目录

[引入DSP数学库 2](#_Toc193919846)

[开启外部晶振 5](#_Toc193919847)

[时钟源配置 6](#_Toc193919848)

[SWD接口配置 6](#_Toc193919849)

[串口配置 7](#_Toc193919850)

[LED灯配置 12](#_Toc193919851)

[定时器配置 13](#_Toc193919852)

[ADC配置 18](#_Toc193919853)

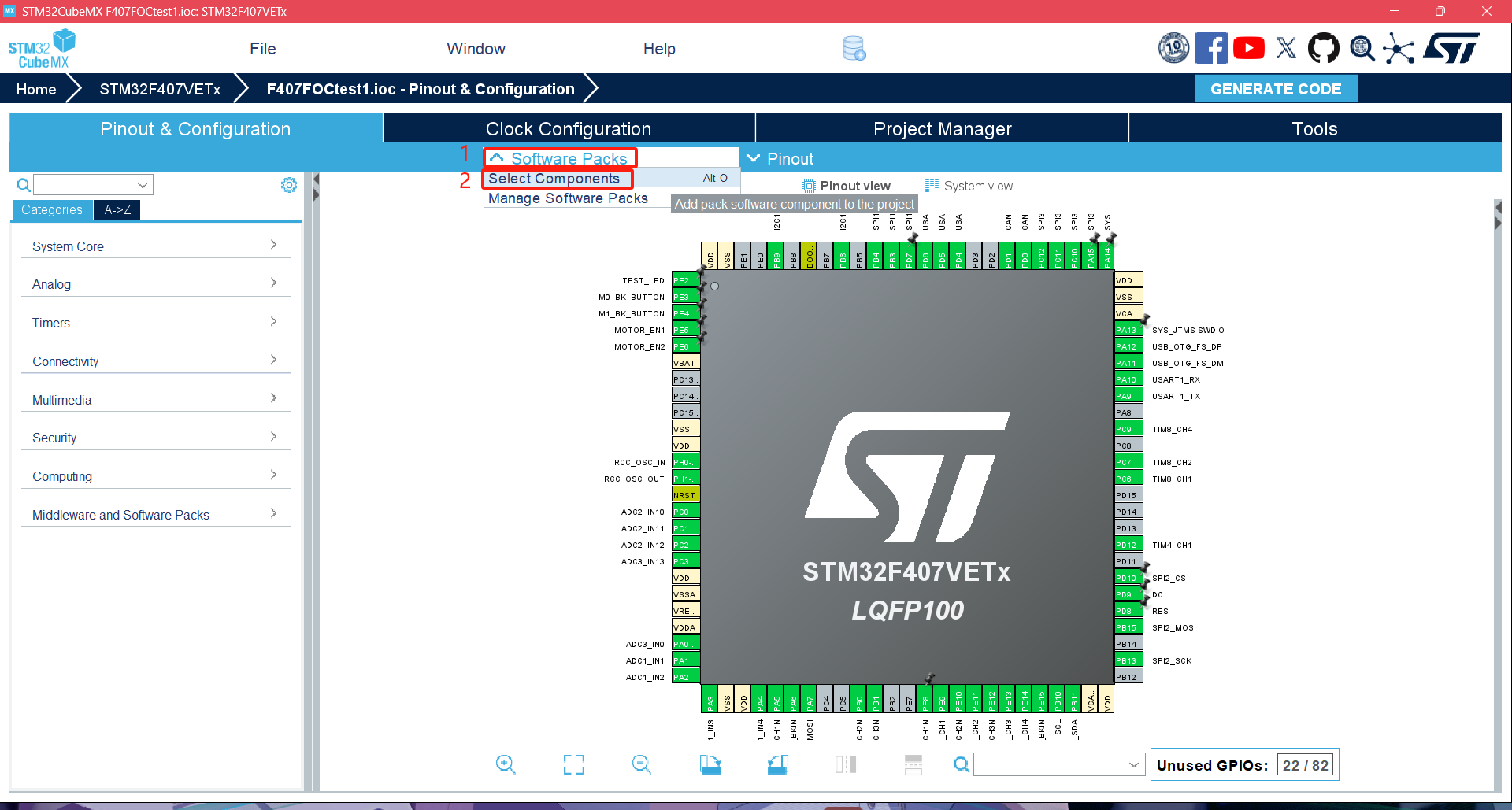
[ADC中断 22](#_Toc193919854)

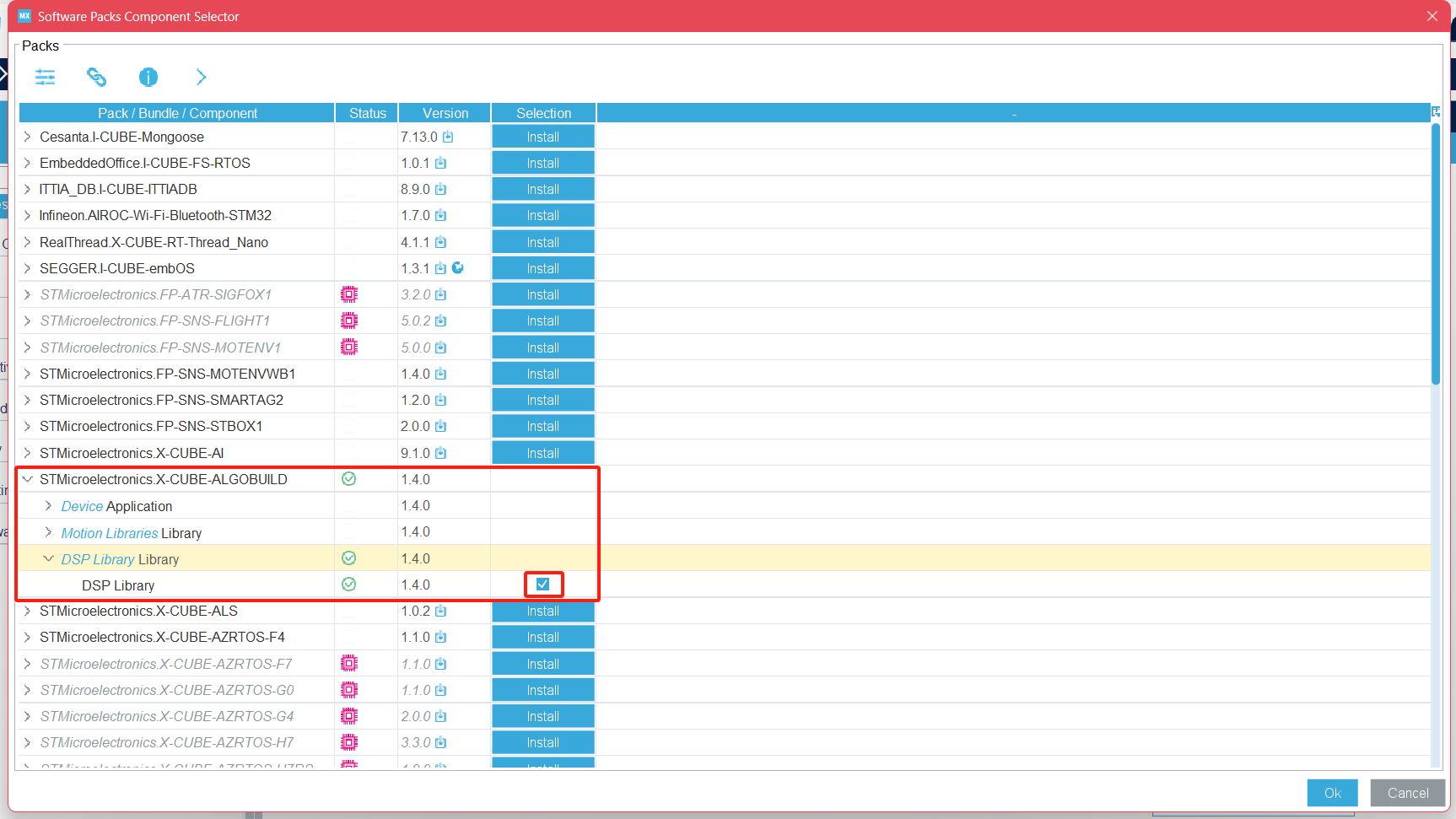
[TFT屏幕（ST7789V） 22](#_Toc193919855)

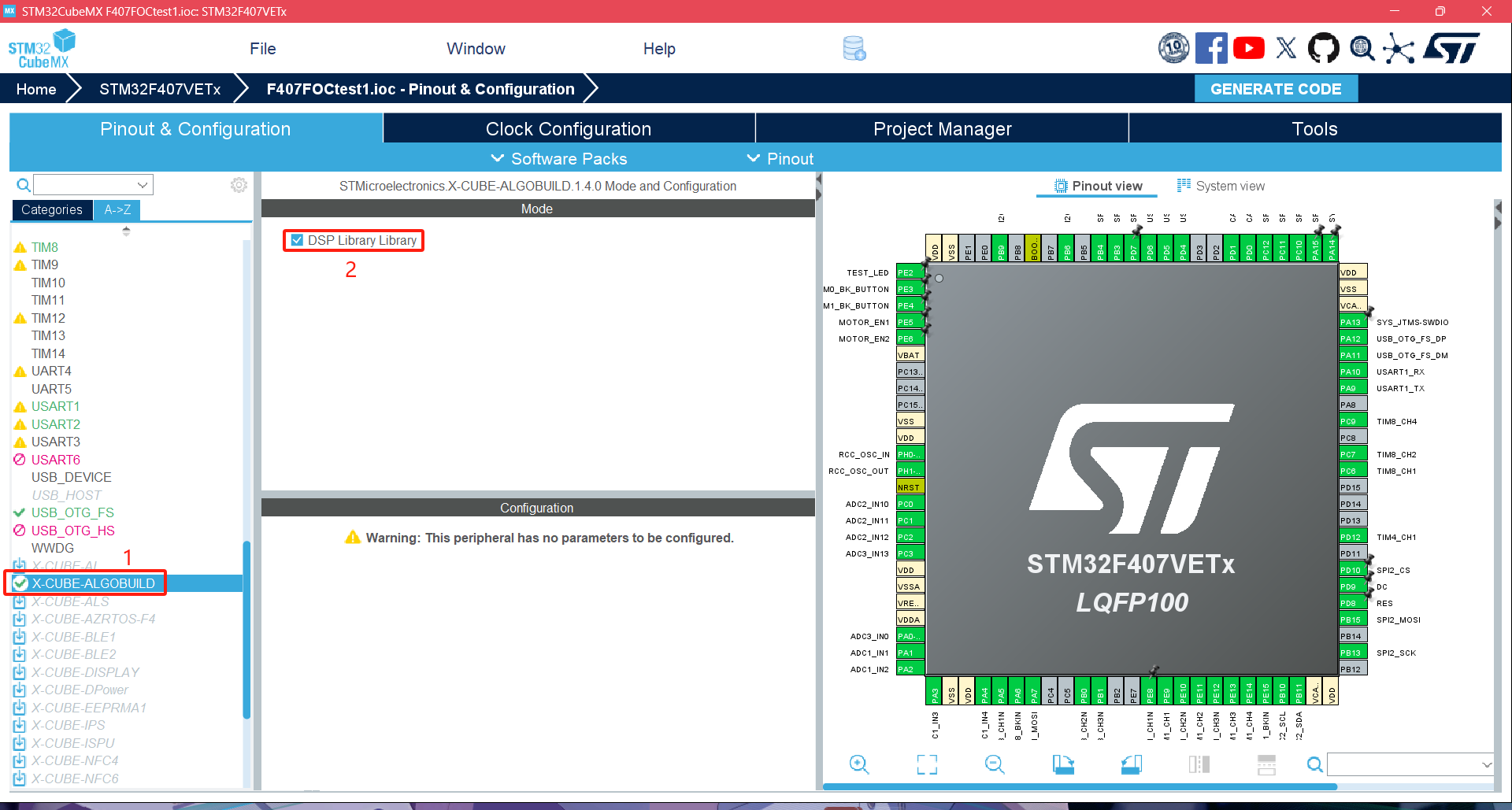
[I2C（AS5600） 26](#_Toc193919856)

[Hal库bug 28](#_Toc193919857)

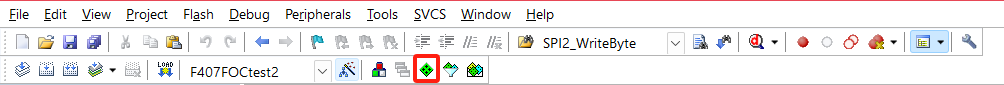
# 引入DSP数学库

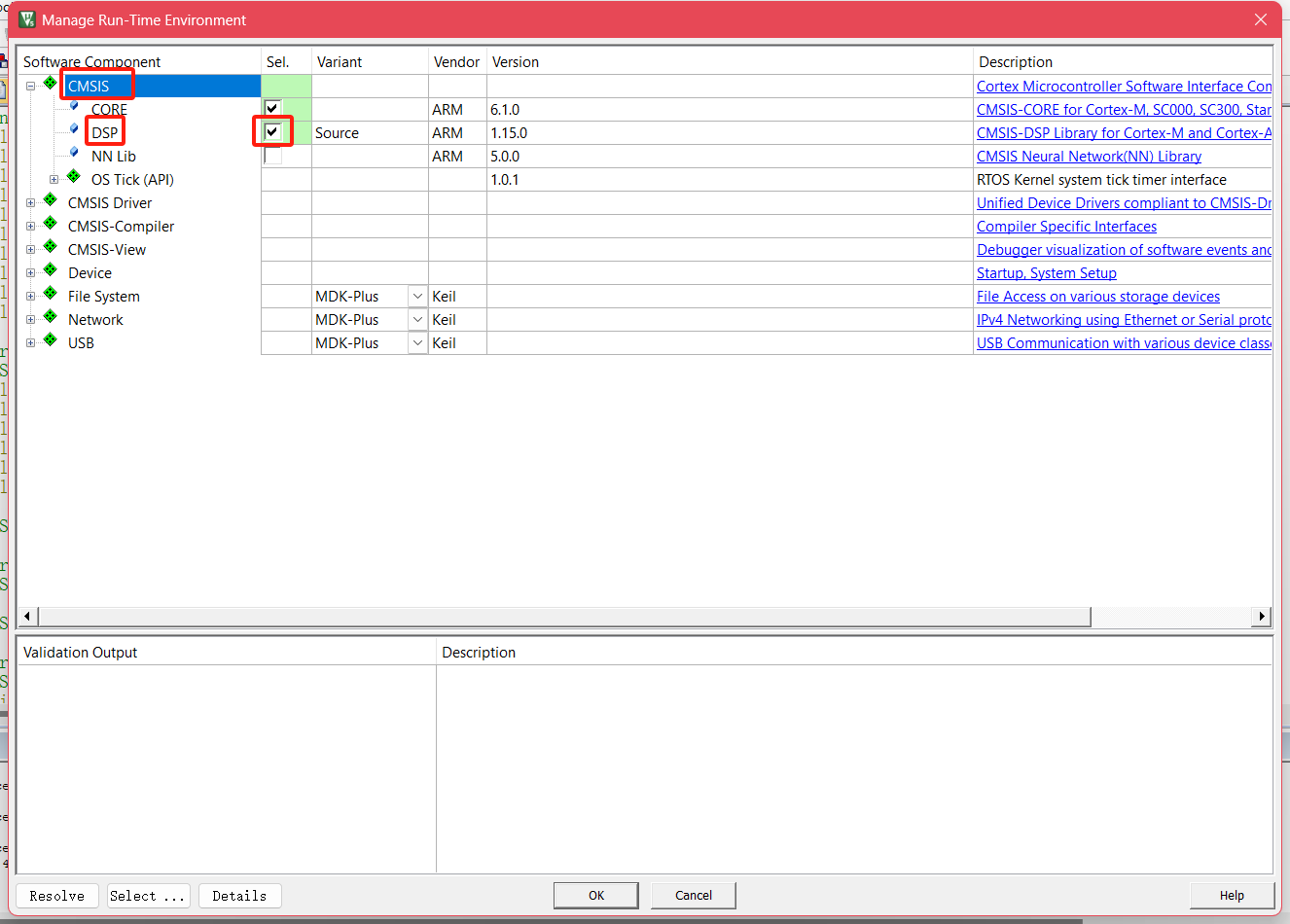




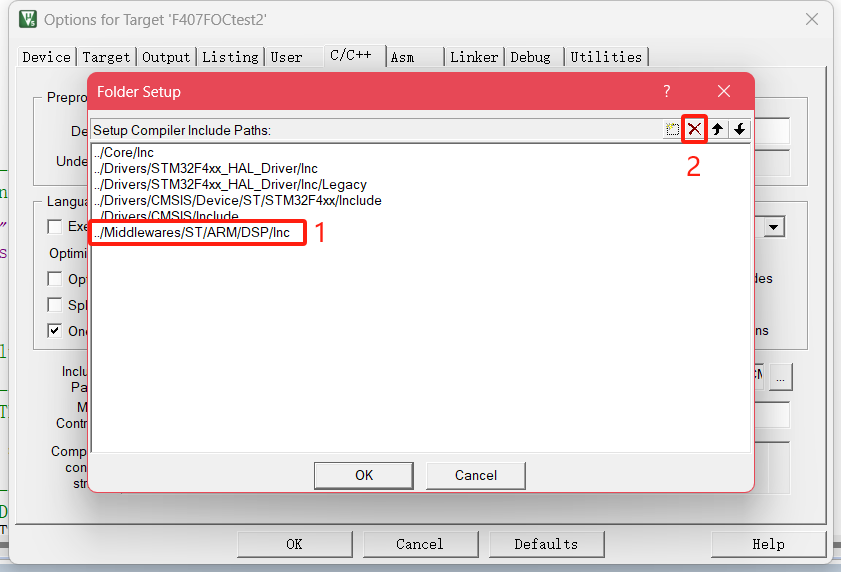


Keil里面：

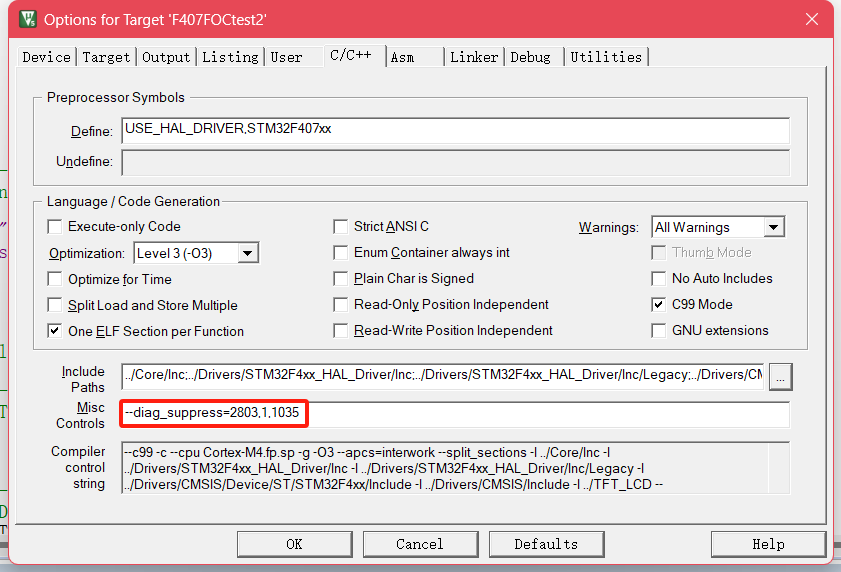




每次从Cubemx生成都要删除掉DSP的这个Path：



魔术棒里加入这句话：--diag\_suppress=2803,1,1035，编译器在编译代码时会忽略低级别的警告。



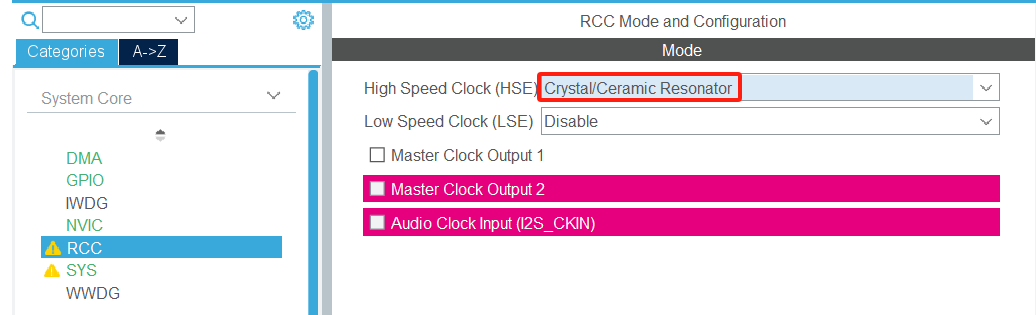
main.c里面添加：

#include "stdio.h"

#include "arm\_math.h"

#include "arm\_const\_structs.h"

# 开启外部晶振



Crystal/Ceramic Resonator晶体/陶瓷谐振器

BYPASS Clock Source旁路时钟源（内部时钟源）

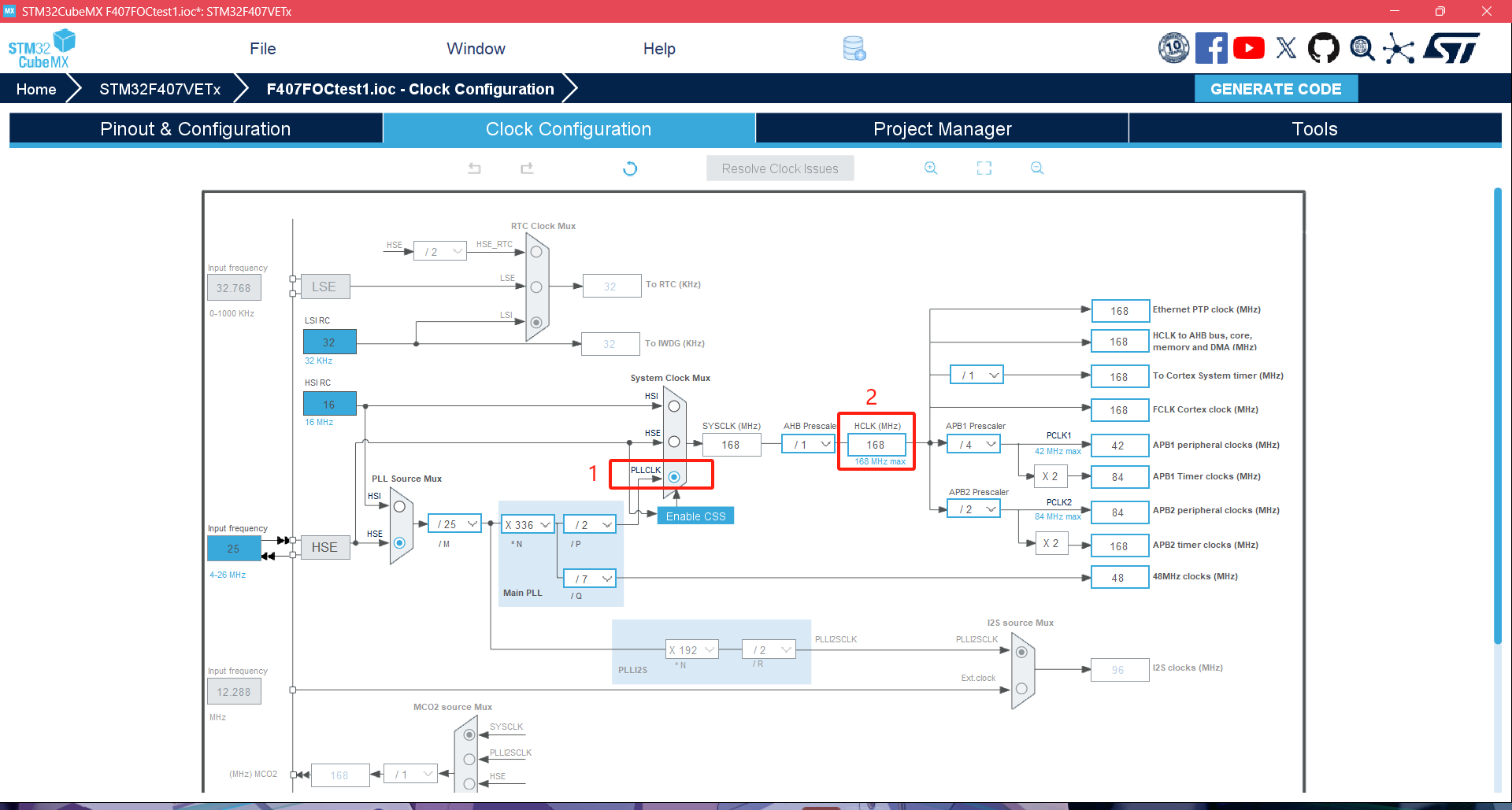


**Master Clock Output**（MCO）允许将内部时钟信号（如系统时钟、PLL时钟或外部晶振时钟）输出到指定的GPIO引脚上。这种功能通常用于为外部设备提供时钟信号，实现时钟同步。

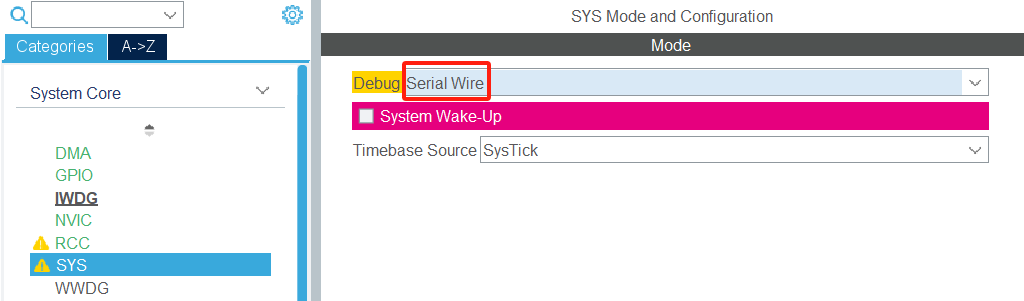
**Audio Clock Input** (2S CkIN) 允许从外部输入音频时钟信号。该功能通常用于I2S（Inter-IC Sound）接口，用于同步音频数据传输。



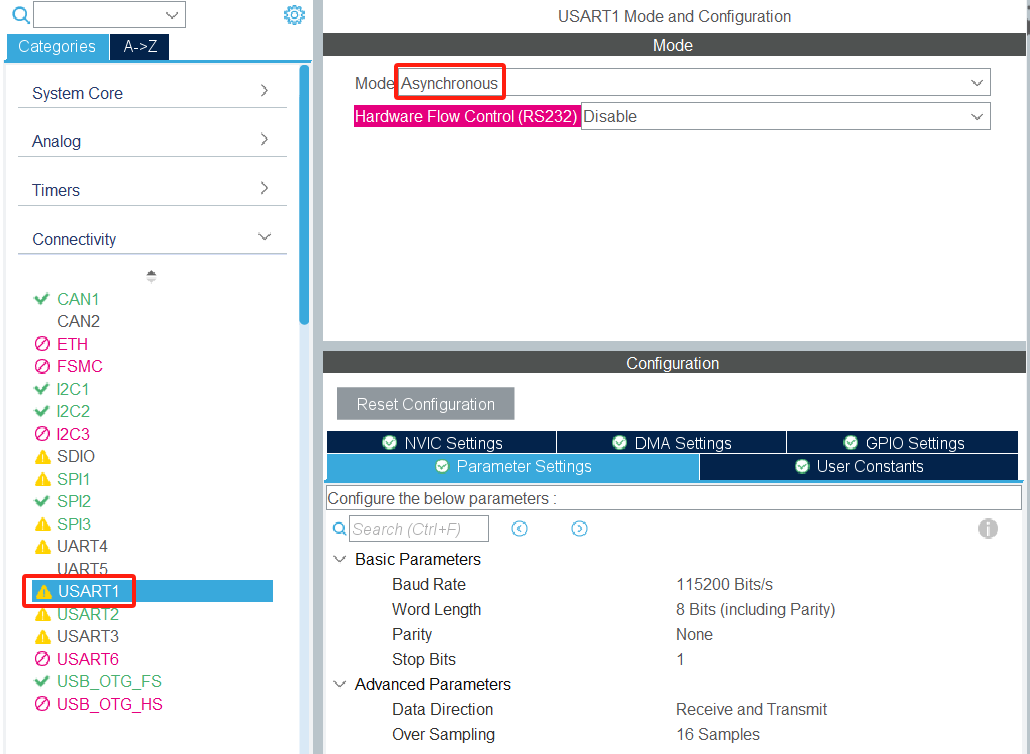
# 时钟源配置



# SWD接口配置



# 串口配置



**Asynchronous** 异步

**Synchronous** 同步

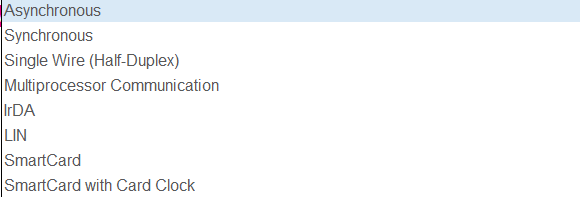
**Single Wire** (Half-Duplex) Multiprocessor Communication单线半双工通信模式，多个处理器或设备共享同一根通信线，数据传输在同一根线上双向进行（半双工），即同一时间只能有一个设备发送数据，而其他设备接收数据。

**IrDA** (Infrared Data Association)红外通信协议

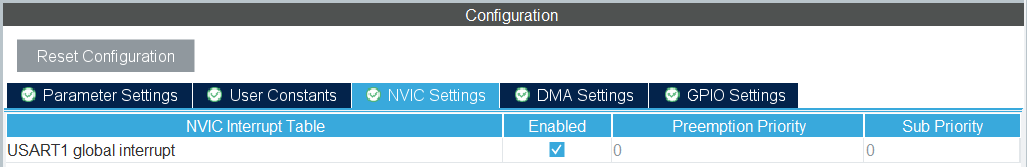
**LIN** (Local Interconnect Network) 串行通信协议，主要用于汽车电子系统中的传感器和执行器通信。LIN协议基于单主多从架构，主节点负责控制通信时序，从节点根据主节点的指令进行数据传输。

**SmartCard** (ISO7816)是一种智能卡接口模式，支持ISO7816标准的接触式智能卡通信。广泛应用于金融、身份认证等领域。

**SmartCard with Card Clock** 增强型的智能卡通信模式，除了支持ISO7816标准的通信功能外，还提供了卡时钟（Card Clock）功能。卡时钟是一种独立的时钟信号，用于为智能卡提供稳定的时钟源。这种模式适用于需要高精度时钟控制的智能卡应用，例如金融交易和安全认证等场景。



开启NVIC



Keil里面，在main.c里面，添加

/\* Private variables------------------------------------------------------------\*/

/\* USER CODE BEGIN PV \*/

uint8\_t rx\_buffer[100]; // 串口缓存

uint8\_t rx\_len; //接收到的数据长度

/\* USER CODE END PV \*/

以及在USART1的中断函数USART1\_IRQHandler里面添加：

void USART1\_IRQHandler(void)

{

/\* USER CODE BEGIN USART1\_IRQn 0 \*/

/\* USER CODE END USART1\_IRQn 0 \*/

HAL\_UART\_IRQHandler(&huart1);

/\* USER CODE BEGIN USART1\_IRQn 1 \*/

if(\_\_HAL\_UART\_GET\_FLAG(&huart1, UART\_FLAG\_IDLE) != RESET)

{

HAL\_UART\_DMAStop(&huart1);

rx\_len = 50 - \_\_HAL\_DMA\_GET\_COUNTER(&hdma\_usart1\_rx);

if(rx\_len > 0)

{

// 创建格式化缓冲区, %.\*s用于指定长度字符串

uint8\_t tx\_buffer[60];

int tx\_len = snprintf((char\*)tx\_buffer, sizeof(tx\_buffer), "%.\*s\r\n", rx\_len, rx\_buffer);

if(tx\_len > 0)

{

HAL\_UART\_Transmit(&huart1, tx\_buffer, tx\_len, 50);

}

Parse\_Command(rx\_buffer, rx\_len);

}

memset(rx\_buffer, 0, 50);

HAL\_UART\_Receive\_DMA(&huart1, rx\_buffer, 50);

\_\_HAL\_UART\_CLEAR\_IDLEFLAG(&huart1);

rx\_len = 0;

}

/\* USER CODE END USART1\_IRQn 1 \*/

}

printf重定义，在usart.c中添加：

/\* USER CODE BEGIN 0 \*/

#include "stdio.h"

/\* USER CODE END 0 \*/

以及

/\* USER CODE BEGIN 1 \*/

/\*\*

\* 函数功能: 重定向c库函数printf到DEBUG\_USARTx

\* 输入参数: 无

\* 返 回 值: 无

\* 说 明：无

\*/

int fputc(int ch, FILE \*f)

{

HAL\_UART\_Transmit(&huart1, (uint8\_t \*)&ch, 1, 0xffff);

return ch;

}

/\*\*

\* 函数功能: 重定向c库函数getchar,scanf到DEBUG\_USARTx

\* 输入参数: 无

\* 返 回 值: 无

\* 说 明：无

\*/

int fgetc(FILE \*f)

{

uint8\_t ch = 0;

HAL\_UART\_Receive(&huart1, &ch, 1, 0xffff);

return ch;}

/\* USER CODE END 1 \*/

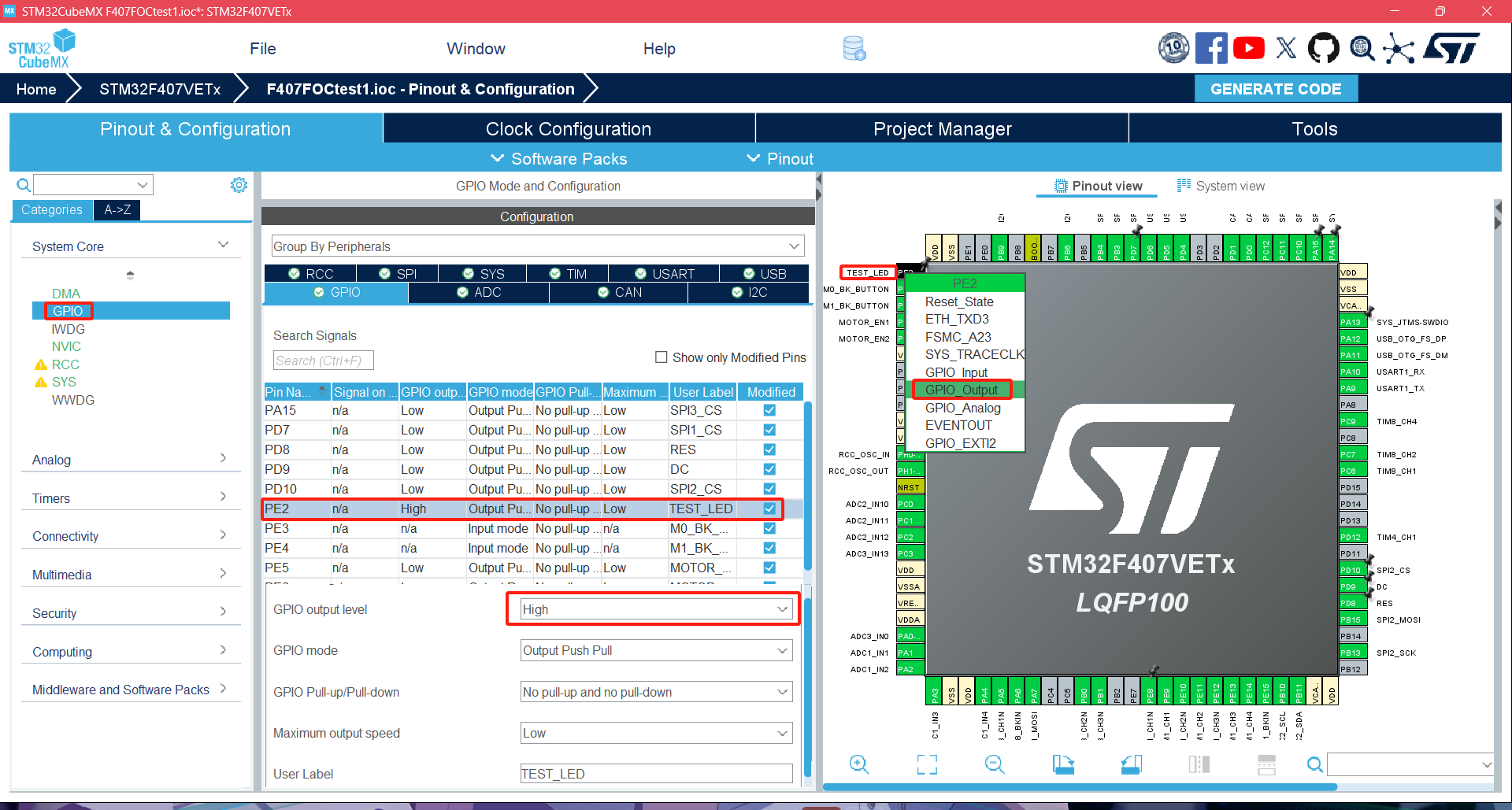
在while测试：printf("串口测试\r\n");

随机找个串口助手发送“123qwe测试”，接受区可收到返回信息。



# LED灯配置

由于硬件的LED正极接3.3V，所以GPIO初始化为High



# 定时器配置

TIM1、TIM8

**Channel1、2、3为**PWM互补输出，Channel4为舵机PWM输出

**PSC**：1 - 1，Timer的运行频率为168MHz。

**Counter Mode**：中心对齐方式1。

**Dithering**：PWM抖动模式，在连续的PWM周期内稍微改变CCR或ARR的实际值，从而实现更高的分辨率。例如，STM32微控制器的Dithering模式可以在16个连续的PWM周期内，通过增加或减少一个定时器时钟周期来调整占空比，从而将分辨率提高16倍。

**Counter Period**：计数周期，PWM频率为15kHz，在中心对齐模式下，计数最高点为（168MHz / 15kHz / 2）- 1 =5600 – 1。

**CKD**：用于设置定时器时钟频率的分频系数，从而影响定时器的计数频率（只影响输入捕获）。

**RCR**：重复计数器设置为0，即每1个定时器溢出产生1个更新事件，更新事件会触发ADC采样， FOC代码放在ADC采样完成中断里计算的，计算频率等于15KHz。

**auto-reload preload**：自动重装载寄存器，启用预加载功能，提高信号的稳定性和实时性。

**Master/Slave Mode**：不使用主从模式。

**Trigger Event Selection**（TRGO）：选择Update Event（UEV），当定时器计数器溢出或更新时，产生触发输出。

**BRK State**：刹车功能开启。

**BRK Polarity**：当刹车输入信号为低电平的时候，触发刹车功能。

**Automatic Output State**：刹车信号失效后是否自动恢复输出。

**Lock Configuration**：用于保护定时器的某些关键寄存器，防止在运行过程中被意外修改，从而提高系统的可靠性和安全性。LOCKLEVEL\_1：锁定死区时间寄存器（DTG）和断路功能寄存器（BKE和BKP）。LOCKLEVEL\_2：进一步锁定OSSR（运行模式下的关闭状态选择）和OSSI（空闲模式下的关闭状态选择）状态寄存器。LOCKLEVEL\_3，进一步锁定 AOE（自动输出使能） 和 MOE（主输出使能） 状态寄存器

**Dead Time**：死区时间，TDTS = 168M / 1000ns = 5.95ns，要设置1000ns的死区时间，计算DTG寄存器的值： DTG=1000ns​ / 5.95ns ≈ 168。

**Mode**：PWM Mode 1，计数值Counter小于CCR时PWM输出有效电平。

**Pulse**：设置CCR寄存器的值（代码手动设置）。

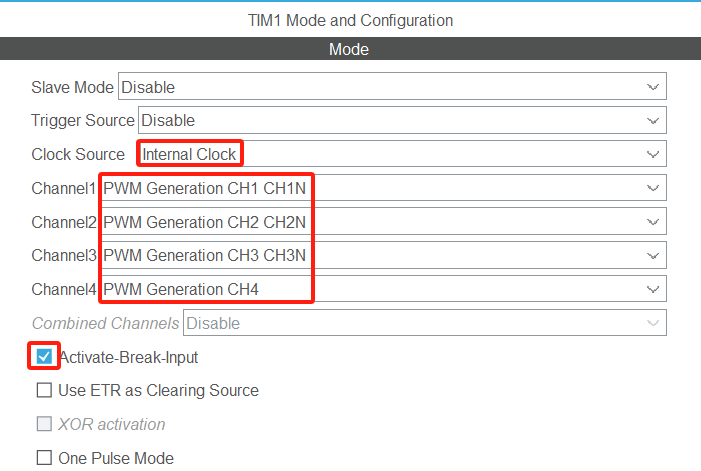
**Output compare preload**：CCR预装载使能，可以确保PWM信号的占空比在每个周期内保持一致。

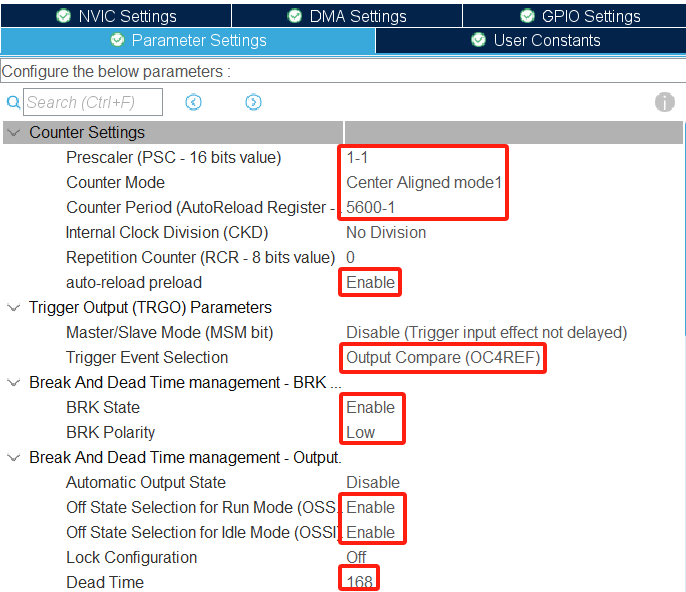
**Fast Mode**：快速输出管脚电平，不启用。

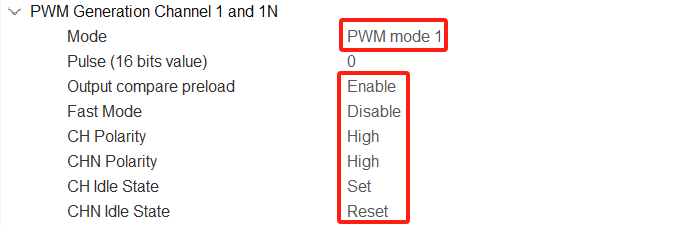
**CH/CHN Polarity**：CH/CHN有效电平的极性，根据预驱动芯片手册时序图，分别设置为High/Low。

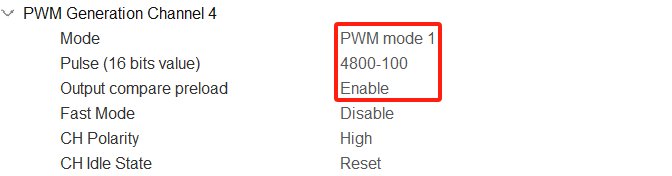
**CH/CHN ldle State**：当PWM不输出或空闲时的状态，SET为高电平，RESET为低电平。

\*\*\*对于PWM Channel4不再赘述。\*\*\*

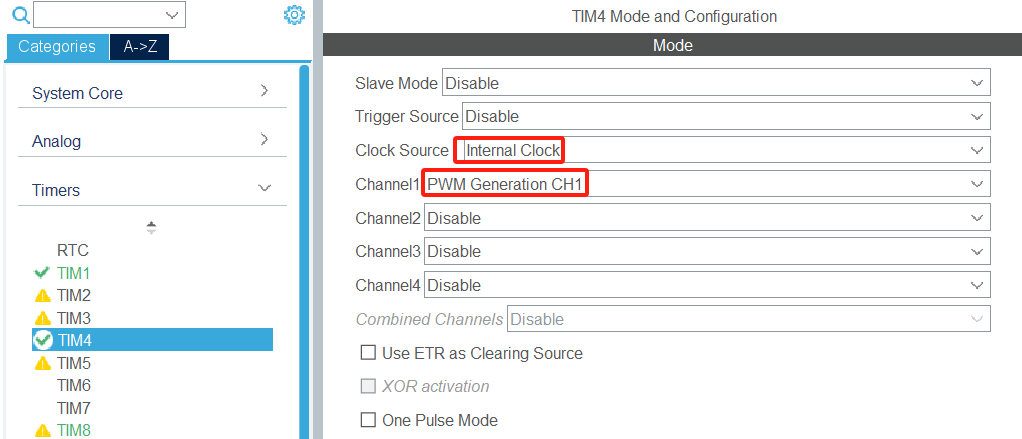


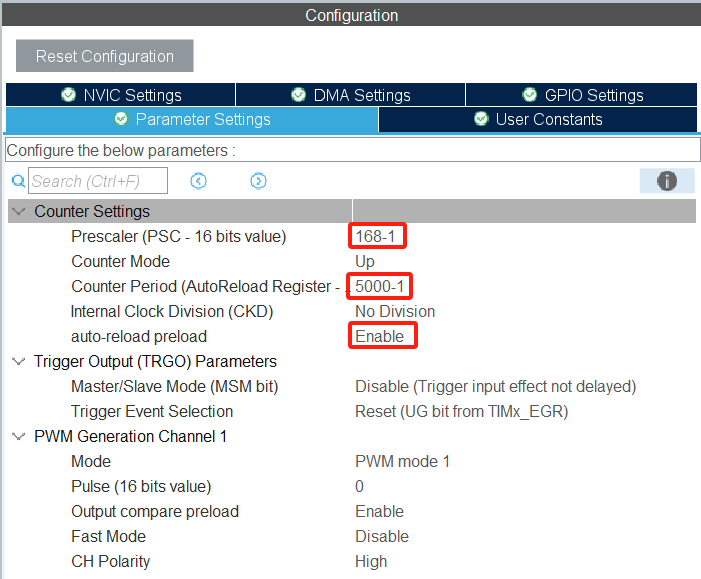






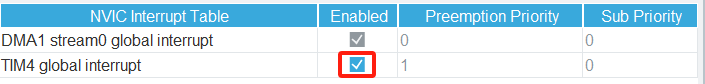
TIM4（用于屏幕的调光）：

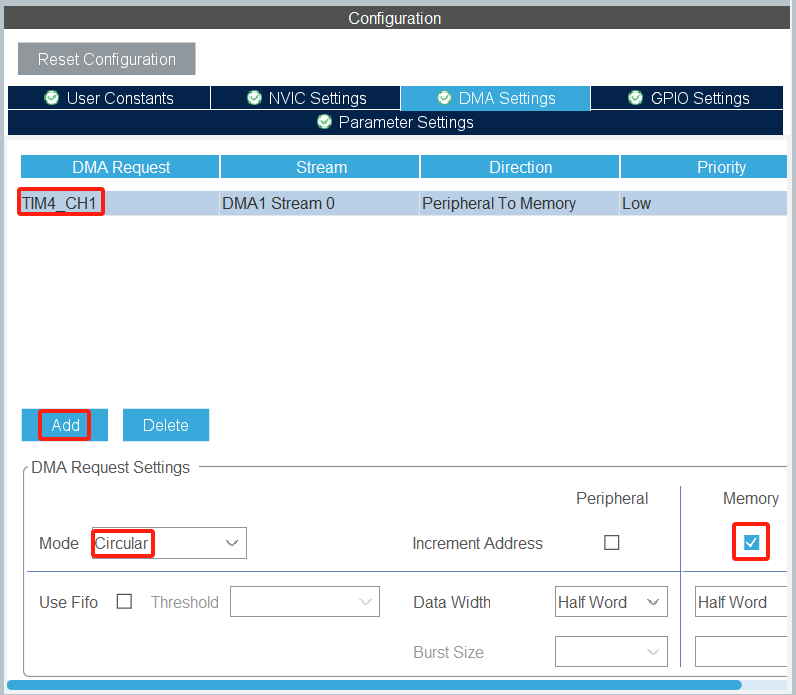




定时器溢出频率为：168M / 168 / 5000 = 200Hz，即5ms。

打开NVIC、DMA。





# ADC配置

ADC1、ADC2、ADC3：IN2/3/4用于M0；IN10/11/15用于M1；IN1用于电压检测；IN0用于TFT屏幕；IN12/13用于温度检测。

**Mode**：独立模式。

**Clock Prescaler**：4分频。

**Resolution**：12bits的分辨率。

**Data Alignment**：右对齐。

规则组：

**Scan Conversion Mode**：规则采样使用连续扫描模式。

**ContinuousConversion Mode**：不连续转换，单次转换。

**Discontinuous Conversion Mode**：不连续转换。

**DMA Continuous Requests**：允许在一次DMA传输完成后自动请求新的传输，配置为不允许。

**End Of Conversion Selection**：选择End of sequence of conversion，会在一组数据转换完成后发出EOS标志。

**Number Of Conversion**：规则组通道数量，1个通道。

**External Trigger Conversion Source**：软件触发。

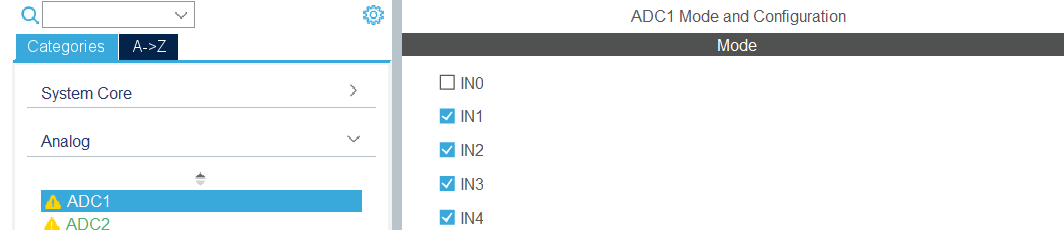
**External Trigger Conversion Edge**：软件触发，None。

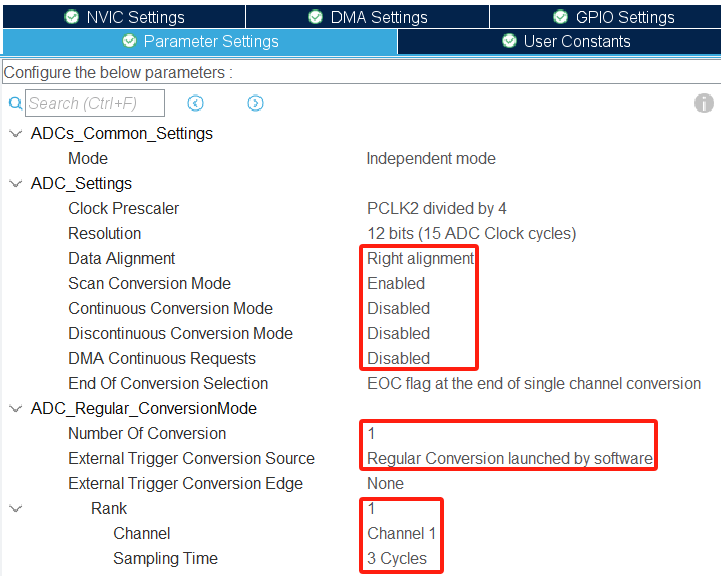
**Channe**：Channe 4。

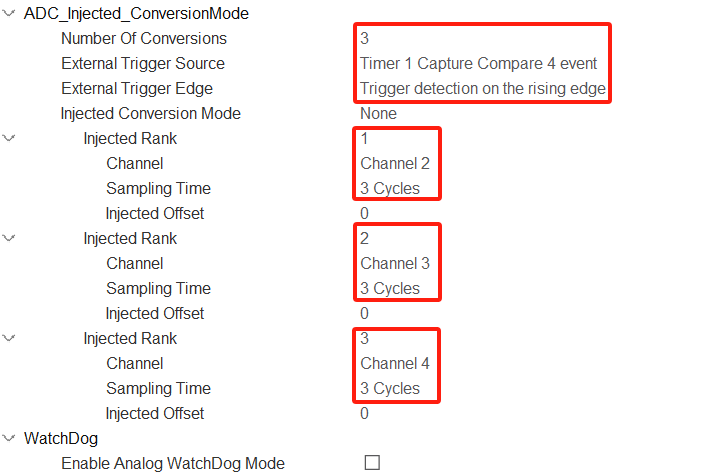
**Sampling Time**：3个周期。

注入组：

3个通道，Timer1触发事件，上升沿触发，不开启注入组转换模式，采样时间3个周期。

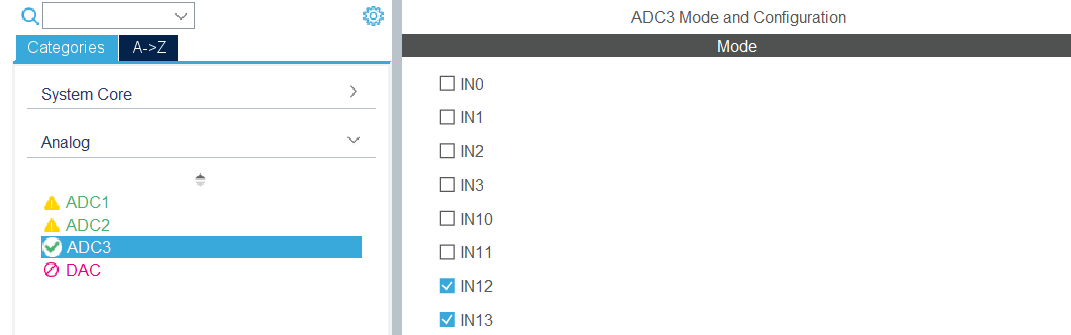


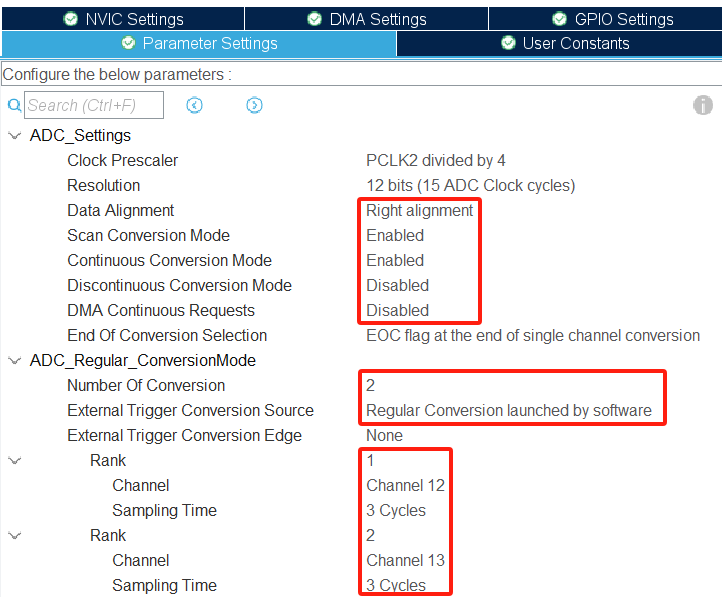




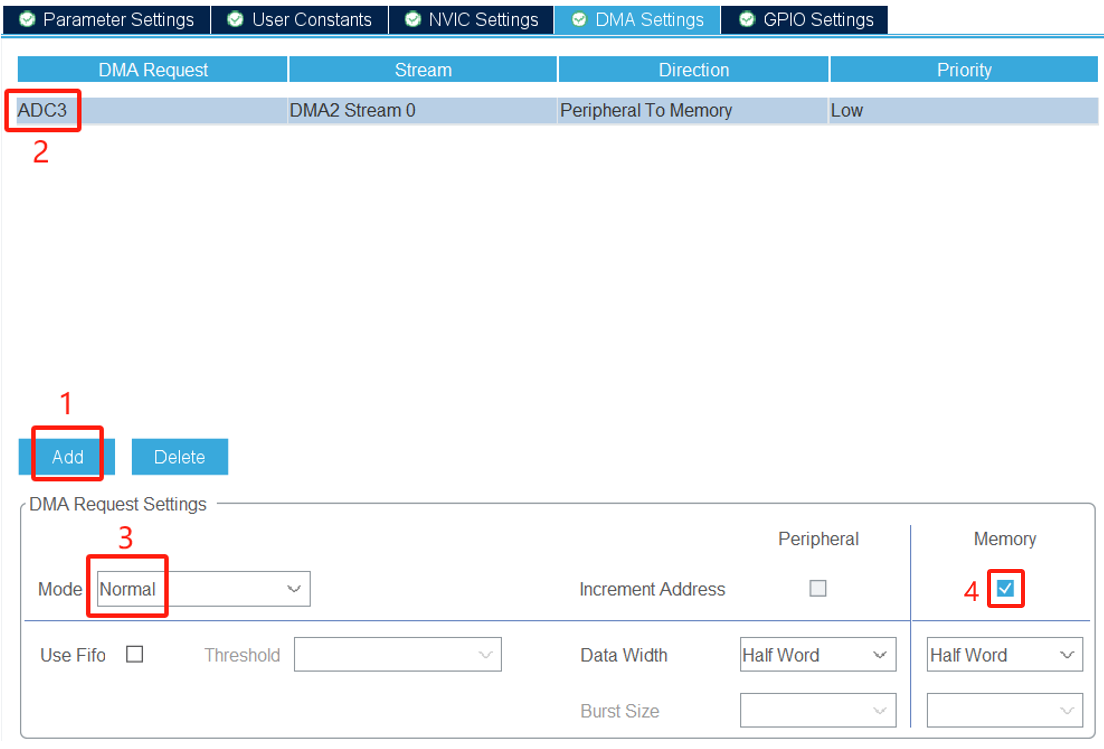


ADC1（ADC2和ADC1类似，不再赘述）



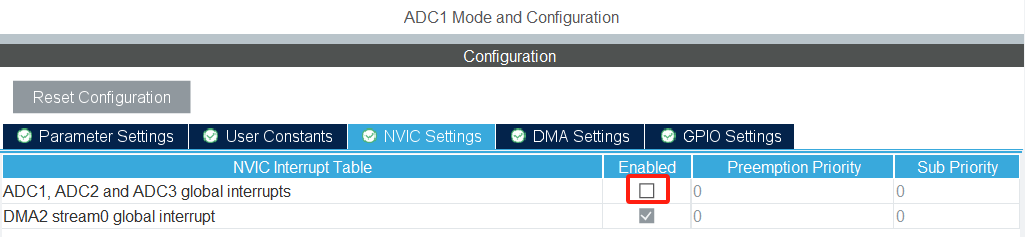


ADC3

打开ADC3的DMA。循环模式；Memory打勾，存储ADC值的内存地址自增。

# ADC中断

打开ADC中断，需要在ADC中断中完成FOC计算。



# TFT屏幕（ST7789V）

**Mode**：作为主机传输。

**Frame Format**：帧格式。Motorola SPI（大端模式）协议中，高字节存储在低地址，低字节存储在高地址。TI SPI（小端模式）协议中，低字节存储在低地址，高字节存储在高地址。

**Data Size**：数据大小，指定每个数据帧包含的位数，8位（1字节）。

**First Bit**：首比特，指定数据传输时最先发送的是最高有效位（MSB）还是最低有效位（LSB）。

**Prescaler (for Baud Rate)**：波特率预分频器，用于设置SPI的时钟频率。

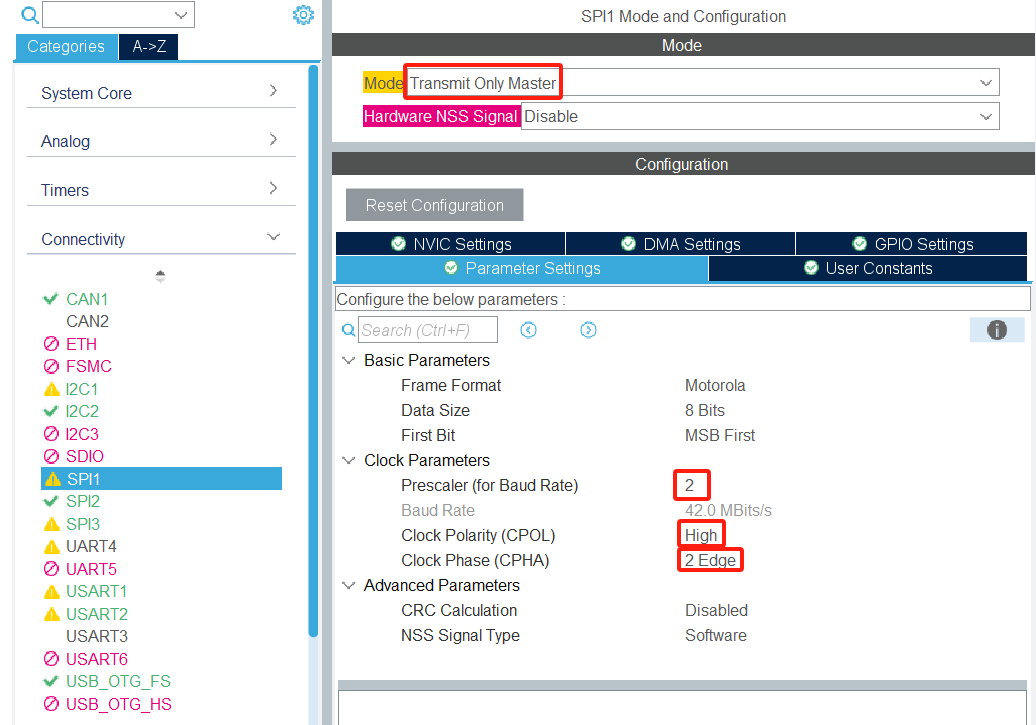
**Baud Rate**：波特率。

**Clock Polarity (CPOL)**：时钟极性，指定时钟信号的空闲状态是高电平还是低电平。设置为High，空闲状态为高电平。

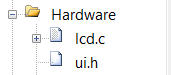
**Clock Phase (CPHA)**：时钟相位，指定数据是在时钟信号的哪个边沿被采样。2 Edge，数据在时钟信号的第二个边沿（上升沿）被采样，提高数据的稳定性。

**CRC Calculation**：CRC计算，指定是否启用循环冗余校验（CRC）来检测数据传输中的错误。不启用CRC计算。

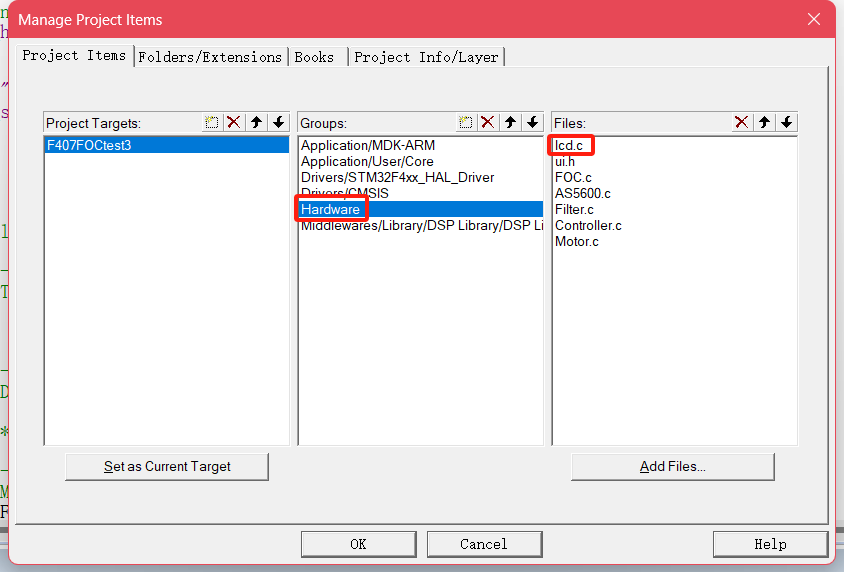
**NSS Signal Type**：NSS信号类型，NSS（从设备选择）信号可以由硬件或软件控制。设置为Software，NSS信号由软件控制。



加载lcd.c、lcd.h、ui.h等文件。



Group里添加Hardware，再把lcd.c、lcd.h导入进去，再进行编译。编译成功后在lcd.c的文件夹里面会出现lcd.h。把Hardware里面多余的的lcd.h删掉即可。



.c文件定义LCD\_TOTAL\_BUF\_SIZE 240\*135\*2，2表示个像素用2字节表示，用于16位颜色深度。LCD\_Buf\_Size定义为648，即240\*135\*2的1/100，spi传输100次刚好传输完。

#define LCD\_TOTAL\_BUF\_SIZE (240\*135\*2)

#define LCD\_Buf\_Size 648

.h文件定义屏幕分辨率大小240\*135

#define LCD\_Width 240

#define LCD\_Height 135

main.c文件里添加：

#include "ui.h"

以及

/\* USER CODE BEGIN 4 \*/

uint8\_t SPI2\_WriteByte(uint8\_t\* data, uint16\_t size)

{

return HAL\_SPI\_Transmit(&hspi2, data, size, 0xff);

}

/\* USER CODE END 4 \*/

以及main函数里添加下面函数即可进行测试：

/\* USER CODE BEGIN 2 \*/

LCD\_Init();

LCD\_Clear(BLACK);

HAL\_Delay(1000);

LCD\_Show\_Image(0, 0, 240, 135, fixed\_graphic);

/\* USER CODE END 2 \*/

# I2C（AS5600）

**Speed Mode**：选择I2C通信的速度模式。这里设置为Standard Mode速度为100kHz。

**I2C Clock Speed (Hz)**：设置I2C总线的时钟速度。这里设置为100000Hz，即100kHz。

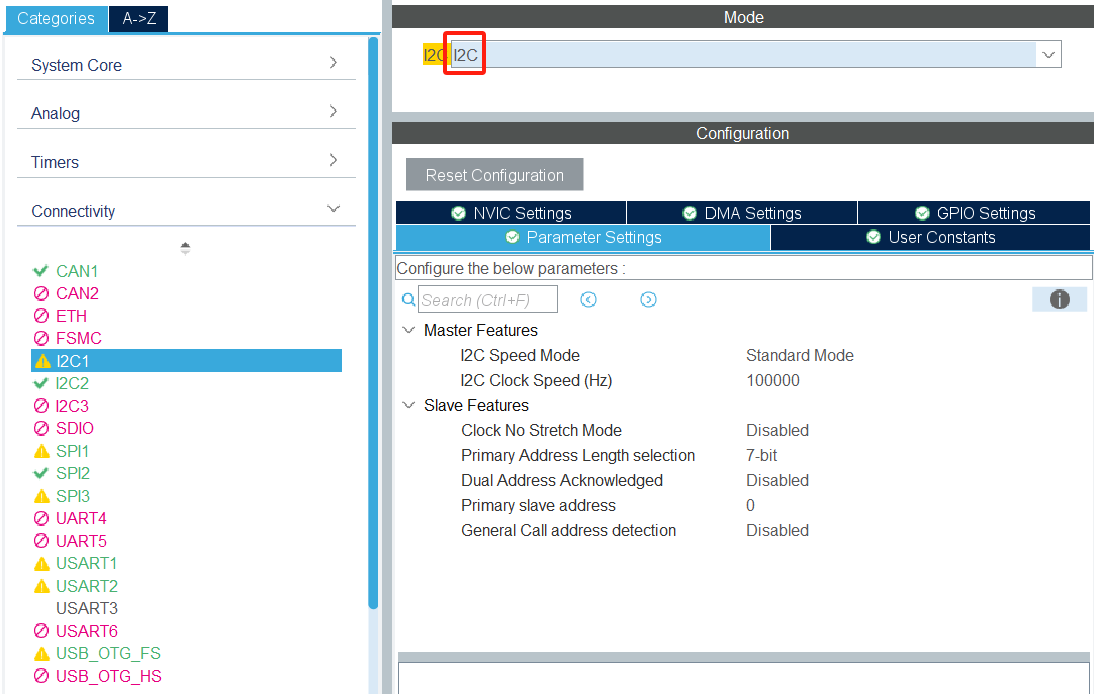
**Clock No Stretch Mode**：时钟拉伸模式，允许从设备在需要时钟信号上添加额外的等待时间，以减慢主设备的读取速度。这里设置为Disabled，不使用时钟拉伸模式。

**Primary Address Length selection**：选择从设备的地址长度。这里设置为7-bit，表示从设备的地址为7位长。

**Dual Address Acknowledged**：允许从设备有两个地址，主设备可以通过任一地址访问它。这里设置为Disabled，表示不使用双地址。

**Primary slave address**：设置从设备的主地址。这里设置为0，表示从设备的地址为0x00。

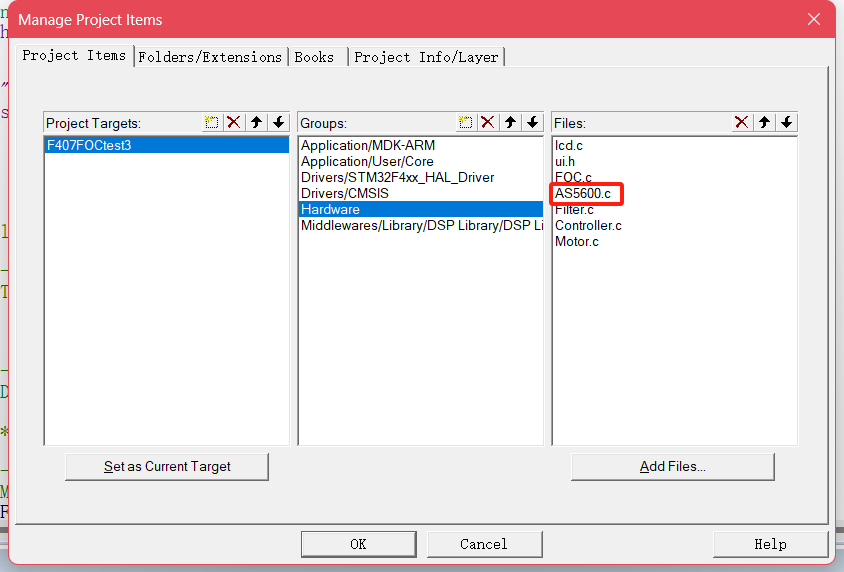
**General Call address detection**：通用调用地址检测，主设备可以向所有从设备发送通用调用。这里设置为Disabled，表示不检测通用调用地址。



加载AS5600.c、AS5600.h等文件。



把AS5600.c、AS5600.h导入进去，再进行编译。编译成功后在AS5600.c的文件夹里面会出现AS5600.h。把Hardware里面多余的的AS5600.h删掉即可。



\*\*\*对于I2C2不再赘述。\*\*\*

至此，基本外设已经配置完成

# Hal库bug

进入tim.c，找到：

\_\_HAL\_TIM\_CLEAR\_FLAG(&htim1, TIM\_FLAG\_UPDATE);

\_\_HAL\_TIM\_CLEAR\_FLAG(&htim8, TIM\_FLAG\_UPDATE);

\_\_HAL\_TIM\_CLEAR\_FLAG(&htim6, TIM\_FLAG\_UPDATE);

将两者分别移动到

/\* USER CODE BEGIN TIM1\_Init 0 \*/

/\* USER CODE END TIM1\_Init 0 \*/

以及

/\* USER CODE BEGIN TIM8\_Init 0 \*/

/\* USER CODE END TIM8\_Init 0 \*/

以及

/\* USER CODE BEGIN TIM6\_Init 0 \*/

/\* USER CODE END TIM6\_Init 0 \*/

进入i2c.c，找到

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();（I2C1和I2C2都要）

将两者分别移动到

/\* USER CODE BEGIN I2C1\_MspInit 0 \*/

/\* USER CODE END I2C1\_MspInit 0 \*/

以及

/\* USER CODE BEGIN I2C2\_MspInit 0 \*/

/\* USER CODE END I2C2\_MspInit 0 \*/