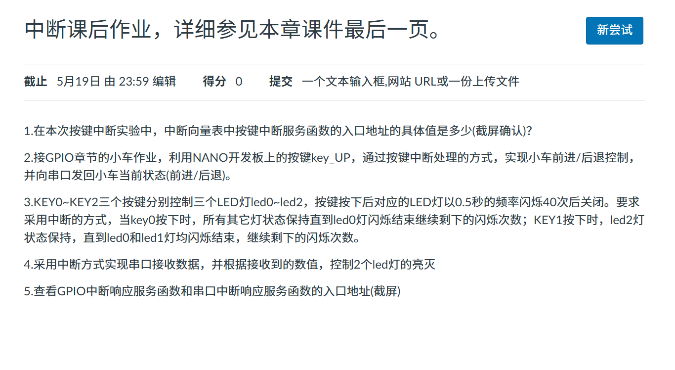
**作业**



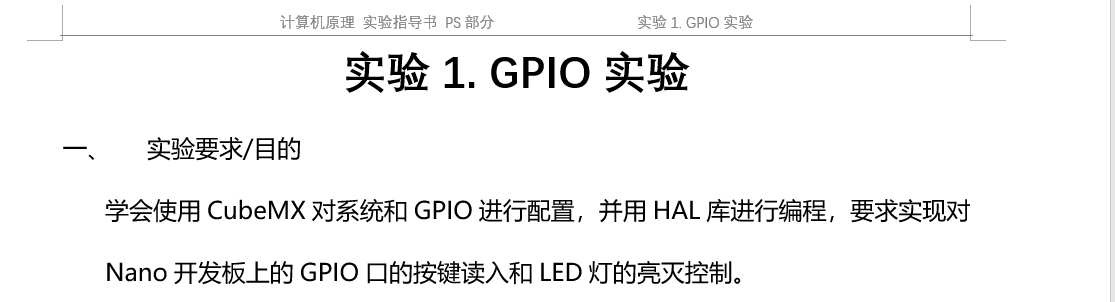




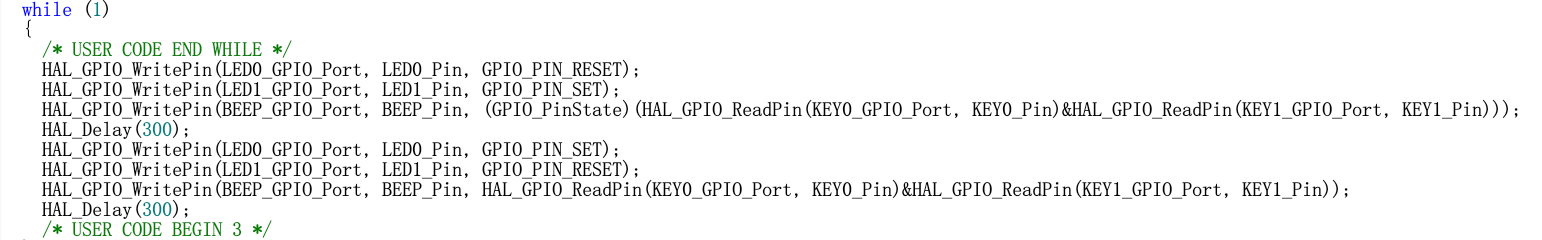




**实验课**

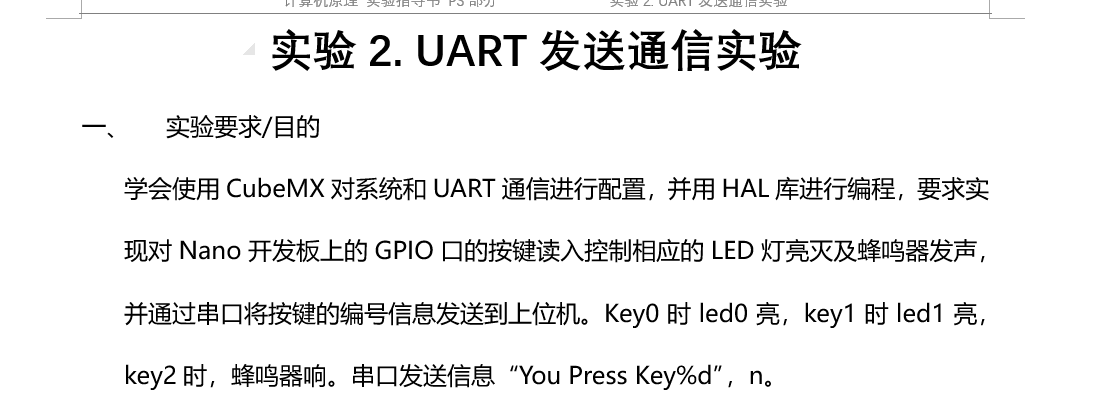


关键步骤与关键代码

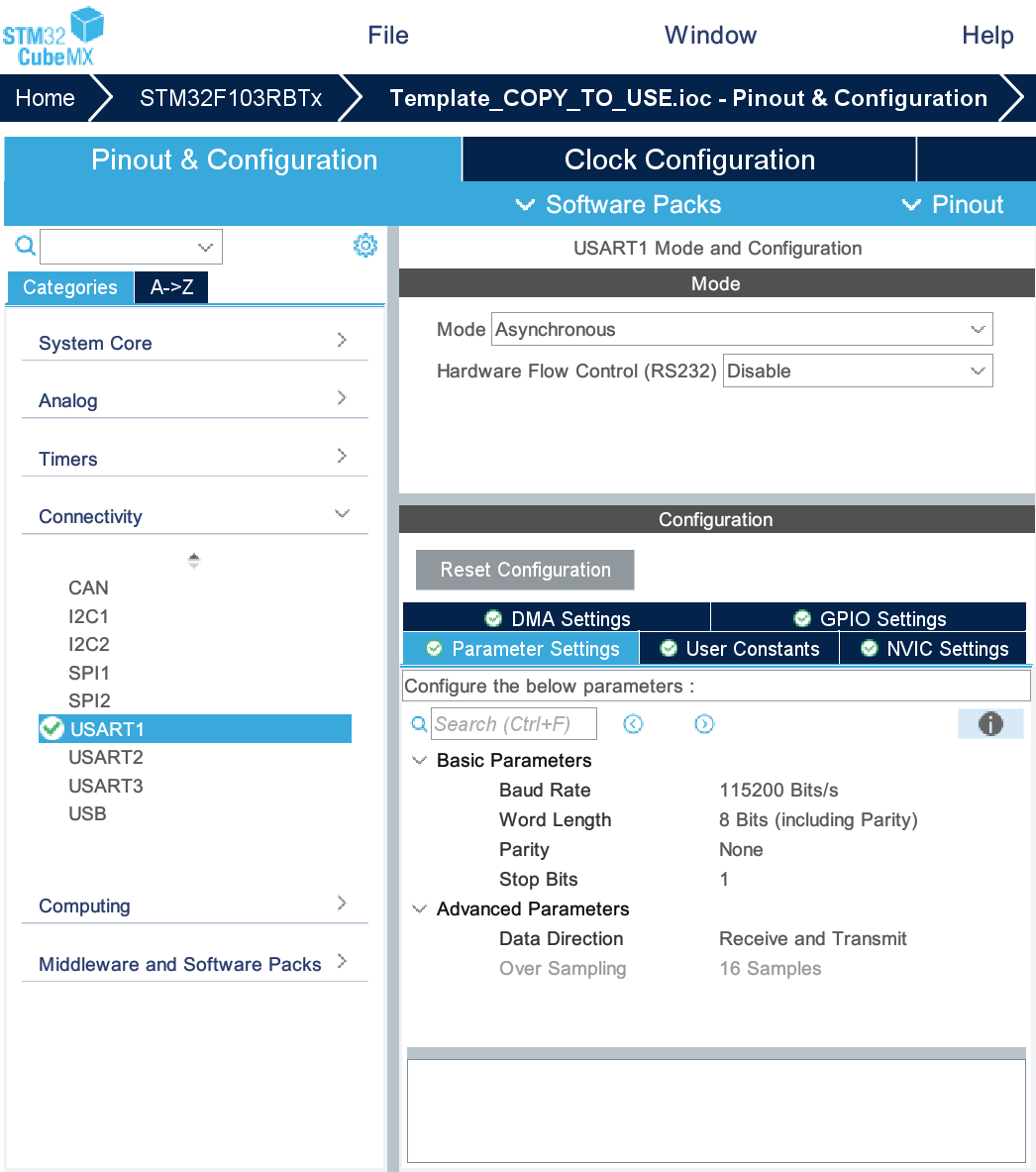


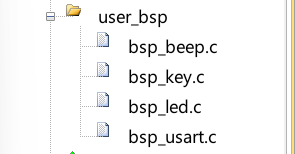
实验收获/心得/体会/踩坑记录

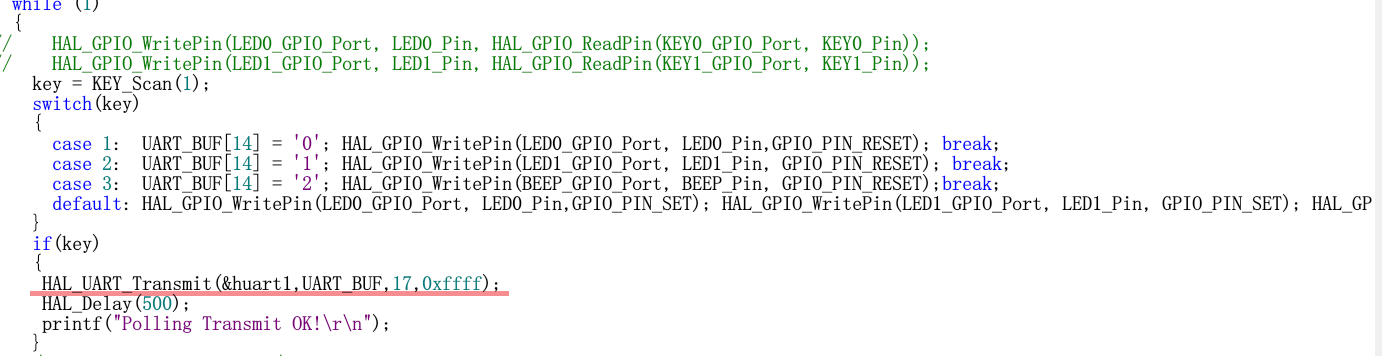
* 通过本实验熟悉掌握了HAL\_GPIO\_WritePin函数的使用，也直观理解了GPIO的作用

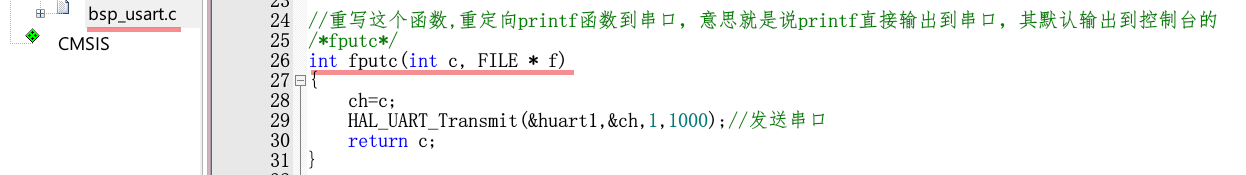


关键步骤与关键代码

配置usart

按键消除抖动





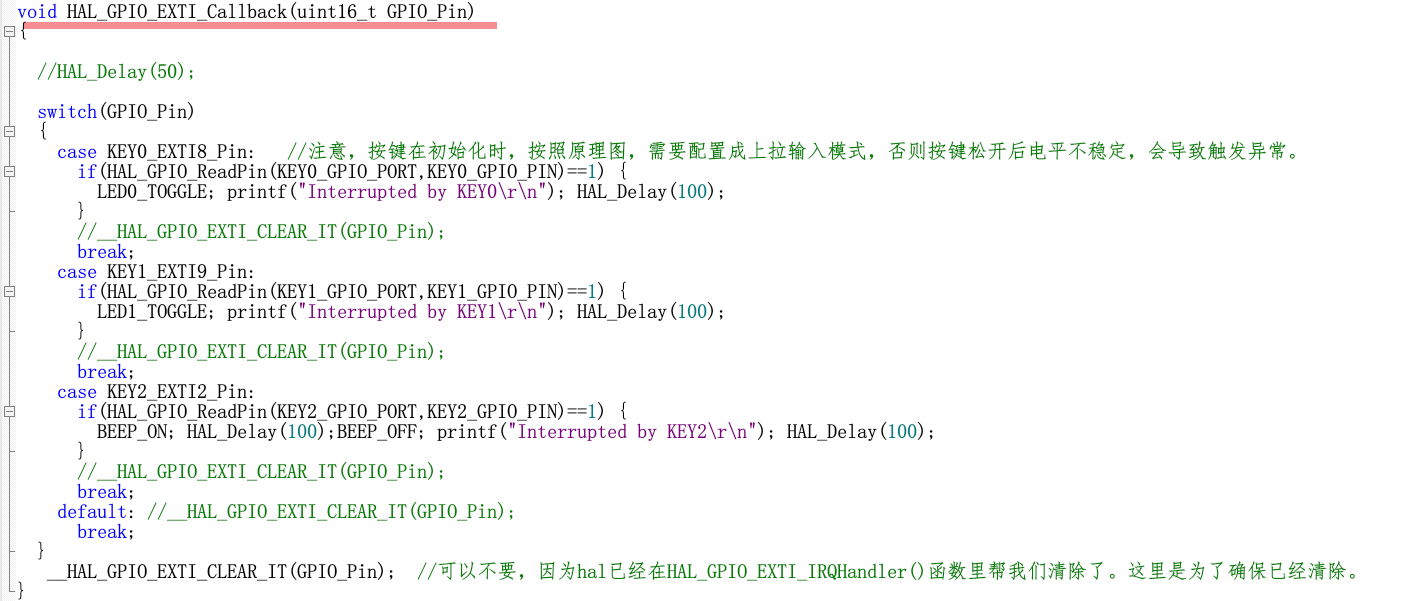
实验收获/心得/体会/踩坑记录

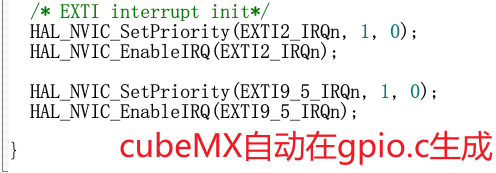
[STM32 出现SWD/JTAG Communication Failure\_swd和jtagcommunication failure-CSDN博客](https://blog.csdn.net/weixin_51928022/article/details/128100908)

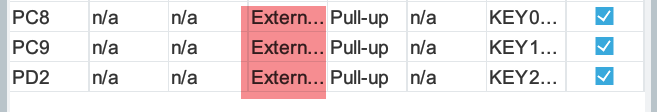
* 原因：使用CMSIS进行Debug，从ST\_link切换之后，需要切换到under reset



关键代码



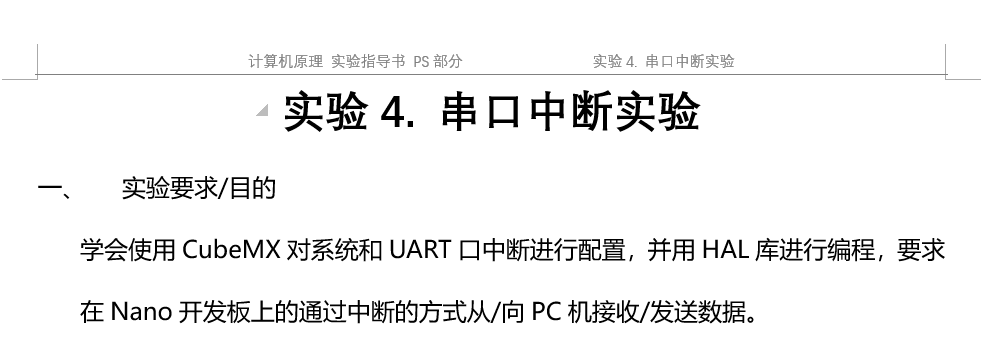






实验收获/心得/体会/踩坑记录

* 了解了中断callback函数的基本写法
* 中断服务函数的定义可以直接在CubeMX设置NVIC那里勾选GPIO或者USART就可以自动定义（EXTI2\_IRQn 与 EXTI9\_5\_IRQn），无需你AI去找，AI找甚至找的不对

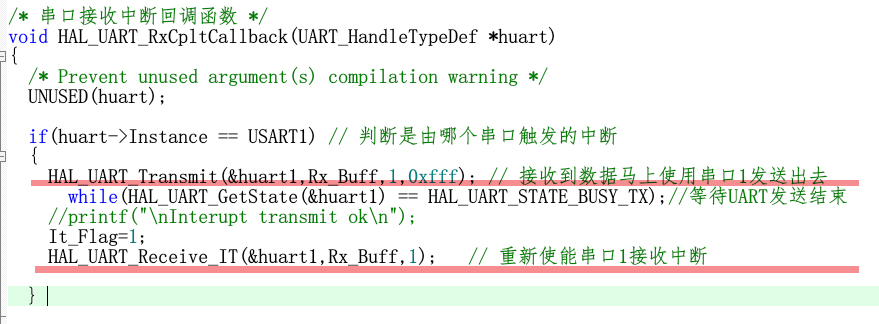


关键步骤与关键代码





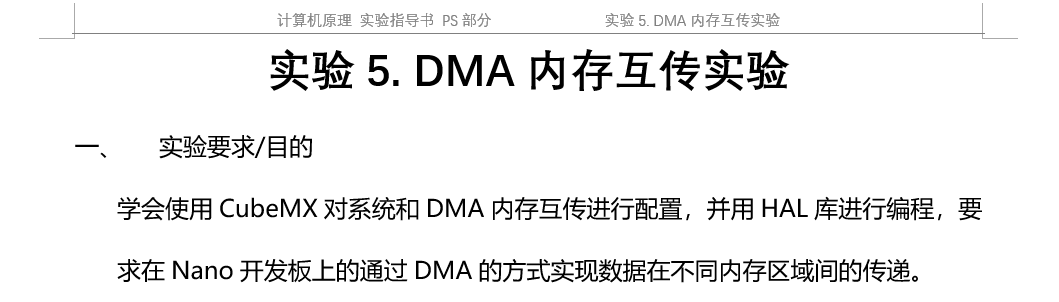
回调函数（可以放在bsp\_usart.c，当然main.c也可以）



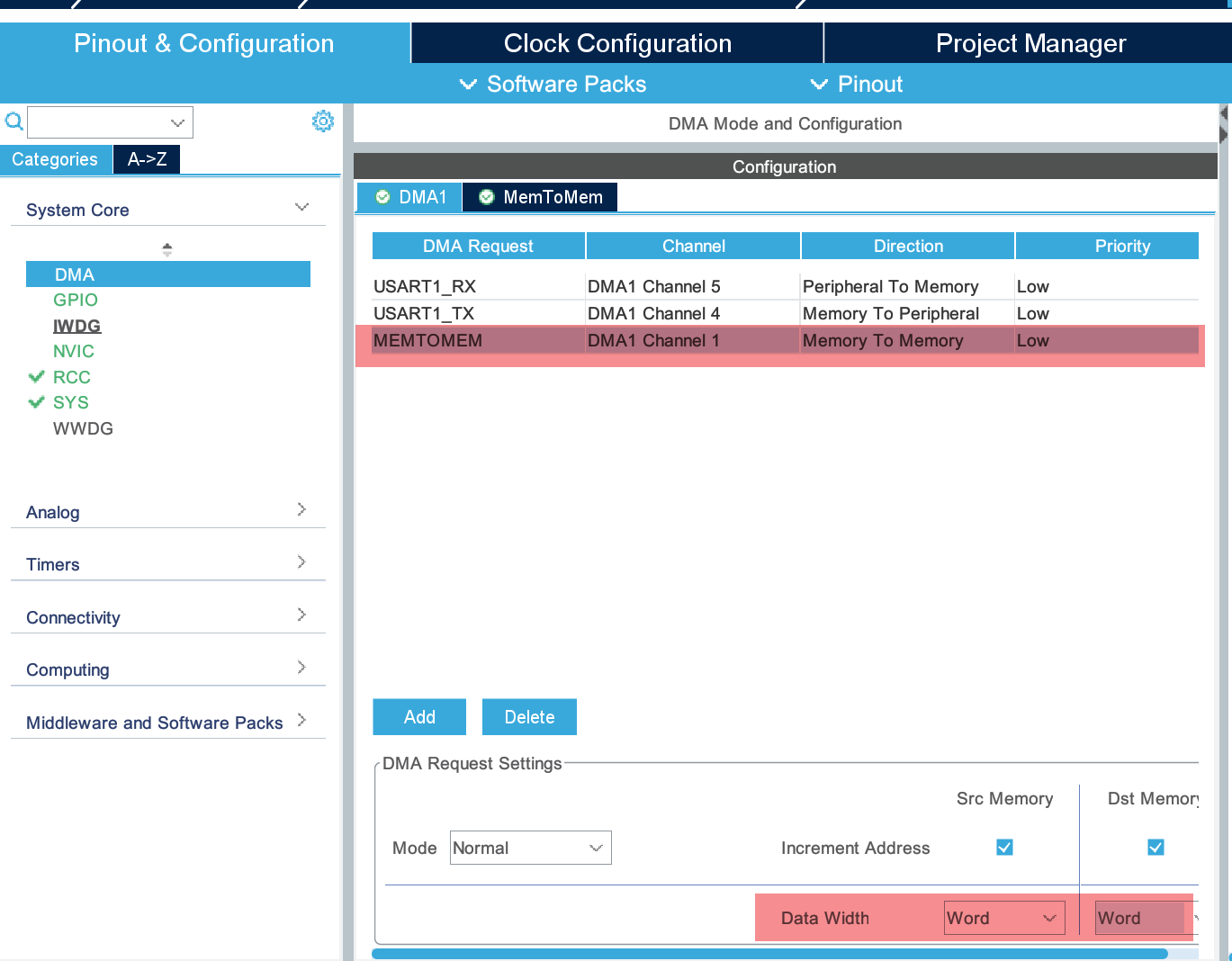
实验收获/心得/体会/踩坑记录

* 学会了串口中断函数的编写：最后要重新使能串口Receive，这点和GPIO不同
* 串口的中断服务函数是USART1\_IRQHandler()，获取方式同GPIO的

**老师说：实验报告写这里之后任意一个实验即可**

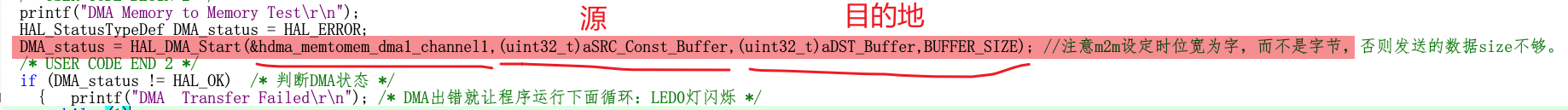


关键步骤与关键代码



注意位宽是word (32 bits)



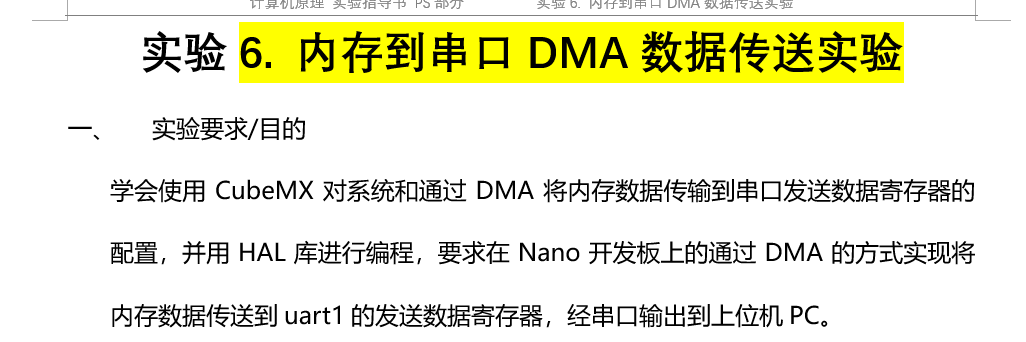


核心代码

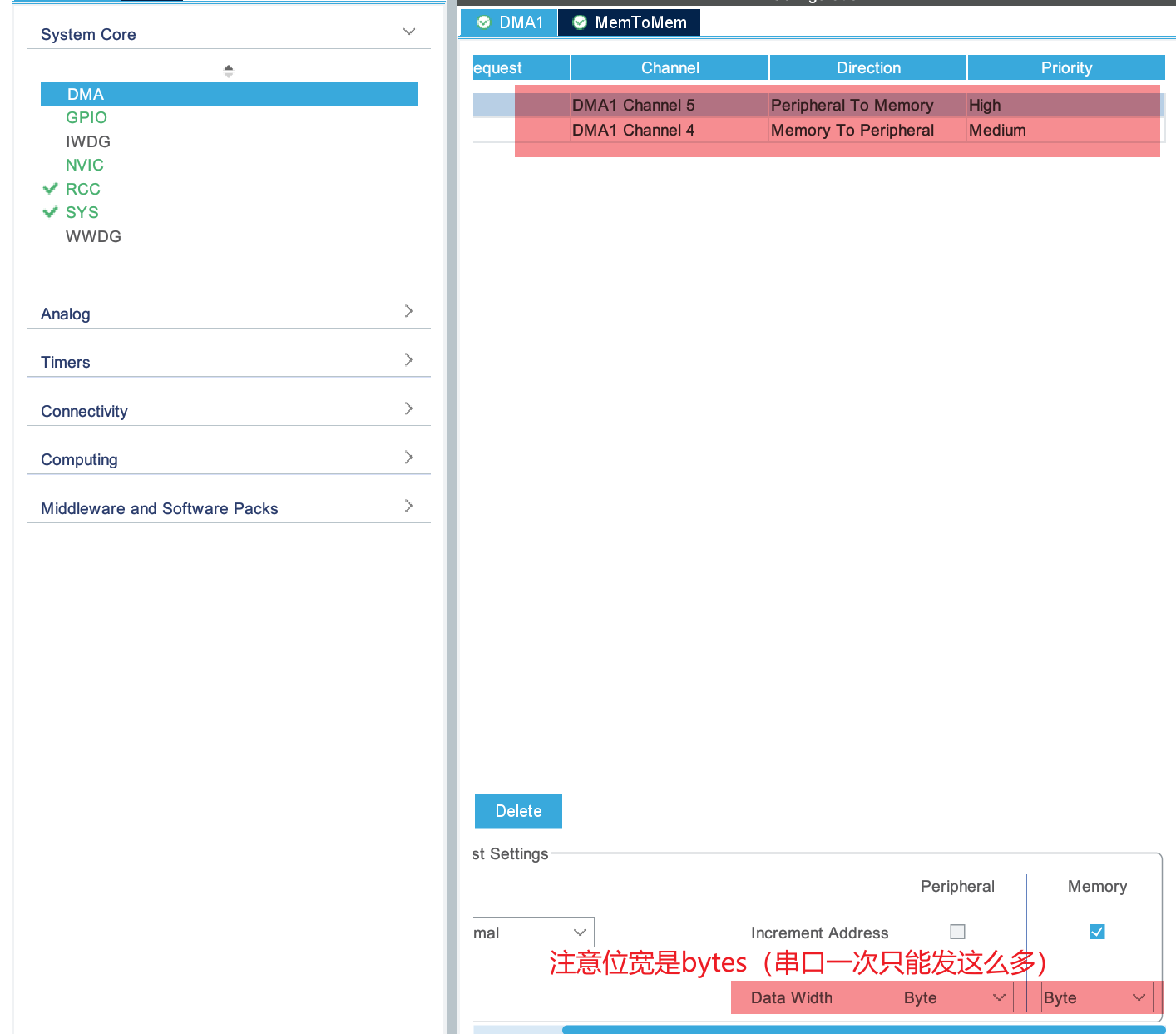


实验收获/心得/体会/踩坑记录

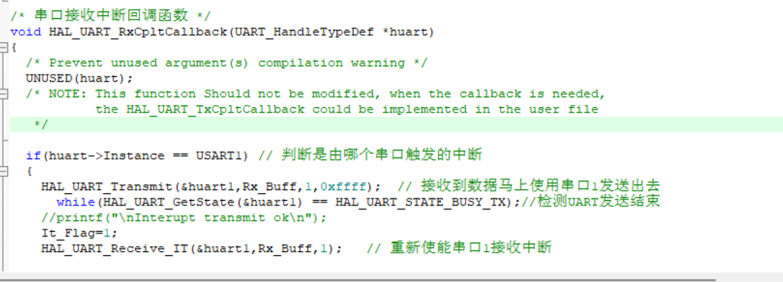
* M2M的传递方式需要写明源&目的地Memory的起始地址
* 传送函数很简单，HAL\_DMA\_START()



关键步骤与关键代码

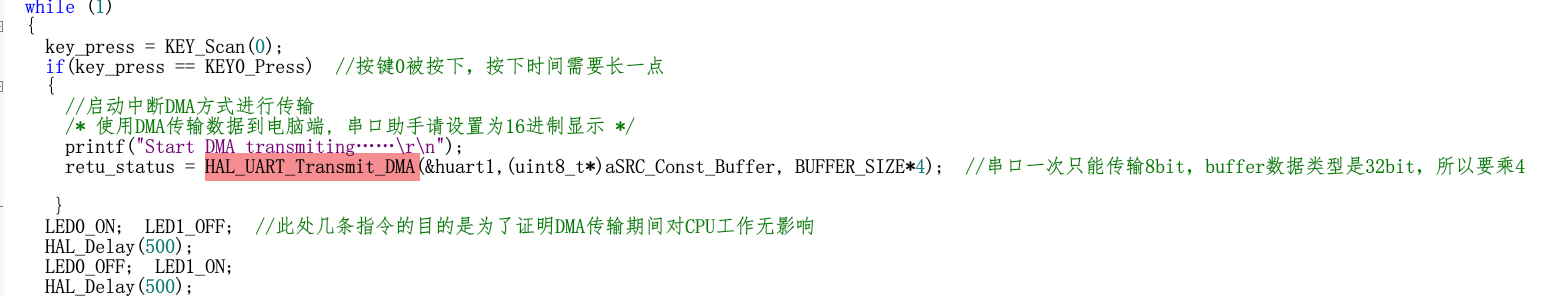


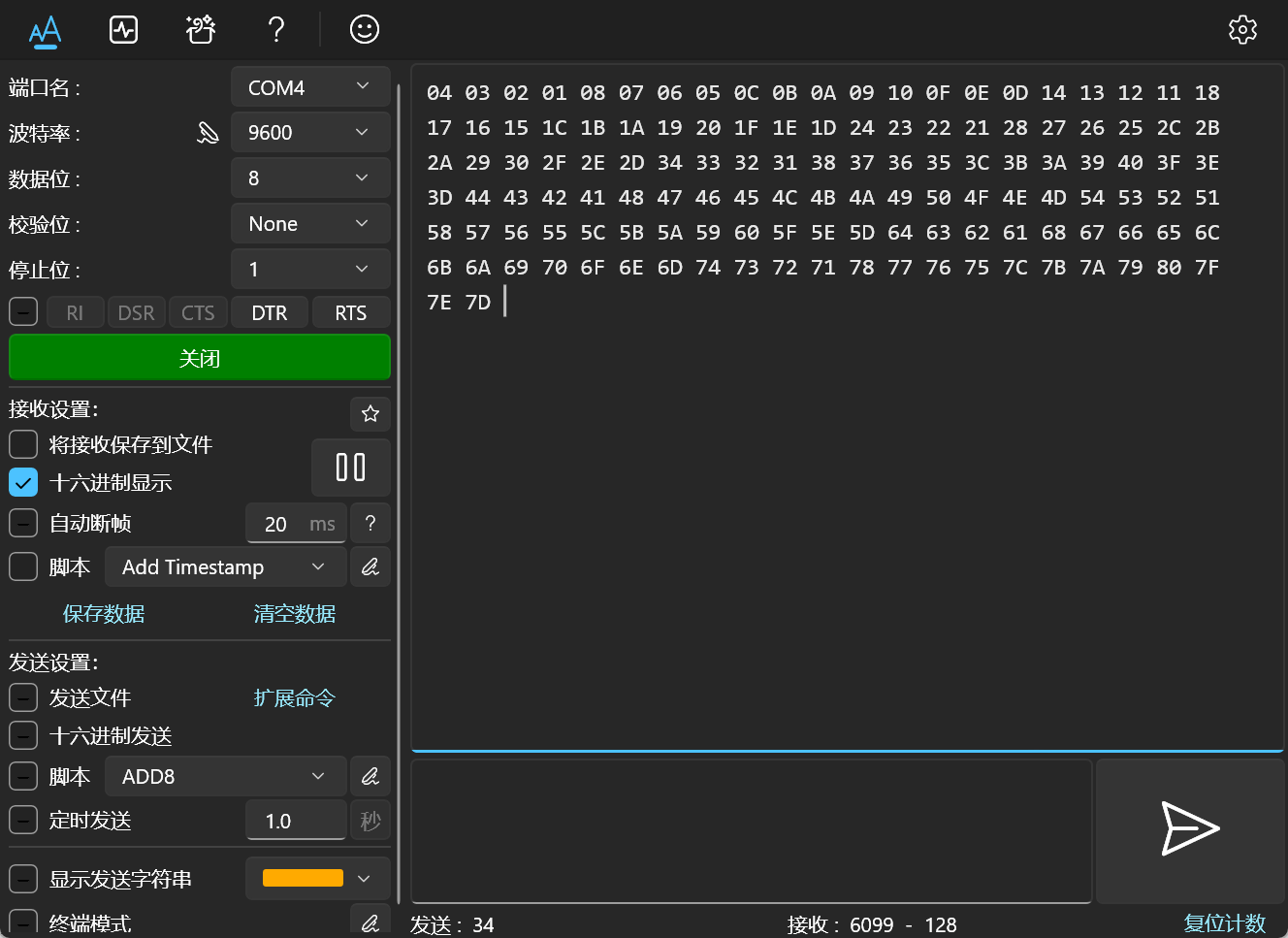
注意现在M2UART的位宽是byte

另外，串口中断也要打开

串口中断回调函数



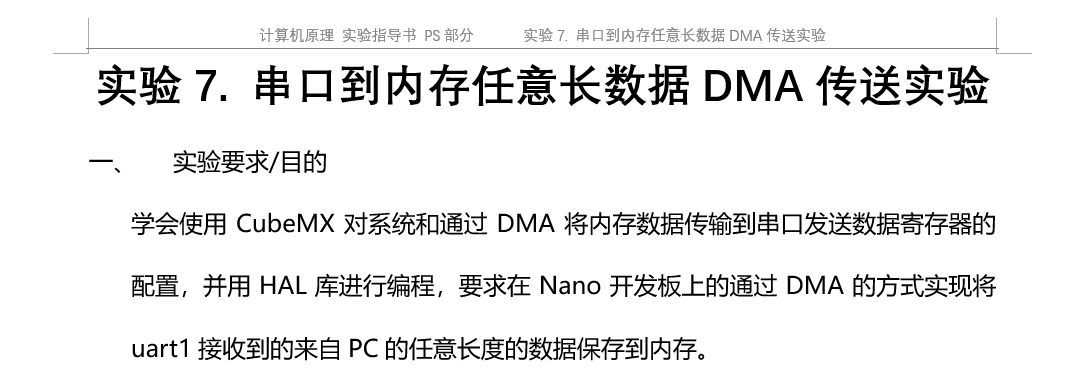




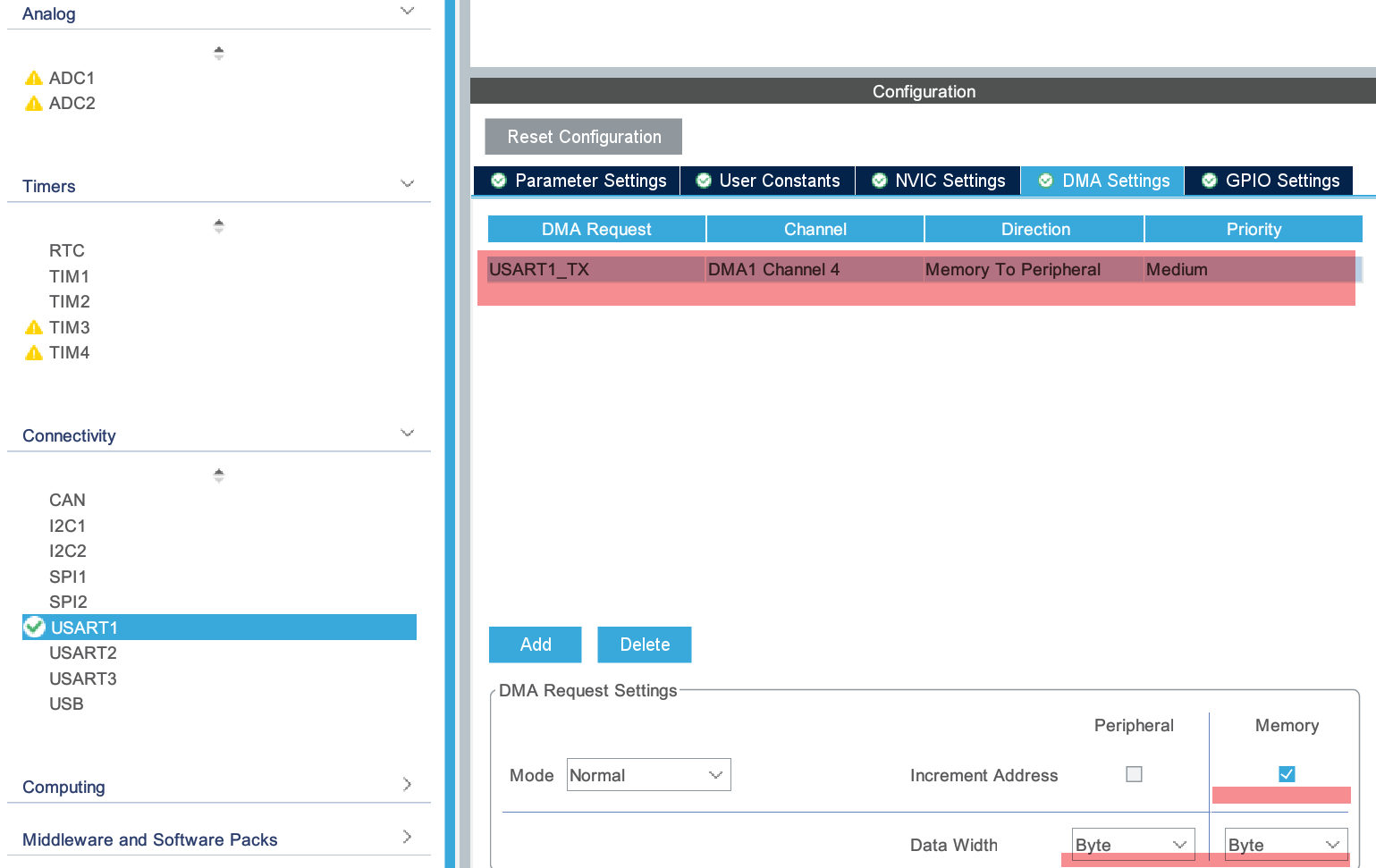
实验现象

实验收获/心得/体会/踩坑记录

* M2M可以直接按Word为单位传输，但是M2USART不行，串口按byte传输，于是定义几个uint\_8\_t的buffer
* 首先串口从用户处接收数据，因此使用一个串口中断HAL\_UART\_Receive\_IT( )并写回调函数RxCallBack
* 串口的发送数据寄存器接受memory的数据，使用的是HAL\_UART\_Transmit\_DMA()



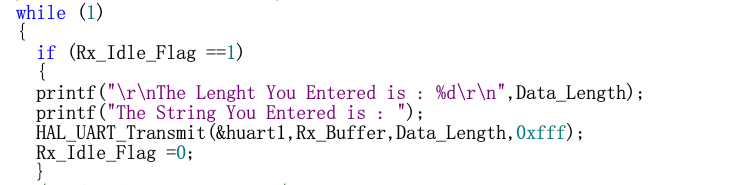
关键步骤与关键代码



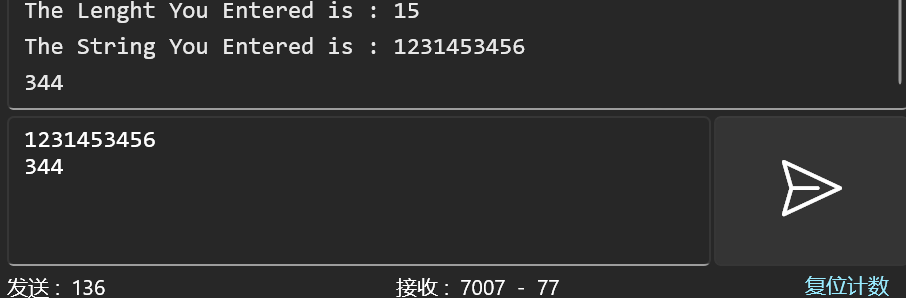
只保留Tx这个设置（transmit），其他和之前一样（后来发现搞错了，其实应该保留Rx receive）



回调函数（3个）

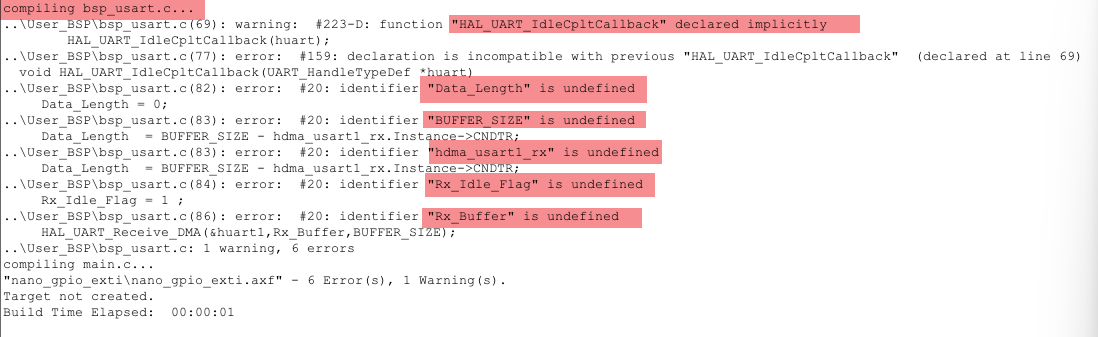


Main.c



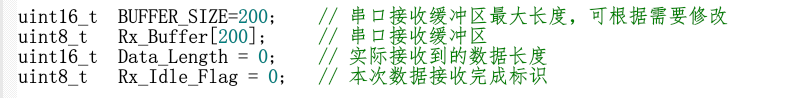
实验收获/心得/体会/踩坑记录

在编写完bsp\_usart和main的内容后，bsp\_usart.c编译报错



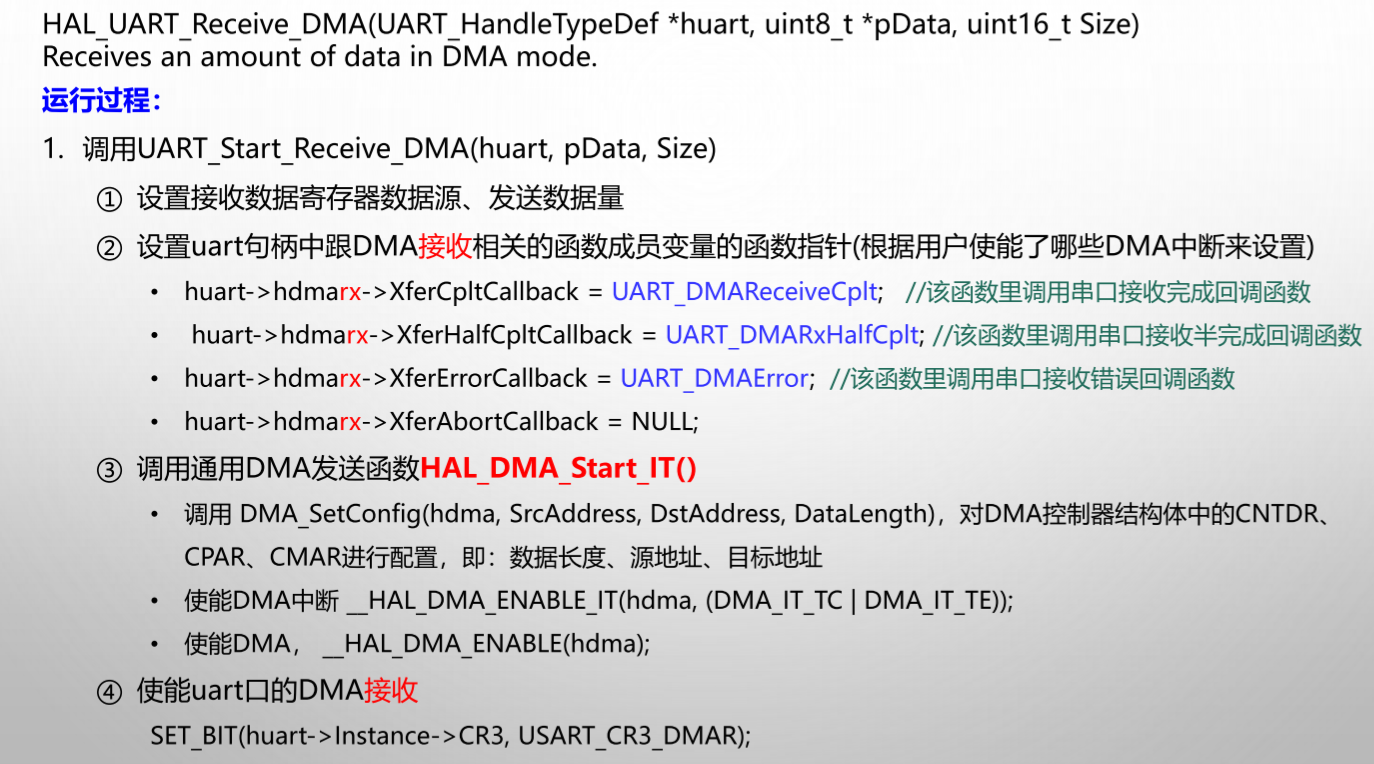
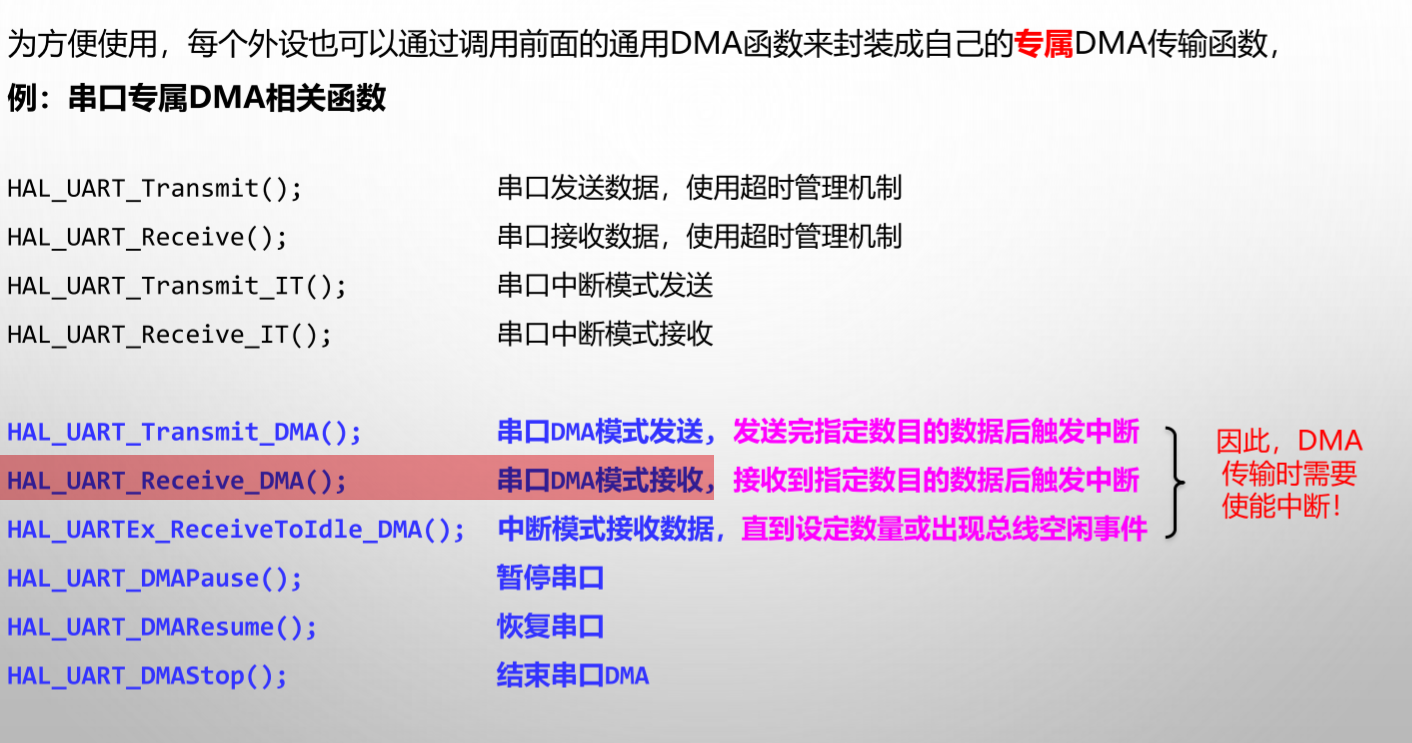
解决方法:

后面5个报错是因为没有定义Rx\_Buffer等变量

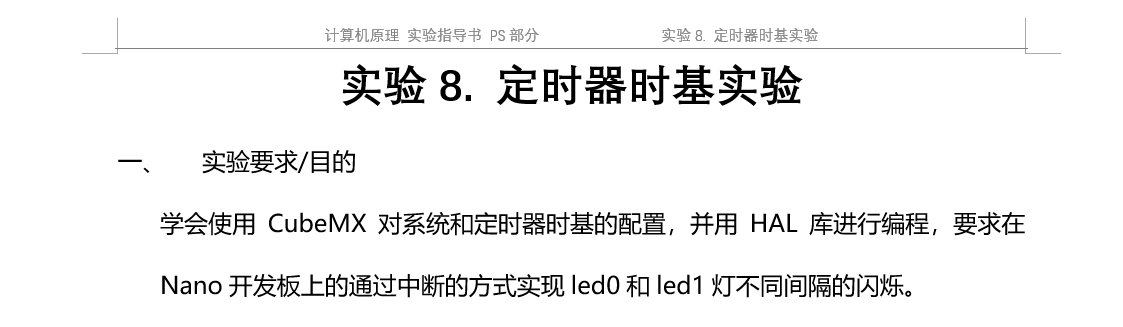


然后发现，应该设置的是Rx的中断

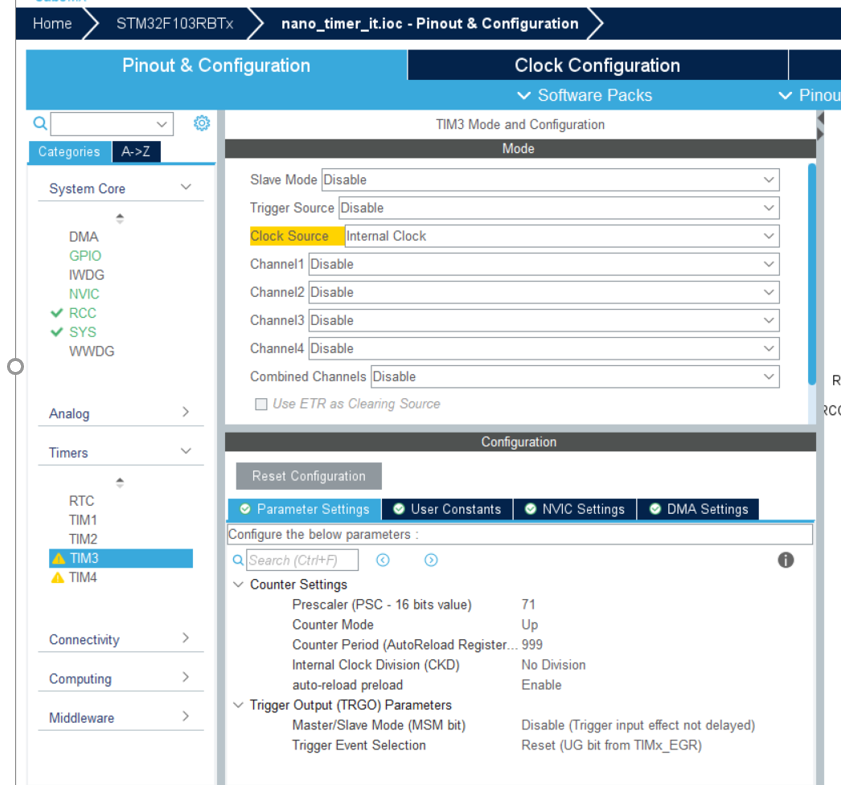
修改之后，成功运行



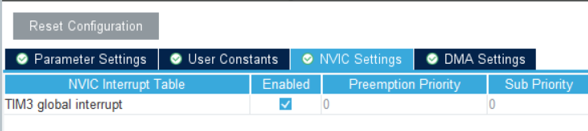
这个是HAL\_UART\_Receive\_DMA()原理



关键步骤与关键代码



设置计时器TIM3，设定1us的时钟输出



同时打开计时器的中断

,

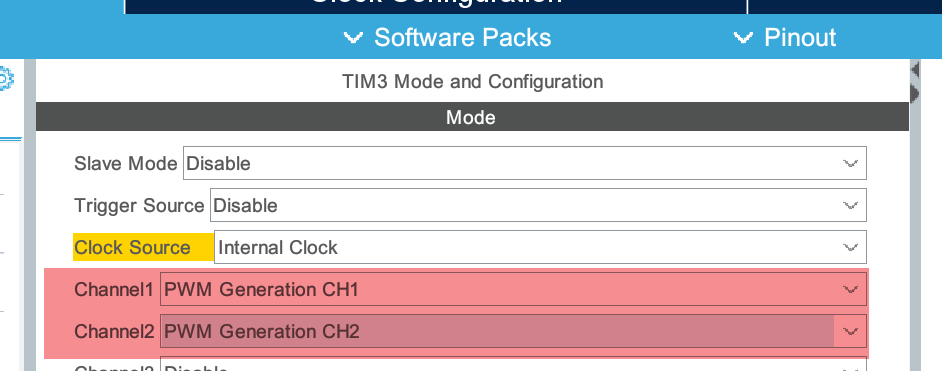
TIMCallBack的实现较为简单（设置count变量，随时间相加，满足一定时间toggle led即可，故略过）

实验收获/心得/体会/踩坑记录

* 最开始timer未成功设定1us的时钟输出，导致led虽然按照不同的间隔闪烁，但是速度很快。原因是上面cubeMX配置处的auto-reload period错写成了99，正确是999



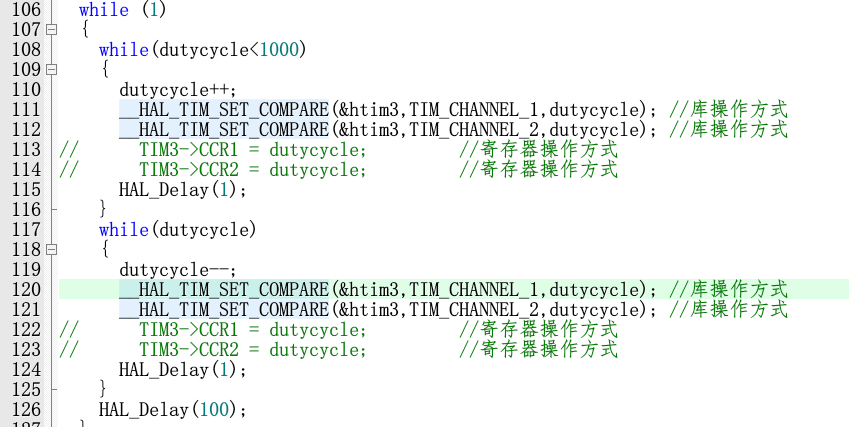
关键步骤与关键代码



在刚才的基础上，配置TIM3的channel1 channel2为PWM



启动PWM



通过TIM\_SET\_COMPARE来实现PWM调制

上面是变量，下面是变暗

实验收获/心得/体会/踩坑记录

[ ] 什么是ch1 ch3?

* 就是TIM时钟的输出channel

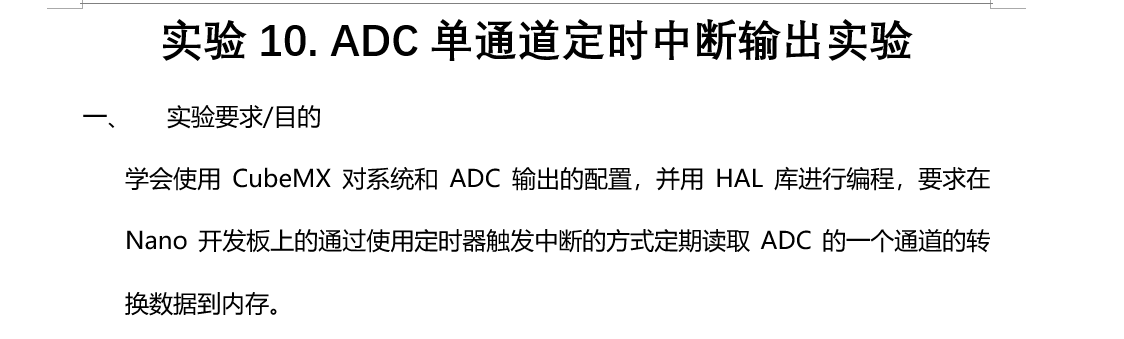
“通道（Channel） ” 是定时器（TIMx）的一个重要概念，它代表了定时器与外部引脚之间进行输入/输出操作的接口。

捕获比较寄存器（TIMx\_CCR1~4） 该寄存器总共有4个，对应4个输通道CH1~4

通过实验代码理解了STM32 定时器PWM输出生成原理: 用定时器的比较输出通道，可以产生PWM波。

[ ] 什么是dutycycle?

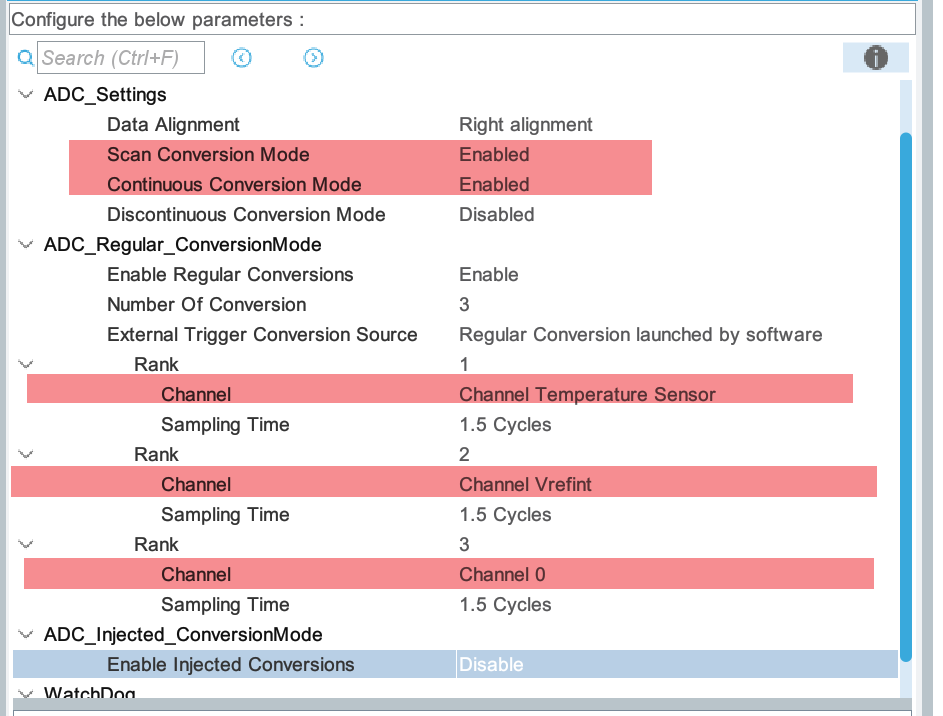
* 就是占空比



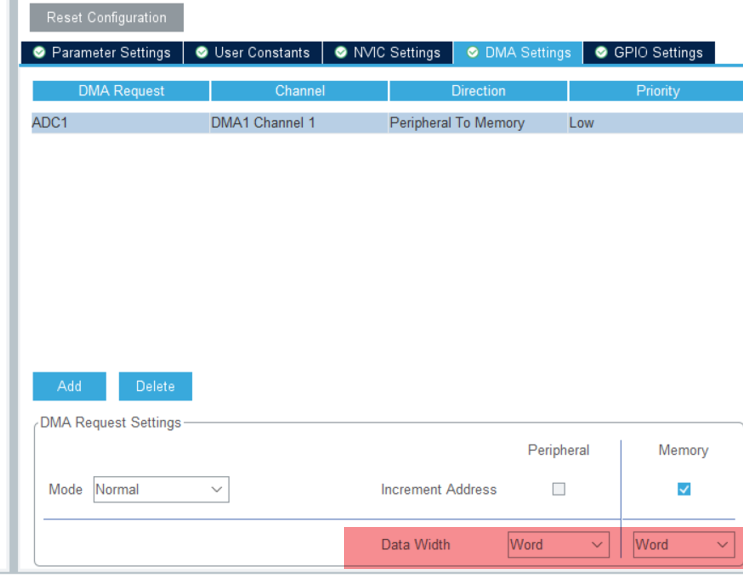
单通道不难，就是多通道的一部分，实验报告见多通道



关键步骤与关键代码



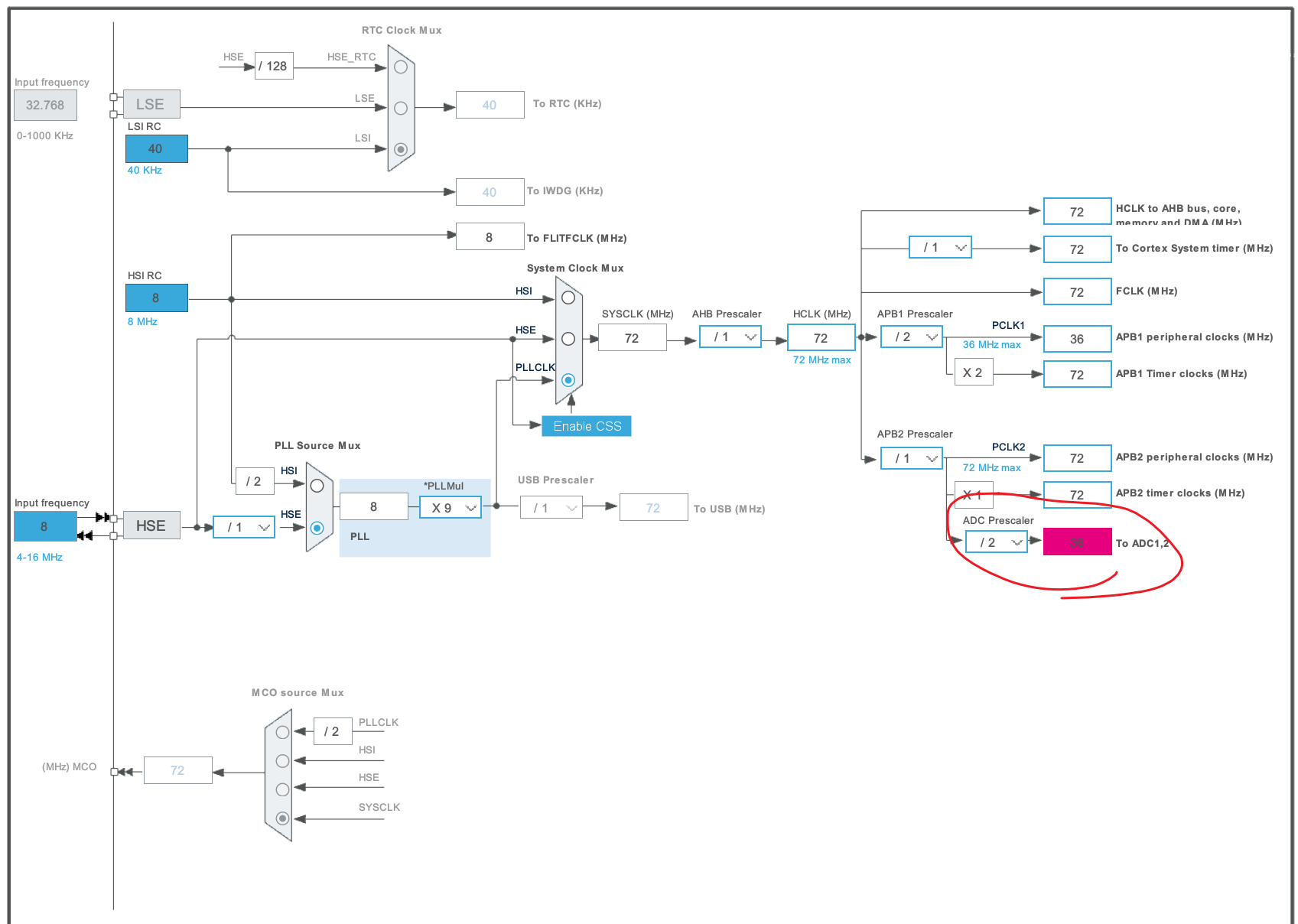
ADC处一大堆要设置的channel



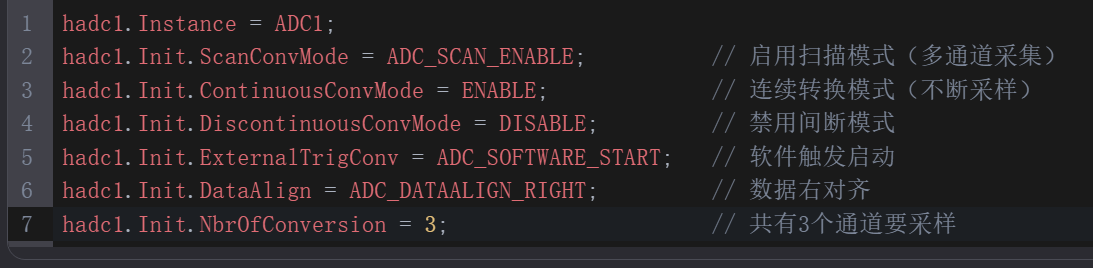
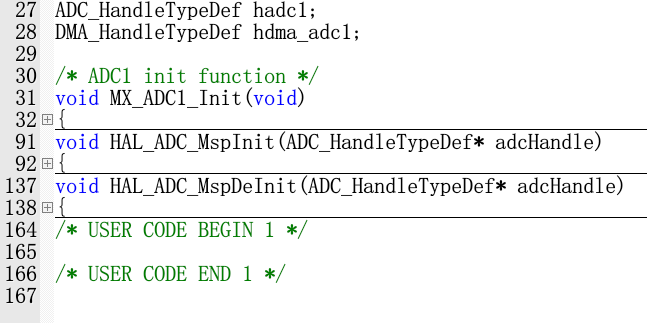
以及ADC的DMA



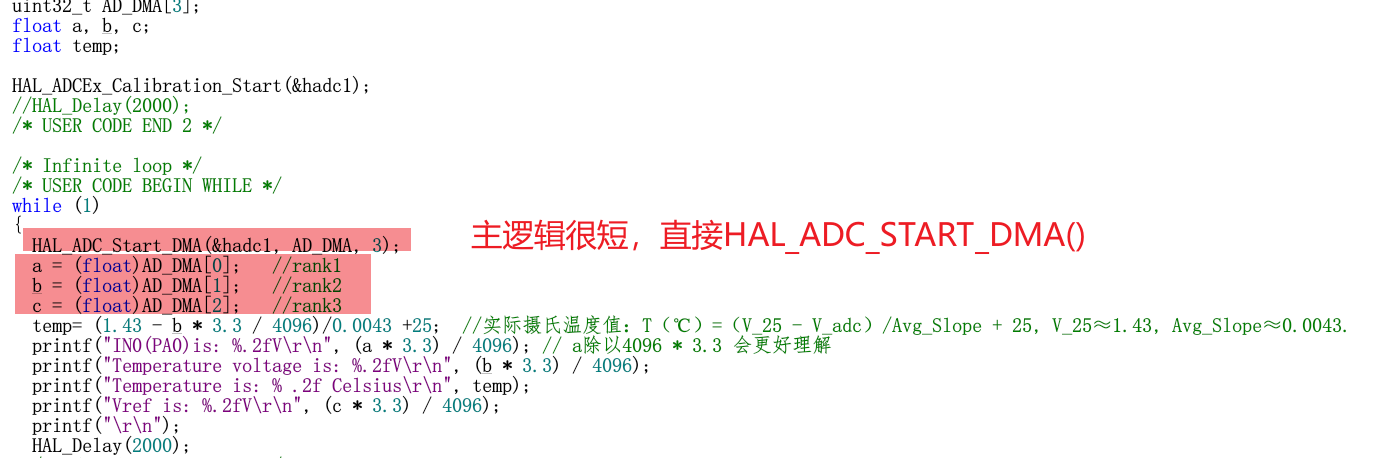
以及ADC的中断



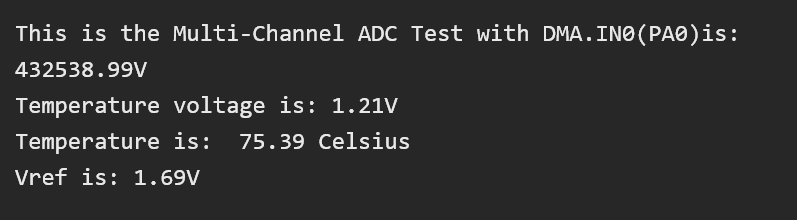
注意这里出现ADC时钟树报错



生成的代码是这些



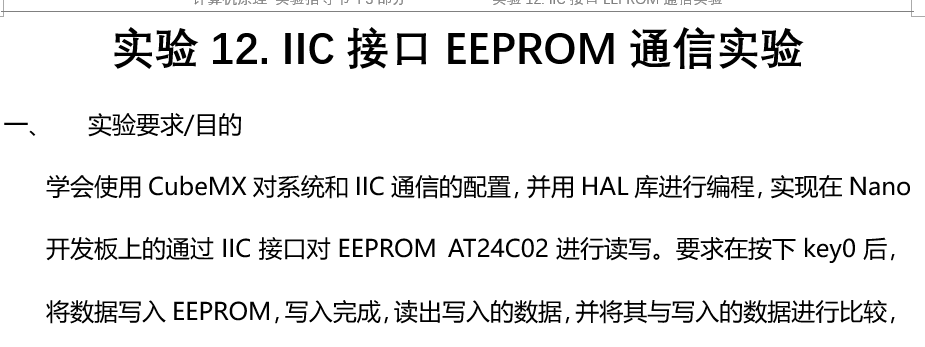
代码主逻辑



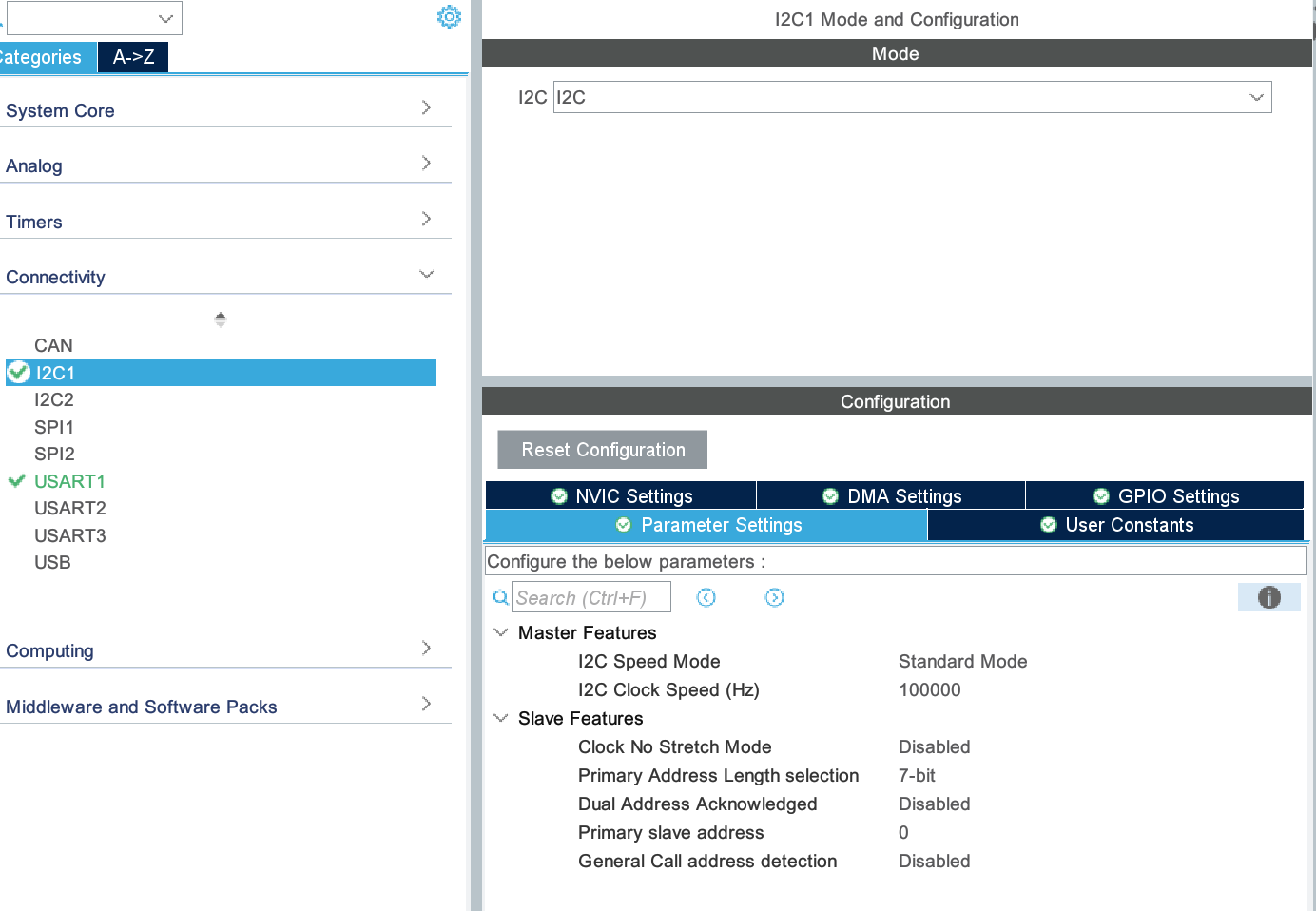
实验效果——没有问题

实验收获/心得/体会/踩坑记录

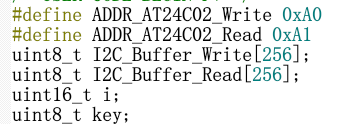
* 傻瓜式编程，选择好ADC的channel便可以读取模拟信号。HAL\_ADC\_START()直接就把模拟量转成数字，保存子啊AD\_DMA[ ]这个数组里



关键步骤与关键代码



直接使能I2C即可，其他参数默认

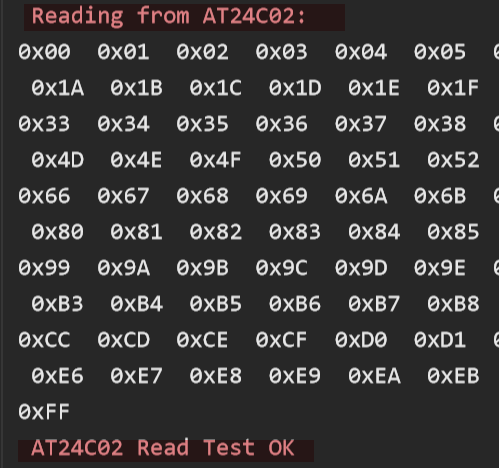
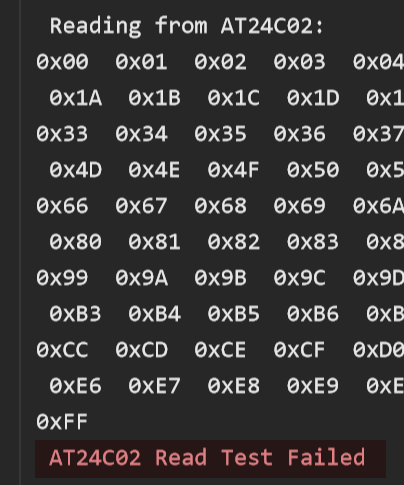


准备slave（AT24C02）的addr与buffer





读写逻辑

现象：如果按了KEY0发送之后再按KEY2读取，则显示success

如果不按KEY0发送就直接按KEY1读取，则failed，蜂鸣器响起

实验收获/心得/体会/踩坑记录

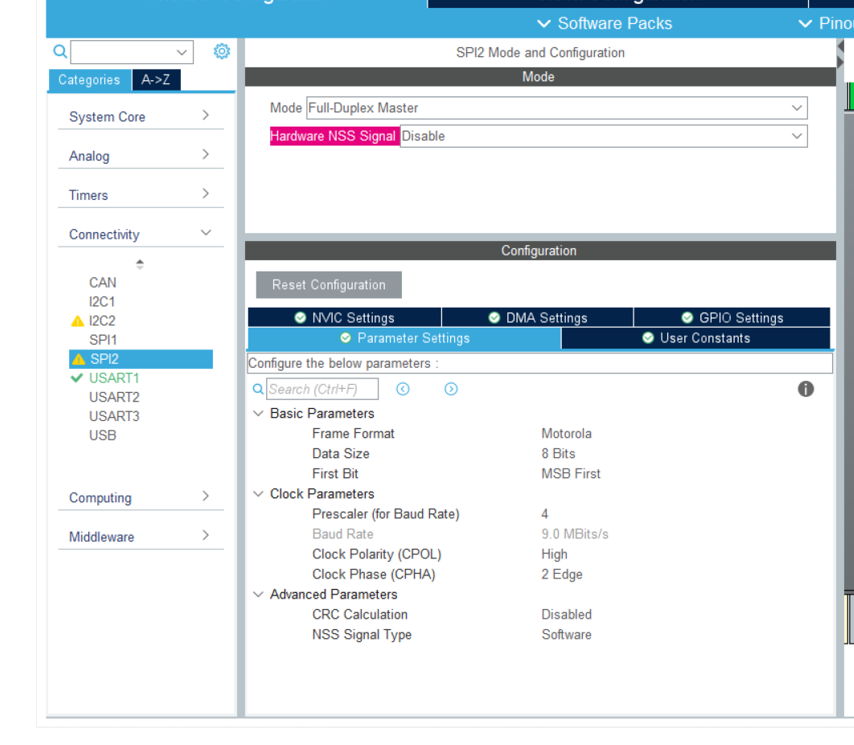
* 函数封装的太好了，找不到I2c主从通信和仲裁的特点
* AI回答：
* 主机（Master）角色由 MCU 扮演
* 从机（Slave）地址作为参数传入，在这里就是EEPROM的地址（AT24C02的地址）

[ ] 为什么EEPROM读和写的起始地址不一样 ？

* I²C 协议要求在每次通信开始时，主机（Master）必须发送一个地址帧，以指定目标设备。地址帧的最后一位是 R/W 位 ，用于区分当前操作是读还是写。
* 写操作 ：主机发送设备地址 + **0**，表示接下来要向该设备写数据。
* 读操作 ：主机发送设备地址 + **1**，表示接下来要从该设备读取数据。



关键步骤与关键代码



使能SPI，用mode3的配置来写FLASH



实验现象（同上一个）





最重要的内容封装在bsp\_w25xx.c里（flash驱动文件）

实验收获/心得/体会/踩坑记录

* 通过本次实验直观理解了SPI全双工传输的原理（在FLASH驱动程序使用HAL\_SPI\_Transmit\_Receive()函数）
* 对**单主单从连接已经掌握，但是对一主多从（例如作业中菊花链连接操作MAX5232 DAC）不熟悉**