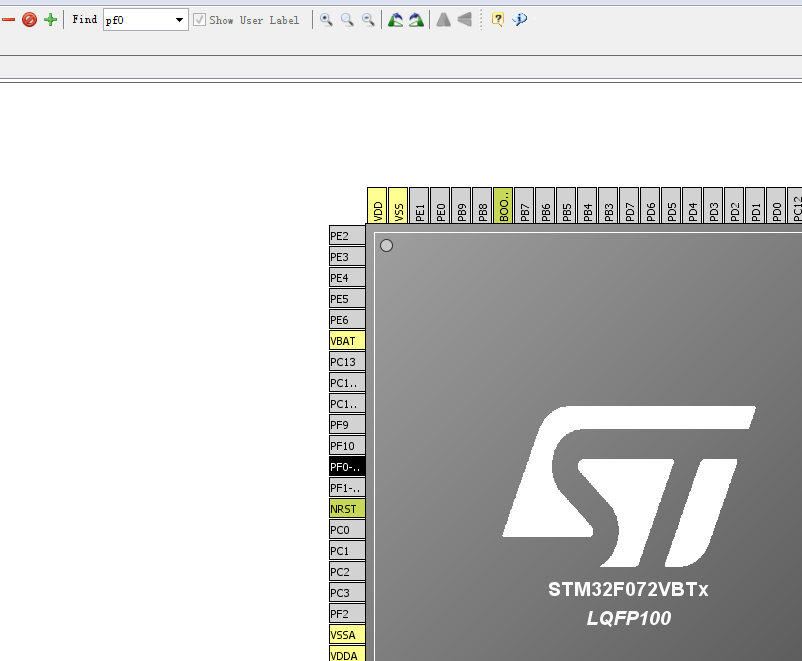


这个黄色的是电源和地(属于特殊引脚)，无法用鼠标设置

控制GPIO点亮LED

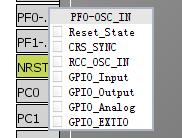


我们MCU运行需要时钟，所以要设置PF0,PF1管脚

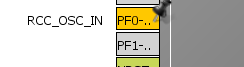


找到你要的管脚它会闪烁

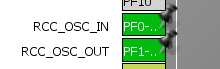
如果你觉得眼睛找起困难，可以在这里输入管脚编号

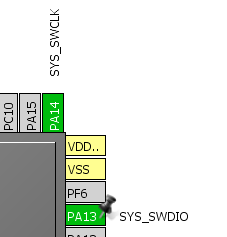


我们双击PF0引脚，会出现一个勾选列表，你需要管脚设置为什么功能，你勾选就是了，我这里设置管脚为时钟接口



这就是设置成功后的样子

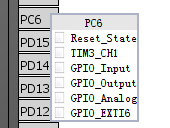
我们把PF0和PF1都设置上



SWD仿真接口也要设置

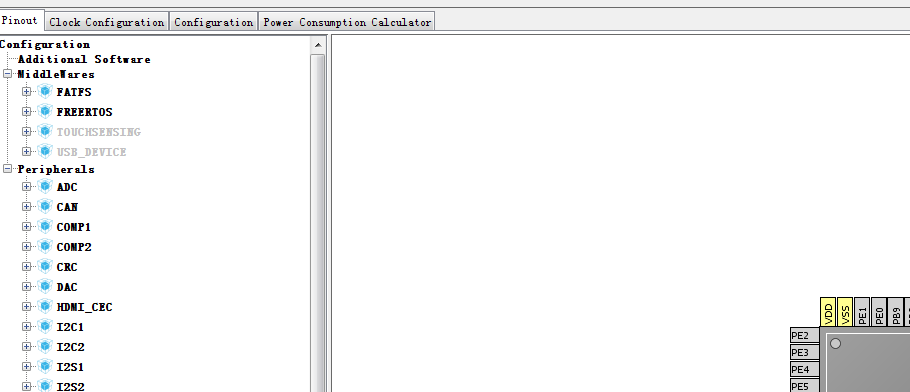
因为我们是控制LED，所以我们还有选择PC6管脚



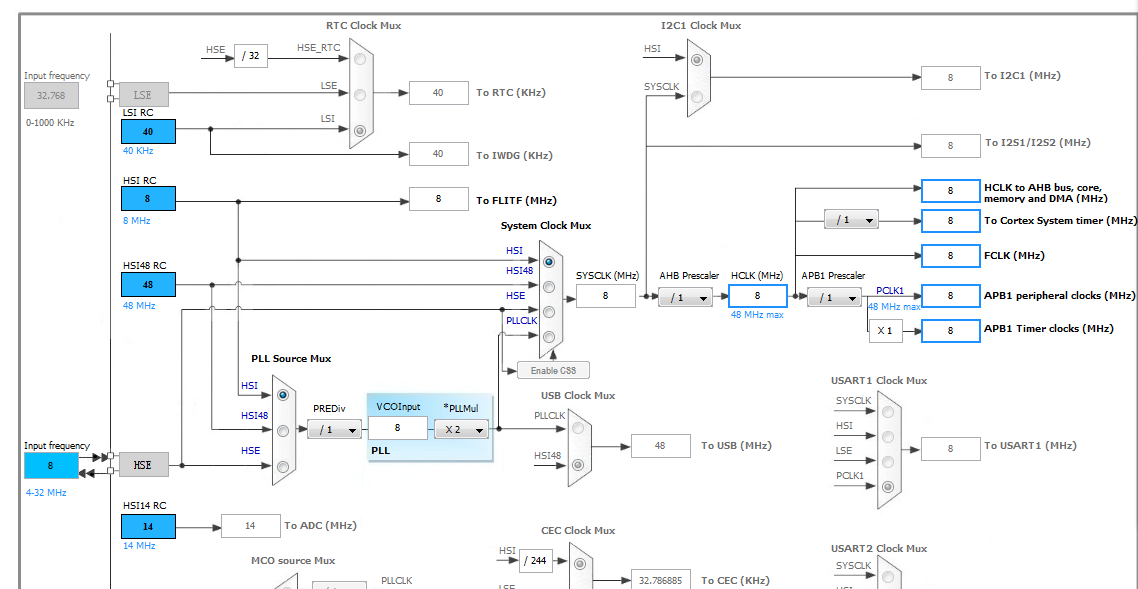


PC6设置为输出

现在MCU要使用的引脚我们已经配置完成，下一步要设置时钟树



选择时钟配置

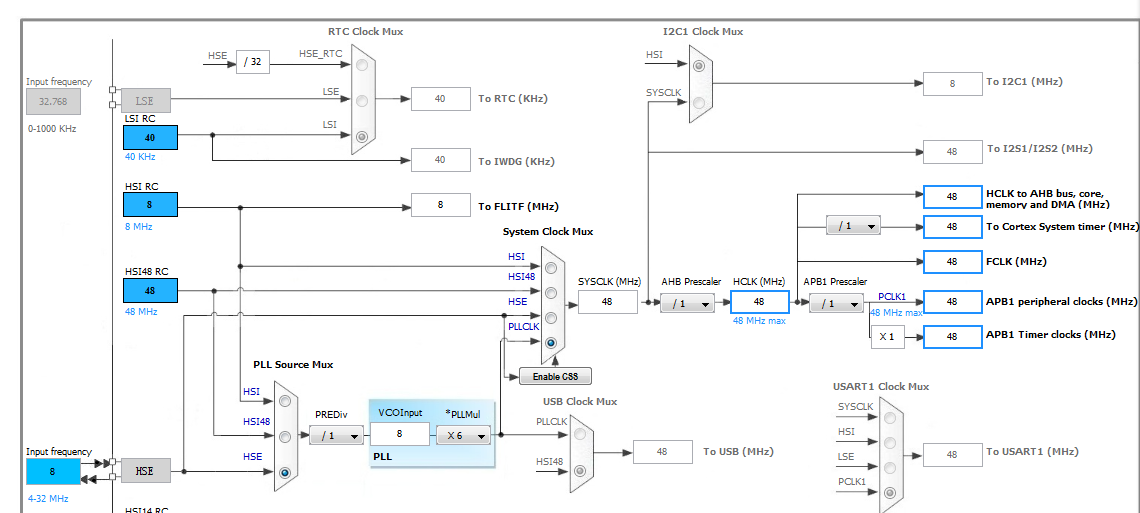


然后选择PLLCLK开关把48M给CPU时钟

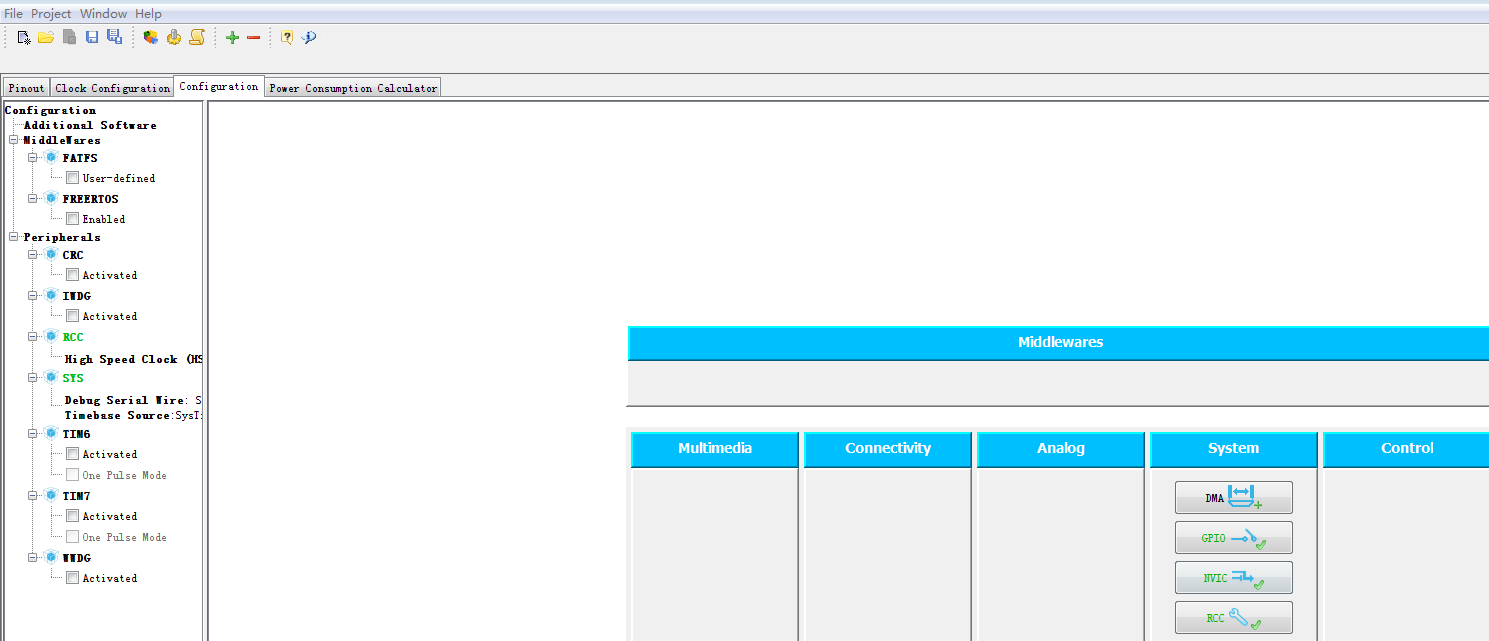
得到不分频的8M晶振我们要X6得到48M

原始进来的晶振不分频

我们是用的外部8M晶振，所以选择这个

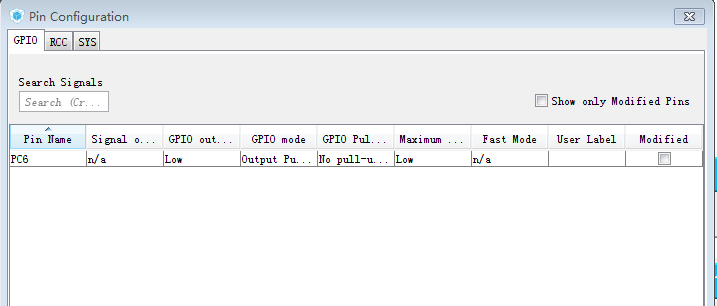


这就是设置后的结果。时钟配置完成

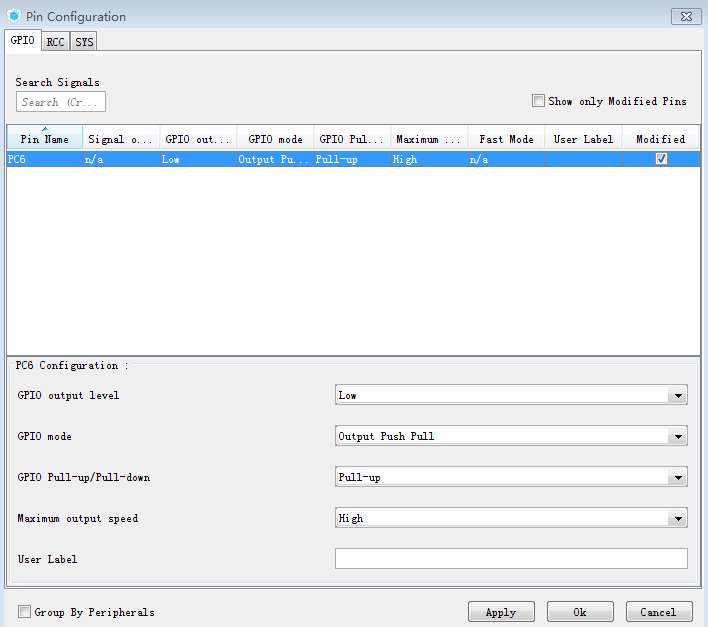


选择GPIO设置

我们还有选择configuration选项，来设置GPIO上电初始化状态



因为我们只使用了一个管脚去控制LED，所以这里只有一个



最后点击OK

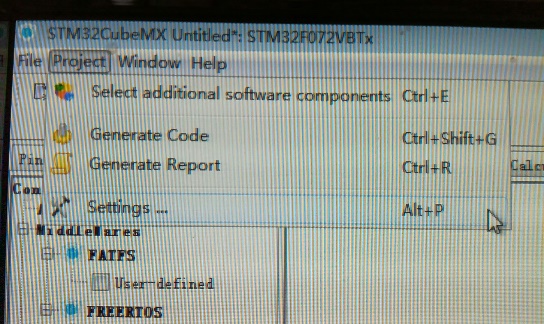
给你的管脚取个名字，然后在Keil就能看到有这个名字的代码

GPIO为高速输出模式

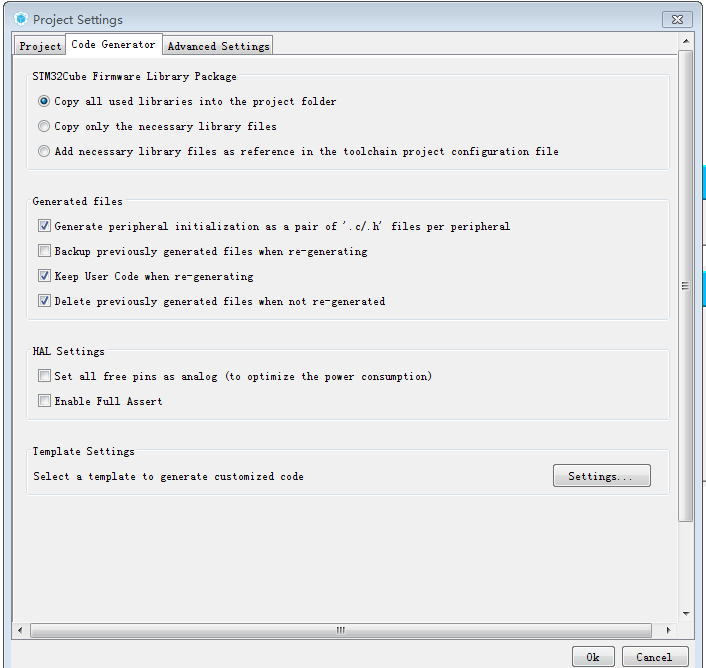
输出接上拉电阻

输出模式为推挽输出

我们选择管脚上电为底电平

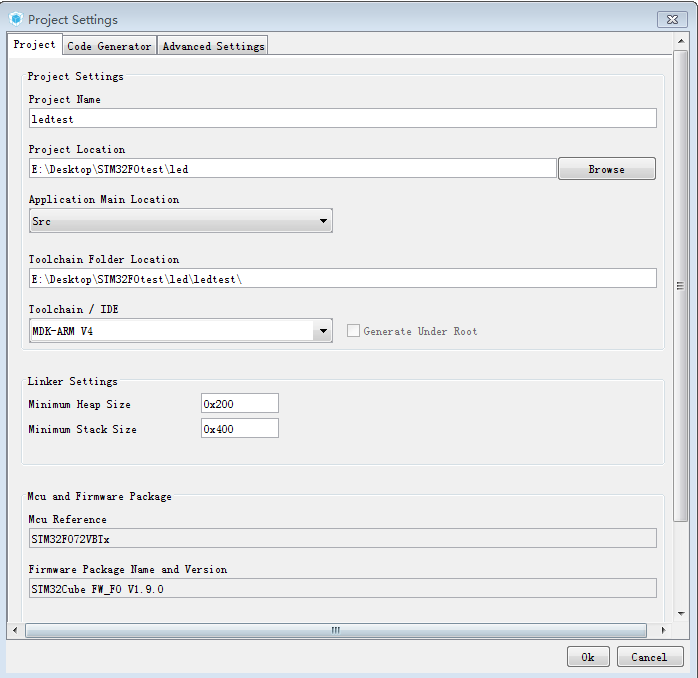


然后还要设置工程



勾选上生成C和H文件

先选择第2项



点击工具栏按钮，生成keil工程

然后点击OK

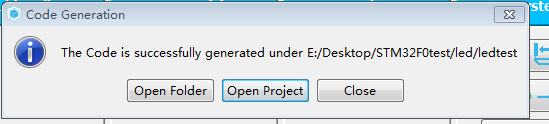
因为我电脑安装的是keil4 IDE，所以选择keil4

选择生成的keil文件保存在什么目录下

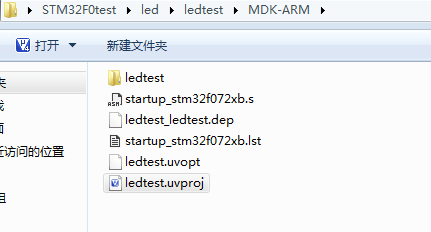
取个生成keil文件的工程名

然后选择第1项





工程生成完成，然后你点击close，再关闭CubeMX软件，用keil打开工程



工程文件就在你CubeMX指定生成keil工程的目录下

编译通过，证明工程建立没有问题，后面按照常规的操作keil方法操作就是了。

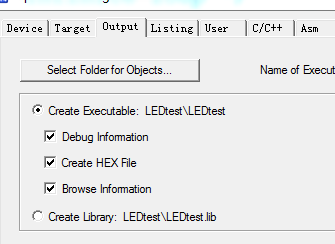
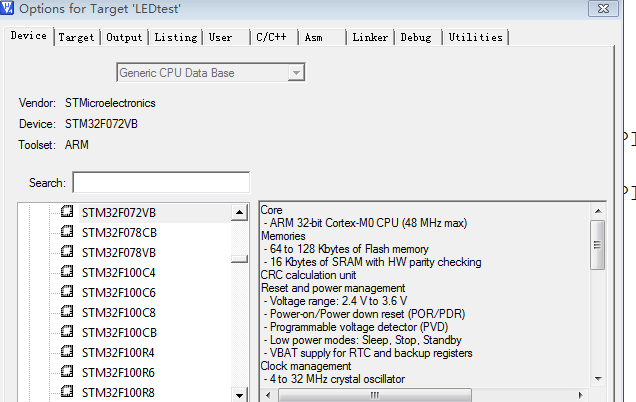
**设置keil的下载功能**

我们这里用的是st-link v2下载器



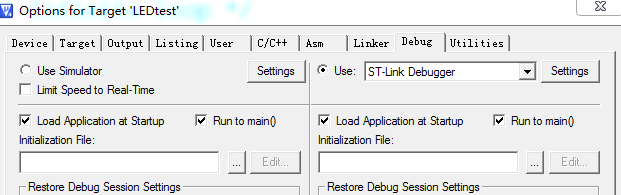
有些电脑可能在这里显示

安装ST-LINK驱动成功这里会显示节点



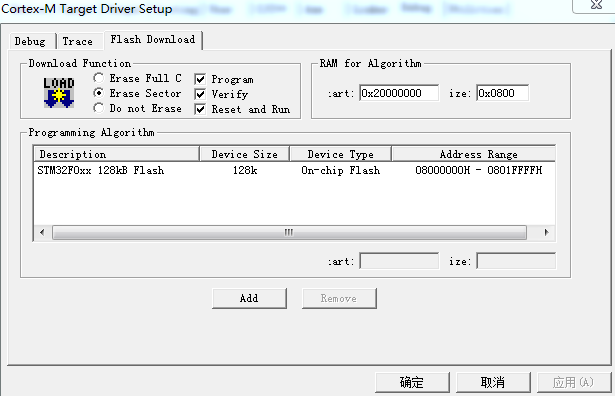
确认输出hex文件

确认型号



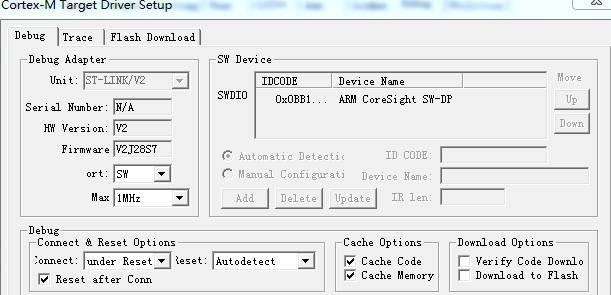
然后点击设置

选择st-link下载器



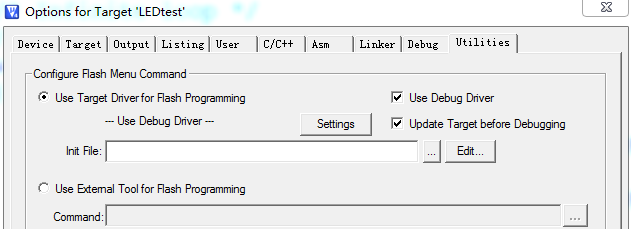
型号确认无误

复位要选择上

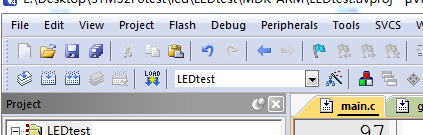


确认ST-LINK连接上

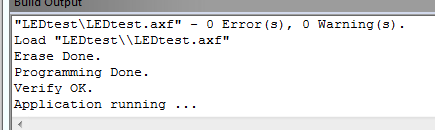
选择SW模式



确认这两个勾选上

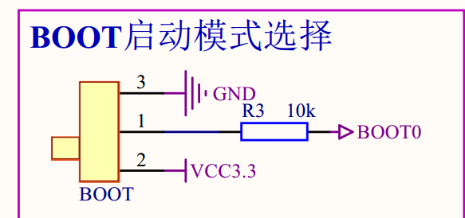


点击下载

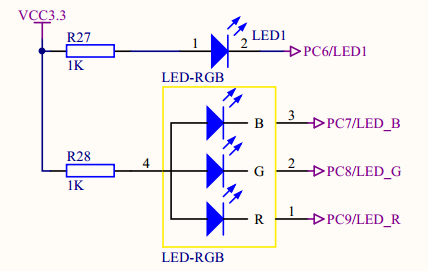


显示下载成功

如果程序正常下载后STM32F072没有运行起来，那么确认下boot引脚的启动模式

boot0要接地，才能从flash启动。所以这里要注意

**GPIO管脚输出操作**

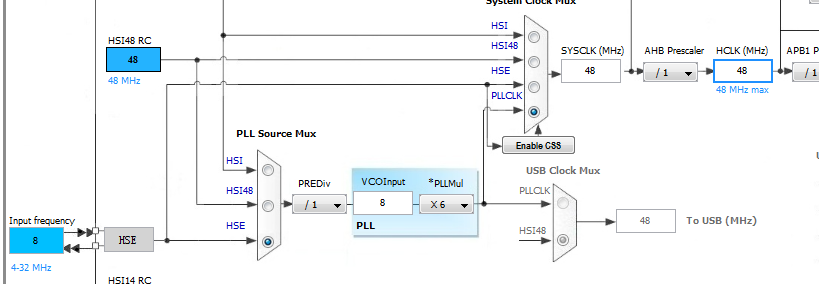
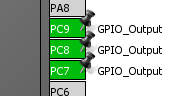


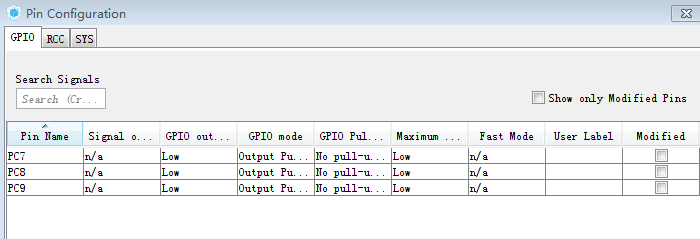
时钟设置外部晶振8M倍频48M

选择3个GPIO

为输出

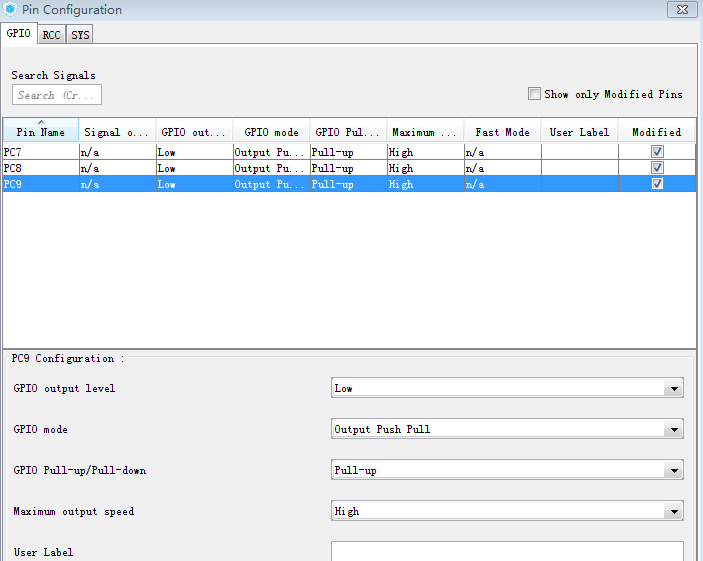
点亮三基色发光二极管

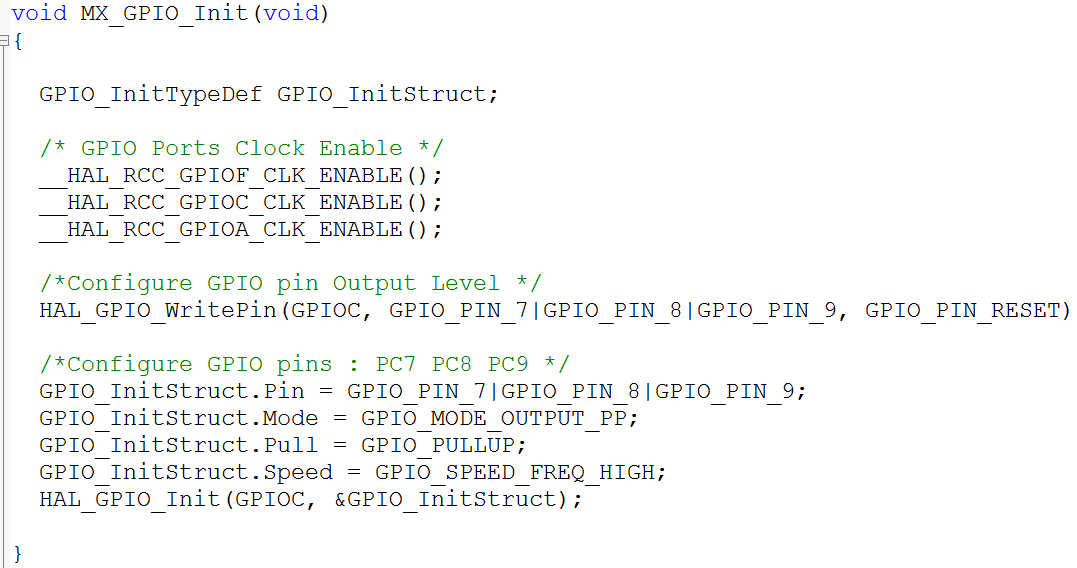




这个表里面就是3个GPIO了

设置3个GPIO上电初始化模式

然后按照上面CubeMX的方法生成keil工程



设置某个GPIO管脚模式

向某个GPIO管脚写0或者写1函数

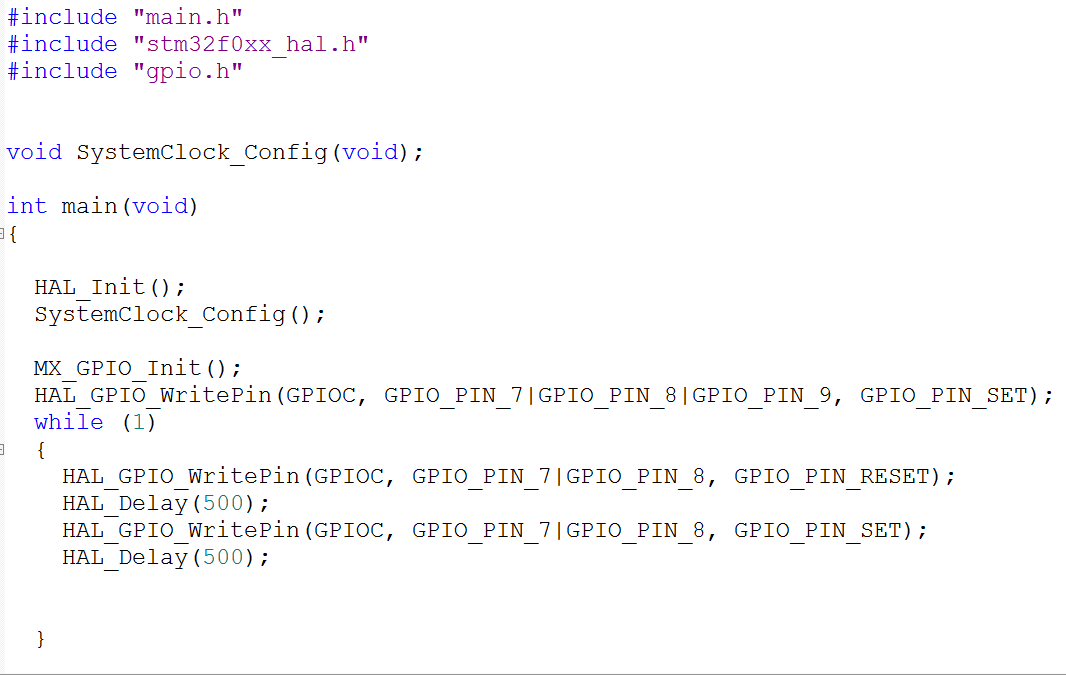
各组GPIO管脚时钟初始化函数

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP; //GPIO推挽输出模式

GPIO\_InitStruct.Pull = GPIO\_PULLUP; //GPIO带上拉电阻

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_HIGH;//GPIO电平输出速度为高速

HAL\_GPIO\_Init(GPIOC, &GPIO\_InitStruct);//将设置的GPIO参数初始化



设置完成LED每延迟500ms闪烁

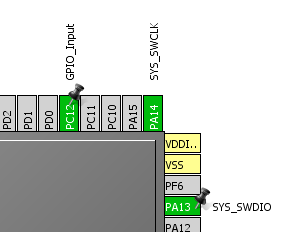
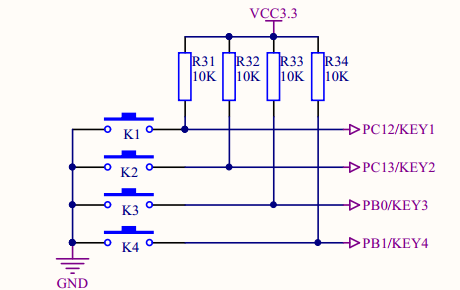
HAL\_GPIO\_WritePin (GPIO组, GPIO单个引脚, 电平状态);

GPIO\_PIN\_RESET = 0

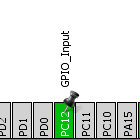
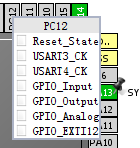
GPIO\_PIN\_SET = 1

**GPIO管脚输入操作**

我用PC12,PC13做按键输入



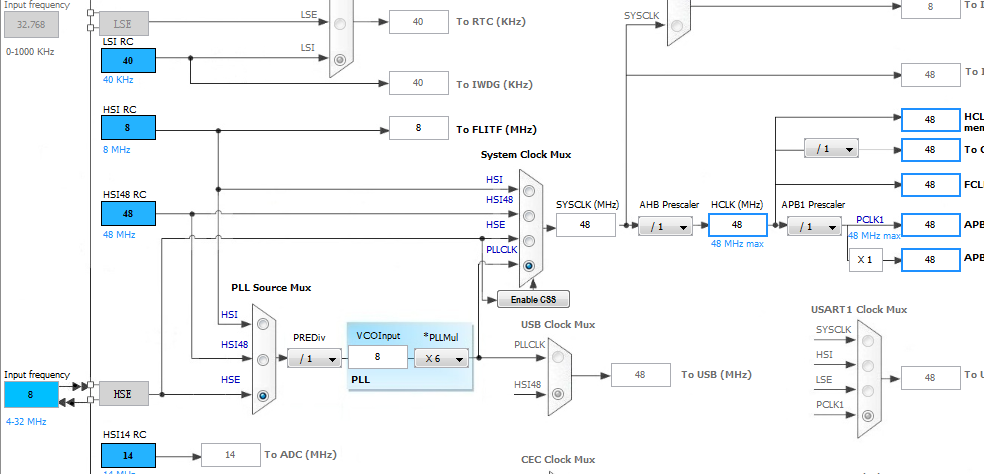
PC12设置为输入



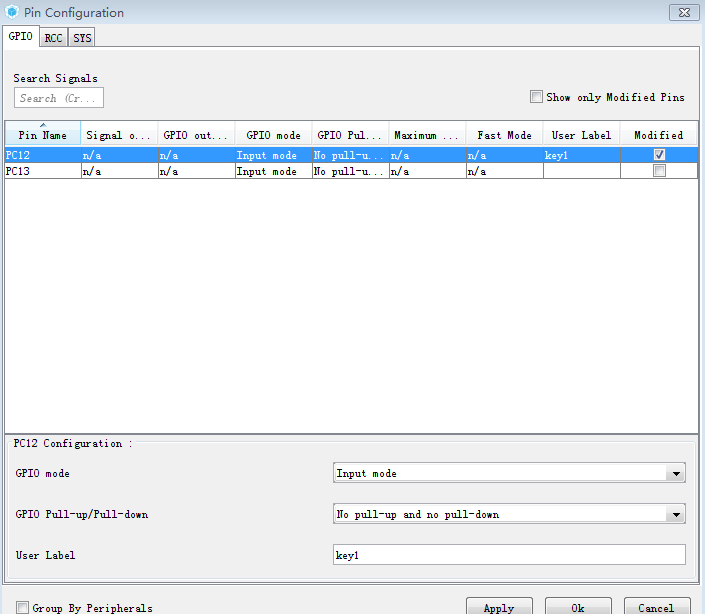
还是用外部时钟

PC13设置为输入

PC12设置为输入



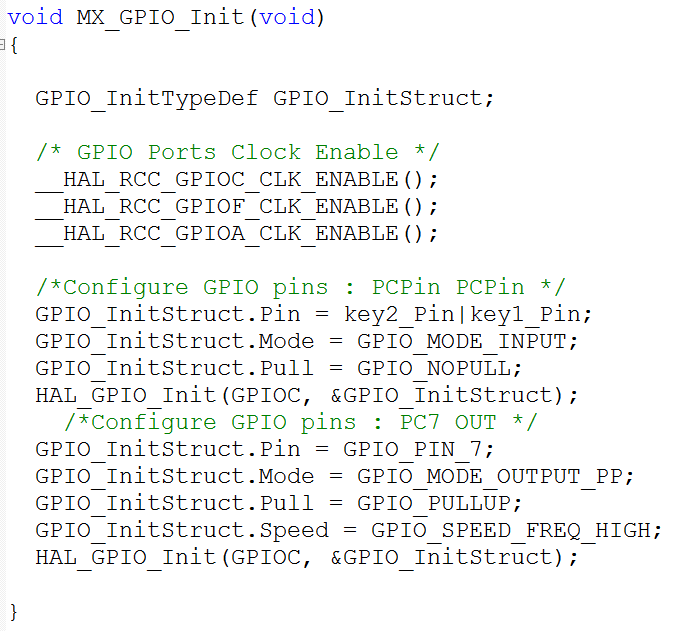
时钟设置还是老样子，外部晶振8M，倍频48M

 PC13设置方式一样的

设置管脚功能

因为我们PC12硬件有上拉电阻了，这里就不需要内部上拉

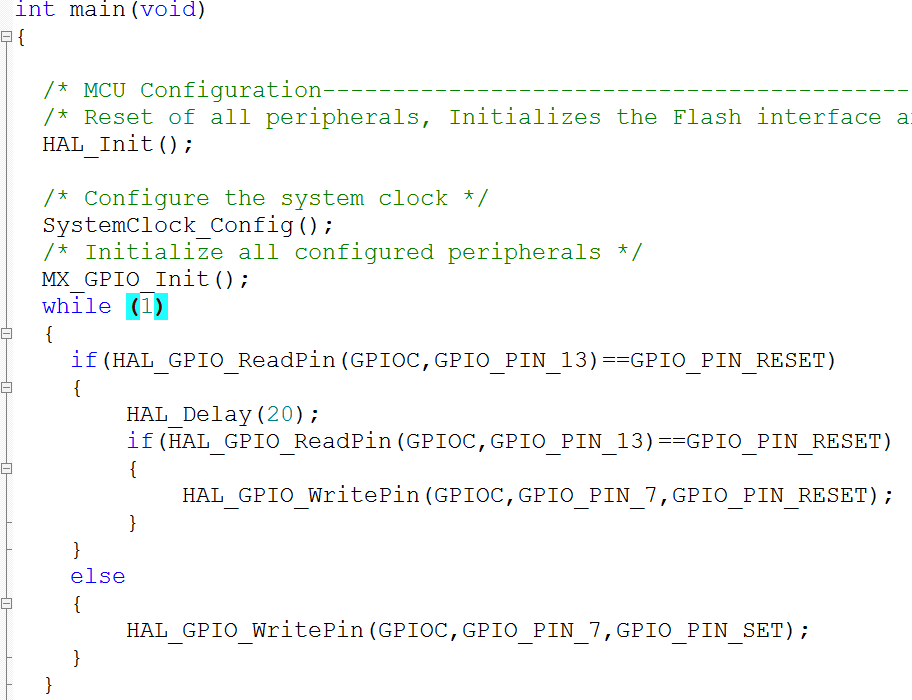
然后生成keil工程



设置另外GPIO管脚输出模式

设置GPIO管脚输入模式

打开GPIO时钟



GPIO\_PinState HAL\_GPIO\_ReadPin(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin) 读单个GPIO电平

HAL\_GPIO\_ReadPin(GPIOC,GPIO\_PIN\_13)返回GPIO\_PIN\_RESET就是0返回GPIO\_PIN\_SET是1

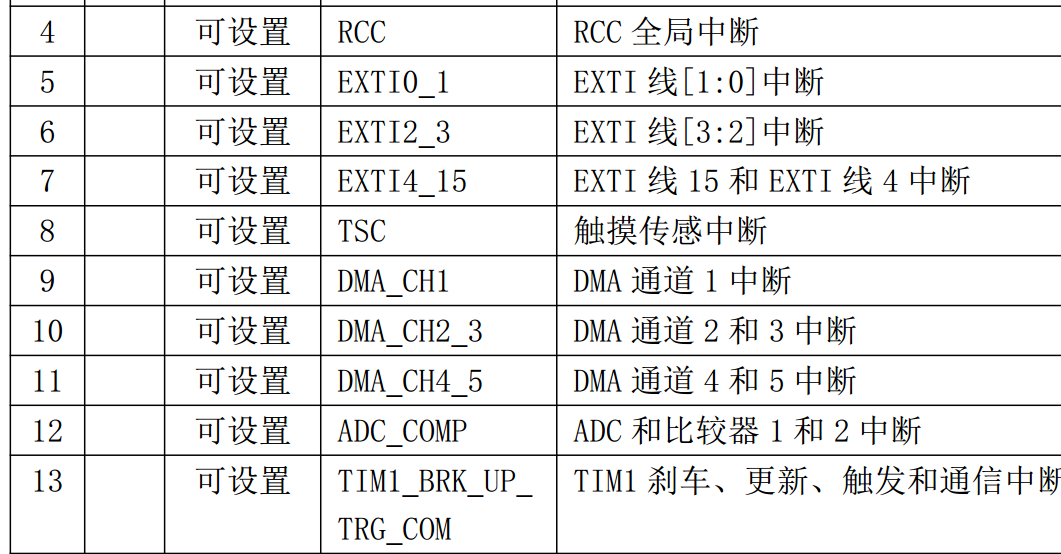
**STM32F0中断操作**



软件配置中断优先级可以选择4种，设置0优先级最高，3优先级最低

如果两个IO管脚软件设置的中断优先级一样，比如都设置为优先级1，那么先触发哪个IO口的中断函数呢？，这就要根据中断向量表来决定了。





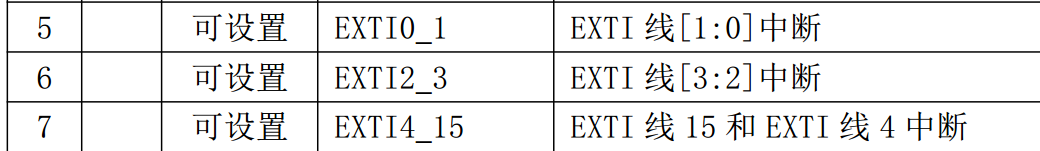
根据中断向量表，程序会先执行最小的那个向量中断函数，这里是5，EXTI0\_1

EXTI2\_3中断线软件设置优先级也为1

EXTI0\_1中断线软件设置优先级为1

我现在来设置IO管脚的输入中断

STM32F0这款芯片GPIO分为A,B,C,D,E,F组，每一组有0～15个引脚，但是F组引脚不足15

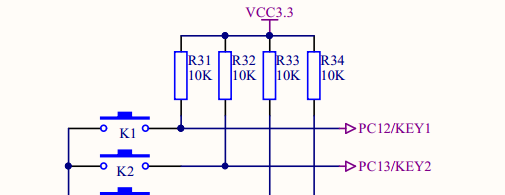


EXTI0\_1对应GPIOA0,GPIOA1，或者是GPIOB0,GPIOB1其余的GPIO组也是一样

EXTI2\_3对应GPIOA2,GPIOA3，或者是GPIOB2,GPIOB3其余的GPIO组也是一样

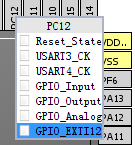
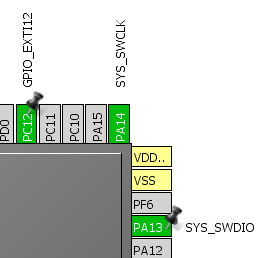
EXTI4\_15对应GPIOA4 ～ GPIOA15,或者是GPIOB4～GPIOB15其余的GPIO组也是一样

GPIO组里面每个管脚的EXTI中断线不一样，但是GPIO组的中断线一样，比如GPIOA0,GPIOB0用的都是同一个中断线，所以这两个IO口共享一个中断函数。

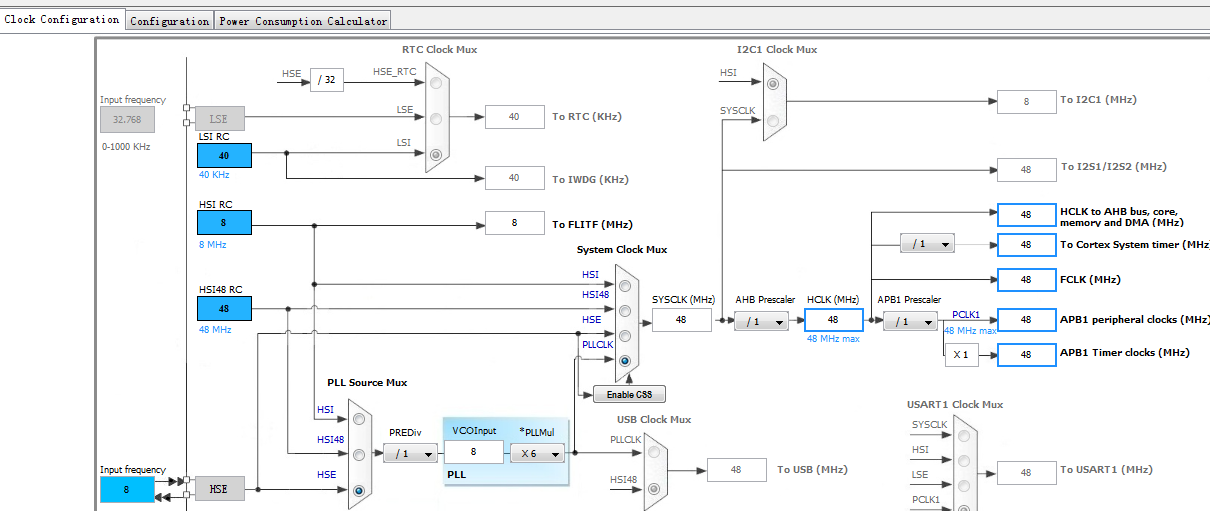


设置PC12引脚为外部中断，所以选择EXTI12

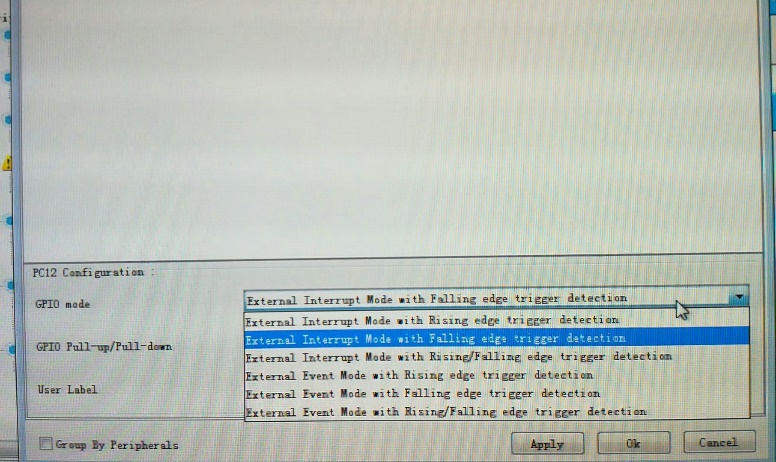
我们还是以PC12,PC13按键为例。

PC13一样设置

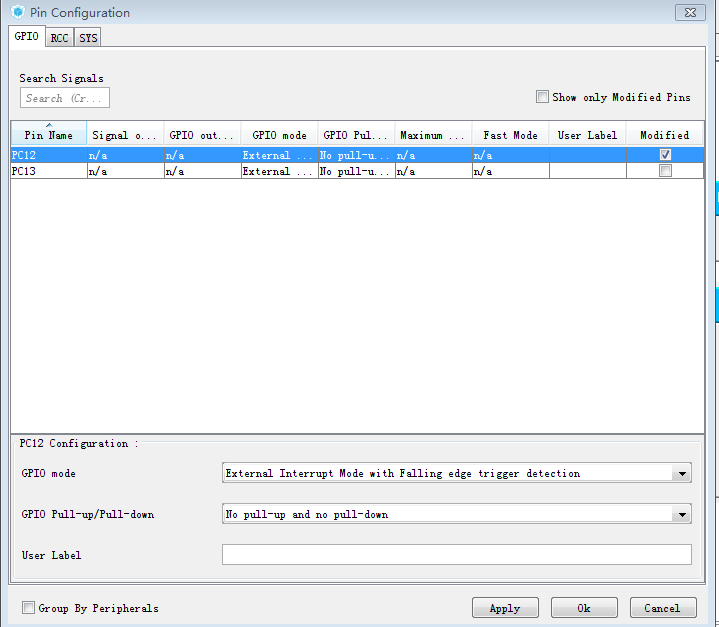


时钟设置还是和标准的外部8M晶振一样。倍频48M



和内部事件触发

中断引脚功能分外部中断触发



因为电路有上拉电阻这里我们就不用内部上拉

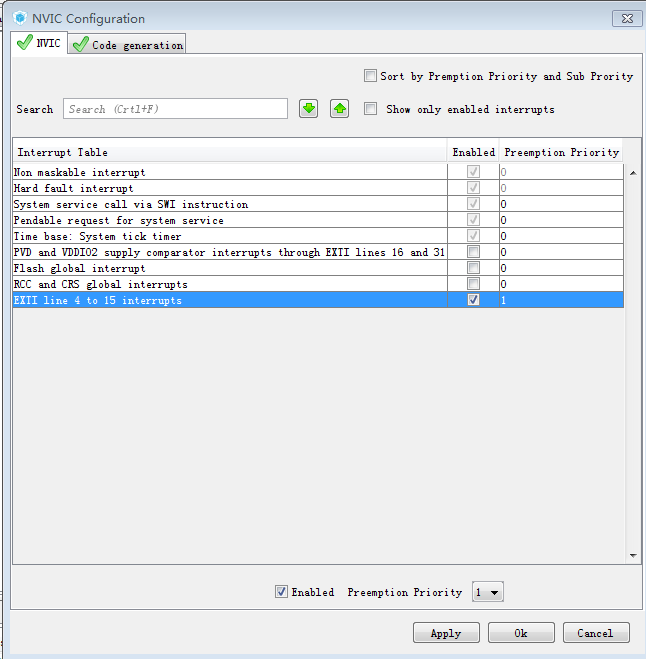
我选择按钮下降沿触发

然后取名字确定。引脚外部中断功能只分为上升沿触发，下降沿触发，双边沿触发，没有高电平触发和低电平触发。

设置中断优先级



我们还要选择NVIC设置中断线EXTI的中断优先级

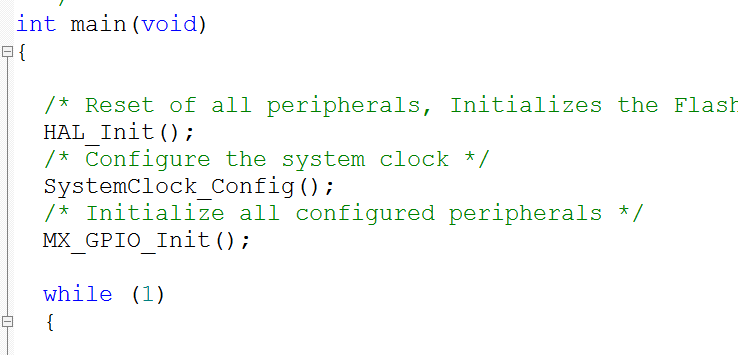


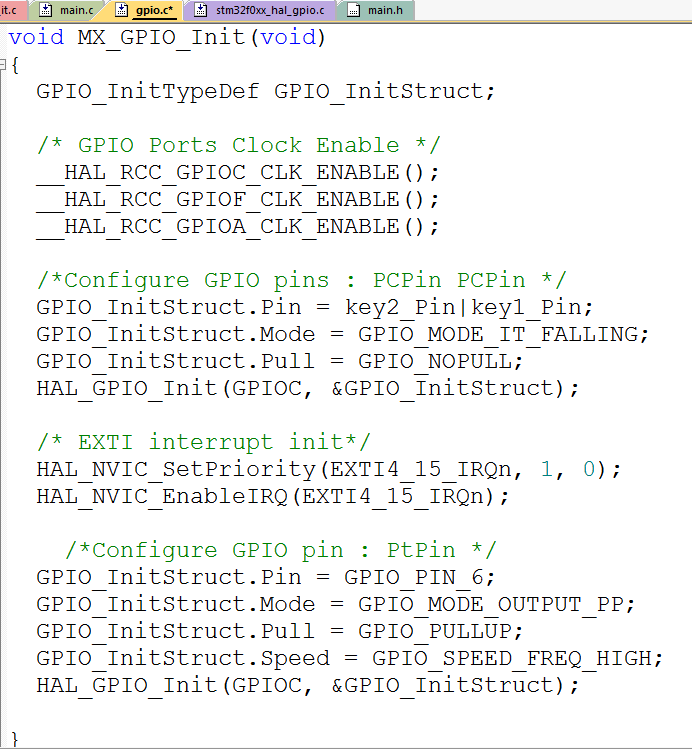
这里是设置中断优先级，我这里设置为1

因为我们是PC12,PC13，所以对应的中断线是EXTI4～15,勾选上

然后点击OK

生成keil工程





设置IO口输出的代码

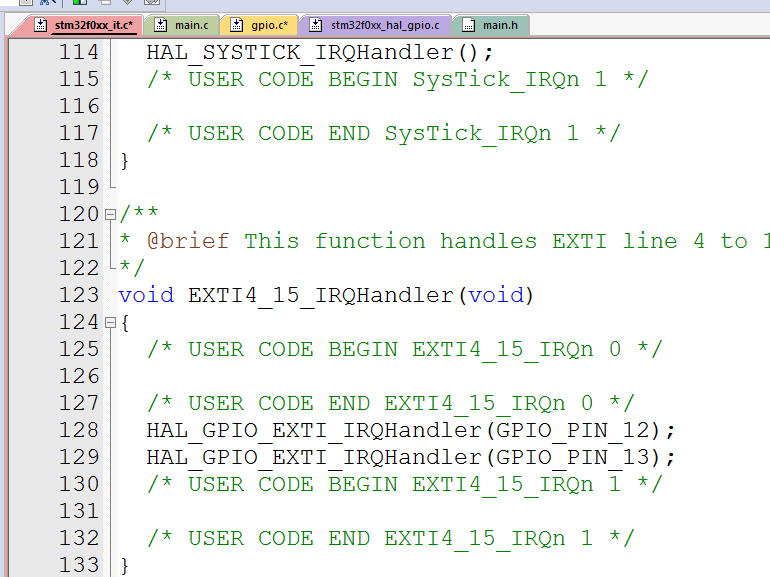
设置中断线优先级

不需要内部上下拉电阻

管脚下降沿触发中断

写入具有中断功能的管脚

打开你使用IO口的时钟



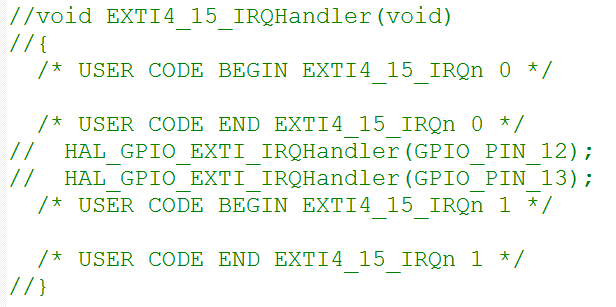
这个函数里面有清除中断标志位，所以你就直接用就是了

这就是EXTI4\_15中断线上的中断处理函数

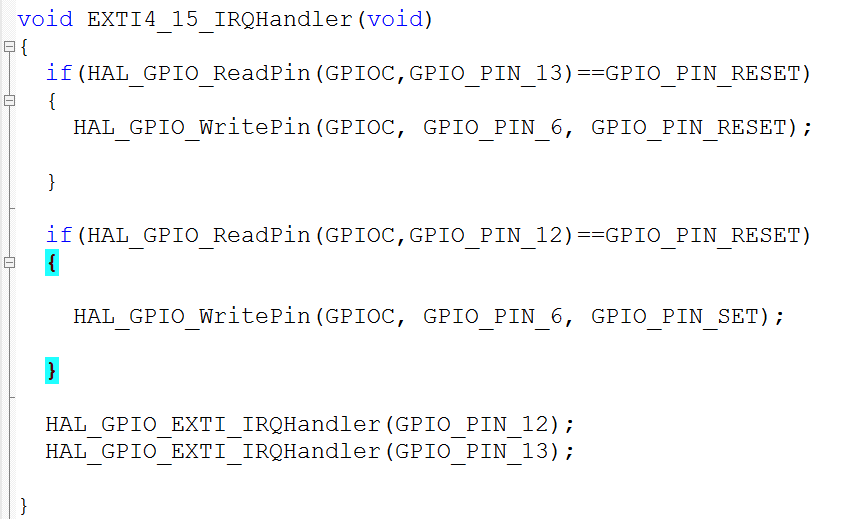
CubeMX生成的keil工程中断函数都放在stm32f0xx\_it.c文件这里

我觉得把中断函数放在单独的一个文件里面我感觉不爽

把中断函数复制到你想要的文件下面，删除it.c里面的中断函数



我直接把中断函数复制到main.c主函数文件下面



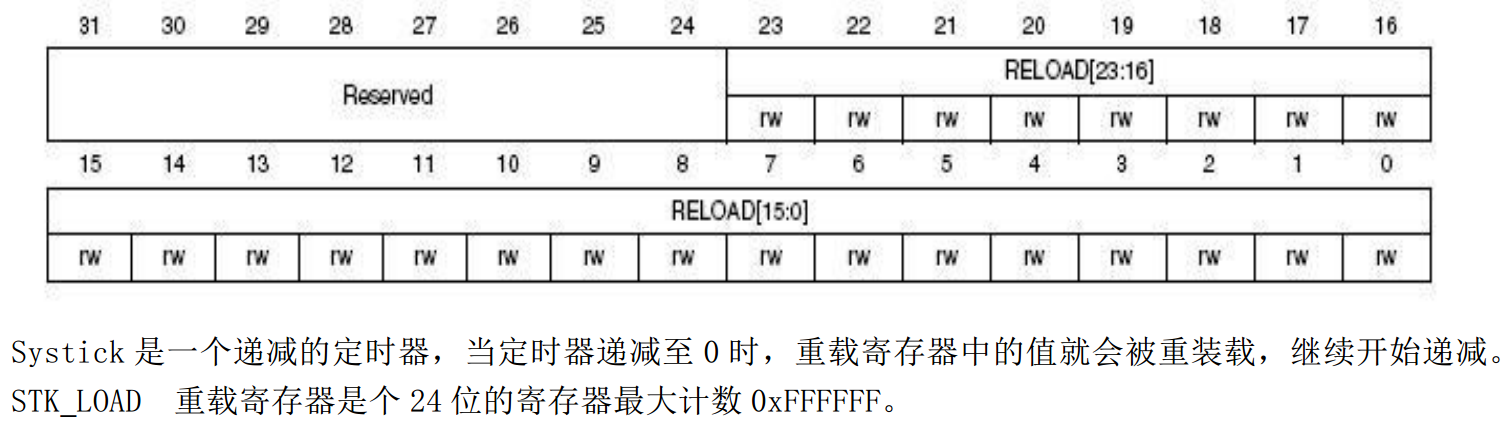
清除中断标志位

因为我的两个IO口都是用的同一个中断线，所以我就只有用判断IO口的方法来确定是哪个IO口发生了中断

外部中断就讲解完毕了

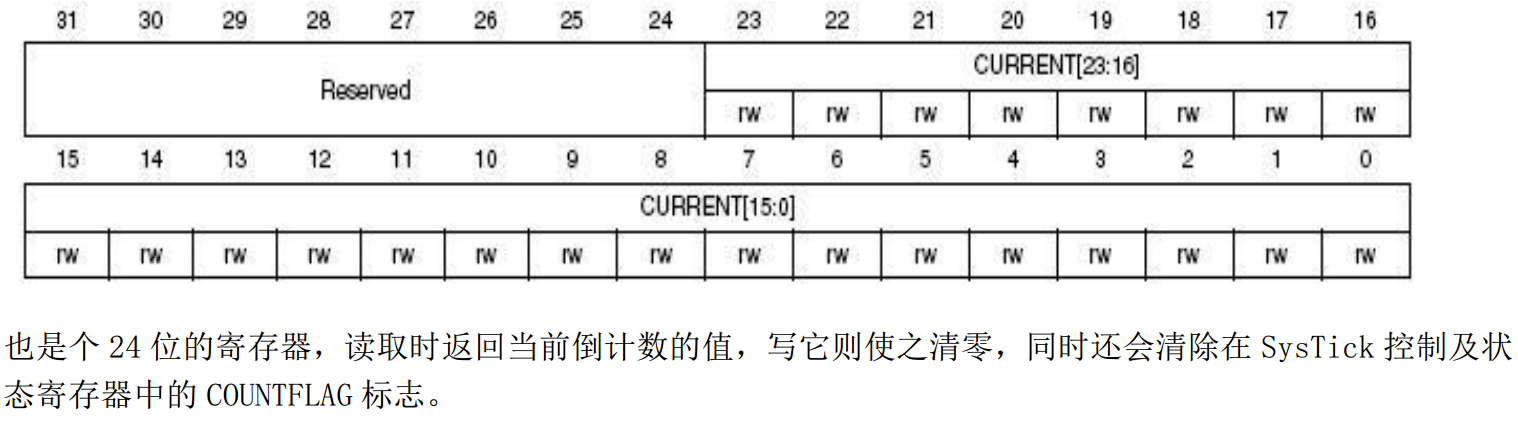
**Systick内核定时器做延时函数**

我写1000值给STK\_LOAD寄存器



每来一个时钟我的LOAD寄存器就减1

STK\_VAL寄存器记录LOAD的值现在减了多少



Systick时钟开启流程



STK\_LOAD寄存器接收到脉冲减1

HCLK(48M)/8 产生一个脉冲

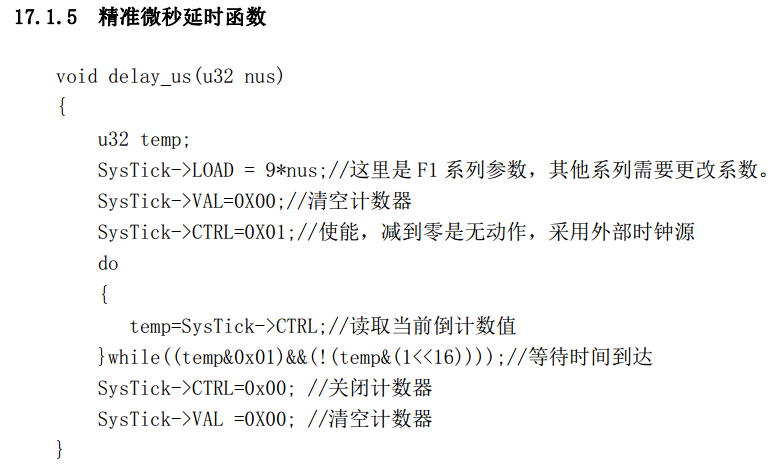
关闭中断使能位，我们用查询

开启SYstick功能

这个HAL库里面的延时函数采用了中断功能，所以这个延时函数精确度不高。我们还是用查询法来完成延时函数。

STM32F1系列是72M，所以这里应该是72/8 = 9M，那么9M=9000000给systick，也就是每秒钟可以记录9000000个数。

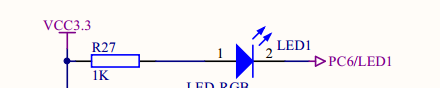
因为1秒记录9000000个数，那么1ms就要记录9000个数，那么1us就要记录9个数。所以这里是9



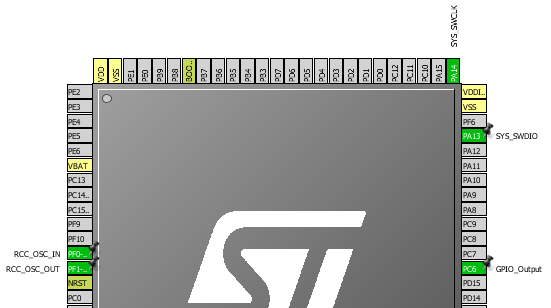
如果STM32f1精确延时1ms,就将LOAD的9改成9000即可



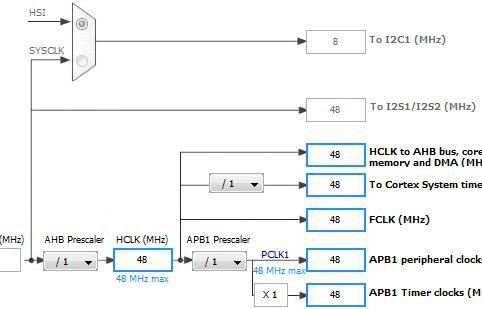
下面我们再来看看STM32F0的延时函数



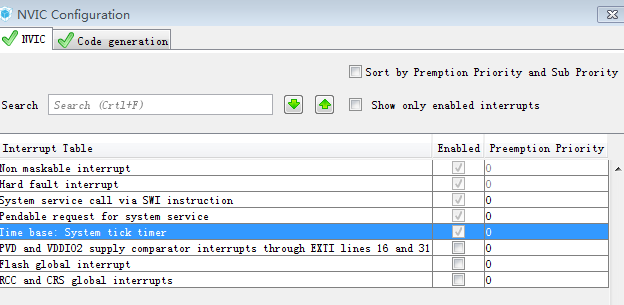
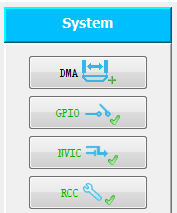
我们还是用PC6引脚驱动LED延时为例



引脚定义老样子



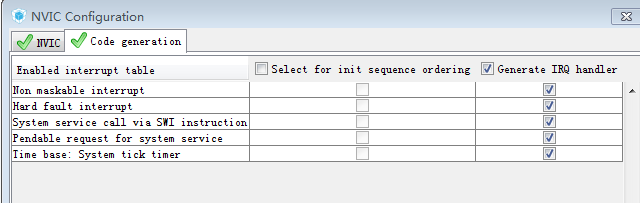
时钟还是外部晶振48M



你如果要cubeMX自动生成systick中断函数就在这里打勾

CubeMX默认必须生成systick时钟

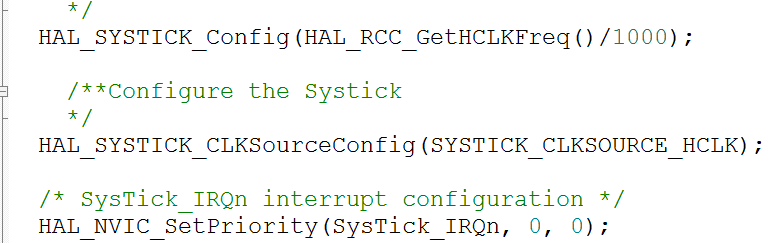
Systick系统设置在NVIC里面

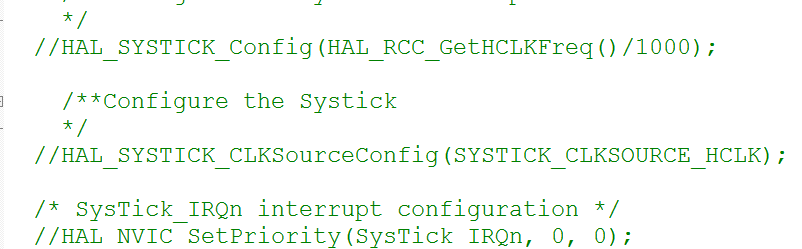


然后生成keil工程

因为cubeMX生成的systick时钟是不准的，但是为了芯片上电初始化systick时钟

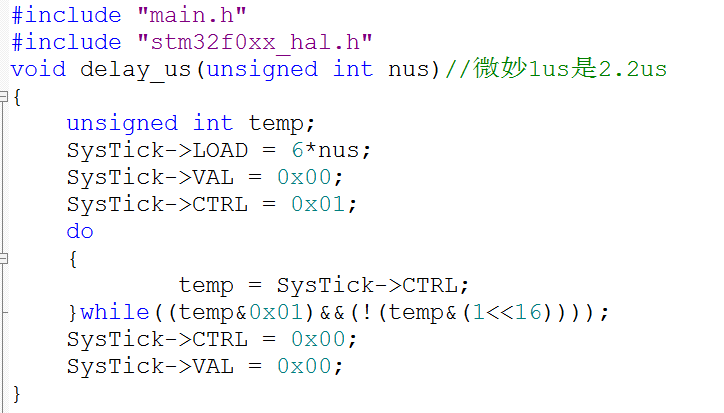
将keil工程主函数下的SystemClock\_Config里面的systick部分初始化函数去掉

屏蔽掉这三个函数

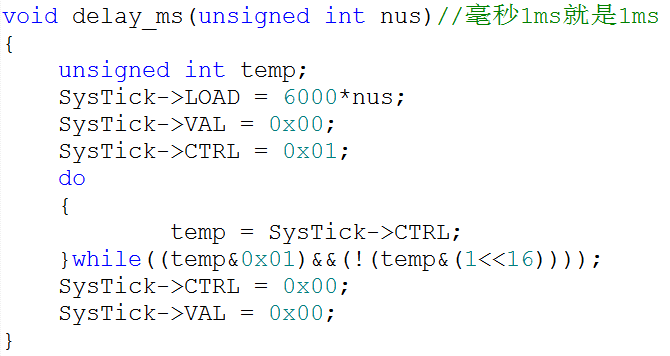


屏蔽成功。

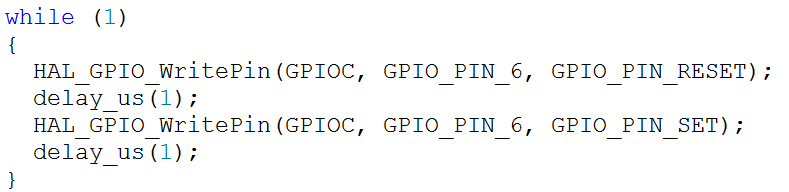
然后在主函数上自己实现systick查询方式的微妙和毫秒延时



和STM32F1一样，HCLK(48M)/8=6000000，1微妙=6



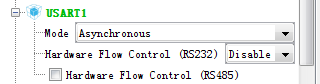
这里自然是6000

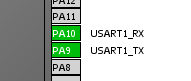
程序执行正常，systick延时讲解完毕

**USART串口通讯**



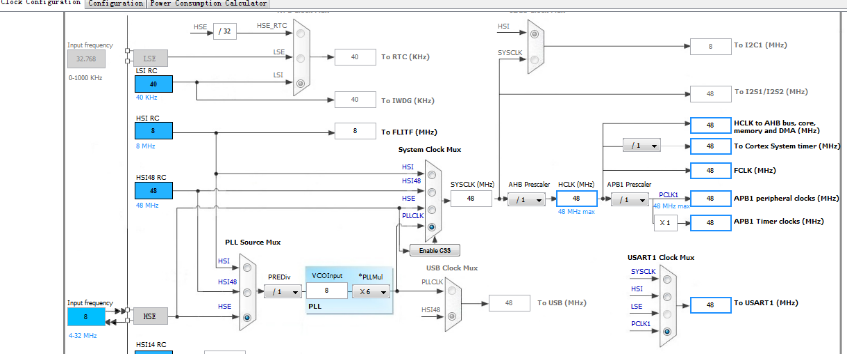
我们选择异步模式



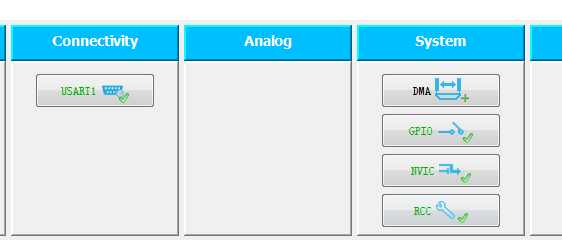


选择异步模式后USART1就自动选择上了

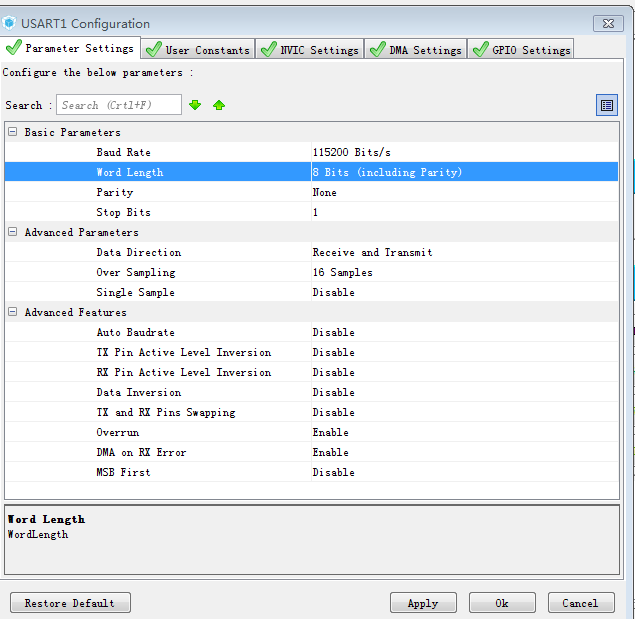
其余的晶振啊，SWD接口都要选择上。



时钟还是外部8M倍频48M，这里选择PCLK1的48M时钟给串口



这里生成了一个USART1选项，点击进入



1位停止位

无校验位

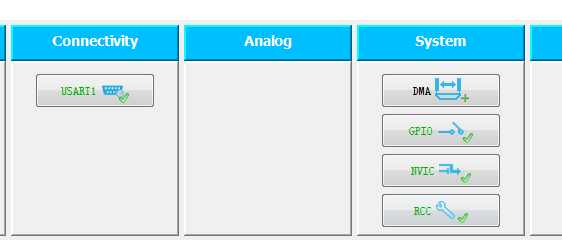
数据位选择8位

这里我们写115200波特率

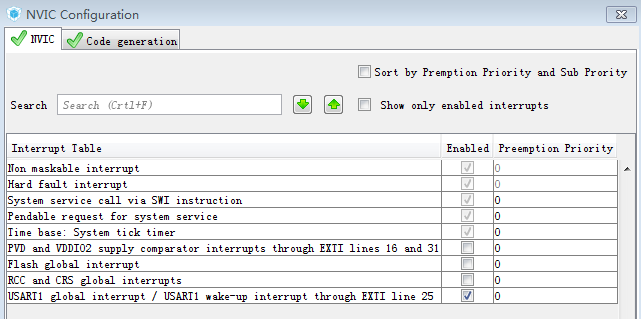


把串口中断打开

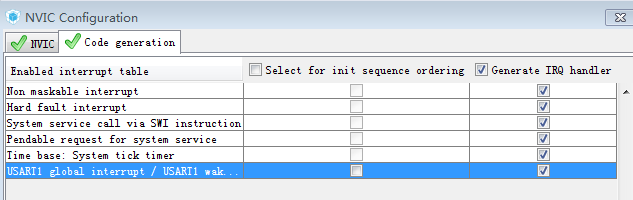
然后点击OK



再来设置串口中断优先级

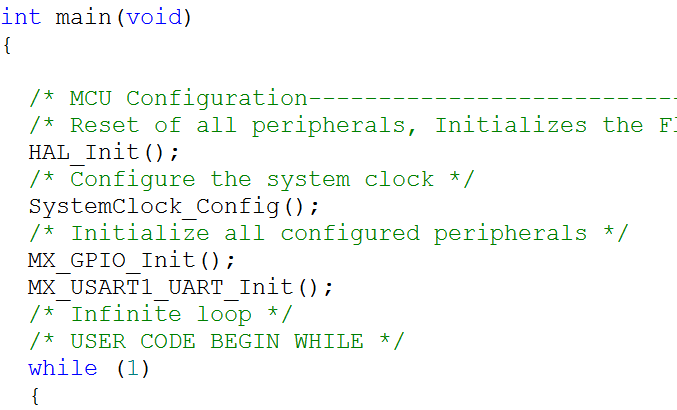


确认串口中断已经选上



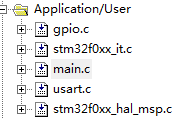
生成串口中断服务程序

然后点击OK，设置一下工具栏，生成keil工程

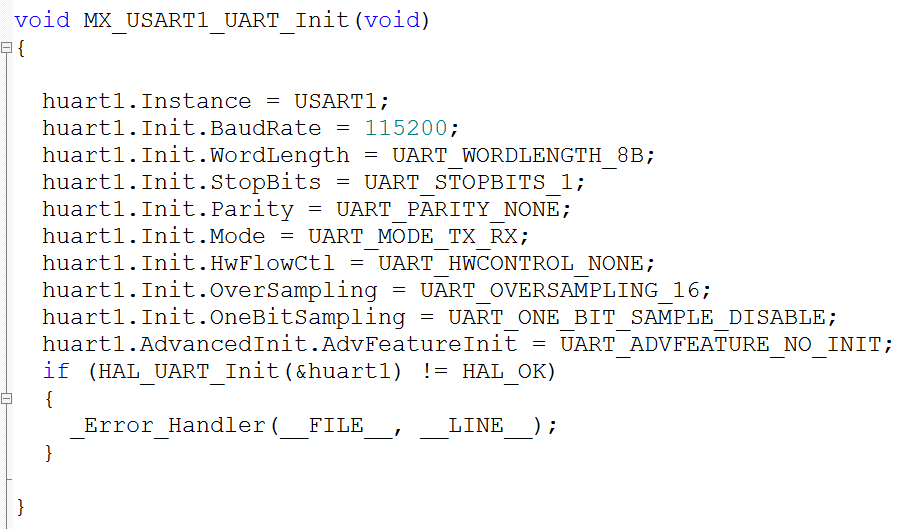


这里初始化串口

这里只初始化IO时钟



这里生成了串口程序文件



无校验位

1位停止位

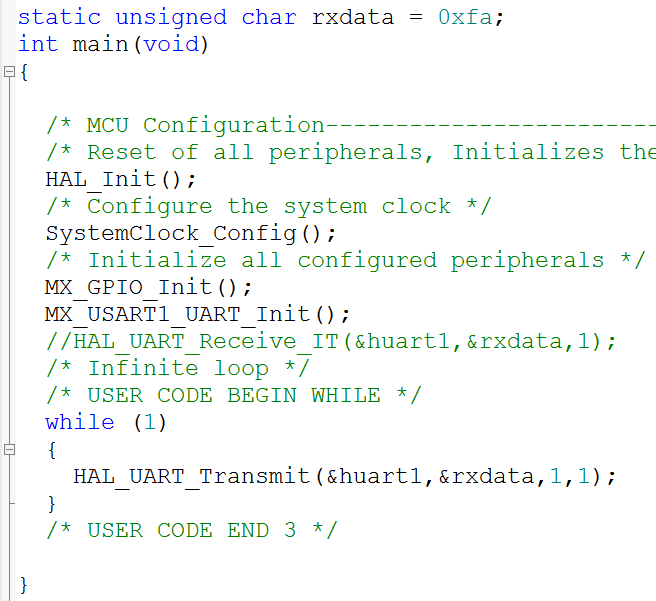
8位数据位

波特率115200

使用串口USART1

串口配置完毕，发送一个字节数据

定义一个要发送到数据



用HAL\_UART\_Transmit发送

HAL\_StatusTypeDef HAL\_UART\_Transmit(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size, uint32\_t Timeout)

UART\_HandleTypeDef \*huart ：串口号

uint8\_t \*pData: 1个字节的数据，8位，也可以传入一个数组的数据

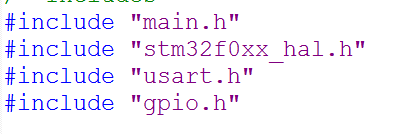
uint16\_t Size：发送多少个字节，我这里是一个字节，所以天

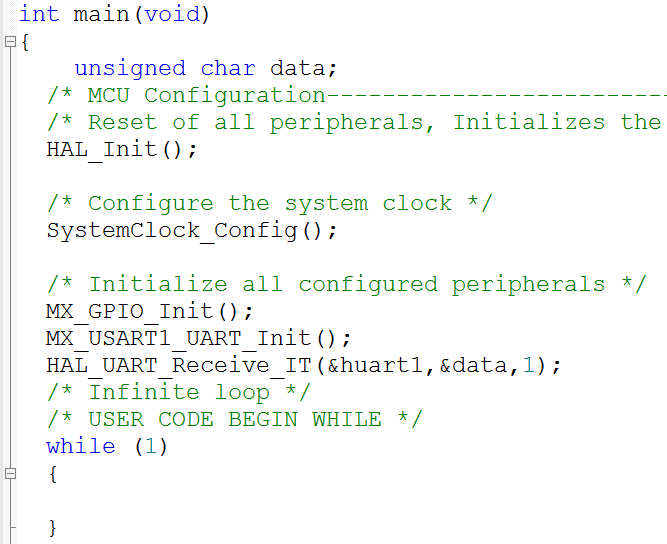
uint32\_t Timeout：这个timeou是决定发送一个字节要多少时间,单位为毫秒

当你波特率是9600bit的时候，9600/8 = 1200个字节，1秒钟发送1200个字节(1000ms/1200) = 0.83ms，那么我的timeout就要设置为1ms

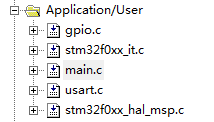
当你的波特率为115200的时候，1秒钟发送14400字节，一个字节需要0.0695ms，所以timeout设置1ms也是可以的

串口中断接受一个字节的数据



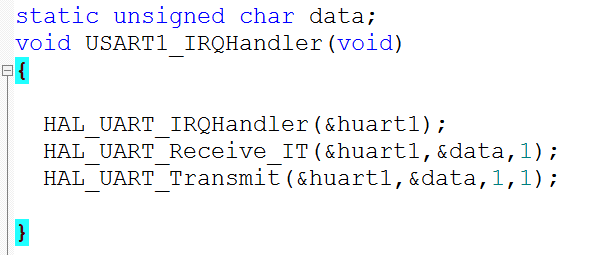


因为串口接受程序和串口中断使能程序都在这个函数里面，所以主函数先执行这个函数是为了打开中断使能



CubeMX生成的串口中断服务程序在这个文件里面，

接受到串口数据进入中断服务程序



最后用发送函数转发出去

然后用Receive函数接受数据

一定要先执行handler函数清除中断标志位

HAL\_StatusTypeDef HAL\_UART\_Receive\_IT(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size)

UART\_HandleTypeDef \*huart：串口号

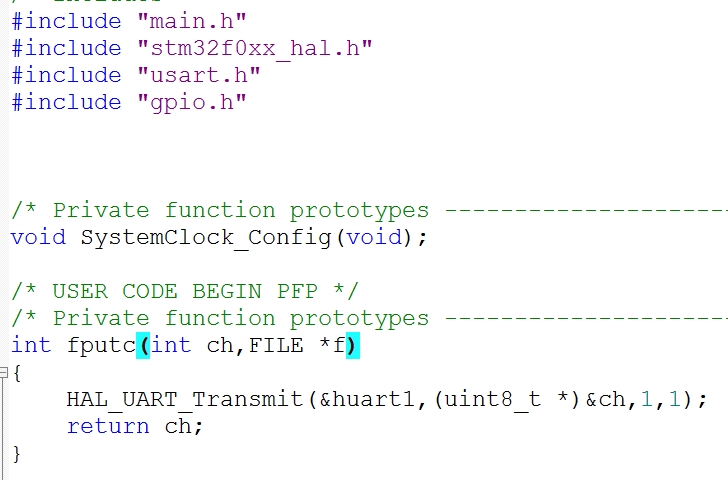
uint8\_t \*pData：要接受的数据类型

uint16\_t Size：接受多少个字节数据

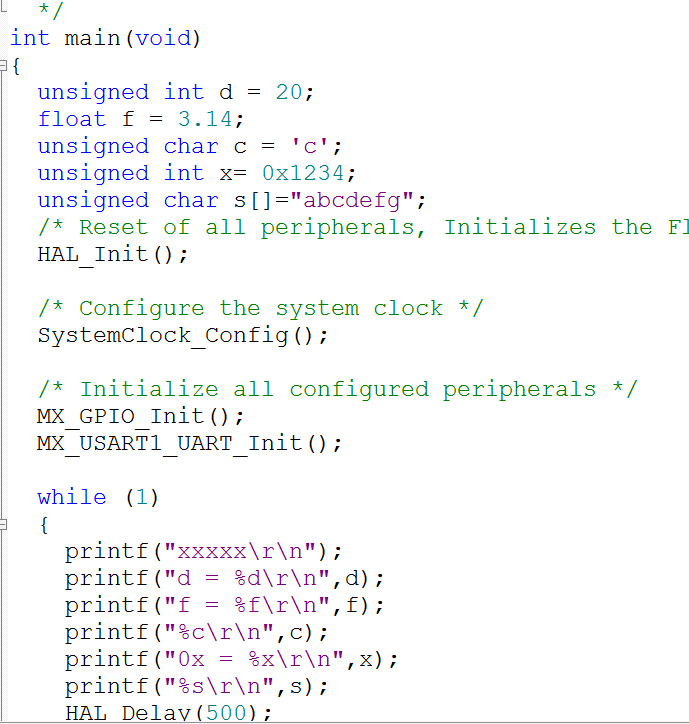
这就是电脑发送给F0的数据又转回电脑。

**串口映射成printf实现**

串口程序生成方法和前面的USART串口通讯方式一样。

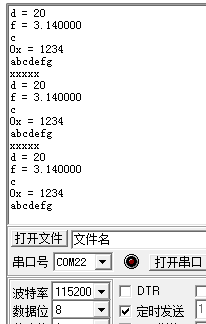


找个放全局函数的文件添加fputc函数

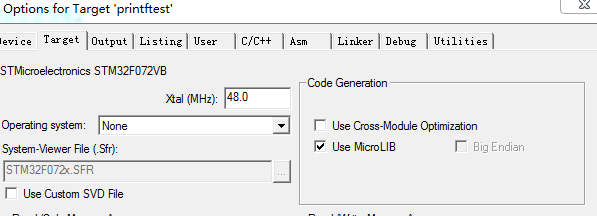


输出结果正常

打印各种类型



确保keil的Use配置上了的



定义整形，浮点型，字符型，16进制。

注意字符串初始化必须用数组承载，用指针不行

这就是STM32F0芯片printf函数实现

**CAN总线实现**

STM32f0CAN总线固件库函数波特率计算

CAN波特率=CAN总线频率/(perscaler分频器系数\*(tq+tBS1+tBS2))

tq：值固定为1

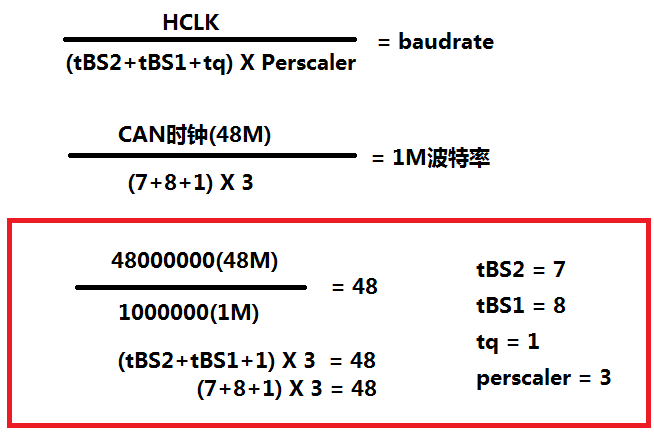
BRP： 设置范围1～4，一般设置为3，尽量别动。

CAN\_BS1：设置范围1～16

CAN\_BS2：设置范围1～8

设置注意：BS1 ≥ BS2(BS1值必须大于等于BS2)，BS2 ≥1个CAN时钟周期，BS2 ≥ 2perscaler

公式分解:



tq不要变

满足BS2大于2倍perscaler的要求

满足BS1大于BS2的要求

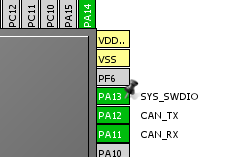
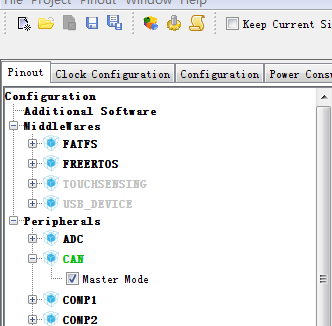
我们也可以设定波特率然后计算BS1,BS2

计算1M波特率

计算CAN波特率公式

现在得到tBS2 = 7, tBS1 = 8, tq = 1

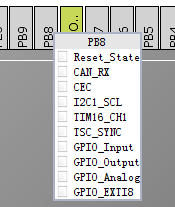
打开CubeMX软件，生成CAN代码



这里就会默认生成CAN\_TX和CAN\_RX的引脚

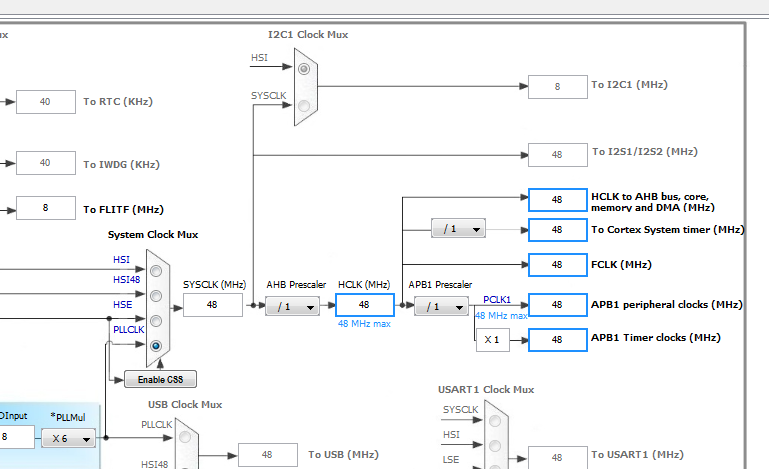
选择CAN打上勾

如果你的PA12,PA11引脚被其它功能占用了，那么你可以改成PB8,PB9



软件自动帮你取消PA12,PA11，然后把CAN映射到PB9,PB8引脚

点击PB8,PB9，选择CAN功能

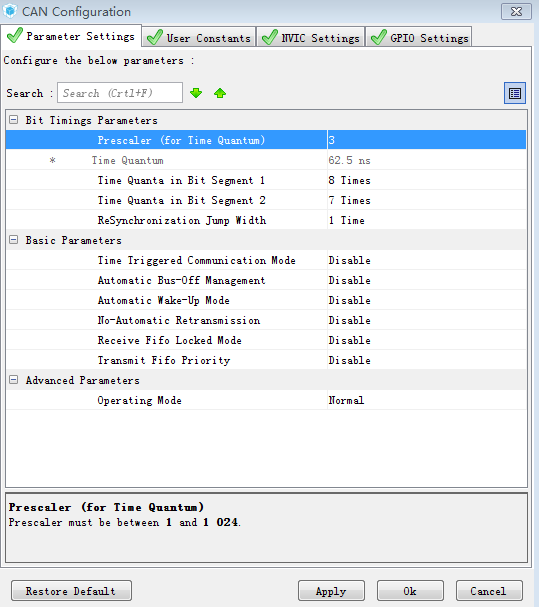


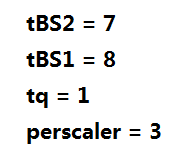
老样子选择外部晶振8M，倍频48M系统时钟，CAN时钟和系统时钟HCLK一样48M



这里输入3分频perscaler

设置CAN功能





注意：CubeMX软件比较奇怪，必须先选择BS1,BS2然后才能去修改perscaler

选择正常模式

tBS2选择7

tBS1选择8



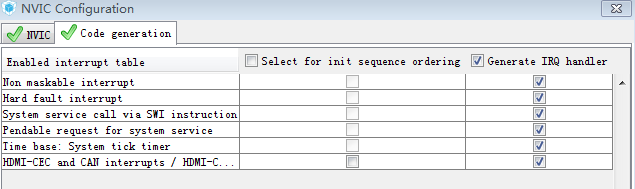
打开CAN接受中断



选择CAN中断优先级

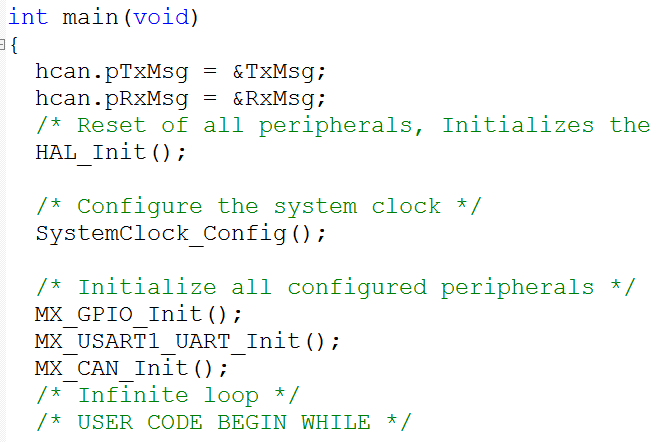


确认CAN中断产生函数



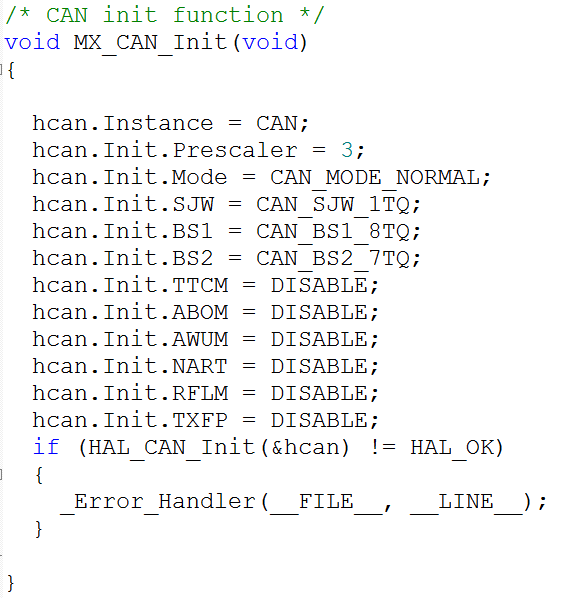
确认生成中断函数

CubeMX设置完毕，按照上面方法设置选项栏，然后生成keil文件。



确认初始化函数

在can.c文件下



tBS2 = 7

tBS1 = 8

tq = 1

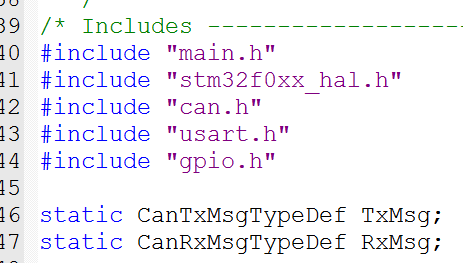
正常模式

3分频

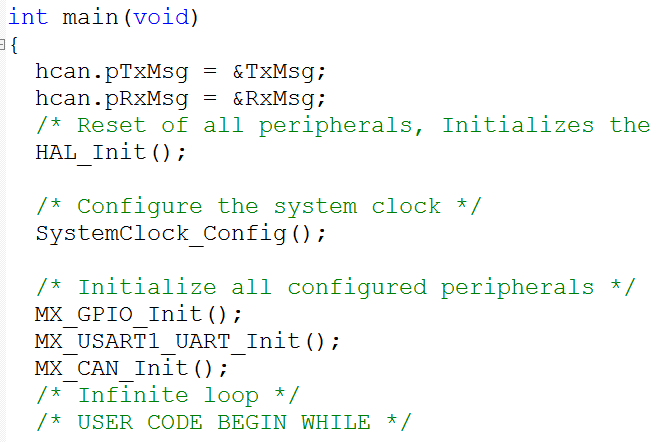
设备节点为CAN

CAN总线初始化完成，波特率为1M

然后我们STM32F0的CAN控制器发送数据给电脑，我们发送标准数据帧(ID是16位)

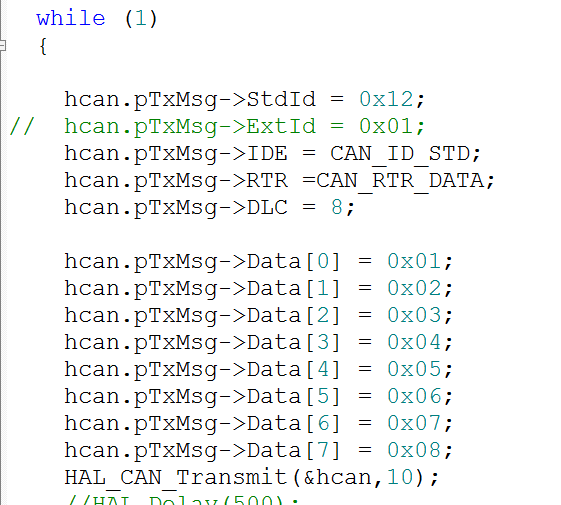


我们要在内存开辟一个CAN总线的结构体,这个一定要做



CAN总线发送的数据全部要向hcan.pTxMsg写入。所以我们得让hcan.pTxMsg指向TxMsg这个开辟的内存空间

这样hcan.pTxMsg指针就有了确切的地址



将设置好的CAN参数写入CAN寄存器

发送数据个数

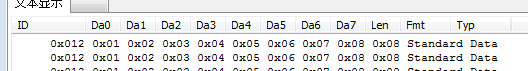
发送数据

数据帧

CAN\_ID\_STD是标准ID，因为我们Extid没有写，所以发送的是标准ID

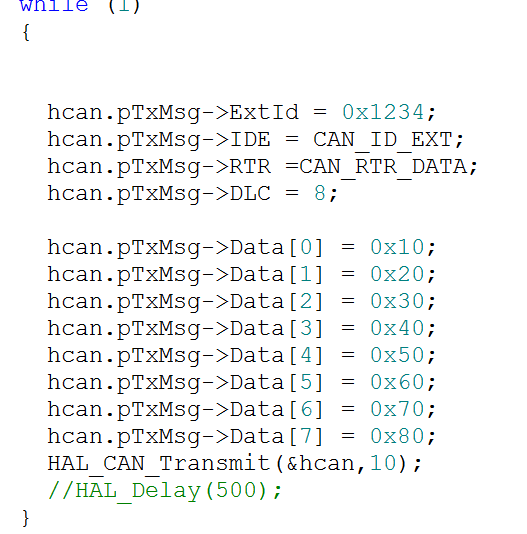
写入标准ID,这个值最大范围是0～0x7FF

我们发送标准帧，所以ExtId不写



我们再来发送扩展数据帧(ID是29位)

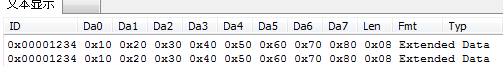
扩展ID发送，can波特率和上面一样，can缓冲区pTxMsg和上面一样，就修改ID就行



将CAN\_ID\_STD标准ID模式改成扩展ID模式CAN\_ID\_EXT

将StdId去掉，用ExtId来承载扩展数据ID，范围0～0x1FFFFFFF

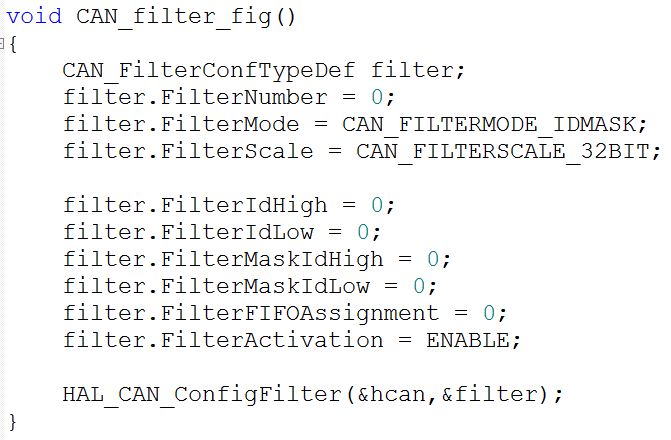
其余不变，编译下载运行。



CAN总线实现接受

CAN总线是在中断程序里面接受主机发过来的数据,然后我用串口打印出来

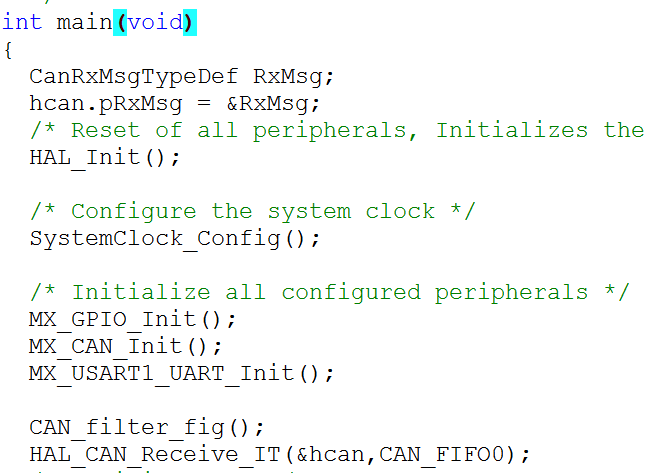
我们要先在主函数上加屏蔽滤波功能，接受程序必须先这样,因为内核规定要加屏蔽滤波函数CAN才能正常工作。



然后将CAN总线设置的屏蔽滤波参数写入寄存器

在屏蔽滤波里面一定要选择激活该功能

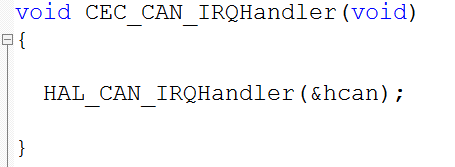
现在我们接受所有的数据,所以我们这里不做屏蔽处理



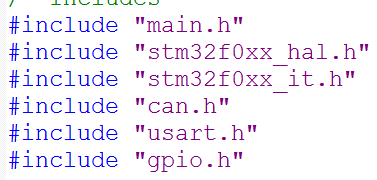
一定要在初始化时提前打开一次CAN接受中断，接受的数据放在FIFO0

一定要初始化屏蔽滤波

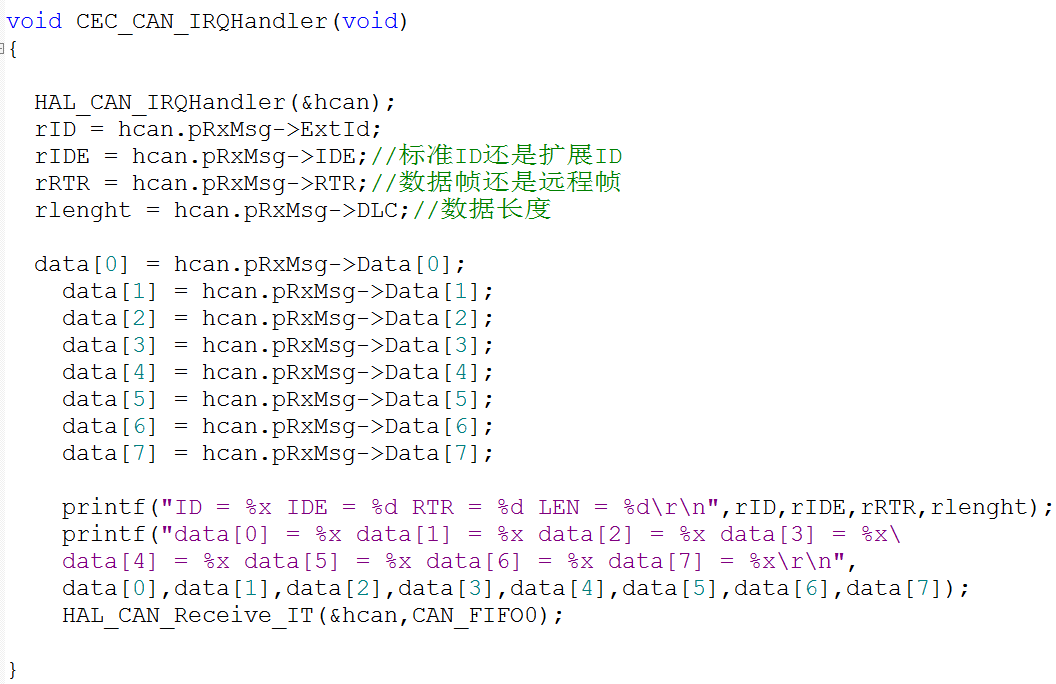
在主函数里面要先给接受CAN总线的指针赋地址



本来CAN中断服务函数是在it.c文件里面的，但是我为了方便就剪贴到主函数去



记着主函数要加it.h和can.h头文件

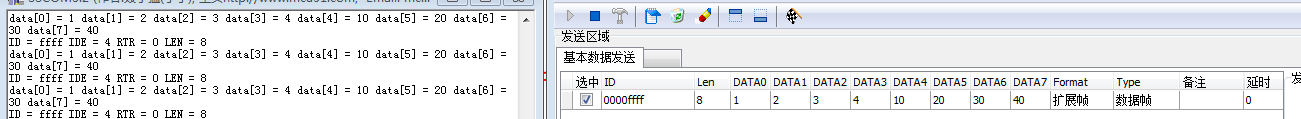


CAN中断服务程序接受完数据会自动关闭CAN中断，所以这里还要再打开

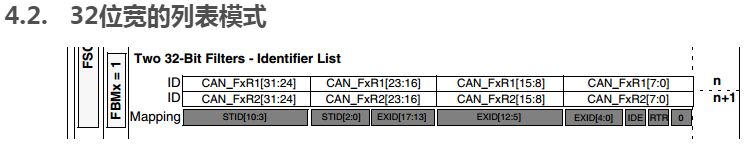
接受数据

在中断函数里面先清除中断标志位

主函数死循环什么都不用做。



CAN总线实现接收屏蔽滤波



修改CAN\_filter\_fig函数里面的屏蔽参数

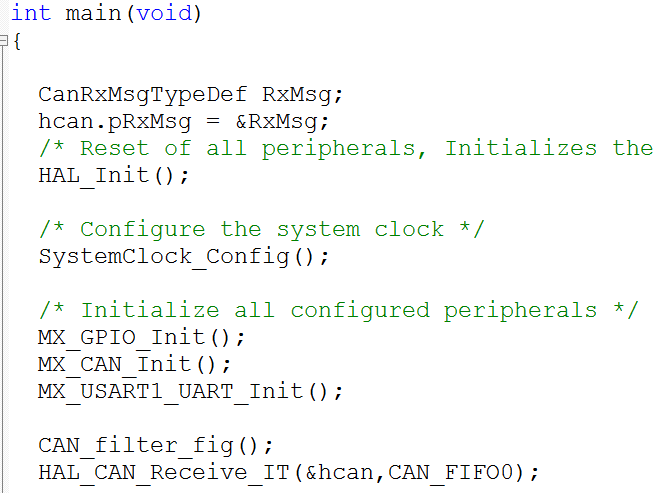


接受的数据放入fifo0

设置STM32允许接受的扩展ID

设置STM32允许接受的标识符ID

选择为标识符模式，就是电脑发送的ID和我CAN接受到ID一一对应的ID数据才允许接收进STM32芯片

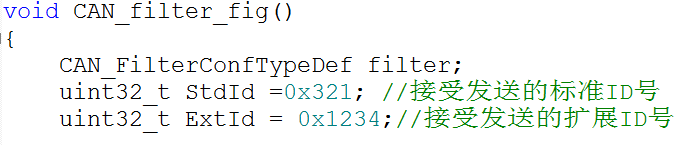


前面说了要用这个函数先打开CAN接受中断

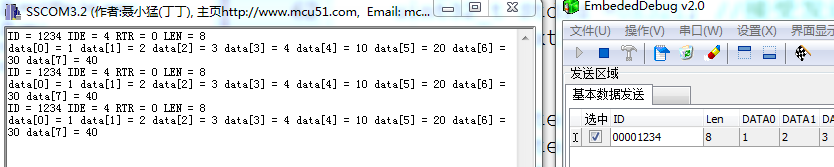
设置我要接受的ID

设置CAN接受数据的地址，和上面一样

中断函数和CAN总线接受实现章节将的一样。

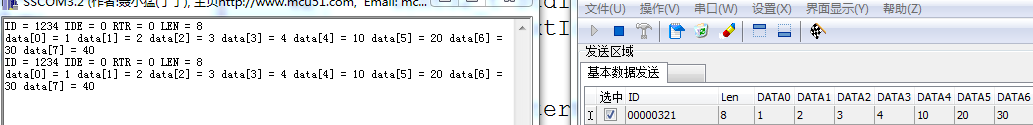


我写的扩展接受ID是1234



这里接受的也是1234，没有问题

这里是1234是因为我中断里面没有写读取标识符ID的函数



表示符ID接受的是321也是没有问题的。

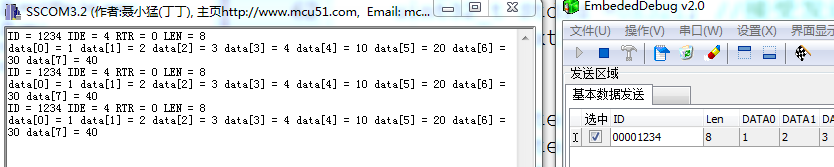
但是现在我不想我的STM32芯片除了接受扩展ID还要接受标识符ID。我只想接受扩展ID



STM32CAN就只判断扩展ID

我们只需要把标识符ID的代码屏蔽了

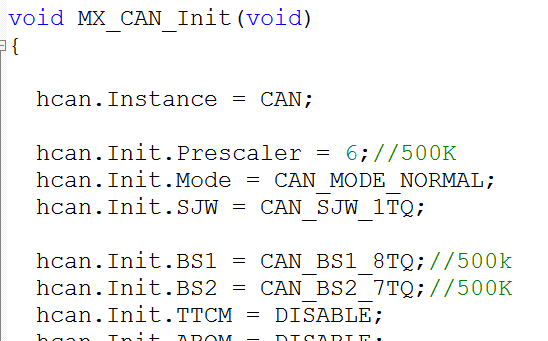
我们只需要把标识符ID的代码屏蔽了



**CAN波特率档位统计，1M，500K，125K**

修改CAM初始化函数里面的内容 MX\_CAN\_Init

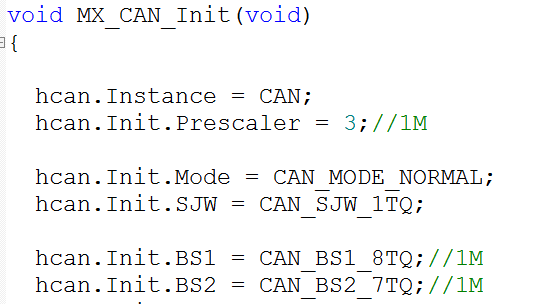
500k波特率



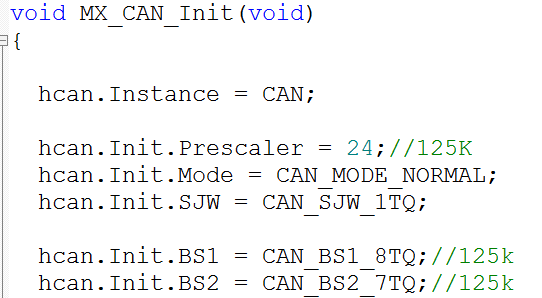
只需修改BS1,BS2

只需修改分频系数

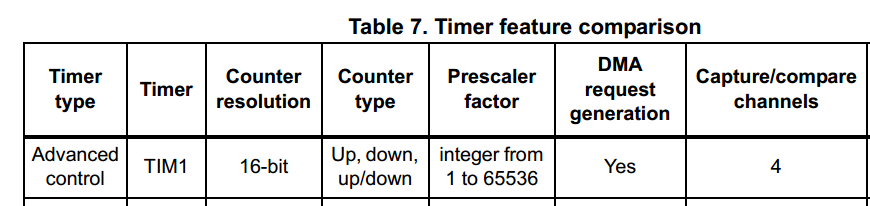
1M波特率



125K波特率



**TIM1定时器使用**



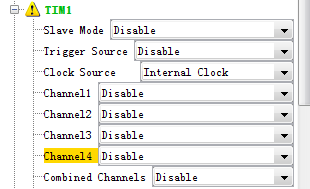
计数时间可以用主时钟分频，分频系数为1～65536

定时器产生4路PWM

Up向上计数从 0加到65535溢出

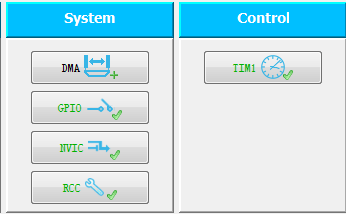
Down向下，从65535减到0溢出

定时器1是16位可以计数从0～65535溢出

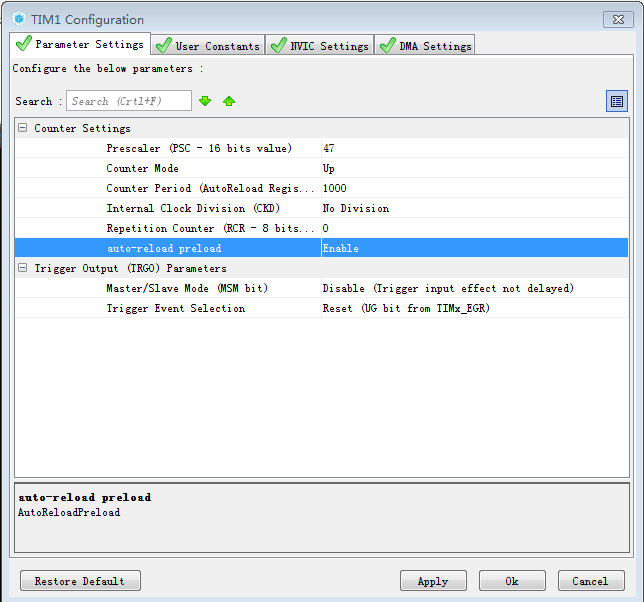


选择内部时钟48M





设置定时器



这里选择自动重载，就是加到1000溢出后，不用写代码去给定时器赋初值，软件自动去帮你赋初值

这里设置加到1000就产生中断

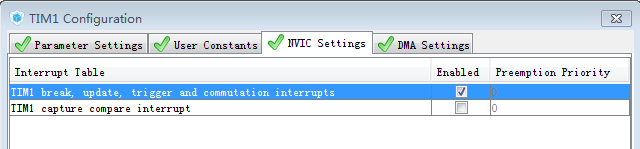
定时器每个时钟向上+1

我们要设置tim1定时器1ms(毫秒)产生一次中断

因为用的是内部时钟48M，那么就是20ns产生一次中断，太快了，所以要分频。

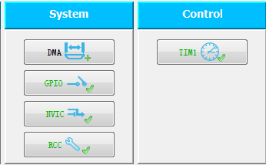
48000000/48=1M，那么就是1us产生一次中断

这里本应该写48分频的，因为软件内部要自动+1，所以写47，最后软件自动分频是48

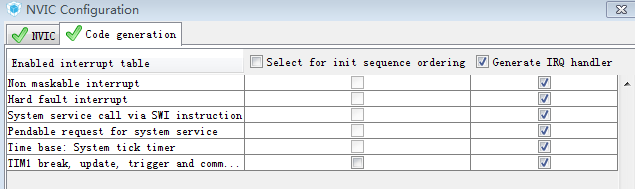
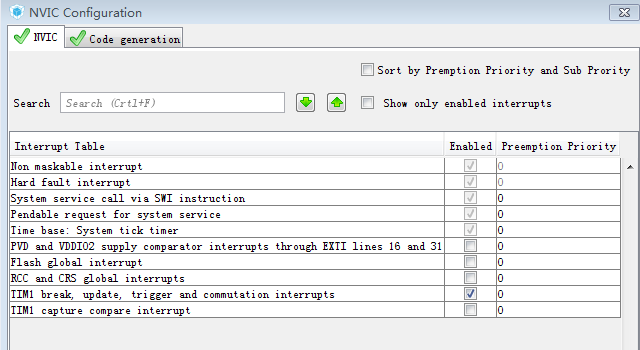


定时器溢出产生中断，进入中断子程序

点击确定

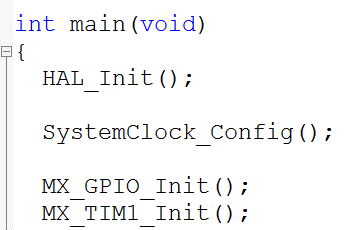


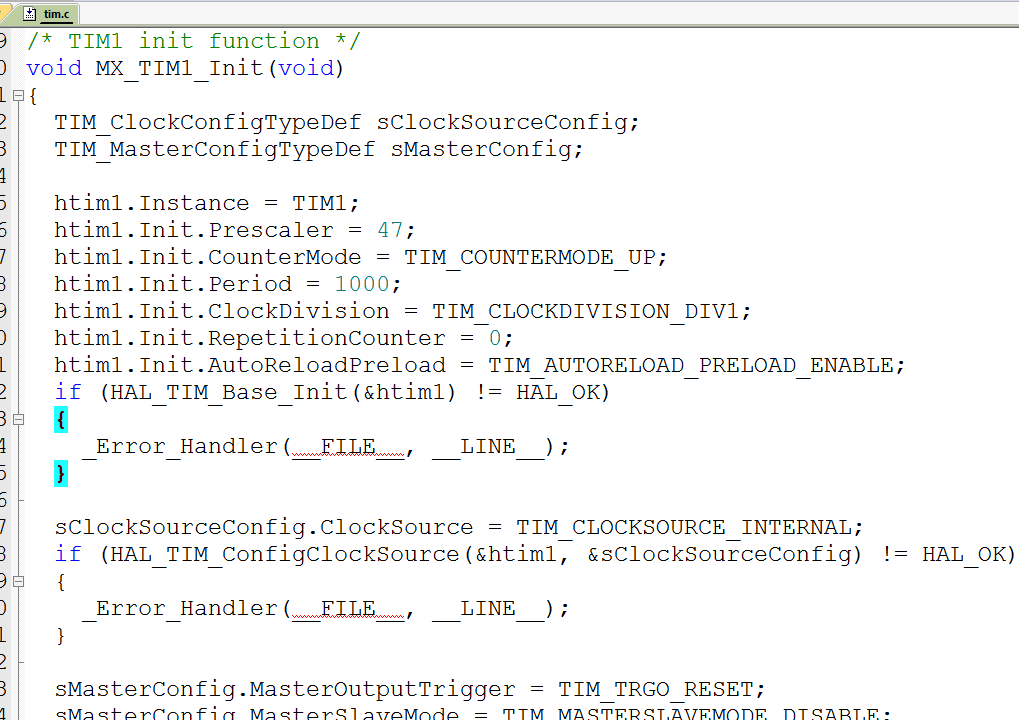
点击中断管理

点击OK生成代码

确认产生tim1中断服务函数，然后配置生成代码的选项

确认中断使能





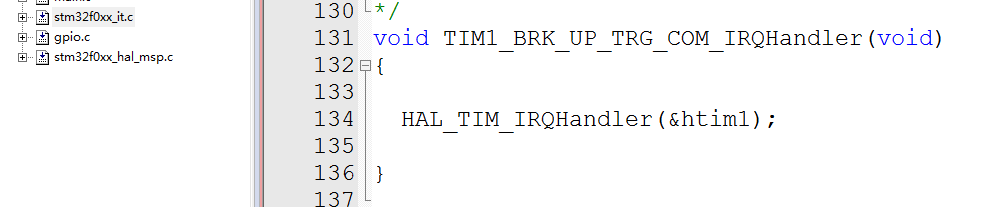
溢出后自动重装初值

计数加到1000溢出

向上计数

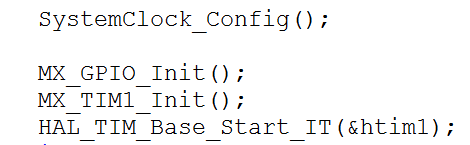
+1后分频48

使用tim1定时器



清除中断标志位

TIM1溢出后产生中断的入口函数



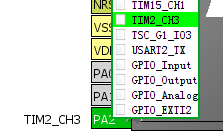
在主函数中一定要打开这个，tim1才开始工作

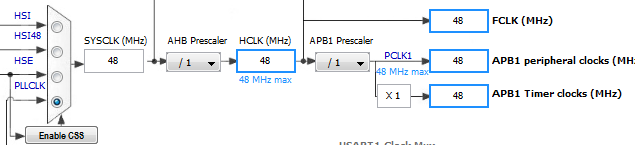
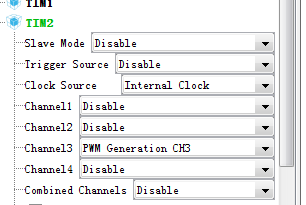
Tim1使用完毕，申请变量在中断函数里面自动+1，加到1000就是1秒

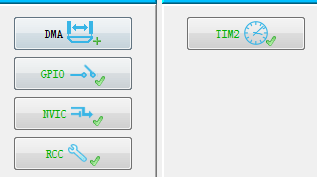
**用定时器2做PWM输出**

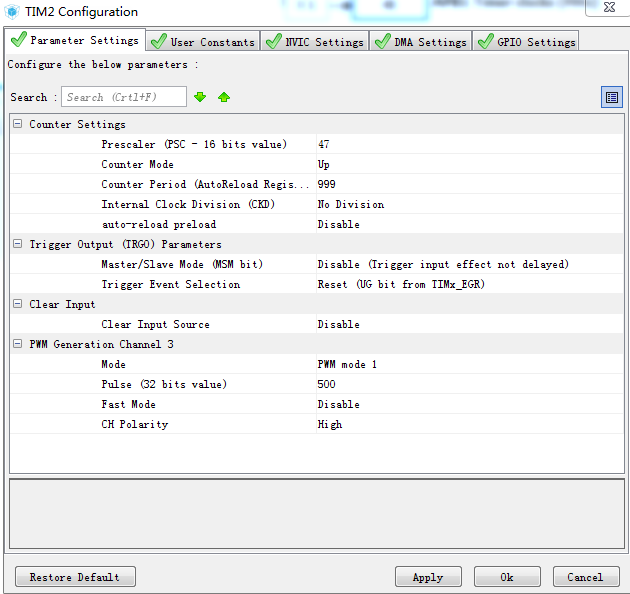
我们用TIM2定时器，CH3通道做PWM









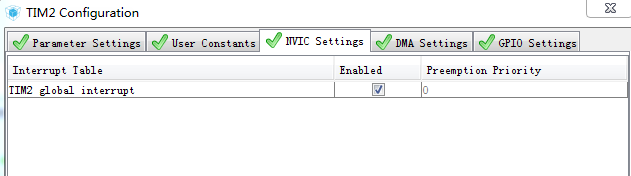


这里是PWM占空比为500，意思就是你上面设置的pwm周期是1ms，这里500就是0.5占空比，这个占空比是可以随时调节的

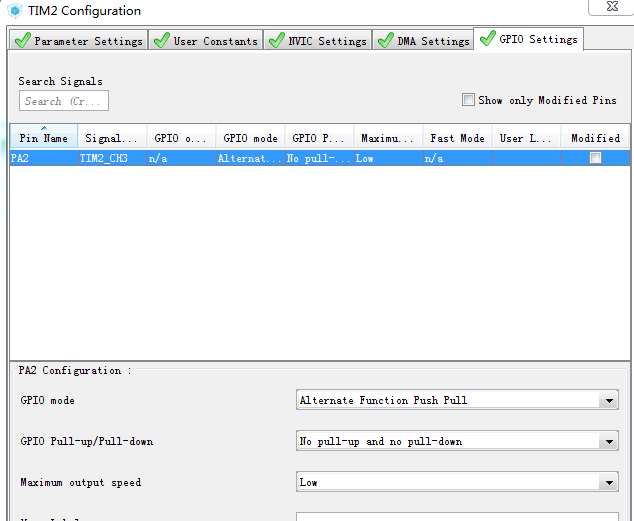
设置PWM模式1

这里是999+1，所以PWM周期是1ms

48M分频为1M，为什么不是48而是47，请看tim1定时器使用

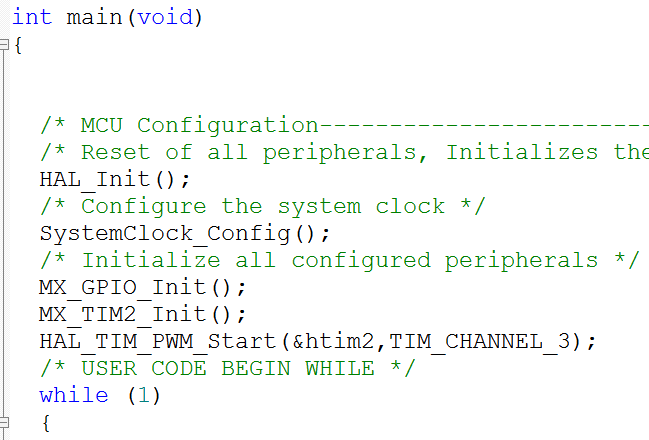


设置tim2产生中断

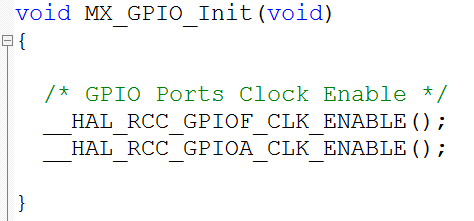
点击确定设置完毕。

GPIOA2引脚为浮空输出

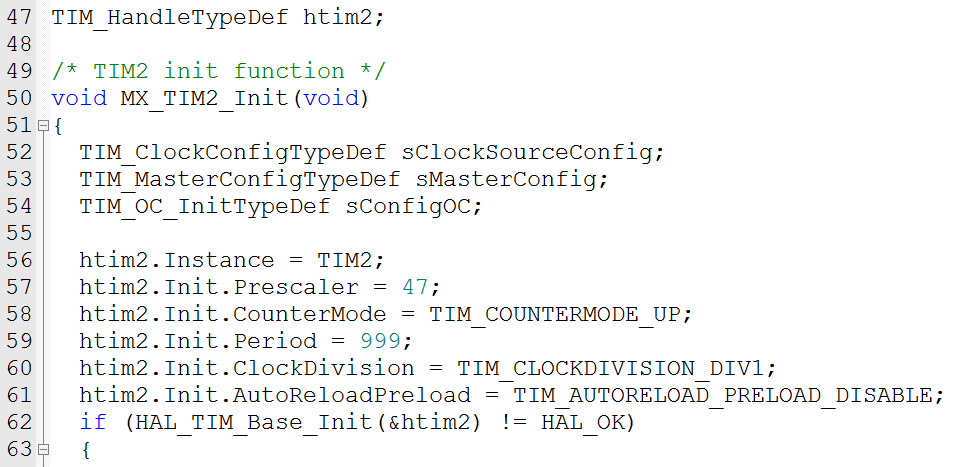
然后配置代码生成选项，生成代码。



这是开启TIM2,CH3通道PWM功能，这个函数不能少



这里只设置GPIO的时钟，GPIO的PWM引脚功能交由其他函数设置

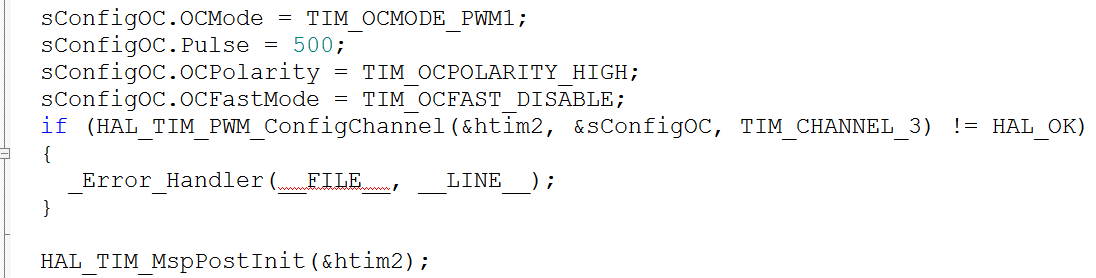


设置PWM频率为1K

向上计数

PWM使用的定时器

设置分频系数

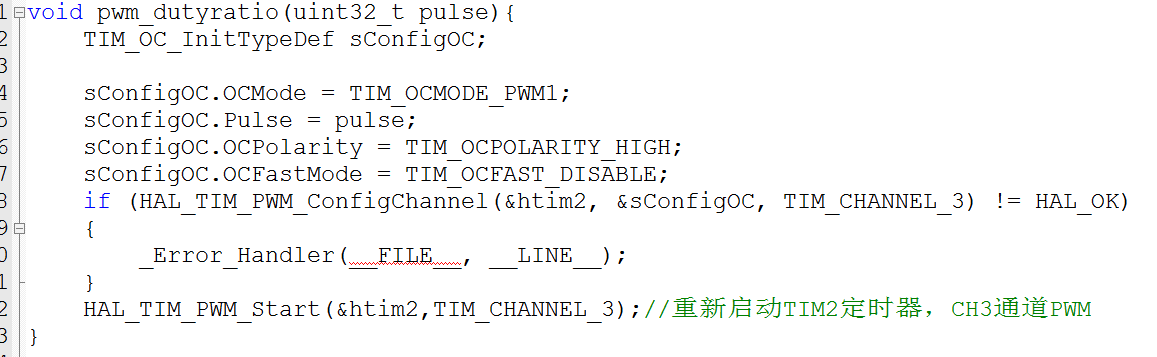


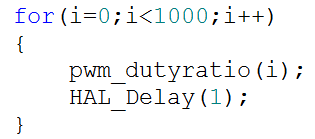
设置GPIO管脚复用功能

设置占空比

然后你的单片机就输出0.5占空比的PWM波形了，浮空输出也是输出的0～3.3V

如果你要随时调节PWM的占空比



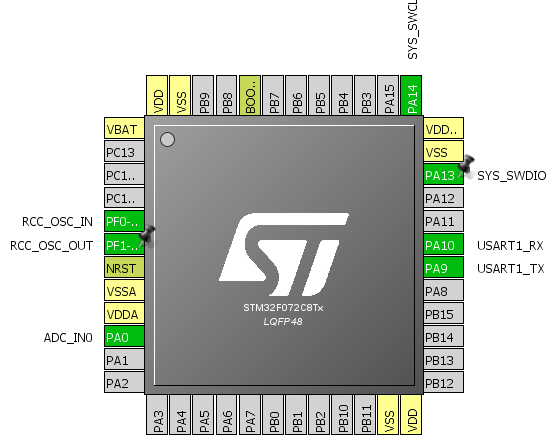


这样就可以了

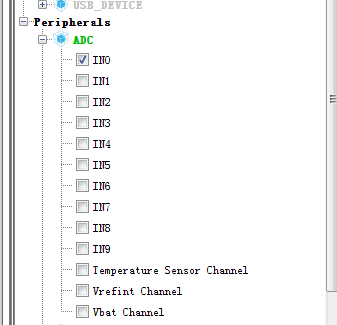
**ADC单通道轮询问电压采集**



我们采用STM32F072的PA0引脚来采集电源电压，记得在该引脚并联一个1uf的电容，这样采集的电压数据更稳定，跳变在个位数。

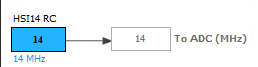


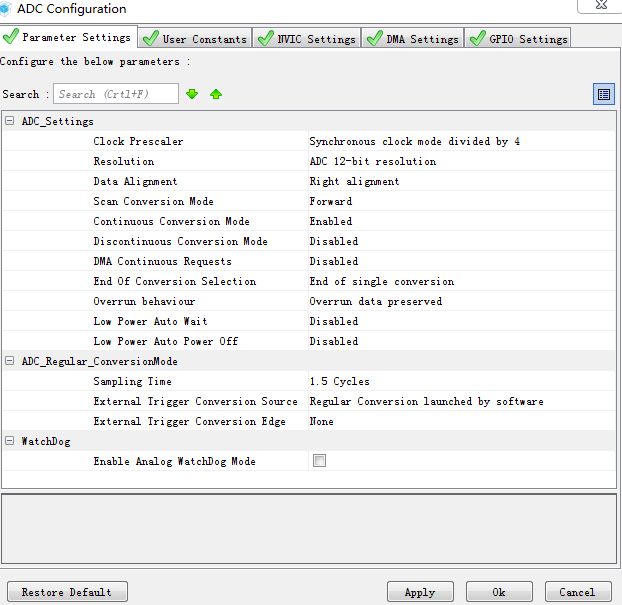
因为用的PA0引脚所以我们选择PA0来做ADC



设置ADC参数

PA0是ADC通道0，这里ADC通道0就是IN0，所以我们选择上

ADC时钟最大14M



关闭间断转换模式

打开连续转换模式

采集的数字量数据右对齐

ADC分辨率选择12位

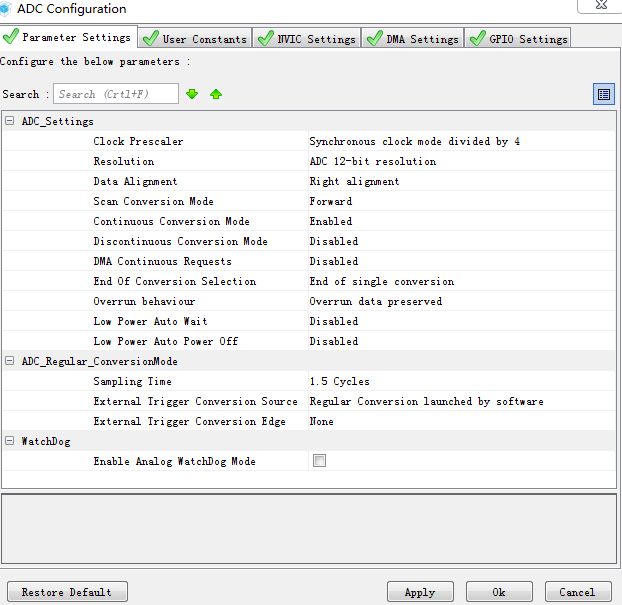
ADC时钟不能大于14M，所以这里选择分频

Forward 扫描通道从第一个通道到最后一个通道

Backward 扫描通道从最后一个通道到第一个通道

要求进行一次ADC转换：配置为单次模式使能，扫描模式失能。这样ADC的这个通道，转换一次后，就停止转换。

要求进行连续ADC转换：配置为连续模式使能，扫描模式失能。这样ADC的这个通道，转换一次后，接着进行下一次转换，不断连续



ADC采样周期为1.5个采样周期

定时器触发转换？还是比较触发转换？还是自己写代码触发转换？

我选择代码写代码触发

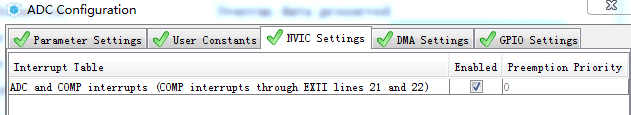
外部边沿触发ADC转换，关闭

低电压ADC自动关闭，设置为关闭

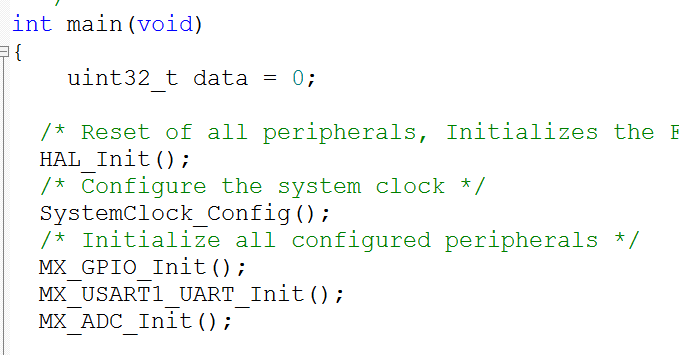
低电压自动等待关闭

单通道转换结束

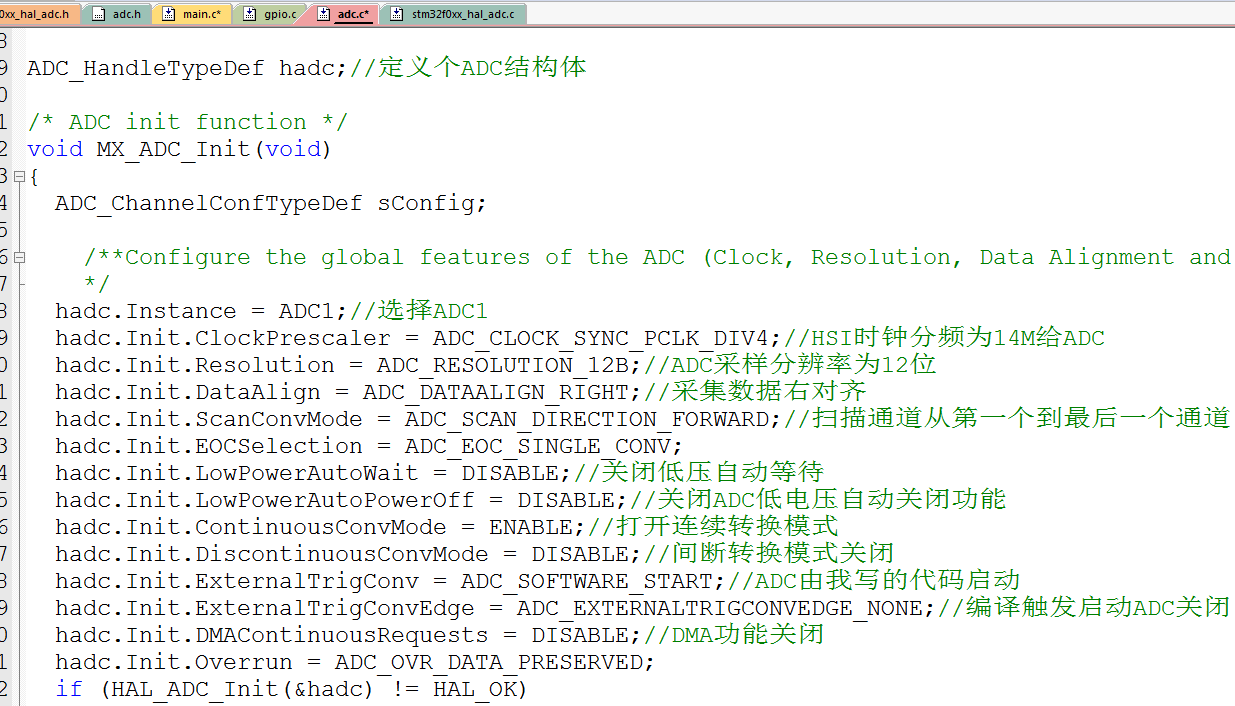
关闭DMA

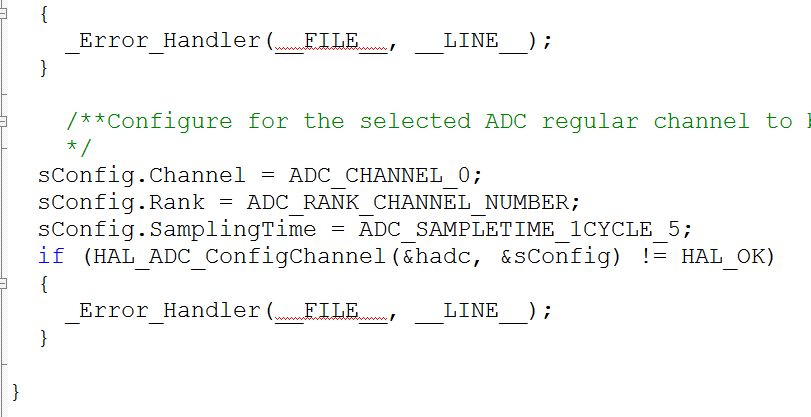


打开ADC中断，其实我软件触发不知道打开ADC中断有什么用？



ADC初始代码

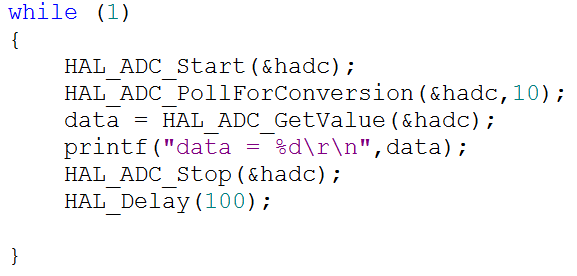




ADC采样周期为1.5个采样周期

ADC通道0，也就是PA0

下面是主函数死循环启动ADC



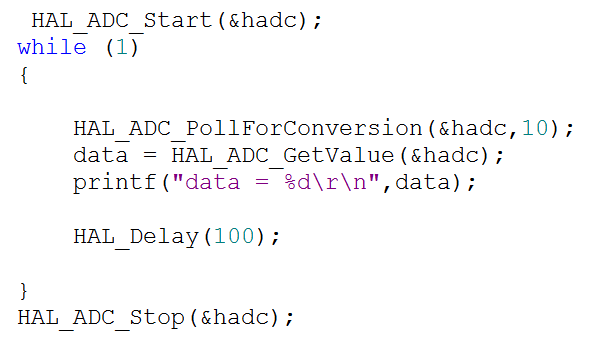
关闭ADC转换

获取ADC转换后的数据

不知道ADC转换时间是多久，我这里等待10毫秒

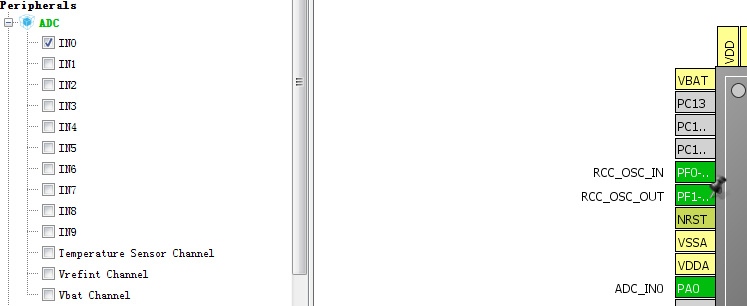
启动ADC转换

因为是连续转换，所以启动ADC之后可以不关闭



每次在这里等待AD采样结束就是

**ADC单通道DMA循环采集**

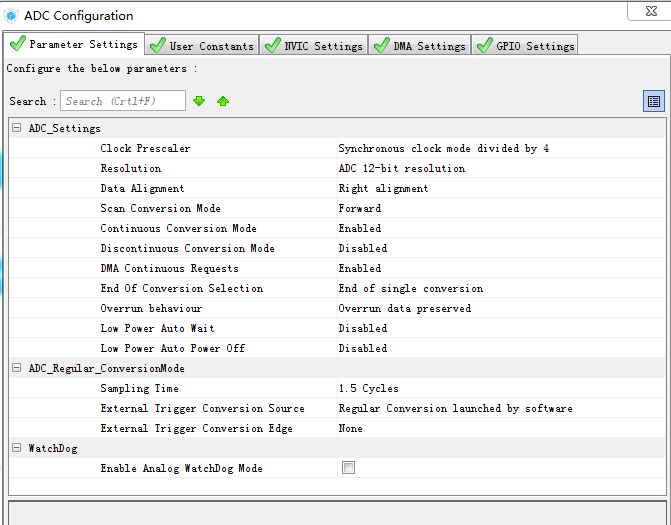


这里选择通道



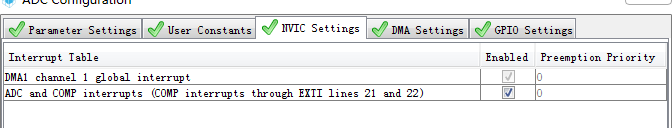
主要修改这里

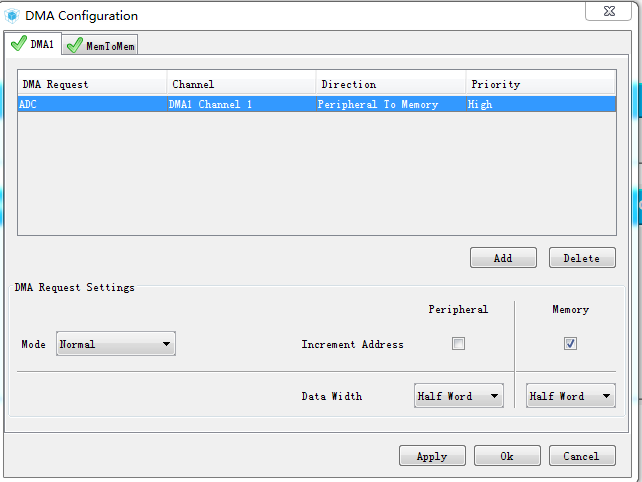
主要修改这里



确认DMA中断打开，这里默认打开

按照前面ADC轮询章节把基本功能选择好，这里只需要增加DMA使能选择就行了





因为ADC是12位的，所以选择半个字，也就是16位承载

确认DMA数据放在内存

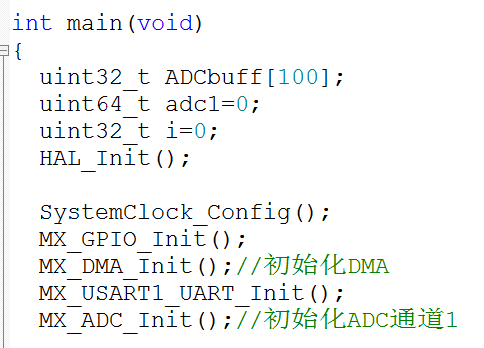
4.DMA采用高优先级

3.DMA采集的ADC数据放在内存

2.根据ADC通道，对应DMA通道1

1.添加ADC的DMA通道

设置选项生成代码

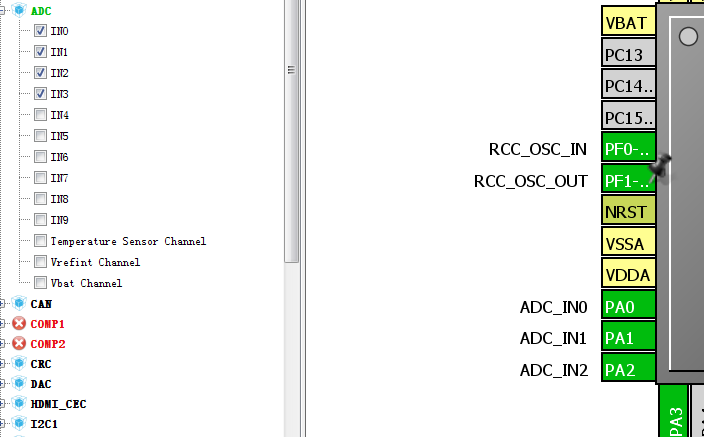


单通道DMA采集这里一定要写采集100次

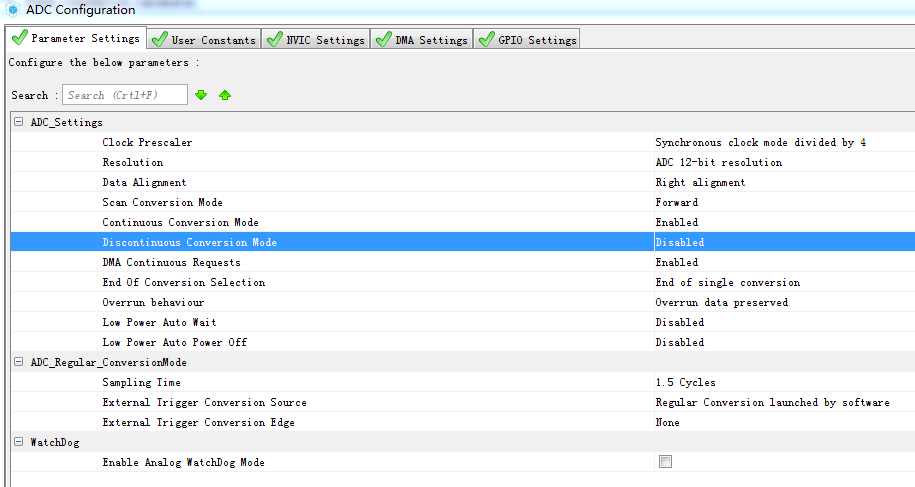


代码使用例程

**ADC多通道轮询采集**



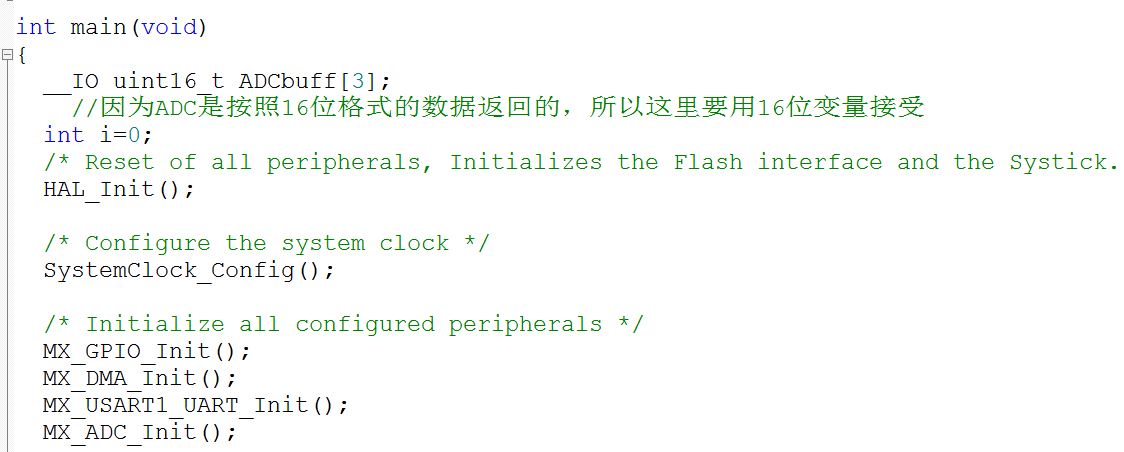
打开ADC的 PA0/PA1/PA2/PA3/四个通道

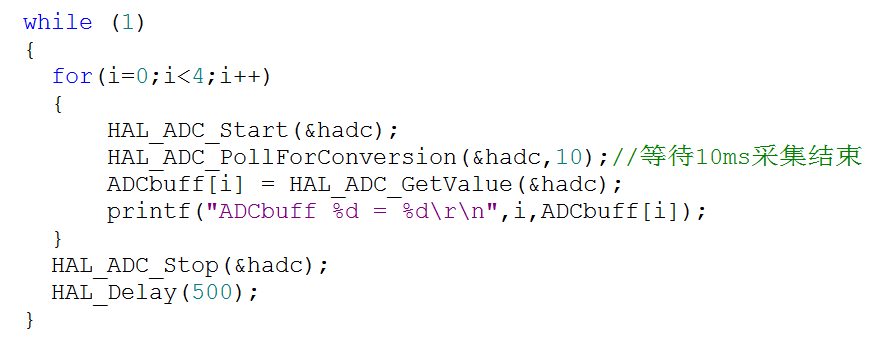


配置保持ADC单通道轮询的设置，这里加入了DMA，但是没有用到DMA

然后生成代码

这里如果不设置16位变量接受数据，而是用32位，或者8位，那么数据就会乱，出现最大最小值





必须等待4个ADC通道采集完，也就是start四次，循环4次之后才能执行stop

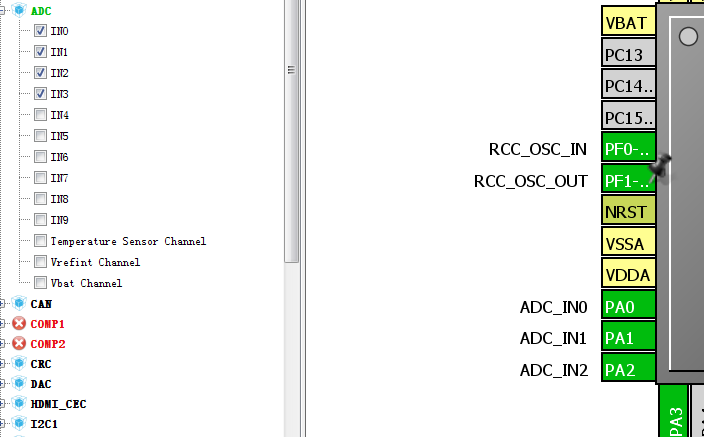
因为ADC是start一次通道向上增加，所以我这里是要采集四个ADC通道的数据，所以不能start一次就用stop关闭ADC

这里轮询记住，每次start一次ADC通道就会向前增加，所以注意

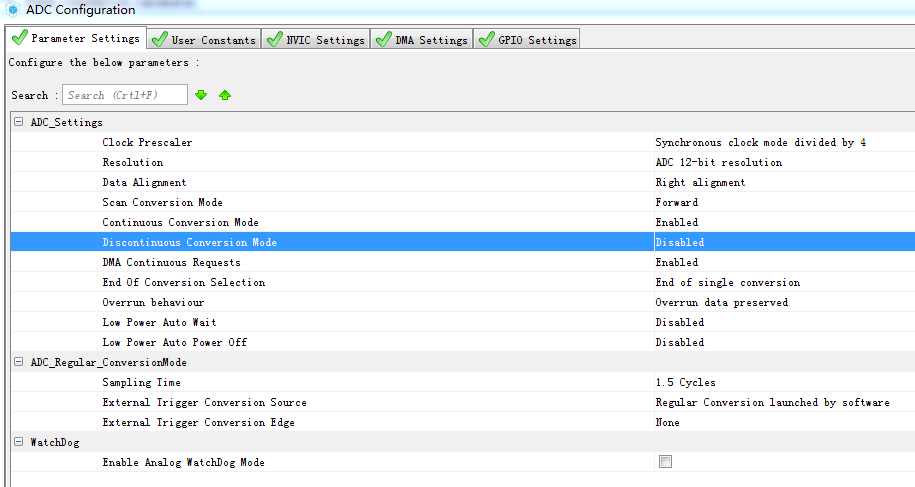
所以数组里面存放的是4个通道ADC采集的值

所以轮询模式下你要获取ADC某个通道的值只有，先四个通道轮询完，然后获取数组对应ADC通道的值。

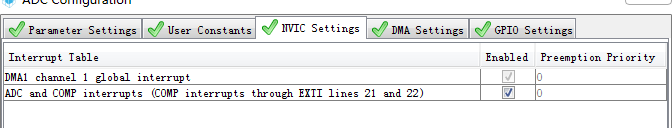
**ADC多通道DMA采集**



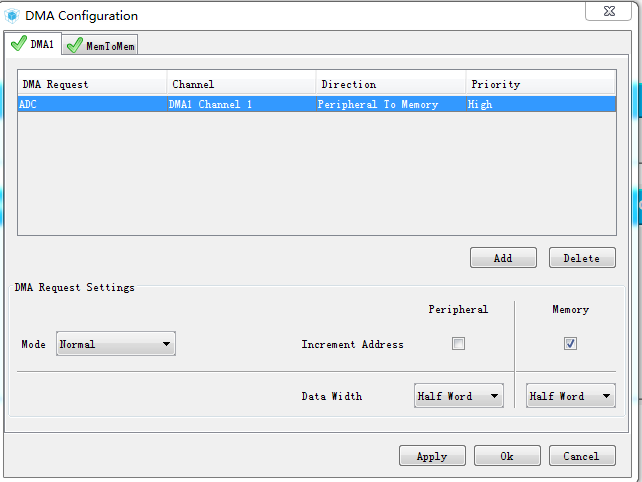
四个通道打开

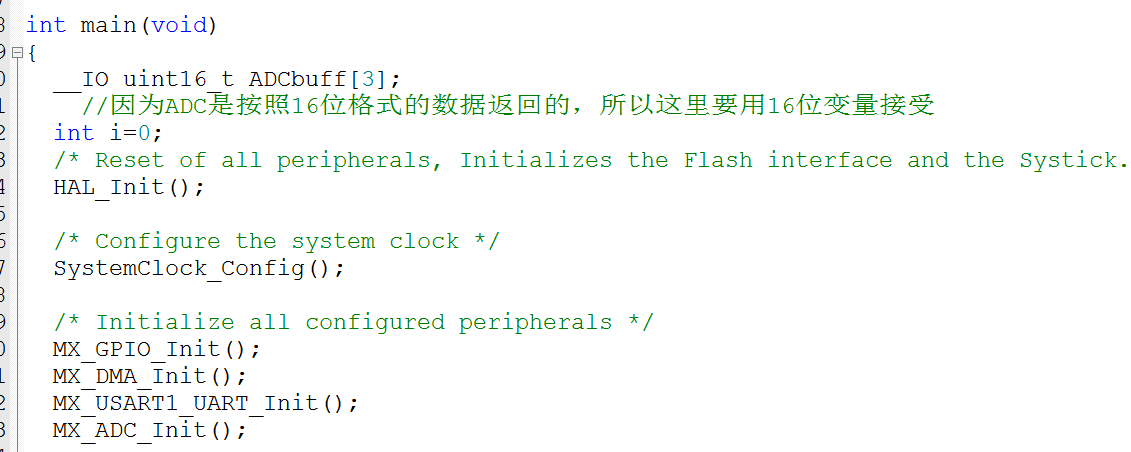


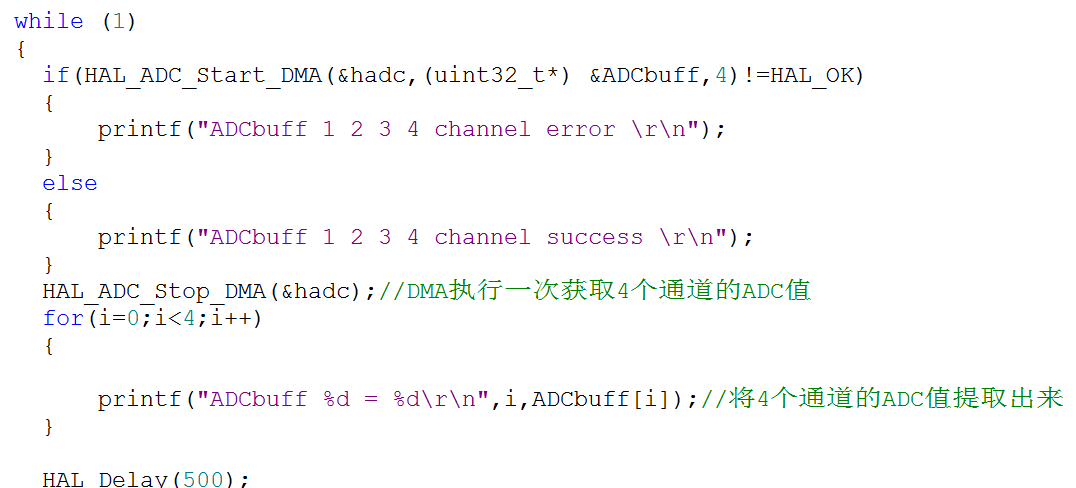
打开DMA



确认勾选上

设置ADC对应的DMA通道





这就是DMA模式获取ADC多通道值的方法