Bluetooth® Low Energy CC2540 Mini Development Kit User's Guide

Document Number: SWRU270B

Document Version: 1.1

Development Kit Part Number: CC2540DK-MINI

TABLE OF CONTENTS

1. References

The following references provide additional information on the CC2540, the Texas Instruments *Bluetooth*® low energy (BLE) stack, and the BLE specification in general. (All path and file references in this document assume that the BLE development kit software has been installed to the default path C:\Texas Instruments\BLE-CC2540-1.1\)

1.1 Printed Copy Included in the Box with the Kit

[1] CC2540 Mini Development Kit Quick Start Guide (SWRU272)

1.2 Included with Texas Instruments *Bluetooth* Low Energy Software Installer

(The software installer is available for download at **http://www.ti.com/blestack**)

[2] Texas Instruments *Bluetooth*® Low Energy Software Developer's Guide (SWRU271A)
C:\Texas Instruments\BLE-CC2540-1.1\Documents\TI_BLE_Software_Developer's_Guide.pdf

[3] TI BLE Vendor Specific HCI Reference Guide
C:\Texas Instruments\BLE-CC2540-1.1\Documents\TI_BLE_Vendor_Specific_HCI_Guide.pdf

[4] Texas Instruments BLE Sample Applications Guide (SWRU297)
C:\Texas Instruments\BLE-CC2540-1.1\Documents\TI_BLE_Sample_Applications_Guide.pdf

1.3 Available from *Bluetooth* Special Interest Group (SIG)

[5] *Specification of the Bluetooth System*, Covered Core Package version: 4.0 (30-June-2010)
https://www.bluetooth.org/technical/specifications/adopted.htm

2. Introduction

Thank you for purchasing the Texas Instruments (TI) *Bluetooth*® low energy (BLE) CC2540 Mini Development Kit (CC2540DK-MINI). The purpose of this document is to give an overview of the hardware and software contained in the CC2540DK-MINI.

The information in this guide will get you up and running with the kit; however for more detailed information on BLE technology and the CC2540 BLE protocol stack, please consult [2].

2.1 Kit Contents Overview

The CC2540DK-MINI kit is composed of hardware and software, with details on both in the sections below.

2.1.1 Hardware

The kit contains the following hardware components:

1 CC2540 USB Dongle

This is the device that will be acting as the BLE Master. It connects to a Windows PC's USB port, and is pre-loaded with the master demo application software.

1 CC2540 "Keyfob"

This is the device that will be acting as the BLE Slave. The PCB sits inside a plastic case, and can be removed with a small Philips screwdriver. It operates on a single CR2032 coin cell battery, and contains a two-colored LED, a buzzer, an accelerometer, and two buttons.

1 CC Debugger with mini USB cable, converter board, and a 10-pin connector cables

This is used to flash the software onto both the USB dongle as well as the keyfob. It also can be used for debugging software using IAR Embedded Workbench.

Figure 1 – Hardware Included with CC2540DK-MINI

2.2 System Requirements

To use the CC2540 software, a PC running Microsoft Windows (XP or later; 32-bit support only) is required, as well as Microsoft .NET Framework 3.5 Service Pack 1 (SP1) or greater.

In order to check whether your system has the appropriate .NET Framework, open up the Windows Control Panel, and select "Add or Remove Programs". Amongst the list of currently installed programs, you should see "Microsoft .NET Framework 3.5 SP1", as such:

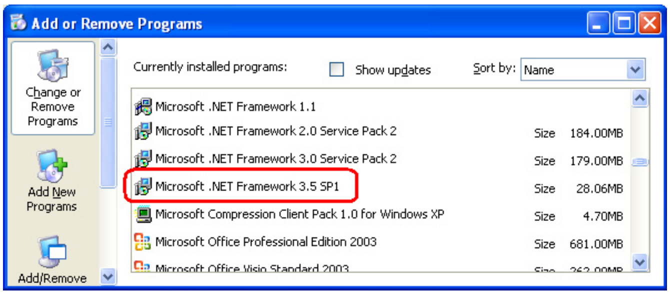

Figure 2

If you do not see it in the list, you can download the framework from Microsoft.

From a hardware standpoint, the Windows PC must contain one free USB port. An additional free USB port is required in order to use the CC Debugger and the USB Dongle simultaneously.

IAR Embedded Workbench for 8051 development environment is required in order to make changes to the keyfob software. More information on IAR can be found in [1].

For the keyfob, a small Philips screwdriver (not included in the kit) is required if you want to enclose the keyfob in the plastic case, and a CR2032 coin cell battery (included in the kit) is required for power.

3. Getting Started

This section describes how to set up the software and get started with the CC2540 Mini Development Kit. It is assumed that the instructions found in [1] (a printed copy of the quick start guide is included with the kit) have already been completed, with both the keyfob and the dongle having been programmed with the latest hex files. In addition, this section assumes that the latest version of the CC2540 BLE software (v1.1 as of the release of this document) has been installed. The latest BLE software can be downloaded at www.ti.com/blestack.

3.1 Associate Driver with USB Dongle

After the software installation is complete, the USB Dongle driver must be associated with the device in order to use the demo application. To associate the USB Dongle driver, first you must connect the USB Dongle to the PC's USB port, or to a USB hub that connects to the PC.

The first time that the dongle is connected to the PC, a message will pop-up, indicating that Windows does not recognize the device.

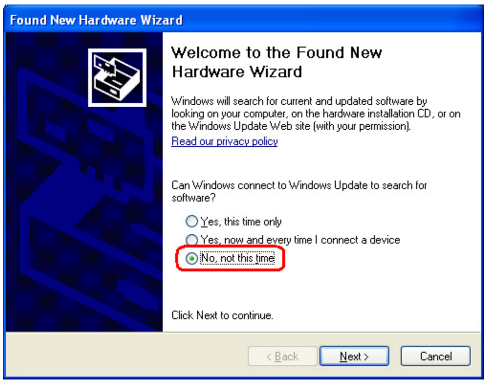

Figure 3

When prompted whether to use Windows Update search for software, select "No, not this time" and press the "Next" button. On the next screen, select the option "Install from a list or specific location (Advanced)", and press the "Next" button:

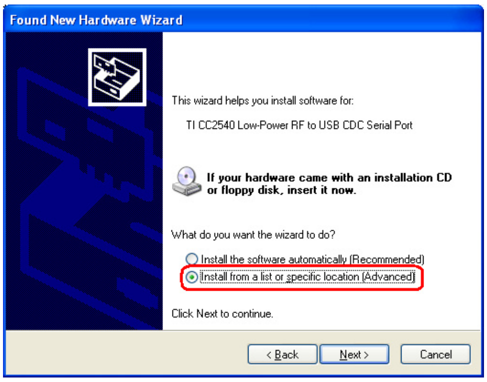

Figure 4

On the next screen, click the checkbox labeled "Include this location in the search:", and click the "Browse" button. Select the following directory (assuming the default installation path was used):

C:\Texas Instruments\BLE-CC2540-1.1\Accessories\Drivers

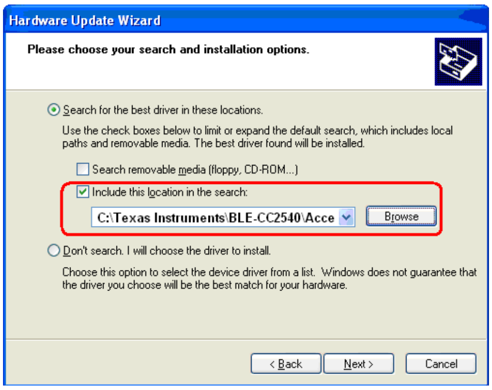

Figure 5

Click the "Next" button. This should install the driver. It will take a few seconds for the file to load. If the installation was successful, you should see the screen to the below. Click the "Finish" button to complete the installation.

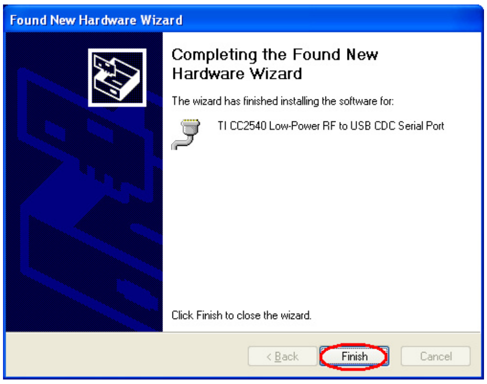

Figure 6

3.2 Determining the COM Port

Once the driver is installed, you need to determine which COM port Windows has assigned to the USB Dongle. After you have completed the USB Dongle driver association in section 3.1, right-click on the "My Computer" icon on your desktop and select "Properties":

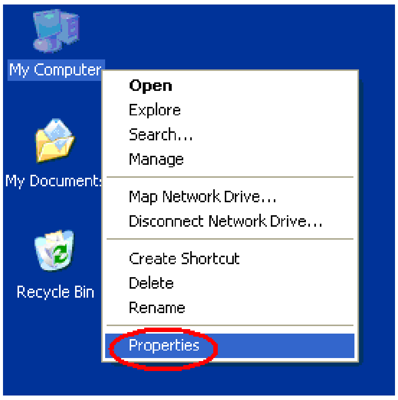

Figure 7

The "System Properties" window should open up. Under the "Hardware" tab, select "Device Manager":

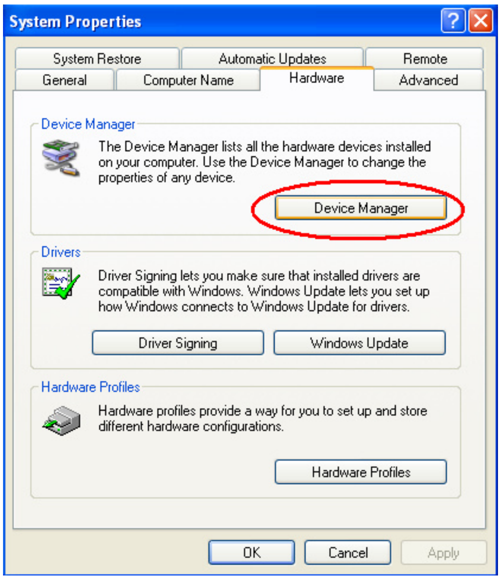

Figure 8

A list of all hardware devices should appear. Under the section "Ports (COM & LPT)", the device "TI CC2540 Low-Power RF to USB CDC Serial Port" should appear. Next to the name should be the port number (for example, COM15 in the image below):

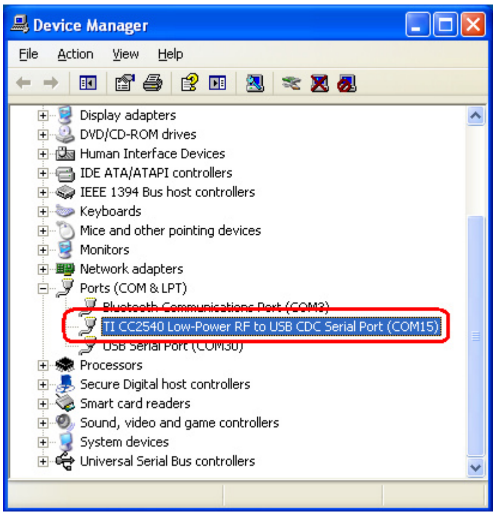

Figure 9

Take note of this port number, as it will be needed in order to use BTool. You may close the device manager at this point.

4. Using BTool

BTool is a PC Application that allows a user to form a connection between two BLE devices. BTool works by communicating with the CC2540 by means of HCI vendor specific commands. The USB Dongle software (when running the HostTestRelease project) and driver create a virtual serial port over the USB interface. BTool, running on the PC, communicates with the USB Dongle through this virtual serial port.

More information on the HCI interface, as well as details on the HCI vendor specific commands that are used by the CC2540, can be found in [3].

4.1 Starting the Application

To start the application go into your programs by choosing Start > Programs > Texas Instruments > Bluetooth-LE-1.1 > BTool. You should see the following window open up:

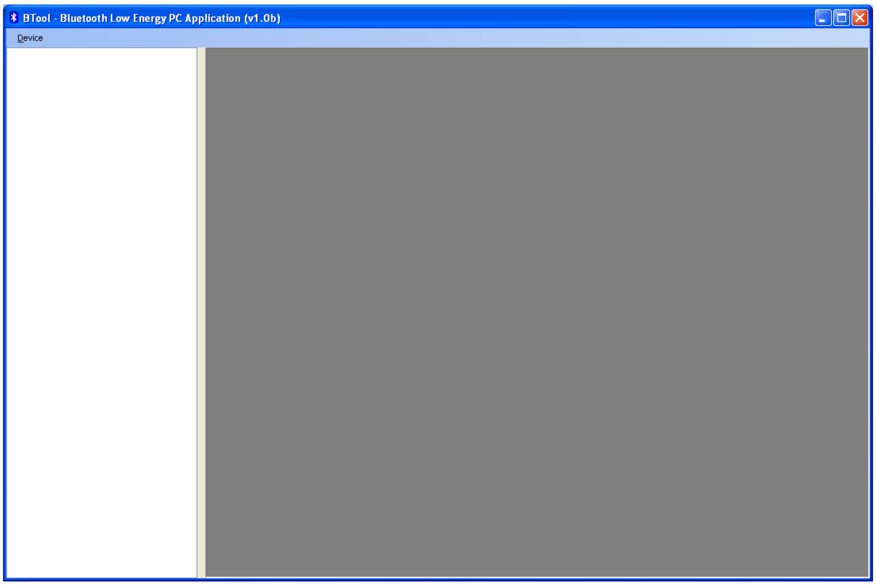

Figure 10

In the upper left corner of the window, click the “Device” drop menu and select the option “New Device”:

Figure 11

The following window should pop-up:

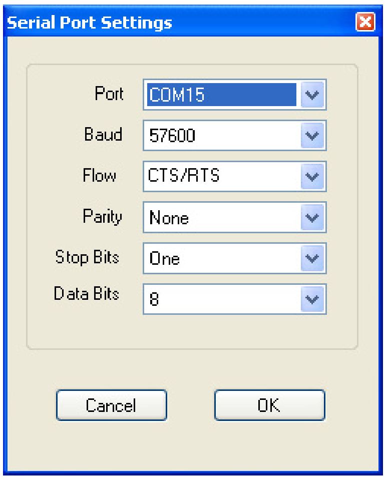

Figure 12

If using the USB Dongle, set the "Port" value to the COM port of the USB Dongle from step 1. For the other settings, use the default values as shown in Figure 12. Press "OK" to connect to the board. The following screen will appear:

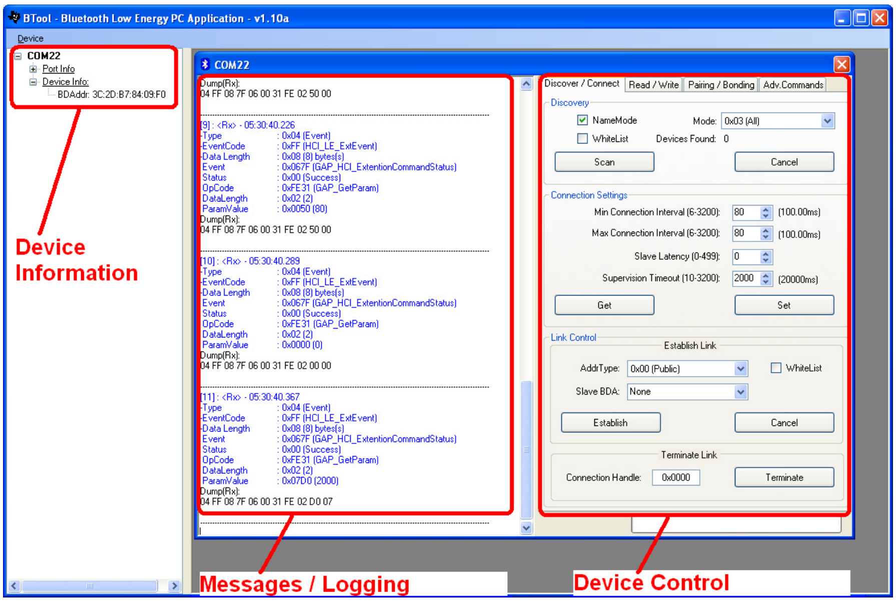

Figure 13

This screen indicates that you are now connected to the USB Dongle. The screen is divided up into a few sections: the left sidebar contains information on the USB Dongle's status. The left side of the sub-window contains a log of all messages sent from the PC to the dongle and received by the PC from the dongle. The right side of the sub-window contains a GUI for control of the USB Dongle.

4.2 Creating a BLE Connection between USB Dongle and Keyfob

At this point the USB Dongle (central) is ready to discover other BLE devices that are advertising. If you have followed the instructions in the [1], the keyfob should be preloaded with the SimpleBLEPeripheral application. The full project and application source code files for SimpleBLEPeripheral is included in the BLE software development kit. For more details on how the SimpleBLEPeripheral application works, please see the [2].

At this time you will want to insert the battery (or remove and re-insert the battery to reset the device) into the keyfob (peripheral).

4.2.1 Making the Keyfob Discoverable

When the keyfob powers up, it will not immediately go into a discoverable state. To enable advertising and make the keyfob discoverable, press the right-hand button on the keyfob once. This will turn advertisements on; making the device discoverable for 30 seconds (this value is defined in [5]). After that time, the device will return to standby mode. To make the device discoverable again, simply press the button once again.

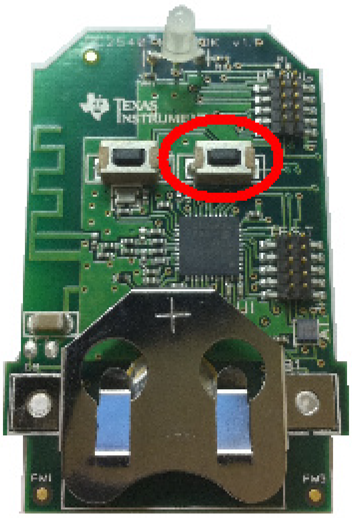

Figure 14

4.2.2 Scanning for Devices

Press the "Scan" button under the "Discover / Connect" tab:

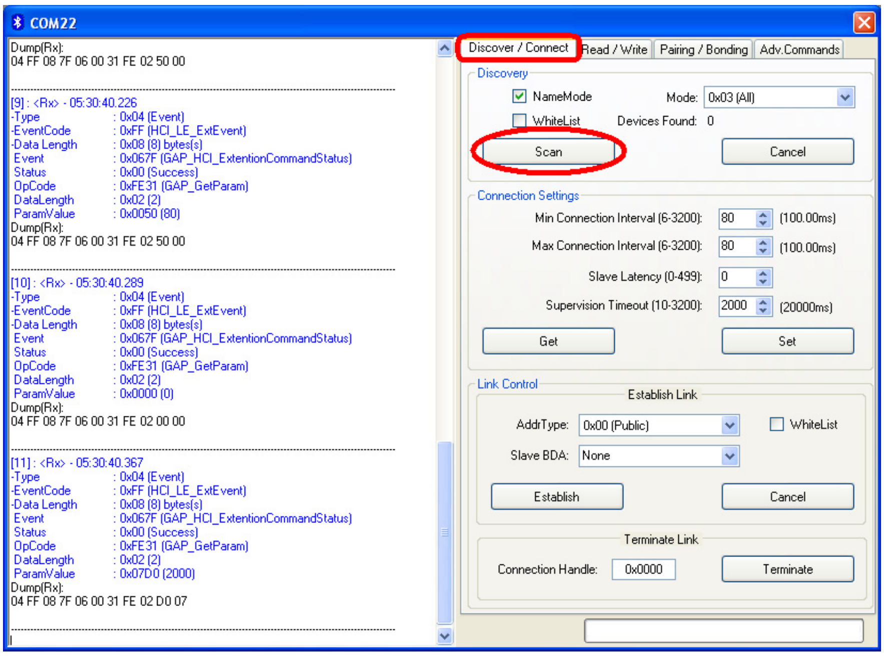

Figure 15

The USB Dongle will begin search for other BLE devices. As devices are found, the log on the left side of the screen will display the devices discovered. After 10 seconds, the device discovery process will complete, and the USB Dongle will stop scanning. A summary of all the scanned devices will be displayed in the log window. In the example in Figure 16, one peripheral device was discovered while scanning. If you do not want to wait through the full 10 seconds of scanning, the "Cancel" button can be pressed alternatively, which will stop the device discovery process. The address of any scanned devices will appear in the "Slave BDA" section of the "Link Control" section in the bottom right corner of the sub-window.

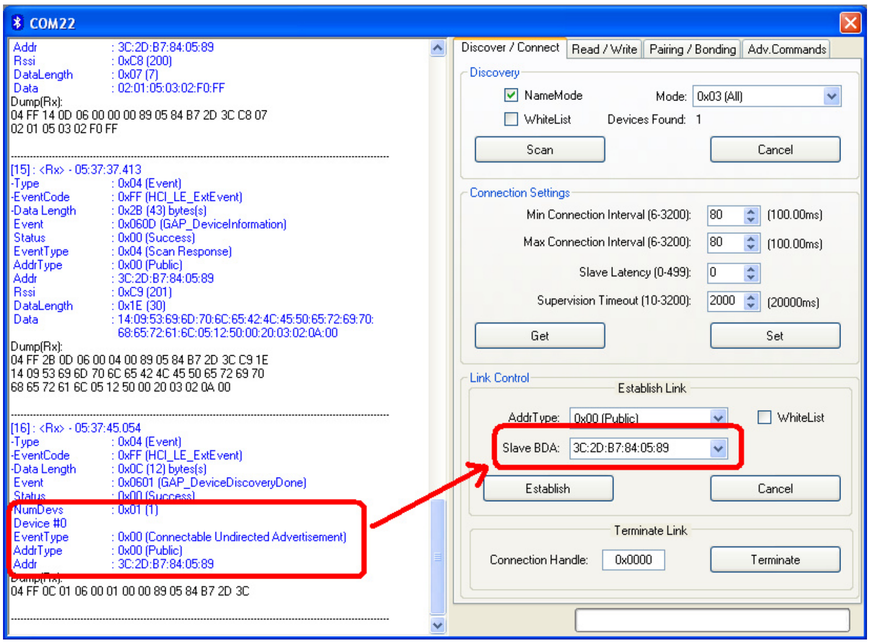

Figure 16

4.2.3 Selecting Connection Parameters

Before establishing a connection, you will want to set up the desired connection parameters. The default values of 100ms connection interval, 0 slave latency, and 20000ms supervision timeout should serve as a good starting point; however for different applications you may want to experiment with these values.

Once the desired values have been set, be sure to click the “Set” button; other wise the settings will not be saved. Note that the connection parameters must be set before a connection is established; changing the values and clicking the “Set” button while a connection is active will not change the settings of an active connection. The connection must be terminated and re-established to use the new parameters. (The *Bluetooth* specification does support connection parameter updates while a connection is active; however this must be done using either an L2CAP connection parameter update request, or using a direct HCI command. More information can be found in [5])

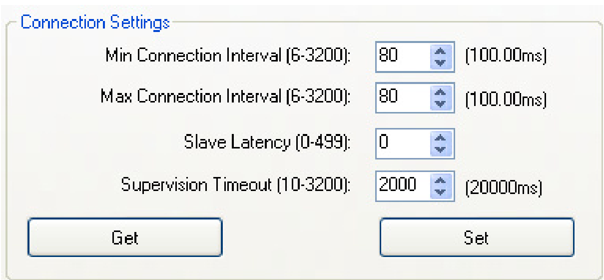

Figure 17

4.2.4 Establishing a Connection

To establish a connection with the keyfob, select the address of the device to connect with, and click the “Establish” button. If the set of connection parameters are invalid (for example, if the combination of connection parameters violates the specification), the message window will return a “GAP_EstablishLink” event message with a “Status” value of “0x12 (Not setup properly to perform that task)”, as shown in Figure 18. The parameters will have to be corrected before a connection can be established.

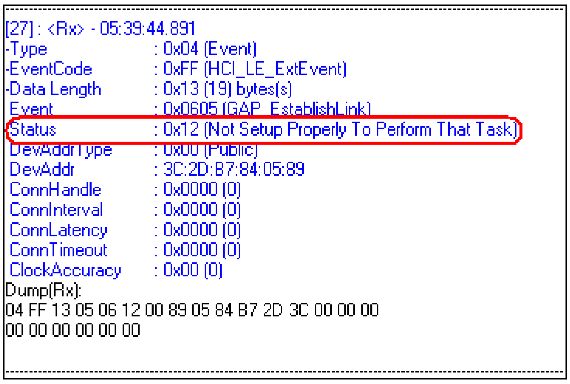
[27] : <Rx> - 05:39:44.891
-Type : 0x04 (Event)
-EventCode : 0xFF (HCI_LE_ExtEvent)
-Data Length : 0x13 (19) bytes(s)
Event : 0x0605 (GAP_EstablishLink)
Status : 0x12 (Not Setup Properly To Perform That Task)
DevAddrType : 0x00 (Public)
DevAddr : 3C:2D:B7:84:05:89
ConnHandle : 0x0000 (0)
ConnInterval : 0x0000 (0)
ConnLatency : 0x0000 (0)
ConnTimeout : 0x0000 (0)
ClockAccuracy : 0x00 (0)
Dump(Rx):
04 FF 13 05 06 12 00 89 05 84 B7 2D 3C 00 00 00
00 00 00 00 00 00

Figure 18

If the keyfob is still in discoverable mode, a connection should be established (press the right button on the keyfob once again if the device if more than 30 seconds have passed since the device was previously made discoverable and the process has completed). Once a connection is established, the message window will return a "GAP_EstablishLink" event message with a "Status" value of "0x00 (Success)":

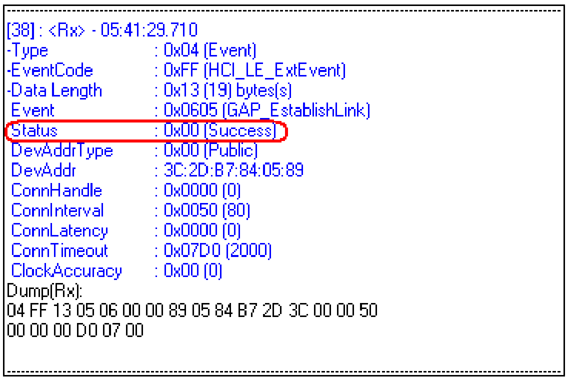
[38] : <Rx> - 05:41:29.710
-Type : 0x04 (Event)
-EventCode : 0xFF (HCI_LE_ExtEvent)
-Data Length : 0x13 (19) bytes(s)
Event : 0x0605 (GAP_EstablishLink)
Status : 0x00 (Success)
DevAddrType : 0x00 (Public)
DevAddr : 3C:2D:B7:84:05:89
ConnHandle : 0x0000 (0)
ConnInterval : 0x0050 (80)
ConnLatency : 0x0000 (0)
ConnTimeout : 0x07D0 (2000)
ClockAccuracy : 0x00 (0)
Dump(Rx):
04 FF 13 05 06 00 00 89 05 84 B7 2D 3C 00 00 50
00 00 00 D0 07 00

Figure 19

4.3 Using the Simple GATT Profile

The SimpleBLEPeripheral software contains one sample GATT service profile (More information on the SimpleGATTProfile can be found in [2]). GATT services contain data values known as "characteristic values". All application data that is being sent or received in BLE must be contained within characteristic value. This section details a step-by-step process that demonstrates several processes for reading, writing, discovering, and notifying GATT characteristic values using BTool.

Note that the types (UUIDs) of the five characteristic values (0xFFF1, 0xFFF2, 0xFFF3, 0xFFF4, and 0xFFF5), as well as the simple profile primary service UUID value (0xFFF0), do not conform to any specifications in the *Bluetooth* SIG. They are simply used as a demonstration.

The tables in Figure 20 and Figure 21 below show the SimpleBLEPeripheral complete attribute table, and can be used as a reference. Services are shown in yellow, characteristics are shown in blue, and characteristic values / descriptors are shown in grey. When working with the SimplyBLEPeripheral application, it might be useful to print out the table as a reference.

SimpleBLEPeripheral Application: Complete Attribute Table						
handle (hex)	handle (dec)	Type (hex)	Type (#DEFINE)	Hex / Text Value (default)	GATT Server Permissions	Notes
0x1	1	0x2800	GATT_PRIMARY_SERVICE_UUID	0x1800 (GAP_SERVICE_UUID)	GATT_PERMIT_READ	Start of GAP Service (Mandatory)
0x2	2	0x2803	GATT_CHARACTER_UUID	02 (properties: read only) 03 00 (handle: 0x0003) 00 2A (UUID: 0x2A00)	GATT_PERMIT_READ	Device Name characteristic declaration
0x3	3	0x2A00	GAP_DEVICE_NAME_UUID	"Simple BLE Peripheral"	GATT_PERMIT_READ	Device Name characteristic value
0x4	4	0x2803	GATT_CHARACTER_UUID	02 (properties: read only) 05 00 (handle: 0x0005) 01 2A (UUID: 0x2A01)	GATT_PERMIT_READ	Appearance characteristic declaration
0x5	5	0x2A01	GAP_APPEARANCE_UUID	0x0000	GATT_PERMIT_READ	Appearance characteristic value
0x6	6	0x2803	GATT_CHARACTER_UUID	0A (properties: read/write) 07 00 (handle: 0x0007) 02 2A (UUID: 0x2A02)	GATT_PERMIT_READ	Peripheral Privacy Flag characteristic declaration
0x7	7	0x2A02	GAP_PERI_PRIVACY_FLAG_UUID	0x00 (GAP_PRIVACY_DISABLED)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Peripheral Privacy Flag characteristic value
0x8	8	0x2803	GATT_CHARACTER_UUID	0A (properties: read/write) 09 00 (handle: 0x0009) 03 2A (UUID: 0x2A03)	GATT_PERMIT_READ	Reconnection address characteristic declaration
0x9	9	0x2A03	GAP_RECONNECT_ADDR_UUID	00:00:00:00:00:00	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Reconnection address characteristic value
0xA	10	0x2803	GATT_CHARACTER_UUID	02 (properties: read only) 0B 00 (handle: 0x000B) 04 2A (UUID: 0x2A04)	GATT_PERMIT_READ	Peripheral Preferred Connection Parameters characteristic declaration
0xB	11	0x2A04	GAP_PERI_CONN_PARAM_UUID	50 00 (100ms preferred min connection interval) A0 00 (200ms preferred max connection interval) 00 00 (0 preferred slave latency) E8 03 (10000ms preferred supervision timeout)	GATT_PERMIT_READ	Peripheral Preferred Connection Parameters characteristic declaration
0xC	12	0x2800	GATT_PRIMARY_SERVICE_UUID	0x1801 (GATT_SERVICE_UUID)	GATT_PERMIT_READ	Start of GATT Service (mandatory)
0xD	13	0x2803	GATT_CHARACTER_UUID	20 (properties: indicate only) 0E 00 (handle: 0x000E) 05 2A (UUID: 0x2A05)	GATT_PERMIT_READ	Service Changed characteristic declaration
0xE	14	0x2A05	GATT_SERVICE_CHANGED_UUID	(null value)	(none)	Service Changed characteristic value
0xF	15	0x2800	GATT_PRIMARY_SERVICE_UUID	0x180A (DEVINFO_SERV_UUID)	GATT_PERMIT_READ	Start of Device Information Service
0x10	16	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 11 00 (handle 0x0011) 23 2A (UUID 0x2A23)	GATT_PERMIT_READ	System ID characteristic declaration
0x11	17	0x2A23	DEVINFO_SYSTEM_ID_UUID	xx xx xx 00 00 xx xx xx (xx's are IEEE address)	GATT_PERMIT_READ	System ID
0x12	18	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 13 00 (handle 0x0013) 24 2A (UUID 0x2A24)	GATT_PERMIT_READ	Model Number String characteristic declaration
0x13	19	0x2A24	DEVINFO_MODEL_NUMBER_UUID	"Model Number"	GATT_PERMIT_READ	Model Number String
0x14	20	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 15 00 (handle 0x0015) 25 2A (UUID 0x2A25)	GATT_PERMIT_READ	Serial Number String characteristic declaration
0x15	21	0x2A25	DEVINFO_SERIAL_NUMBER_UUID	"Serial Number"	GATT_PERMIT_READ	Serial Number String
0x16	22	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 17 00 (handle 0x0017) 26 2A (UUID 0x2A26)	GATT_PERMIT_READ	Firmware Revision String characteristic declaration
0x17	23	0x2A26	DEVINFO_FIRMWARE_REV_UUID	"Firmware Revision"	GATT_PERMIT_READ	Firmware Revision String
0x18	24	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 19 00 (handle 0x0019) 27 2A (UUID 0x2A27)	GATT_PERMIT_READ	Hardware Revision String characteristic declaration
0x19	25	0x2A27	DEVINFO_HARDWARE_REV_UUID	"Hardware Revision"	GATT_PERMIT_READ	Hardware Revision String
0x1A	26	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 1B 00 (handle 0x001B) 28 2A (UUID 0x2A28)	GATT_PERMIT_READ	Software Revision String characteristic declaration
0x1B	27	0x2A28	DEVINFO_SOFTWARE_REV_UUID	"Software Revision"	GATT_PERMIT_READ	Software Revision String
0x1C	28	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 1D 00 (handle 0x001D) 29 2A (UUID 0x2A29)	GATT_PERMIT_READ	Manufacturer Name String characteristic declaration
0x1D	29	0x2A29	DEVINFO_MANUFACTURER_NAME_UUID	"Manufacturer Name"	GATT_PERMIT_READ	Manufacturer Name String
0x1E	30	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 1F 00 (handle 0x001F) 2A 2A (UUID 0x2A2A)	GATT_PERMIT_READ	IEEE 11073-20601 Regulatory Certification Data List characteristic declaration
0x1F	31	0x2A2A	DEVINFO_11073_CERT_DATA_UUID	FE 00 65 78 70 65 72 69 6D 65 6E 74 61 6C	GATT_PERMIT_READ	IEEE 11073-20601 Regulatory Certification Data List

Figure 20

SimpleBLEPeripheral Application: Complete Attribute Table

	handle (hex)	handle (dec)	Type (hex)	Type (#DEFINE)	Hex / Text Value (default)	GATT Server Permissions	Notes
	0x20	32	0x2800	GATT_PRIMARY_SERVICE_UUID	0xFFF0 (SIMPLEPROFILE_SERV_UUID)	GATT_PERMIT_READ	Start of Simple GATT Profile Service
	0x20	32	0x2803	GATT_CHARACTER_UUID	0A (properties: read/write) 22 00 (handle: 0x0022) F1 FF (UUID: 0xFFF1)	GATT_PERMIT_READ	Characteristic 1 declaration
	0x22	34	0xFFF1	SIMPLEPROFILE_CHAR1_UUID	1 (1 byte)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Characteristic 1 value
	0x23	35	0x2901	GATT_CHAR_USER_DESC_UUID	"Characteristic 1" (17 bytes)	GATT_PERMIT_READ	Characteristic 1 user description
	0x24	36	0x2803	GATT_CHARACTER_UUID	02 (properties: read only) 25 00 (handle: 0x0025) F2 FF (UUID: 0xFFF2)	GATT_PERMIT_READ	Characteristic 2 declaration
	0x25	37	0xFFF2	SIMPLEPROFILE_CHAR2_UUID	2 (1 byte)	GATT_PERMIT_READ	Characteristic 2 value
	0x26	38	0x2901	GATT_CHAR_USER_DESC_UUID	"Characteristic 2" (17 bytes)	GATT_PERMIT_READ	Characteristic 2 user description
	0x27	39	0x2803	GATT_CHARACTER_UUID	08 (properties: write only) 28 00 (handle: 0x0028) F3 FF (UUID: 0xFFF3)	GATT_PERMIT_READ	Characteristic 3 declaration
	0x28	40	0xFFF3	SIMPLEPROFILE_CHAR3_UUID	3 (1 byte)	GATT_PERMIT_WRITE	Characteristic 3 value
	0x29	41	0x2901	GATT_CHAR_USER_DESC_UUID	"Characteristic 3" (17 bytes)	GATT_PERMIT_READ	Characteristic 3 user description
	0x2A	42	0x2803	GATT_CHARACTER_UUID	10 (properties: notify only) 2B 00 (handle: 0x002B) F4 FF (UUID: 0xFFF4)	GATT_PERMIT_READ	Characteristic 4 declaration
	0x2B	43	0xFFF4	SIMPLEPROFILE_CHAR4_UUID	4 (1 byte)	(none)	Characteristic 4 value
	0x2C	44	0x2902	GATT_CLIENT_CHAR_CFG_UUID	00:00 (2 bytes)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Characteristic 4 configuration
	0x2D	45	0x2901	GATT_CHAR_USER_DESC_UUID	"Characteristic 4" (17 bytes)	GATT_PERMIT_READ	Characteristic 4 user description
	0x2E	46	0x2803	GATT_CHARACTER_UUID	02 (properties: read only) 2F 00 (handle: 0x002F) F5 FF (UUID: 0xFFF5)	GATT_PERMIT_READ	Characteristic 5 declaration
	0x2F	47	0xFFF5	SIMPLEPROFILE_CHAR5_UUID	01:02:03:04:05 (5 bytes)	GATT_PERMIT_ AUTHEN_READ	Characteristic 5 value
	0x30	48	0x2901	GATT_CHAR_USER_DESC_UUID	"Characteristic 5" (17 bytes)	GATT_PERMIT_READ	Characteristic 5 user description
CC2540DK-MINI keyfob ony	0x31	49	0x2800	GATT_PRIMARY_SERVICE_UUID	0xFFE0 (SK_SERVICE_UUID)	GATT_PERMIT_READ	Start of Simple Keys Service
	0x32	50	0x2803	GATT_CHARACTER_UUID	10 (properties: notify only) 33 00 (handle: 0x0033) E1 FF (UUID: 0xFFE1)	GATT_PERMIT_READ	Key Press State characteristic declaration
	0x33	51	0xFFE1	SK_KEYPRESSED_UUID	0 (1 byte)	(none)	Key Press State characteristic value
	0x34	52	0x2902	GATT_CLIENT_CHAR_CFG_UUID	00:00 (2 bytes)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Key Press State characteristic configuration
	0x35	53	0x2901	GATT_CHAR_USER_DESC_UUID	"Key Press State" (16 bytes)	GATT_PERMIT_READ	Key Press State characteristic user description

Figure 21

4.3.1 Reading a Characteristic Value by UUID

The first characteristic of the SimpleGATTProfile service has both read and write permissions, and has a UUID of 0xFFF1. The simplest way to read its value is to use the "Read Characteristic by UUID" sub-procedure. To do this, you will first need to click the "Read / Write" tab in BTool. Select the option "Read Using Characteristic UUID" under the "Sub-Procedure" option in the "Characteristic Read" section at the top of the screen. Enter "F1:FF" (note that the LSB is entered first, and the MSB is entered last) in the "Characteristic UUID" box, and click the "Read" button.

An attribute protocol *Read by Type Request* packet gets sent over the air from the dongle to the keyfob, and an attribute protocol *Read by Type Response* packet gets sent back from the keyfob to the dongle. The value "01" is displayed in the "Value" box, and "Success" is displayed in the "Status" box. In addition, the message window will display information on the *Read by Type Response* packet that was received by the dongle. The message includes not only the characteristic's data value, but also the handle of the characteristic value (0x0022 in this case).

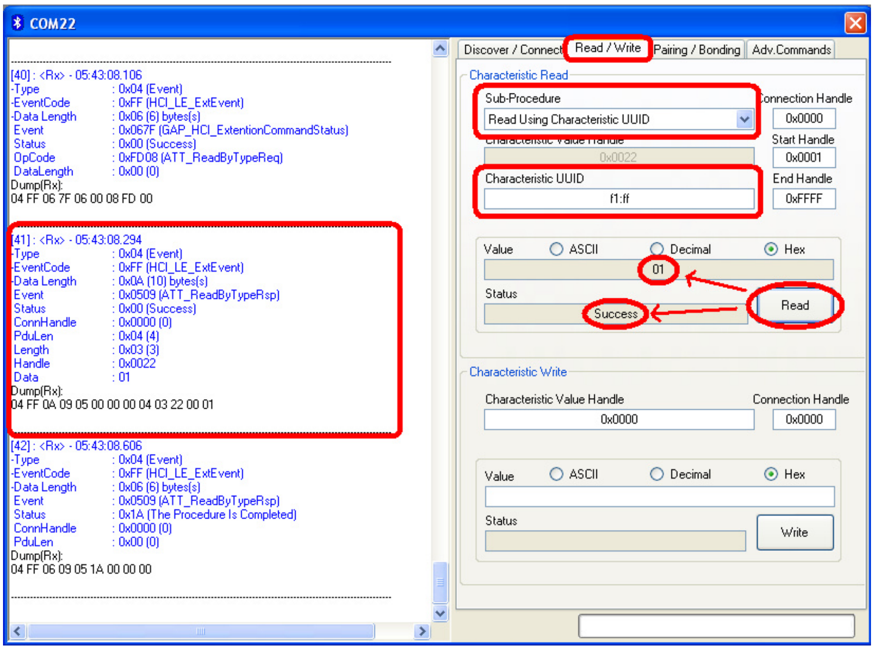

Figure 22

4.3.2 Writing a Characteristic Value

In the previous section, the handle of the first characteristic in the SimpleGATTProfile was found to be 0x0022. Knowing this, and based on the fact that the characteristic has both read and write permissions, it is possible for us to write a new value. Enter "0x0022" into the "Characteristic Value Handle" box in the "Characteristic Write" section, and enter any 1-byte value in the "Value" section (the format can be set to either "Decimal" or "Hex"). Click the "Write Value" button.

An attribute protocol *Write Request* packet gets sent over the air from the dongle to the keyfob, and an attribute protocol *Write Response* packet gets sent back from the keyfob to the dongle. The status box will display "Success", indicating that the write was successful.

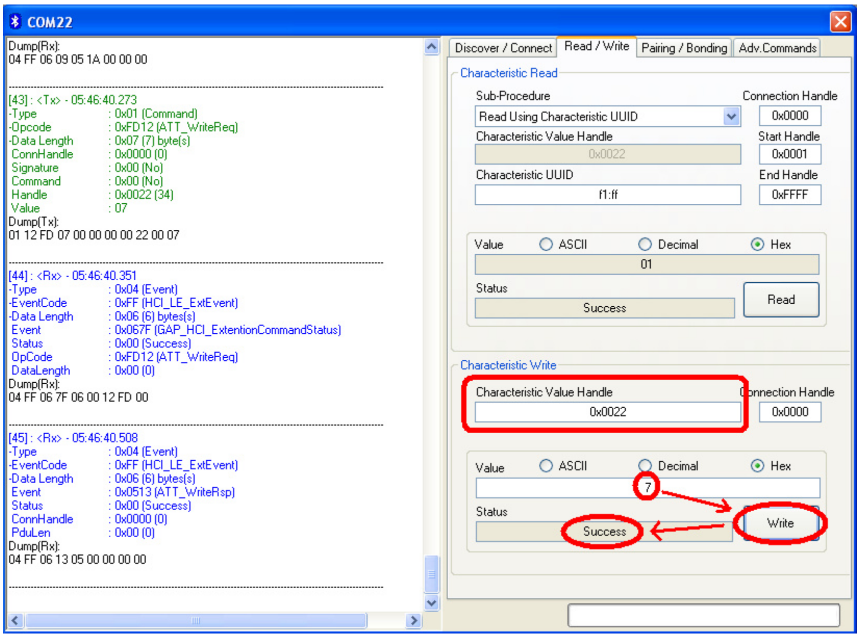

Figure 23

4.3.3 Reading a Characteristic Value by Handle

After writing a new value to the first characteristic in the profile, we can read the value back to verify the write. This time, instead of reading the value by its UUID, the value will be read by its handle. Select the option “Read Characteristic Value / Descriptor” under the “Sub-Procedure” option in the “Characteristic Read” section. Enter “0x0011” in the “Characteristic Value Handle” box, and click the “Read” button.

An attribute protocol *Read Request* packet gets sent over the air from the dongle to the keyfob, and an attribute protocol *Read Response* packet gets sent back from the keyfob to the dongle. The new value is displayed in the “Value” box, and “Success” is displayed in the “Status” box. This value should match the value that was written in the previous step.

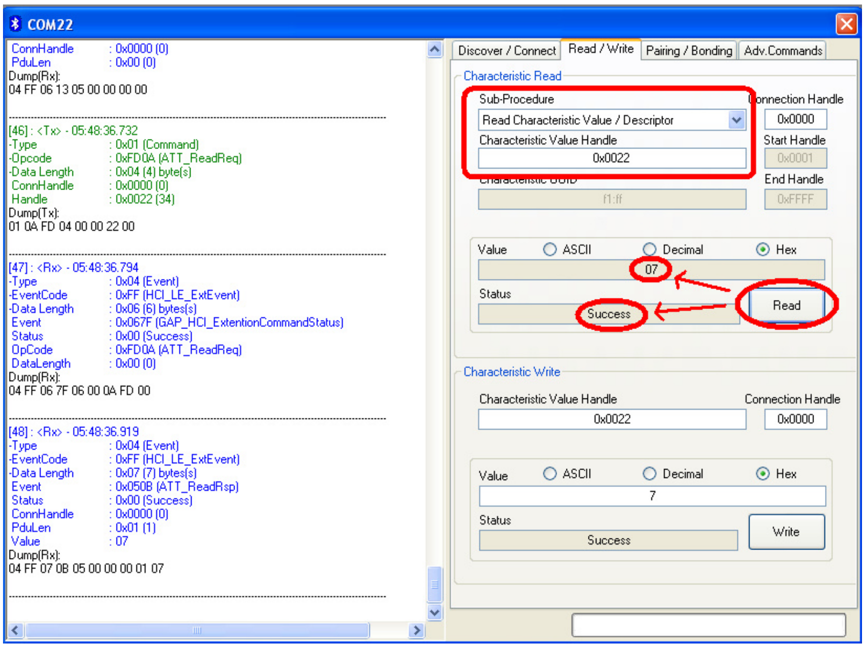

Figure 24

4.3.4 Discovering a Characteristic by UUID

The next thing to do is to discover a characteristic by its UUID. By doing this, we will not only get the handle of the UUID, but we will also get the properties of the characteristic. The UUID of the second characteristic in the SimpleGATTProfile is 0xFFF2. Select the option “Discover Characteristic by UUID” under the “Sub-Procedure” option in the “Characteristic Read” section at the top of the screen. Enter “F2:FF” in the “Characteristic UUID” box, and click the “Read” button.

A series of attribute protocol *Read by Type Request* packets get sent over the air from the dongle to the keyfob, and for each request an attribute protocol *Read by Type Response* packet gets sent back from the keyfob to the dongle. Essentially, the dongle is reading every attribute on the keyfob with a UUID of 0x2803 (this is the UUID for a characteristic declaration as defined in [5]), and checking the “Characteristic Value UUID” portion of each declaration to see if it matches type 0xFFF2. The procedure is complete once every characteristic declaration has been read.

The procedure will find one instance of the characteristic with type 0xFFF2, and display “02 25 00 F2 FF” (the value of the declaration) in the “Value” box, with “Success” displayed in the “Status” box. As per the *Bluetooth* specification, the first byte “02” tells us that the properties of the characteristic are read-only. The second and third bytes “25 00” tell us that the handle of the characteristic value is 0x0025. The fourth and fifth bytes tell the UUID of the characteristic, 0xFFF2.

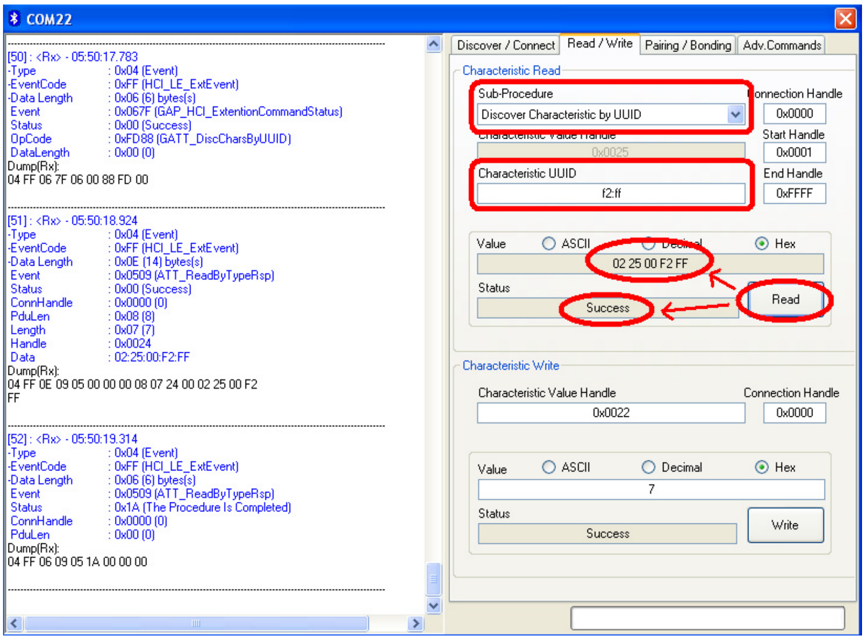

Figure 25

4.3.5 Reading Multiple Characteristic Values

It is also possible to read multiple characteristic values with one request, as long as the handle of each value is known. To read the values of both of the characteristics that we previously read, select the option “Read Multiple Characteristic Values” under the “Sub-Procedure” option in the “Characteristic Read” section at the top of the screen. Enter “0x0022;0x0025” in the “Characteristic Value Handle” box, and click the “Read” button.

An attribute protocol *Read Multiple Request* packet gets sent over the air from the dongle to the keyfob, and an attribute protocol *Read Multiple Response* packet gets sent back from the keyfob to the dongle. The values of the two characteristics are displayed in the “Value” box, and “Success” is displayed in the “Status” box. This first byte should match the value that was written in the previous step, and the second byte should be “02”.

One important note about reading multiple characteristic values in a single request is that the response will not parse the separate values. This means that the size of each value being read must be fixed, and must be known by the client. In the example here, this is not an issue since there are only two bytes in the response; however care must be taken when using this command.

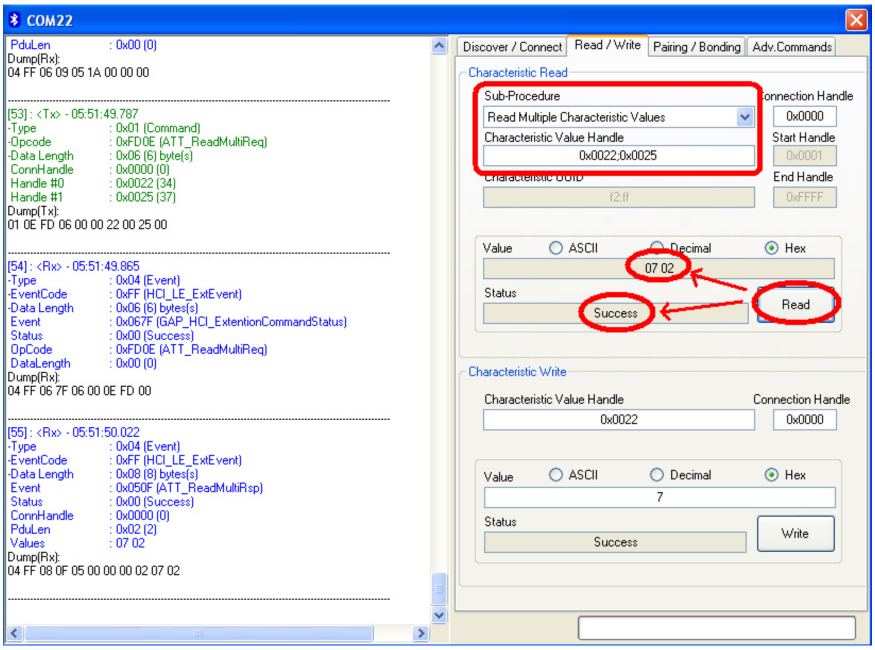

Figure 26

4.3.6 Enabling Notifications

In BLE, it is possible for a GATT server device to "push" characteristic value data out to a client device, without being prompted with a read request. This process is called a "characteristic value notification". Notifications are useful in that they allow a device in a BLE connection to send out as much or as little data as required at any point in time. In addition, since no request from the client is required, the overhead is reduced and the data is transmitted more efficiently. The SimpleBLEPeripheral software contains an example in which notifications can be demonstrated.

The third characteristic in the SimpleGATTProfile has write-only properties, while the fourth characteristic in the profile has notify-only properties. Every five seconds, the SimpleBLEPeripheral application will take the value of the third characteristic and copy it into the fourth characteristic. Each time the fourth characteristic value gets set by the application, the profile will check to see if notifications are enabled. If they are enabled, the profile will send a notification of the value to the client device.

Before notifications can be enabled, the handle of the fourth characteristic must be found. This can be done by using the "Discover Characteristic by UUID" process (see section 4.3.4), with the UUID value set to "F4:FF". The procedure will find one instance of the characteristic with type 0xFFF4, and display "10 2B 00 F4 FF" (the value of the declaration) in the "Value" box, with "Success" displayed in the "Status" box. As per the *Bluetooth* specification, the first byte "10" tells us that the properties of the characteristic are notify-only. The second and third bytes "2B 00" tell us that the handle of the characteristic value is 0x002B. The fourth and fifth bytes tell the UUID of the characteristic, 0xFFF4.

In order to enable notifications, the client device must write a value of 0x0001 to the client characteristic configuration descriptor for the particular characteristic. The handle for the client characteristic configuration descriptor immediately follows the characteristic value's handle. Therefore, a value of 0x0001 must be written to handle 0x002C. Enter "0x002C" into the "Characteristic Value Handle" box in the "Characteristic Write" section, and enter "01:00" in the "Value" section (note that the LSB is entered first, and the MSB is entered last). Click the "Write Value" button. The status box will display "Success", indicating that the write was successful.

Every five seconds, an attribute protocol *Handle Value Notification* packet gets sent from the keyfob to the dongle. With each notification, the value of the characteristic at handle is displayed in the log window.

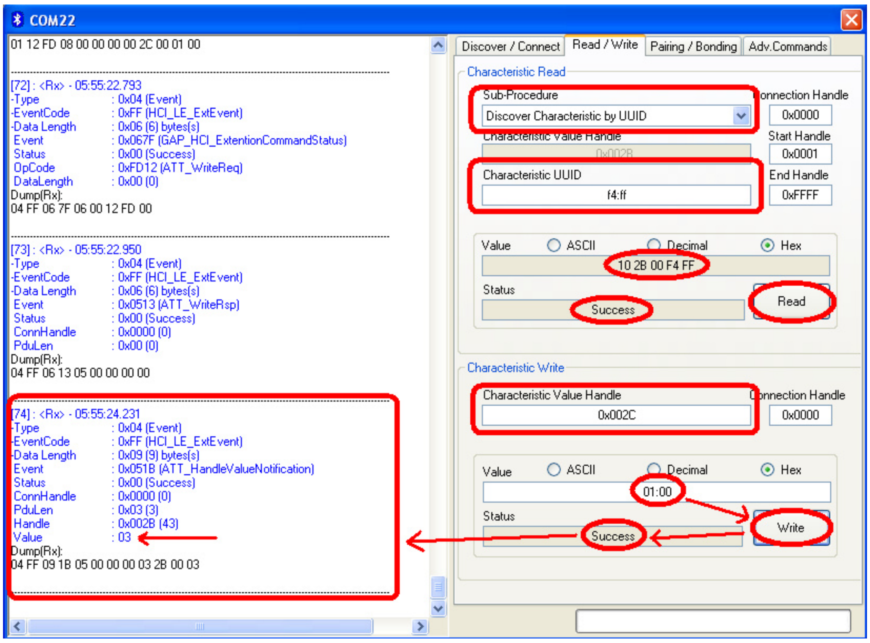

Figure 27

The value should be “03” in each notification, since it is copied from the value of the third characteristic in the profile (which has a default value of 3). The third characteristic has write-only properties, and therefore can be changed. By following the procedure from section 4.3.4, the handle of the third characteristic can be found to be 0x0028. By following the procedure from section 4.3.2, a new value can be written to handle 0x0028. Once this is done, the value of the fourth characteristic will change. This new value is reflected in the incoming notification messages.

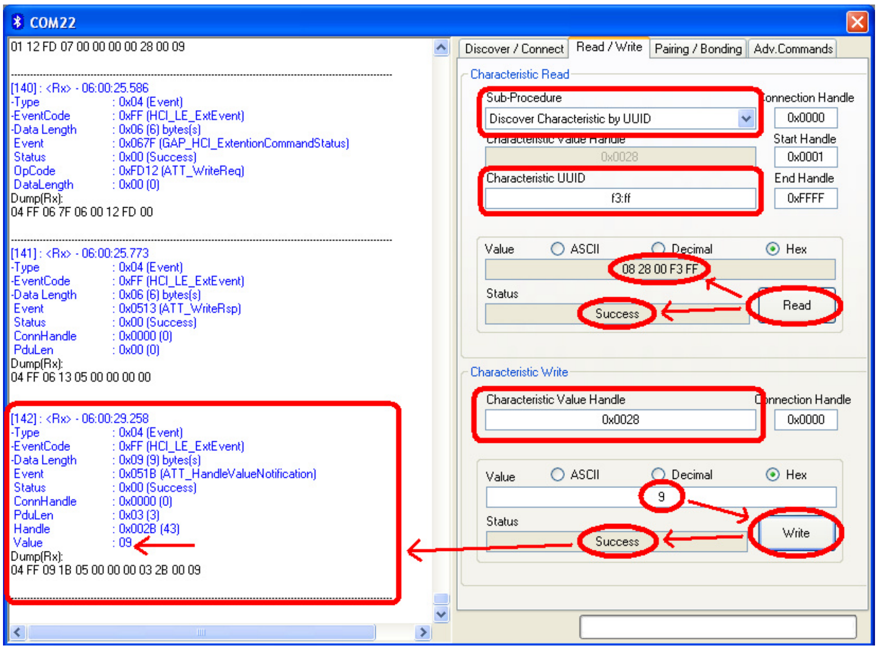

Figure 28

4.4 Using the Simple Keys GATT Profile

The simple keys profile on the keyfob allows the device to send notifications of key presses and releases to a central device. The UUID of the simple keys data characteristic value is 0xFFE1.

Using the same discovery process as before with the “Discover Characteristic by UUID” command, it can be determined that the handle of the simple keys data is 0x0033. Like the fourth characteristic value in the simple GATT profile, the simple keys data is a “configurable” characteristic, in that the client device can configure the server to send notifications of the characteristic value. The handle immediately following the characteristic value is the client characteristic configuration descriptor.

The characteristic configuration of the simple keys data is the attribute at handle 0x0034. To turn on notifications, enter 0x0034 into the “Characteristic Value Handle” box in the “Characteristic Write” section, and enter “01:00” in the “Value” section. The format can be set to either “Hex” or “Decimal”. Click the “Write” button to send the write request over the air. When the keyfob receives the request, it will turn on notifications of the simple keys data, and send a write response to indicate success.

With notifications enabled, an attribute protocol *Handle Value Notification* packet gets sent from the keyfob to the dongle as you press or release either of the buttons on the keyfob. The notifications should show up in the message window. A value of “00” indicates that neither key is pressed. A value of “01” indicates that the left key is pressed. A value of “02” indicates that the right key is pressed. A value of “03” indicates that both keys are pressed.

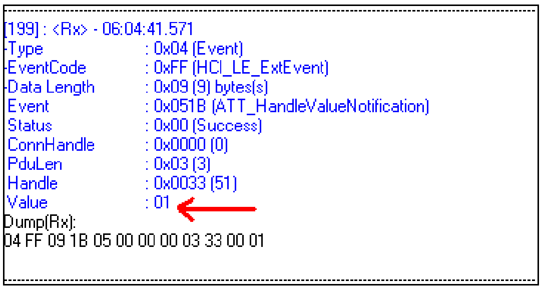

Figure 29

It is important to note that the simple keys profile included with the BLE development kit does not conform to any standard profile specification available from the *Bluetooth* SIG. At the time of the release of the software, no official GATT service profile specifications have been approved by the *Bluetooth* SIG. Therefore the profile, including the GATT characteristic definition, the UUID values, and the functional behavior, was developed by Texas Instruments for use with the CC2540DK-MINI development kit.

As the *Bluetooth* SIG begins to approve specifications for different service profiles, Texas Instruments plans to release updates to the BLE software development kit with source code conforming to the specifications.

4.5 Using BLE Security

BTool also includes the ability to make use of security features in BLE, including encryption, authentication, and bonding.

4.5.1 Encrypting the Connection

The SimpleGATTProfile contains a fifth characteristic with a UUID of 0xFFF5. Like the second characteristic, this characteristic has read-only permissions; however this characteristic can only be read if the link is encrypted.

Using the same discovery process as before with the “Discover Characteristic by UUID” command, it can be determined that the handle of the simple keys data is 0x002F. If you attempt to read this characteristic, however, an error will occur with a status of “INSUFFICIENT_ENCRYPTION”.

To encrypt the link, the pairing process must be initiated. Click on the “Pairing / Bonding” tab in BTool. In the “Initiate Pairing” section at the top of the screen, check the boxes labeled “Bonding Enabled” and “Authentication (MITM) Enabled”, and click the button “Send Pairing Request”. This will send the request to the peripheral device.

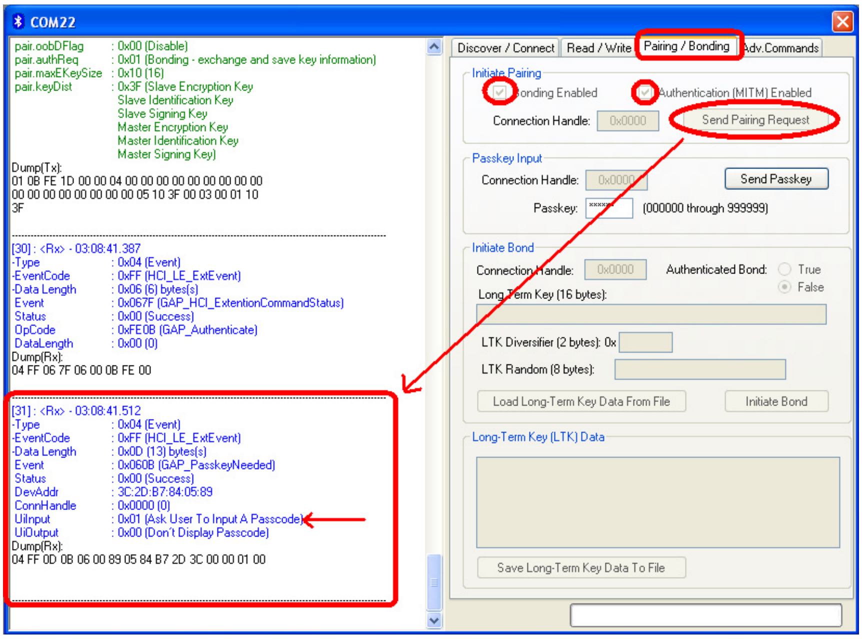

Figure 30

The peripheral will send a pairing response in return, which will require a six-digit passcode to be entered by the user in order to complete the process. Typically, this passcode is intended to be used by a peripheral device containing a display. By displaying the passkey on the peripheral device and requiring the user to enter it in on the central device's user interface, the link is authenticated, in that it has been verified that the connection has not been hijacked using a man-in-the-middle (MITM) attack.

In the case of the SimpleBLEPeripheral software, a fixed passcode "000000" is used, since the keyfob does not have a display (this value can be modified in the source code). In the box labeled "Passkey" in the "Passkey Input" section, enter the value "000000" and click the "Send Passkey" button. Note that if you do not send the passkey within 30 seconds after receiving the pairing response message, the pairing process will fail, and you will need to re-send the pairing request.

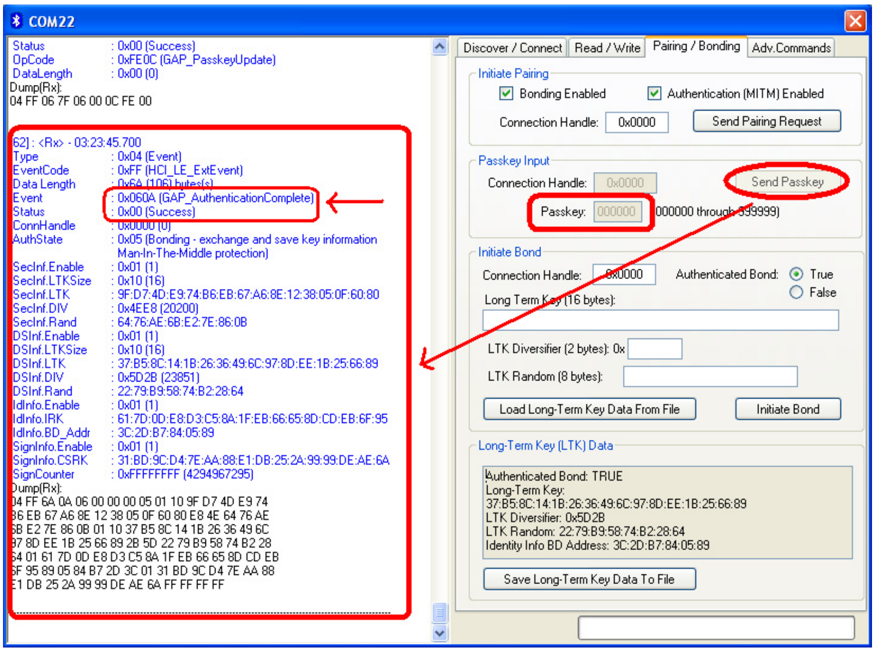

Figure 31

When pairing is successfully completed, you will see a "GAP_AuthenticationComplete" event in the log window, with a "Success" status. The BLE connection is now encrypted. You will now be able to read the fifth characteristic value (handle 0x002F) from the peripheral. The five-byte value of the characteristic is "01 02 03 04 05".

4.5.2 Using Bonding and Long-Term Keys

Bonding is a feature in BLE that allows a device, after initial pairing with a peer, to remember specific information about that peer device. In particular, the long-term key data that is generated during the initial pairing process can be stored locally. If the connection is then terminated and the two devices later reconnect, this data can be used to quickly re-initiate encryption without needing to go through the full pairing process and/or use a passkey. In addition, if a client device had enabled notifications of any characteristics on the server device while the two devices were bonded, the server device will remember the setting and the client will not have to re-enable them.

After pairing has been completed with bonding enabled, the "Long-Term Key (LTK) Data" will be populated with some of the data from the "GAP_AuthenticationComplete" event that was generated during the encryption process. This data is required for re-initiating encryption upon reconnect. Click the "Save Long-Term Key Data to File" button to save this information to file. The data is saved as in a "comma separated value" (CSV) format as simple text, and can be store anywhere on disk. Be sure to note the location that the file is stored.

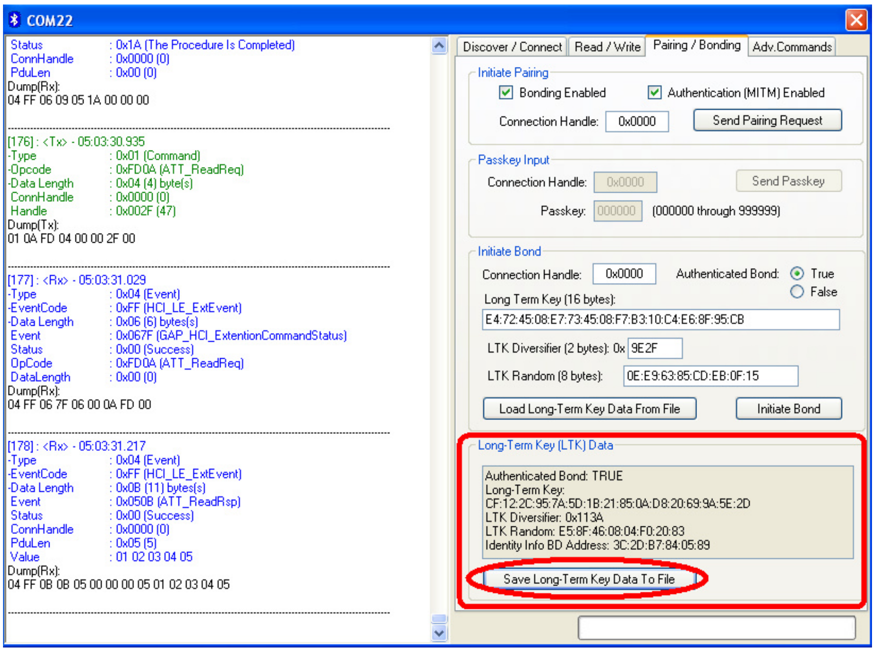

Figure 32

Within the keyfob, a similar process is going on, in that the SimpleBLEPeripheral software contains a bond manager that is storing the long-term key data that it had generated during encryption. Since the SimpleBLEPeripheral does not have a file system, it is simply storing the data in the nonvolatile memory of the CC2540. More information on the bond manager can be found in [2].

With a bond now active, you can enable notifications of a characteristic value and have that setting remembered for later. Note that if notifications were enabled before going through the pairing process, then the setting will not be stored. Therefore, you will need to re-write the value "01:00" to a client characteristic configuration descriptor. For example, write "01:00" to handle 0x0034 to enable notifications of key presses, as was done in section 4.4. You should now be receiving notifications whenever the buttons are pressed or released. Because the devices are paired with bonding enabled, the bond manager in the SimpleBLEPeripheral software will store the client characteristic configuration descriptor data in nonvolatile memory.

To verify that bonding worked, you will need to disconnect and re-connect. Click on the "Discover / Connect" tab and click the "Terminate" button at the bottom of the screen to disconnect from the keyfob. The message window will show a "GAP_TerminateLink" event with "Success" status. In addition, the connection information in the upper-left corner of the screen will disappear.

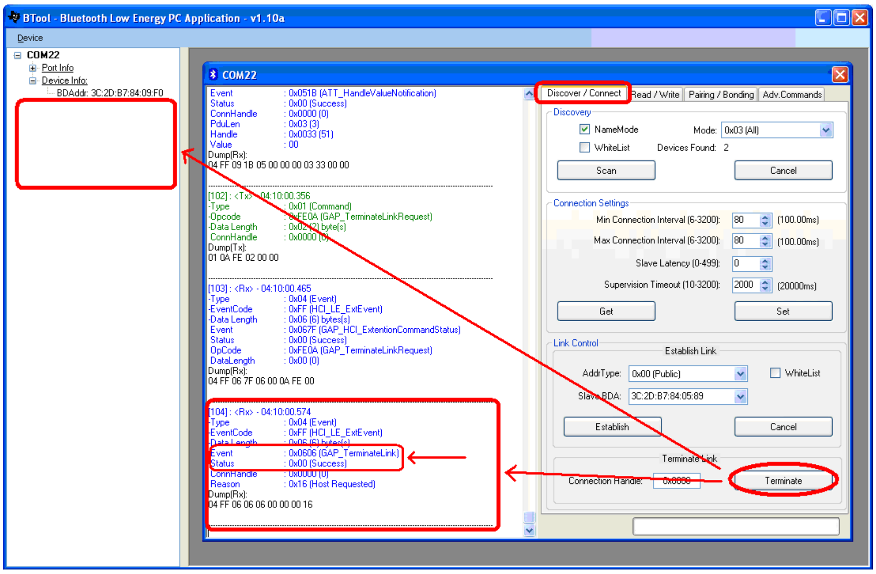

Figure 33

At a later time, re-connect with the keyfob following the procedure in section 4.2.4. Once connected, you will notice that the simple keys notifications are no longer enabled. This is because the Simple Keys profile will always reset the value of the client characteristic configuration descriptor back to “00:00” if a connection is terminated or if the device resets.

To re-initiate encryption and re-enable notifications of key presses, return to the “Pairing / Bonding” tab. In the “Initiate Bond” section, click the “Load Long-Term Key Data From File” button, and select the file in which the data was previously stored. The data fields will get automatically populated from the data in the file. Click the “Initiate Bond” button to re-enable encryption.

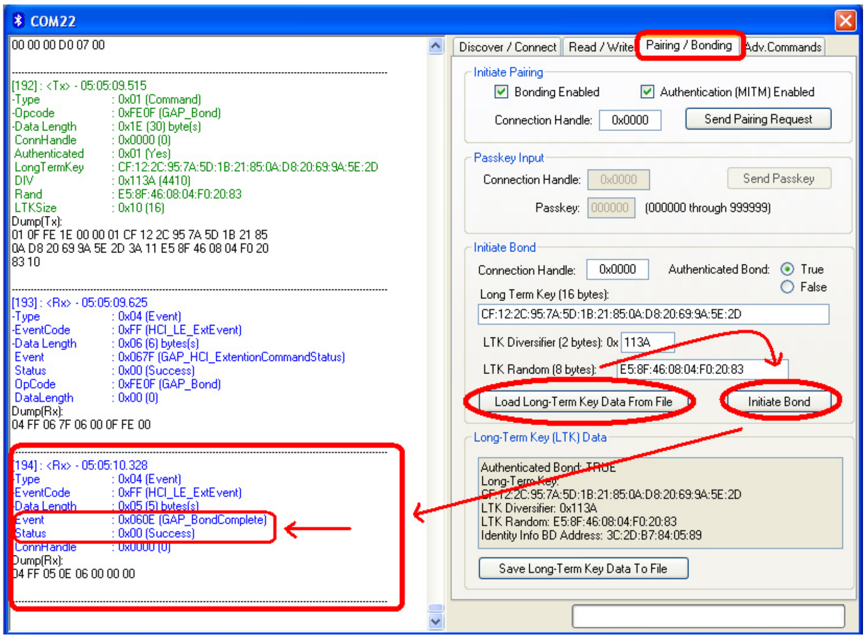

Figure 34

A “GAP_BondComplete” event with “Success” status will be displayed in the message window. This indicates that the link has been re-encrypted, which can be verified by reading the fifth characteristic value in the SimpleGATTProfile at handle 0x002F. You will also now be able to receive notifications now when the buttons on the keyfob are pressed or released, as the client characteristic configuration descriptor value of the key press characteristic has been stored. Any changes to the client characteristic configuration descriptor value (i.e. turning off notifications) will be saved to nonvolatile memory and remembered for next time that encryption is initiated using the long-term key.

4.6 Additional Sample Applications

In addition to the SimpleBLEPeripheral application, the BLE software development kit includes project and source code files for several additional applications and profiles, including:

- Blood Pressure Sensor- with simulated measurements
- Emulated Keyboard- press the two buttons on the keyfob to simulate keyboard presses
- Heart Rate Sensor- with simulated measurements
- Health Thermometer- with simulated measurements
- KeyFob Demo- uses the accelerometer and buzzer on the keyfob, also a proximity alarm

More information on these projects can be found in [4].

5. Programming / Debugging the CC2540

The CC Debugger included with the CC2540DK-MINI kit allows for debugging using IAR Embedded Workbench, as well as for reading and writing hex files to the CC2540 flash memory using the SmartRF Flash Programmer software. SmartRF Flash Programmer also has the capability to change the IEEE address of the CC2540 device. The BLE software development kit includes hex files for both the USB Dongle as well as the keyfob. This section details the hardware setup when using the CC Debugger, as well as information on using SmartRF Flash Programmer. Information on using IAR Embedded Workbench for debugging can be found in [2]

5.1 Hardware Setup for Keyfob

If the keyfob is viewed with the LED's on top and the coin cell battery holder at the bottom, then the set of pins closer to the top are the ones that should be used for connecting to the debugger. Pin 1 is the pin on the lower right side:

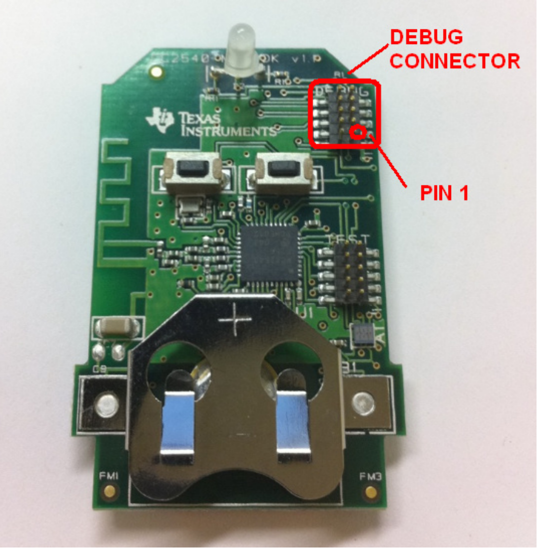

Figure 35

Connect the CC Debugger to the keyfob as shown below. Be sure that the ribbon cable is oriented properly, with the red stripe connected to pin 1:

Figure 36

Connect the CC Debugger to the PC's USB port and insert a coin cell battery in the keyfob. The status indicator LED on the CC Debugger should turn on. If the LED is red, that means no CC2540 device was detected. If it is green, then a CC2540 device has been detected. If the keyfob is connected and the LED is red, try pressing the reset button on the CC Debugger. This resets the debugger and re-checks for a CC2540 device. If the LED still does not turn green, re-check that all cables are securely connected. Also verify that the CC Debugger has the latest firmware (see section 5.3.1).

Figure 37

Once the CC Debugger is set up with the status indicator LED showing green, you are ready to either read or write a hex file from the board, or to begin debugging a project using IAR.

Power Savings Tip: Do not leave the CC Debugger connected to the keyfob for extended periods of time with the battery in the keyfob. As long as the CC Debugger is connected to the keyfob, power management on the keyfob will be disabled. This will cause a high, constant current draw from the battery, and will significantly reduce the battery life.

If you intend to perform a lot of debugging and expect to leave the debugger connected to the keyfob for long periods of time, it is possible to draw power directly from the USB bus through the CC Debugger, in which case a battery will not be required as long as the debugger is connected. To do this, locate the pads for resistor R1, which are located immediately next to the debug header on the keyfob. Using a soldering iron, solder a small piece of wire across the two pads, shorting them together.

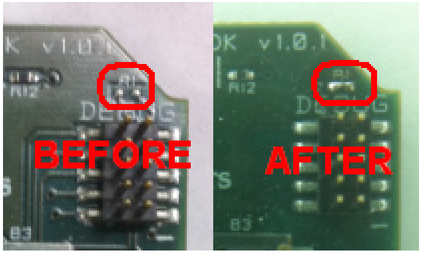

Figure 38

5.2 Hardware Setup for USB Dongle

The setup process for flashing the USB Dongle is very similar to the process when flashing the keyfob. First, plug the USB Dongle into a PC USB port (or a USB hub):

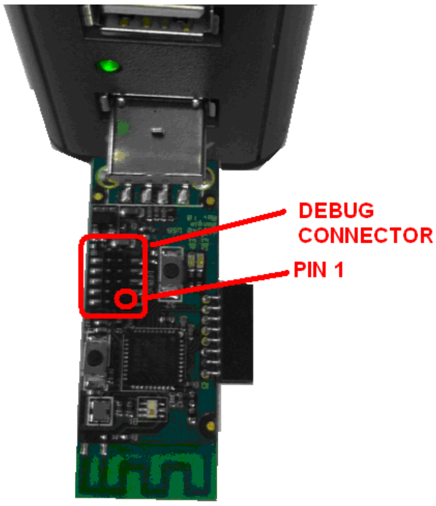

Figure 39

Connect the CC Debugger to the USB Dongle as shown below. Be sure that the ribbon cable is oriented properly, with the red stripe connected to pin 1:

Figure 40

Connect the CC Debugger to the PC USB port. The status indicator LED on the CC Debugger should turn on. If the LED is red, that means no CC2540 device was detected. If it is green, then a CC2540 device has been detected. If the USB Dongle is connected and the LED is red, try pressing the reset button on the CC Debugger. This resets the debugger and re-checks for a CC2540 device. If the LED still does not turn green, re-check that all cables are securely connected.

Figure 41

Once the CC Debugger status LED is showing green, you are ready to use IAR to debug or to read or write a hex file from/to the USB Dongle.

5.3 Using SmartRF Flash Programmer Software

Note: the instructions in the section apply to the latest version of SmartRF Flash Programmer (version 1.9.0.0), which is available at the following URL:

http://focus.ti.com/docs/toolsw/folders/print/flash-programmer.html

To start the application go into your programs by choosing Start > Programs > Texas Instruments > SmartRF Flash Programmer > SmartRF Flash Programmer. The program should open up the following window:

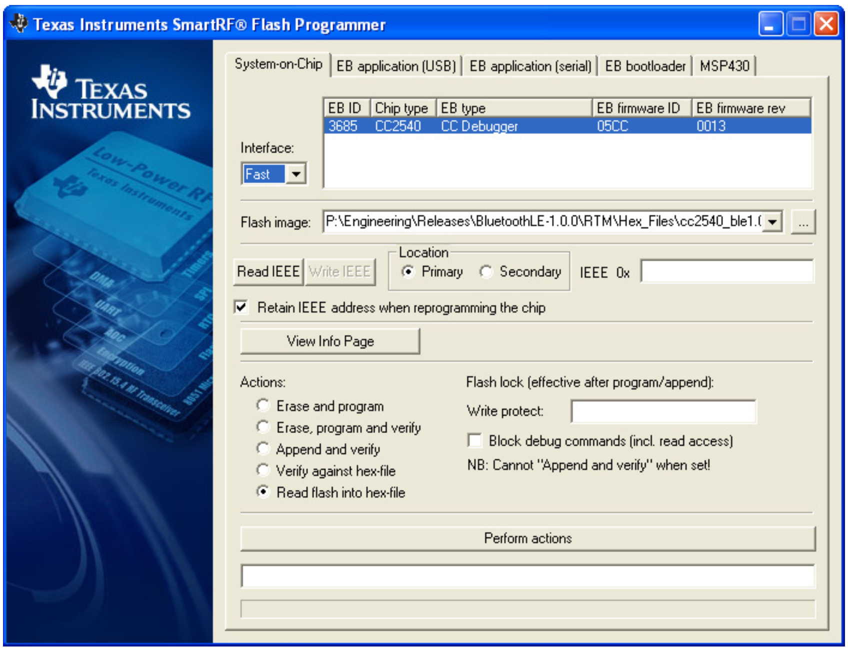

Figure 42

5.3.1 Checking the CC Debugger Firmware

Note: this step is only required when using a CC Debugger that was purchased separate from the CC2540DK-MINI kit, or one that was included with an older Texas Instruments development kit. If you are using the CC Debugger contained within the CC2540DK-MINI kit, this step should not be necessary.

The firmware of the CC Debugger can be seen clicking the "EB Application (USB)" tab in the SmartRF Flash Programmer window, and viewing the "EB firmware rev" value in the device list.

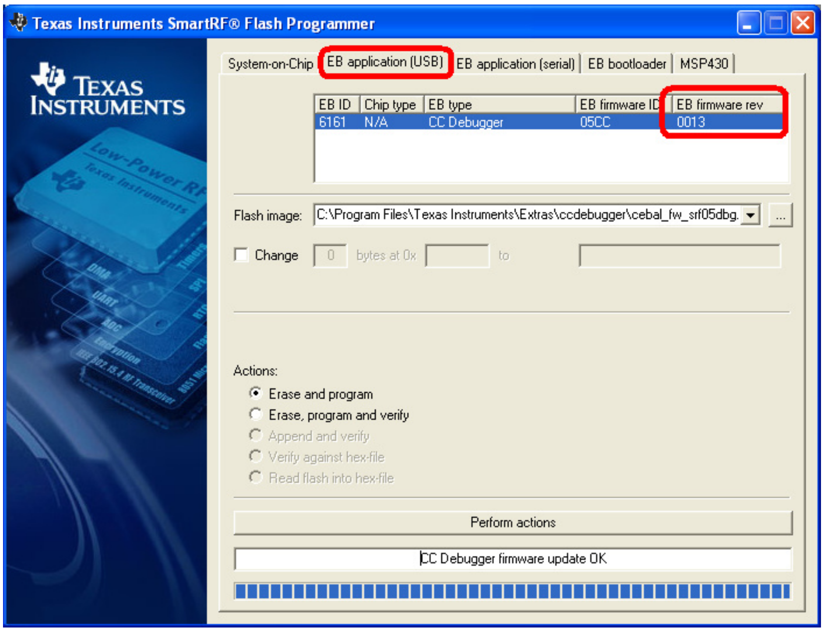

Figure 43

The value shown must be "0008" or higher; otherwise the debugger will not recognize the CC2540. If the firmware is "0007" or less, the firmware will need to be updated.

To update the firmware, click the "…" button next to the "Flash image" text box, and select the following file:

C:\Program Files\Texas Instruments\Extras\ccdebugger\cebal_fw_srf05dbg.hex

With the "Erase, program and verify" selected under "Actions", click the "Perform actions" button. This will update the CC Debugger firmware to version "0013".

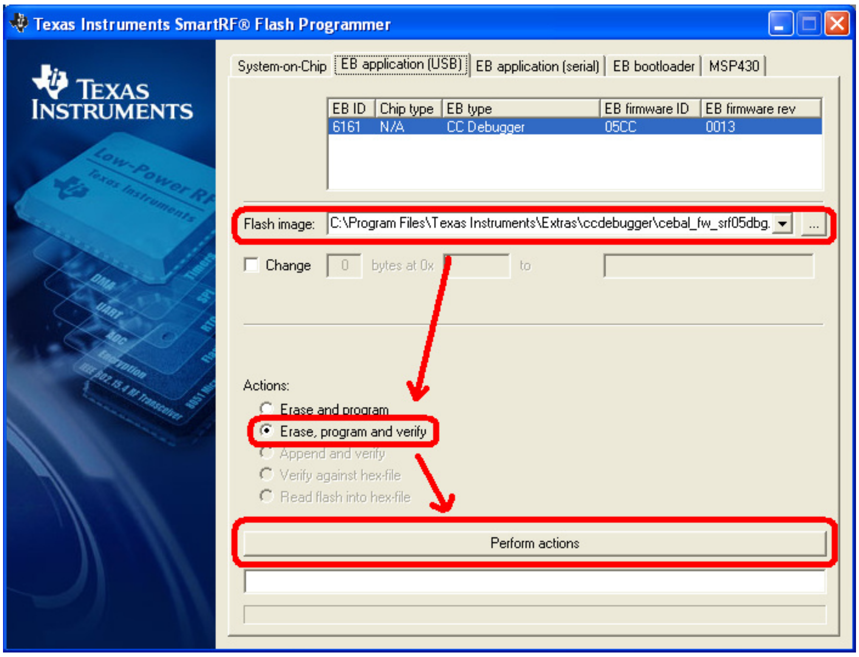

Figure 44

The process will take around 10 seconds. Once complete the "EB firmware rev" value should change to "0013". The debugger should now be able to recognize the CC2540.

5.3.2 Reading or Writing a Hex File to the CC2540

To read or write a hex file to the CC2540, select the "System-on-Chip" tab. The connected CC2540 should be detected and show up in the list of devices. Under "Flash image" select the desired hex file that you would like to write to the device. If you are reading from the CC2540, under "Flash image" enter the desired path and filename for the hex file. To write to the CC2540, under "Actions" select "Erase, program and verify". To read from the CC2540, under "Actions" select "Read flash into hex-file". To begin the read or write, click the button "Perform actions".

If the action completes successfully, you should see the progress bar at the bottom of the window fill up, and either one of the following two messages, depending on whether a write or a read was performed: "CC2540 - ID2000: Erase, program and verify OK" or "CC2540 - ID2000: Flash read OK".

You may see the following error message:

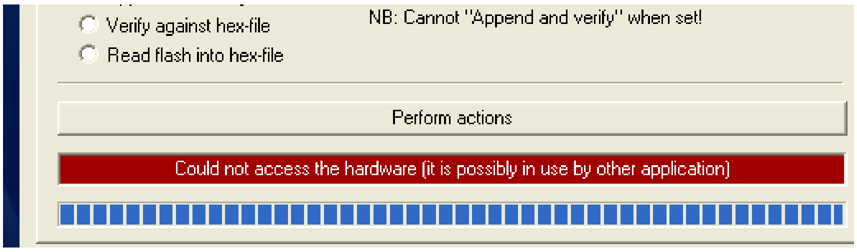

Figure 45

If this comes up, it most likely means that you have IAR open and are debugging. You will need to stop debugging before you can use SmartRF Flash Programmer to communicate with the CC Debugger.

5.3.3 Reading or Writing the CC2540 Device Address

Every CC2540 device comes pre-programmed with a unique 48-bit IEEE address. This is referred to as the device's "primary address", and cannot be changed. It is also possible to set a "secondary address" on a device, which will override the primary address upon power-up. SmartRF Flash Programmer can be used to read the primary address, as well as to read or write the secondary address.

To read back the primary address of a device connected to the CC Debugger, select “Primary” under the “Location” option, and click the “Read IEEE” button. The primary device address should appear in the box on the right. Click the “Perform Actions” button at the bottom to perform the read.

To read back the secondary address, select “Secondary” under the “Location” option, and click the “Read IEEE” button. The secondary device address should appear in the box on the right. Click the “Perform Actions” button at the bottom to perform the read.

To set a new secondary address, select “Secondary” under the “Location” option, and enter the desired address in the box on the right. Click the “Perform Actions” button at the bottom to perform the write. If the secondary device is set to “FF FF FF FF FF FF”, the device will use the primary address. If the secondary device is set to anything else, the secondary address will be used.

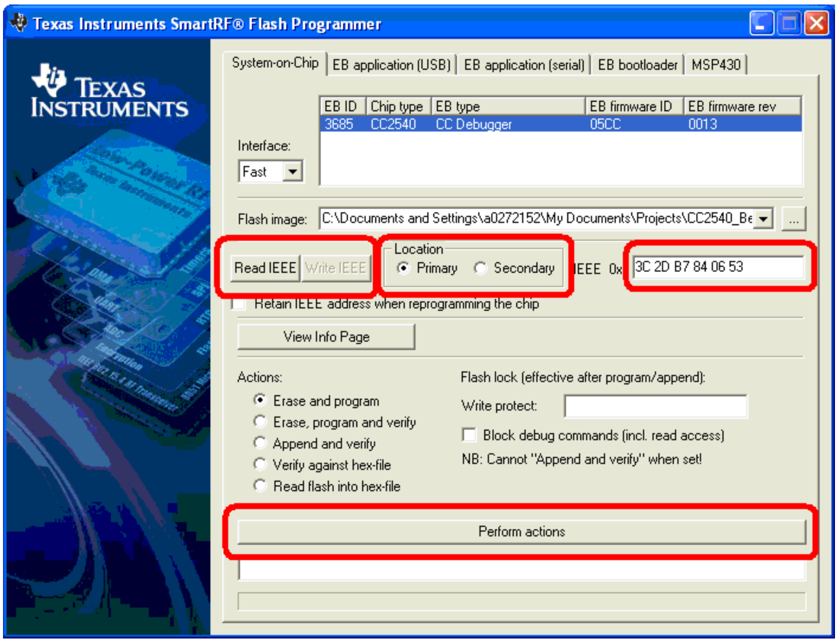

Figure 46

Note that every time you re-program the device using SmartRF Flash Programmer, the secondary address of the device will get set to FF:FF:FF:FF:FF:FF. This can be avoided by selecting the option “Retain IEEE address when reprogramming the chip”. A similar situation exists when a device is reprogrammed through IAR Embedded Workbench, in that the secondary address will get set to FF:FF:FF:FF:FF:FF each time. To avoid this, the IAR option “Retain unchanged memory”, under the “Debugger” > “Texas Instruments” project option can be selected.

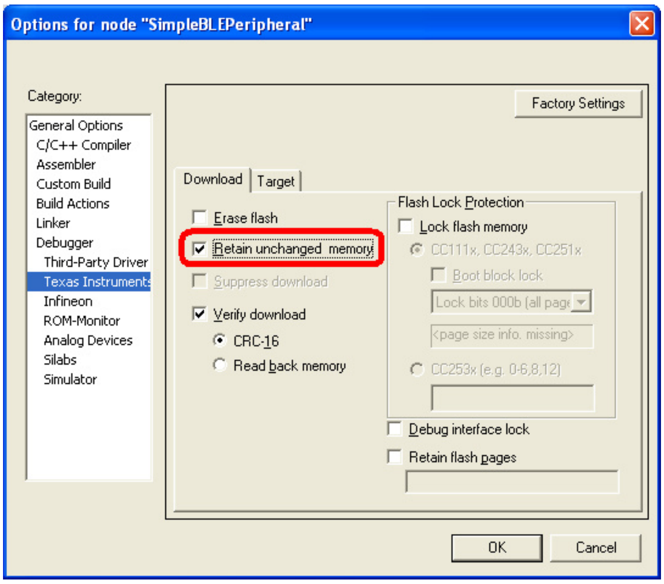
Options for node "SimpleBLEPeripheral"
Category:
General Options
C/C++ Compiler
Assembler
Custom Build
Build Actions
Linker
Debugger
Third-Party Driver
Texas Instruments
Infineon
ROM-Monitor
Analog Devices
Silabs
Simulator
Factory Settings
Download
Target
Erase flash
Retain unchanged memory
Suppress download
Verify download
CRC-16
Read back memory
Flash Lock Protection
Lock flash memory
CC111x, CC243x, CC251x
Boot block lock
Lock bits 000b (all page
<page size info. missing>
CC253x (e.g. 0-6,8,12)
Debug interface lock
Retain flash pages
OK
Cancel

Figure 47

6. SmartRF™ Packet Sniffer

The SmartRF™ Packet Sniffer is a PC software application used to display and store RF packets captured with a listening RF hardware node. Various RF protocols are supported, included BLE. The Packet Sniffer filters and decodes packets and displays them in a convenient way, with options for filtering and storage to a binary file format.

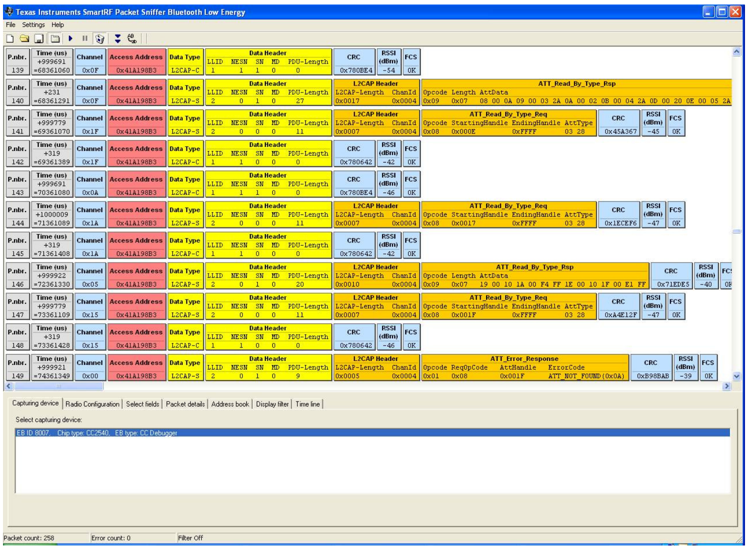

Figure 48

The USB Dongle included with the CC2540 Mini Development Kit can be used as the listening hardware node, and can be useful when debugging BLE software applications. The SmartRF™ Packet Sniffer software can be downloaded at the following link:

http://focus.ti.com/docs/toolsw/folders/print/packet-sniffer.html

7. General Information

7.1 Document History

Revision	Date	Description/Changes
1.0	2010-10-08	Initial release
1.0.1	2010-11-29	Added information about packet sniffer and KeyFobDemo application
1.1	2011-07-13	Updated with information from BLEv1.1 software release