STM32F1外设操作

**作者:向仔州**

Stm32 I2C固件库操作，有BUG……………………………………………………………………..1

Stm32f105双CAN同时使用，有CAN1,CAN2………………………………………………..5

ADC使用………………………………………………………………………………………………………….10

**STM32 I2C固件库操作**

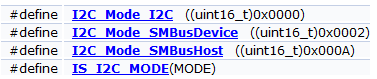
typedef struct

{

uint32\_t I2C\_ClockSpeed; //设置 I2C SCL时钟频率，此值要低于 400000比如我要400K的速率我就写400000

uint16\_t I2C\_Mode;

uint16\_t I2C\_DutyCycle;



通常都是选择I2C\_Mode\_I2C

uint16\_t I2C\_OwnAddress1;

uint16\_t I2C\_Ack;

uint16\_t I2C\_AcknowledgedAddress;

}[I2C\_InitTypeDef](mk:@MSITStore:E:\\Desktop\\【固件库】STM32F10x_StdPeriph_Lib_V3.5.0\\stm32f10x_stdperiph_lib_um.chm::/structI2C__InitTypeDef.html" \o "I2C Init structure definition.);

指定STM32芯片自己的I2C地址，STM32做从机可以用



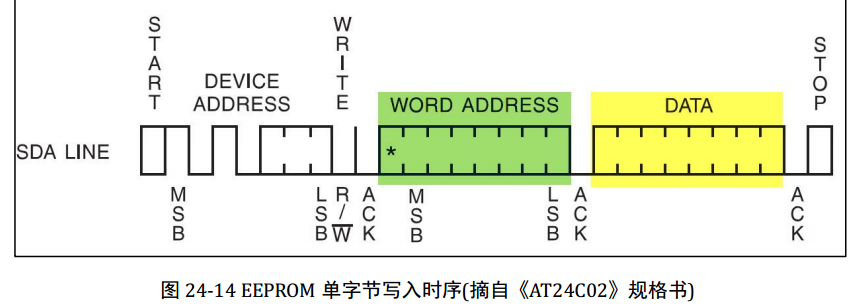
16\_9或者DutyCycle\_2都可以选

Ack在初始化的时候要配置I2C为允许应答I2C\_Ack\_Enable

根据从机地址决定主机I2C是发送7位地址还是10位地址

记住如果我STM32做从机的话， I2C\_OwmAddress1变量支持10位地址，I2C\_OwmAddress2只支持7位模式。

配置完I2C模式后下面我们来看看I2C读写外设的函数介绍



I2C\_GenerateSTART(I2C\_TypeDef\* I2Cx, FunctionalState NewState);//I2C产生start开始信号

I2C\_TypeDef\* I2Cx : 写I2C1还是I2C2(就是I2C控制器选择)

FunctionalState NewState: 写ENABLE或者DISABLE

I2C\_GenerateSTOP(I2C\_TypeDef\* I2Cx, FunctionalState NewState);//I2C主机产生stop信号

I2C\_TypeDef\* I2Cx : 写I2C1还是I2C2(就是I2C控制器选择)

FunctionalState NewState: 写ENABLE或者DISABLE

I2C\_Send7bitAddress(I2C\_TypeDef\* I2Cx, uint8\_t Address, uint8\_t I2C\_Direction);

//I2C主机发送或者接受器件的I2C地址

I2C\_TypeDef\* I2Cx : 写I2C1还是I2C2(就是I2C控制器选择)

Address: 写从机I2C地址

I2C\_Direction: I2C\_Direction\_Transmitter (STM32作为发送器)/

I2C\_Direction\_Receiver (STM32作为接受器)

I2C\_SendData(I2C\_TypeDef\* I2Cx, uint8\_t Data);//I2C主机发送数据给从机

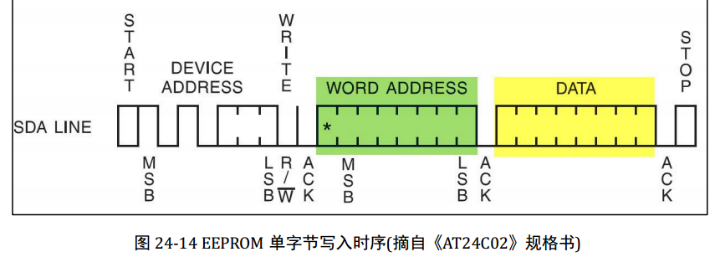
I2C\_TypeDef\* I2Cx : 写I2C1还是I2C2(就是I2C控制器选择)

Data:就是8位数据

uint8\_t I2C\_ReceiveData(I2C\_TypeDef\* I2Cx)

I2C\_TypeDef\* I2Cx : 写I2C1还是I2C2(就是I2C控制器选择)

uint8\_t: 接受的数据返回值



I2C\_AcknowledgeConfig(I2C\_TypeDef\* I2Cx, FunctionalState NewState);

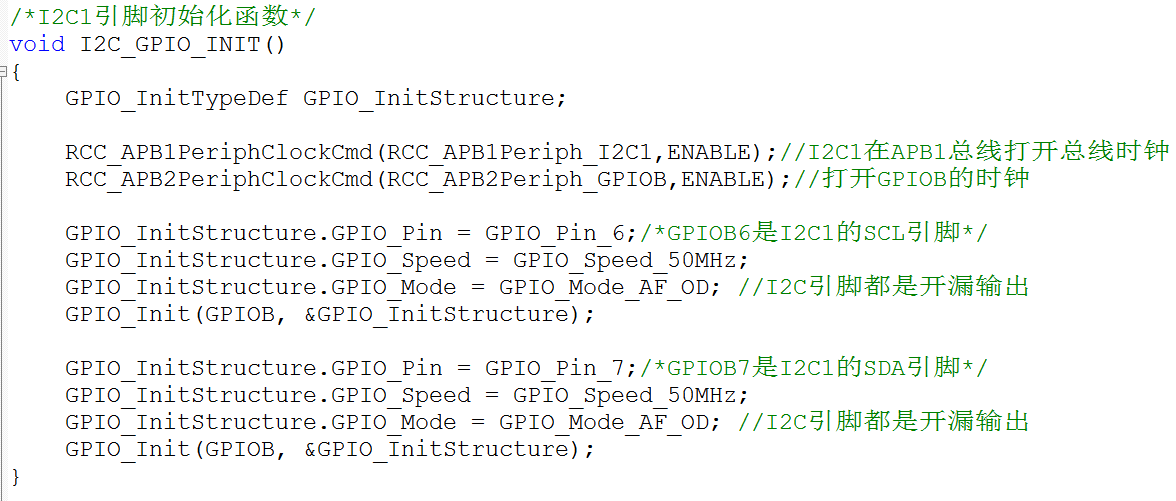
//当我们主机觉得接受从机的数据量差不多了，不想接受从机数据了，想结束这次通信，我们就给从机发送个NACK信号表示接受完成

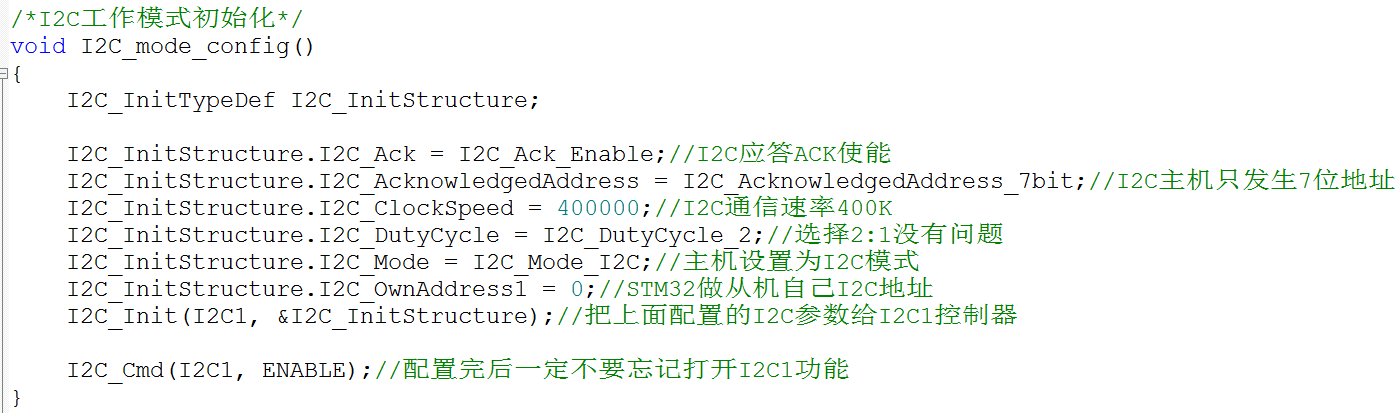
I2C\_TypeDef\* I2Cx : 写I2C1还是I2C2(就是I2C控制器选择)

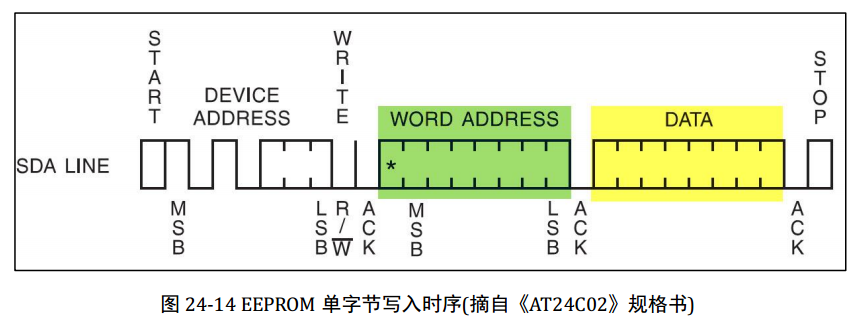
FunctionalState NewState: 写ENABLE(产生Nack)或者DISABLE(不产生Nack)

有些I2C从机器件不需要主机产生Nack

下面写个实例STM32 I2C读取EEPROM数据的程序

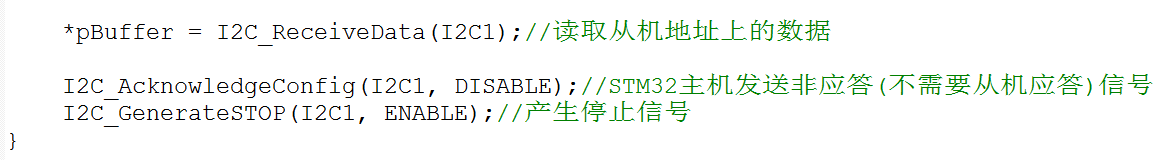


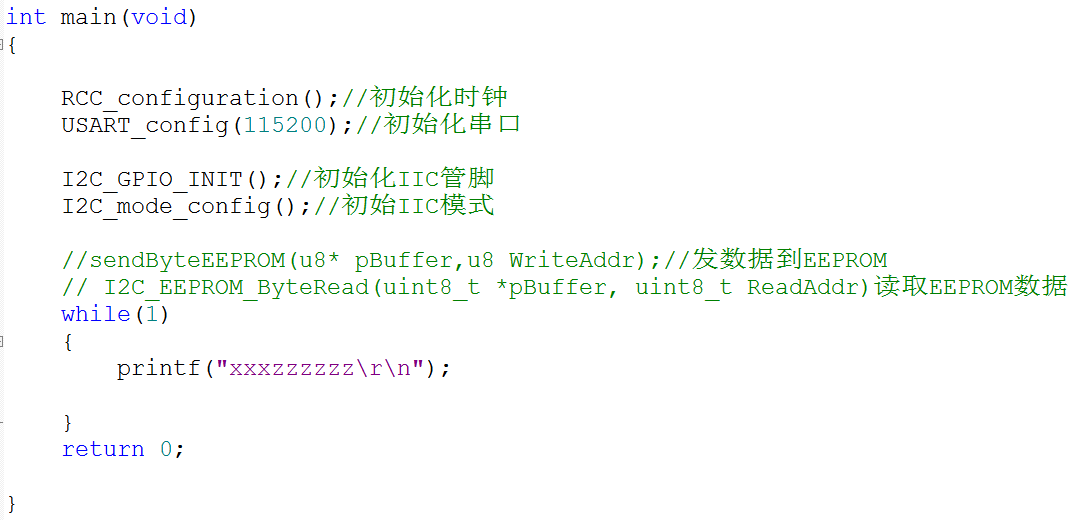












这就是STM32主机操作EEPROM的测试程序，但是没有上板子试过，库函数使用方法就是这样。

这里要注意，不只是I2C1控制器有问题，I2C2控制器在有其它程序中断的情况下也可能出问题。所以还是建议用模拟的I2C接口

**STM32双CAN同时使用，CAN1,CAN2**

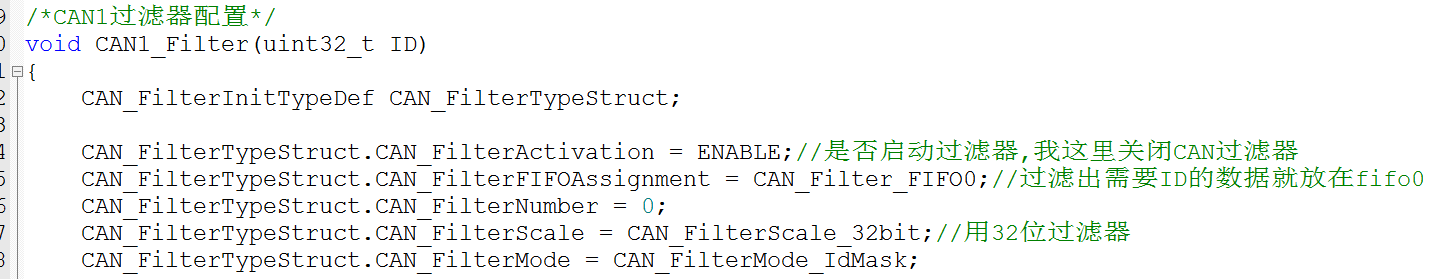
STM32F103是没有两个CAN的，F103只有CAN1。STM32F105才有CAN1和CAN2。

在移植STM32F103的CAN程序到STM32F105芯片上时要做一些变动。

STM32F103和STM32F105的CAN配置有80%是一模一样，但是还是有20%不一样。

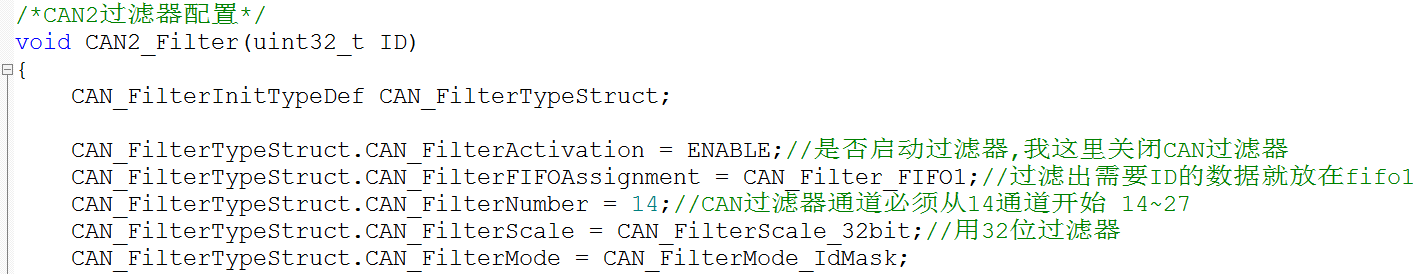
1过滤器差异

STM32F105芯片复位后默认使用0～13号过滤寄存器，过滤数据到CAN1的 FIFO1或者FIFO0



但是CAN2不能使用0～13过滤器

CAN2只能使用14～17号过滤器，过滤数据到FIFO0或者FIFO1



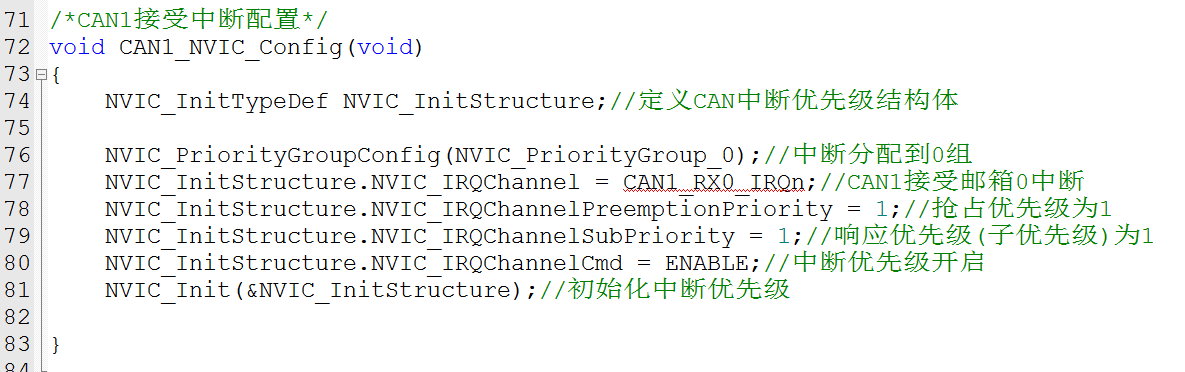
过滤器修改完成之后还要修改CAN的中断优先级的入口函数名

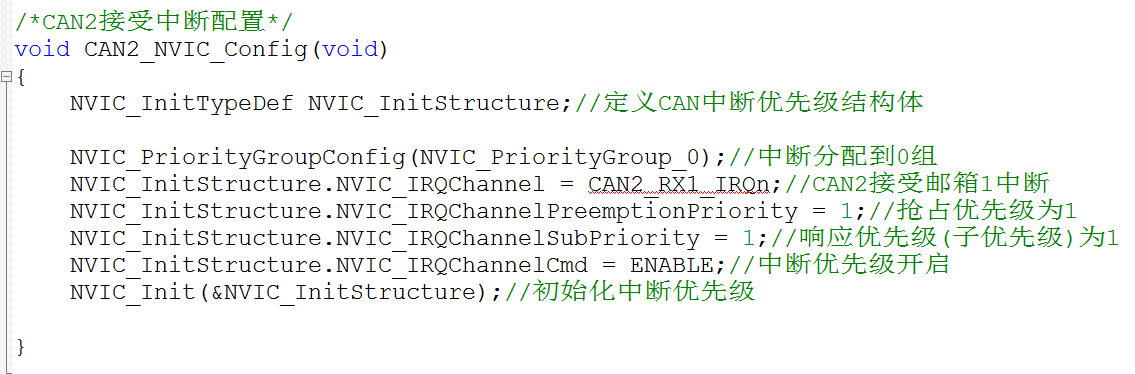
因为STM32F103只有1个CAN1,而且CAN1和USB是复用的，所以接受数据的中断函数入口名是USB\_LP\_CAN1\_RX0\_IRQn



但是STM32F105是双CAN，所以中断函数名发送了变化

STM32F105CAN1中断函数名是CAN1\_RX0\_IRQn，这个RX0是在初始化CAN1的时候设置的邮箱0来接受数据





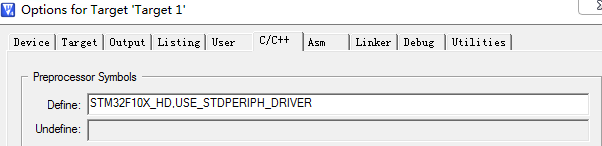
STM32F105CAN2中断函数名是CAN1\_RX1\_IRQn，这个RX1是在初始化CAN2的时候设置的邮箱1来接受数据

因为修改了中断函数名，导致编译不过！！

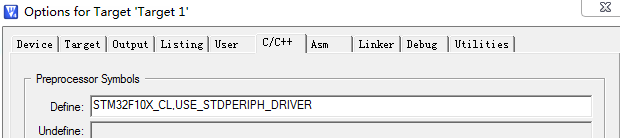
因为CAN1\_RX0\_IRQn和CAN2\_RX1\_IRQn是在stm32f10x.h文件里面的STM32F10X\_CL宏下面定义的，但是添加这个宏就要修改启动文件。

将f103的hd.s文件改成cl.s文件

 这个cl.s文件在keil创建工程的时候就有了，只是我们没有要，现在用STM32F105就需要使用了。



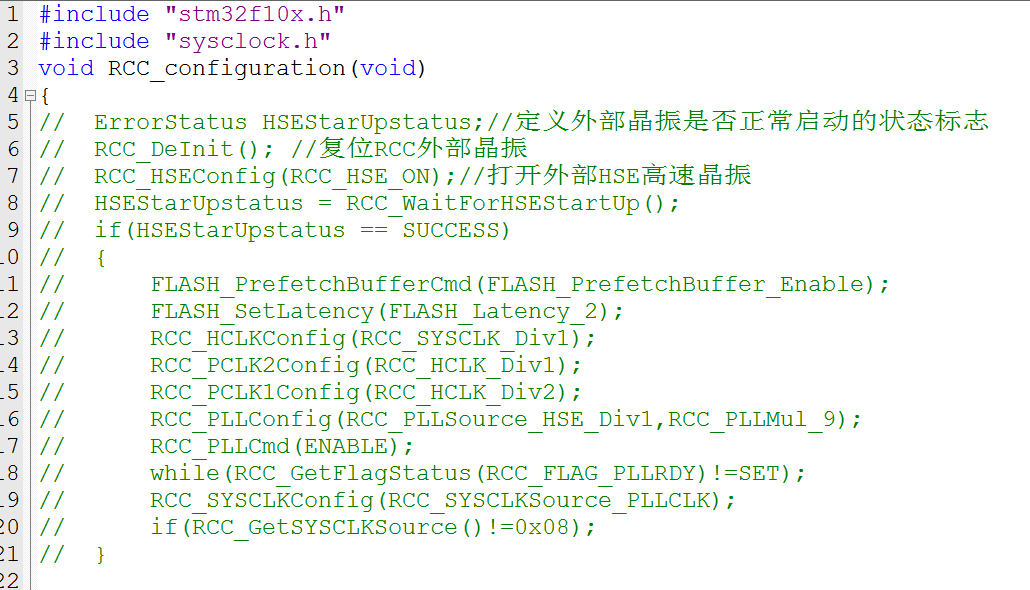
将f103的编译宏STM32F10X\_HD改为STM32F10X\_CL



这样就可以成功编译CAN1\_RX0\_IRQn和CAN2\_RX1\_IRQn了

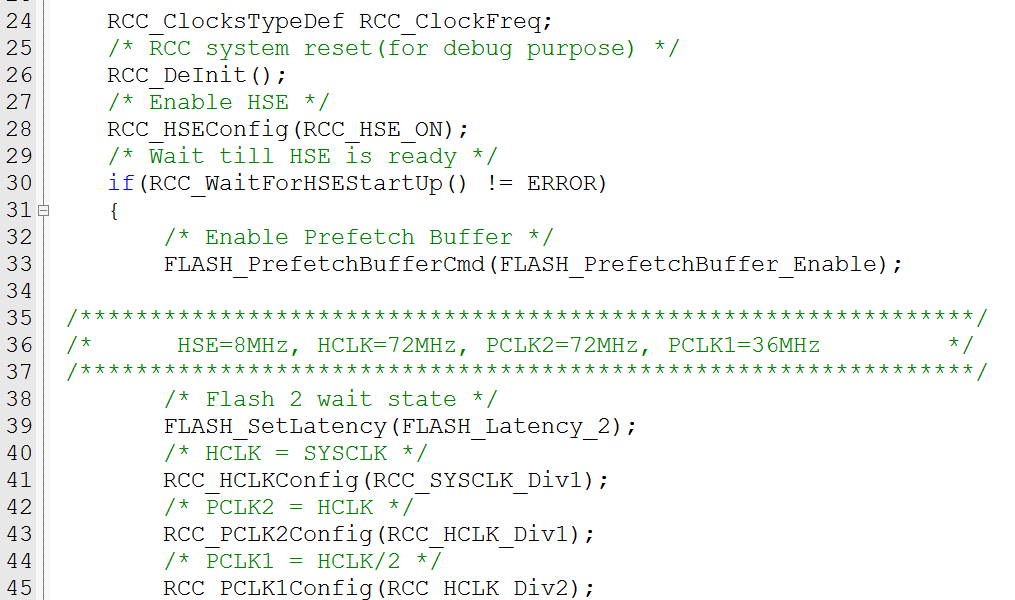
但是编译过程中发现RCC初始化报错

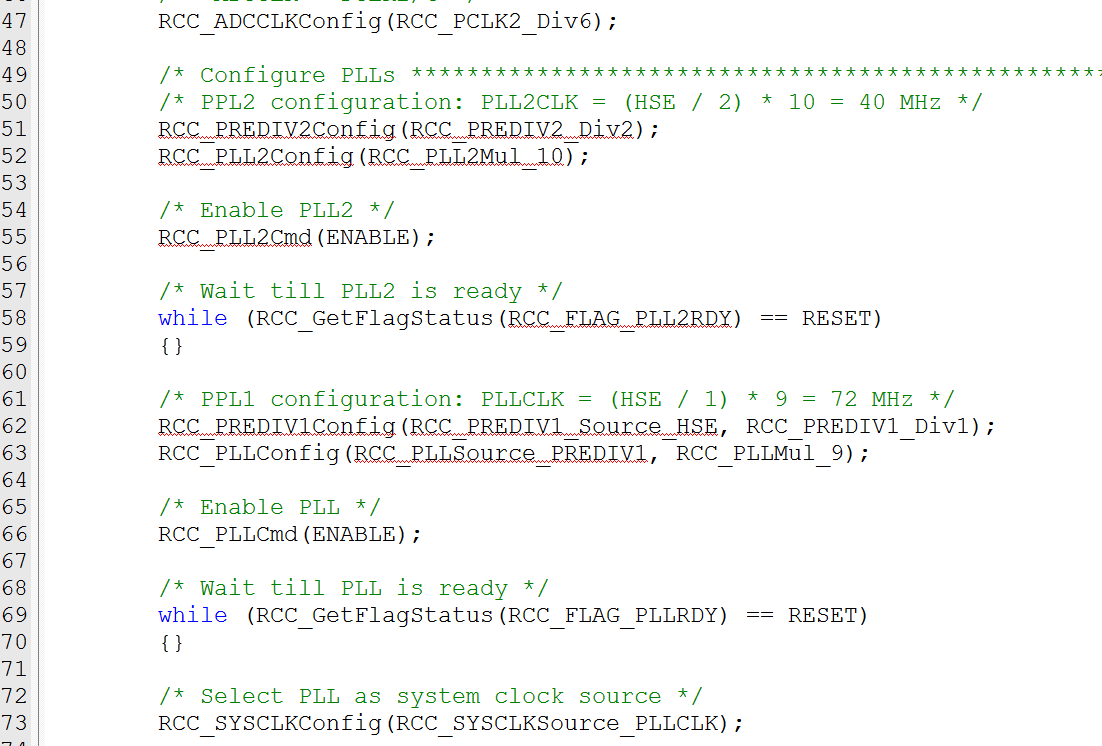
这是因为STM32F105的RCC初始化方式和F103不一样

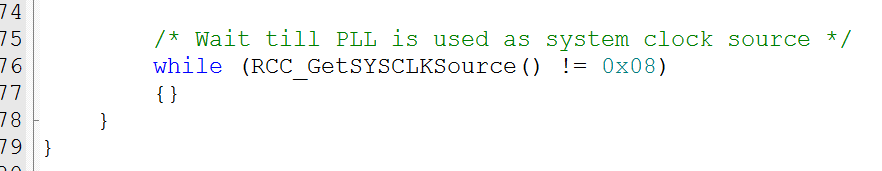


修改为F105的RCC初始化程序

取消掉F103的RCC初始化程序







下面是RCC初始化代码可以复制片段

RCC\_ClocksTypeDef RCC\_ClockFreq;

/\* RCC system reset(for debug purpose) \*/

RCC\_DeInit();

/\* Enable HSE \*/

RCC\_HSEConfig(RCC\_HSE\_ON);

/\* Wait till HSE is ready \*/

if(RCC\_WaitForHSEStartUp() != ERROR)

{

/\* Enable Prefetch Buffer \*/

FLASH\_PrefetchBufferCmd(FLASH\_PrefetchBuffer\_Enable);

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

/\* HSE=8MHz, HCLK=72MHz, PCLK2=72MHz, PCLK1=36MHz \*/

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

/\* Flash 2 wait state \*/

FLASH\_SetLatency(FLASH\_Latency\_2);

/\* HCLK = SYSCLK \*/

RCC\_HCLKConfig(RCC\_SYSCLK\_Div1);

/\* PCLK2 = HCLK \*/

RCC\_PCLK2Config(RCC\_HCLK\_Div1);

/\* PCLK1 = HCLK/2 \*/

RCC\_PCLK1Config(RCC\_HCLK\_Div2);

/\* ADCCLK = PCLK2/4 \*/

RCC\_ADCCLKConfig(RCC\_PCLK2\_Div6);

/\*\*\*\*\*ConfigurePLLs \*\*\*\*\*\*/

/\* PPL2 configuration: PLL2CLK = (HSE / 2) \* 10 = 40 MHz \*/

RCC\_PREDIV2Config(RCC\_PREDIV2\_Div2);

RCC\_PLL2Config(RCC\_PLL2Mul\_10);

/\* Enable PLL2 \*/

RCC\_PLL2Cmd(ENABLE);

/\* Wait till PLL2 is ready \*/

while (RCC\_GetFlagStatus(RCC\_FLAG\_PLL2RDY) == RESET)

{}

/\* PPL1 configuration: PLLCLK = (HSE / 1) \* 9 = 72 MHz \*/

RCC\_PREDIV1Config(RCC\_PREDIV1\_Source\_HSE, RCC\_PREDIV1\_Div1);

RCC\_PLLConfig(RCC\_PLLSource\_PREDIV1, RCC\_PLLMul\_9);

/\* Enable PLL \*/

RCC\_PLLCmd(ENABLE);

/\* Wait till PLL is ready \*/

while (RCC\_GetFlagStatus(RCC\_FLAG\_PLLRDY) == RESET)

{}

/\* Select PLL as system clock source \*/

RCC\_SYSCLKConfig(RCC\_SYSCLKSource\_PLLCLK);

/\* Wait till PLL is used as system clock source \*/

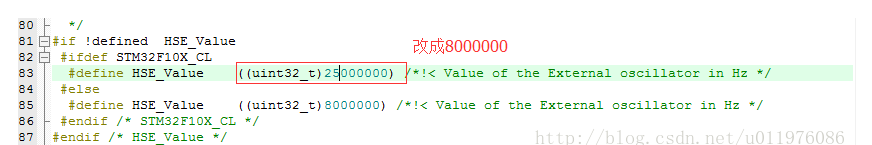
while (RCC\_GetSYSCLKSource() != 0x08)

{}

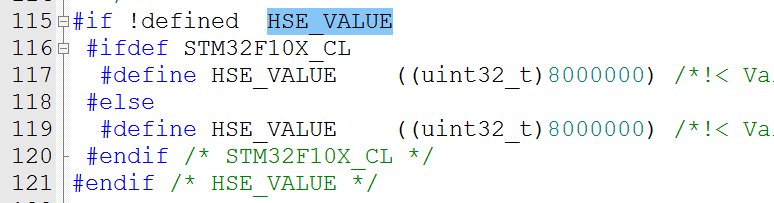
}

编译完全通过，但是串口输出数据不正常，CAN时钟也没有反应，这是因为外部时钟没有设置对。

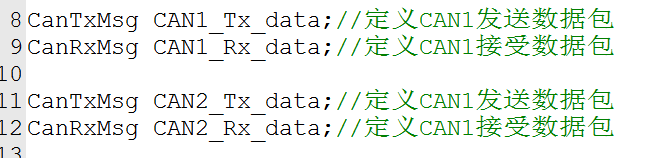
打开STM32F10x.h头文件去修改HSE\_VALUE宏



官方为了兼容STM32F105 USBOTG和STM32F107互联网型的网络功能，要求外部晶振必须配25M。

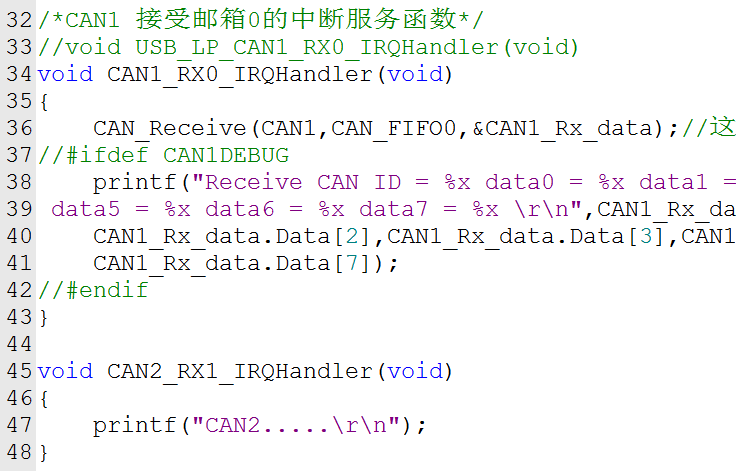


我们用的是外部8M晶振，所以这里改成8就是了，这个宏会自动拿去给RCC初始化运算。

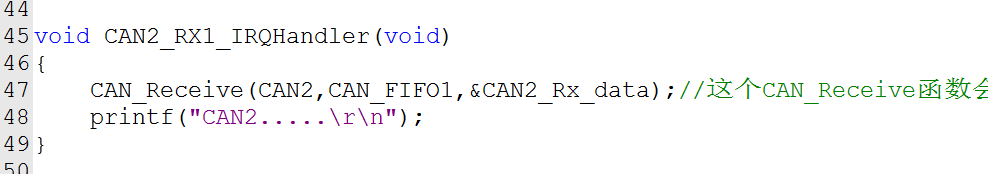


定义CAN1和CAN2的发送数据变量和接受数据变量

实现CAN1和CAN2接受数据中断



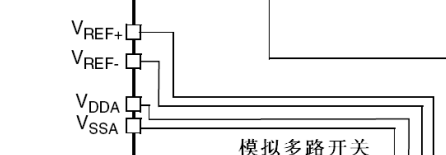
这里会无限打印CAN2… 因为没有执行CAN\_Receive清楚CAN总线中断标志位



这些CAN1和CAN2就驱动起来了

**ADC使用**

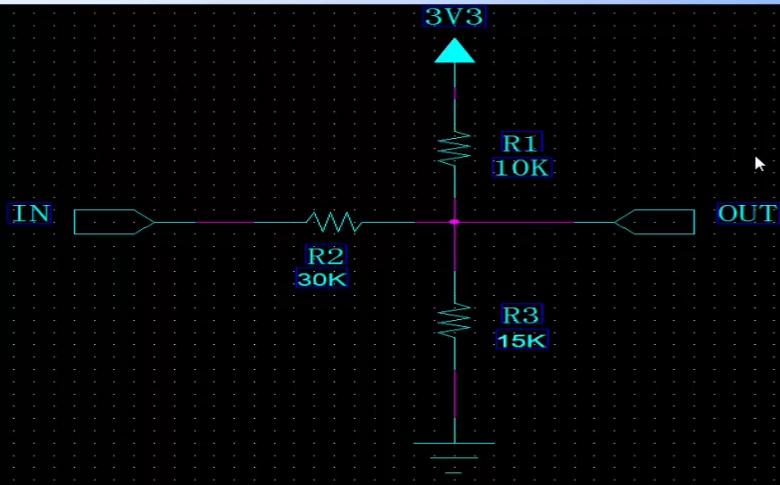
ADC电源供电部分



意思就是VREF-是ADC采集的最小电压，VREF+是ADC采集的最大电压,所以这个VREF电压是自己设置的。

如果VREF+接3.3V，VREF-接0V，那么ADC电压采集范围就是0～3.3V

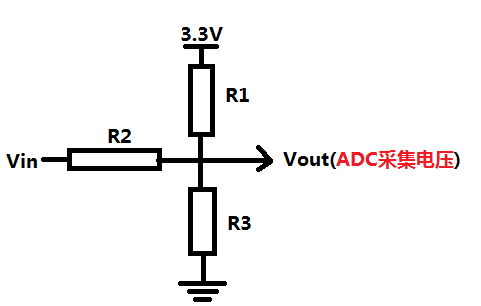
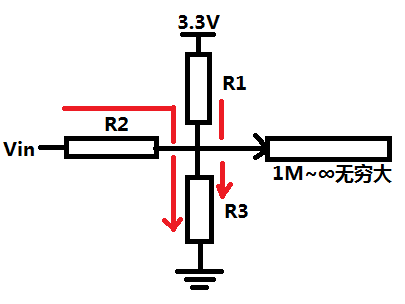
如果我要采集-10V～+10V的电压怎么办呢？在ADC VREF+参考电压接在3.3V，VREF-电压接在0V情况下。如果VREF+接的是+10V，VREF—接的是-10V，那么下面信号变换电路就不需要了

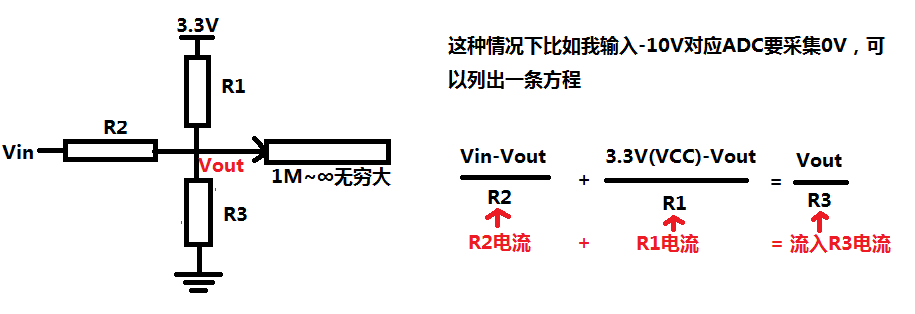


ADC采集的电压输入端要进行信号变换电路设计，才能让ADC采集-10V~+10V的电压

这种电路该怎么计算?

因为ADC电阻很大，所以流入R2的电流流进R3，流入R1的电流流进R3

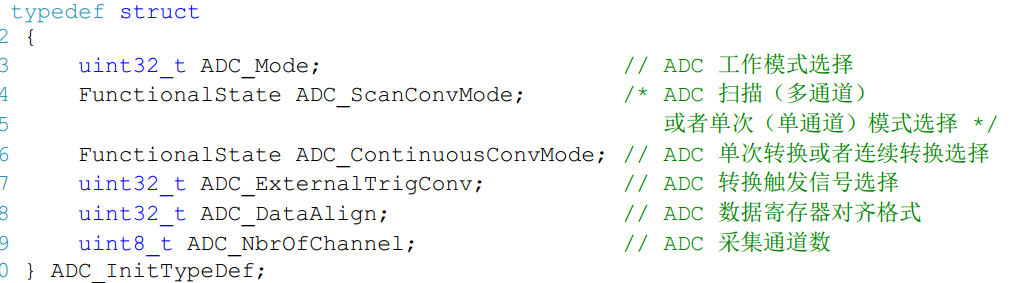
 



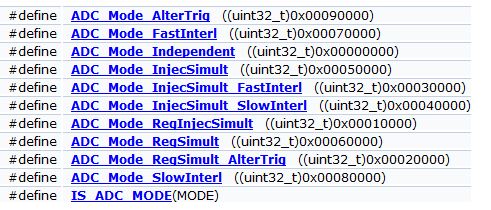
如果是+10V输入，也就是Vin设置为+10V, Vout设置为3.3V，代入上面公式去算

(Vin-Vout)/R2 + (VCC(3.3V)-Vout)/R1=Vout/15k = (10-3.3)/30K+(3.3-3.3)/10K=3.3/15K 正好两边电流相等。

ADC固件库使用



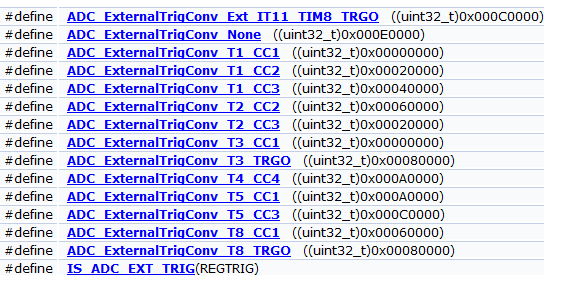
ADC\_Mode: 一般选择ADC\_Mode\_Independent独立模式，这个模式只针对一个ADC外设，其余的参数是双ADC下选择的。



ADC\_ScanConvMode:选择ENABLE，那么ADC1采集多个引脚输入的模拟量数据，比如ADC1转换完1脚的模拟量会自动去转换2脚模拟量，以此类推….。如果选择DISABLE那么ADC1只采集一个指定引脚的模拟量。

ADC\_ContinuousConvMode:选择DISABLE那么该ADC采集一次通道中的数据就不再采集了，如果选择ENABLE，那么ADC会不停的重复采集通道中的数据。我选择ENABLE。

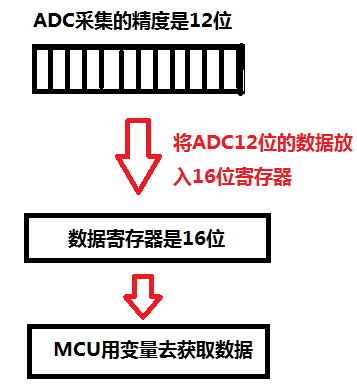
ADC\_ExternalTrigConv:ADC通道触发源选择

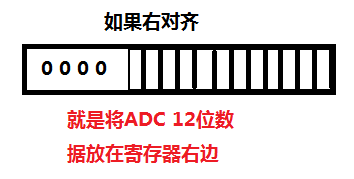


ADC\_DataAlign:数据对齐选择，我们一般选择ADC\_DataAlign\_Right 右对齐

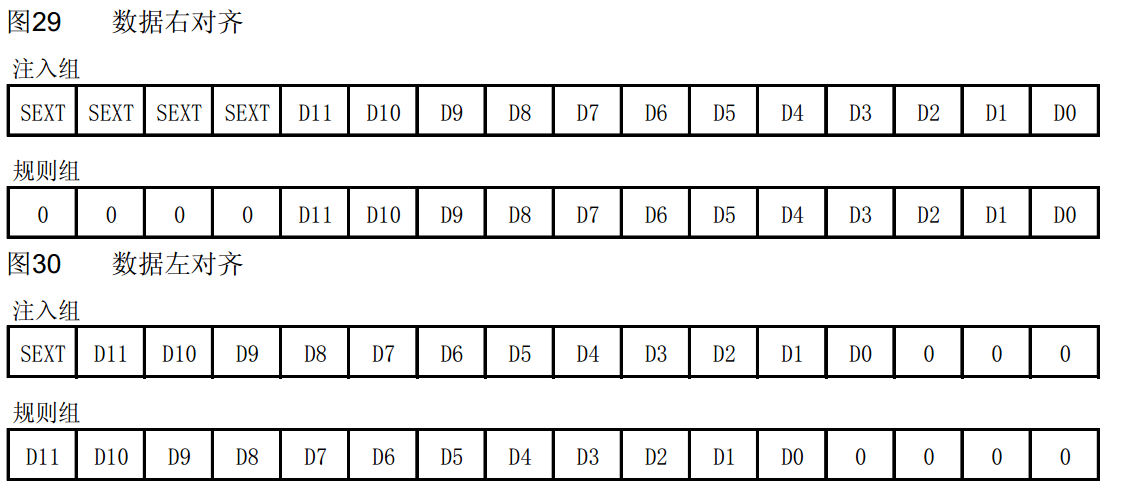


左对齐,右对齐什么意思？



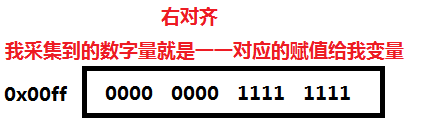


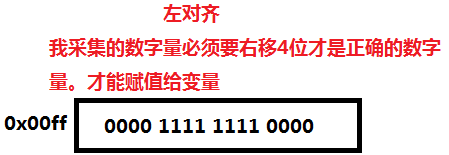




这个左对齐或者右对齐决定了你MCU变量读取出来的数据是否和ADC采集的数据一致。

右对齐的数据形式，比如我采集到了0x00ff数字量。放在数据寄存器的形式如下:



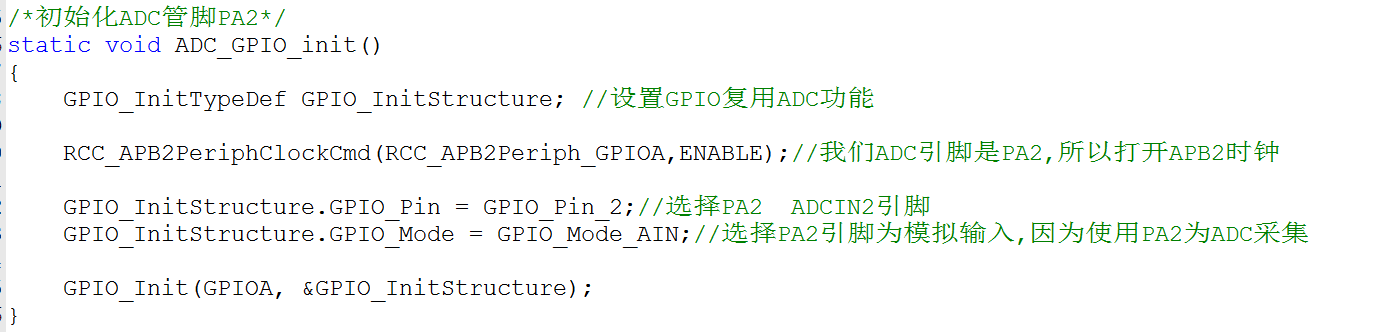


ADC\_NbrOfChannel:这是选择你要ADC转换几个通道，我MCU外部有16个引脚支持ADC转换，所以最大支持16个通道。

ADC单通道中断读取数据



采集PM2.5的模拟量



STM32有ADC1和ADC2两个外设，但是都是采集同样的模拟引脚，我们选择ADC1

设置PA2 GPIO时钟和IO功能为模拟输入



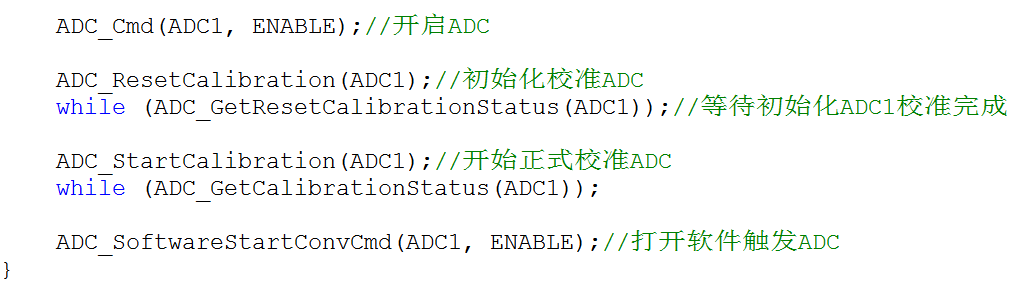
ADC时钟我设置的9M，Tconv转换最快时间 = 1.5 周期 + 12.5 周期 = 14 周期 = 1us。我这里设置55个周期，那么就是55+12.5=67.5=7.5us

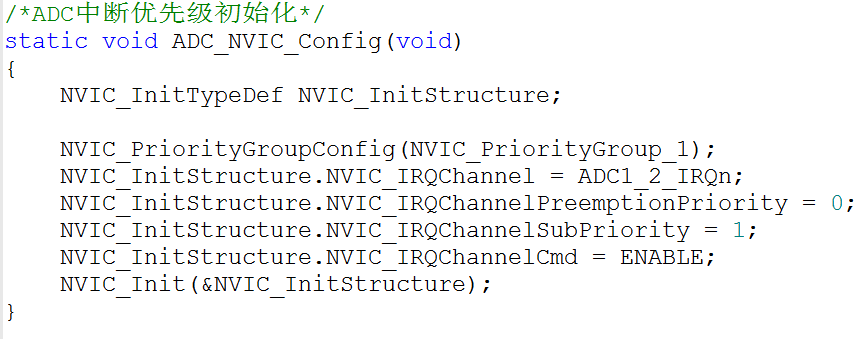
配置ADC时钟为9M,STM32的ADC时钟最大不能超过14M

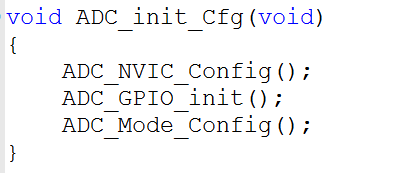
我们用的PA2引脚，就是ADC\_IN2,所以这里必须要写Channel1\_2

这里默认写1因为是单通道模式，如果是多通道，那么就要在这里设置规则

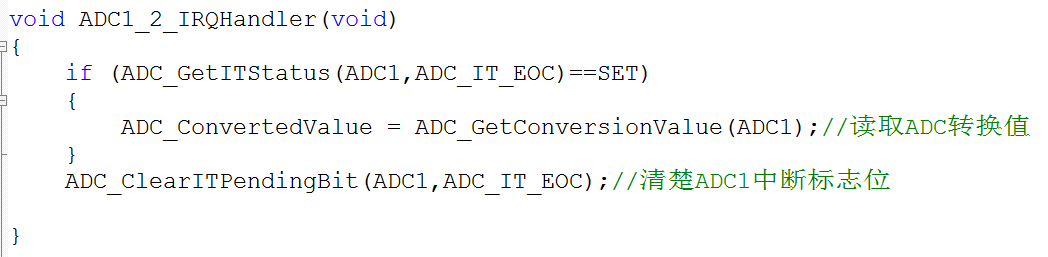
因为我们是PA2采集数据就一个ADC通道，所以设置为独立模式



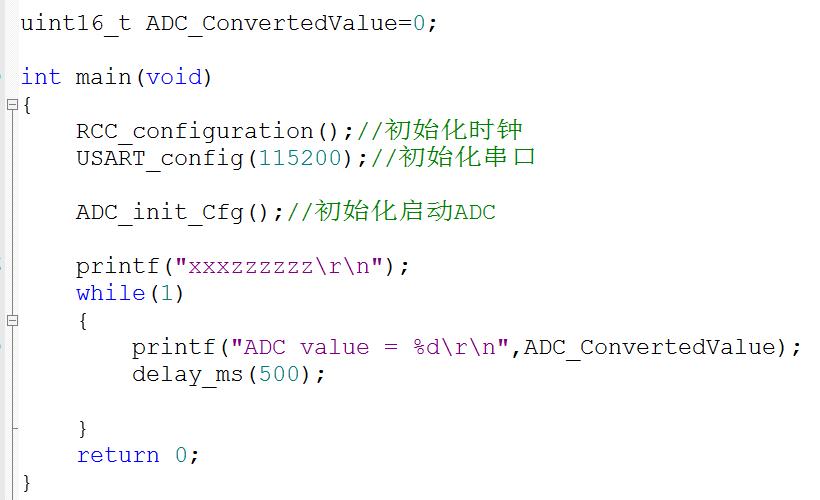


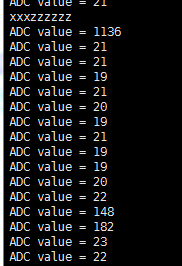


初始化ADC引脚，ADC功能和ADC中断优先级的顺序



ADC每次转换结束的中断函数，记住中断函数名不要写错了，否则会开机卡死



运行一切正常