密级：公开资料

TTC SDK V3.x
使用说明

文档版本：V1.1

适用 SDK 版本：V3.x

深圳市昇润科技有限公司

2017 年 05 月 03 日

版本	修订日期	修订人	审稿人	修订内容
1.0	2017-03-31	郭高亮/廖健焜	张眼	初版发布
1.1	2017-05-03	郭高亮	张眼	1.增加 UART 引脚及时序描述 2.增加主机+广播角色功耗测试结果 3.增加程序剩余空间计算结果 4.增加 ADC 使用的注意事项 5.增加 128bit UUID 描述 6.增加 APP OAD 操作说明

目　　录

1. 概述

1.1. SDK 简介

TTC SDK 是由我司提供的CC2640 快速开发库，重点优化了蓝牙协议栈及 RTOS，且提供常用硬件驱动 API。

旨在让开发人员不再需要将大量精力放在蓝牙调试方面，只需将精力放在对 CC2640 功能上的开发。TTC SDK 提供了蓝牙参数设置、蓝牙数据收发、蓝牙状态处理等 API，同时也提供了生产时所需测试程序，开发人员无需再设计测试程序。

使用 TTC SDK 能适配我司提供的手机 APP(TTC-BLE)，方便调试数据收发，并且支持数据加密解密功能。能极大的缩短 CC2640 的开发周期。

在 BLE 通信中，GAP 角色分为主机、从机、观察者及广播模式。

照实际常用开发需求，TTC SDK V3.x 版本增加了更加丰富的蓝牙角色的示例，其中还包含两种组合模式。用户可使用相关 AT 指令即可完成不同角色的功能控制，可更高效的完成蓝牙主机、从机等角色的开发。

SDK 角色包含以下 4 种：

1. 从机
2. 从机+观察者组合模式
3. 主机
4. 主机+广播组合模式

通过 AT 指令即可完成的基本操作有：

1. 从机
   (1) 开启广播
   (2) 关闭广播
   (3) 向主机发送数据
   (4) 断开与主机的链接
2. 从机+观察者
   (1) 广播扫描
   (2) 开启广播
   (3) 关闭广播
   (4) 向主机发送数据
   (5) 断开与主机的链接
3. 主机
   (1) 扫描设备
   (2) 连接设备
   (3) 向从机发送数据
   (4) 断开与从机的链接
4. 主机+广播
   (1) 扫描设备
   (2) 连接设备
   (3) 向从机发送数据
   (4) 断开与从机的链接
   (5) 开启广播(不可链接)

(6) 关闭广播

以上 demo 示例使用 TTC SDK 开发套件、手机 APP(TTC-BLE)以及 PC 端软件(TTCDemo)即可完成功能演示。

另外，TTC SDK V3.x 也同时加入 AT 指令版本的驱动 demo。

1.2. 公共特征

- 蓝牙服务 UUID：1000

蓝牙通道	UUID	通道特性	功能概述
UUID1	1001	Write_NoRsp/Read/Notify	蓝牙数据接收
UUID2	1002	Read/Notify	蓝牙数据发送
UUID3	1003	Write_NoRsp	寄存器写数据
UUID4	1004	Read	寄存器读数据
UUID5	1005	Write_NoRsp/Read	选定寄存器

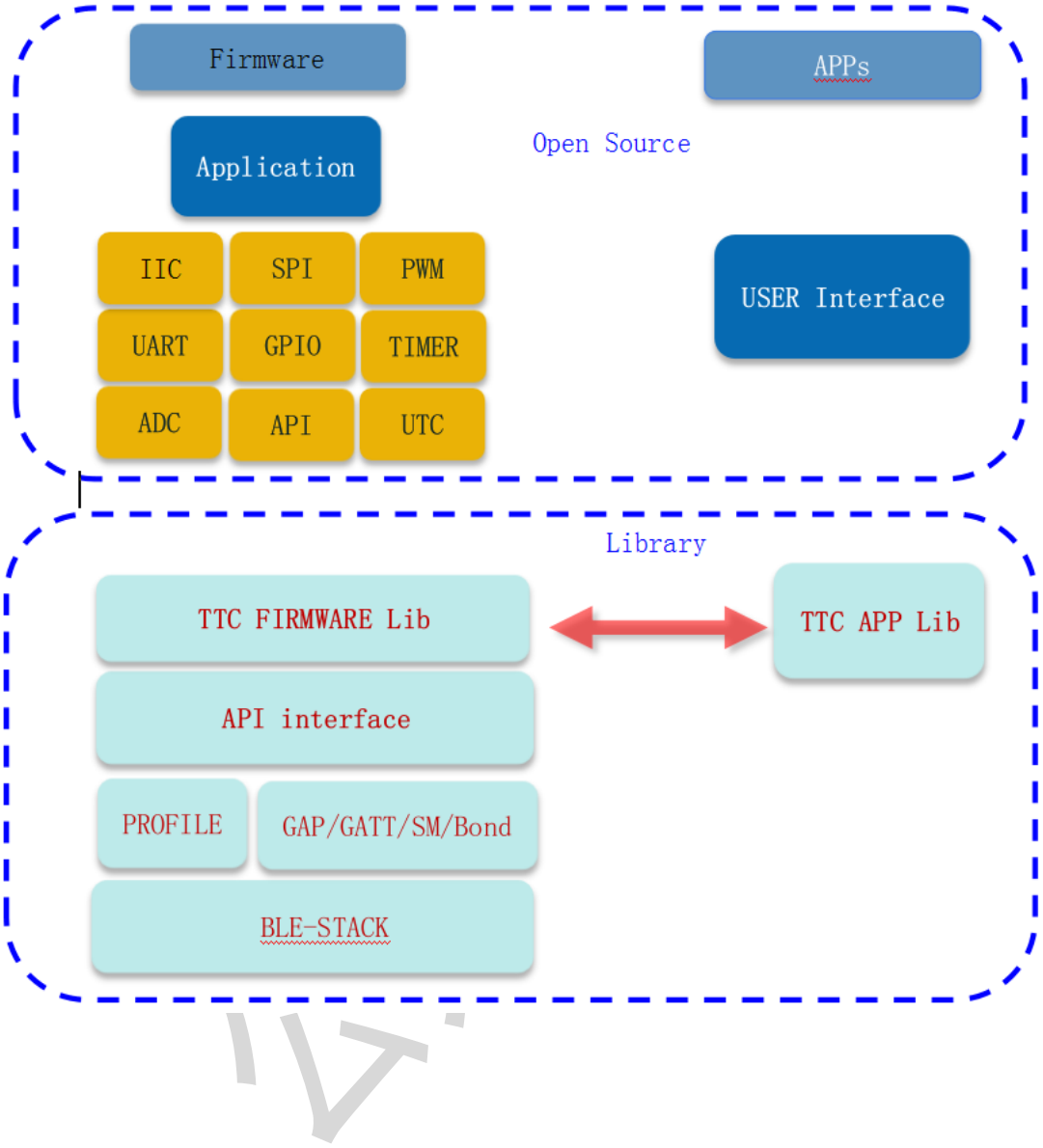
- 128 bit UUID 描述如下
  - Service: 00001000-0000-1000-8000-00805f9b34fb
  - UUID1 : 00001001-0000-1000-8000-00805f9b34fb
  - UUID2 : 00001002-0000-1000-8000-00805f9b34fb
  - UUID3 : 00001003-0000-1000-8000-00805f9b34fb
  - UUID4 : 00001004-0000-1000-8000-00805f9b34fb
  - UUID5 : 00001005-0000-1000-8000-00805f9b34fb

备注：主机、主机+广播：只能连接带有服务 UUID 0x1000 的从机设备（如 SDK 从机）。

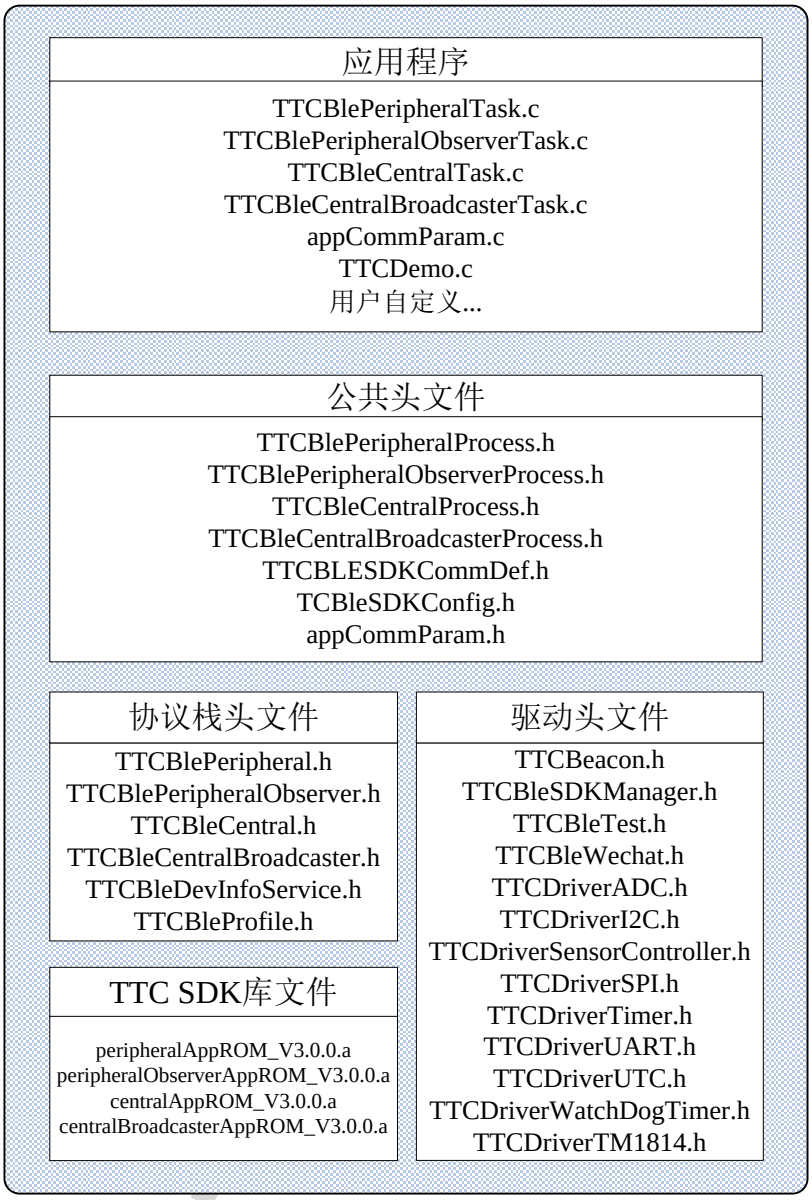
1.3. TTC SDK 优势

- 完整的蓝牙解决方案(IC+固件+APP+云端)
- 简单的蓝牙设置以及轻松更新固件
- 类似串口(UART)数据收发的蓝牙交互模式
- 完整的 SDK 以及工具提供
- 快速启动时间（RTOS < 500ms , OSAL < 500ms）
- 超低功耗特性，CC2640 低至 1.5uA 睡眠电流，可用电池供电
- 数据支持 AES 加密解密
- 完整的蓝牙参数验证，适应安卓、IOS 两大平台，用户无需烦恼蓝牙参数适配问题
- 完整的测试方案提供，SDK 已包含测试程序，用户无需设计蓝牙测试程序
- 配套的专业测试架、SDK 包、开发调试工具、DEMO 板
- 减少由于蓝牙导致设备工作异常的情况出现

1.4. SDK 结构图


Firmware
APPs
Application
Open Source
IIC
SPI
PWM
USER Interface
UART
GPIO
TIMER
ADC
API
UTC
Library
TTC FIRMWARE Lib
TTC APP Lib
API interface
PROFILE
GAP/GATT/SM/Bond
BLE-STACK

SDK 中文件的关系如下图：



TTCBleSDK 路径下各个文件介绍：

1. TTCDriverSPIFlash 及 TTCDriverSPISRAM 文件夹提供外部 Flash、RAM 操作的开源代码，方便用户灵活应用。

2. TTC SDK Lib 库文件，各自包含蓝牙及相关驱动程序：

(1) peripheralAppROM_V3.0.0.a
从机角色对应的库文件；

(2) peripheralObserverAppROM_V3.0.0.a
从机+观察者角色对应的库文件；

(3) centralAppROM_V3.0.0.a
主机角色对应的库文件；

(4) centralBroadcasterAppROM_V3.0.0.a
主机+广播角色对应的库文件。

3. 以下为库相关的头文件，每个头文件作用介绍如下：

(1)TTCBeacon.h

SDK 支持 Beacon 功能，此头文件包含 Beacon 相关参数初始值定义(宏定义)，参数设置、读取等相关函数的声明。

(2) TTCBleCentral.h

主机角色线程声明

(3) TTCBleCentralProcess.h

主机相关操作处理声明

(4)TTCBleCentralBroadcaster.h

主机+广播角色线程声明

(5)TTCBleCentralBroadcasterProcess.h

主机+广播角色相关操作处理声明

(6) TTCBleDevInfoService.h

此头文件与蓝牙设备信息有关，可以设置读取软件版本、硬件版本、生产厂商等信息。

(7) TTCBlePeripheral.h

从机角色线程声明

(8) TTCBlePeripheralProcess.h

从机相关操作处理声明

(9)TTCBlePeripheralObserver.h

从机+观察者角色线程声明

(10)TTCBlePeripheralObserverProcess.h

从机+观察者角色相关操作处理声明

(11)TTCBleProfile.h

此头文件与蓝牙服务有关，有设置蓝牙通道数据的声明，可通过此函数发送数据至 APP。

(12)TTCBleSDKConfig.h

此头文件与 GPIO 相关，包含 GPIO 所有函数的声明，以及相关参数的说明。

(13)TTCBleSDKManager.h

(14) TTCBleTest.h

此头文件与生产测试程序有关，包含注册用户自定义测试，以及测试程序版本回调函数。

(15) TTCBleWechat.h

SDK 支持微信(WeChat)功能，此头文件包含设置微信广播数据、微信蓝牙初始化、添加微信蓝牙处理、微信数据发送等函数的声明，及相关使用说明。

(16) TTCDriverADC.h

此头文件与 ADC 有关，包含 ADC 采样等函数的声明，以及相关参数说明。

(17) TTCDriverI2C.h

此头文件与 I2C 有关，包含 I2C 读写等函数的声明，以及相关参数及使用说明。

(18) TTCDriverSensorController.h

此头文件仅包含 SensorController 的初始化函数，若项目开发需要使用 SensorController，更多信息可与我们联系。

(19) TTCDriverSPI.h

此头文件与 SPI 有关，包含 SPI 读写等函数的声明，以及相关参数及使用说明。

(20) TTCDriverTimer.h

此头文件与硬件定时器有关，包含 PWM 输出、计时、定时等功能相关的函数声明，以及相关参数及使用说明。

(21) TTCDriverUART.h

此头文件与 UART 有关，包含 UART 发数据函数的声明，以及相关参数及使用说明。

(22) TTCDriverUTC.h

此头文件与 UTC 有关，包含 UTC 设置时间、读取时间等函数的声明，以及相关参数及使用说明。

(23) TTCDriverWatchDogTimer.h

此头文件与看门狗有关，包含看门狗初始化及“喂狗”函数的声明，以及相关参数及使用说明。

1.5. 程序空间

SDK 中含有 4 种蓝牙角色，部分角色根据不同的功能有不同的配置，配置区别以及可用空间计算如下表。

说明：以下程序空间计算时，均未使能驱动 Demo 相关功能，即在默认程序的基础上再屏蔽宏定义 TTC_DEBUG 及 TTCDRIVER_UART。

1.5.1. 从机示例空间

OAD	测试程序	ROM(byte)			RAM(byte)		
		总量	用量	剩余	总量	用量	剩余
无	无	94208	29299	64909	16360	8832	7528
	有		38426	55782		10319	6041
片内	无	69632	35118	34514	16360	9149	7211
	有		44484	25148		10636	5724
片外	无	94208	36710	57498	16360	9486	6874
	有		45436	48772		10973	5387

1.5.2. 从机+观察者示例空间

OAD	测试程序	ROM(byte)			RAM(byte)		
		总量	用量	剩余	总量	用量	剩余
无	无	94208	30426	63782	16304	9156	7148
	有		39585	54623		10635	5669
片内	无	69632	36239	33393	16304	9473	6831
	有		45630	24002		10952	5352
片外	无	94208	37834	56374	16304	9810	6494
	有		46596	47612		11289	5015

1.5.3. 主机示例空间

OAD	测试程序	ROM(byte)			RAM(byte)		
		总量	用量	剩余	总量	用量	剩余
无	无	81920	26605	55315	16280	7728	8552

1.5.4. 主机+广播示例空间

OAD	测试程序	ROM(byte)			RAM(byte)		
		总量	用量	剩余	总量	用量	剩余
无	无	81920	27795	54125	16280	7880	8400

1.6. SDK 功耗

说明：以下不同角色功耗测试，均未使用 UART 等驱动。

1. 从机角色

蓝牙状态	设置参数	实际间隔时间(ms)	平均电流(uA)
关闭广播	\	\	1.47
广播间隔	32	20	925.73
	160	100	257.13
	800	500	52.45
	1600	1000	26.60
连接间隔	16	20	455.0
	24	30	287.08
	80	100	97.52
	160	200	48.29
	400	500	23.66
	800	1000	15.48

2. 从机+观察者角色

蓝牙状态	设置参数	实际间隔时间(ms)	平均电流(uA)
关闭广播	\	\	1.47
广播间隔	32	20	1010
	160	100	267.82
	800	500	57.35
	1600	1000	30.22
连接间隔	24	30	282.55
	80	100	91.65
	160	200	48.36
	400	500	22.90
	800	1000	14.83
未连接扫描	\	(1000ms 广播)	6054
连接后扫描	\	(500ms 连接间隔)	6339

3. 主机角色

蓝牙状态	设置参数	实际间隔时间(ms)	平均电流(uA)
未连接	\	\	1.51
连接间隔	24	30	272.23
	80	100	87.85
	160	200	45.85
	400	500	45.23
	800	1000	21.55

4.主机 + 广播角色

蓝牙状态	设置参数	实际间隔时间(ms)	平均电流(uA)
未连接关闭广播	\	\	1.48
不同广播间隔 未连接	160	100	167.24
	800	500	35.84
	1600	1000	18.53
不同连接间隔 不广播	24	30	278
	80	100	95.8
	160	200	51.2
	400	500	23.3
	800	1000	14.7
连接间隔 500ms 不同广播间隔	160	100	195
	800	500	58
	1600	1000	37
广播间隔 500ms 不同连接间隔	24	30	312
	80	100	124
	160	200	85
	400	500	65.1
	800	1000	54.8

备注：

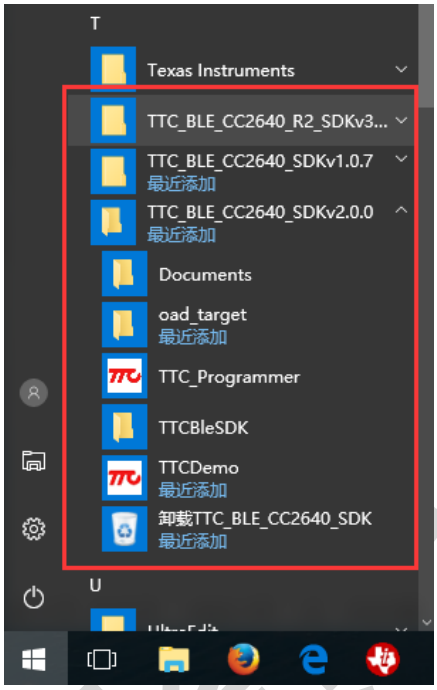
(1)非连接广播，最小广播间隔为 100ms.

(2)主机+广播角色功耗测试时，不同连接间隔与不同的广播间隔可以有多重组合。从上表可以得出的规律：连接时再开启广播的功耗，大致等于连接功耗与广播功耗之和。

(3)正在扫描时，功耗为 6mA 左右。

2. 工具使用说明

SDK 安装完成后，可以在 Windows 开始菜单中找到对应的快捷方式，如下图。PC 端软件(TTCDemo)可从快捷方式启动，手机 APP(TTC-BLE)下载二维码方式见附录 A。



2.1. 模组连接

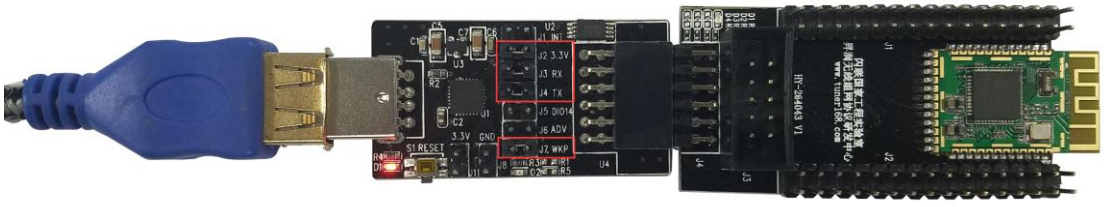
SDK demo 相关功能是使用 AT 指令的方式实现的，若需要演示 dmeo 功能，则需要使用到 USB 转串口工具，以下介绍两种连接方式。

➢ 方式一：使用 USB 转接板

不同封装的 CC2640 模组，所使用的 USB 转接板是一致的，则连接方式也一致。CC2640 模组通过 USB 转接板（需用跳线帽短接 J2-3.3V、J3-RX、J4-TX、J7-WKP），连接至电脑 USB 接口，S1(RESET)为复位按钮，如下图。从机上电后，demo 默认广播名称为”TTC_CC2640_SDK”。

注意：程序默认使用 CC2640 R2 6*6 封装，也可修改为其他封装。可通过宏定义选择，定义方法见2.4.2 节。

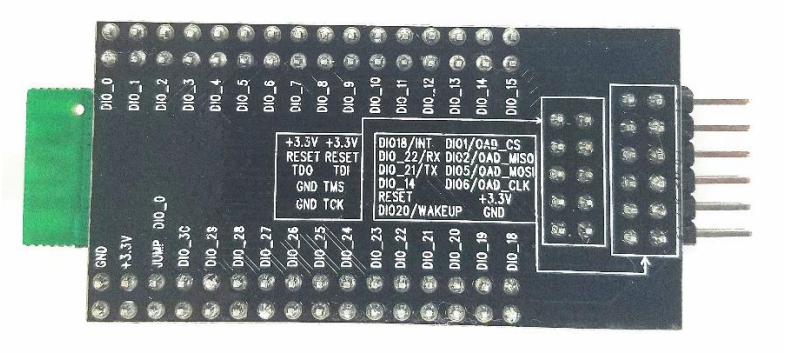
CC2640 IC 版本	封装	对应的宏
R2	6*6	CC26XX_R2/CC2650DK_6ID
	4*4	CC26XX_R2/CC2650DK_4ID
R1	7*7	CC26XX_R1/CC2650DK_7ID
	5*5	CC26XX_R1/CC2650DK_5XD



➢ 方式二：使用其他 USB 转串口工具

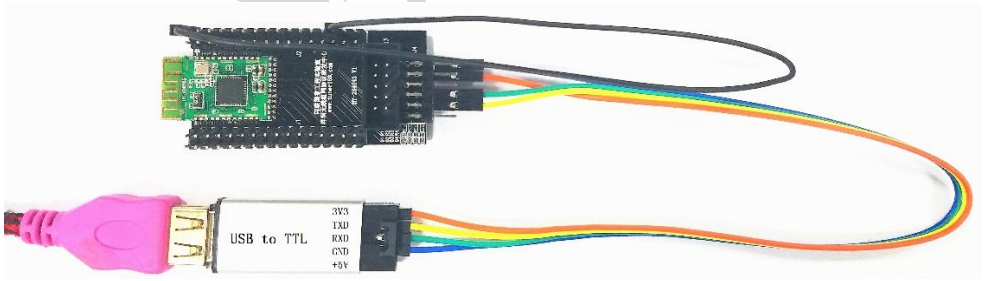
如果没有以上的 USB 转接板，则可以使用常见的 USB 转 UART 的小工具实现连接。需注意，模块供电范围是 1.8V~3.8V，可以使用 3.3V 供电，RX/TX 信号的高电平也应是对应的 3.3V。模组与小工具需要 5 条杜邦线连接，分别是 VCC/GND/WAKEUP/RX/TX.

HY-264043V1（CC2640 6*6 模组）标明标明引脚排序如下：



在测试时，可将透传模组 WAKEUP(DIO20)引脚接 GND，使模组一直处于唤醒状态，便于测试。最终连接如下：

USB to TTL 工具	HY-264022V1（CC2640 模组）
3V3	+3.3V
TXD	RX
RXD	TX
\	DIO20 ←→ GND
GND	GND

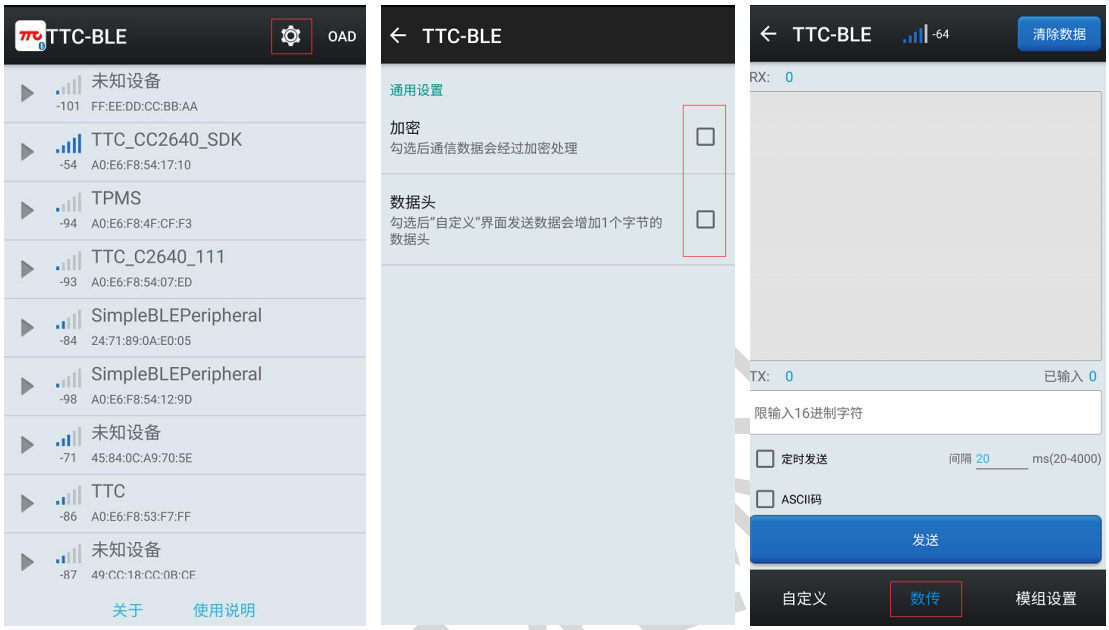


2.2. 手机 APP(TTC-BLE)使用

安装手机 APP(TTC-BLE)，进入扫描界面，如下图 1；

点击设置，取消勾选“加密”(TTC SDK demo 在传输数据时默认设置为不加密)、“数据头”，如下图 2；

点击相应的蓝牙设备，如 “TTC_CC2640_SDK”，即可与设备建立链接，进入“数传”界面，如下图 3。

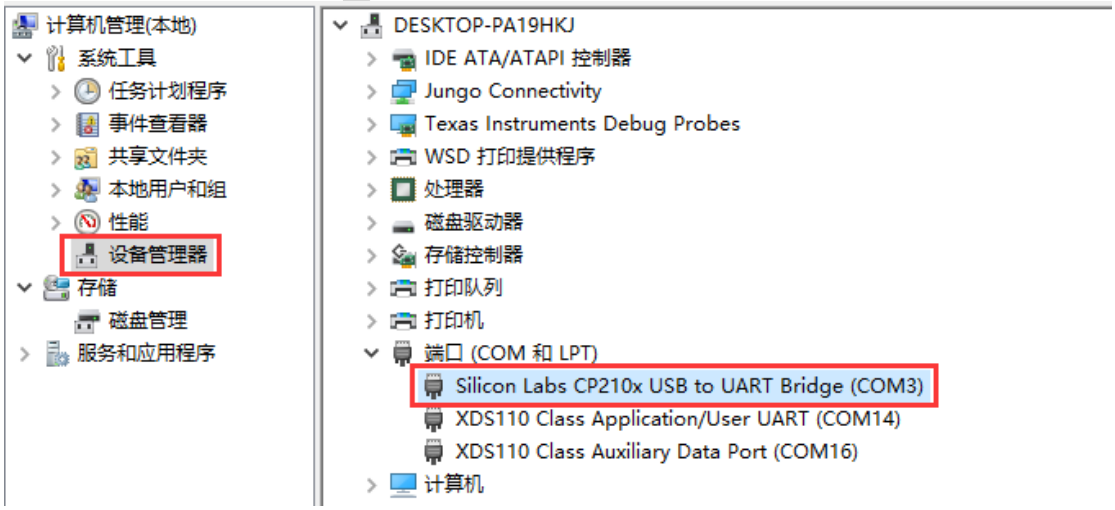


2.3. PC 端软件(TTCDemo)使用

(1) 串口设置(UART)

选择对应的串口号，数据格式为：长度 8bit，无校验位，1bit 停止位，设置波特率为 115200bps(TTC SDK demo 默认波特率)，点击“打开串口”，如下图。

设备管理器查看 UART 端口号，选择 Silicon Labs Cp210x USB to UART Bridge 对应的端口号，请勿选择 XDS110 虚拟出的端口，如下图：



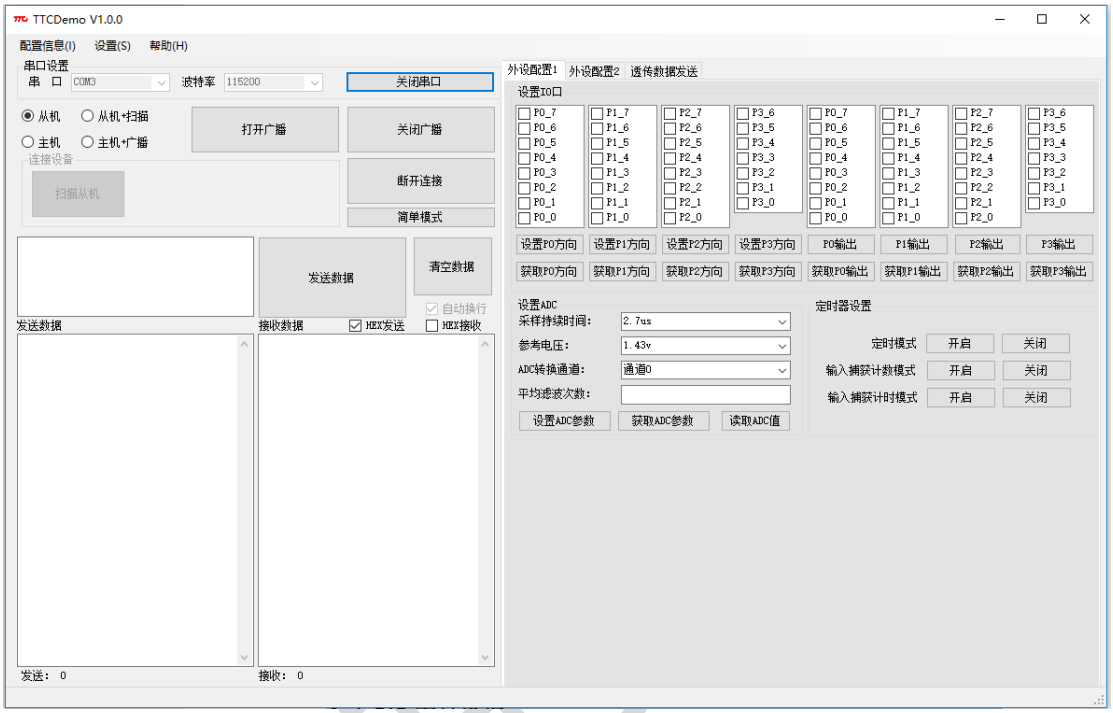
(2) 蓝牙角色选择

根据 demo 程序所实现的蓝牙角色，设置 PC 端软件(TTCDemo)对应的角色。不同角色时，所支持的 AT 指令不一样，灰色为不可使用的指令。如下图。

(3) 外设配置

“外设配置 1”包含 GPIO、ADC 及定时器的设置，“外设配置 2”包含 PWM、IIC、SPI 及 UTC 的配置，如下图。

(4) 透传数据发送



2.4. IAR 常见操作

2.4.1. IAR 版本

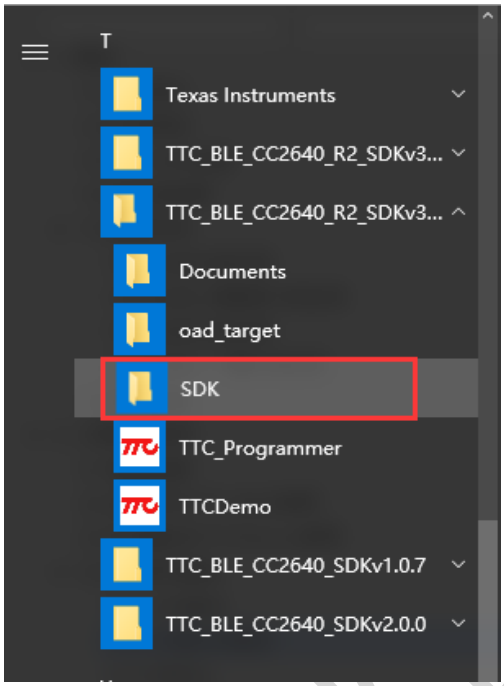
TTC SDK V3.x 使用 IAR 7.80.4 版本，开启 IAR 时需要以管理员身份运行。

2.4.2. 工程文件路径

(1)IAR WorkSpace 文件：SDK.eww

(2)默认所在路径，如 SDK 3.1.0 所在路径如下，并可通过开始菜单的快捷方式快速访问。

C:\TTC_BLE_CC2640_R2_SDK\3.1.0\TTC_CC2640_R2_SDK\examples\rtos\CC2640R2_LAUNCHXL\blestack\SDK



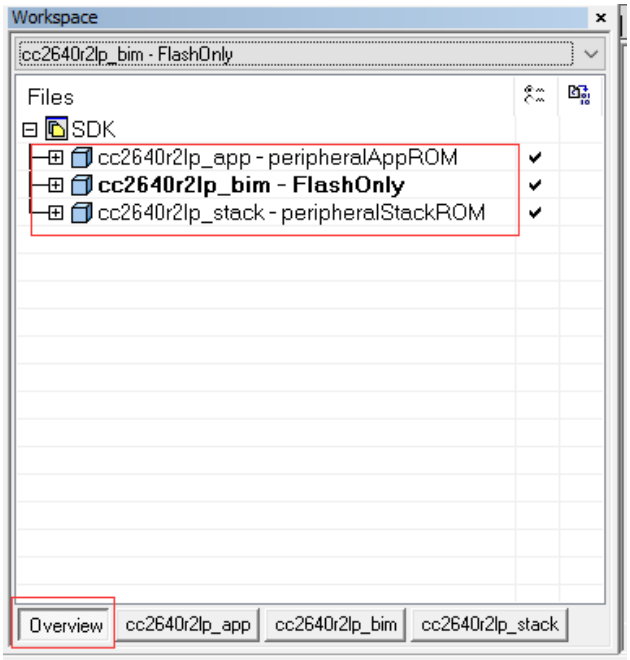
2.4.3.宏定义

蓝牙角色 demo 及驱动 demo 均需要调整部分宏定义，操作步骤如下：

IAR -> Project -> Options -> C/C++ Compiler -> Preprocessor -> Defined symbols 选项中定义，例如 TTCDRIVER_GPIO 为 GPIO demo 运行所需要的宏，删除宏如 TTCDRIVER_GPIOx，以便再次开启。

2.4.4.工程配置

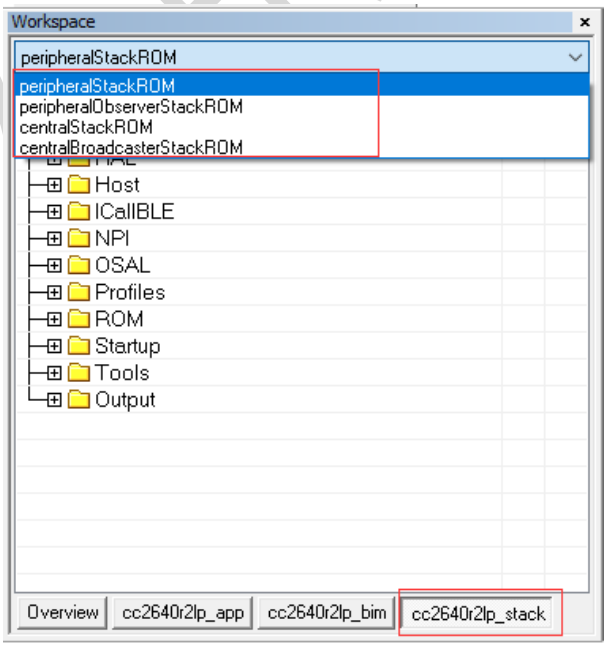
1. TTCBleSDK WorkSpace 包含 3 个工程：
   (1) cc2640r2lp_app：应用程序工程
   (2) cc2640r2lp_bim：仅用于片外 OAD
   (3) cc2650_stack：协议栈工程



2.每个工程有不同的配置

(1)协议栈工程的 4 个配置

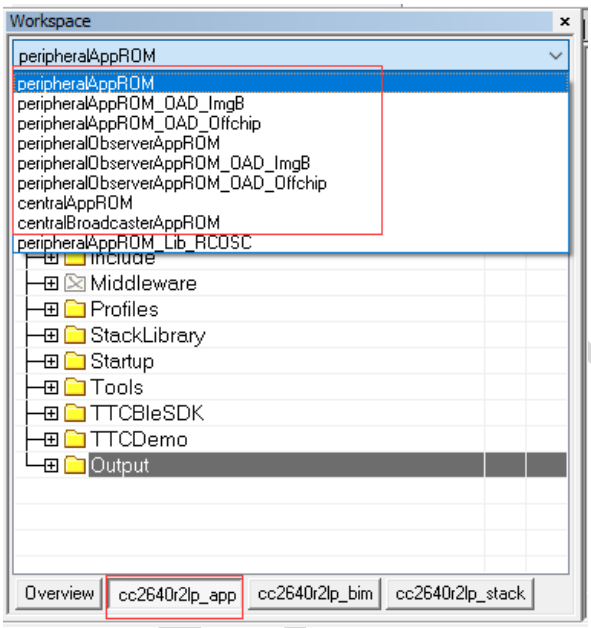
- peripheralStackROM：从机协议栈
- peripheralObserverStackROM：从机+观察者协议栈
- centralStackROM：主机协议栈
- centralBroadcasterStackROM：主机+广播协议栈



(2)应用程序工程的 8 个配置

注意：带 OAD 功能的配置，不支持在线调试仿真，不可直接使用 IAR 下载程序，相关操作请参见章节 5 空中升级说明（OAD）.

- peripheralAppROM：从机应用程序
- peripheralAppROM_OAD_ImgB：从机（支持片内 OAD）应用程序
- peripheralAppROM_OAD_OffChip：从机（支持片外 OAD）应用程序
- peripheralObserverAppROM：从机+观察者应用程序
- peripheralObserverAppROM_OffChip：从机+观察者（支持片外 OAD）应用程序
- centralAppROM：主机应用程序
- centralBroadcasterAppROM：主机+广播应用程序
- FlashROM_RCOSC:无需使用

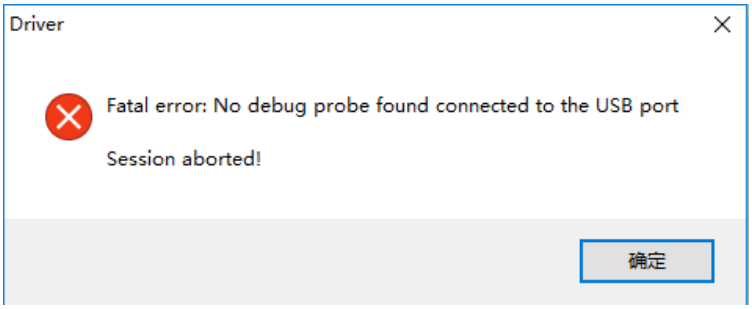


3.demo 程序，需要先下载协议栈程序，再下载应用程序。

注意：协议栈工程配置，必须与应用程序工程配置匹配。比如协议栈使用从机功能 peripheralStackROM，那么应用程序也必须是从机的配置，可以是 peripheralAppROM、peripheralAppROM_OAD_ImgB、peripheralAppROM_OAD_OffChip。

2.4.5.IAR 程序仿真提示错误

1.问题：使用 IAR 仿真下载程序时，如果出现以下错误提示“Fatal error: Nodebug probe found connected to the USB port Session aborted!”



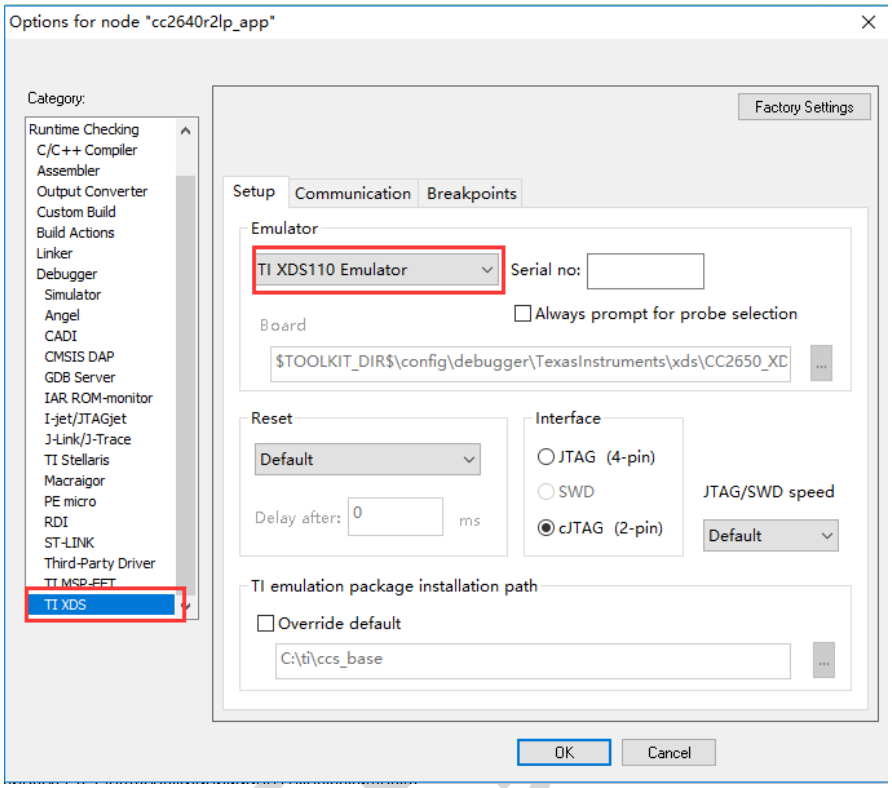
2. 出错原因

(1) IAR 配置不正确；

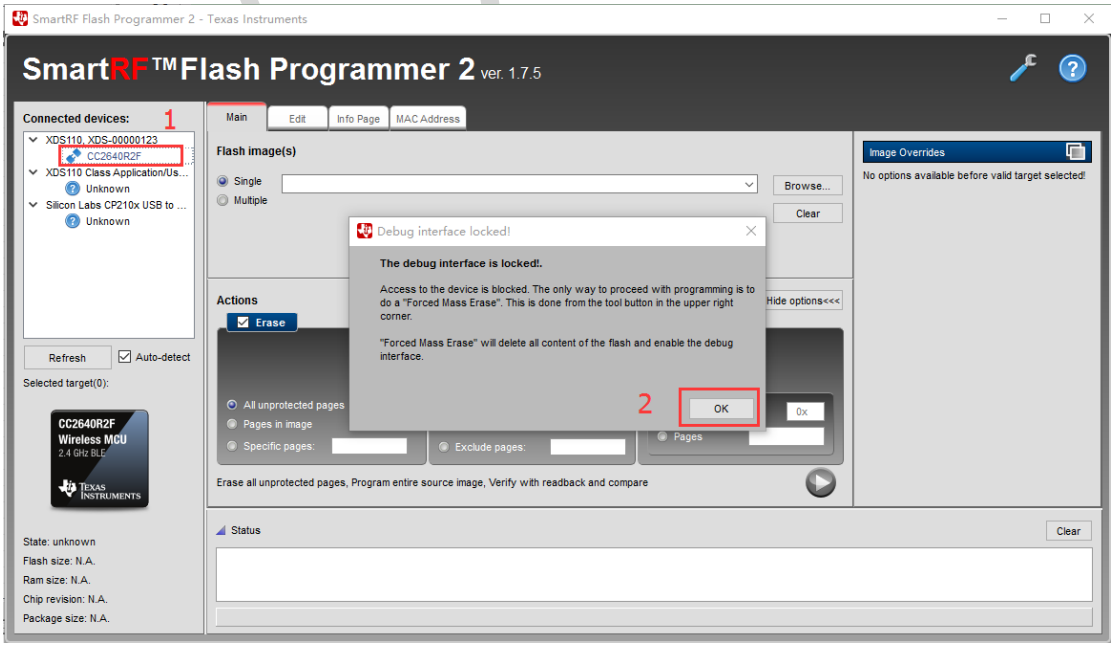
(2) 模组可能已经烧录了我司的透传程序，并且调试接口已经被锁住。

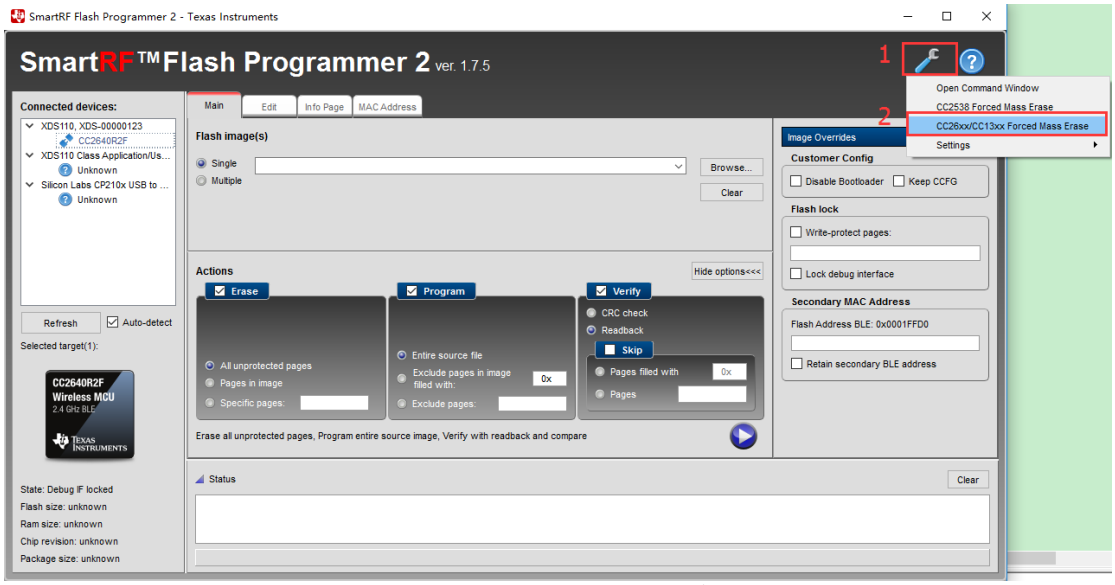
3. 解决方法

(1) 修改配置：选择正确的下载器，如图



(2) 使用 Flash Programmer 2,若提示“The debug interface is locked!”，需强制擦除默认程序，解锁调试接口.





2.5. 仿真器 XDS110 调试引脚说明

在线仿真调试，XDS110 使用 4 pin 仿真模式时，会占用 GPIO(TDI/TDO),导致对应 GPIO 口功能异常。当然，不连接下载器 XDS110 时，GPIO(TDI/TDO)功能正常。GPIO 与 TDI/TDO 的对应关系，见《CC2640R2F Datasheet_SWRS204A》。如 CC2650DK_6ID 封装 DIO_16 对应 TDO，DIO_17 对应 TDI,如下：

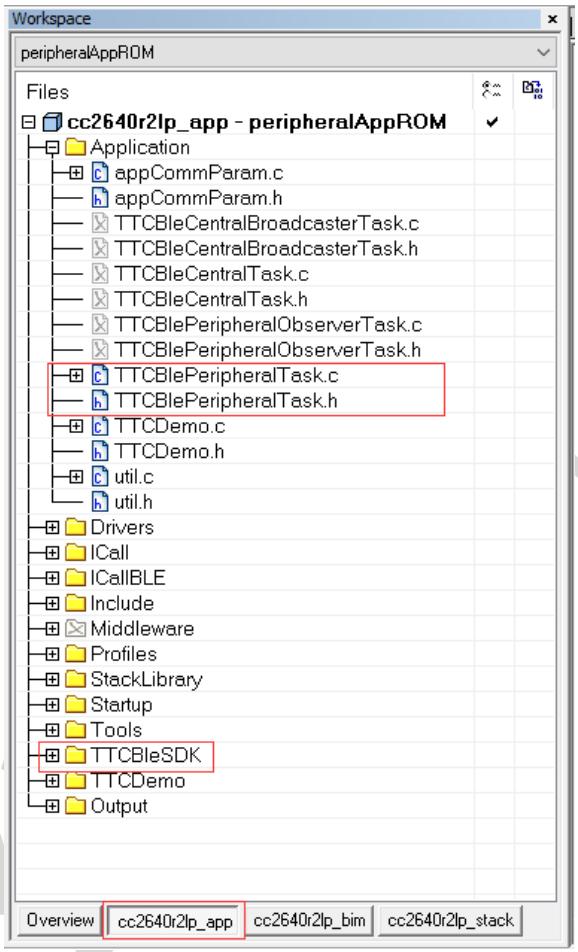
DIO_16	26	Digital I/O	GPIO, JTAG_TDO, high-drive capability
DIO_17	27	Digital I/O	GPIO, JTAG_TDI, high-drive capability

3. 蓝牙角色 Demo 说明

TTC SDK V3.x 蓝牙角色 demo 通过 AT 指令的方式，实现不同功能的控制。在调试之前，需要先使能串口(UART)demo，以处理 AT 指令。

3.1. 整体结构介绍

以从机工程为例，简要说明应用程序中蓝牙角色 demo 的整体结构：



1.<TTCBlePeripheralTask.c>文件

此文件包含角色 demo 的初始化，及相关事件、消息的处理。

(1)整 TTCBlePeripheralTaskFxn()

```
static void TTCBlePeripheralTaskFxn(UArg a0, UArg a1){
    //1.初始化,包含蓝牙角色 demo 初始化
    TTCBlePeripheralTaskInit();
    ……
    //2.等待线程唤醒
    events = Event_pend(syncEvent,Event_Id_NONE,TTC_ALL_EVENTS,ICALL_TIMEOUT_FOREVER);
    ……
    //3.消息处理
    TTCBlePeripheralTaskProcessAppMsg(pMsg);
    ……
}
```

(2)初始化函数 TTCBlePeripheralTaskInit()

TTCBlePeripheralConfigSet()函数包含蓝牙参数设置：

```
static void TTCBlePeripheralTaskInit(void){
  ……
//1.注册事件、消息并添加蓝牙服务等
  ICall_registerApp(&selfEntity, &syncEvent);
  appMsgQueue = Util_constructQueue(&appMsg);          // 创建消息队列
  GAP_RegisterForMsgs(selfEntity);                     //注册 HCI 消息
  GATT_RegisterForMsgs(selfEntity);                    //注册 GATT 事件和 ATT 回应绑定
  ……
  Reset_addService();                                  //复位服务

 //2.设置蓝牙参数
  TTCBlePeripheralConfigSet();                         //初始化蓝牙配置
  ……
}
```

(3)消息处理函数 TTCBlePeripheralTaskProcessAppMsg()

TTCDriverUartProcess()包含蓝牙角色 demo、驱动 demo 的 AT 指令处理。

```
static void TTCBlePeripheralTaskProcessAppMsg(TTCMsg_t *pMsg){
  ……
  //1.处理收到的蓝牙数据（UUID:0X1002）
  case TTCSDK_MSG_GET_BLE_DATA_EVENT:{
    TTCBlePeripheralTaskGetBleData( (TTCMsg_t *)pMsg );
  }break;

  //2.定时刷新 RSSI
  case TTCSDK_MSG_REFRESH_RSSI_EVENT:{
    TTCBlePeripheralTaskRefreshRSSI( (s8 *)pMsg->pValue );
  }break;

  //3.获取蓝牙先关参数
  case TTCSDK_MSG_GET_BLE_PARAM_EVENT:{
    TTCBlePeripheralTaskGetBleParam( (TTCBleParamUpdate_t *)pMsg->pValue );
  }break;

  //4.串口消息处理：解析串口接收到的 AT 指令(包含蓝牙角色 demo 及驱动 demo 的指令)
  case TTCSDK_MSG_DRIVER_UART_EVENT:{
      TTCDriverUartProcess(pMsg);
  }break;
  ……
```

```
}
```

(4)从机 demo 相关 AT 指令处理：

```
static void TTCDriverUartProcess(TTCMsg_t * TTCMsg) {
    ……
    //1.断开连接指令处理
    if(!memcmp(buf,CMD_Disconnect,DISCONNECT_AT_LEN)){
    ……
    //2.发送数据指令处理
    }else if(!memcmp(buf, CMD_SendData, BLE_SEND_DATA_AT_LEN)){
    ……
    //3.广播开关指令处理
    }else if(!memcmp(buf, CMD_advState, BLE_ADV_STATE_AT_LEN)){
    ……
}
```

2.<appCommParam.c>文件

驱动 demo 均使用了串口(UART)接收 AT 指令，此文件为驱动 demo 的公共文件：包含串口初始化(包含读写回调)及调试信息打印函数。

3.<TTCDemo.c>文件

包含所有驱动 demo 的初始化、驱动 demo AT 指令表、驱动 demo AT 指令解析并处理。

```
CMDState_t TTCDemoProcessCmd(u8 * buf,u16 size){
   ……
   //1.查找表，解析 AT 指令
   for( u8 i = 0 ; i < TTCDEMO_CMD_TABLE_SIZE ; i++ ){
     if(!memcmp(buf,TTCDemoCmdTable[i].cmd,TTCDemoCmdTable[i].cmdSize)){
   ……

   //2.根据 AT 指令，实现不同功能
   status = TTCDemoCmdTable[i].cmdApi(i,operation,p,&dataSize,TTCDemoCmdTable[i].cmdSize);
   ……
}
```

4.<TTCDemo>组

包含库文件(蓝牙及驱动)及相关头文件，头文件中有相关 API 说明。

5.<TTCBleSDK>组

包含所有的驱动相关 demo。

3.2. 从机示例

3.2.1. 示例功能说明

3.2.1.1 功能概述

1. 手机 APP(TTC-BLE)与 CC2640 模块建立连接后，PC 端软件(TTCDemo)通过 UART(115200bps)下发数据给 CC2640 模块，CC2640 模块接收到数据后，通过 BLE 对应的数据上行通道(UUID：0X1001)转发给手机 APP 端；数据流方向，如下图红线所示。

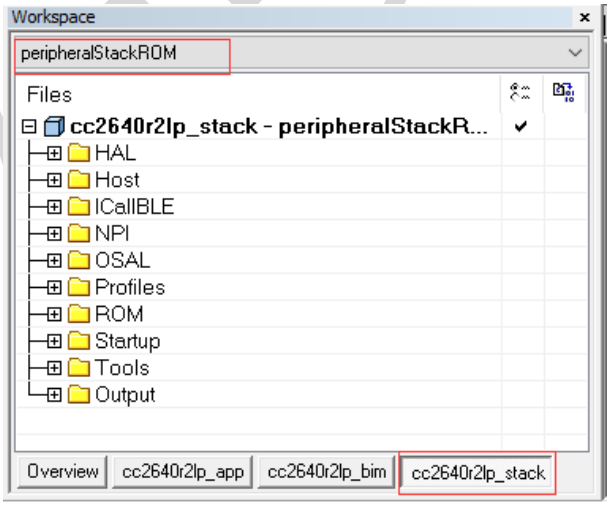
2. 手机 APP(TTC-BLE)与 CC2640 模块建立连接后，手机 APP 端通过 BLE 对应的数据下行通道(UUID：0X1002)发送数据给 CC2640 模块，CC2640 模块接收到数据后，通过 UART(115200bps)将数据转发给 PC 端软件(TTCDemo)；数据传输方向，如下图蓝线所示。

当然，CC2640 从机也可通过串口(UART)与其它 MCU 通信。

3.2.1.2.工程配置

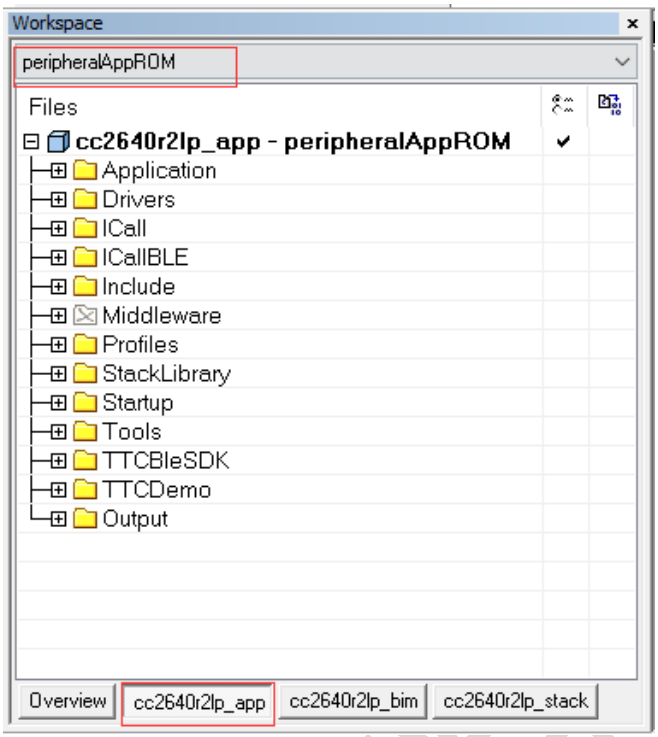
1.协议栈程序部分(cc2640r2lp_stack)

选中“cc2640r2lp_stack”选项卡，然后在配置下拉框中选择“peripheralStackROM”，重新全编译并下载程序,如下图:



2.应用程序部分(cc2640r2lp_app)

选中“cc2640r2lp_app”选项卡，然后在配置下拉框中选择“peripheralAppROM”，重新全编译并下载程序,如下图:



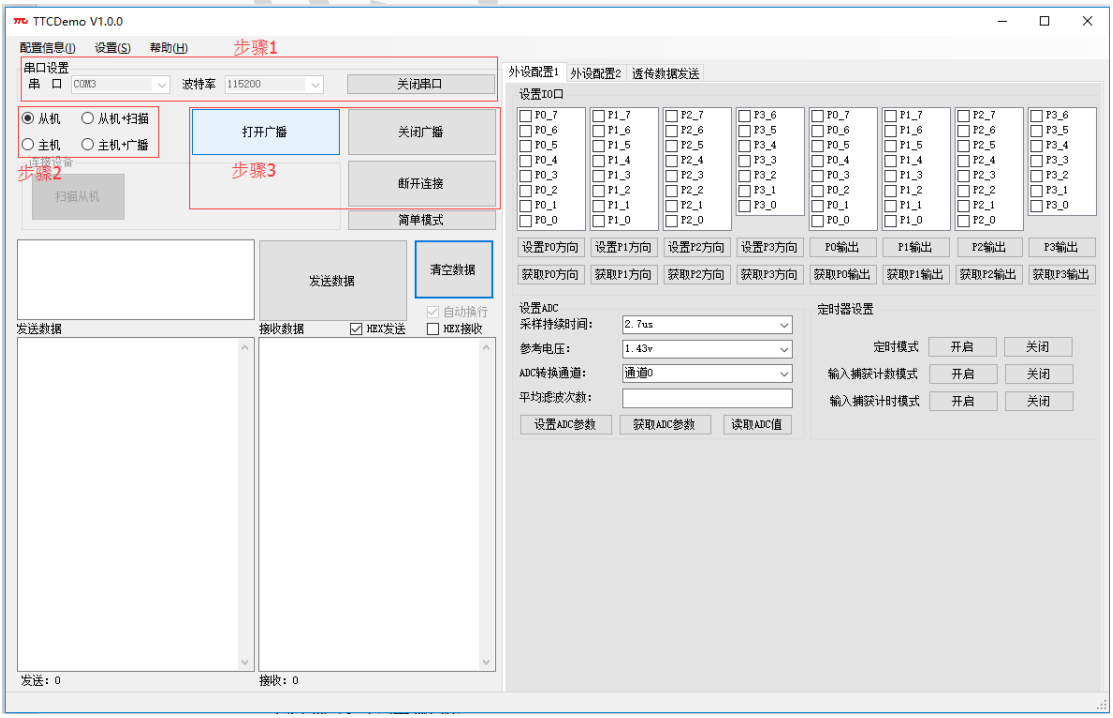
3.2.1.3.操作步骤

1. APP、PC 端软件(TTCDemo)及模组的连接，相关基本操作参照章节 2。

2. 蓝牙角色相关功能

(1)角色选择

选择“从机”角色，在未建立链接的情况下，点击相应按钮，可实现“打开广播”、“关闭广播”功能；建立链接之后，可进行数据收发，“断开连接”按钮可断开与主机的链接；如下图。

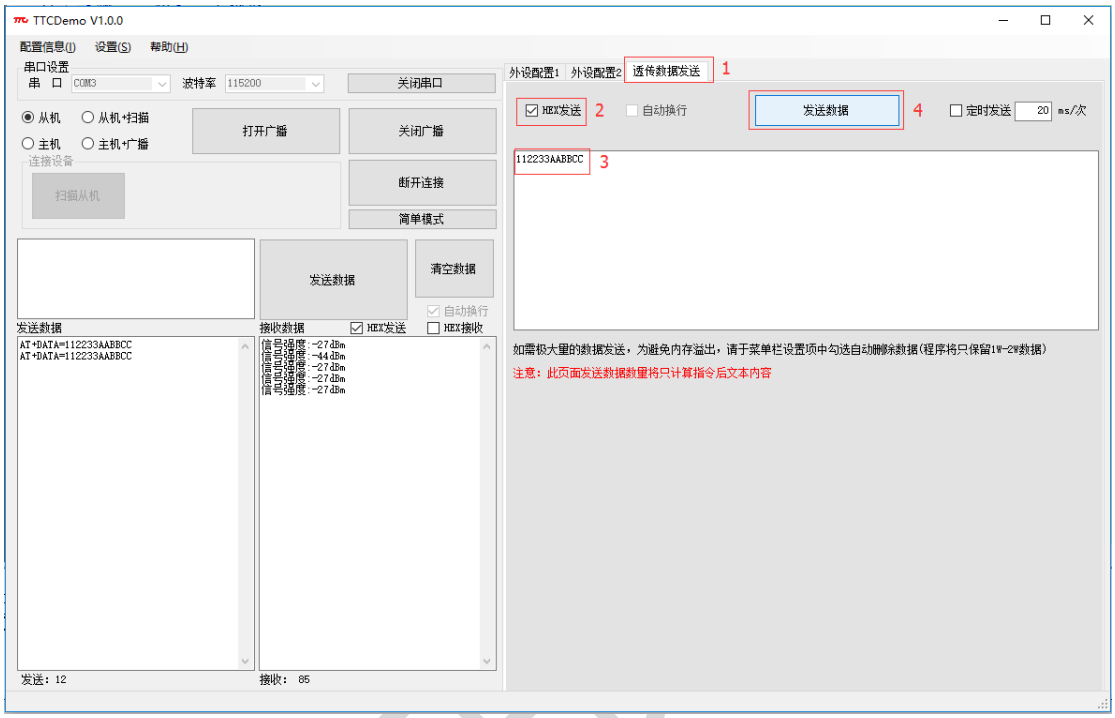


(4)蓝牙发送数据

数据流向:PC端软件(TTCDemo) → CC2640模块(从机) → 手机APP(TTC-BLE)

➢ PC端软件(TTCDemo)发送数据:

如下图,0x112233AABBCC为用户输入的6Byte数据,PC端软件(TTCDemo)自动按照“AT+DATA=112233AABBCC\r\n”格式发送。



CC2640模块(从机)收到PC端软件(TTCDemo)发送的数据后,将数据转发给手机APP(TTC-BLE):

➢ 手机APP(TTC-BLE)接收数据

如下图, 0x112233AABBCC为收到的6Byte数据。

(5)接收蓝牙数据

数据流向：手机 APP(TTC-BLE) → CC2640 模组(从机)→ PC 端软件(TTCDemo)接收

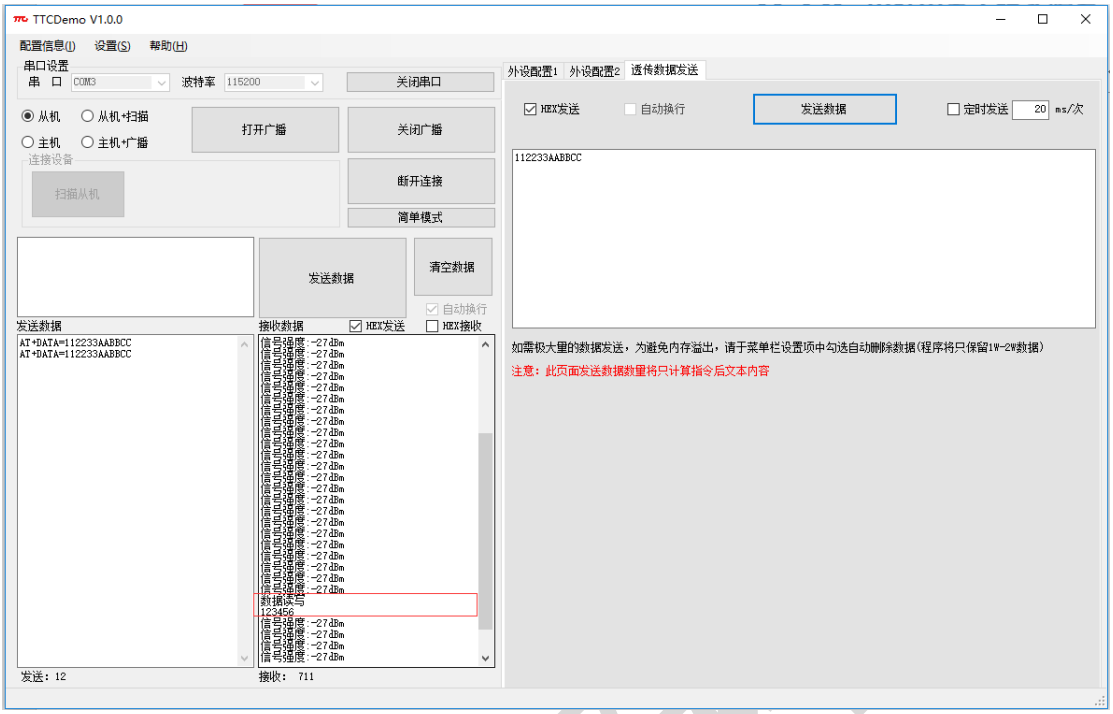
- 手机 APP(TTC-BLE)发送

  如下图，0x3132333435360D0A,对应字符“123456\r\n”

CC2640 模组(从机)接收到手机 APP(TTC-BLE)的数据，将数据发送给 PC 端软件(TTCDemo)：

- PC 端软件(TTCDemo)接收数据

接收字符为“123456\r\n”



3.2.2. API 说明

见 TTCBlePeripheral.h 及 TTCBlePeripheralProcess.h

3.2.3. 示例

见 TTCBlePeripheralTask.c，以下为从机处理接收到串口指令的部分代码：

```
/*******************************************************************************
【函    数】  TTCDriverUartProcess(TTCMsg_t * TTCMsg)
【概    述】  从机处理接收到串口指令
【入口参数】  TTCMsg : 处理串口接收到的数据
*******************************************************************************/
static void TTCDriverUartProcess(TTCMsg_t * TTCMsg) {
    u16 len  = TTCMsg->hdr.state;
    u8 buf[TTCDEMO_MAX_UART_RXSIZE];
    memset(buf,NULL,TTCDEMO_MAX_UART_RXSIZE);
    memcpy(buf, TTCMsg->pValue, TTCMsg->hdr.state);
    if(!memcmp(buf,CMD_Disconnect,DISCONNECT_AT_LEN)){
    //断开连接指令 CMD_Disconnect
        TTCBlePeripheralGetParameter(GAPROLE_CONNHANDLE, &connHandle);
        if(TTCBlePeripheralGAPRoleTerminateConnection(connHandle) == SUCCESS){
            TTCDebugLogPrint("断开......\r\n", strlen("断开......\r\n"));
        }else{
            TTCDebugLogPrint("断开失败\r\n", strlen("断开失败\r\n"));
        }
```

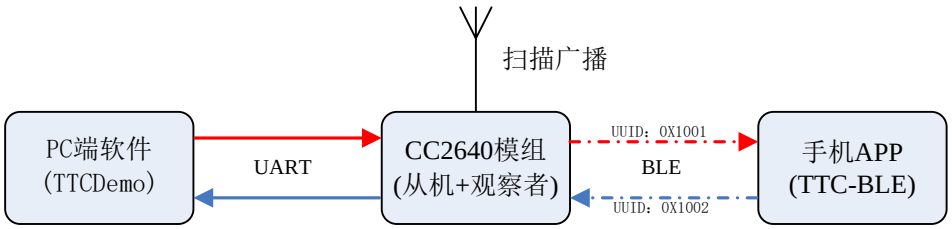
```
    }else if(!memcmp(buf, CMD_SendData, BLE_SEND_DATA_AT_LEN)){
    //发送数据指令 CMD_SendData
        u8 endChar0 = *(buf + len - 2);
        u8 endChar1 = *(buf + len - 1);
        if( endChar0 == '\r' && endChar1 == '\n'){
            u8 * send = NULL;
            u8 sendLen = len - BLE_SEND_DATA_AT_LEN - 2;
            send  = buf;
            send += BLE_SEND_DATA_AT_LEN;
#ifndef TTCBLE_WECHAT
            TTCBleProfileSetParameter(TTCBLE_PROFILE_CHAR2,
                                      sendLen,
                                      (u8*)send);
#else
            TTCDemoWechatSendData((const u8*)send,sendLen);
#endif
        }
    }else if(!memcmp(buf, CMD_advState, BLE_ADV_STATE_AT_LEN)){
    //发开关广播指令 CMD_advState 开启“1”广播/“0”关闭广播
        u8 endChar0 = *(buf + len - 2);
        u8 endChar1 = *(buf + len - 1);
        u8 advSwitch = 0;
        if( endChar0 == '\r' && endChar1 == '\n'){
            if(*(buf+BLE_ADV_STATE_AT_LEN) == '0') {
              advSwitch = 0;
              TTCBlePeripheralSetParameter(GAPROLE_ADVERT_ENABLED,
                                           sizeof(u8),
                                           &advSwitch);
            }else if(*(buf+BLE_ADV_STATE_AT_LEN) == '1'){
              advSwitch = 1;
              TTCBlePeripheralSetParameter(GAPROLE_ADVERT_ENABLED,
                                           sizeof(u8),
                                           &advSwitch);
            }
        }
    }else{
        TTCDemoProcessCmd(buf,len);
    }
}
```

3.3. 从机+观察者示例

3.3.1. 示例功能说明

3.3.1.1.功能概述

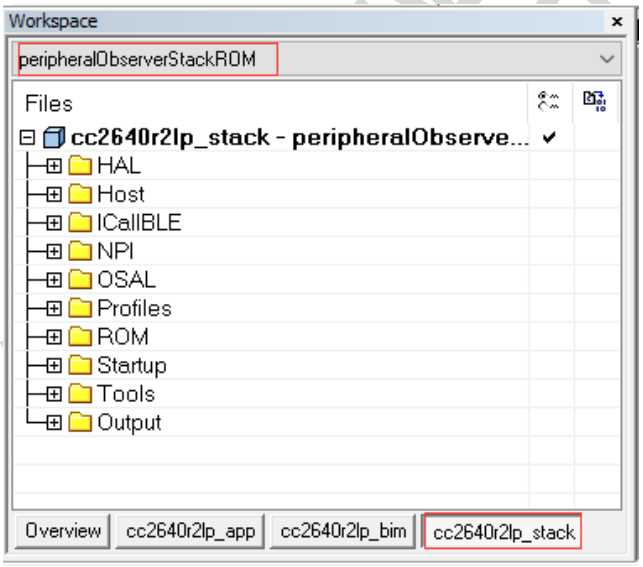
从机+观察者角色比从机角色增加了观察者角色，可以扫描广播。基本功能请参见 3.2 从机示例，以下仅介绍观察者功能。



3.3.1.2.工程配置

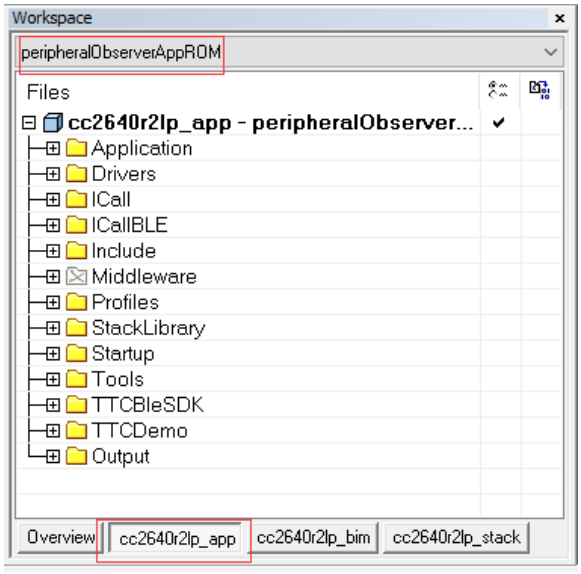
1. 协议栈部分(cc2640r2lp_stack)

选中“cc2640r2lp_stack”选项卡，然后在配置下拉框中选择“peripheralOberverStackROM”，重新全编译并下载程序,如下图:



2. 应用程序部分(cc2640r2lp_app)

选中“cc2640r2lp_app”选项卡，然后在配置下拉框中选择“peripheralOberverAppROM”，重新全编译并下载程序,如下图:



3.3.1.3.操作步骤

数据收发步骤请参见 3.2 从机示例，以下介绍扫描广播的功能：

PC 端软件(TTCDemo)操作,选择对应的串口号，设置波特率 115200。选择从机+扫描角色，点击“扫描从机”：设备列表显示为灰色，表示不能向扫描到的设备发起链接请求，接收区域显示扫描到的从机信息。

3.3.2.API 说明

见 TTCBlePeripheralObserver.h 及 TTCBlePeripheralObserverProcess.h.

3.3.3.示例

见 TTCBlePeripheralObserverTask.c，以下为从机+观察者角色处理串口接收到的指令部分代码：

```
/*********************************************************************************
【函    数】  TTCDriverUartProcess(TTCMsg_t * TTCMsg)
【概    述】  从机+观察者角色处理串口接收到的指令
【入口参数】  TTCMsg : 处理串口接收到的数据
*********************************************************************************/
static void TTCDriverUartProcess(TTCMsg_t * TTCMsg) {
    u16 len  = TTCMsg->hdr.state;
    u8 buf[TTCDEMO_MAX_UART_RXSIZE];
    memset(buf,NULL,TTCDEMO_MAX_UART_RXSIZE);
    memcpy(buf, TTCMsg->pValue, TTCMsg->hdr.state);

    if(!memcmp(buf, CMD_Scan, SLAVE_SCAN_AT_LEN)){                          //开始扫描
        u8 endChar0 = *(buf + len - 2);
        u8 endChar1 = *(buf + len - 1);
        if( endChar0 == '\r' && endChar1 == '\n'){
            if(TTCBlePeripheralObserverProcStartDiscovery()){
                TTCDebugLogPrint("扫描中...\r\n", strlen("扫描中...\r\n"));
            }else{
                TTCDebugLogPrint("开启扫描失败！\r\n", strlen("开启扫描失败！\r\n"));
            }
        }
    }else if(!memcmp(buf,CMD_Disconnect,DISCONNECT_AT_LEN)){
    //断开连接指令 CMD_Disconnect
        TTCBlePeripheralObserverGetParameter(GAPROLE_CONNHANDLE, &connHandle);
        if(TTCBlePeripheralObserverGAPRoleTerminateConnection(connHandle)){
            TTCDebugLogPrint("断开......\r\n", strlen("断开......\r\n"));
        }else{
            TTCDebugLogPrint("断开失败\r\n", strlen("断开失败\r\n"));
        }
    }else if(!memcmp(buf, CMD_SendData, BLE_SEND_DATA_AT_LEN)){
    //发送数据指令 CMD_SendData
        u8 endChar0 = *(buf + len - 2);
        u8 endChar1 = *(buf + len - 1);
        if( endChar0 == '\r' && endChar1 == '\n'){
            u8 * send = NULL;
            u8 sendLen = len - BLE_SEND_DATA_AT_LEN - 2;
            send  = buf;
            send += BLE_SEND_DATA_AT_LEN;
#ifndef TTCBLE_WECHAT
```

```
            TTCBleProfileSetParameter(TTCBLE_PROFILE_CHAR2,
                                      sendLen,
                                      (u8*)send);
#else
            TTCDemoWechatSendData((const u8*)send,sendLen);
#endif
        }
    }else if(!memcmp(buf, CMD_advState, BLE_ADV_STATE_AT_LEN)){
    //发开关广播指令 CMD_advState 开启“1”广播/“0”关闭广播
        u8 endChar0 = *(buf + len - 2);
        u8 endChar1 = *(buf + len - 1);
        u8 advSwitch = 0;
        if( endChar0 == '\r' && endChar1 == '\n'){
            if(*(buf+BLE_ADV_STATE_AT_LEN) == '0') {
            advSwitch = 0;
            TTCBlePeripheralObserverSetParameter(GAPROLE_ADVERT_ENABLED,
                                                 sizeof(u8),
                                                 &advSwitch);
            }else if(*(buf+BLE_ADV_STATE_AT_LEN) == '1'){
            advSwitch = 1;
            TTCBlePeripheralObserverSetParameter(GAPROLE_ADVERT_ENABLED,
                                                 sizeof(u8),
                                                 &advSwitch);
            }
        }
    }else{
        TTCDemoProcessCmd(buf,len);
    }
}
```

3.4. 主机示例

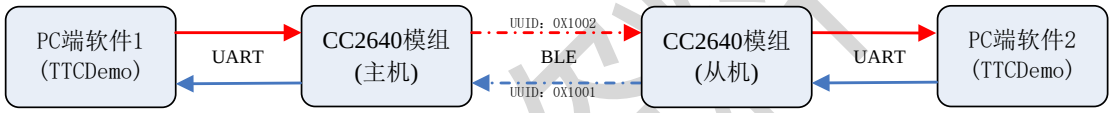
3.4.1. 示例功能说明

3.4.1.1.功能概述

主机扫描从机，建立链接后，可以实现数据互传，主机及从机均可主动断开连接。

1.CC2640 主从模块建立连接后,PC 端软件 1 (TTCDemo)通过 UART(115200bps)发送数据到 CC2640 主模块，CC2640 主模块接收到数据后，通过蓝牙下行通道(UUID:0X1002)将数据转发给 CC2640 从模块，CC2640 从模块接收到数据后，通过 UART(115200bps)转发到 PC 端软件 2 (TTCDemo)数据流方向,如下图蓝线所示。

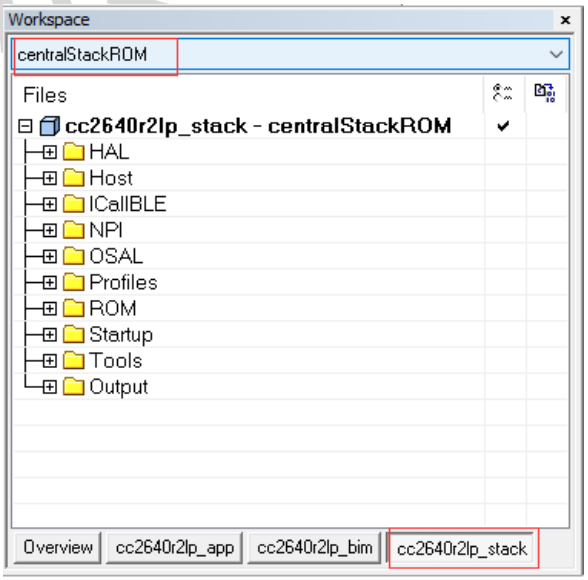
2.CC2640 主从模块建立连接后,PC 端软件 2 (TTCDemo)通过 UART(115200bps)发送数据到 CC2640 从模块，CC2640 从模块接收到数据后，通过蓝牙上行通道(UUID:0X1001)将数据转发给 CC2640 主模块，CC2640 主模块接收到数据后，通过 UART(115200bps)转发到 PC 端软件 1 (TTCDemo)，数据流方向，如下图红线所示。当然，PC 端软件(TTCDemo)1、2 只是演示作用，也可以用 mcu 代替，做到产品中去。



3.4.1.2.工程配置

1.协议栈部分(cc2640r2lp_stack)

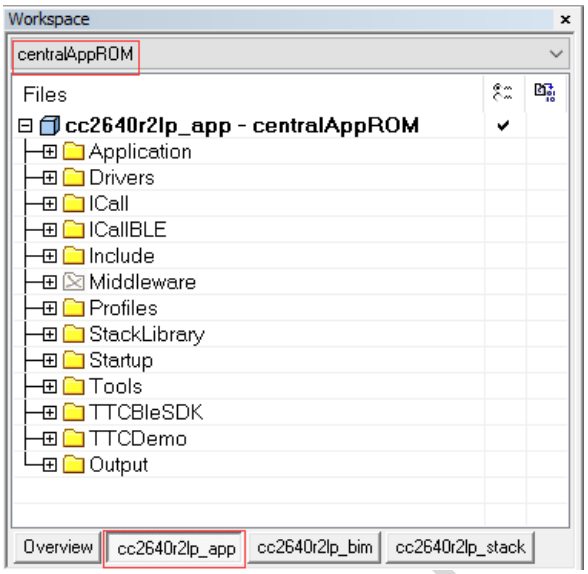
选中“cc2640r2lp_stack”选项卡，然后在配置下拉框中选择“centralStackROM”，重新全编译并下载程序,如下图:



2.应用程序部分(cc2640r2lp_app)

选中“cc2640r2lp_app”选项卡，然后在配置下拉框中选择

“centralAppROM”，重新全编译并下载程序,如下图:



3.4.1.3.操作步骤

1.PC 端软件(TTCDemo)操作

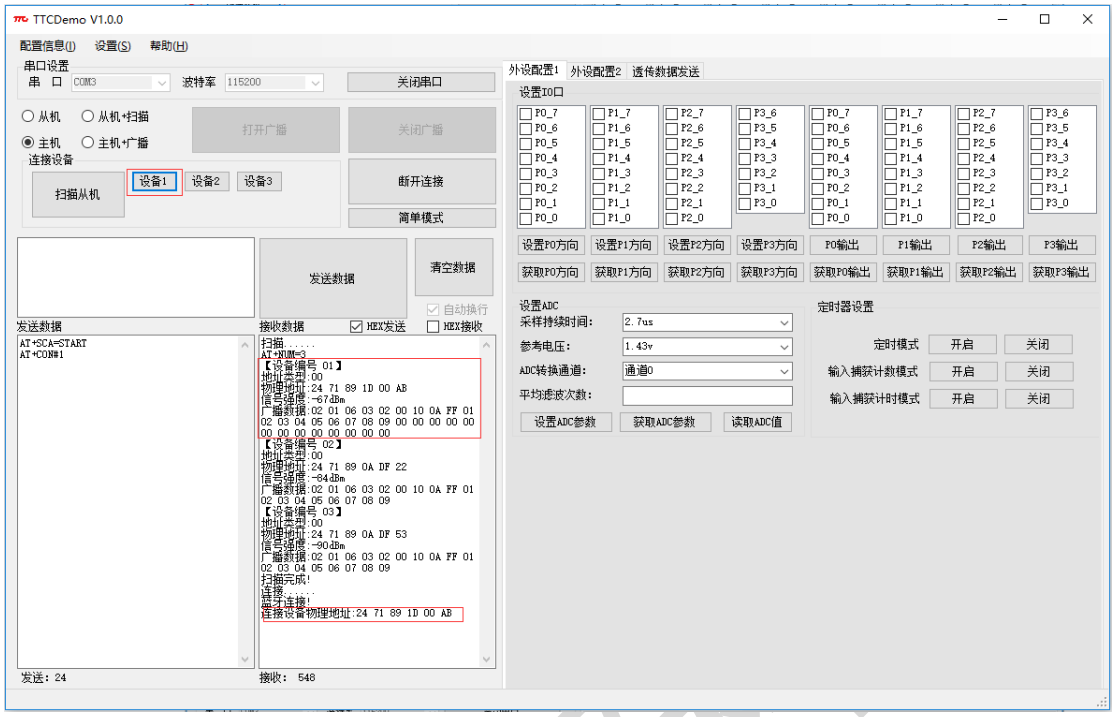
CC2640 模组通过 USB 转接板连接至电脑 USB 接口，开启 PC 端软件(TTCDemo)并选择对应的串口号，设置波特率为 115200，点击“打开串口”；选择蓝牙角色为“主机”，即可点击相应按钮实现对应的 AT 指令发送，无需自行编辑 AT 指令。

(1) 选择文本方式接收，“扫描从机”

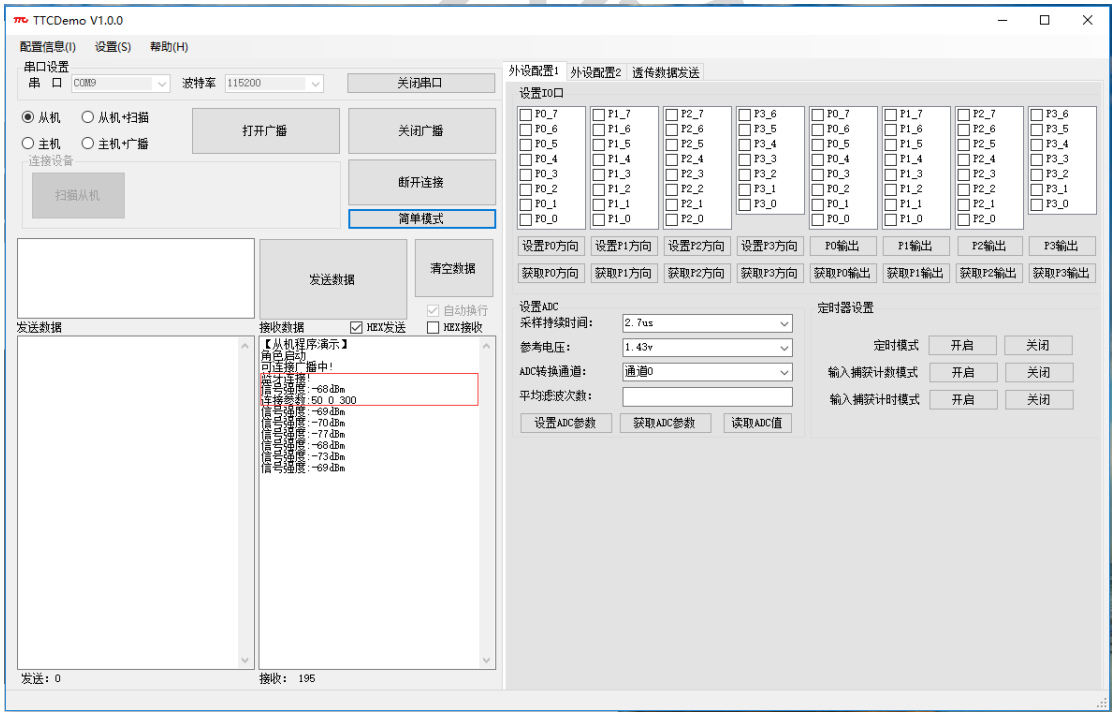
(2)“连接设备”：扫描完毕，接收窗口显示主机扫描到的从机列表，例如需要连接“设备编号 01”的设备，则点击“连接设备”中“设备 1”。

➢ 与主机连接的 PC 端软件(TTCDemo)1：主机连接成功之后，显示对应的从

机 Mac 地址 0X2471891000AB，如下图：



➢ 与从机连接的 PC 端软件(TTCDemo)：被连接成功之后，显示连接参数，如下图：

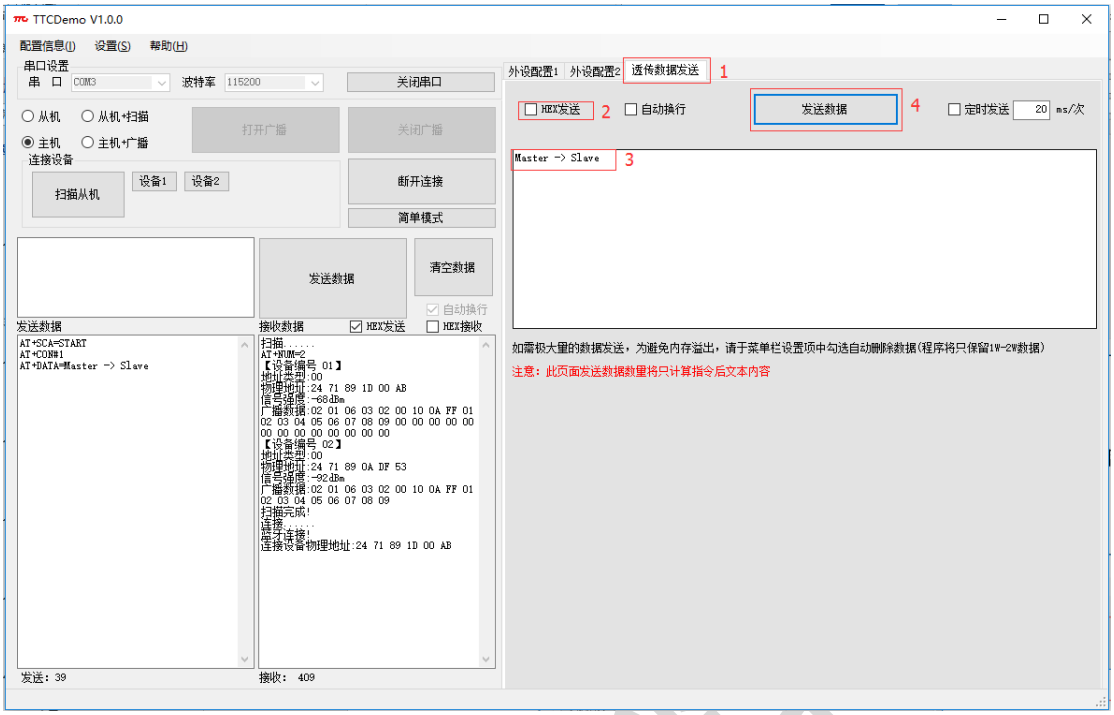


(2)数据收发

➢ 主机数据发送，与主机连接的 PC 端软件(TTCDemo)1：

如下图，“Master -> Slave”为用户键盘输入的 16Byte 数据，PC 端软件(TTCDemo)自动按照“AT+DATA=Master -> Slave\r\n”格式发送。

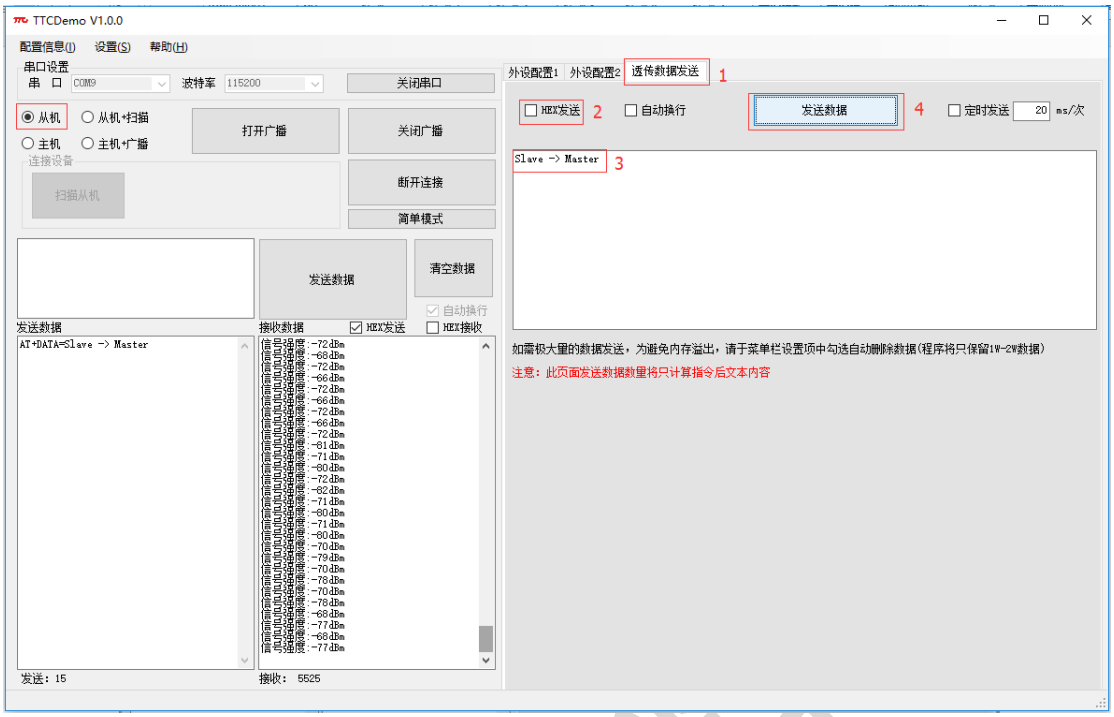
主机发送数据：



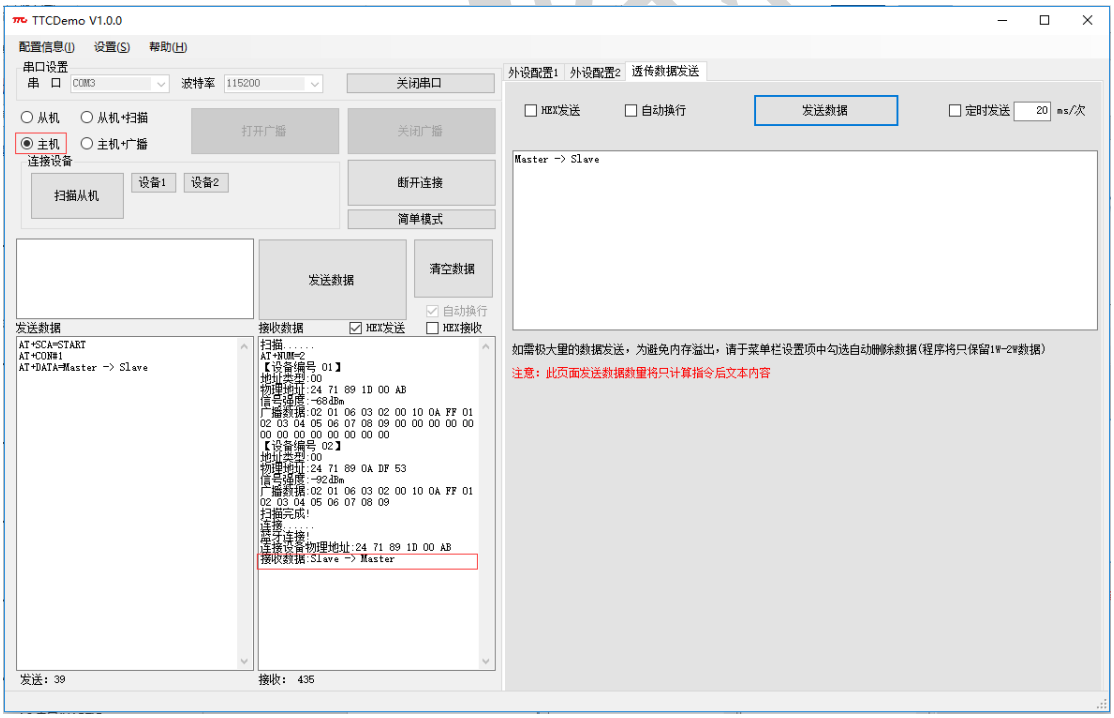
从机接收数据“Central -> Slave”，与从机连接的 PC 端软件(TTCDemo)2：

➢ 主机数据接收

从机发送数据“Slave -> Master”，与从机连接的 PC 端软件(TTCDemo)2：



主机接收数据“Slave -> Master”，与主机连接的 PC 端软件(TTCDemo)1：



3.4.2. API 说明

见 TTCBleCentralProcess.h 及 TTCBleCentral.h

3.4.3. 示例

见 TTCBleCentralTask.c，以下为主机处理串口接收到的指令部分代码：

```
/*****************************************************************************
【函    数】  TTCDriverUartProcess(TTCMsg_t * TTCMsg)
【概    述】  主机处理 UART 接收到的指令
【入口参数】  TTCMsg : 串口接收到的数据
*****************************************************************************/
static void TTCDriverUartProcess(TTCMsg_t * TTCMsg) {
    u16 len  = TTCMsg->hdr.state;
    u8 buf[TTCDEMO_MAX_UART_RXSIZE];
    memset(buf,NULL,TTCDEMO_MAX_UART_RXSIZE);
    memcpy(buf, TTCMsg->pValue, TTCMsg->hdr.state);
    if(!memcmp(buf, CMD_Scan, SLAVE_SCAN_AT_LEN)){                              //开始扫描
        if(TTCBleCentralProcStartDiscovery()){
            TTCDebugLogPrint("扫描......\r\n", strlen("扫描......\r\n"));

        }else{
            TTCDebugLogPrint("扫描失败\r\n", strlen("扫描失败\r\n"));
        }

    }else if(!memcmp(buf, CMD_Connect, CONNECT_AT_LEN)){                        //连接
        if( (*(buf + CONNECT_AT_LEN + 1) == '\r') &&
            (*(buf + CONNECT_AT_LEN + 2) == '\n') &&
            ( *(buf + CONNECT_AT_LEN) >= 0x30) ){
              //连接索引范围 1-8，不能超过这个范围
           s8 index = *(buf + CONNECT_AT_LEN) - 0x30;
           if(index >= 1 && index <= 8){
                if(TTCBleCentralProcEstablishLink(index-1)) {                   //
                    TTCDebugLogPrint("连接......\r\n", strlen("连接......\r\n"));

                }else{
                    TTCDebugLogPrint("连接失败\r\n", strlen("连接失败\r\n"));
                }
           }
        }
    }else if(!memcmp(buf,CMD_Disconnect,DISCONNECT_AT_LEN)){                    //断开连接
        if(TTCBleCentralProcTerminateLink(connHandle) == SUCCESS ){
            TTCDebugLogPrint("断开......\r\n", strlen("断开......\r\n"));
        }else{
            TTCDebugLogPrint("断开失败\r\n", strlen("断开失败\r\n"));
        }
    }else if(!memcmp(buf, CMD_SendData, BLE_SEND_DATA_AT_LEN)){
        u8 endChar0 = *(buf + len - 2);
        u8 endChar1 = *(buf + len - 1);
```

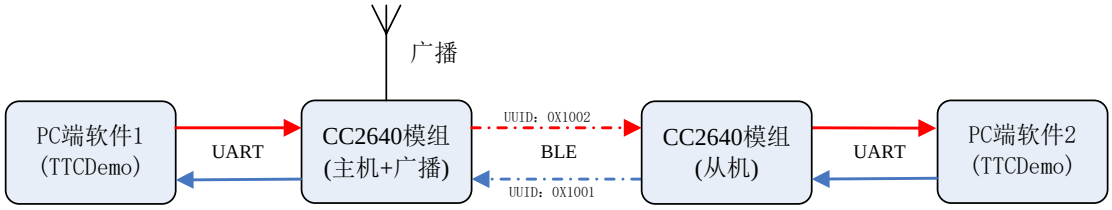
```
        if( endChar0 == '\r' && endChar1 == '\n'){
            u8 * send = NULL;
            u8 sendLen = len - BLE_SEND_DATA_AT_LEN - 2;
            send  = buf;
            send += BLE_SEND_DATA_AT_LEN;                                    //指向发送数据
            TTCBleCentralProcWriteData(connHandle,
                                       (u8*)send,
                                       sendLen);                             //发送蓝牙数据
        }
    }else{
        TTCDemoProcessCmd(buf,len);
    }
}
```

3.5. 主机+广播示例

3.5.1. 示例功能说明

3.5.1.1.功能概述

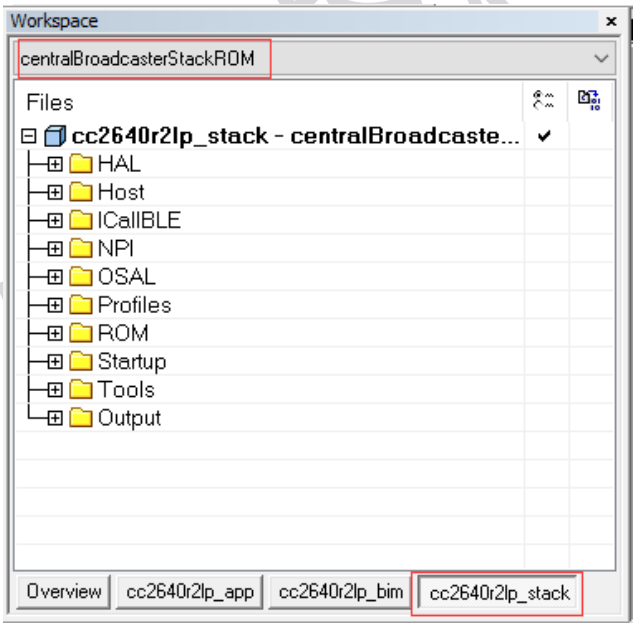
主机+广播角色比主机角色增加了广播角色，可以发出广播(不可链接广播)。基本功能请参见 3.3 主机示例，以下仅介绍广播功能。



3.5.1.2.工程配置

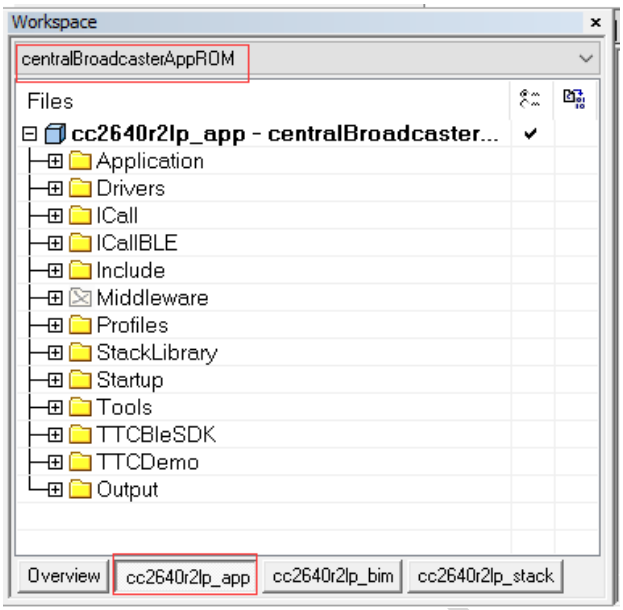
1. 协议栈部分(cc2640r21p_stack)

选中“cc2640r21p_stack”选项卡，然后在配置下拉框中选择“centralBroadcasterStackROM”，重新全编译并下载程序,如下图:



2. 应用程序部分(cc2640r21p_app)

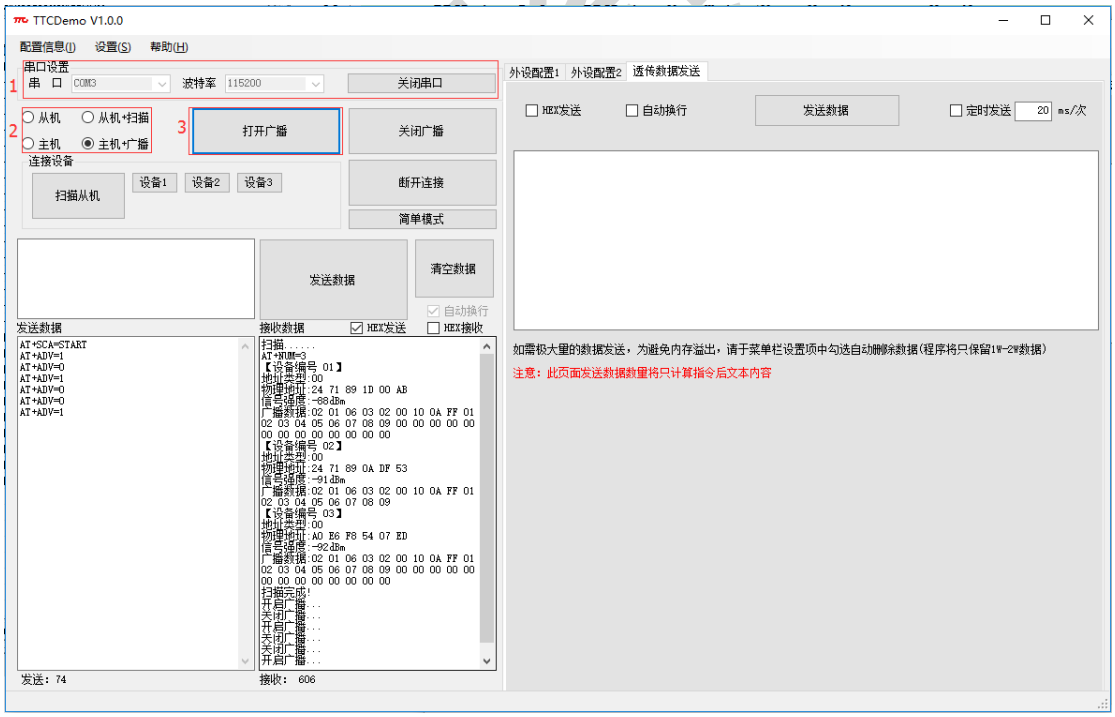
选中“cc2640r21p_app”选项卡，然后在配置下拉框中选择“centralBroadcasterAppROM”，重新全编译并下载程序,如下图:



3.5.1.3.操作步骤

数据收发步骤请参见 3.4 主机示例，以下介绍扫描广播的功能：

PC 端软件(TTCDemo)操作,选择对应的串口号，设置波特率 115200，选择蓝牙角色为“主机+广播”，点击“打开广播”、“广播广播”即可实现广播开启、关闭，接收区域显示操作是否成功：



3.5.2.API 说明

见 TTCBleCentralBroadcaster.h 及 TTCBleCentralBroadcasterProcess.h.

3.5.3.示例

见 TTCBleCentralBroadcasterTask.c，以下为主机+观察者处理串口接收到的指令部分代码：

```
/*******************************************************************************
【函    数】  TTCDriverUartProcess(TTCMsg_t * TTCMsg)
【概    述】  主机+观察者处理串口接收到的指令
【入口参数】  TTCMsg : 处理串口接收到的数据
*******************************************************************************/
    u16 len  = TTCMsg->hdr.state;
    u8 buf[TTCDEMO_MAX_UART_RXSIZE];
    memset(buf,NULL,TTCDEMO_MAX_UART_RXSIZE);
    memcpy(buf, TTCMsg->pValue, TTCMsg->hdr.state);
    if(!memcmp(buf, CMD_Scan, SLAVE_SCAN_AT_LEN)){                              //开始扫描
        if(TTCBleCentralBroadcasterProcStartDiscovery()){
            TTCDebugLogPrint("扫描......\r\n", strlen("扫描......\r\n"));
        }else{
            TTCDebugLogPrint("扫描失败\r\n", strlen("扫描失败\r\n"));
        }
    }else if(!memcmp(buf, CMD_Connect, CONNECT_AT_LEN)){                        //连接
        if( (*(buf + CONNECT_AT_LEN + 1) == '\r') &&
            (*(buf + CONNECT_AT_LEN + 2) == '\n') &&
            ( *(buf + CONNECT_AT_LEN) >= 0x30) ){
            //连接索引范围 1-8，不能超过这个范围
           s8 index = *(buf + CONNECT_AT_LEN) - 0x30;
           if(index >= 1 && index <= 8){
                if(TTCBleCentralBroadcasterProcEstablishLink(index-1)) {
                    TTCDebugLogPrint("连接......\r\n", strlen("连接......\r\n"));

                }else{
                    TTCDebugLogPrint("连接失败\r\n", strlen("连接失败\r\n"));   }
           }
        }
    }else if(!memcmp(buf,CMD_Disconnect,DISCONNECT_AT_LEN)){                    //断开连接
        if(TTCBleCentralBroadcasterProcTerminateLink(connHandle) == SUCCESS){
            TTCDebugLogPrint("断开......\r\n", strlen("断开......\r\n"));
        }else{
            TTCDebugLogPrint("断开失败\r\n", strlen("断开失败\r\n"));
        }
    }else if(!memcmp(buf, CMD_SendData, BLE_SEND_DATA_AT_LEN)){
        u8 endChar0 = *(buf + len - 2);
        u8 endChar1 = *(buf + len - 1);
        if( endChar0 == '\r' && endChar1 == '\n'){
```

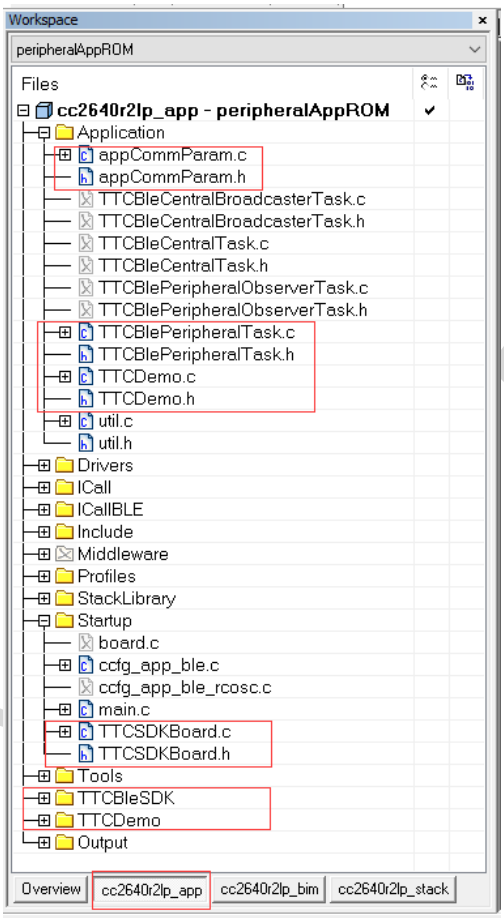
```
            u8 * send = NULL;
            u8 sendLen = len - BLE_SEND_DATA_AT_LEN - 2;
            send  = buf;
            send += BLE_SEND_DATA_AT_LEN;                                    //指向发送数据
            TTCBleCentralBroadcasterProcWriteData(connHandle, (u8*)send, sendLen);
            //发送蓝牙数据
        }
    }else if(!memcmp(buf, CMD_advState, BLE_ADV_STATE_AT_LEN)){
    //发开关广播指令 CMD_advState 开启“1”广播/“0”关闭广播
        u8 endChar0 = *(buf + len - 2);
        u8 endChar1 = *(buf + len - 1);
        u8 advSwitch = 0, advRet = 0;
        if( endChar0 == '\r' && endChar1 == '\n'){
            if(*(buf+BLE_ADV_STATE_AT_LEN) == '0') {
              advSwitch = 0;
              advRet = TTCBleCentralBroadcasterSetParameter(
                                GAP_CENTRALBROADCASTER_ADVERT_ENABLED,
                                                        sizeof(u8),
                                                        &advSwitch);
              if(!advRet) {
                TTCDebugLogPrint("关闭广播...\r\n", strlen("关闭广播...\r\n"));
              }else {
                TTCDebugLogPrint("关闭广播失败...\r\n", strlen("关闭广播失败...\r\n"));
              }
            }else if(*(buf+BLE_ADV_STATE_AT_LEN) == '1'){
              advSwitch = 1;
              advRet = TTCBleCentralBroadcasterSetParameter(
                                GAP_CENTRALBROADCASTER_ADVERT_ENABLED,
                                                        sizeof(u8),
                                                        &advSwitch);
              if(!advRet) {
                TTCDebugLogPrint("开启广播...\r\n", strlen("开启广播...\r\n"));
              }else {
                TTCDebugLogPrint("开启广播失败...\r\n", strlen("开启广播失败...\r\n"));
              }
            }
        }
    }else{
        TTCDemoProcessCmd(buf,len);
    }
}
```

4. 驱动 Demo 说明

TTC SDK V3.x 驱动 demo 通过 AT 指令的方式，实现不同功能的控制。在调试其他驱动之前，需要先使能串口(UART)demo，以处理 AT 指令。

4.1. 整体结构介绍

以从机工程为例，简要说明应用程序中驱动 demo 整体结构，另外三个蓝牙角色工程驱动使用方式相同。如不使用 demo 示例，可参照 demo 的处理流程进行项目开发。从机工程总体结构如下：



1.<TTCBlePeripheralTask.c>文件

此文件包含驱动 demo 的初始化，及相关事件、消息的处理。

(1)整体结构 TTCBlePeripheralTaskFxn()

```
static void TTCBlePeripheralTaskFxn(UArg a0, UArg a1){
    //1.初始化,包含驱动 demo 初始化
    TTCBlePeripheralTaskInit();
    ……
    //2.消息处理：包含串口驱动的消息处理
    TTCBlePeripheralTaskProcessAppMsg(pMsg);
    ……
    //3.驱动事件处理
    TTCDriverSpiEvent();
    ……
```

```
}
```

(2)初始化函数 TTCBlePeripheralTaskInit()

TTCDebugInit()函数包含串口(UART)驱动，及其他驱动的初始化。

```
static void TTCBlePeripheralTaskInit(void){
    ……
    TTCDebugInit(syncEvent,
                 appMsgQueue,
                 &TTCBlePeripheralTaskCls);
    ……
}
```

(3)消息处理函数 TTCBlePeripheralTaskProcessAppMsg()

TTCDriverUartProcess()包含蓝牙角色 demo、驱动 demo 的 AT 指令处理。

```
static void TTCBlePeripheralTaskProcessAppMsg(TTCMsg_t *pMsg){
    ……
    //1.串口消息处理：解析串口接收到的 AT 指令(包含蓝牙角色 demo 及驱动 demo 的指令)
    case TTCSDK_MSG_DRIVER_UART_EVENT:{
        TTCDriverUartProcess(pMsg);
    }break;
    ……
}
```

2.<appCommParam.c>文件

驱动 demo 均使用了串口接收 AT 指令，此文件为驱动 demo 的公共文件：包含串口初始化(包含读写回调)及调试信息打印函数。

3.<TTCDemo.c>文件

包含所有驱动 demo 的初始化、驱动 demo AT 指令表、驱动 demo AT 指令解析并处理。

```
CMDState_t TTCDemoProcessCmd(u8 * buf,u16 size){
    ……
    //1.查找表，解析 AT 指令
    for( u8 i = 0 ; i < TTCDEMO_CMD_TABLE_SIZE ; i++ ){
      if(!memcmp(buf,TTCDemoCmdTable[i].cmd,TTCDemoCmdTable[i].cmdSize)){
    ……

    //2.根据 AT 指令，实现不同功能
    status = TTCDemoCmdTable[i].cmdApi(i,operation,p,&dataSize,TTCDemoCmdTable[i].cmdSize);
    ……
}
```

4.<TTCDemo>组

包含库文件(蓝牙及驱动)及相关头文件，头文件中有相关 API 说明。

5.<TTCBleSDK>组

包含所有的驱动相关 demo。

6. TTCSDKBoard.h

根据实际 CC2640 的封装，打开对应头文件，以修改 GPIO 的定义：

```
#if (defined(CC2650DK_5XD) || defined(CC2650DK_7ID)) && defined(CC26XX_R2)
    #error "注意：CC2640 R2 SDK 只有 'CC2650DK_6ID', 'CC2650DK_4ID'"
#elif (defined(CC2650DK_4ID) || defined(CC2650DK_6ID)) && defined(CC26XX_R1)
    #error "注意：CC2640 R1 SDK 只有 'CC2650DK_7ID', 'CC2650DK_5XD'"
#endif

#if defined(CC2650DK_7ID)
    #include "CC2640_7ID/TTCSDKBoard.h"
#elif defined(CC2650DK_5XD)
    #include "CC2640_5XD/TTCSDKBoard.h"
#elif defined(CC2650DK_6ID)
    #include "CC2640_6ID/TTCSDKBoard.h"
#elif defined(CC2650DK_4ID)
    #include "CC2640_4ID/TTCSDKBoard.h"
#else
    #error "注意…'"
#endif
```

4.2. 串口(UART) Demo

CC2640 的 UART 具有如下特点：

(1) 具备可编程的波特率发生器，最高速率高达 3 Mbps。

(2) 具备独立的 32x8 发送（TX）和 32x12 接收（RX）FIFO 缓冲器，可以减少 CPU 的中断处理动作。

(3) 具备开始、停止和奇偶校验的标准异步通信位。

(4) 支持 CTS 和 RTS 功能。

(5) 使用 uDMA 传输数据。

(6) 具备可编程的硬件流控制。

4.2.1. 引脚说明

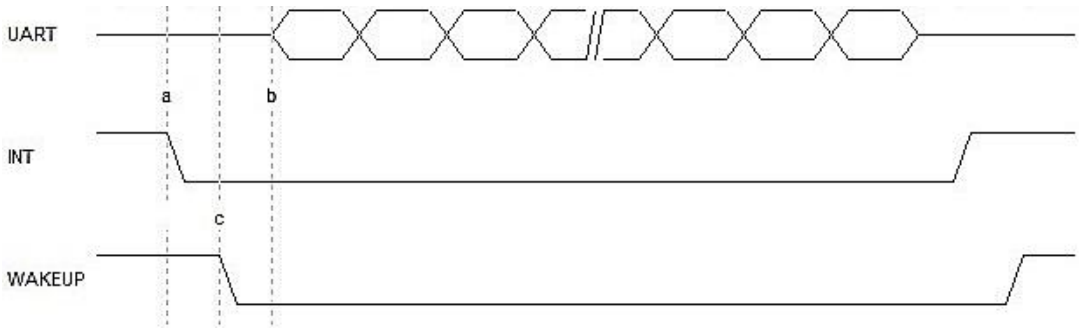
- TX：模组数据发送。
- RX：模组数据接收。
- WAKEUP：唤醒引脚（BLE 模组的输入引脚）
  (1) 当外部 MCU 需要向 BLE 模组发送数据时，需先拉低 WAKEUP 引脚，以便先唤醒 BLE 模组。
  (2) 当 BLE 模组向外部 MCU 发送数据时，BLE 模组将 INT 拉低以唤醒外部 MCU；MCU 在检测到 INT 低电平后，需将 WAKEUP 引脚拉低，BLE 模组才会启动串口数据发送。
- INT：中断引脚（BLE 模组的输出引脚）
  (1) 当外部 MCU 需要向 BLE 模组发送数据时，INT 引脚仅作为状态指示，可忽略：BLE 模组被唤醒之后，会将 INT 引脚拉低一段时间，告知外部 MCU 可以开始发送数据。
  (2) 当 BLE 模组需向外部发送数据时，会先自动拉低 INT 引脚（可用于唤醒外部 MCU）。

4.2.2. 时序图

透传模组与外部 MCU 通信时，需严格按照通信时序操作。当然，在测试时，可将透传模组 WAKEUP 引脚拉低，使模组一直处于唤醒状态，便于测试。

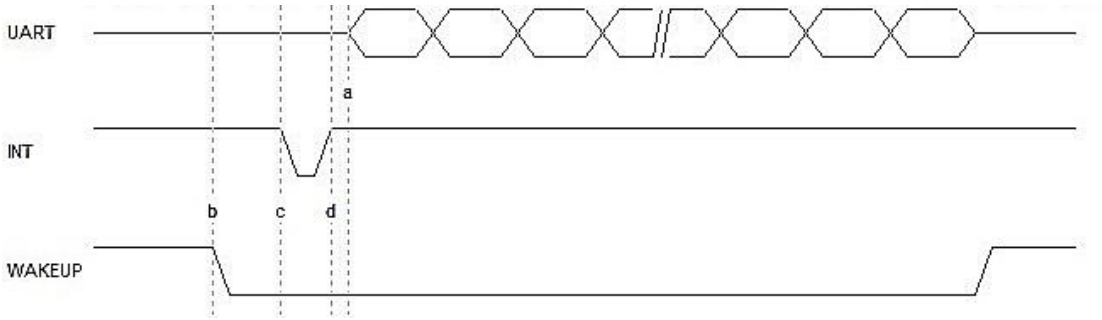
(1) 外部 MCU 中断方式读数据（BLE 模组发送数据给外部 MCU）

注意事项：△a-c >=600us，△a-b > △a-c，当 WAKEUP=0 时，串口数据才会发送到主控制器端。



(2) 外部 MCU 写数据（外部 MCU 发送数据给 BLE 模组）

注意事项：△b-c >=600us，△a-b > △b-c，△c-d ≈ 400ns



(3)中断时序（BLE 模组发送数据给外部 MCU）

当 BLE 模块接收到数据，立刻拉低“INT”信号，主控制器未读取数据情况下，BLE 模块会自动间隔拉高"INT"信号，一直到主控制器被 BLE 模块唤醒，并拉低“WAKEUP”信号为止。建议主控制器端，采取下降沿方式获取数据。

4.2.3. demo 设定及功能

定义宏 TTCDRIVER_UART、TTC_DEBUG：使用串口驱动相关的函数，需要定义宏 TTCDRIVER_UART；使用串口 demo 及其他驱动 demo，需要定义宏 TTC_DEBUG。

参照章节 2 使用相关工具。

功能：CC2640 通过串口(UART)接收 AT 指令，并输出相关调试信息。

4.2.4. 初始化

配置相关参数，并注册串口读写回调函数(callback function)

```
void TTCDebugInit(void){
    ......
    //1.串口相关参数设置
    const TTCDriverUartParams_t uartParam = {
    ......
    TTCDriverUartEnLevel_t      wakeupPinEnLevel;          //唤醒引脚使能电平
    TTCDriverUartEnLevel_t      intPinEnLevel;             //中断输出使能电平
    u32                         transferDelay;             //唤醒 CC2640 后延迟发送串口数据
    };
    //2.串口驱动初始化
    TTCDriverUartInit(pTTCSDKCBHandle,                    // lib function
```

```
                    &TTCDebugWriteCB,                   //write callback function
                    &TTCDebugReadCB,                    //read callback function
                    &uartHandle,
                    uartParam);
    ......
  }
```

说明：

(1)若 CC2640 处于睡眠状态，需要将唤醒引脚 WKP 拉低 80us 以上，再发送 AT 指令；

(2)初始化参数中 wakeupPinEnLevel，可设置 CC2640 的唤醒电平；

(3)初始化参数中 intPinEnLevel，可设置中断输出使能电平，用于唤醒外部 MCU 接收 UART 数据。

(4)初始化参数中 transferDelay，可设置中断输出使能电平的时间延迟，默认值为 80us。若延迟设置为 1 时，则电平宽度为 80us + 1*100us = 180us.

4.2.5.回调函数

1.读回调函数(read callback function)

串口接收到数据（如 AT 指令）后，会进入读回调函数函数，可进行数据处理：

```
static void TTCDebugReadCB(void * handle, u8 * buffer, u16 len){
    ......
    //1.接收到串口数据，发消息
      Util_enqueueMsg(pTTCSDKDebugQueueHandle,
                      pTTCSDKDebugaEvent,
                      (u8*)pMsg);
     ......
}
```

2.写回调函数(write callback function)

串口发送完数据后，会进入写回调函数，可做相应操作，如状态指示等。

```
static void TTCDebugWriteCB (void * handle, u8 * buffer, u16 len){
    ......
}
```

4.2.6.处理流程

接收到串口数据后，进入读回调函数(发消息)，再进行消息处理。

1.串口收到数据，发消息

```
static void TTCDebugReadCB(void * handle, u8 * buffer, u16 len){
    ......
    //1.接收到串口数据，发消息
      Util_enqueueMsg(pTTCSDKDebugQueueHandle,
                      pTTCSDKDebugaEvent,
```

```
                        (u8*)pMsg);
    ......
}
```

2.处理串口消息处理

```
static void TTCBlePeripheralTaskProcessAppMsg(TTCMsg_t *pMsg){
    ......
     //1.串口消息处理：解析串口接收到的 AT 指令(包含蓝牙角色 demo 及驱动 demo 的指令)
    case TTCSDK_MSG_DRIVER_UART_EVENT:{
        TTCDriverUartProcess(pMsg);
    }break;
    ......
}
```

2.解析 AT 指令，并实现相关功能

```
CMDState_t TTCDemoProcessCmd(u8 * buf,u16 size){
   ......
   //1.查找表，解析 AT 指令
   for( u8 i = 0 ; i < TTCDEMO_CMD_TABLE_SIZE ; i++ ){
     if(!memcmp(buf,TTCDemoCmdTable[i].cmd,TTCDemoCmdTable[i].cmdSize)){
   ......

   //2.根据 AT 指令，实现不同功能
   status = TTCDemoCmdTable[i].cmdApi(i,operation,p,&dataSize,TTCDemoCmdTable[i].cmdSize);
   ......
}
```

4.2.7.API 说明

串口(UART)相关 API 见 TTCDriverUART.h.

4.3. GPIO Demo

CC2640 拥有丰富的 GPIO 资源，满足各种开发需求。最多可提供 31 个 GPIO 供开发人员使用，支持多种配置比如上下拉、开漏、推挽输出等。每个 GPIO 都可以配置中断功能，中断的方式也可以灵活配置，比如上升沿中断、下降沿中断、上升下降沿都中断等。另外每个 GPIO 都可以任意映射片内的外设资源，比如 PWM 输出口，ADC 输入口等。

4.3.1. demo 设定及注意事项

(1)demo 设置

定义宏 TTCDRIVER_UART、TTC_DEBUG、TTCDRIVER_GPIO，并关闭其他驱动的宏定义。使用串口驱动相关的函数，需要定义宏 TTCDRIVER_UART；使用串口 demo 及其他驱动 demo，需要定义宏 TTC_DEBUG；调用 GPIO 相关 API,需要定义宏 TTCDRIVER_GPIO.

参照章节 2使用相关工具。

功能：AT 指令控制 IO 口输入/输出。

(2)注意事项

在程序初始化时，可以统一设置 GPIO 的初始配置。

如从机角色初始化时，GPIO 默认设置为输入，并配置为下拉。当需要使用 GPIO 时，可再次根据实际需求进行配置，如设置输入输出，上拉下拉等。

```
<TTCBlePeripheralTask.c>
TTCBleSDKManagerInit(CC2650EM_TYPE, CC2650EM_POWER_SAVING,PIN_PULLDOWN);     //初始化外设管理
```

4.3.2. 初始化

1.在 TTCDemoInit()函数中进行 GPIO 初始化

```
void TTCDemoInit( TTCSdkClass_t *pCB,
                  Event_Handle eventHandle,
                  Queue_Handle queueHandle) {
    ......
    //1.GPIO 驱动 demo 初始化
    #ifdef TTCDRIVER_GPIO
      TTCDemoGPIOInit();
    #endif //TTCDRIVER_GPIO
    ......
}
```

2.根据 IO 口配置，进行初始化。IO 口如何分组，见 TTCSDKBoard.h,或附录 B.

```
//1.IO 口有预先分组，组内 io 口再具体配置
PIN_Config GpioPin0[] = {
    IOO_3 | PIN_INPUT_EN  | PIN_GPIO_OUTPUT_DIS  | PIN_PULLUP   | PIN_HYSTERESIS,
    PIN_TERMINATE
```

```
};
//2.驱动初始化函数
void TTCDemoGPIOInit(void){
    ......
    TTCDriverIOOpen(&GpioPINHandle[0],&GpioPINState[0],(const PIN_Config *)GpioPin0);
    ......
}
```

4.3.3.回调

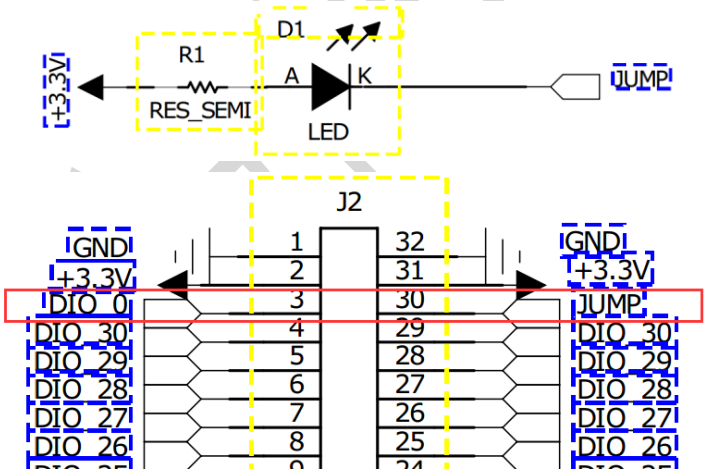
如使用 GPIO 中断，需注册中断回调函数。

4.3.4.处理流程

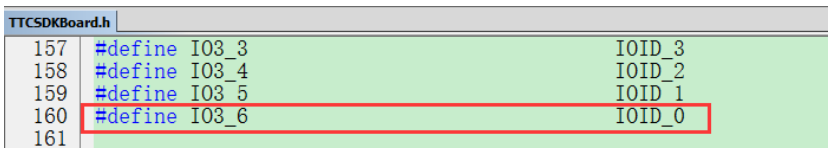
1.接收串口 AT 指令，进入串口写回调，并发消息；
2.线程唤醒，处理消息，解析指令；
3.指令正确，实现相关功能。
备注：串口指令处理，请参见 4.2.2 节串口指令处理流程。

4.3.5.AT 指令操作

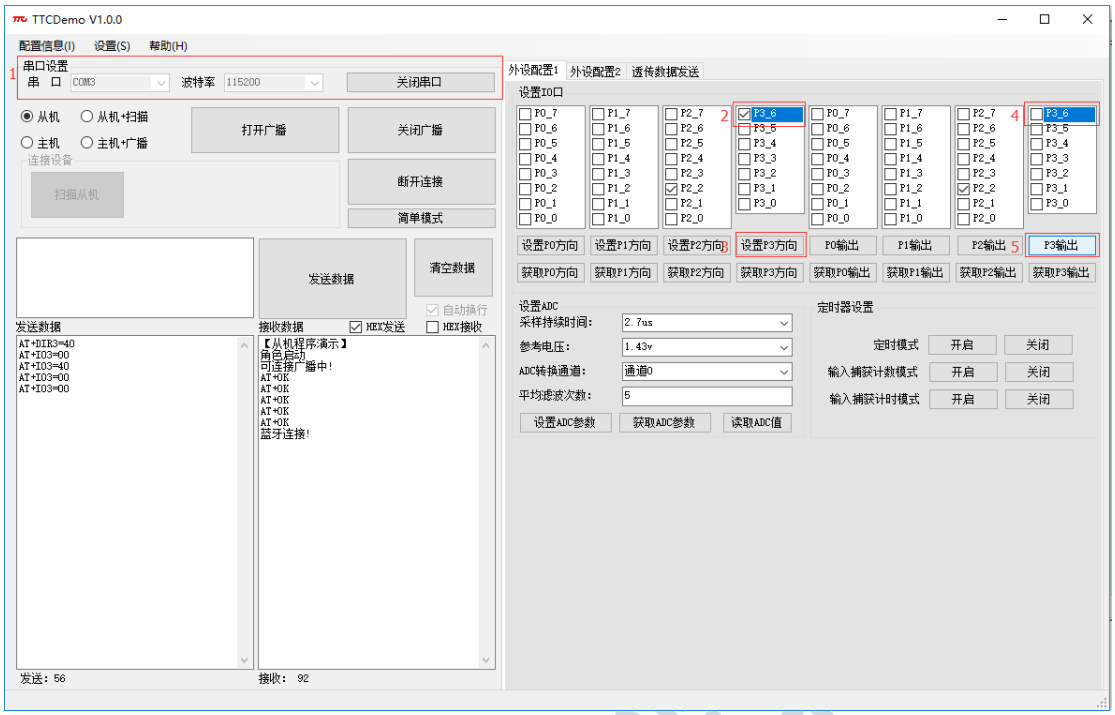
1.如 2.1 节模组连接，并用跳线帽短接 JUMP。



2.例如，要实现 DIO0 控制 LED D1 亮灭.
(1) 查看 TTCSDKBoard.h,或附录 B，IOID_0(DIO0)对应 IO3_6.



(2)IO3_6 对应 P3_6,设置方向时勾选表示输出，输出时勾选表示高电平。



4.3.6. API 说明

GPIO 相关 API 见 TTCBleSDKManager.h.

4.4. 定时器(Timer) Demo

CC2640 拥有 8 个 16 bit 的定时器，每个定时器都可单独配置成不同的模式使用。支持可编程的计数方式，另外它支持同时启动 1 个以上的定时器，适用一些特殊的使用场合。

4.4.1. demo 设置

定义宏 TTCDRIVER_UART、TTC_DEBUG、TTCDRIVER_TIMER，并关闭其他驱动的宏定义。使用串口驱动相关的函数，需要定义宏 TTCDRIVER_UART；使用串口 demo 及其他驱动 demo，需要定义宏 TTC_DEBUG。

参照章节 2使用相关工具。

4.4.2. 定时模式

功能：AT 指令控制定时功能开启、关闭，开启后，IOID_14 每 500us 翻转一次。

4.4.2.1 初始化

1. 定时模式初始化参数设置

```
static TTCDriverTimerCounterParam_t counterParam = {
        .prescaler    = 48+1,        // 定时器时钟源是 48M，1 表示不分频，如 Demo 所示为 48 分频
        .counterMode = CC2650_TIMER_PERIODIC_MODE,
        .countValue   = 500,         //分频后每个周期 1us，500 * 1us = 500us
        .upCountMode = false,        //向下计数
}
```

2. 驱动 demo 初始化函数

```
void TTCDemoInit( TTCSdkClass_t *pCB,
                  Event_Handle eventHandle,
                  Queue_Handle queueHandle) {
    ......
    //1.定时器 demo 驱动初始化
    TTCDemoTimerCounterInit(eventHandle,queueHandle);
    ......
}
```

3. 设置参数并注册回调函数

```
void TTCDemoTimerCounterInit(Event_Handle eventHandle,
                             Queue_Handle queueHandle) {
    ......
    //1.设置参数，驱动初始化
    TTCDriverTimerInit(timerInitParam,
                       &TTCDemoTimerCounterHandle);
    TTCDriverTimerCounterSetParam(&TTCDemoTimerCounterHandle,
                                  counterParam,
                                  false);
    ......
```

```
    //2.注册回调函数 TTCDemoTimerCounterCB
    TTCDriverTimerRegisterIntCallBack(&TTCDemoTimerCounterHandle, &TTCDemoTimerCounterCB );
}
```

4.4.2.2.回调

如 demo 所设置参数，每 500us 进一次回调函数，比如翻转 IO 口输出电平。

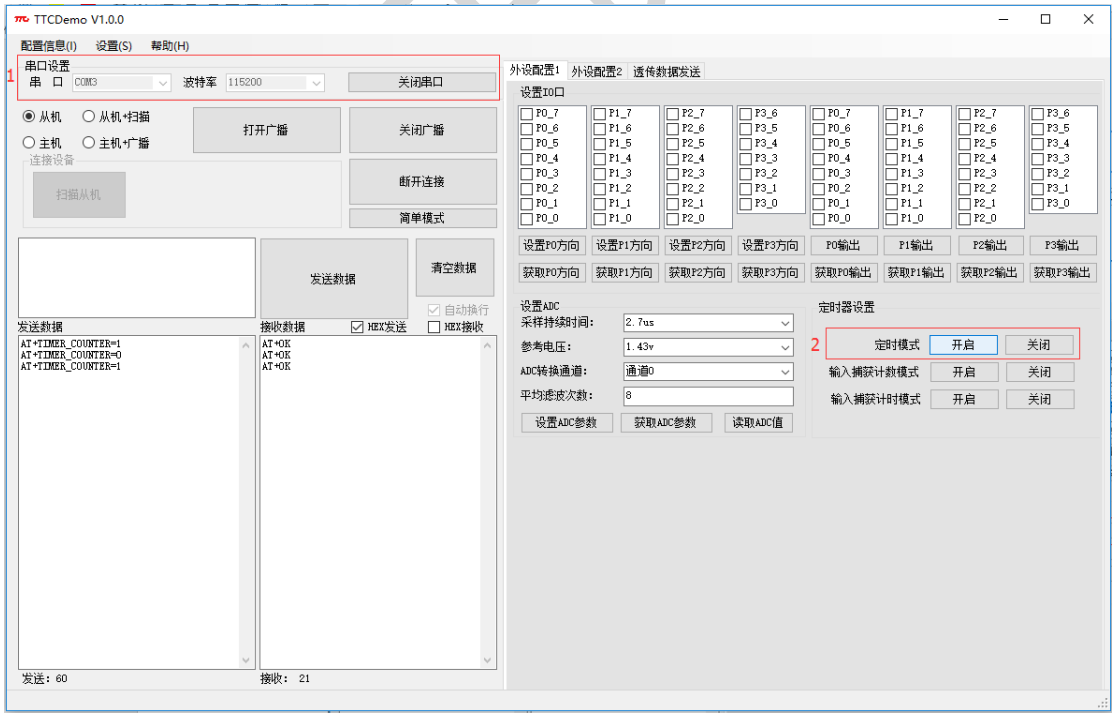
```
static void TTCDemoTimerCounterCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue ){
  ......
}
```

4.4.2.3.处理流程

1.接收串口 AT 指令，进入串口写回调，并发消息；
2.线程唤醒，处理消息，解析指令；
3.指令正确，实现相关功能。
备注：串口指令处理，请参见 4.2.2 节串口指令处理流程。

4.4.2.4.AT 指令操作

定时模式，可通过 AT 指令开启、关闭定时功能；开启时，IOID_14 每 500us 翻转一次。如下图：



4.4.3. PWM 输出模式

功能：AT 指令控制 IOID_10 ~ IOID13 共 4 路 PWM 输出，可设置频率及占空比。

4.4.3.1 初始化

1. PWM 输出模式初始化参数设置

```
static TTCDriverTimerPwmParams_t TTCDemoTimerPwmParam = {              //PWM 输出参数
    TTCDEBUG_PWM_DEFAULT_TIMER_VALUE,
    TTCDEBUG_PWM_DEFAULT_MATCH_VALUE,
    PWM_OUTPUT_NOT_INVERTED,
    false,
};
```

2. 驱动 demo 初始化函数

```
void TTCDemoInit( TTCSdkClass_t *pCB,
                  Event_Handle eventHandle,
                  Queue_Handle queueHandle) {
    ......
    //1. 定时器 demo 驱动初始化
    TTCDemoTimerPWMInit(eventHandle,queueHandle);
    ......
}
```

3. 设置参数

```
void TTCDemoTimerPWMInit(Event_Handle eventHandle,
                         Queue_Handle queueHandle){
    ......
    //1. 设置 4 路 PWM 参数
    for(u8 i=0;i<4;i++){
        timerInitParam.timerName = (CC2650_TimerName)i;
        TTCDriverTimerInit(timerInitParam, &TTCDemoTimerPwmHandle[i]);
        TTCDriverTimerPwmSetParam(&TTCDemoTimerPwmHandle[i],
                                  TTCDemoTimerPwmParam,
                                  Board_PWM0+i, //IOID_0 ~ IOID3 连续 4 个 IO 输出 PWM
                                  false);
    }
    readParam = TTCDemoTimerPwmHandle[0].param;
}
```

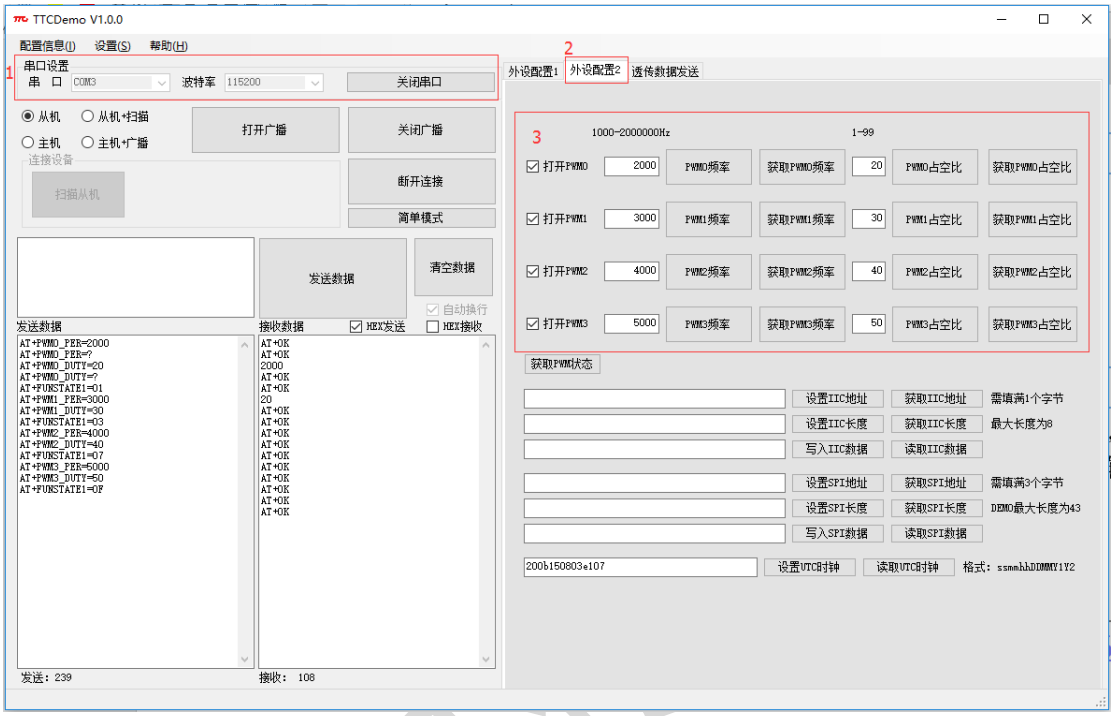
4.4.3.2.处理流程

1. 接收串口 AT 指令，进入串口写回调，并发消息；
2. 线程唤醒，处理消息，解析指令；
3. 指令正确，实现相关功能。

备注：串口指令处理，请参见 4.2.2 节串口指令处理流程。

4.4.3.3.AT 指令操作

如下图，依次开启 PWM0/PWM1/PWM2/PWM3,并设置 PWM0 频率为 2KHz，占空比(duty)为 20%(低电平占比)，PWM1/PWM2/PWM3 依次设置如下可用示波器或者逻辑分析仪观察 IOID10 ~ IOID13 的输出。



4.4.4.输入捕获边沿计数模式

功能：捕获一定数量的边沿，串口打印调试信息。

4.4.4.1 初始化

1. 输入捕获边沿计数模式初始化参数设置

```
static TTCDriverTimerEdgeCountParam_t edgCountParam = {              //捕获计数模式参数
    .countValue         = 2000,
    .upCountMode        = 0,
    .matchValue         = 1000,
    .edgeMode           = TIMER_POSITIVE_EDGE,
}
```

2. 驱动 demo 初始化函数

```
void TTCDemoInit( TTCSdkClass_t *pCB,
                  Event_Handle eventHandle,
                  Queue_Handle queueHandle){
    ......
    //1.定时器 demo 驱动初始化
    TTCDemoTimerEdgeCountInit(eventHandle,queueHandle);
    ......
}
```

3.设置参数并注册回调函数

```
void TTCDemoTimerEdgeCountInit(Event_Handle eventHandle,
                               Queue_Handle queueHandle){
    ......
    //1.设置参数
    TTCDriverTimerInit(timerInitParam,
                       &TTCDemoTimerEdgeCountHandle);
    TTCDriverTimerEdgCountSetParam(&TTCDemoTimerEdgeCountHandle,
                                   edgCountParam,
                                   Board_TIMER_EDGE_COUNT,
                                   false);
    //2.注册回调函数 TTCDemoTimerEdgeCountCB
    TTCDriverTimerRegisterIntCallBack(&TTCDemoTimerEdgeCountHandle,
                                      &TTCDemoTimerEdgeCountCB );
}
```

4.4.4.2.回调

如 demo 所设置参数，捕获一定的上升沿进一次回调函数，比如在串口打印信息等。

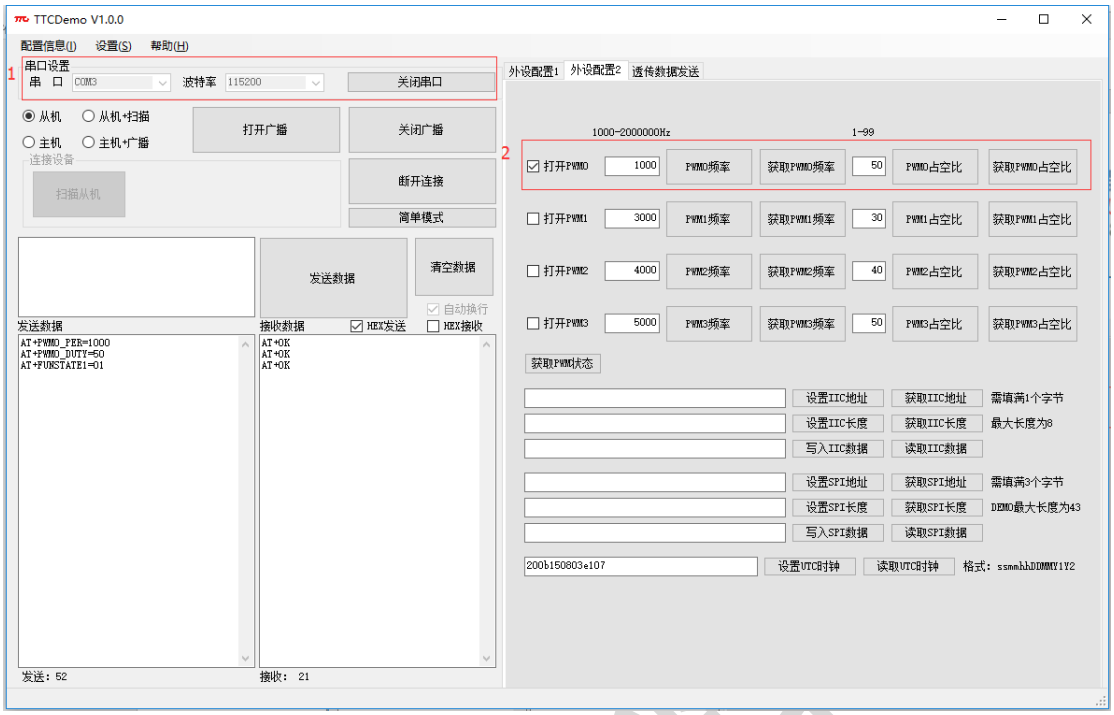
```
static void TTCDemoTimerEdgeCountCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue ){
    ......
}
```

4.4.4.3.处理流程

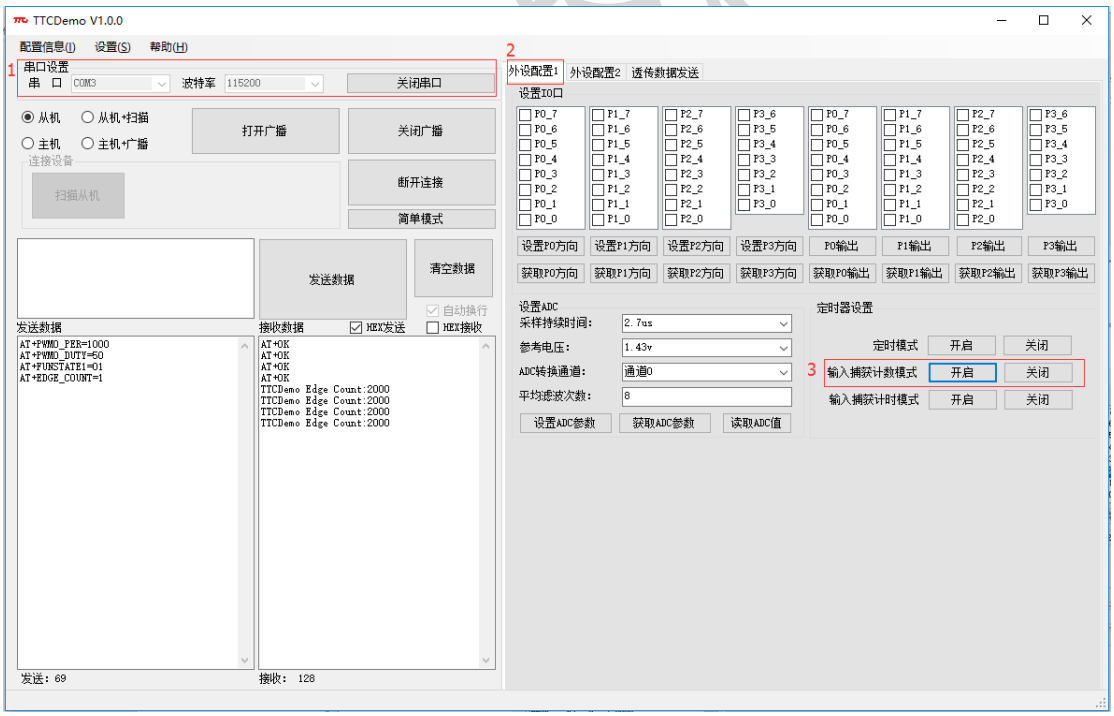
1.接收串口 AT 指令，进入串口写回调，并发消息；
2.线程唤醒，处理消息，解析指令；
3.指令正确，实现相关功能。
备注：串口指令处理，请参见 4.2.2 节串口指令处理流程。

4.4.4.4.AT 指令操作

1.按上节步骤 PWM0(IOID_10)输出 1KHz 波形，如下图：



2. PWM0 作为输入捕获的输入(IOID_15),即短接 IOID_10 与 IOID_15。



4.4.5. 输入捕获边沿计时模式

4.4.5.1 初始化

1. 输入捕获边沿计时模式初始化参数设置

```
static TTCDriverTimerEdgeTimingParam_t edgTimingParam = {
    .countValue                 = 65535,
```

```
    .upCountMode                = 1,
    .edgeMode                   = TIMER_BOTH_EDGE,
}
```

2.驱动 demo 初始化

```
void TTCDemoInit( TTCSdkClass_t *pCB,
                  Event_Handle eventHandle,
                  Queue_Handle queueHandle){
    ……
    //1.定时器 demo 驱动初始化
    TTCDemoTimerEdgeTimingInit(eventHandle,queueHandle);
    ……
}
```

3.设置参数并注册回调函数

```
void TTCDemoTimerEdgeTimingInit(Event_Handle eventHandle,
                                Queue_Handle queueHandle){
    ……
    //1.驱动初始化
    TTCDriverTimerInit(timerInitParam,
                       &TTCDemoTimerEdgeTimingHandle);
    TTCDriverTimerEdgTimingSetParam(&TTCDemoTimerEdgeTimingHandle,
                                    edgTimingParam,
                                    Board_TIMER_EDGE_TIMING,
                                    false);
    //2.注册回调函数 TTCDemoTimerEdgeTimingCB
    TTCDriverTimerRegisterIntCallBack(&TTCDemoTimerEdgeTimingHandle,
                                      &TTCDemoTimerEdgeTimingCB );
}
```

4.4.5.2.回调函数

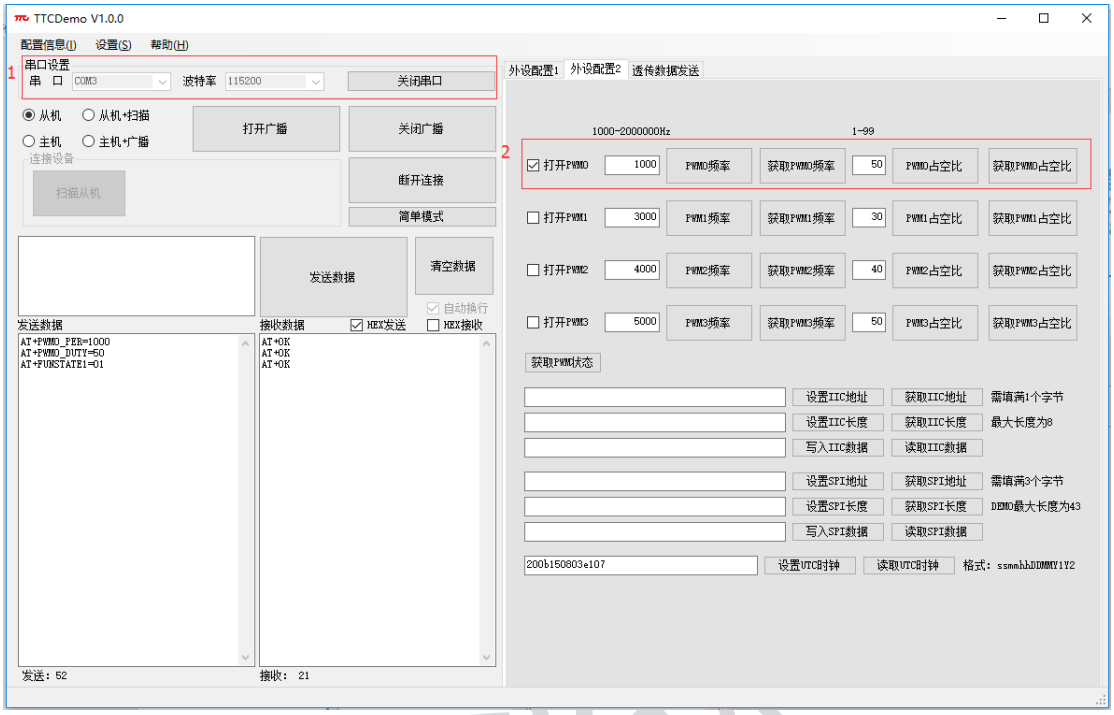
```
static void TTCDemoTimerEdgeTimingCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue ){
    ……
}
```

4.4.5.3.处理流程

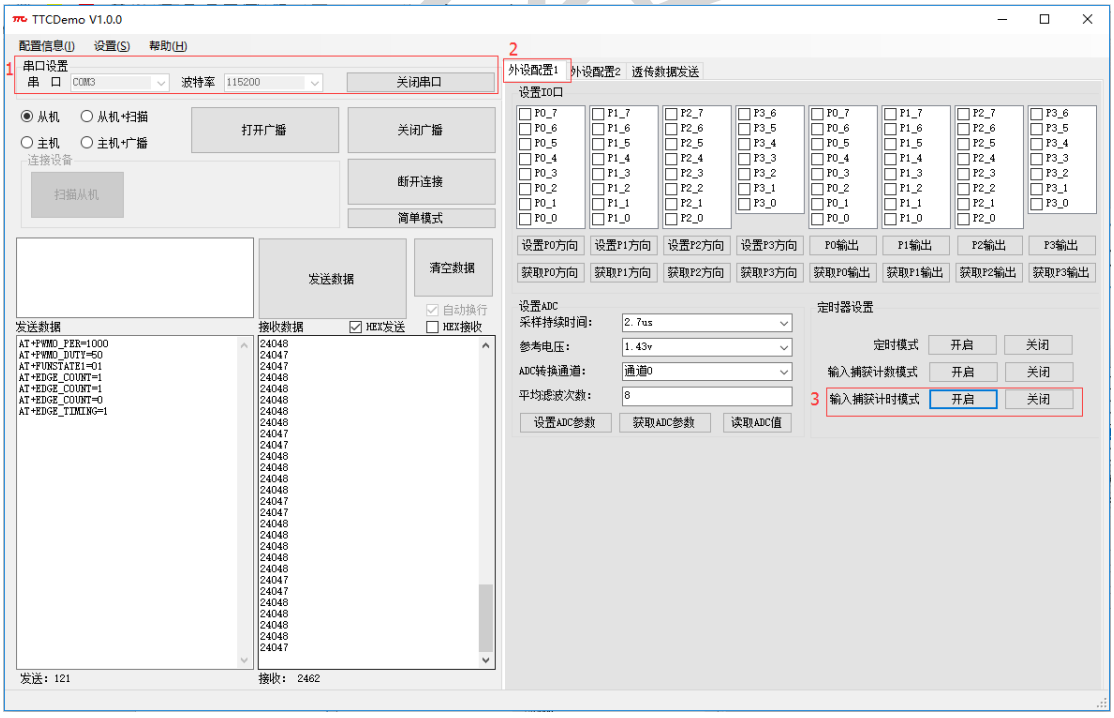
1.接收串口 AT 指令，进入串口写回调，并发消息；
2.线程唤醒，处理消息，解析指令；
3.指令正确，实现相关功能。
备注：串口指令处理，请参见 4.2.2 节串口指令处理流程。

4.4.5.4.AT 指令操作

1. 开启 PWM0 频率为 1KHz，并开启



2. 将 PWM0(IOID_10)作为捕获输入计时模式的输入，即连接 IOID_10 与 IOID_25.



4.4.6.API 说明

Timer 相关 API 见 TTCDriverTimer.h

4.5. ADC Demo

CC2640 拥有 8 路 12bit 的 ADC 通道，支持 200Ksamples 的采样率，它的时钟源可以自由设置，包括定时器，I/O 引脚，软件，模拟比较器和 RTC。另外它可以采集到片内温度传感器的当前温度值以及通过内部电路采集到电源电压，方便实现电池的管理。

4.5.1. demo 设置及注意事项

（1）设置

定义宏 TTCDRIVER_UART、TTC_DEBUG、TTCDRIVER_ADC，并关闭其他驱动的宏定义。使用串口驱动相关的函数，需要定义宏 TTCDRIVER_UART；使用串口 demo 及其他驱动 demo，需要定义宏 TTC_DEBUG。

参照章节 2使用相关工具。

（2）功能

设置 ADC 参数并输出转换结果。

（3）注意事项

每次 ADC 采集完成后，ADC 对应的 IO 口会变为 ADC 之前的配置。所以**每次启动 ADC 之前，需要将对应的 IO 口设置为浮空输入。**倘若启动 ADC 时，对应的 IO 口为输出状态，会导致两个问题：首先，结合分压电路，会影响功耗；其次，当启动 ADC 采集时，对应的 IO 口才会被配置为输入，由于电容 C1 的存在，会影响电压的变化，最终影响 ADC 采集准确度。

每次 ADC 采集的软件流程如下：

① 设置 ADC 对应的 IO 口为输入（GPIO 输入输出寄存器）；

② 设置 ADC 配置寄存器；

③ 读 ADC 数据寄存器。

4.5.2. 初始化

1. 初始化参数设置、设置引脚

```
//1.设置 ADC 参数
static AdcInfo_t  TTCDemoAdcInfo = {
    .adcReady = true,
    .filterTimes = 1,
    .TTCDemoTimerADCParam = {
                                .refsource = ADC_REF_FIXED,
                                .ref       = ADC_REF_1P43V,
                                .time      = ADC_TIME_2P7_US,
                                .trigger   = ADC_TRIGGER_MANUAL,
                                .auxIo     = ADC_COMPB_IN_AUXIO7,
                            },
    .adConfirmValue = 0,
    .adValue        = NULL,
}
```

```
//2.设置 ADC 引脚
const PIN_Config adPinConfig[] = {
    Board_ADC0 | PIN_GPIO_OUTPUT_DIS | PIN_INPUT_EN | PIN_NOPULL,        //配置为浮空输入
    ......
};
```

2.驱动 demo 初始化

```
void TTCDemoTimerEdgeTimingInit(Event_Handle eventHandle,
                                Queue_Handle queueHandle){
    ......
    //1.demo 初始化
    TTCDemoADCInit();
    ......
}
```

3.ADC 初始化

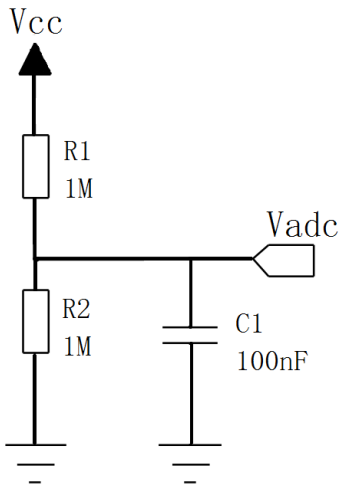
```
//1.此处只是设置引脚，并未设置 ADC 参数。需使用 AT 指令启动检测!
void TTCDemoADCInit(void){
    TTCDriverIOOpen(&adPinHandle,&adPinState,(const PIN_Config *)adPinConfig);
}
```

4.5.3.处理流程

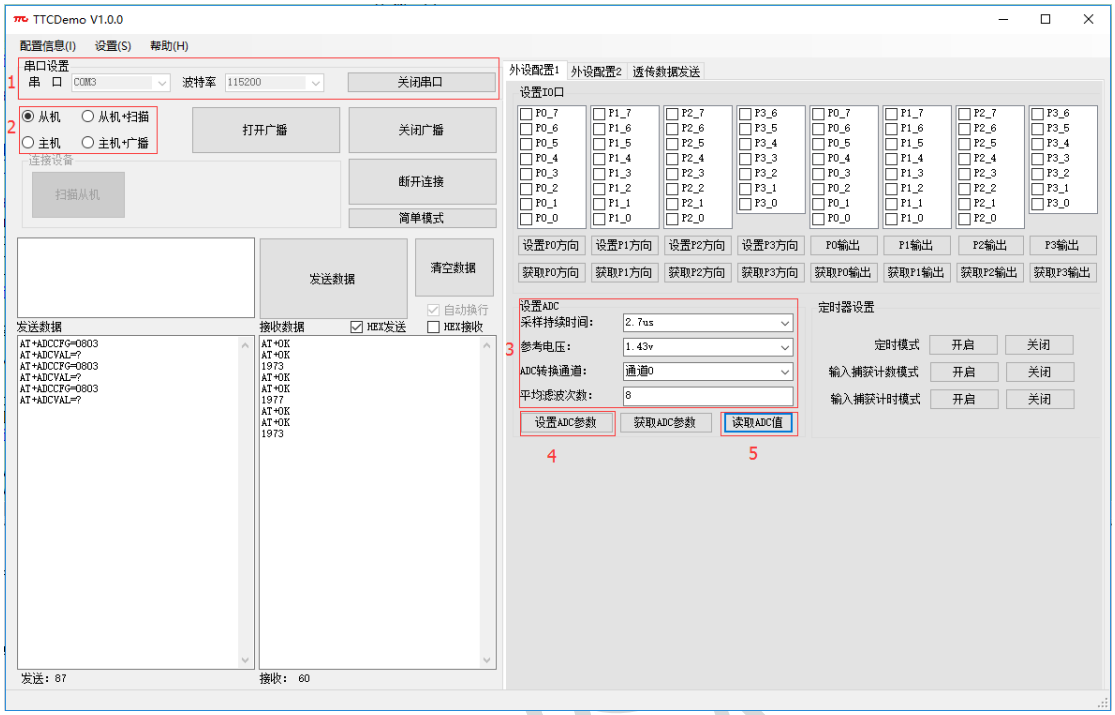
1.接收串口 AT 指令，进入串口写回调，并发消息；
2.线程唤醒，处理消息，解析指令；
3.指令正确，实现相关功能。
备注：串口指令处理，请参见 4.2.2 节串口指令处理流程。

4.5.4.AT 指令操作

1. 根据外部电压范围，ADC 输入 IO 需要外接分压电路，以及一个 100nF 电容如 C1，分压电阻可根据需求修改，可参照下图：需测量电压 Vcc,经过 R1/R2 分压，Vadc 连接至 ADC 输入引脚。Board_ADC0 对应 IDIO_23.



2. 设置 ADC 采集参数，点击“设置 ADC 参数”则启动一次检测，在点击“读取 AD 值”获取转换结果，需再转换为电压值：



4.5.5. API 说明

ADC 相关 API 见 TTCDriverADC.h

4.6. UTC Demo

CC2640 的 RTC 时钟来源于 32Khz 的外部晶振，在不断电的情况下会一直自动计数。另外它还拥有一个 70bit 的可编程的计数器以及三个通用的通道，配合它的比较寄存器使用，可以产生与时间相关的通知来告知应用层，以实现一些必要的功能。

4.6.1. demo 设置及功能

定义宏 TTCDRIVER_UART、TTC_DEBUG、TTCDRIVER_UTC，并关闭其他驱动的宏定义。使用串口驱动相关的函数，需要定义宏 TTCDRIVER_UART；使用串口 demo 及其他驱动 demo，需要定义宏 TTC_DEBUG。

参照章节 2使用相关工具。

功能：可设置、读取 UTC 时间。

4.6.2. 初始化

1. 初始化参数设置

```
//1.设置初始化时间、及 UTC 刷新时间
static TTCDriverUTCParams_t utcConfig = {
    ......
    .time = {
        .year    = 2017,
        .month   = 2,
        .day     = 28,
        .hour    = 15,
        .minutes = 7,
        .seconds = 00,
        .week    = 2,
    },
    .upDatePeriod = 1000,      //每 1000ms 更新一次时间
    .updateMsgFlag = false,    //若设置为 TRUE，则需要注册回调函数
}
```

2. 驱动 demo 初始化

```
void TTCDemoInit( TTCSdkClass_t *pCB,
                  Event_Handle eventHandle,
                  Queue_Handle queueHandle){
    ......
    //1.UTC demo 驱动初始化
    TTCDemoUTCInit(pCB, eventHandle,queueHandle);
    ......
}
```

3. 设置参数并注册回调函数(若 updateMsgFlag)为 TRUE

```
void TTCDemoUTCInit(TTCSdkClass_t *pCB, Event_Handle eventHandle, Queue_Handle queueHandle){
```

```
    ……
    TTCDriverUTCInit(pCB,utcConfig);
}
```

4.6.3. 回调函数

updateMsgFlag = false, //若设置为 TRUE，则需要注册回调函数
每次到达 UTC 刷新时间均会进入回调函数，可以检测闹钟等。

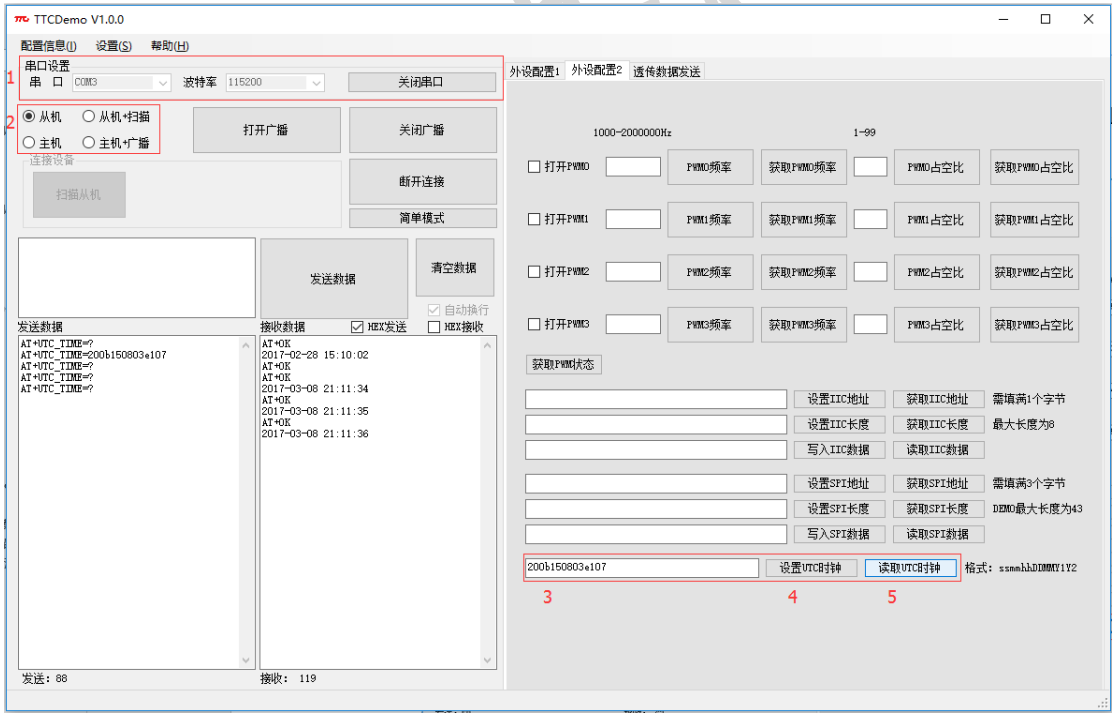
4.6.4. 处理流程

1. 接收串口 AT 指令，进入串口写回调，并发消息；
2. 线程唤醒，处理消息，解析指令；
3. 指令正确，实现相关功能。

备注：串口指令处理，请参见 4.2.2 节串口指令处理流程。

4.6.5. AT 指令操作

例如设置时间 2017/03/08 21:11:32，格式 0x200b150803e107,并可读取 UTC 时钟：



4.6.6. API 说明

UTC 相关 API 见 TTCDriverUTC.h

4.7. IIC Demo

IIC 接口可用与其他支持 IIC 协议的器件通信，如 ROM，LCD 及多种传感器等。普通模式速率为 100KHz，快速模式速率为 400KHz。

4.7.1. demo 设置及注意事项

(1)设置及功能

定义宏 TTCDRIVER_UART、TTC_DEBUG、TTCDRIVER_I2C，并关闭其他驱动的宏定义。使用串口驱动相关的函数，需要定义宏 TTCDRIVER_UART；使用串口 demo 及其他驱动 demo，需要定义宏 TTC_DEBUG。

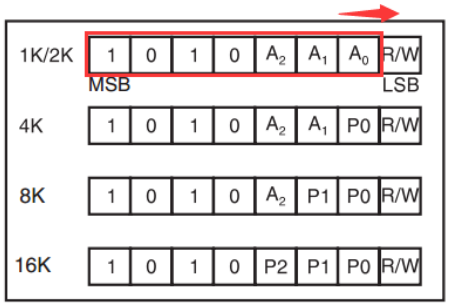
参照章节 2 使用相关工具。

功能：IIC 接口实现 EEPROM 读写。

(2)注意

在 demo 程序中，所使用 EEPROM 为 AT24C02，器件地址(Device Address)说明如：(#define I2C_DEMO_24C02_ADDR 0x50) 其中，0x50 为器件地址去除最低位”R/W”的值，即整体右移 1bit 后的结果。

Figure 7. Device Address



4.7.2. 初始化

1. 初始化参数设置

```
static I2cInfo_t TTCDemoI2cInfo = {
    .address = 0,
    .maxTranSize = 8,
    .size = 0,
};
```

2. 驱动 demo 初始化

```
void TTCDemoInit( TTCSdkClass_t *pCB,
                  Event_Handle eventHandle,
                  Queue_Handle queueHandle){
    ......
    //1. demo 驱动初始化
    TTCDemoI2cInit(eventHandle,queueHandle);
    ......
```

```
}
```

3.驱动初始化

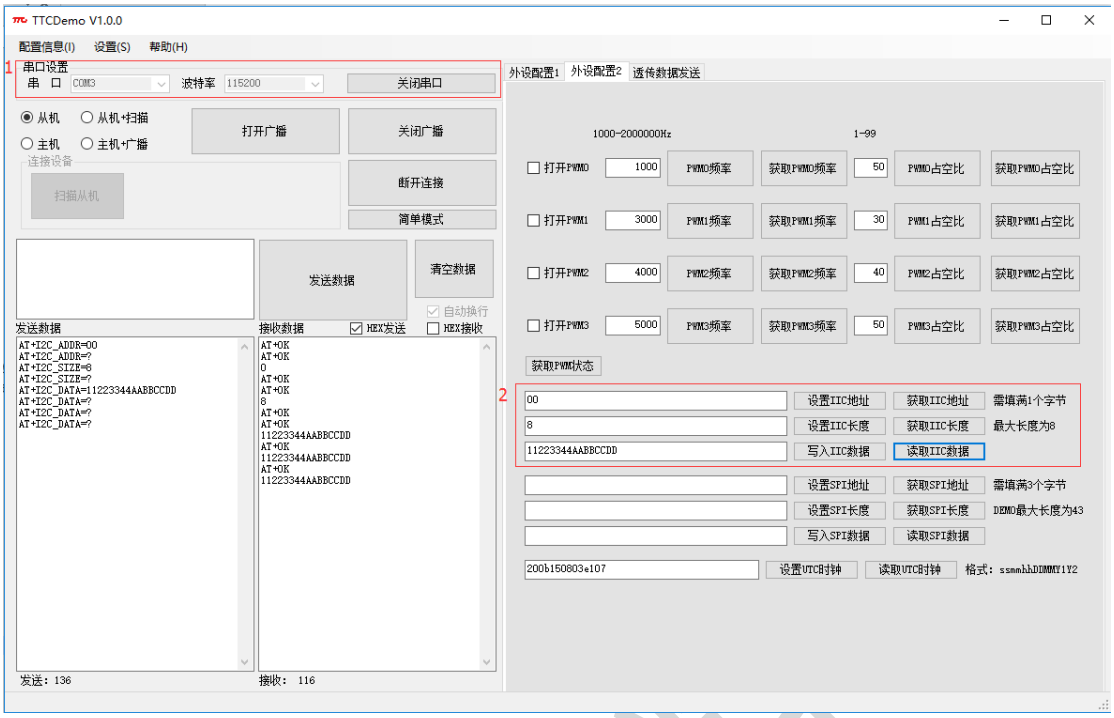
```
void TTCDemoI2cInit(Event_Handle eventHandle,
                    Queue_Handle queueHandle){
    //1.参数设置
    TTCDriverI2cInitDefaultParam(&i2cHandle);
    i2cHandle. eventHandle  = eventHandle;
    i2cHandle.queueHandle   = queueHandle;
    const TTCDriverI2cParams_t param = {
        .i2cName = CC2650_I2C0,
        .bitRate = I2C_400kHz,
        .transferMode = I2C_MODE_BLOCKING,
        .i2cHWAttr = &i2cCC26XXHWAttrs[CC2650_I2C0],
    };
    TTCDriverI2cInit( NULL,
                      &i2cHandle,
                      param);
}
```

4.7.3.处理流程

1.接收串口 AT 指令，进入串口写回调，并发消息；
2.线程唤醒，处理消息，解析指令；
3.指令正确，实现相关功能。
备注：串口指令处理，请参见 4.2.2 节串口指令处理流程。

4.7.4.AT 指令操作

1.IIC 接口连接 AT24C02
2.设置内部地址为 0x00,长度为 0x08,写入数据 0x11223344AABBCCDD.并可读回数据。



4.7.5. API 说明

I2C 相关 API 见 TTCDriverI2C.h

4.8. SPI Demo

SPI 是一种同步串行接口，用于 CPU 与各种外围器件进行全双工，同步串行通讯，有四条线，时钟线 SCLK,主机输入从机输出数据线 MISO、主机输出从机输入数据线 MOSI、片选线 CSN，有频率可编程以及写冲突保护等特点，2640 的 SPI 驱动做主机最大支持到 24M,做从机最大为 4M.

4.8.1. demo 设置及功能

定义宏 TTCDRIVER_UART、TTC_DEBUG、TTCDRIVER_SPI、TTCDRIVER_SPI_FLASH，并关闭其他驱动的宏定义。使用串口驱动相关的函数，需要定义宏 TTCDRIVER_UART；使用串口 demo 及其他驱动 demo，需要定义宏 TTC_DEBUG。

参照章节 2 使用相关工具。

功能：SPI 接口实现 Flash 读写。

4.8.2. 初始化

1. 初始化参数设置

```
static SpiInfo_t TTCDemoSpiInfo = {0
};
```

2. 驱动 demo 初始化

```
void TTCDemoInit( TTCSdkClass_t *pCB,
                  Event_Handle eventHandle,
                  Queue_Handle queueHandle){
    ......
    //1.demo 驱动初始化
     TTCDemoSPIInit(eventHandle,queueHandle);
    ......
}
```

3. 驱动初始化

```
void TTCDemoSPIInit(Event_Handle eventHandle,
                    Queue_Handle queueHandle){
    spiFlashOpen(eventHandle,queueHandle);
}
```

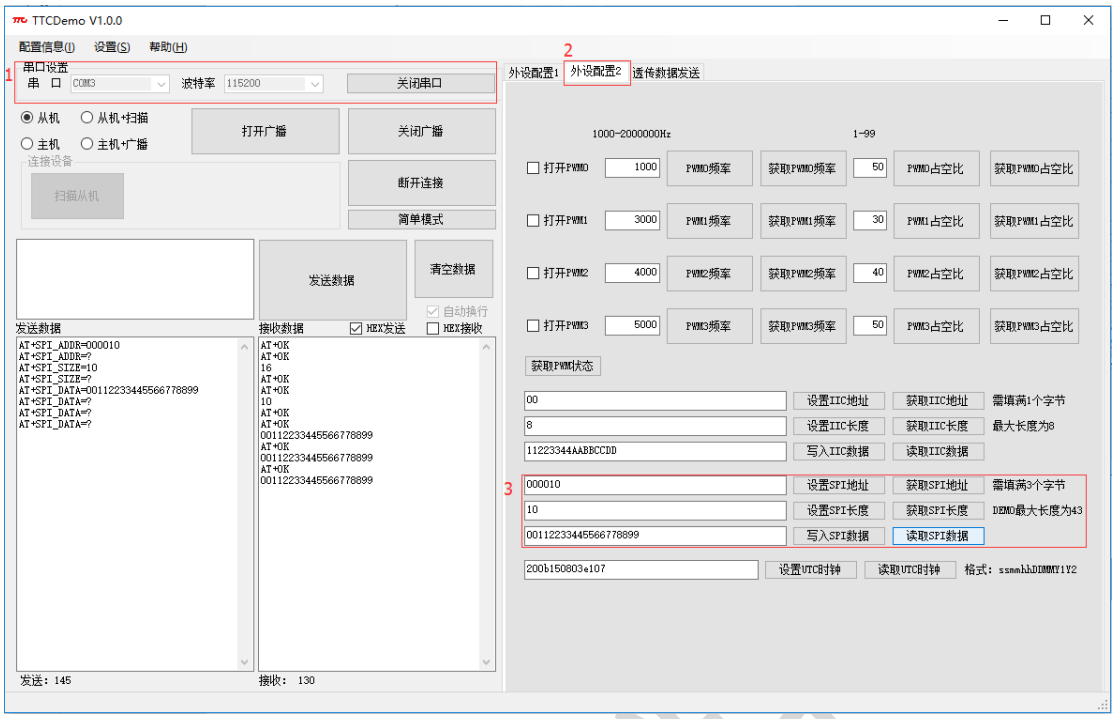
4.8.3. 处理流程

1. 接收串口 AT 指令，进入串口写回调，并发消息；
2. 线程唤醒，处理消息，解析指令；
3. 指令正确，实现相关功能。

备注：串口指令处理，请参见 4.2.2 节串口指令处理流程。

4.8.4. AT 指令操作

1. USB 转接口焊接 falsh-FT25H04
2. 设置 flash 地址，数据长度，便可实现数据读写，如下图：



4.8.5. API 说明

SPI 相关 API 见 TTCDriverSPI.h

4.9 Watchdog 说明

看门狗（Watchdog）用于防止程序跑飞，在程序跑飞时复位 MCU 在一定程度上保证设备的正常运行。超宽范围的溢出时间设置，范围为 1ms~2863311 ms，并可随时重新调整装载值。注意在 MCU 睡眠状态时看门狗的溢出计时器是暂停工作的。

4.9.1.demo 设置

定义宏 TTCDRIVER_WATCHDOG

4.9.2 Watchdog 使用说明

WatchDog 功能使用说明

1. 程序初始化时，看门狗初始化时启动定时：TTCDriverWatchDogInit(1000);//设置看门狗超时时间为 1 秒

2.不对看门狗进行重载，若系统是有任务运行的：看门狗溢出，程序复位。若系统是睡眠的：看门狗停止计时，程序程序不会复位，直到系统醒来，看门狗溢出，程序才会复位。

3.定时对看门狗重载：TTCDriverWatchDogSetReload(1000);系统运行正常，程序不会复位，当程序跑飞导致定时事件不能运行时，程序会复位。

建议：看门狗重载可以使用定时事件，也可以计算出代码运行时间插入到程序中。

4.9. WeChat Demo

微信驱动包含了微信的 AirSync 的蓝牙通信协议；自动处理被微信发现、与微信握手连接、数据的蓝牙分包等，用户只需要进行相关的初始化，并调用相应的 API 进行收、发数据即可。

4.9.1. demo 设置及功能

（1）demo 设置

在从机角色、或者从机+观察者角色时，定义宏 TTCBLE_WECHAT，即可开启 WeChat 相关功能。

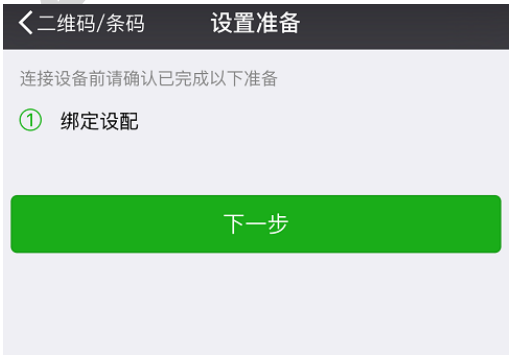
（2）注意事项

以下示例中，微信公众号“深圳市昇润科技有限公司”进入相关界面可以与模组建立链接并进行数据通信。需要注意，SDK 中微信功能所使用的 Mac 地址，是需要经过授权才能正常使用的。示例中仅使用同一个已授权的 Mac 地址及二维码作为测试使用，实际产品生产需使用不同的 Mac 地址，相关授权事宜请与我司联系。

4.9.2. 模块与微信公众号绑定连接

(1)开启微信，扫一扫以下二维码

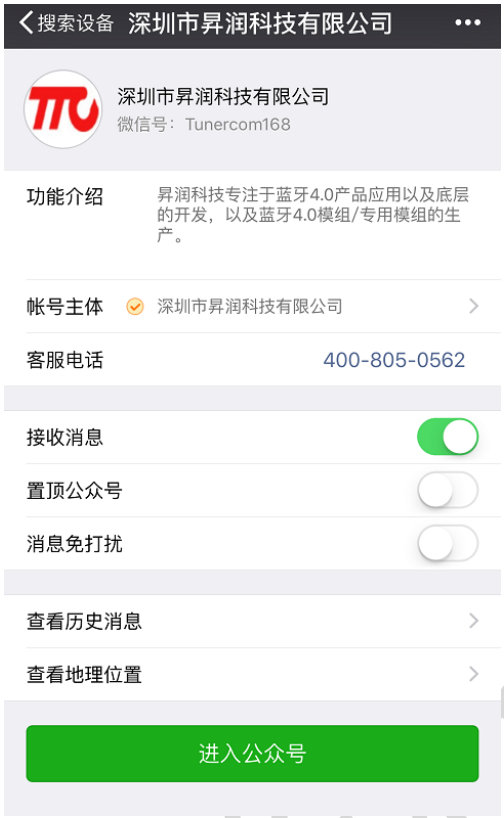
（2）如下为弹出界面，点击“下一步”



（3）进入扫描界面，并扫描到设备，如下图，点击设备

（4）进入公众号，点击绑定设备

(5)点击进入公众号

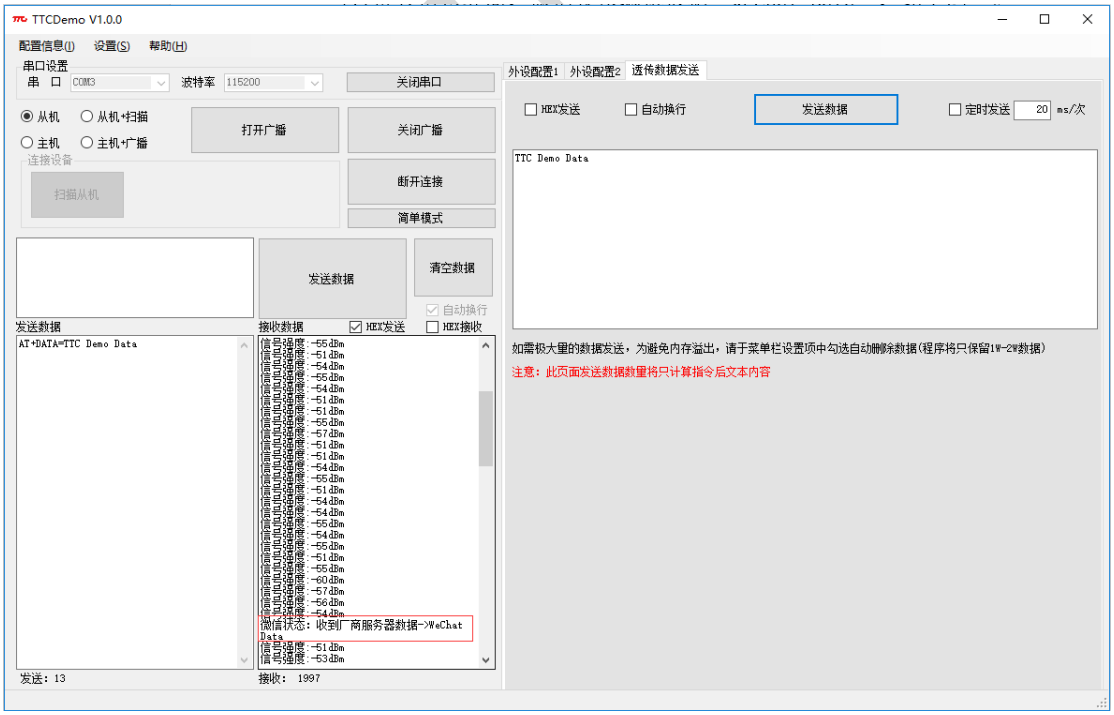


(6)等待与设备建立连接，显示“已连接”；点击“智能硬件”“透传”：

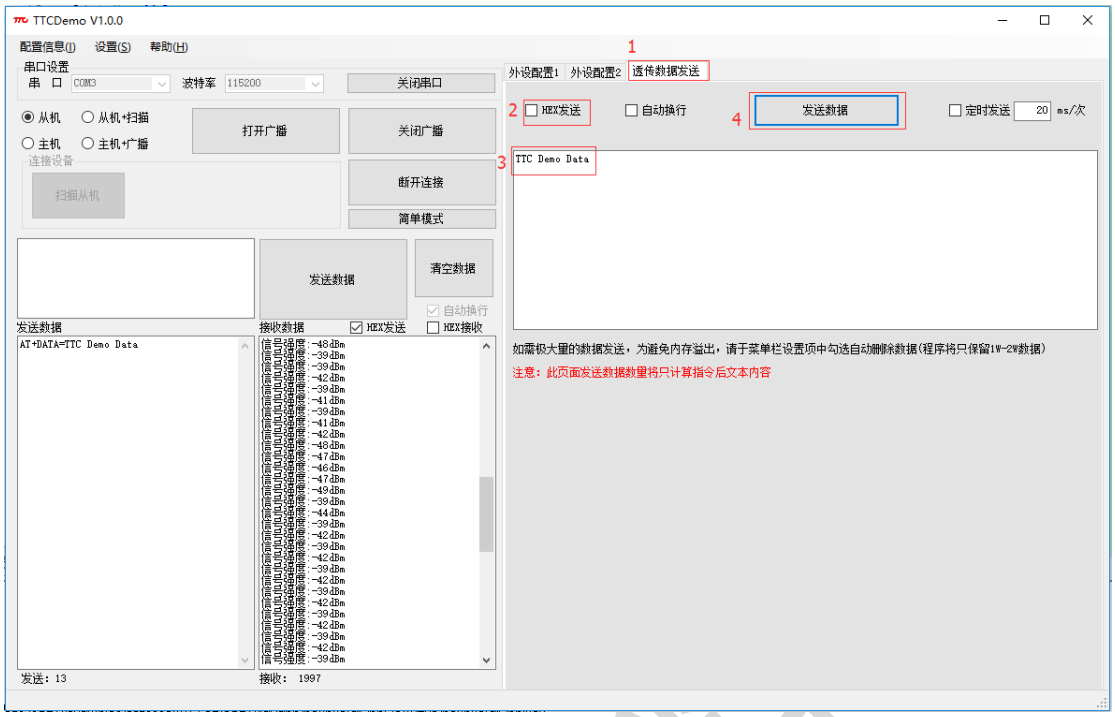
4.9.3. 数据收发

(1)进入透传界面，微信发送测试数据“WeChat Data”，如下图：

CC2640 模组收到微信数据后，发送到 PC 端软件(TTCDemo)显示如下：



(2) PC 端软件(TTCDemo)发送数据“TTC Demo Data”：



CC2640 模组将数据发送给微信，接收框显示接收到的数据，如下图：

4.9.4. API 说明

WeChat 相关 API 见 TTCBleWechat.h

4.10. Beacon Demo

4.10.1. demo 设定及注意事项

在从机角色、或者从机+观察者角色时，定义宏 TTCBLE_IBEACON,则开启 Beacon 功能。

功能：见< TTC Beacon 规格书 V1.6.pdf >

注意：Beacon 应用中参数设置支持掉电储存。用到掉电存储功能，需选择合适的工程配置。

4.10.2. API 说明

Beacon 相关 API 见 TTCBeacon.h

5. 空中升级说明（OAD）

5.1. TTC SDK OAD 简介

OAD, Over-the-Air Download, 即空中升级。

本章主要是介绍 TTC SDK 里面的 OAD 使用方法，OAD 分为两种:一种是外部 OAD(**Off-Chip OAD**)，另外一种是内部 OAD(**On-Chip OAD**)，什么是外部？什么是内部？其实外部和内部在硬件上的差别仅仅是多了一个外部 flash，使用带外部 flash 的方式，我们成为外部 OAD 升级，如果没有使用外部 flash，我们称之为内部 OAD 升级。

在软件方面，外部 OAD 升级原理是，先将 BLE 透传过来的数据，存贮在外部 flash 中，然后比对 BLE 下发校验和与 flash 实际数据的计算的校验和，如果比对成功，设置相应的启动元数据，然后重启系统，加载外部 flash 镜像到内部 flash，加载完毕，系统会设置对应的元数据，并确认固件已经更新最新版本。然后系统将会跳转到指定镜像地址开始执行程序。

内部 oad 升级原理是，BLE 发送 OAD 升级命令，唤醒模块的 OAD 升级服务程序，系统接收到 BLE 发送的 OAD 升级命令，设置元数据，复位系统，然后跳转到 OAD 升级服务程序，升级服务程序与 BLE 连接后，BLE 就可以开始下发镜像文件，OAD 升级服务程序将镜像直接写入到原来的镜像空间，写完后，比对 BLE 下发的校验和与写入内部 flash 的校验和，比对成功，更新元数据，并跳转到新的镜像文件，开始执行新的程序，至此，内部 OAD 升级完成。

说明：

(1)当前支持片外升级的角色：从机、从机+观察者

当前支持片内升级的角色：从机

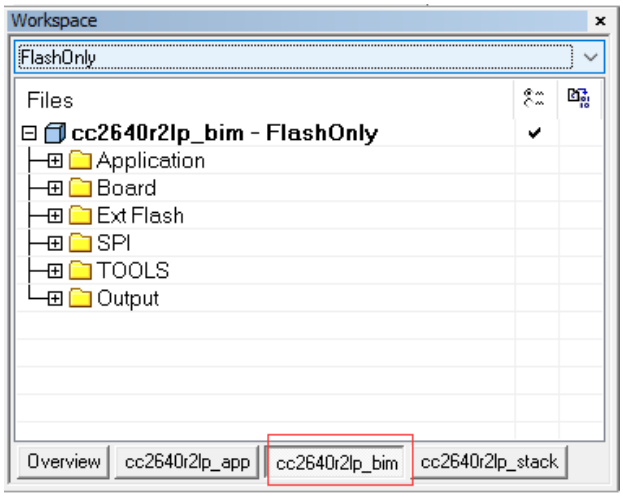
(2)带 OAD 功能的配置，不支持在线调试仿真，不可直接使用 IAR 下载程序。为方便在线仿真调试，可先使用不带 OAD 的工程进行项目开发，功能完善后，再切换至对应的 OAD 配置，即完成 OAD 功能的添加。

5.2. 外部 OAD

5.2.1. bim 文件生成

BIM-Boot Image Manager, the software bootloader, 启动代码。CC2640 上电复位后，由 BIM 检查片外 flash 是否存在新的程序需要升级至 CC2640 内部 flash。

打开 TTCBleSDK 工程，切换至 bim_extflash 工程（此工程仅在外部 OAD 时使用），设置外部 flash 引脚及 ID. 如图：



(1) 查看原理图，设定 flash SPI 接口，在 TTCSDKBoard.h 中设置 SPI 引脚。默认引脚为

```
/*TTC BLE SDK FLASH*/
#define Board_FLASH_CS_ON                0
#define Board_FLASH_CS_OFF               1
#define Board_DEVPK_CS_ON                1
#define Board_DEVPK_CS_OFF               0

#define EXT_FLASH_MAN_ID                 0x0E
#define EXT_FLASH_DEV_ID                 0x12

#define Board_SPI_FLASH_CS               IOID_1
#define Board_SPI_DEVPK_CS               IOID_1
#define Board_SPI_FLASH_MISO             IOID_2
#define Board_SPI_FLASH_MOSI             IOID_5
#define Board_SPI_FLASH_CLK              IOID_6
#define Board_SPI_FLASH_CSN              PIN_UNASSIGNED
```

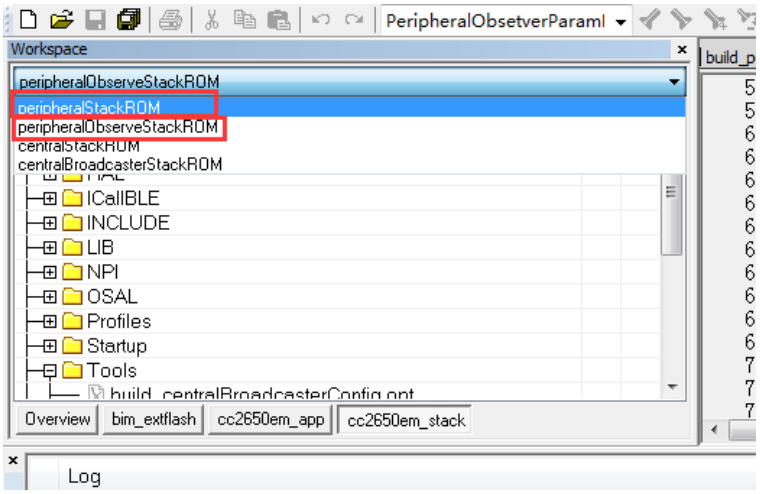
(2) 根据所使用的 flash 的类型，在 ext_flash.c 中设定厂商 ID 及器件 ID;

```
// Supported flash devices
static const ExtFlashInfo_t flashInfo[] =
{
    {
        .manfId = 0xC2,         // Macronics MX25R1635F
        .devId = 0x15,
        .deviceSize = 0x200000  // 2 MByte (16 Mbit)
    },
    {
        .manfId = 0xC2,         // Macronics MX25R8035F
        .devId = 0x14,          //
        .deviceSize = 0x100000  // 1 MByte (8 Mbit)
    },
    {
        .manfId = 0xEF,         // WinBond W25X40CL
        .devId = 0x12,
        .deviceSize = 0x080000  // 512 KByte (4 Mbit)
    },
    {
        .manfId = 0xEF,         // WinBond W25X20CL
        .devId = 0x11,
        .deviceSize = 0x040000  // 256 KByte (2 Mbit)
    },
    {
        .manfId = 0x0E,         // TTC Demo SPI Flash
        .devId = 0x12,
        .deviceSize = 0x080000  // 512 KByte (4 Mbit)
    },
    {
        .manfId = 0x0,
        .devId = 0x0,
        .deviceSize = 0x0
    }
};
```

(3) 重新编译工程，生成 BIM_ext.hex 文件。

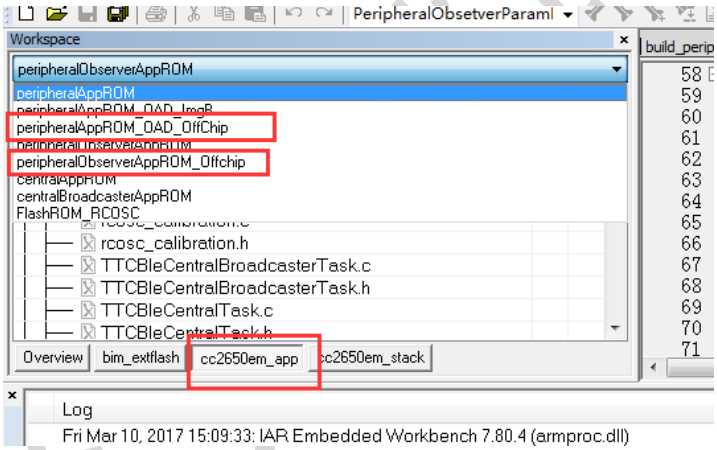
5.2.2. cc2640stack 生成

选择 cc2640r2lp_stack 工程，根据实际使用的角色来选择对应的协议栈编译。



5.2.3. cc2640app 生成

选择 cc2640r21p_app 工程，根据实际使用的角色来选择对应的应用程序编译。



5.2.4. HEX 文件合并

以下以从机工程片外部 OAD 为例：

1. 开启 TTC_Programmer V1.2.0.0.exe；
2. 如下图步骤 1，选择工程目录；
3. 如步骤 2，选择 Python 路径；
4. 如步骤 3，选择“从机·片外 OAD”
5. 如步骤 4，点击“读取”，显示 OAD 相关信息，自动检查相关文件已经生成。
6. 如步骤 5，点击“生成”，提示“生成成功”；
7. 输出文件：

   (1)peripheral_OAD_OffChip1.hex 即为最终合并文件，可用于生产烧录；

   (2)peripheral_OAD_OffChip1.bin 为手机 APP OAD 所需要的文件。

8. 烧录合并文件（peripheral_OAD_Offchip.hex），则 CC2640 正常运行。

5.2.5. 手机 APP(TTC-BLE)操作说明

（1）将上小节生成的 bin 文件 peripheral_OAD_Offchip.bin 导入手机根目录 Download 目录下；

（2）扫描二维码，下载安装 TTC-BLE（见附录），并开启手机蓝牙；

（3）开启 TTC-BLE，点击 OAD，进入 OAD 模式，如下图 1；

（4）点击设备，与设备建立链接，如下图 2；

（5）点击“OAD”，进入 OAD 界面，如下图 3；

（6）选择 OAD 类型“CC2640 R2 OAD”，如下图 4；

（7）点击“+”，导入 bin 文件,文件导入后，File Image Type 为 B，设置传输间隔为 20ms，点击开始，如下图 5；

（8）传输至 100%后，不需要操作 APP。等待 10 秒左右，设备复位，运行新程序，与 APP 断开链接，如下图 6。

图 1 OAD 模式

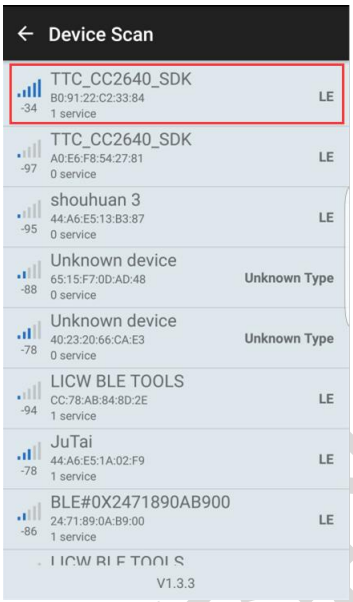


图 2 链接设备

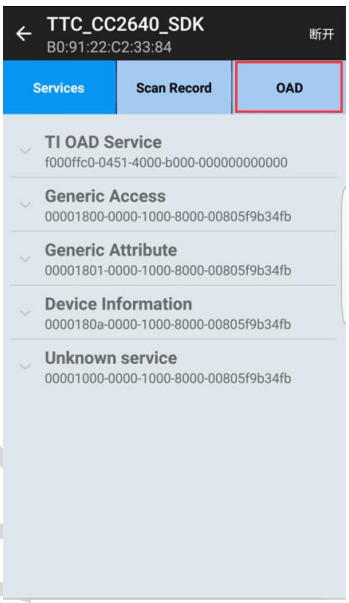


图 3 OAD 界面

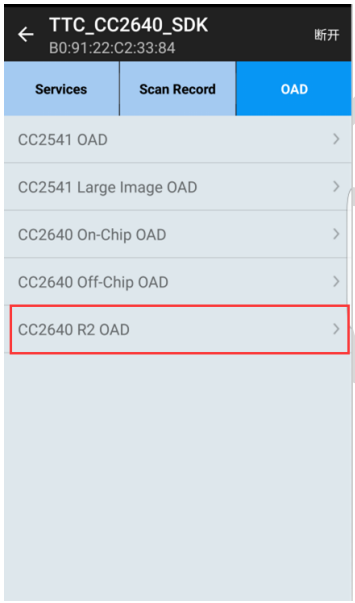


图 4 OAD 类型

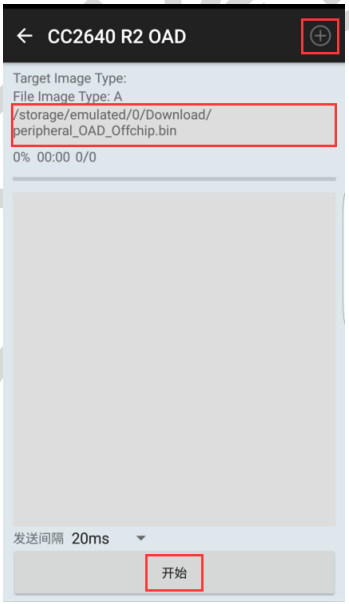


图 5 导入文件

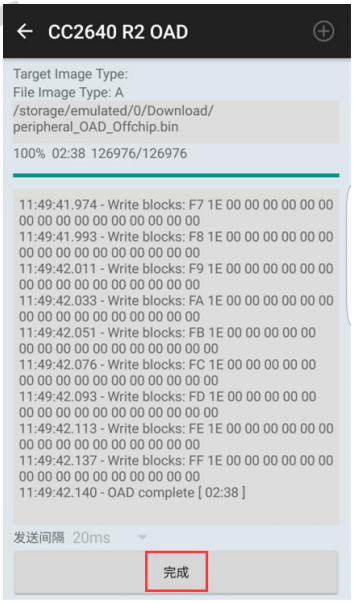
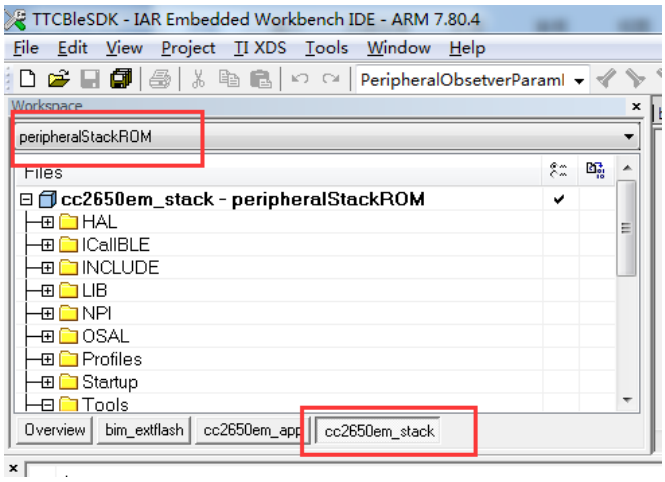


图 6 OAD 完成

5.3. 内部 OAD

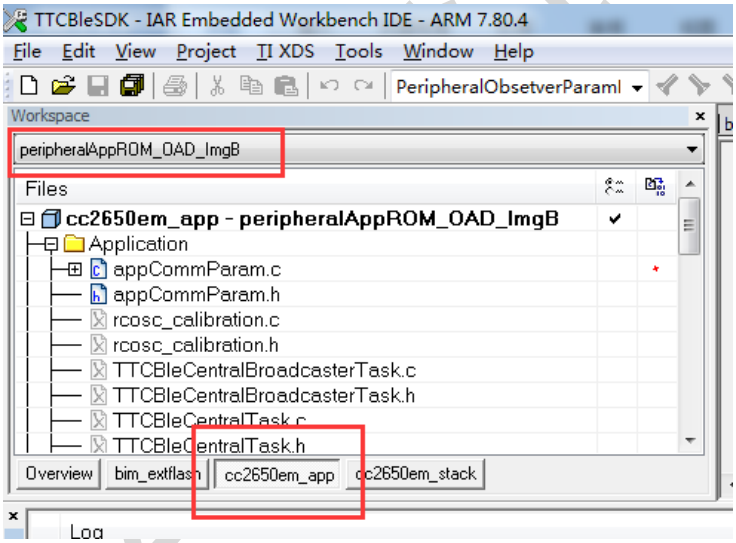
5.3.1. cc2640stack 生成

选择 cc2640r21p_stack 工程，根据选择从机协议栈编译。



5.3.2. cc2640app 生成

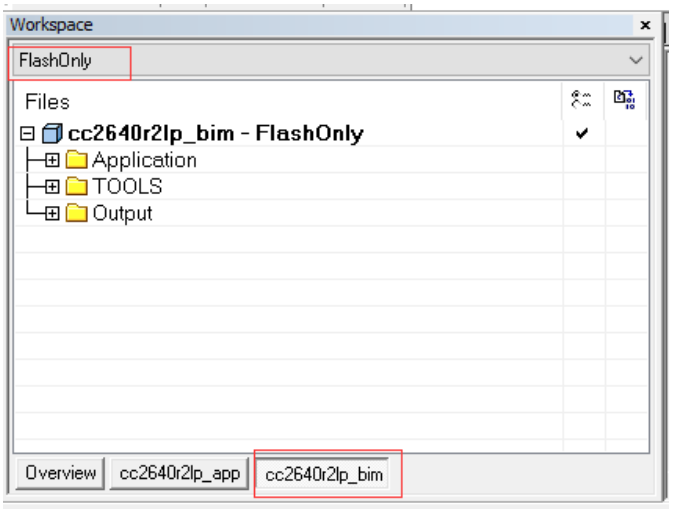
选择 cc2640r2lp_app 工程，根据选择从机角色编译。（从机+观察者模式暂不支持片内 OAD）

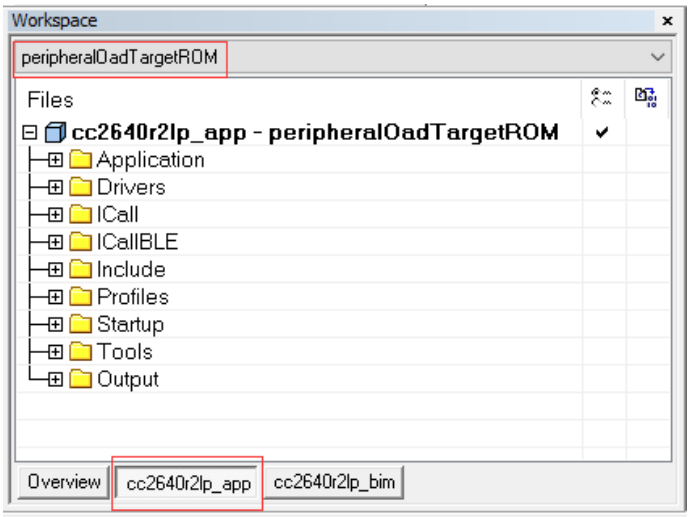


5.3.3. oad_target 工程生成

打开 oad_target 工程，先编译 bim 工程，再编译 cc2640r2lp_app 工程。如 SDK 3.1.0，工程路径：

C:\TTC_BLE_CC2640_R2_SDK\3.1.0\TTC_CC2640_R2_SDK\examples\rtos\CC2640R2_LAUNCHXL\blestack\oad_target\tirtos\iar





5.3.4.HEX 文件合并

以下以从机工程片内部 OAD 为例：

1.开启 TTC_Programmer V1.2.0.0.exe；
2.如下图步骤 1，选择工程目录；
3.如步骤 2，选择 OADTarget 目录；
4.如步骤 3，选择 Python 路径；
5.如步骤 4，选择“从机·片内 OAD”
6.如步骤 5，点击“读取”，显示 OAD 相关信息，自动检查相关文件已经生成。
7.如步骤 6，点击“生成”，提示“生成成功”；
8.输出文件：
(1) peripheralOnchip.hex 即为最终合并文件，用于烧录；
(2) peripheralAppROM_OAD_ImgB_Program.bin 为手机 APP OAD 所需文件。

9. 烧录合并文件（peripheral_OAD_Offchip.hex），则 CC2640 正常运行。

5.3.5. 手机 APP(TTC-BLE)操作说明

（1)将上小节生成的 bin 文件 peripheralAppROM_OAD_ImgB_Program.bin 导入手机根目录 Download 目录下；

（2）扫描二维码，下载安装 TTC_BLE，并开启手机蓝牙；

（3）开启 APP，点击 OAD，进入 OAD 模式，如下图 1；

（4）点击对应设备，与设备建立链接，如下图 2；

（5）展开 TI Reset Service，在 Reset 通道写入 0x01，如下图 3、图 4；

（6）等待设备自动断开，再返回上一个界面，如下图 5；

（7）点击连接，如下图 6；

（8）连接成功后，仅有 TI OAD Service，点击 OAD，如下图 7；

（9）选择 OAD 类型“CC2640 R2 OAD”，如下图 8；

（10）点击“+”，导入 bin 文件，File Image Type 为 B，如下图 9；

（11）设置传输间隔为 20ms，点击开始，如下图 10；

（12）传输至 100%后，不需要操作 APP。设备复位，运行新程序，与 APP 断开链接，如下图 11；

图 1 OAD 模式

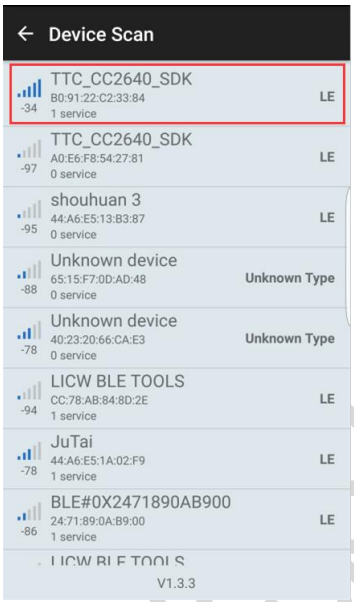


图 2 链接设备

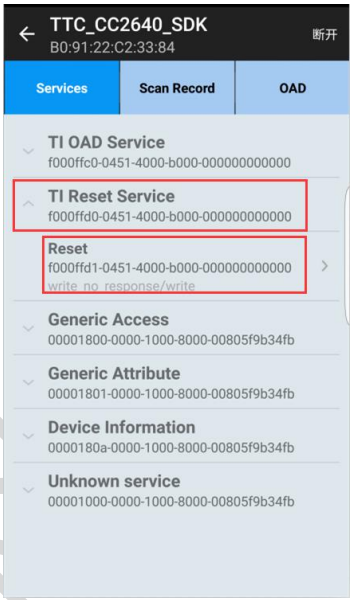


图 3 OAD 指令通道

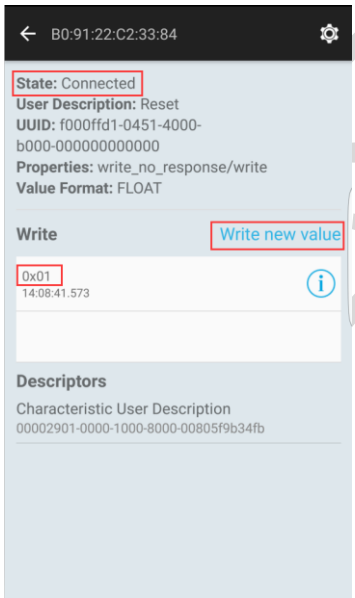


图 4 写 OAD 指令

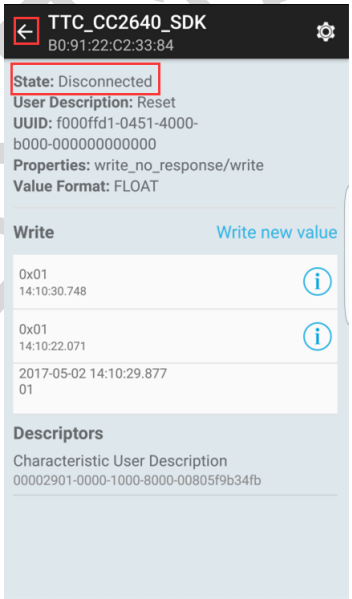


图 5 等待断线

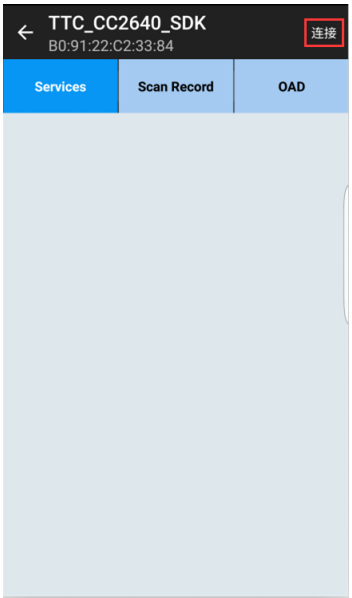


图 6 重新连接

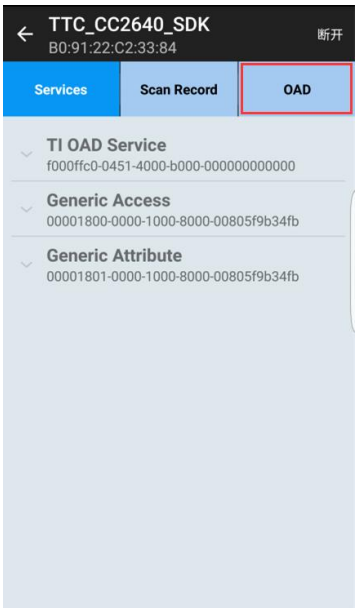


图 7 进入 OAD 界面

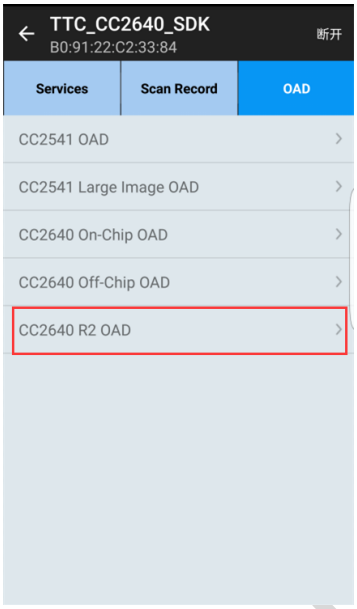


图 8 选择 OAD 类型

图 9 载入文件

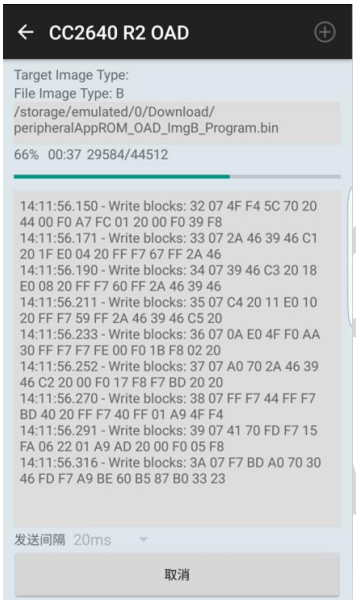


图 10 正在 OAD

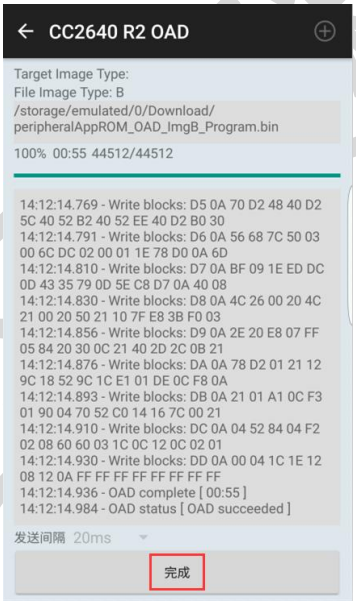


图 11 OAD 完成

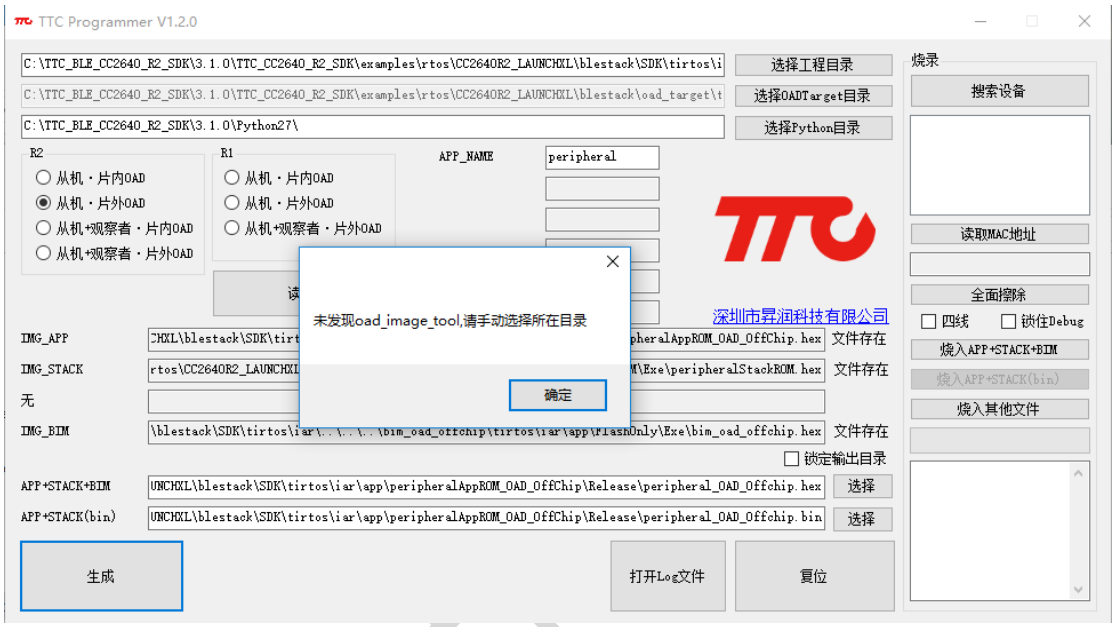
5.4. 常见问题

5.4.1. OAD 工程不能仿真

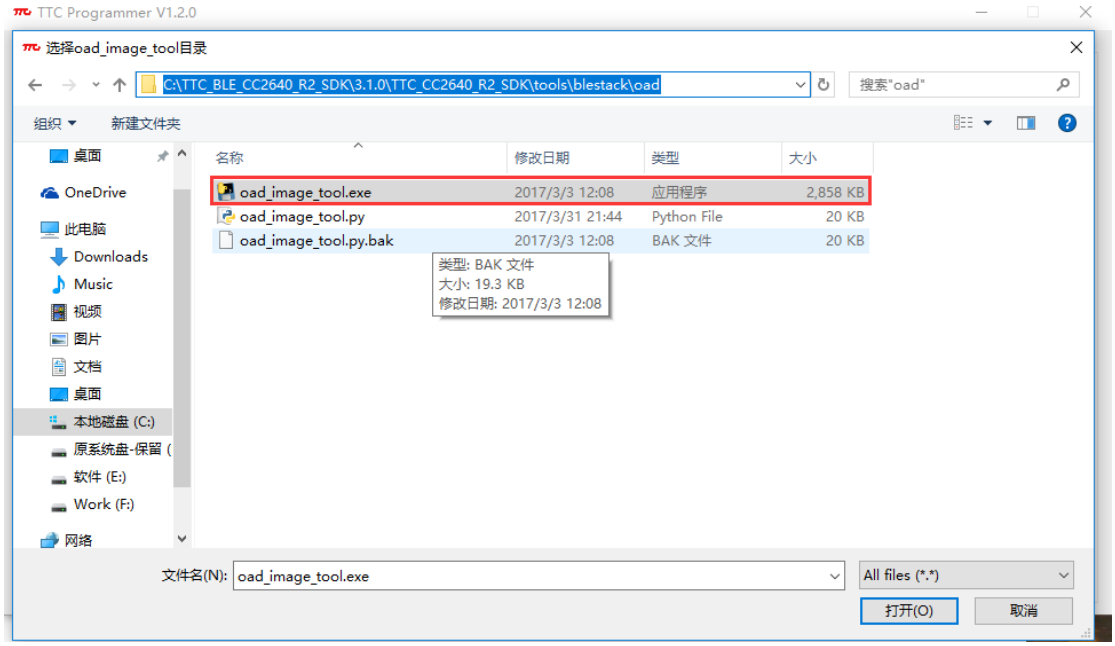
带 OAD 功能的配置，不支持在线调试仿真，不可直接使用 IAR 下载程序。 为方便在线仿真调试，可先使用不带 OAD 的工程进行项目开发，功能完善后，再切换至对应的 OAD 配置，重新编译工程，即完成 OAD 功能的添加。

5.4.2. TTC Programmer 提示错误

点击“生成”时，如果提示“未发现 oad_image_tool,请手动选择所在目录”：



可在点击“确定”后，手动添加 oad_image_tool 的路径，在如下图路径：



添加成功后，则提示“生成成功”，如下图：


TTC Programmer V1.2.0
C:\TTC_BLE_CC2640_R2_SDK\3.1.0\TTC_CC2640_R2_SDK\examples\rtos\CC2640R2_LAUNCHXL\blestack\SDK\tirtos\i
选择工程目录
C:\TTC_BLE_CC2640_R2_SDK\3.1.0\TTC_CC2640_R2_SDK\examples\rtos\CC2640R2_LAUNCHXL\blestack\oad_target\t
选择OADTarget目录
C:\TTC_BLE_CC2640_R2_SDK\3.1.0\Python27\
选择Python目录
烧录
搜索设备
R2
从机・片内OAD
从机・片外OAD
从机+观察者・片内OAD
从机+观察者・片外OAD
R1
从机・片内OAD
从机・片外OAD
从机+观察者・片外OAD
APP_NAME
peripheral
读取
读取MAC地址
全面擦除
深圳市昇润科技有限公司
四线
锁住Debug
IMG_APP
CHXL\blestack\SDK\tirtos\iar\app\peripheralAppROM_OAD_OffChip\Exe\peripheralAppROM_OAD_OffChip.hex
文件存在
烧入APP+STACK+BIM
IMG_STACK
rtos\CC2640R2_LAUNCHXL\blestack\SDK\tirtos\iar\Stack\peripheralStackROM\Exe\peripheralStackROM.hex
文件存在
烧入APP+STACK(bin)
无
烧入其他文件
IMG_BIM
\blestack\SDK\tirtos\iar\..\..\..\bim_oad_offchip\tirtos\iar\app\FlashOnly\Exe\bim_oad_offchip.hex
文件存在
锁定输出目录
APP+STACK+BIM
UNCHXL\blestack\SDK\tirtos\iar\app\peripheralAppROM_OAD_OffChip\Release\peripheral_OAD_OffChip.hex
选择
APP+STACK(bin)
UNCHXL\blestack\SDK\tirtos\iar\app\peripheralAppROM_OAD_OffChip\Release\peripheral_OAD_OffChip.bin
选择
生成
生成成功
打开Log文件
复位

6. 通用测试程序说明

6.1. 测试引脚的定义

在 TTCSDKBoard.h 中定义了相关引脚如下，可修改后面的 IOID，禁止修改前面的宏标志。

```
#define TTCTEST_ENTER_TEST_IO       IOID_0          // 上电前拉低，进入测试模式
#define TTCTEST_PWM_PIN             IOID_10         // 48M
#define TTCTEST_UART_RX             IOID_14         // RX
#define TTCTEST_UART_TX             IOID_13         // TX
#define TTCTEST_UART_WAKEUP         IOID_12         // WAKEUP
```

引脚功能说明：

(1) TTCTEST_ENTER_TEST_IO

测试模式触发引脚，在上电之前先将此 IO 拉低，会使 SDK 进入测试模式，执行相关的测试代码。根据测试的结果我们就可以知道该模块是否工作正常。

(2) TTCTEST_PWM_PIN

48M 测试输出口。

(3) TTCTEST_UART_RX

测试用的 UART 的 RX 脚。

(4) TTCTEST_UART_TX

测试用的 UART 的 TX 脚。

(5) TTCTEST_UART_WAKEUP

测试用的 UART 的 WAKEUP 脚。

测试过程中会通过 AT 指令来进行测试项目的控制。

6.2. 测试指令

6.2.1. 测试指令帧格式

命令格式：	DATA0	DATA1	DATA2	DATA 3	....	DATAn
	0X33	命令头	长度	数据		校验值

- 长度：为数据的总长，不包含 DATA0，DATA1，DATA2，校验值
- 校验值：为 DATA0+DATA1+...DATA(n-1)的总和

6.2.2. 测试命令

检查命令	33 FF 01 00 33
32K 测试	33 C0 01 00 F4
48M 测试	33 C1 01 00 F5
IO 测试	33 C2 01 00 F6
TX 测试	33 C3 01 00 F7
RX 测试	33 C4 01 00 F8
退出测试模式	33 C5 01 00 F9
读取 MAC 地址	33 AD 01 00 E1

版本号测试	33 F0 01 00 24
用户自定义测试	33 E0 01 00 14
用户数据读取	33 E1 01 00 15
用户数据写入	33 E2 LL PP XX XX XX CC(LL：长度 PP：参数 XX：数据 CC：校验)

6.2.3. 测试程序返回测试结果

回复检查命令	33 FF 01 01 34
32K 测试结果	33 C0 01 01 F5
48M 测试结果	33 C1 01 01 F6
IO 测试结果	33 C2 01 01 F7
TX 测试回复命令	33 C3 01 01 F8
RX 测试回复命令	33 C4 01 01 F9
退出测试模式回复命令	33 C5 01 01 FA
MAC 地址读取结果	33 AD 06 XX XX XX XX XX XX XX
版本号测试结果	33 F0 VH VL XX (版本号：0xVHVL，校验：XX)
用户自定义测试成功结果	33 E0 01 01 15
用户自定义测试失败结果	33 E0 02 XX XX XX
用户数据读取内容	33 E1 LL PP XX XX XX CC(LL：长度 PP：参数 XX：数据 CC：校验)
用户数据写入内容回应	33 E2 01 01 17
用户读写数据失败	33 E3 01 FF 16

6.2.4. 相关说明

（1）32K 测试

发送测试指令后，应拉高 WAKEUP 引脚，等待 500ms 后，若能接受到 32K 回复指令则说明正常

（2）48M 测试

发送测试指令后，检测 DIO-11 引脚，会输出周期为 20us 的 PWM 方波，占空比为 50%，持续 300ms 后会自动关闭 PWM

（3）IO 测试

发送测试指令后，立即开始检测所有引脚，除了 UART_TX/UART_RX/WAKEUP 引脚。相邻的两个脚电平不一样。100ms 后所有引脚电　平取反，再次检测。

（4）测试 TX

发送指令后，测试电流，此时拉高 WAKEUP 引脚后，电流应为 6.043mA 左右。测试完成后需要发送退出测试指令

（5）测试 RX

发送指令后，测试电流，此时拉高 WAKEUP 引脚后，电流应为 6.471mA 左右。测试完成后需要发送退出测试指令

（6）读取 MAC 地址

发送指令后会打开蓝牙广播，并返回 MAC 地址。只有该指令才能打开蓝牙广播

7. 联系我们

深圳市昇润科技有限公司
ShenZhen ShengRun Technology Co.,Ltd.
Tel: 0755-86233846 Fax: 0755-82970906
官网地址：www.tuner168.com
阿里巴巴网址：http://shop1439435278127.1688.com
E-mail: marketing@tuner168.com
地址： 广东省深圳市南山区西丽镇龙珠四路金谷创业园 B 栋 6 楼 601-602

附录 A. 手机 APP 下载

1. 手机 APP(TTC-BLE)下载二维码

附录 B. GPIO 分组

1.CC2650DK_4ID

IO 口分组	分组后 IO 口名称	GPIO	备注
IO0	IO0_0	IOID_27	Board_UART_RX 已占用
	IO0_1	IOID_26	Board_UART_TX 已占用
	IO0_2	IOID_25	Board_UART_WAKEUP 已占用
	IO0_3	IOID_24	Board_UART_INT 已占用
	IO0_4	IOID_23	
	IO0_5	IOID_17	
	IO0_6	IOID_16	
	IO0_7	IOID_7	
IO1	IO1_0	IOID_6	
	IO1_1	IOID_5	

2.CC2650EM_5XD

IO 口分组	分组后 IO 口名称	GPIO	备注
IO2	IO2_2	IOID_0	
	IO2_1	IOID_1	
	IO2_0	IOID_2	
IO1	IO1_6	IOID_3	
	IO1_5	IOID_4	
	IO1_4	IOID_5	
	IO1_3	IOID_6	
	IO1_2	IOID_7	
	IO1_1	IOID_8	
	IO1_0	IOID_9	
IO0	IO0_4	IOID_10	Board_UART_INT 已占用
	IO0_3	IOID_11	
	IO0_2	IOID_12	Board_UART_WAKEUP 已占用
	IO0_1	IOID_14	Board_UART_RX 已占用
	IO0_0	IOID_13	Board_UART_TX 已占用

3. CC2650DK_6ID、CC2650EM_7ID 两种封装引脚分组一致

IO 口分组	分组后 IO 口名称	GPIO	备注
IO0	IO0_0	IOID_30	
	IO0_1	IOID_29	
	IO0_2	IOID_28	
	IO0_3	IOID_27	
	IO0_4	IOID_26	
	IO0_5	IOID_25	
	IO0_6	IOID_24	
	IO0_7	IOID_23	
IO1	IO1_0	IOID_22	Board_UART_RX 已占用
	IO1_1	IOID_21	Board_UART_TX 已占用
	IO1_2	IOID_20	Board_UART_WAKEUP 已占用
	IO1_3	IOID_19	
	IO1_4	IOID_18	Board_UART_INT 已占用
	IO1_5	IOID_17	
	IO1_6	IOID_16	
	IO1_7	IOID_15	
IO2	IO2_0	IOID_14	
	IO2_1	IOID_13	
	IO2_2	IOID_12	
	IO2_3	IOID_11	
	IO2_4	IOID_10	
	IO2_5	IOID_9	
	IO2_6	IOID_8	
	IO2_7	IOID_7	
IO3	IO3_0	IOID_6	
	IO3_1	IOID_5	
	IO3_2	IOID_4	
	IO3_3	IOID_3	
	IO3_4	IOID_2	
	IO3_5	IOID_1	
	IO3_6	IOID_0	