一步一步创建 STM32WBA BLE_Custom 工程

关键字：STM32WBA，BLE，定制化

1. 介绍

STM32WBA 是 ST 最新一代的 BLE 芯片。该芯片已经获得 STM32CubeMX 工具的支持，用户可使用 STM32CubeMX 的图形化界面、快速生成具备 IO 配置、时钟配置、外设配置、中间件使用配置、BLE 服务配置等内容的基础源码工程。

本文档将指导用户如何使用 STM32CubeMX 软件一步步生成基于 STM32WBA52 MCU 的一个源码工程；该工程实现了 BLE 外设功能、并具有定制化的 BLE 服务。使用 STM32WBA 系列其他芯片也可以参考此文档的步骤构建对应的基础源码工程。

2. 使用 CubeMX 在 STM32WBA 上构建 Bluetooth® Low Energy 应用

将 STM32CubeMX 生成的源码工程进行简单修改后，便可运行在 STM32WBA Nucleo 板上、实现 BLE 外设功能。

BLE 外设充当 GATT 服务器角色，手机 APP（ST BLE ToolBox）充当 GATT 客户端角色，他们之间可以进行数据收发。

图1. 通信示意图

3. 服务和特征配置

该 GATT 服务器公开了一个定制化的服务（SerialPortService）并包含三个特征：

- 特征 1 具有 Notify 属性，可发送数据给手机
- 特征 2 具有 Write without response 属性，可接收手机的数据
- 特征 3 具有 Read 属性，可接受手机的读请求并发送响应数据包

服务器的服务和特征配置列举如下：

表1. 服务和特征配置

Service Long Name	SerialPortService		
Service Short Name	SP_Service		
UUID Type	128 bits		
UUID	FFF0		
Characteristic Long Name	TX_Characteristic	RX_Characteristic	ReadCharacteristic
Characteristic Short Name	TX_C	RX_C	Read_C
UUID Type	128 bits	128 bits	128 bits
UUID	FFF1	FFF2	FFF3
Char Properties	Notify	Write without response	Read
Char Permissions	None	None	None

4. 工具

4.1. 软件工具

要制作和使用这个项目，完成应用程序所需的软件工具是：

- STM32CubeMX 软件 (v6.8.0 以上)
- STM32CubeWBA MCU Package (v1.0.0 以上)
- IDE: STM32CubeIDE or IAR
- 串口终端 (TeraTerm)
- 手机应用程序 STBLEToolbox

4.2. 硬件工具

还需要一个 STM32WBA Nucleo 板和一个 micro-B 到 Type-A USB 电缆。

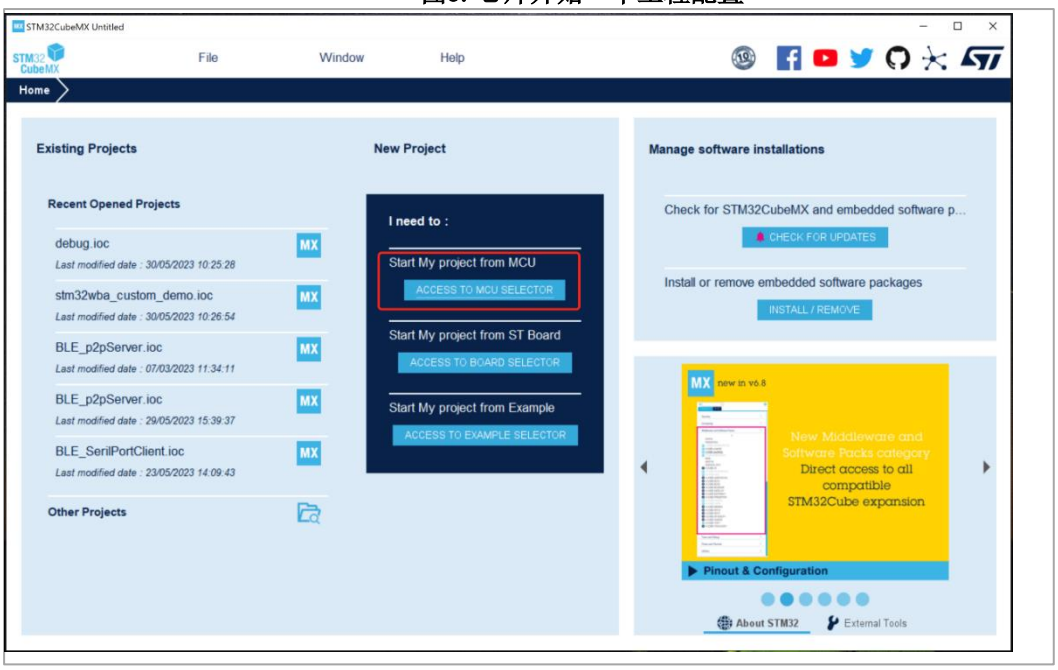
图2. 硬件工具

5. NUCLEO-WBA52CG 的 CubeMX 初始化

5.1. CubeMX 初始化

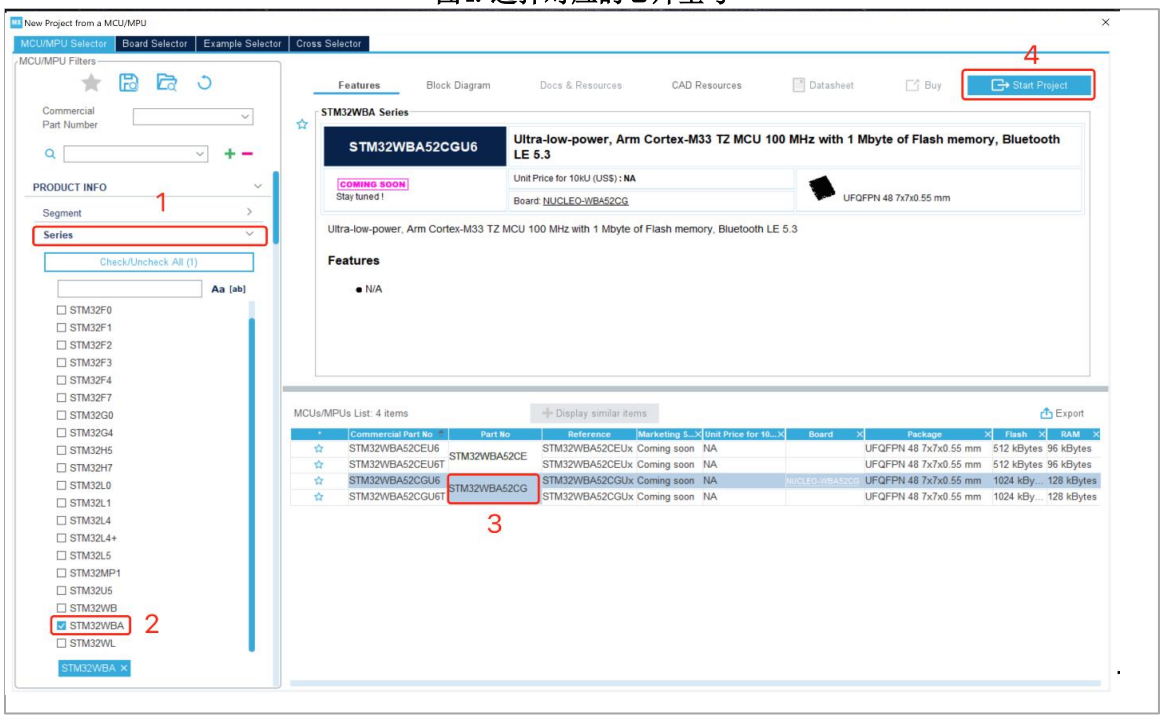
打开 CubeMX 软件及 MCU 选择器，打开的过程可能会进行数据库更新，耐心等待更新完成即可。另外，确保 CubeMX 是 6.8.0 以上版本，否则将找不到 STM32WBA 系列芯片。

图3. 芯片开始一个工程配置



进入 MCU 选择器页面后，按照下图的步骤选中我们需要的开发板。

图4. 选择对应的芯片型号



1. 选择系列分类

2. 选择 STM32WBA 系列
3. 选择对应的 STM32WBA 芯片型号
4. 选择 Start Project

图5. 根据应用需求选择是否使能 TrustZone

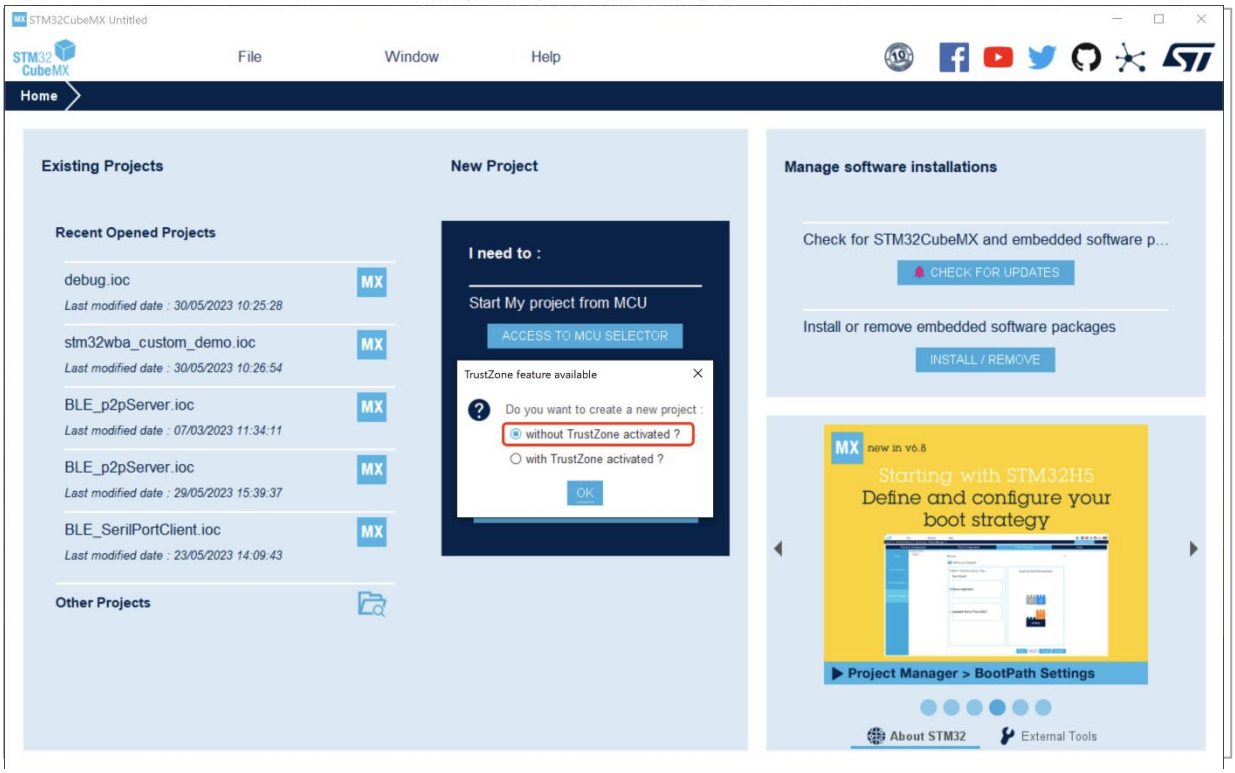
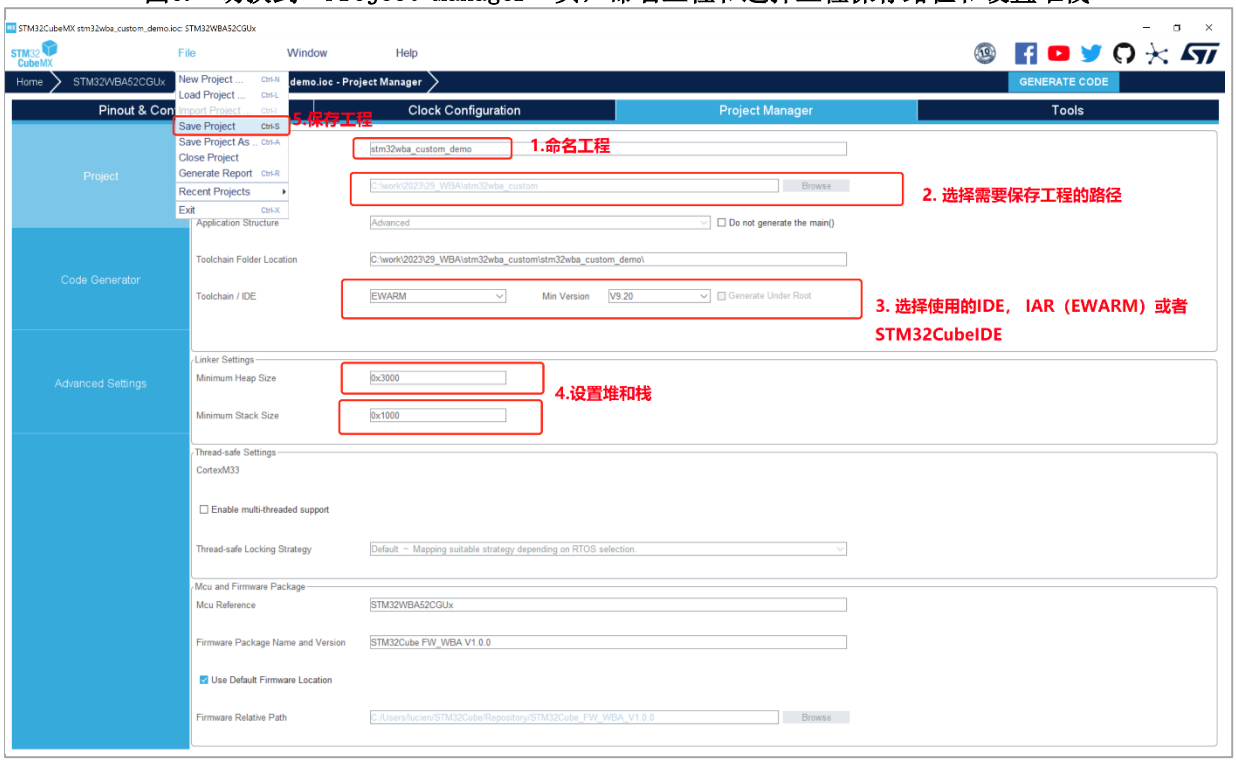


图6. 切换到“Project Manager”页，命名工程和选择工程保存路径和设置堆栈

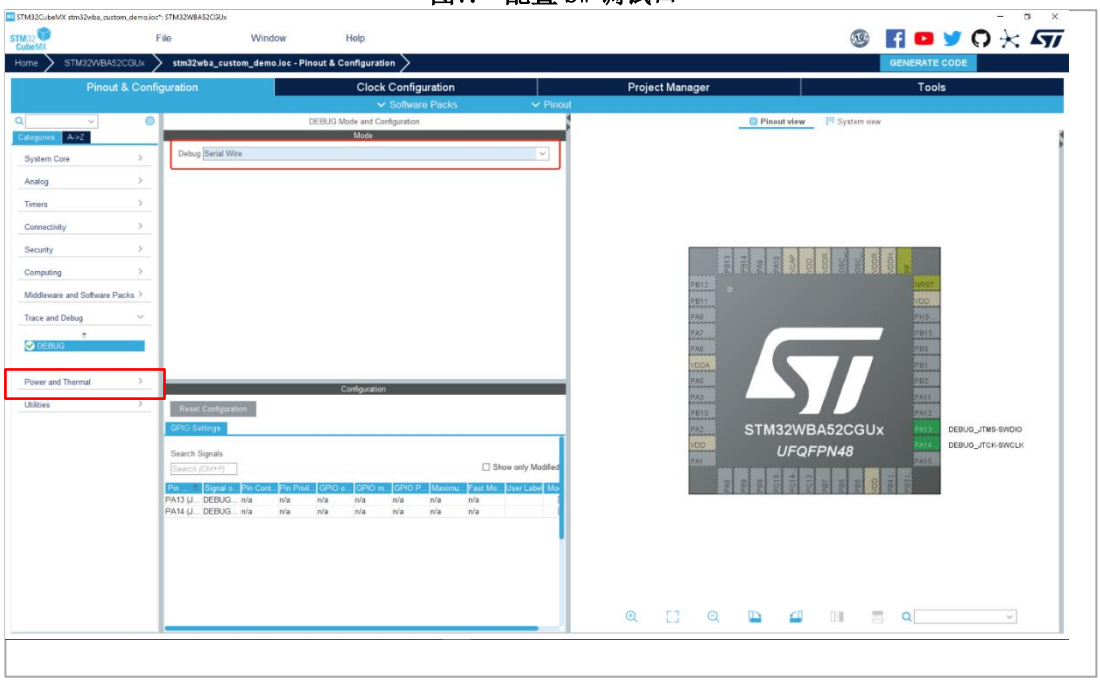


用户可以根据自己的应用需求对工程名称路径，IDE 类型以及堆和栈等进行设置，上图给了一个参考示例。配置完成后，选中 File > Save Project 以保存工程配置（保存为.ioc 文件）。

5.2. 基本外设的配置

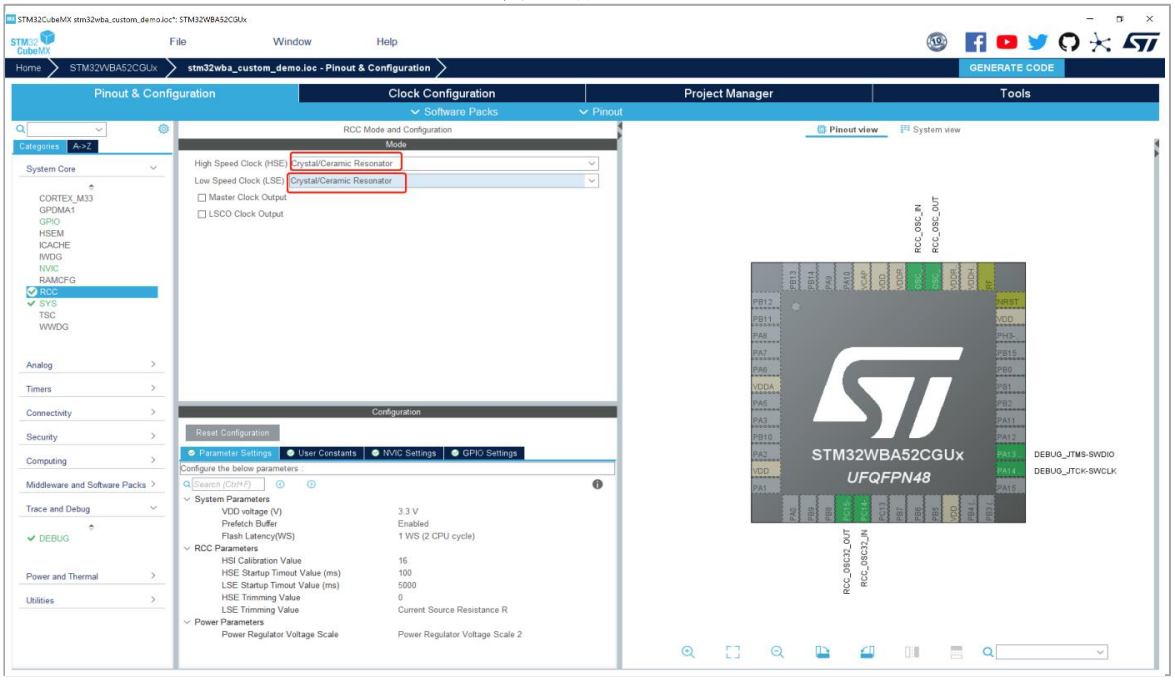
5.2.1. 切换到配置 SW 调试口

图7. 配置 SW 调试口



5.2.2. 配置基本 RCC

图8. 配置 RCC

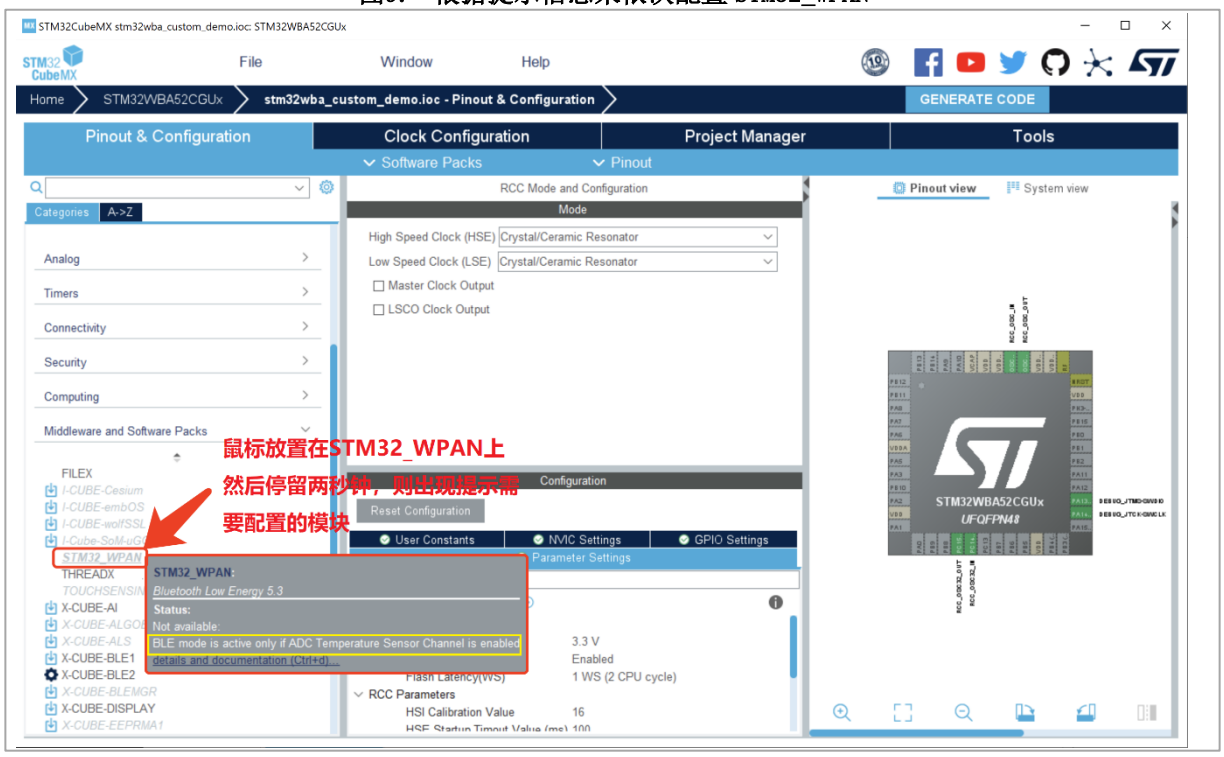


5.3. STM32_WPAN BLE 依赖的外设的配置

STM32WBA 的 STM32_WPAN(BLE)的依赖项比较多。在配置的时候，我们无需死记需要配置的项目，可以将鼠标移到 STM32_WPAN 模块中，停留 2 秒钟，则会出现提示信息，用户可根据这些提示信息来依次配置 STM32_WPAN BLE 需要的模块。

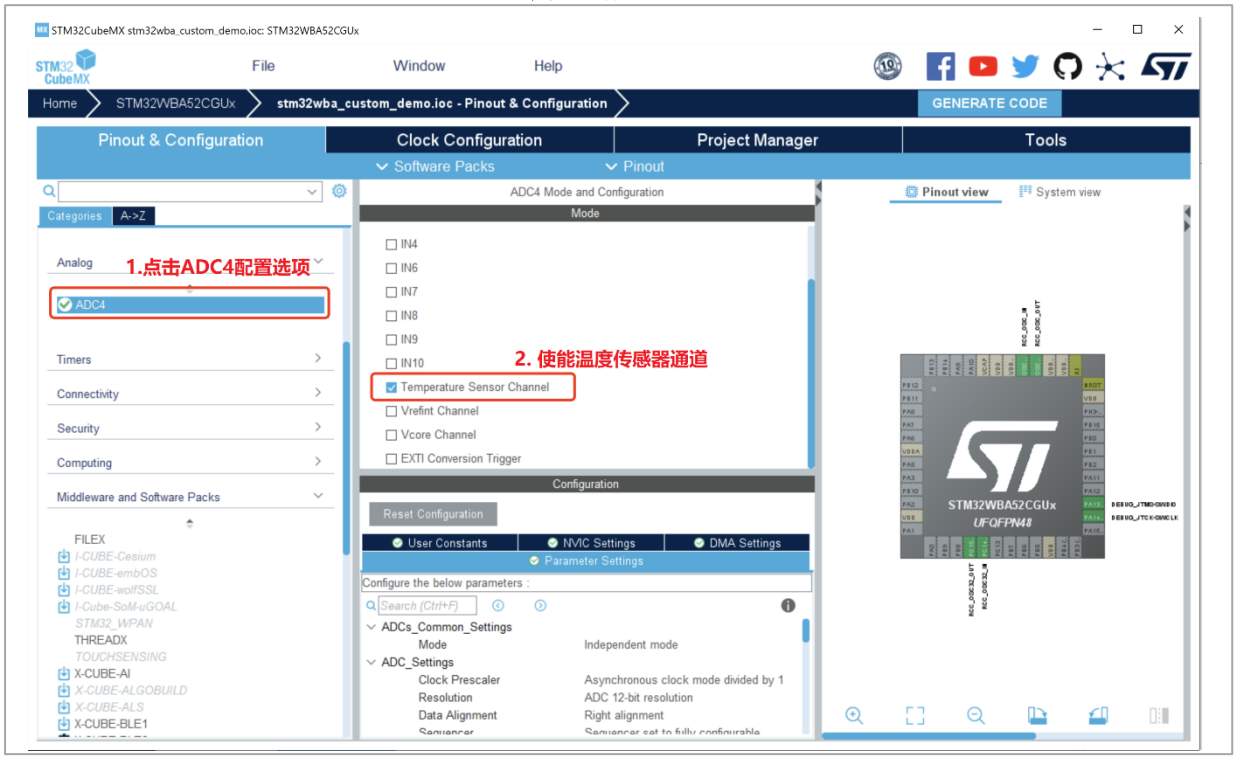
STM32_WPAN BLE 模块依赖的模块包括：ADC、CRC、RAMCFG、ICACHE、RNG、RF、和 RTC。当用户完成一个模块的配置后，鼠标光标回到 WPAN 模块时，提示信息会继续指导用户仍然需要配置的模块，直到全部模块配置完毕。

图9. 根据提示信息来依次配置 STM32_WPAN



5.3.1. 配置 ADC

图10. 配置 ADC



5.3.2. 配置 CRC

图11. 根据提示配置下一个模块（CRC）

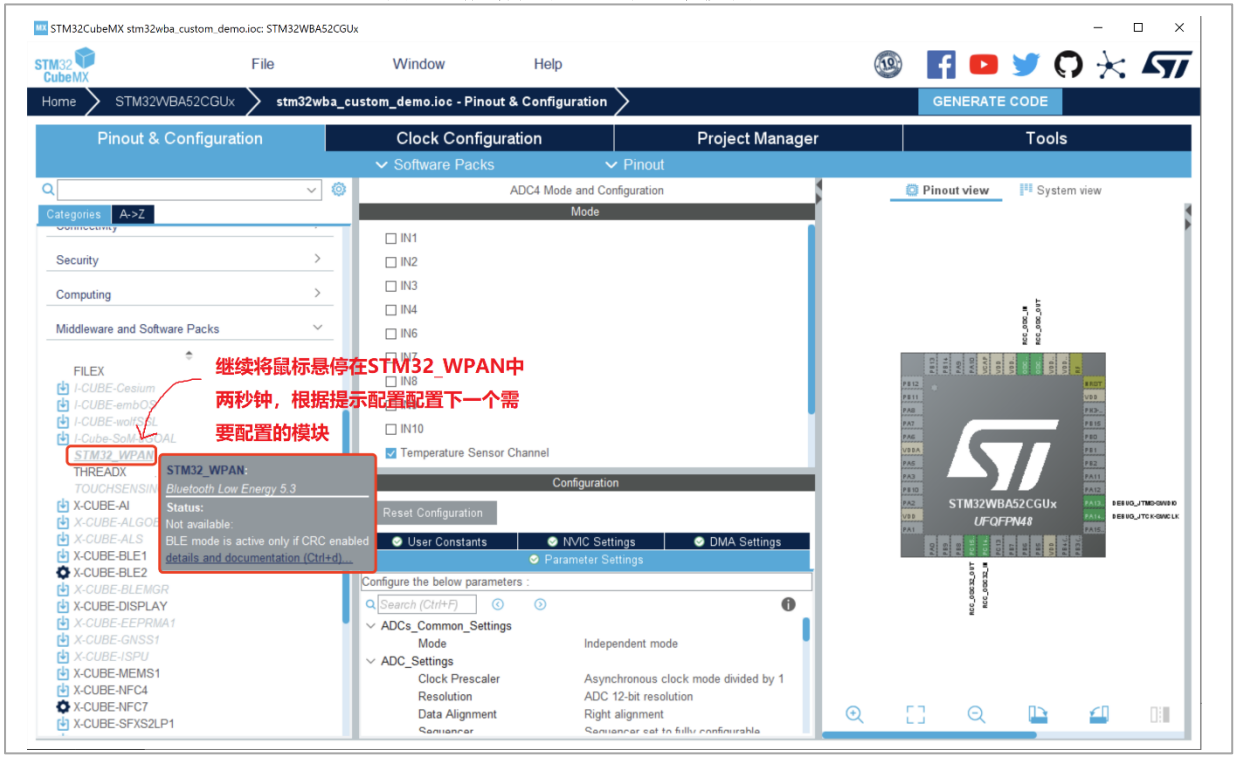
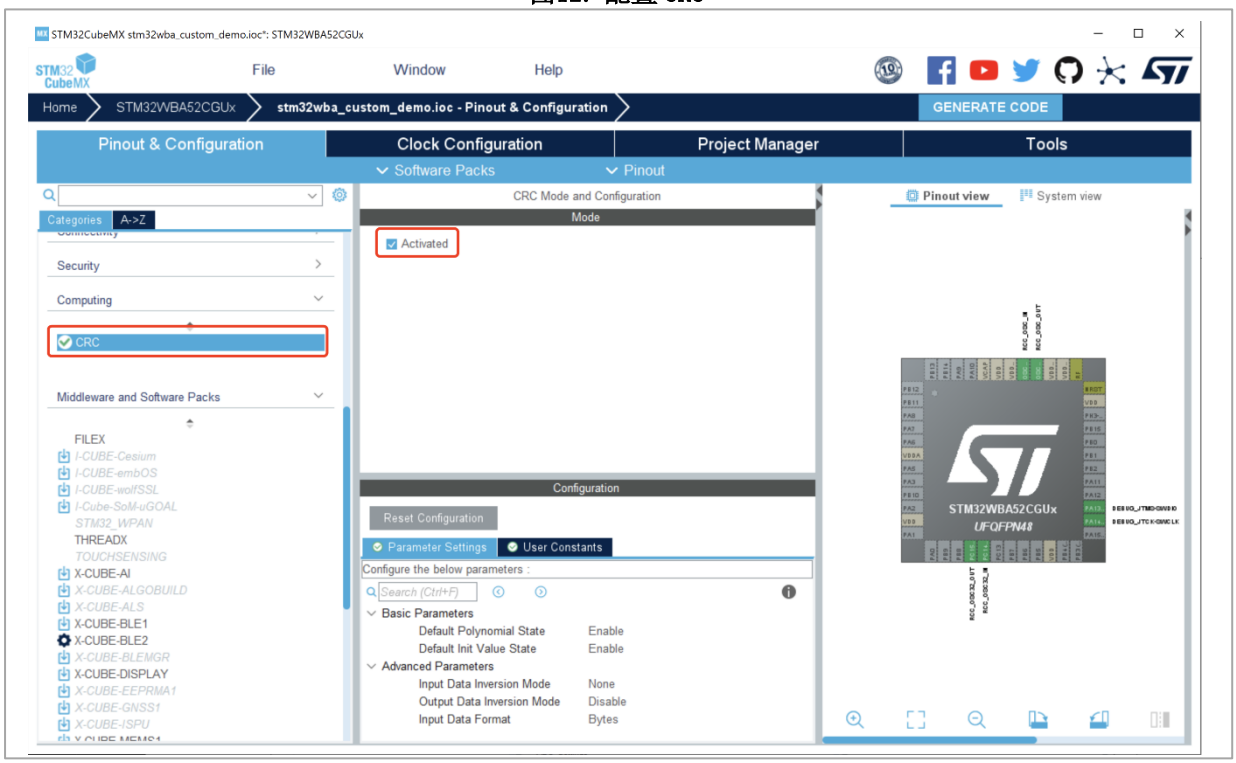
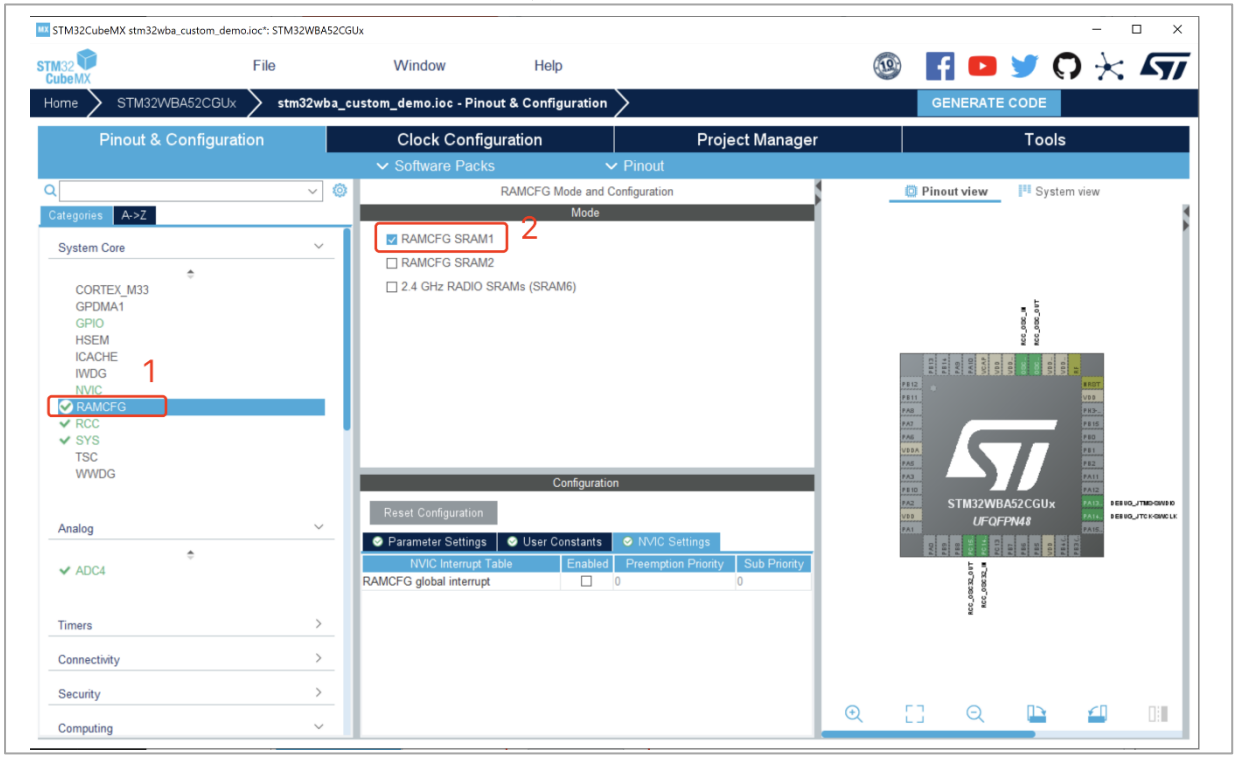


图12. 配置 CRC



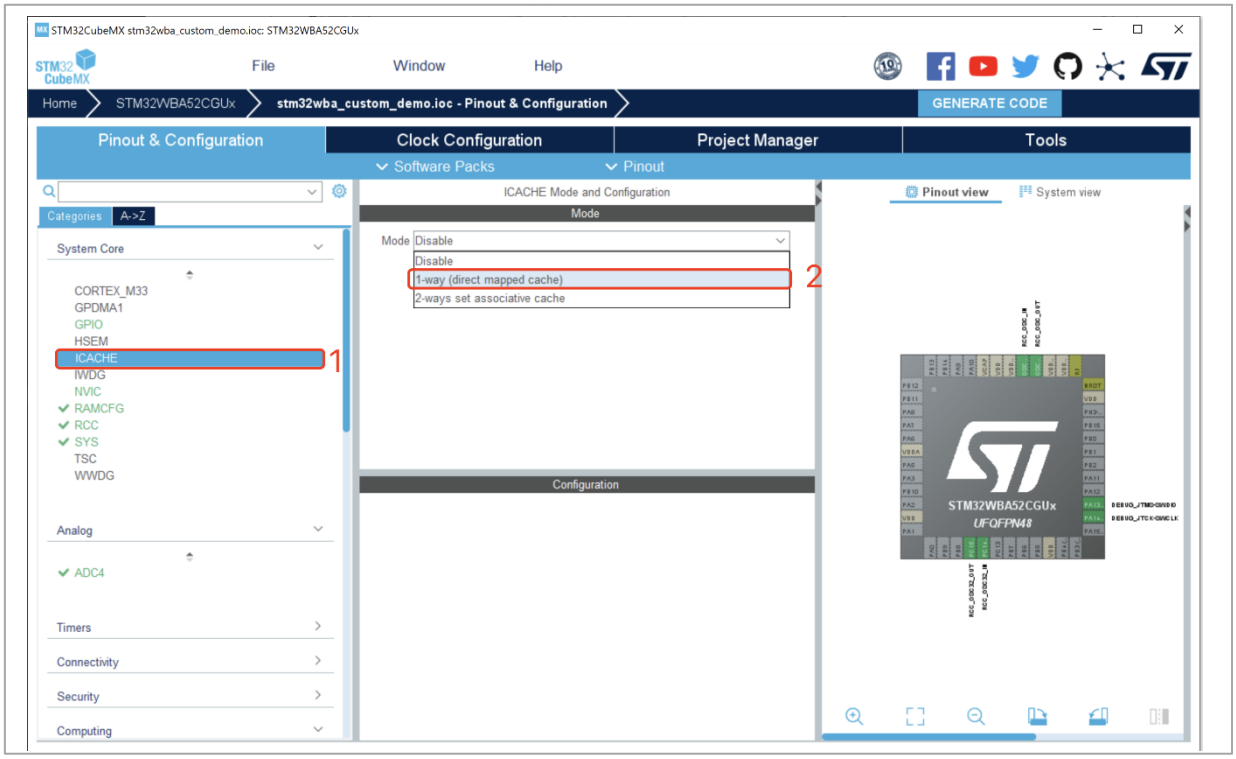
5.3.3. 配置 RAMCFG

图13. 配置 RAM



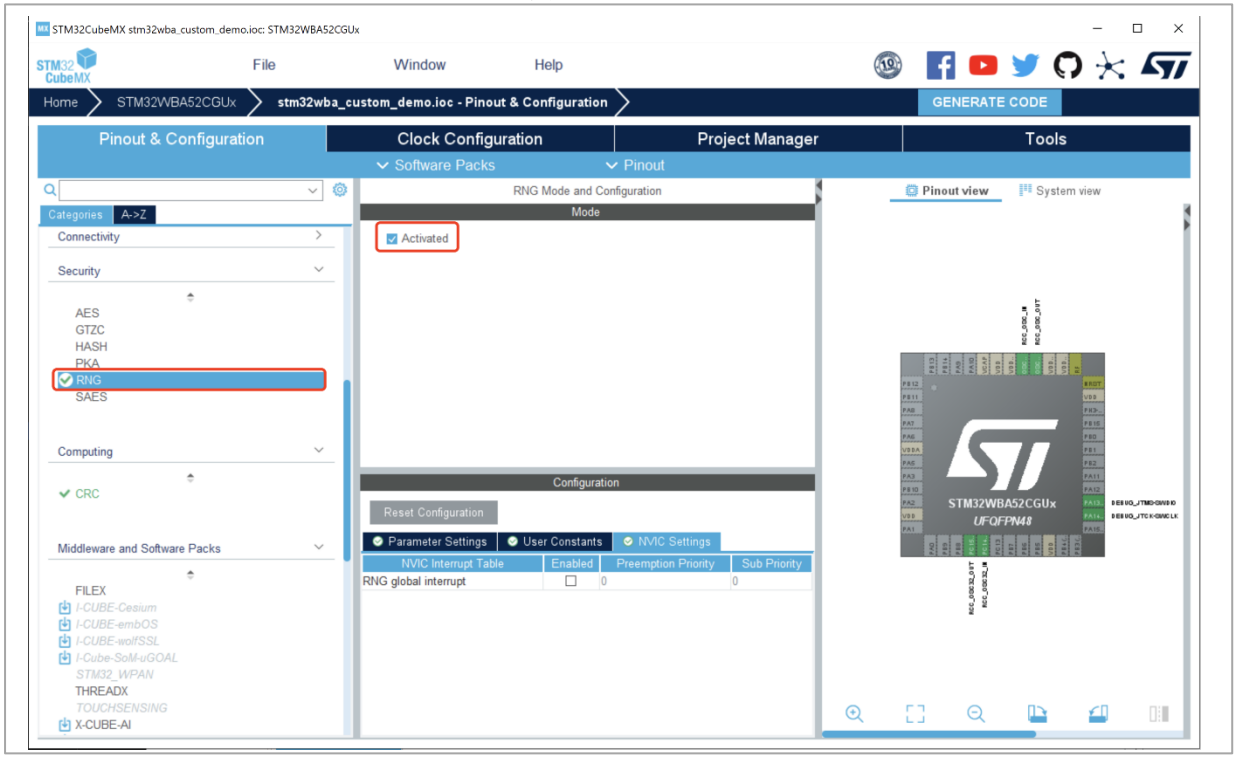
5.3.4. 配置 ICACHE

图14. 配置 ICACHE



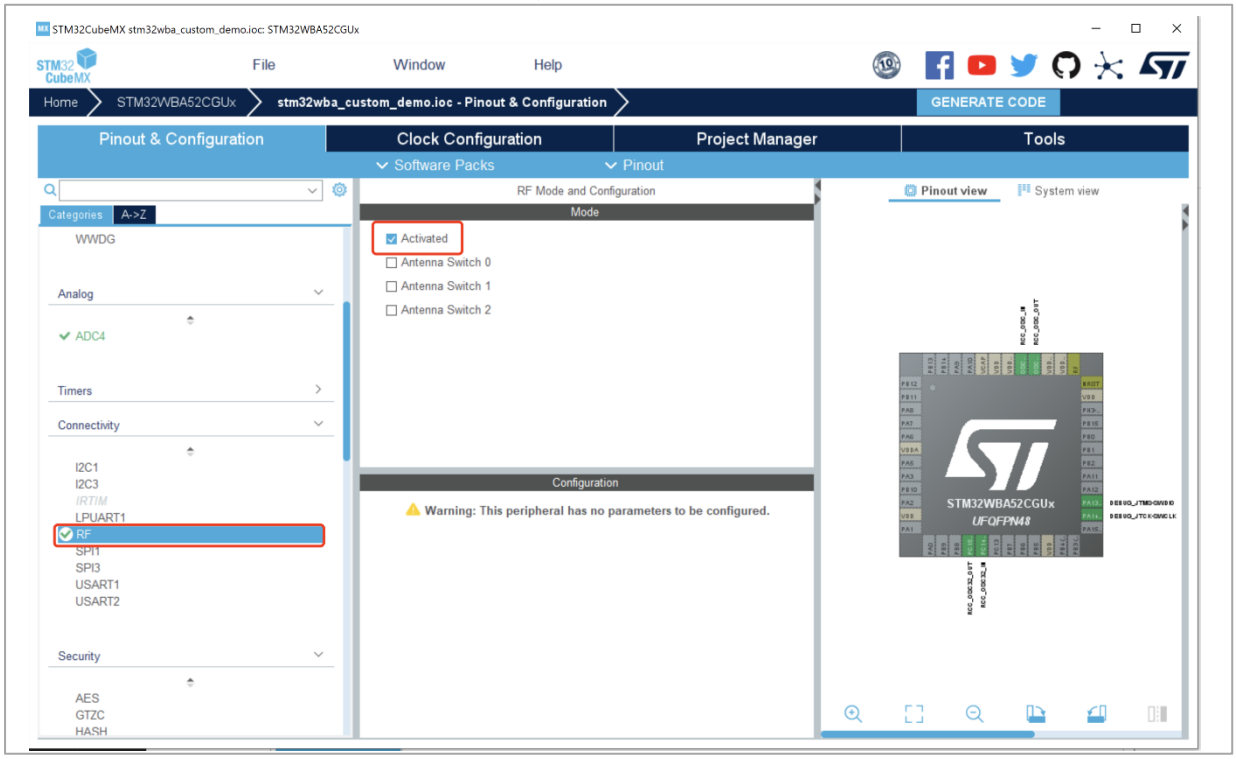
5.3.5. 配置 RNG

图15. 配置 RNG



5.3.6. 配置 RF

图16. 配置 RF



5.3.7. 配置 RTC

图17. 配置 RTC 部分 1

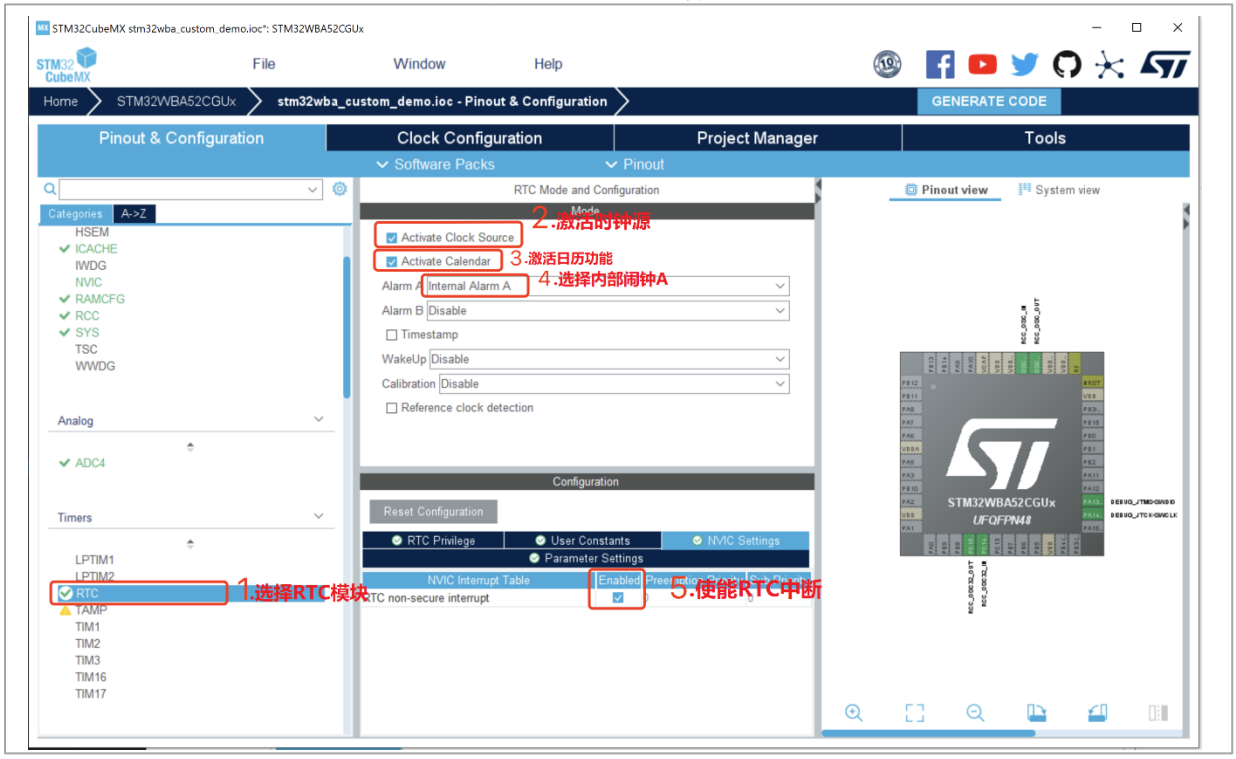
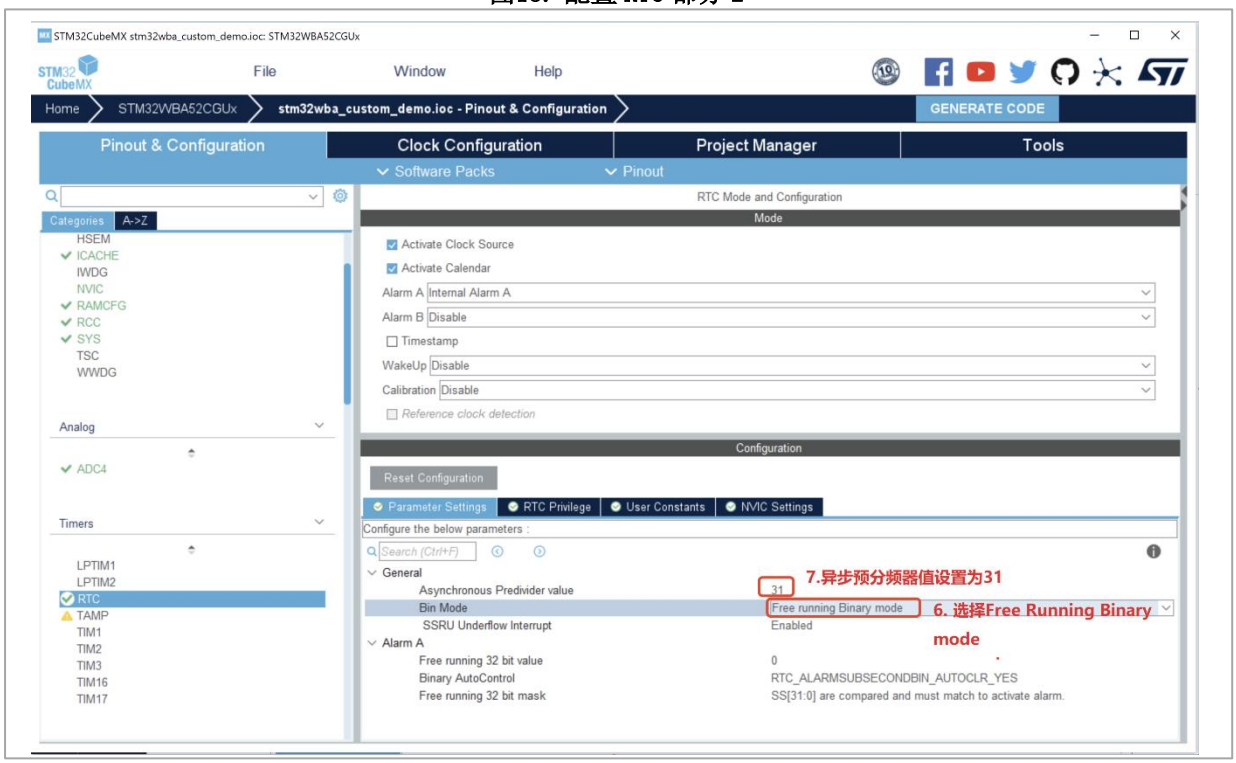


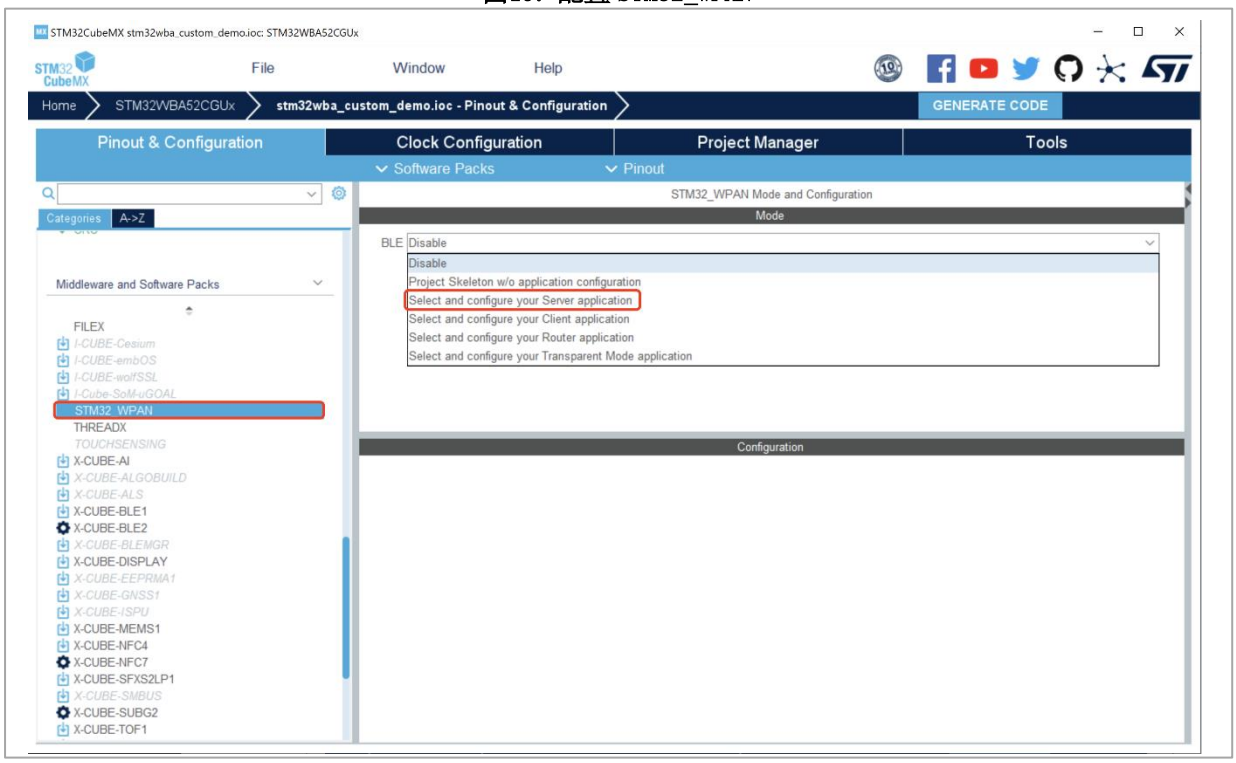
图18. 配置 RTC 部分 2



5.3.8. 配置 STM32_WPAN

当 WPAN 依赖的模块全部配置完毕，我们便可以开始 WPAN 的配置：

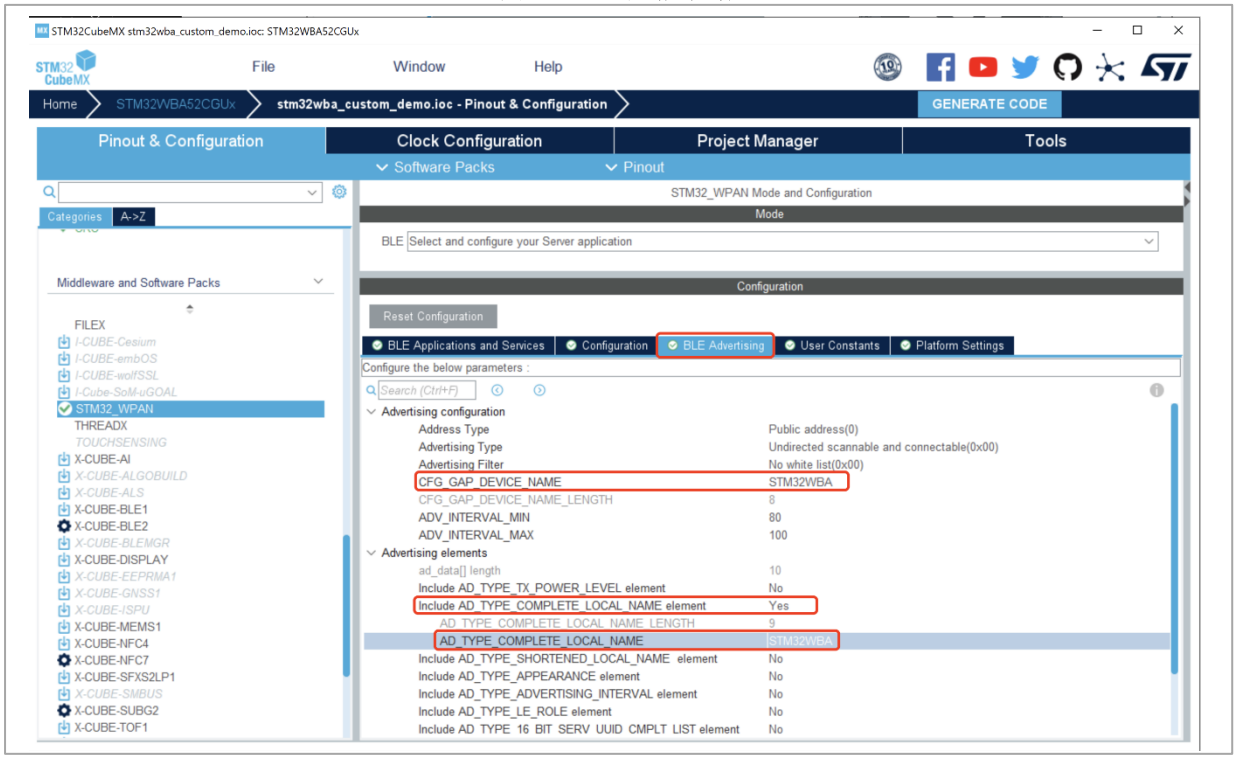
图19. 配置 STM32_WPAN



5.4. BLE GAP 和 GATT 配置

5.4.1. 配置 GAP

图20. 配置广播数据



5.4.2. 配置 GATT 服务

图21. 配置 GATT（添加服务）

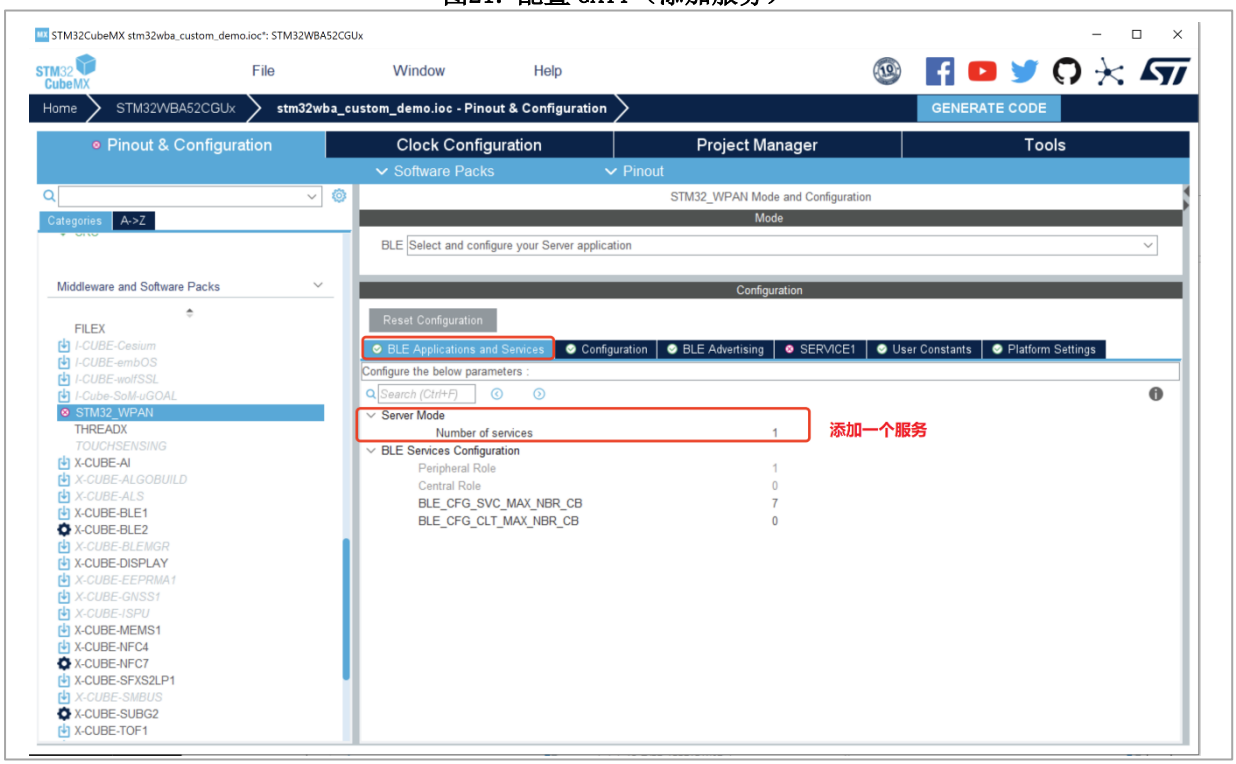
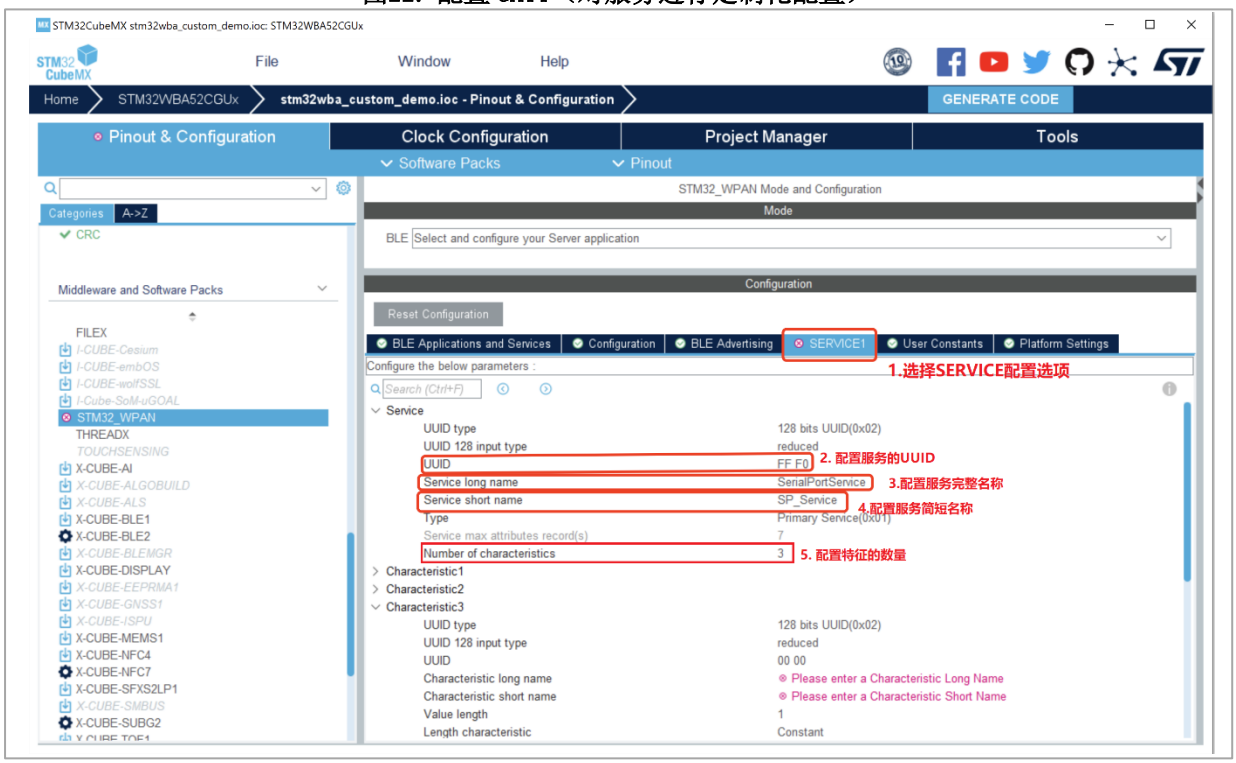
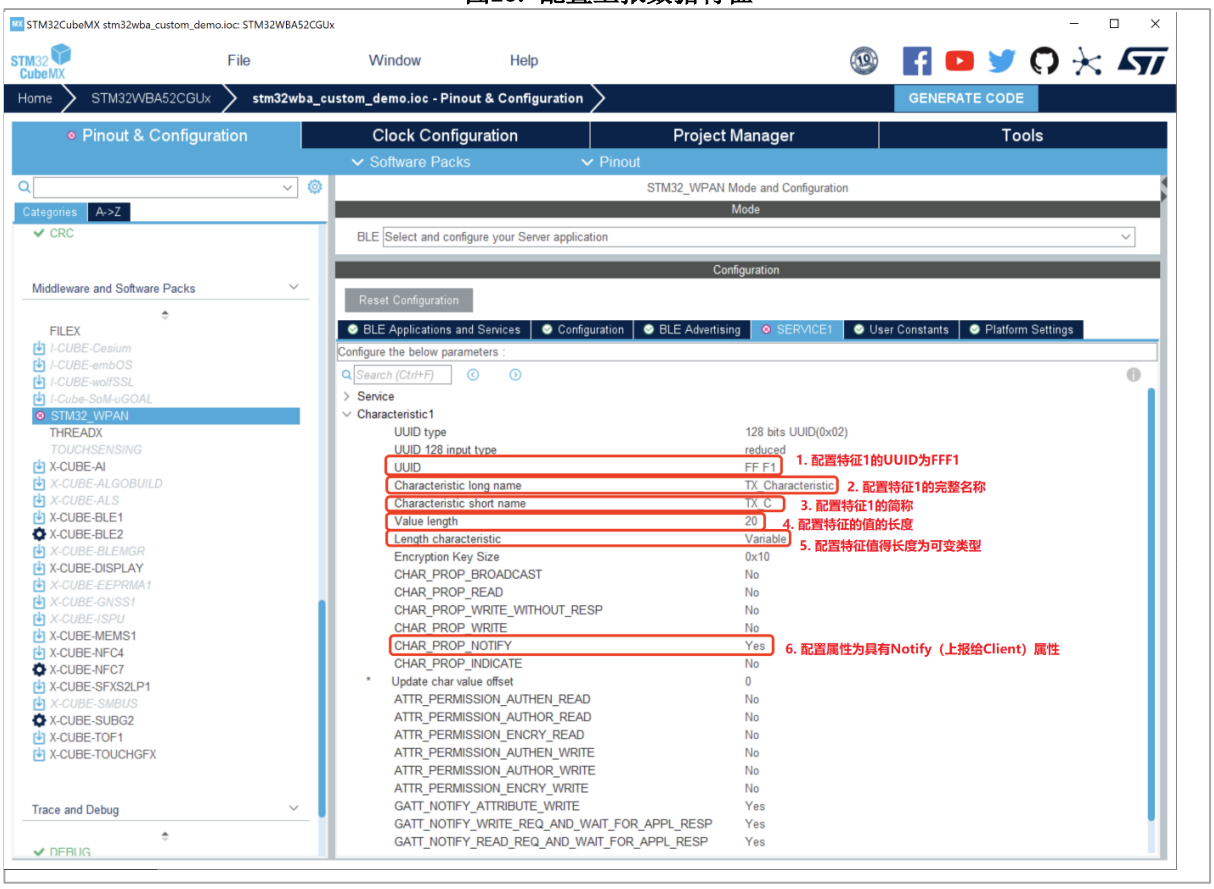


图22. 配置 GATT（对服务进行定制化配置）



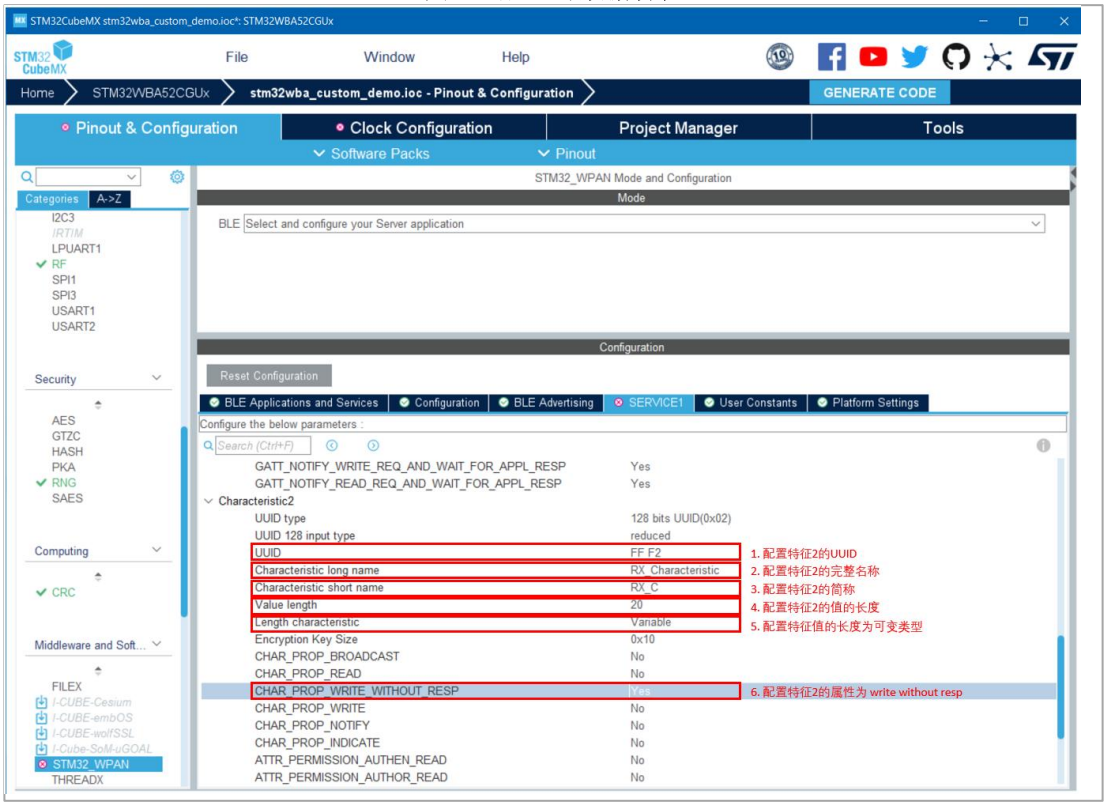
5.4.3. 配置自定义服务的特征 1(上报数据特征)

图23. 配置上报数据特征



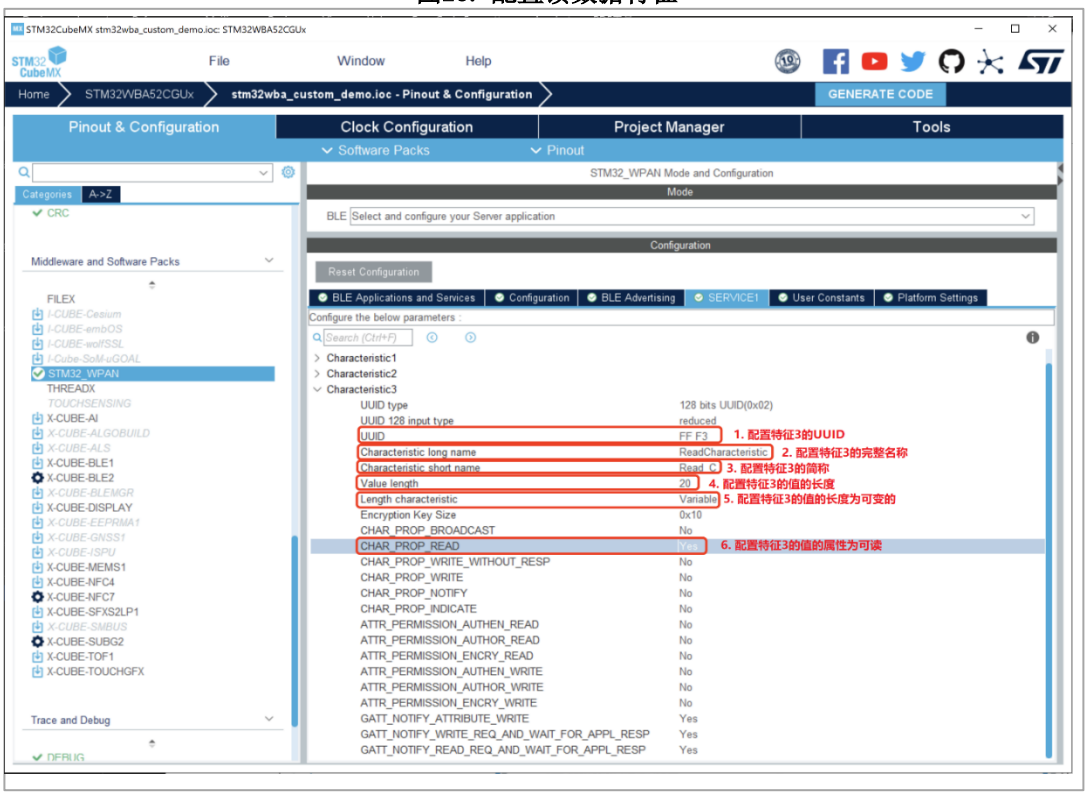
5.4.4. 配置自定义服务的特征 2(写数据特征)

图24. 配置写数据特征



5.4.5. 配置自定义服务的特征 3(读数据特征)

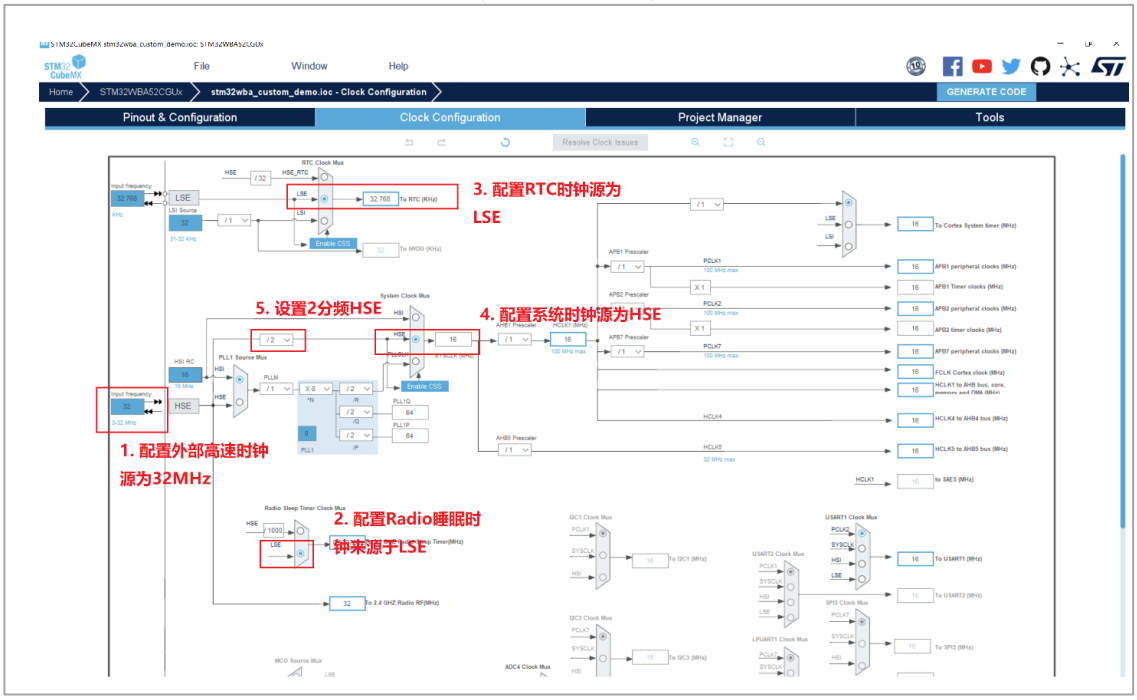
图25. 配置读数据特征



5.5. 其他配置与代码生成

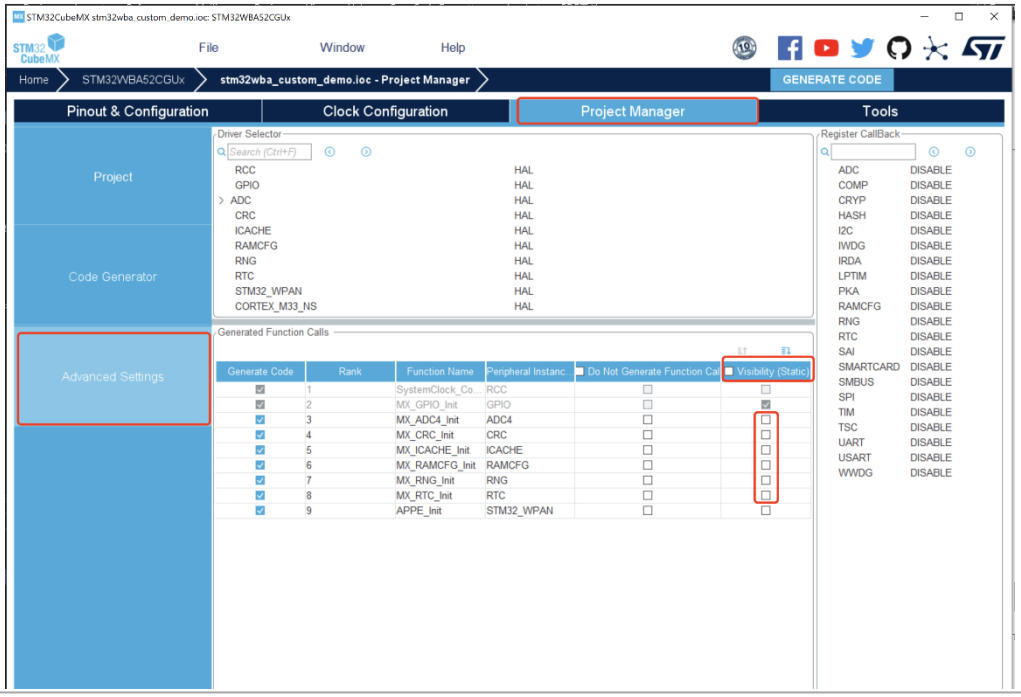
5.5.1. 进入“Clock Configuration”页面，按下图进行时钟配置

图26. 配置时钟



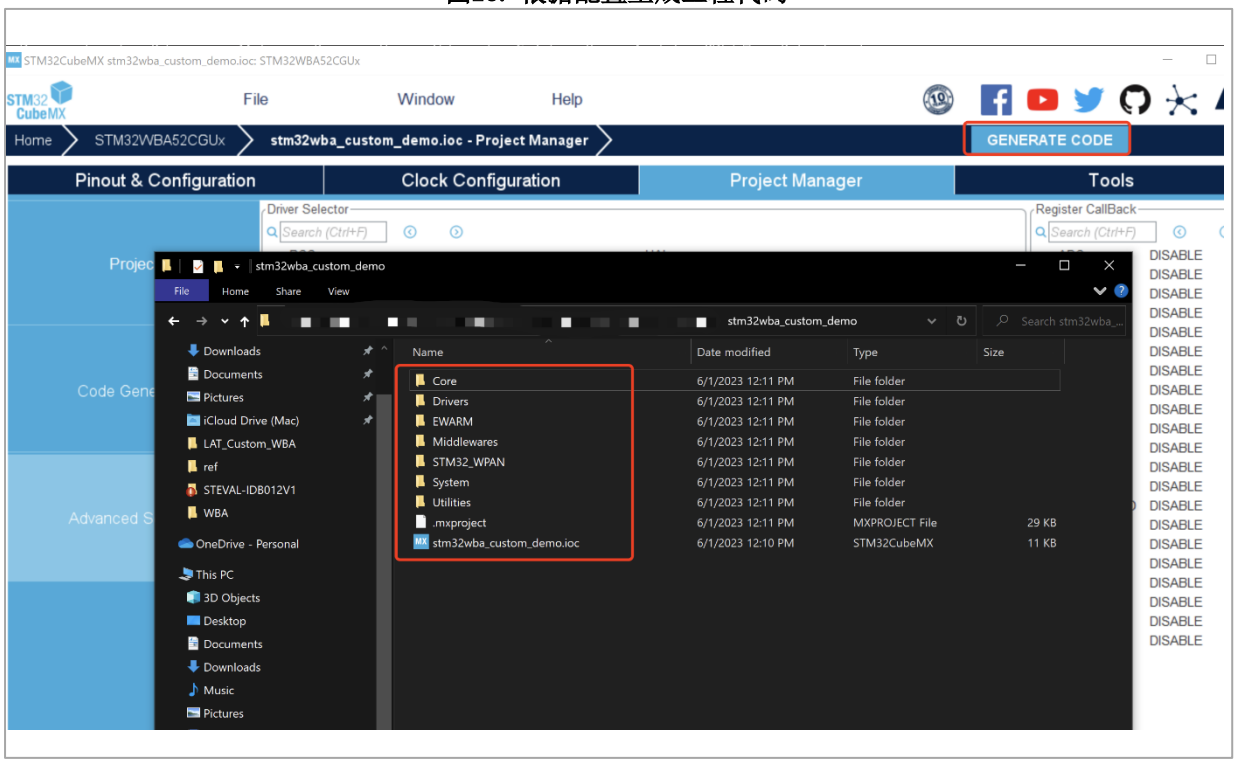
5.5.2. 配置初始化函数

图27. 配置初始化函数



5.5.3. 根据配置生成工程代码，点击“GENERATE CODE”按钮，并等待源码工程的生成。点“Open Folder”后，可看到如下代码工程目录：

图28. 根据配置生成工程代码



6. 验证蓝牙基本连接

本文演示的是 IAR 工程的生成，用户也可生成 CUBEIDE 工程。

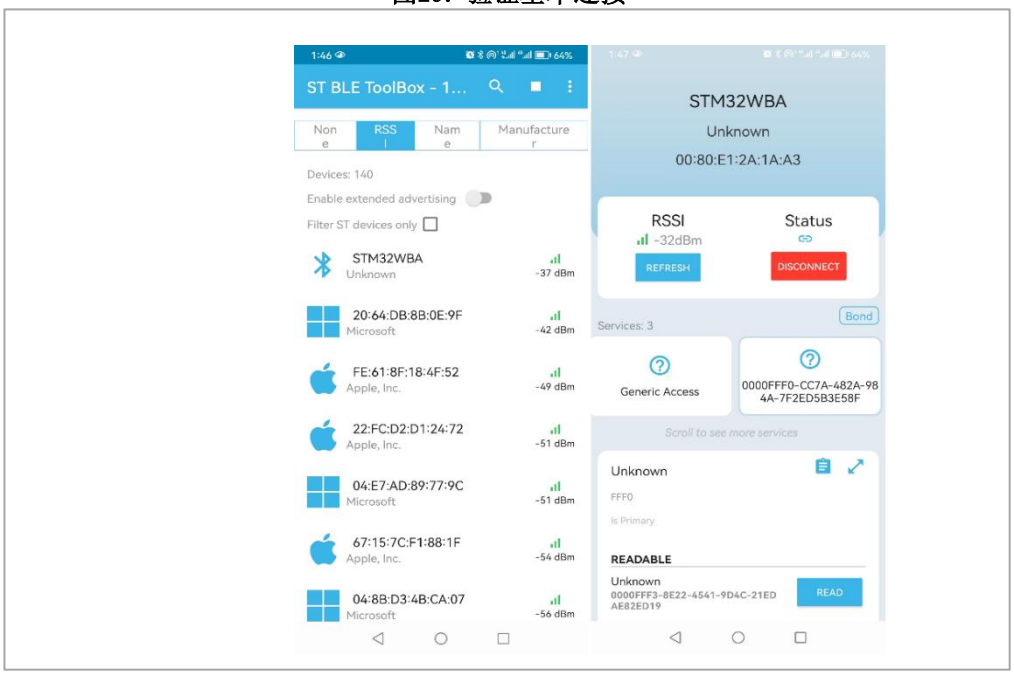
打开 IAR 工程、用户还需要手工增加代码以开启广播，具体为：在 STM32_WPAN > App > app_ble.c > void APP_BLE_Init(void) 函数的尾部增加下图所示的函数：

```
void APP_BLE_Init(void)
{
      …
      /* USER CODE BEGIN APP_BLE_Init_2 */
      APP_BLE_Procedure_Gap_Peripheral(PROC_GAP_PERIPH_ADVERTISE_START_FAST);
      /* USER CODE END APP_BLE_Init_2 */
      …
}
```

然后编译、下载并复位，使代码运行起来。

使用 ST BLE ToolBox 扫描，并连接该外设，可验证我们上面的服务配置是否正确：

图29. 验证基本连接



7. 使能串口日志追踪

7.1. 当用户需要使能串口 LOG 输出功能时，可重新回到 CubeMX 的界面，进行串口的配置：

图30. 配置串口

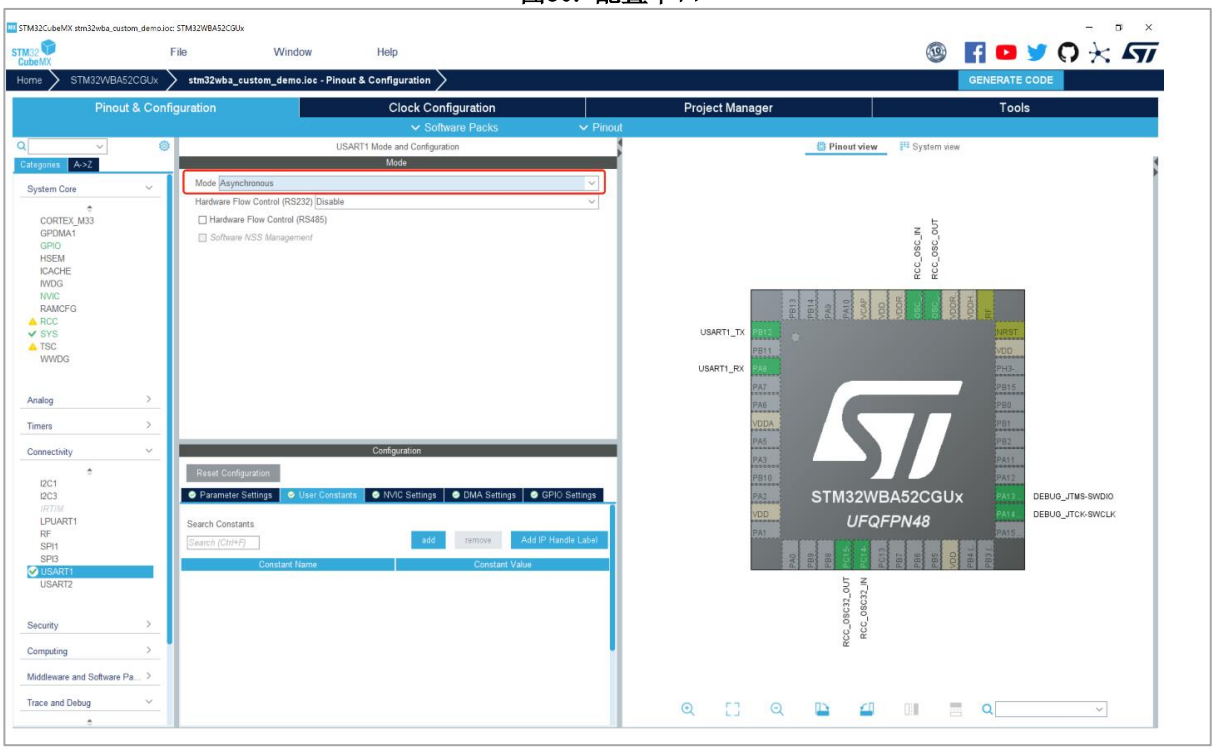
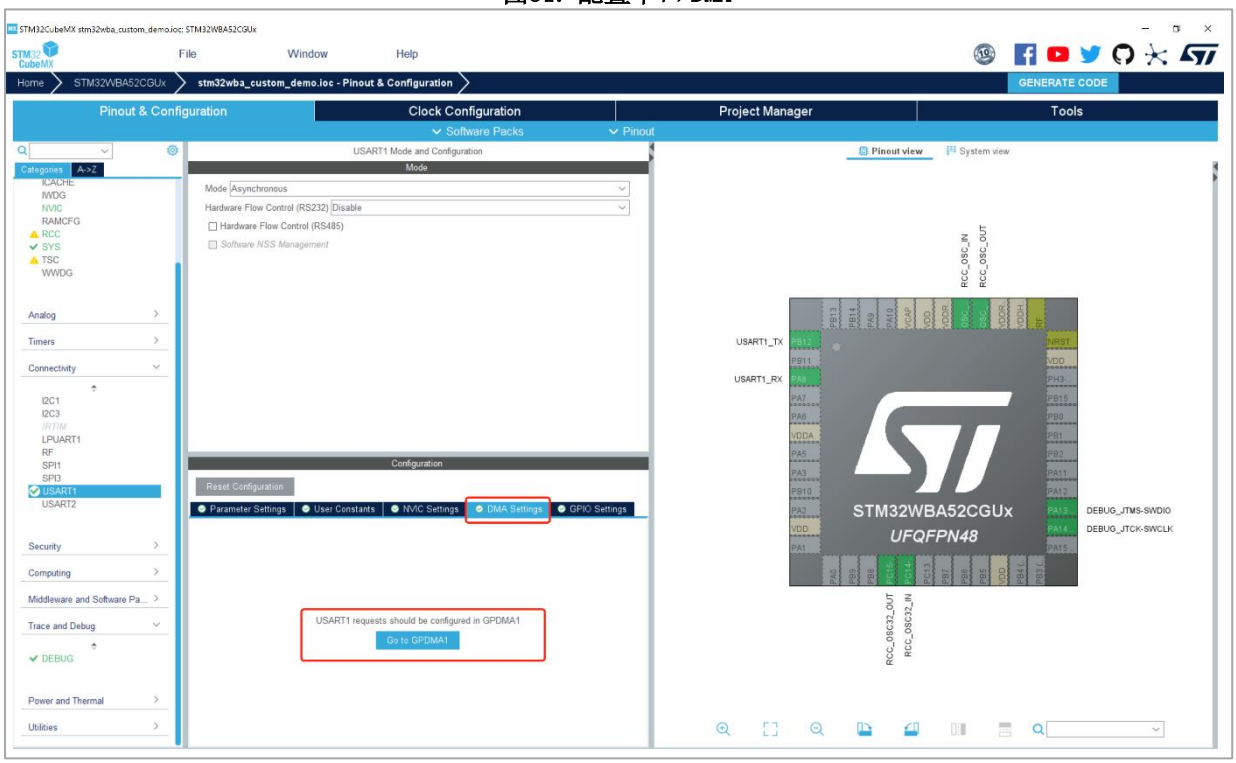


图31. 配置串口 DMA



7.2. 为串口配置 GPDMA

图32. 给串口配置 GPDMA TX

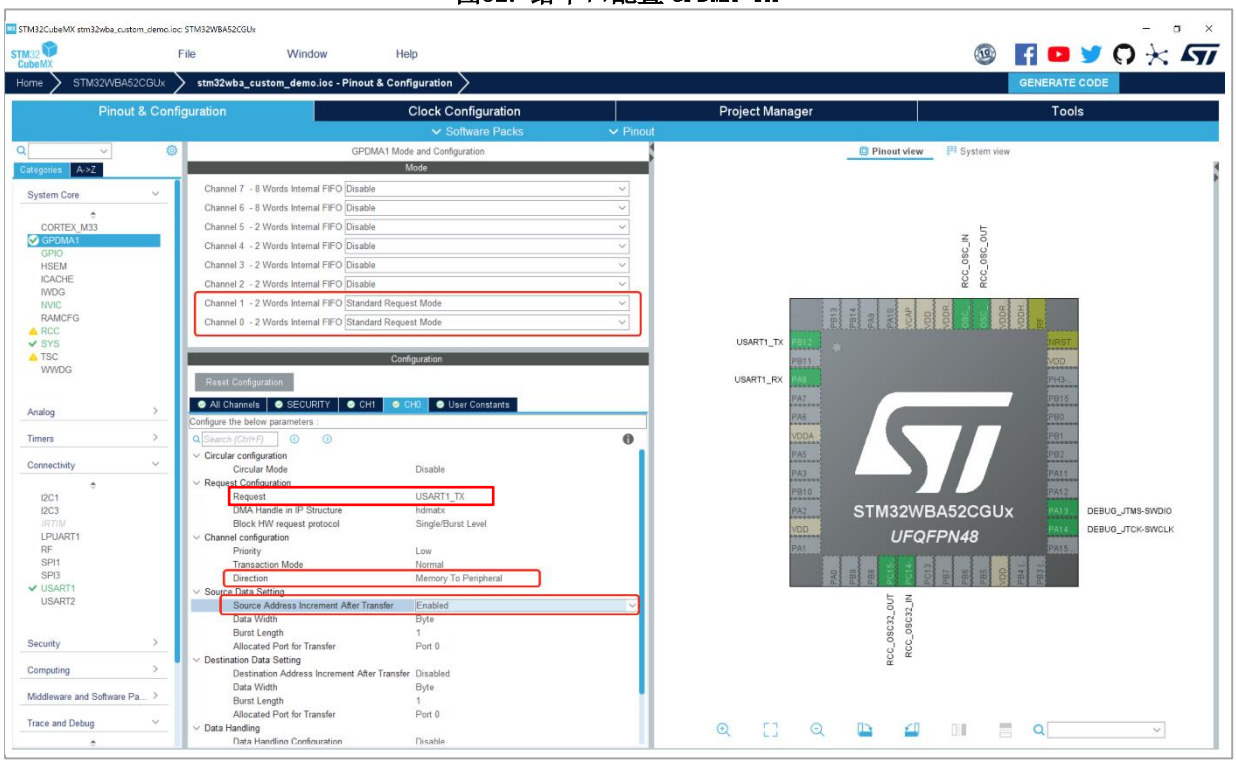


图33. 给串口配置 GPDMA RX

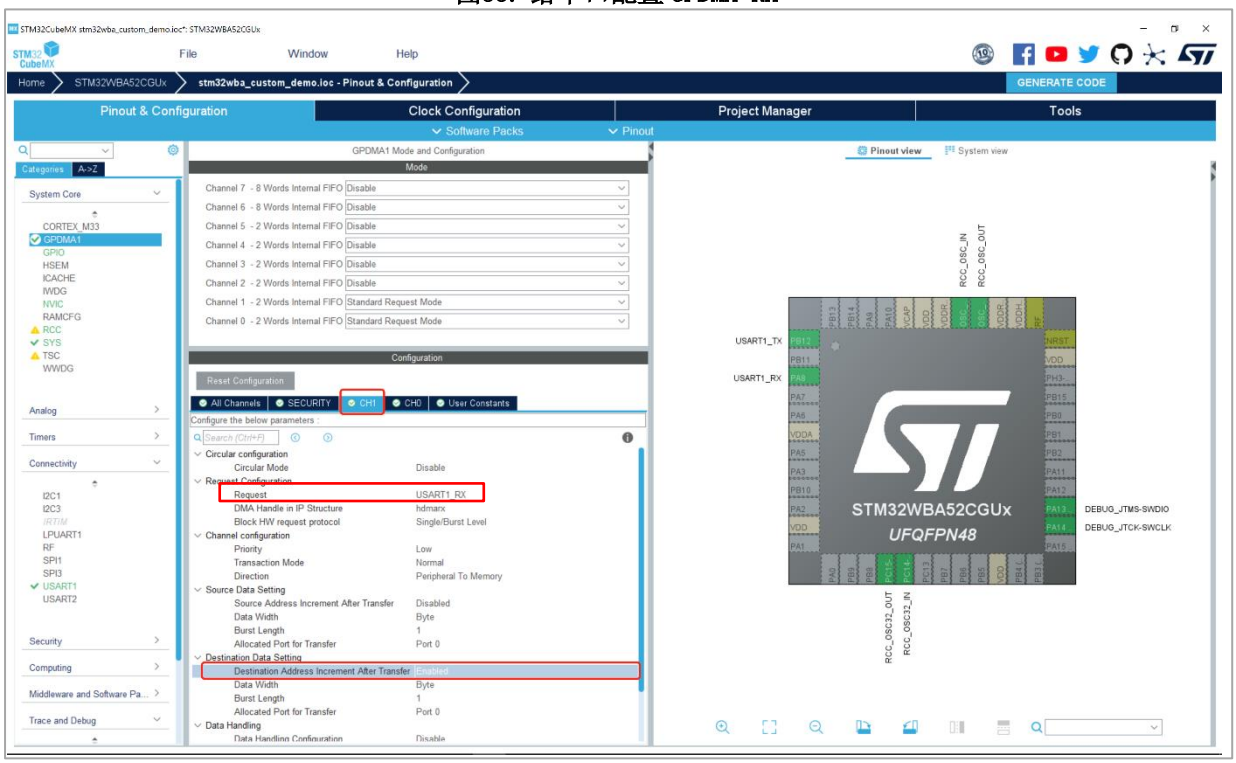


图34. 配置串口追踪 1

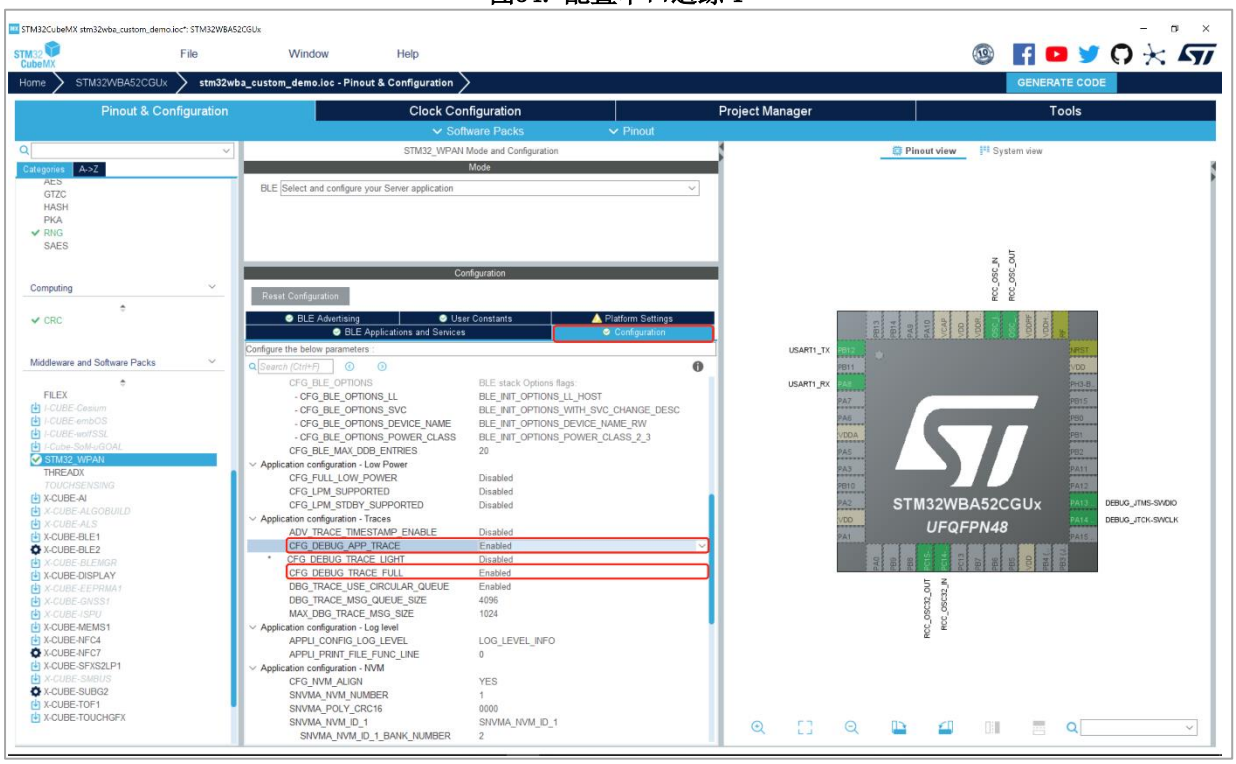


图35. 配置串口追踪 2

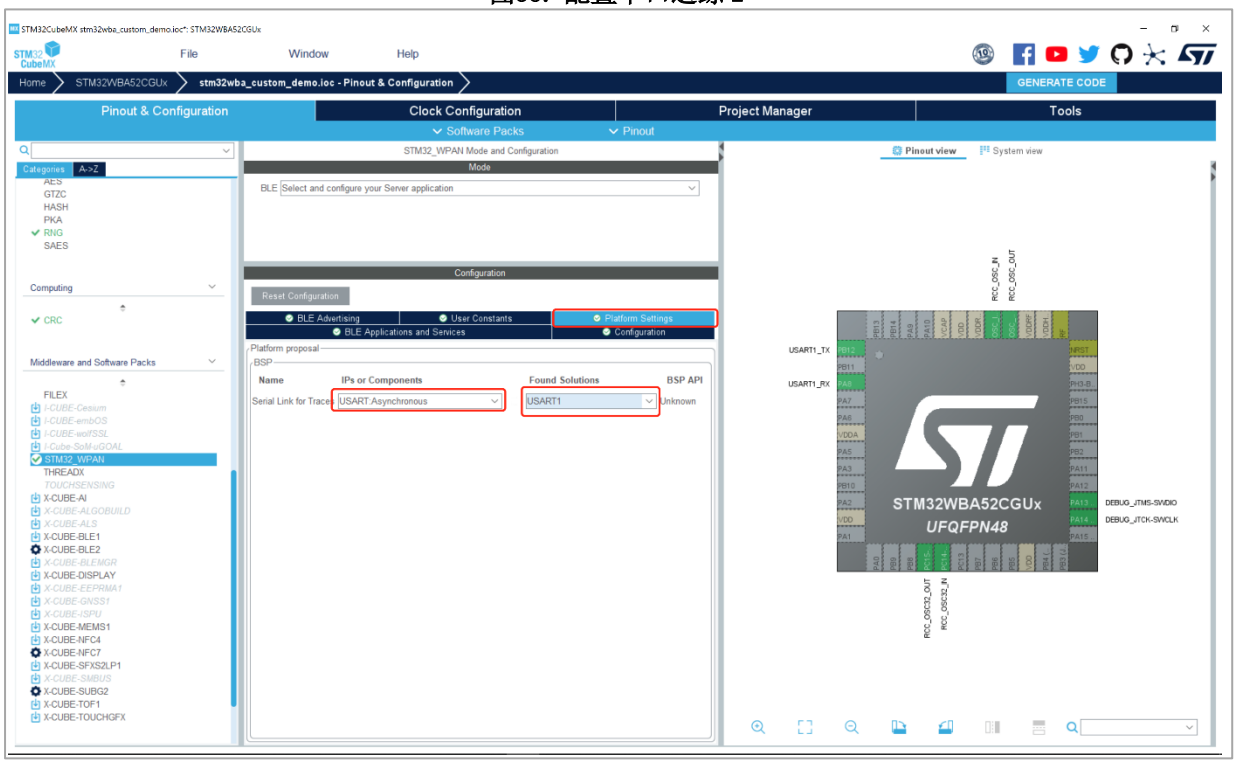


图36. 使能串口中断

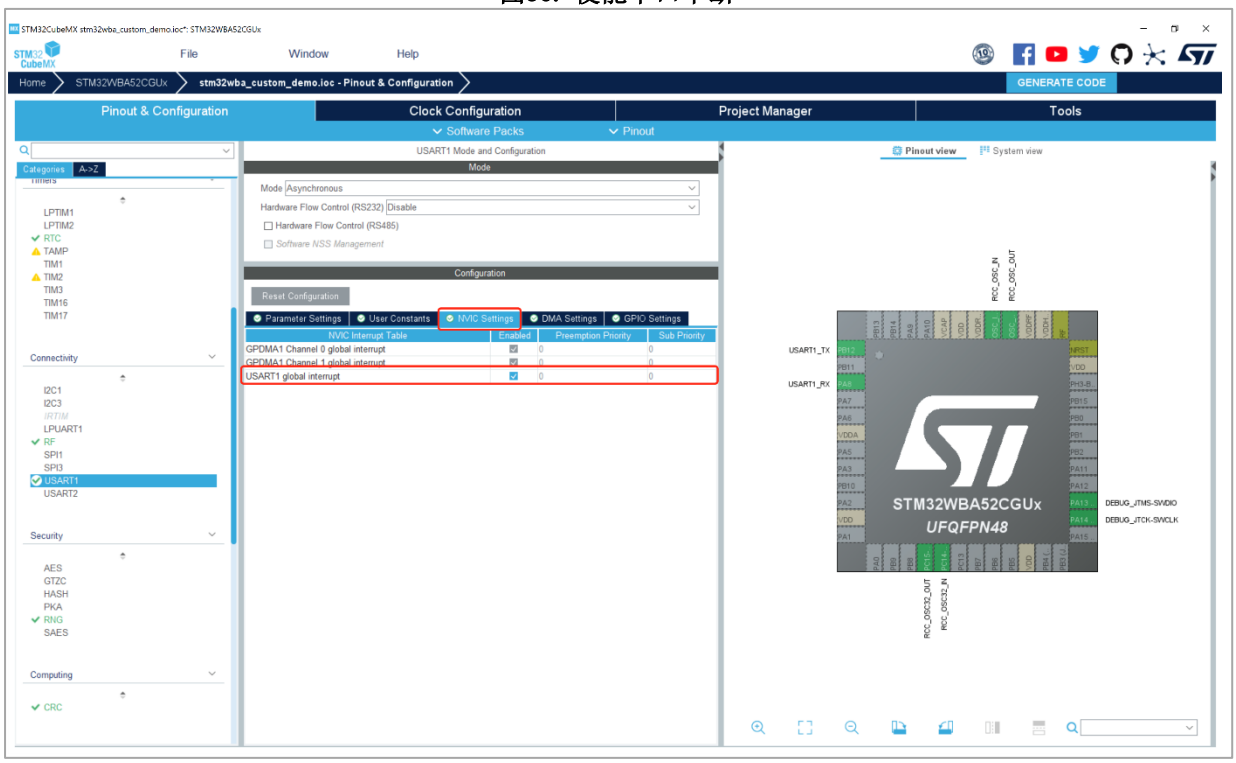
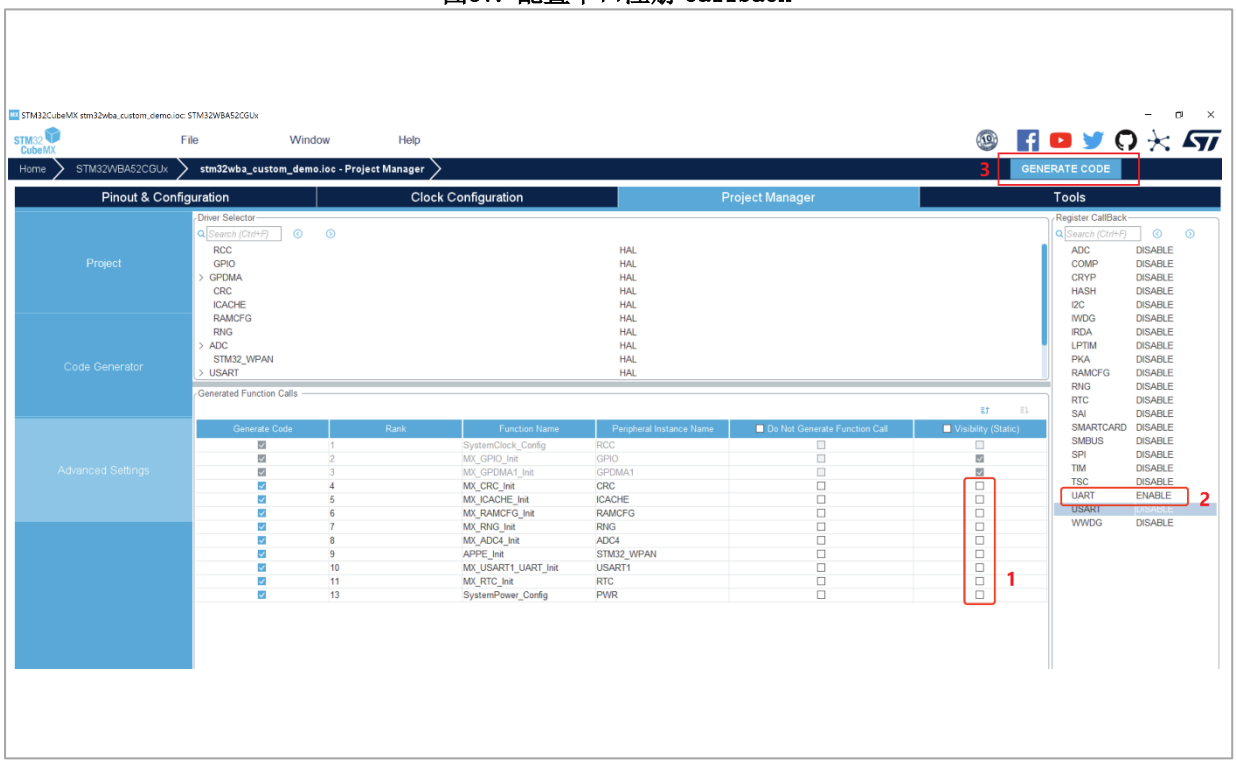


图37. 配置串口注册 callback



完成以上配置后，可再次点击“GENERATE CODE”生成新的代码。新的代码会覆盖掉工程中旧的代码，但不会覆盖工程中的用户代码部分。

7.3. 添加代码、开启串口日志追踪

新的代码生成后，还要再手工添加一些代码：

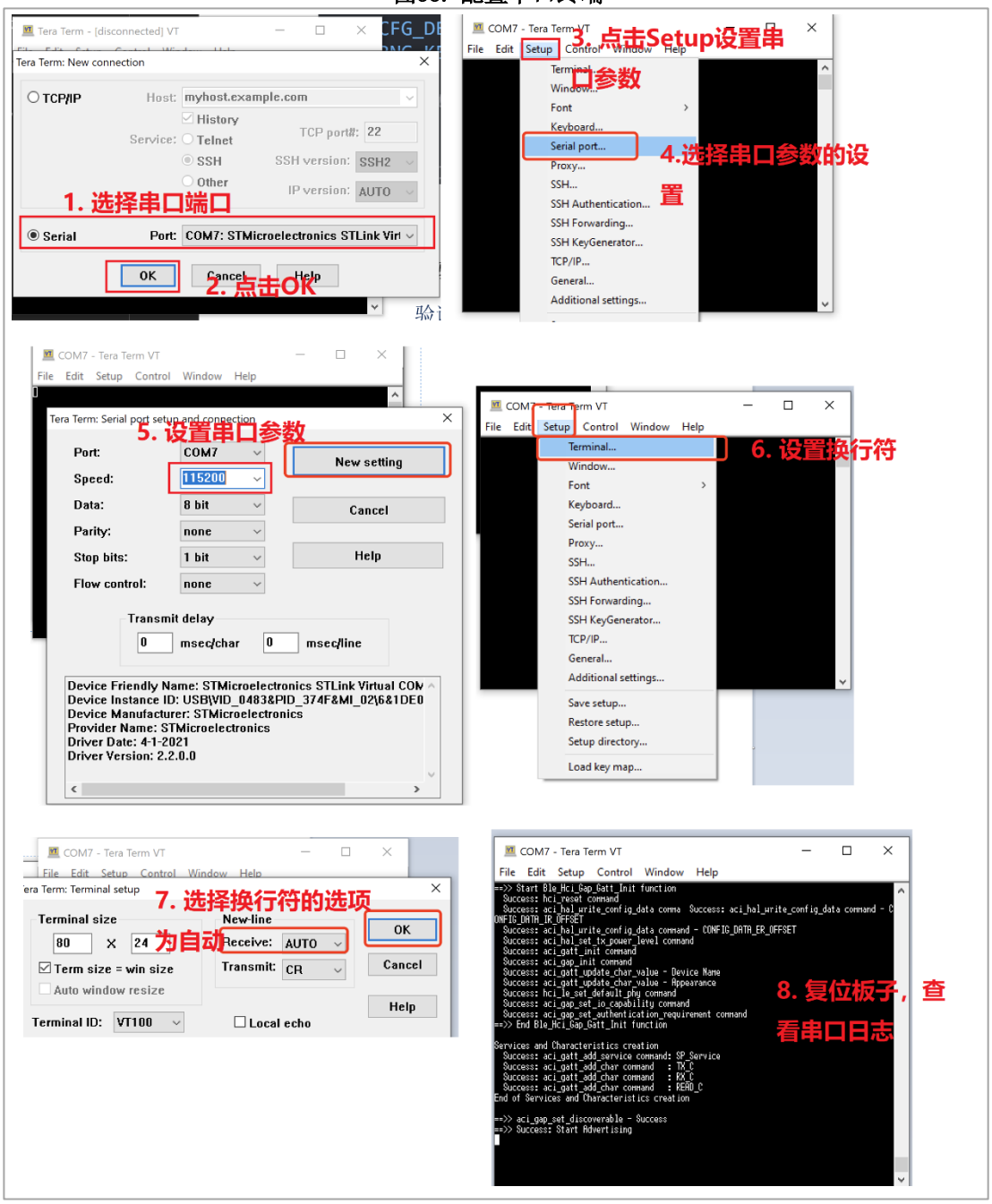
在 app_conf.h 中添加宏定义“CFG_DBG_SUPPORTED”以及在 main.c 中添加函数 RNG_KERNEL_CLK_OFF()

```
// app_conf.h
/* USER CODE BEGIN Low_Power 0 */
#define CFG_DBG_SUPPORTED        (1)
/* USER CODE END Low_Power 0 */

// main.c
/* USER CODE BEGIN 4 */
#if (CFG_DEBUG_APP_TRACE != 0)
void RNG_KERNEL_CLK_OFF(void)
{
  /* Do not switch off HSI clock as it is used for traces */
}
#endif
/* USER CODE END 4 */
```

7.4. 验证串口日志追踪功能

验证串口日志的方法比较简单，直接打开串口助手，选择好对应的串口，然后设置波特率为 115200，无奇偶校验，数据为 8bit，1 位停止位。

图38. 配置串口终端



7.4.1. 验证读特征

验证读特征可以在代码中添加如下代码(每次读取数据，将数值加一后发送出去)

```
// sp_service.c
case ACI_GATT_READ_PERMIT_REQ_VSEVT_CODE :
……
/*USER CODE BEGIN Service1_Char_3_ACI_GATT_READ_PERMIT_REQ_VSEVT_CODE_1 */
 static uint8_t s_count = 0;
 s_count++;
 tBleStatus ret = aci_gatt_update_char_value( SP_SERVICE_Context.Sp_serviceSvcHdle,
                                    SP_SERVICE_Context.Read_CCharHdle,
                                    0,
                                    sizeof(s_count),
```

```
                                        &s_count);
      APP_DBG_MSG("ACI_GATT_READ_PERMIT_REQ_VSEVT_CODE");
      APP_DBG_MSG("aci_gatt_update_char_value ret:%x \n", ret);
      aci_gatt_allow_read(p_read_req->Connection_Handle);
    /*USER CODE END Service1_Char_3_ACI_GATT_READ_PERMIT_REQ_VSEVT_CODE_1*/
     ......
```

7.4.2. 验证写和 Notify（上报数据）特征

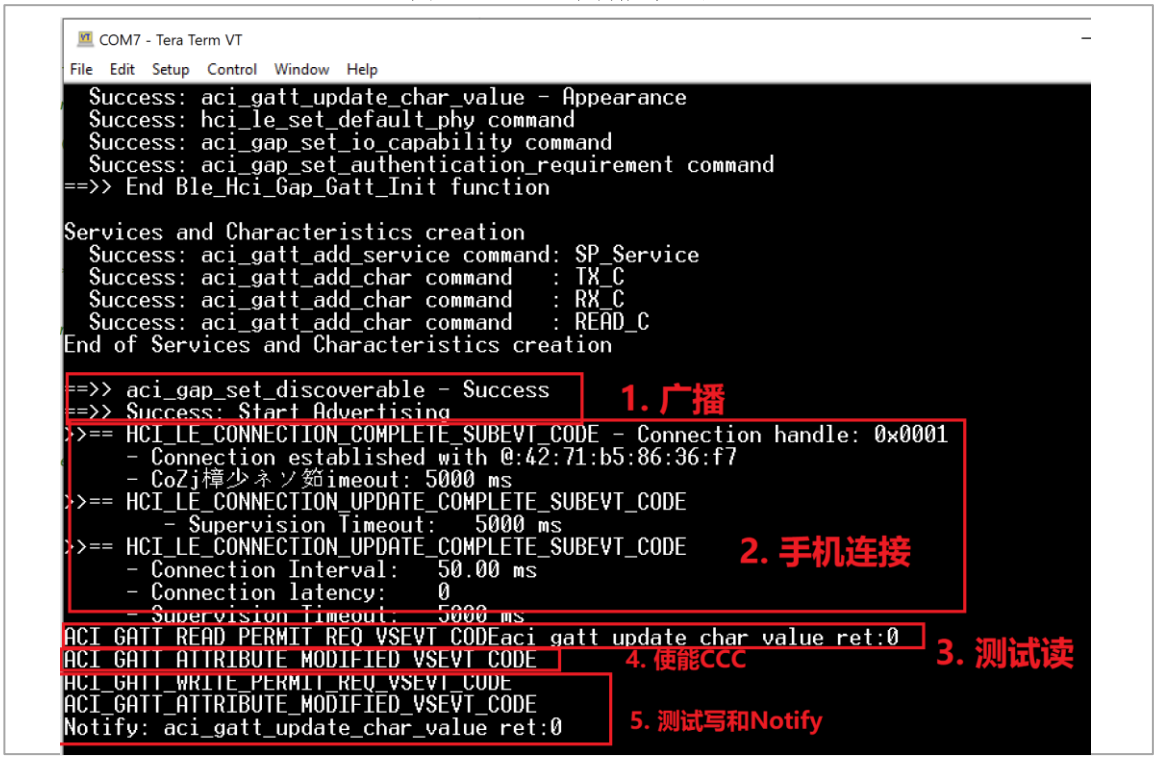
验证写特征和上报数据特征可以一起进行。当收到写特征的数据时，直接将数据更新到上报数据特征中提交给 Client 端，如下图所示，其中主要修改了两个 case：

一个是允许写，一个是将收到的数据写回去。

```
        // sp_service.c
       case ACI_GATT_WRITE_PERMIT_REQ_VSEVT_CODE:
        ......
        /* USER CODE END EVT_BLUE_GATT_WRITE_PERMIT_REQ_BEGIN */
        p_write_perm_req = (aci_gatt_write_permit_req_event_rp0*)p_blecore_evt->data;
        if(p_write_perm_req->Attribute_Handle == (SP_SERVICE_Context.Rx_CCharHdle + CHARACTERISTIC_VALUE_ATTRIBUTE_OFFSET))
        {
          return_value = SVCCTL_EvtAckFlowEnable;
          /*USER CODE BEGIN Service1_Char_2_ACI_GATT_WRITE_PERMIT_REQ_VSEVT_CODE */
          aci_gatt_write_resp(p_write_perm_req->Connection_Handle,
                              p_write_perm_req->Attribute_Handle,
                              0x00,
                              0x00,
                              p_write_perm_req->Data_Length,
                              p_write_perm_req->Data);
//#warning user shall call aci_gatt_write_resp() function if allowed
          /*USER CODE END Service1_Char_2_ACI_GATT_WRITE_PERMIT_REQ_VSEVT_CODE*/
        } /*if(p_write_perm_req->Attribute_Handle == (SP_SERVICE_Context.Rx_CCharHdle +
CHARACTERISTIC_VALUE_ATTRIBUTE_OFFSET))*/

        /* USER CODE BEGIN EVT_BLUE_GATT_WRITE_PERMIT_REQ_END */


      ......
        // sp_service.c

      case ACI_GATT_ATTRIBUTE_MODIFIED_VSEVT_CODE:
           ......
          /* USER CODE BEGIN Service1_Char_2_ACI_GATT_ATTRIBUTE_MODIFIED_VSEVT_CODE */
          // 将收到的数据Notify 回去
          tBleStatus ret = aci_gatt_update_char_value( SP_SERVICE_Context.Sp_serviceSvcHdle,
                                    SP_SERVICE_Context.Tx_CCharHdle,
                                    0,
                                    p_attribute_modified->Attr_Data_Length,
                                    p_attribute_modified->Attr_Data);
          APP_DBG_MSG("Notify: aci_gatt_update_char_value ret:%x \n", ret);
          /* USER CODE END Service1_Char_2_ACI_GATT_ATTRIBUTE_MODIFIED_VSEVT_CODE */
          notification.EvtOpcode = SP_SERVICE_RX_C_WRITE_NO_RESP_EVT;
          SP_SERVICE_Notification(&notification);
        } /* if(p_attribute_modified->Attr_Handle == (SP_SERVICE_Context.Rx_CCharHdle +
CHARACTERISTIC_VALUE_ATTRIBUTE_OFFSET))*/

        /* USER CODE BEGIN EVT_BLUE_GATT_ATTRIBUTE_MODIFIED_END */

          ......
```

7.4.3. 验证日志追踪

图39. 验证蓝牙功能串口日志



8. 小结

本文介绍了如何从芯片开始一步一步配置一个 BLE 工程，实际用户在配置自己的项目时可以自行根据自己的项目需求而做相应的修改。

版本历史

日期	版本	变更
2023 年 06 月 09 日	1.0	首版发布

重要通知 – 请仔细阅读

意法半导体公司及其子公司 （“ST”）保留随时对 ST 产品和 / 或本文档进行变更的权利，恕不另行通知。买方在订货之前应获取关于 ST 产品的最新信息。 ST 产品的销售依照订单确认时的相关 ST 销售条款。

买方自行负责对 ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。若需 ST 商标的更多信息，请参考 *www.st.com/trademarks*。所有其他产品或服务名称均为其 各自所有者的财产。

本文档是 ST 中国本地团队的技术性文章，旨在交流与分享，并期望借此给予客户产品应用上足够的帮助或提醒。若文中内容存有局限或与 ST 官网资料不一致，请以实际应用验证结果和 ST 官网最新发布的内容为准。您拥有完全自主权是否采纳本文档（包括代码，电路图等）信息，我们也不承担因使用或采纳本文档内容而导致的任何风险。

本文档中的信息取代本文档所有早期版本中提供的信息。