GHOSTYU 谷雨

配对与绑定

基于 SimpleBLEPeripheral 的配对绑定实验

Ghostyu.com

2014/1/27

[在此处键入文档的摘要。摘要通常是对文档内容的简短总结。在此处键入文档的摘要。摘要通常是对文档内容的简短总结。]

目录

1 前言

在 ble 实际应用中，需要密码来保证设备的安全性，限制非法连接，该 demo 中，基于 SimpleBLEPeripheral，然后添加了密码配对与绑定。当主机连接该 demo 时，需要提供 SimpleBLEPeripheral 在 lcd 上显示的密码，然后才能建立连接。否则会被拒绝连接。

2 必要条件

A 硬件

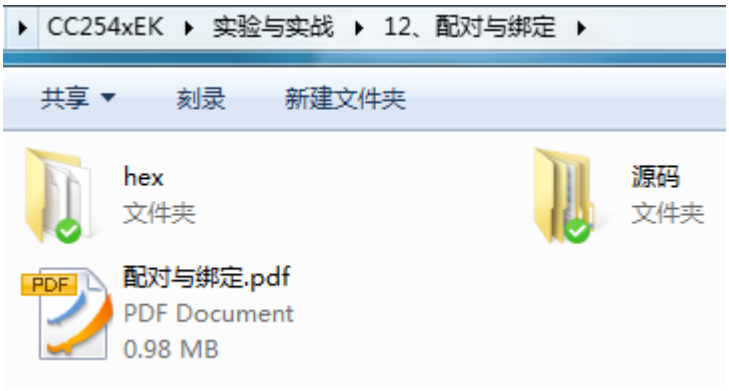
1、 SmartRF 系列开发板，CC2540 或者 CC2541
2、 CC-Debugger 仿真器
3、 iPhone4s 以上（Android4.3 以上的系统也可以安装 ble 的应用来作为主机）

B 软件

1、 ble 协议栈，版本：1.3.2
2、 IAR for 8051 开发环境，版本：8.10
3、 Flash Programmer 固件烧写软件。

3 文件预览

本文档的所有相关源码、说明均位于【CC254xEK\实验与实战\12、配对与绑定】目录下，如下图：



【Hex】文件夹存放我们预先编译 OK 的固件，可以直接下载到 SmartRF 系列开发板中测试运行。

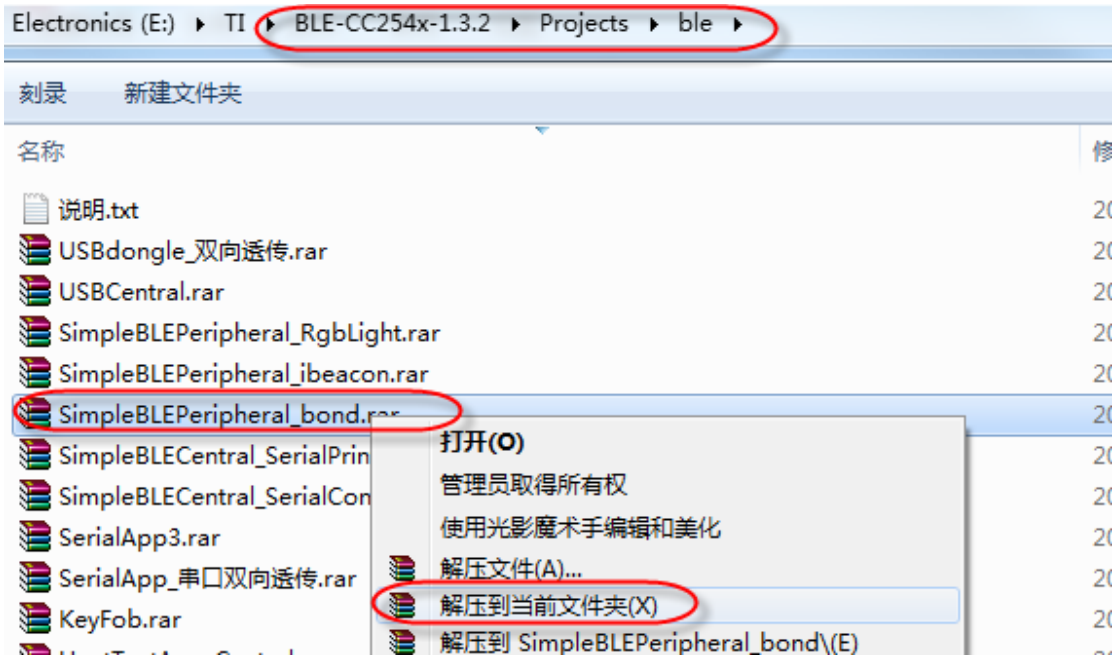
【源码】文件夹存放的是该实践相关的源码程序

【配对与绑定.pdf】也就是本文档，在进行任何操作前请务必先仔细阅读。

4 源码包解压

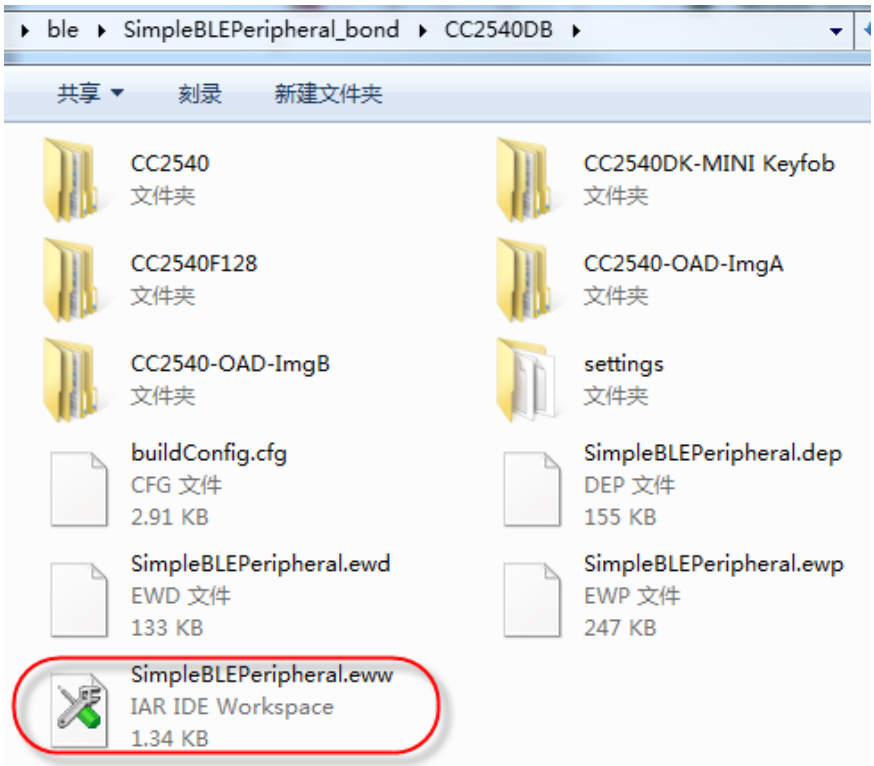
将【\实验与实战\5、配对与绑定.pdf\源码\CC254x】下的压缩包，**复制到 1.3.2 版本的协议栈 projects 目录下，然后右击选择“解压到当前文件夹”，**如下图所示，务必注意，请

勿“解压到 xxx”，否则会多一级目录，造成源码编译不通过。

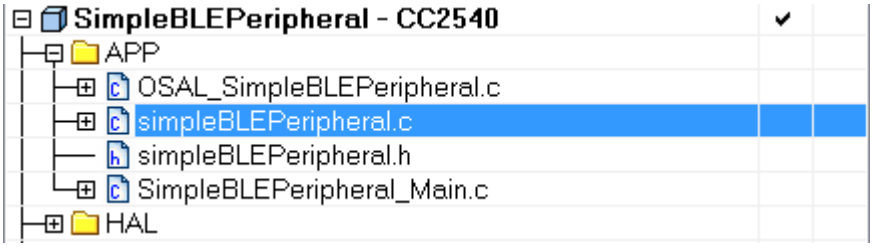


5 打开 IAR 工程

接下来我们打开 SimpleBLEPeripheral_bond 工程，进入【BLE-CC254x-1.3.2\Projects\ble\SimpleBLEPeripheral_bond\CC2540】，打开 IAR 工程，如下图，如果你使用的芯片是 CC2541，进入 CC2541 文件夹打开工程。



全部代码，均在 simpleblePeripheral.c 文件中修改。



首先打开 simpleblePeripheral.c 源文件，第 230 行到 240 行，主要是 ProcessPasscodeCB 和 ProcessPairStateCB 两个回调函数的定义。密码配对与访问均在这两个函数中实现。

```
00229: // Application states
00230: enum
00231: {
00232:   BLE_STATE_IDLE,
00233:   BLE_STATE_CONNECTED,
00234: };
00235: // Application state
00236: static uint8 simpleBLEState = BLE_STATE_IDLE;
00237: //ghostyu bond
00238: static uint8 gPairStatus=0;/*用来管理当前的状态，如果密码不正确，立即取消连接，0表示未配对，1表示已配对*/
00239: void ProcessPasscodeCB(uint8 *deviceAddr,uint16 connectionHandle,uint8 uiInputs,uint8 uiOutputs );
00240: static void ProcessPairStateCB( uint16 connHandle, uint8 state, uint8 status );
```

然后需要绑定管理器的配置，如下图：

需要注意的是第 356 行，需要将 pairMode 设置为 GAPBOND_PAIRING_MODE_INITIATE，才能在连接的时候产生配对请求。另外还需要注意 358 行，当两个手机之间配对时，会在其中一个手机上设置一个密码，然后另外一个手机输入这个密码，才能完成建立，在这里由于 SmartRF 开发板上没有密码输入的功能，所以需要将 ioCap 设置为：GAPBOND_IO_CAP_DISPLAY_ONLY，表示只有显示密码的能力。

```
00352:   // Setup the GAP Bond Manager
00353:   {
00354:     uint32 passkey = 0; // passkey "000000"
00355:     //uint8 pairMode = GAPBOND_PAIRING_MODE_WAIT_FOR_REQ;
00356:     uint8 pairMode = GAPBOND_PAIRING_MODE_INITIATE;
00357:     uint8 mitm = TRUE;
00358:     uint8 ioCap = GAPBOND_IO_CAP_DISPLAY_ONLY;
00359:     uint8 bonding = TRUE;
00360:     GAPBondMgr_SetParameter( GAPBOND_DEFAULT_PASSCODE, sizeof ( uint32 ), &passkey );
00361:     GAPBondMgr_SetParameter( GAPBOND_PAIRING_MODE, sizeof ( uint8 ), &pairMode );
00362:     GAPBondMgr_SetParameter( GAPBOND_MITM_PROTECTION, sizeof ( uint8 ), &mitm );
00363:     GAPBondMgr_SetParameter( GAPBOND_IO_CAPABILITIES, sizeof ( uint8 ), &ioCap );
00364:     GAPBondMgr_SetParameter( GAPBOND_BONDING_ENABLED, sizeof ( uint8 ), &bonding );
00365:   }
```

最后定义上面提到的两个回调函数。

配对密码回调函数。

第 838 行，密码使用 32 为的无符号长整形表示，可以给这个变量直接复制“123456”，密码就为 123456。843 行：由于 SmartRF 没有密码设定能力，所以使用随机产生一个密码。大家也可以自定存储一个密码，然后手机可以动态修改该密码，第 851 行，在 lcd 上显示随机产生的密码。第 855 行，发送密码相应。

```
835: //绑定过程中的密码管理回调函数
836: static void ProcessPasscodeCB(uint8 *deviceAddr,uint16 connectionHandle,uint8 uiInputs,uint8 uiOutputs )
837: {
838:   uint32  passcode;
839:   uint8   str[7];
840:
841:   //在这里可以设置存储，保存之前设定的密码，这样就可以动态修改配对密码了。
842:   // Create random passcode
843:   LL_Rand( ((uint8 *) &passcode), sizeof( uint32 ));
844:   passcode %= 1000000;
845:
846:   //在lcd上显示当前的密码，这样手机端，根据此密码连接。
847:   // Display passcode to user
848:   if ( uiOutputs != 0 )
849:   {
850:     HalLcdWriteString( "Passcode:",  HAL_LCD_LINE_1 );
851:     HalLcdWriteString( (char *) _ltoa(passcode, str, 10),  HAL_LCD_LINE_2 );
852:   }
853:
854:   // Send passcode response
855:   GAPBondMgr_PasscodeRsp( connectionHandle, SUCCESS, passcode );
856: } ? end ProcessPasscodeCB ?
```

配对状态回调函数。

第 861 行，主机开始连接从机，进入配对开始状态。第 866 行，密码配对完成，然后判断返回的状态，如果等于 SUCCESS，也就是 0，表示密码配对成功，否则，密码不正确，或者已经配对。第 883 行，如果配对失败，立即中断连接。

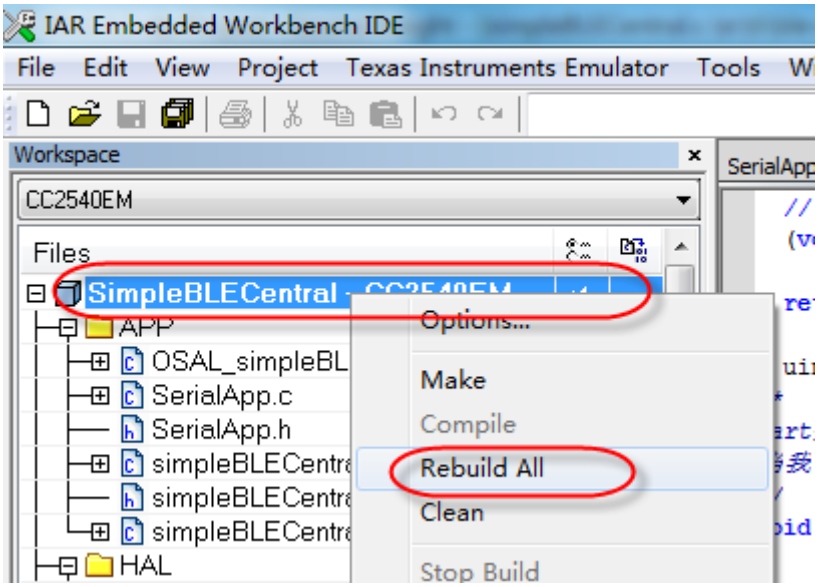
```
00858: //绑定过程中的状态管理，在这里可以设置标志位，当密码不正确时不允许连接。
00859: static void ProcessPairStateCB( uint16 connHandle, uint8 state, uint8 status )
00860: {
00861:   if ( state == GAPBOND_PAIRING_STATE_STARTED )/*主机发起连接，会进入开始绑定状态*/
00862:   {
00863:     HalLcdWriteString( "Pairing started", HAL_LCD_LINE_1 );
00864:     gPairStatus = 0;
00865:   }
00866:   else if ( state == GAPBOND_PAIRING_STATE_COMPLETE )/*当主机提交密码后，会进入完成*/
00867:   {
00868:     if ( status == SUCCESS )
00869:     {
00870:       HalLcdWriteString( "Pairing success", HAL_LCD_LINE_1 );/*密码正确*/
00871:       gPairStatus = 1;
00872:     }
00873:     else
00874:     {
00875:       HalLcdWriteStringValue( "Pairing fail", status, 10, HAL_LCD_LINE_1 );/*密码不正确，或者先前已经绑定*/
00876:       if(status ==8){/*已绑定*/
00877:         gPairStatus = 1;
00878:       }else{
00879:         gPairStatus = 0;
00880:       }
00881:     }
00882:     //判断配对结果，如果不正确立刻停止连接。
00883:     if(simpleBLEState == BLE_STATE_CONNECTED && gPairStatus !=1){
00884:       GAPRole_TerminateConnection();
00885:     }
00886:   } ? end if state==GAPBOND_PAIRIN... ?
00887:   else if ( state == GAPBOND_PAIRING_STATE_BONDED )
00888:   {
00889:     if ( status == SUCCESS )
00890:     {
00891:       HalLcdWriteString( "Bonding success", HAL_LCD_LINE_1 );
00892:     }
00893:   }
00894:
00895: } ? end ProcessPairStateCB ?
```

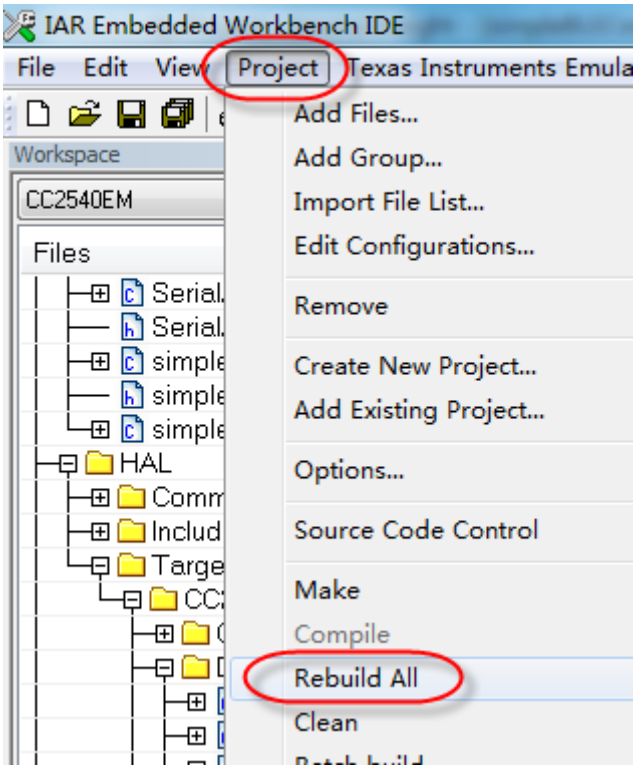
这里返回的状态值，都有详细的解释。详情见： gapbondmgr.h 中有定义，位于协议栈：BLE-CC254x-1.3.2\Projects\ble\Profiles\Roles

```
00148: /** @defgroup SMP_PAIRING_FAILED_DEFINES Pairing failure status values
00149:  * @{
00150:  */
00151: #define SMP_PAIRING_FAILED_PASSKEY_ENTRY_FAILED   0x01 //!< The user input of the passkey failed, for example, the user cancelled
00152: #define SMP_PAIRING_FAILED_OOB_NOT_AVAIL          0x02 //!< The OOB data is not available
00153: #define SMP_PAIRING_FAILED_AUTH_REQ               0x03 //!< The pairing procedure can't be performed as authentication requirement
00154: #define SMP_PAIRING_FAILED_CONFIRM_VALUE          0x04 //!< The confirm value doesn't match the calculated compare value
00155: #define SMP_PAIRING_FAILED_NOT_SUPPORTED          0x05 //!< Pairing isn't supported by the device
00156: #define SMP_PAIRING_FAILED_ENC_KEY_SIZE           0x06 //!< The resultant encryption key size is insufficient for the security req
00157: #define SMP_PAIRING_FAILED_CMD_NOT_SUPPORTED      0x07 //!< The SMP command received is not supported on this device.
00158: #define SMP_PAIRING_FAILED_UNSPECIFIED            0x08 //!< Pairing failed due to an unspecified reason
00159: #define SMP_PAIRING_FAILED_REPEATED_ATTEMPTS      0x09 //!< Pairing or authenication procedure is disallowed because too little ti
```


6 编译下载

在当前 Configuration 上右击，然后选择 Rebuild All，重新编译整个工程。或者选择菜单 Project/Rebuild All。效果相同。





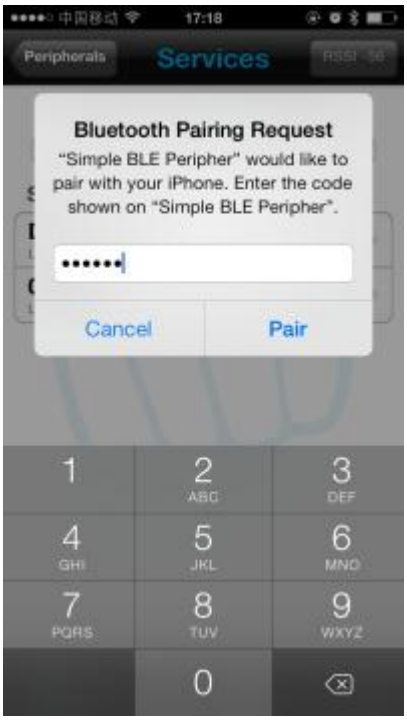
如果源码解压的位置正确，并且使用的是 1.3.2 的 ble 协议栈和 8.10 的 IAR 编译器，不会出现任何编译问题。

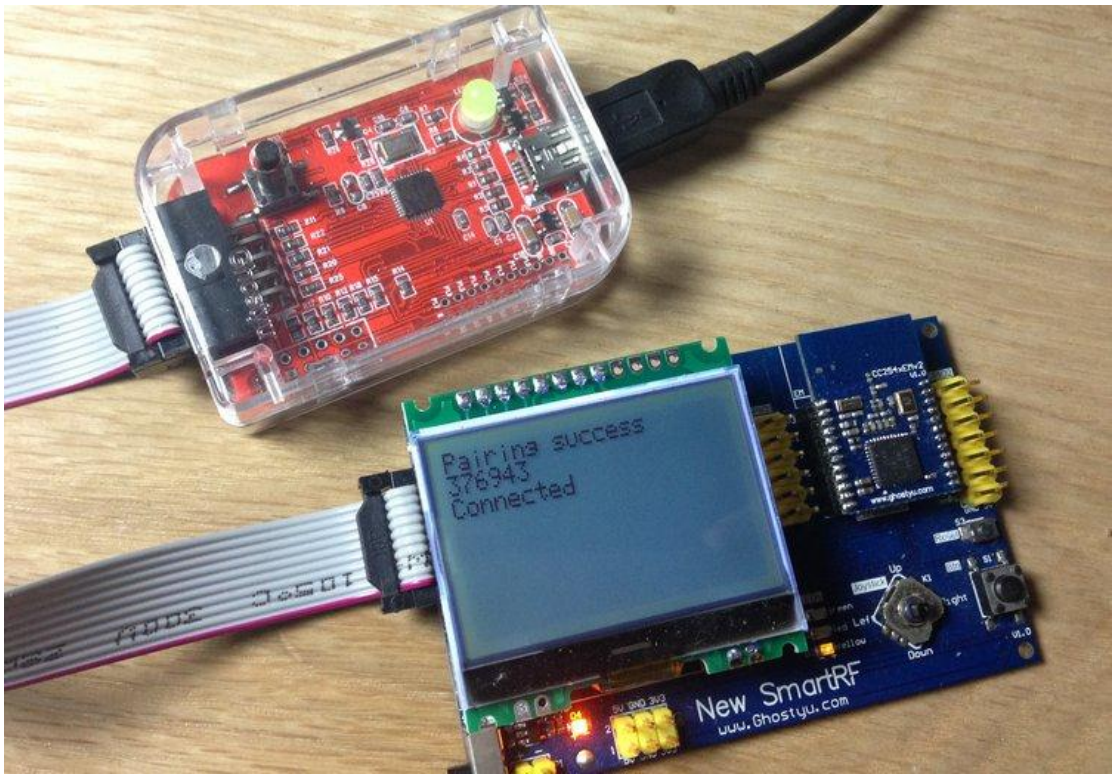
连接 CC-Debugger 仿真器和开发板，准备烧写程序，注意，仿真器连接 OK 后务必按仿真器复位按钮，待 CC-Debugger 指示灯变绿后，再进行下一步操作。另外还需要连接开发板的与 PC 的串口，使用 IAR 下载，或者使用 Flash Programmer 将程序下载到开发板中。

7 测试

如上图所示，连接已烧写了 SimpleBLEPeripheral_Bond 程序的开发板，并且供电。然后，在 iphone 上运行 lightblue 程序。

1、 lightblue 搜索从机，然后点击连接，会跳出输入密码配对的对话框，此时 SmartRF 上会显示当前的连接密码。在 lightblue 里输入密码然后点击 Pair。





2、 这样就成功配对了，simplePeripheral 里的 char5，就可以直接读取他的值。

大家故意输错密码测试。SmartRF 会主动断开连接。由于手机已经和 SmartRF 配对且绑定了，所以重新实验时，需要在手机上取消配对，这样再次连接时就仍需要输入密码。

联系我们：
刘雨 tel:15861666207
网站：http://www.ghostyu.com
技术支持：http://www.ghostyu.com/bbs
在线文档：http://www.ghostyu.com/wiki
官网店铺：http://ghostyu.taobao.com