BLE 协议栈 UART 调试指南

Ghostyu

2013-05

版本

V1.0	2013-05	初始版发布

目的

很多时候我们会用到芯片的 uart 连接一些外围设备，CC2540 的 UART 基础代码我们已测试 OK 并放在源码的源码目录下，但这远远不能达到我们的要求：如何在 TI 的 ble 协议栈中使用 uart 功能，例如开发蓝牙串口透传的程序等等，本来就是针对协议栈中使用 UART 过程中遇到的问题的记录。

阅读本文档前，请先阅读下列文档

OSAL 编程指南

1 概述

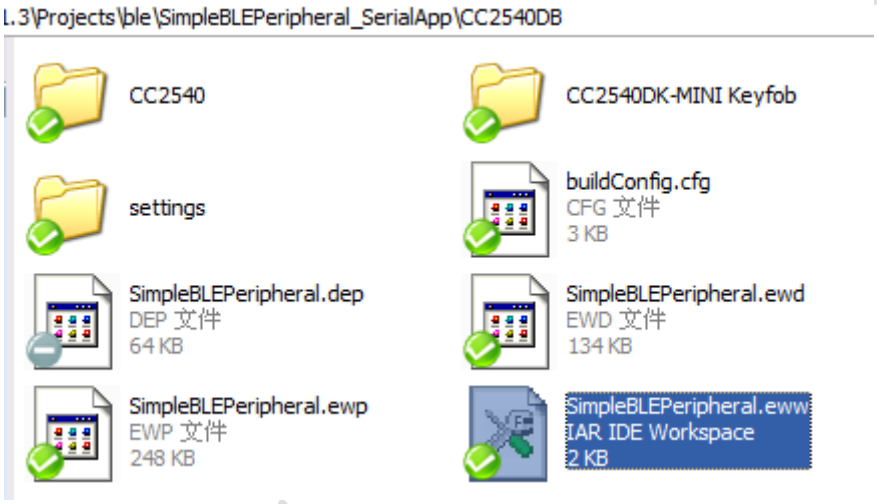
TI BLE 协议栈中已经做了 UART 底层驱动，因此并不需要我们重头编写 UART 的驱动代码，而是直接调用 hal_uart.c 中的 api 函数。该驱动源文件在如下目录：

\BLE-CC254x-1.3\Components\hal\target\CC2540EB\hal_uart.c

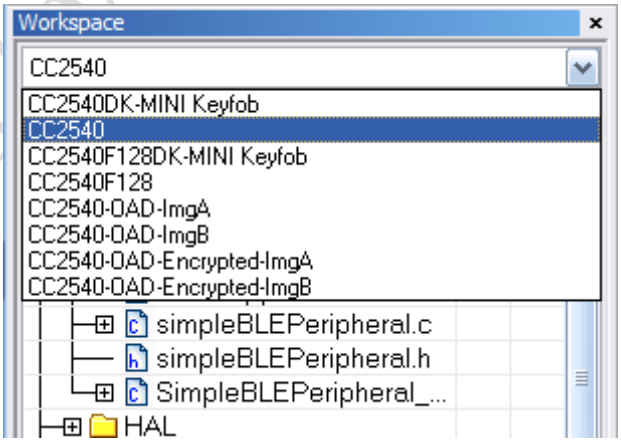
注意红色字体，我们的开发板平台是 SmartRF-CC2540，TI 还有其他的平台，如果 USBDongle-CC2540、SensorTag-CC2541、Keyfob-CC2540，尤其是在打开 IAR 工程后，请注意区分这几个配置。

2 新建带有 UART 的 SimpleBLEPeripheral

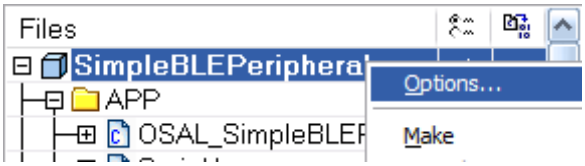
2.1 我们在 SimpleBLEPeripheral 基础上添加串口功能，为了不影响原先代码，复制 SimpleBLEPeripheral 文件夹为 SimpleBLEPeripheral_SerialApp，然后打开 SimpleBLEPeripheral_SerialApp 的 IAR 工程



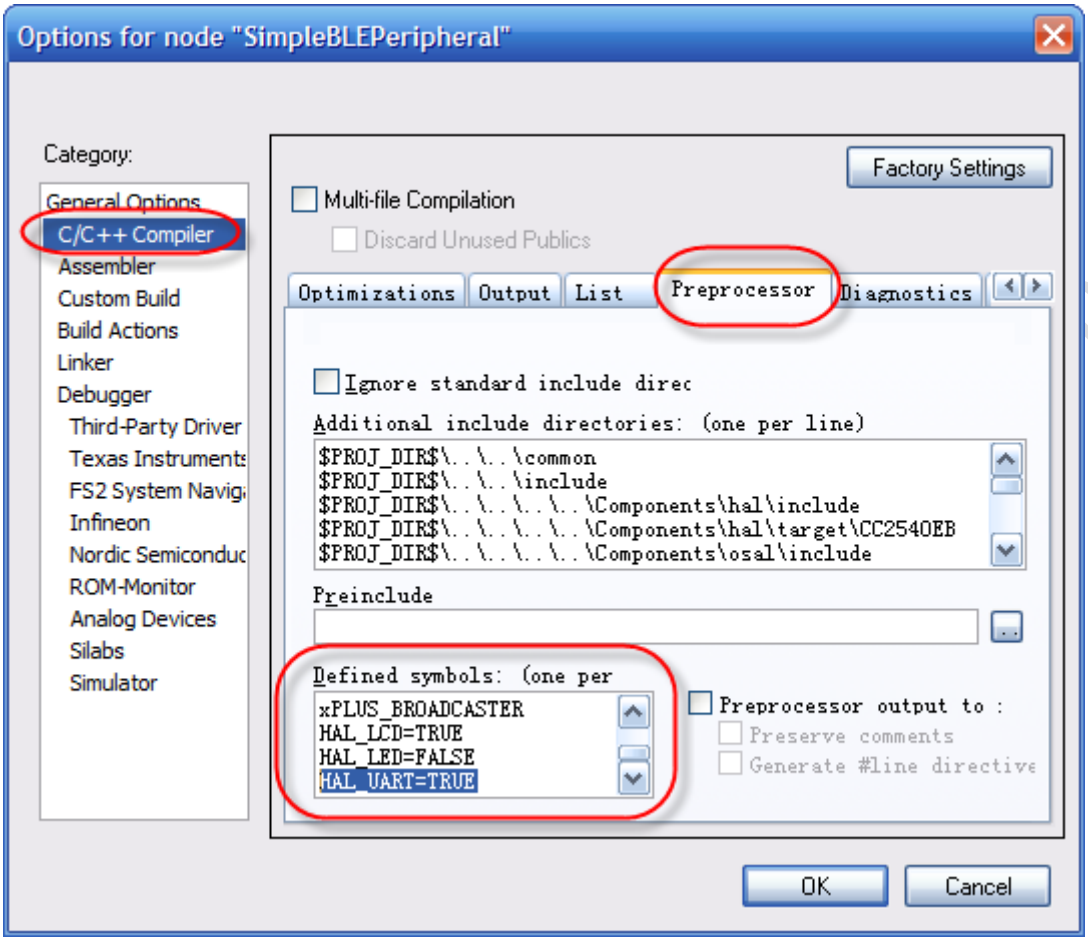
2.2 然后在 IAR 的 Workspace 中选择合适的 Configuration（在概述中提到的开发平台），这里我们选择 CC2540，注意其他的配置。在其他的 IAR 工程中类似。



2.3 打开该配置的 Option 属性，右击 SimpleBLEPeripheral，在出现的对话框中选择左边的 C/C++Compiler 选项。



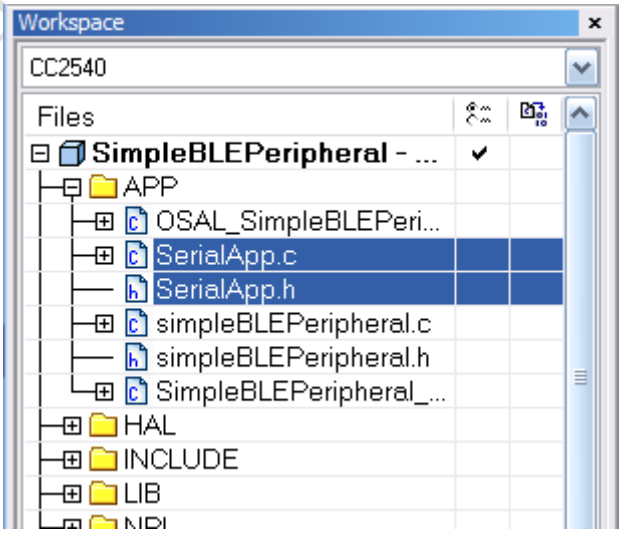
然后再单击右边的 Preprocessor 选项卡，并在 Defined symbols 中添加 HAL_UART=TRUE，如下图：



虽然协议栈代码中是包含 UART 驱动源码的，但是由宏定义 HAL_UART 来决定是否开启 UART，这里提供一个预处理的宏定义，这样 IAR 工程就会包含 UART，在配合自己编写的 UART 初始化代码就可以在协议栈中使用 UART 功能了。

2.3 添加 UART 初始化代码

在 SimpleBLEPeripheral 的 APP 目录下创建两个文件：SerialApp.c 和 SerialApp.h，我们将 UART 的相关配置放到这两个文件里。



SerialApp.h 截图如下：

```
#define SBP_UART_PORT                 HAL_UART_PORT_0
//#define SBP_UART_FC                   TRUE
#define SBP_UART_FC                   FALSE
#define SBP_UART_FC_THRESHOLD         48
#define SBP_UART_RX_BUF_SIZE          128
#define SBP_UART_TX_BUF_SIZE          128
#define SBP_UART_IDLE_TIMEOUT         6
#define SBP_UART_INT_ENABLE           TRUE
#define SBP_UART_BR                   HAL_UART_BR_57600

// Serial Port Related
extern void SerialApp_Init(uint8 taskID);
extern void sbpSerialAppCallback(uint8 port, uint8 event);
void serialAppInitTransport();
```

SerialApp.c 中有三个非常重要的函数，首先是 uart 配置：

```
/*uart初始化代码，配置串口的波特率、流控制等*/
void serialAppInitTransport( )
{
  halUARTCfg_t uartConfig;

  // configure UART
  uartConfig.configured           = TRUE;
  uartConfig.baudRate             = SBP_UART_BR;//波特率
  uartConfig.flowControl          = SBP_UART_FC;//流控制
  uartConfig.flowControlThreshold = SBP_UART_FC_THRESHOLD;//流控制阈值，当开启flowControl时，该设置有效
  uartConfig.rx.maxBufSize        = SBP_UART_RX_BUF_SIZE;//uart接收缓冲区大小
  uartConfig.tx.maxBufSize        = SBP_UART_TX_BUF_SIZE;//uart发送缓冲区大小
  uartConfig.idleTimeout          = SBP_UART_IDLE_TIMEOUT;
  uartConfig.intEnable            = SBP_UART_INT_ENABLE;//是否开启中断
  uartConfig.callBackFunc         = sbpSerialAppCallback;//uart接收回调函数，在该函数中读取可用uart数据

  // start UART
  // Note: Assumes no issue opening UART port.
  (void)HalUARTOpen( SBP_UART_PORT, &uartConfig );

  return;
}
```

另一个是 uart 回调函数，在回调函数中接收可用的串口数据

```
/*uart接收回调函数*/
void sbpSerialAppCallback(uint8 port, uint8 event)
{
  uint8  pktBuffer[SBP_UART_RX_BUF_SIZE];
  // unused input parameter; PC-Lint error 715.
  (void)event;
  HalLcdWriteString("Data form my UART:", HAL_LCD_LINE_4 );
  //返回可读的字节
  if ( (numBytes = Hal_UART_RxBufLen(port)) > 0 ){
    //读取全部有效的数据，这里可以一个一个读取，以解析特定的命令
    (void)HalUARTRead (port, pktBuffer, numBytes);
    HalLcdWriteString(pktBuffer, HAL_LCD_LINE_5 );
  }

}
```

最后是供外部调用的初始化接口函数：

```
/*该函数将会在任务函数的初始化函数中调用*/
void SerialApp_Init( uint8 taskID )
{
  //调用uart初始化代码
  serialAppInitTransport();
  //记录任务函数的taskID，备用
  sendMsgTo_TaskID = taskID;
}
```

2.4 将代码集成到 BLE 协议栈任务函数中

在任务函数中调用刚才提到的初始化接口函数即可；

```
void SimpleBLEPeripheral_Init( uint8 task_id )
{
  simpleBLEPeripheral_TaskID = task_id;
  //serial port initialization
  SerialApp_Init(task_id);

  // Setup the GAP Peripheral Role Profile
  {
```

2.5 编译 IAR 工程并测试

使用直连串口线连接开发板与 PC，在串口调试助手里发送任务的字符串，将能够在开发板的 LCD 显示屏上显示，如下图

对应的代码如下图：

```
/*uart接收回调函数*/
void sbpSerialAppCallback(uint8 port, uint8 event)
{
  uint8  pktBuffer[SBP_UART_RX_BUF_SIZE];
  // unused input parameter; PC-Lint error 715.
  (void)event;
  HalLcdWriteString("Data form my UART:", HAL_LCD_LINE_4 );
  //返回可读的字节
  if ( (numBytes = Hal_UART_RxBufLen(port)) > 0 ){
    //读取全部有效的数据，这里可以一个一个读取，以解析特定的命令
    (void)HalUARTRead (port, pktBuffer, numBytes);
    HalLcdWriteString(pktBuffer, HAL_LCD_LINE_5 );
  }

}
```

2.6 调试中的问题

在得到上面的结果并不是很顺利，主要遇到了下面的两个比较大问题

2.6.1 回调函数中接收的数据不完整

单步调试时回调函数会被调用两次，第一次只能接收一个字符，第二次会接收剩余的字符，也就是在上图的 sbpSerialAppCallback()函数中，第一次 Hal_UART_RxBufLen()函数只返回 1 或者 2，不能完整的返回整个字符串的长度，最后查到的问题是回调函数调用后需等待一小段时间在调用 Hal_UART_RxBufLen 函数，这样就能一次性读到完整的数据。在上图的回调函数中，有一条不起眼的语句：

```
HalLcdWriteString("Data form my UART:", HAL_LCD_LINE_4 );
```

就已经达到了延时的作用。

2.6.2 正常运行时，调试助手发送的数据，cc2540 不能及时接收到，有数据丢失，并且会隔很久。

这个问题是由于开启了 POWER_SAVING，我临时的解决办法取消了 POWER_SAVING 宏定义预处理，如下图：