Chapter 10: Debugging

After completing this chapter, you will have gained a basic understanding of Host Controller Interface (HCI) commands and WICED HCI format. You will then learn how to enable WICED HCI trace messages from the Bluetooth® stack and view them using the BTSpy program.

Table of contents

Document conventions

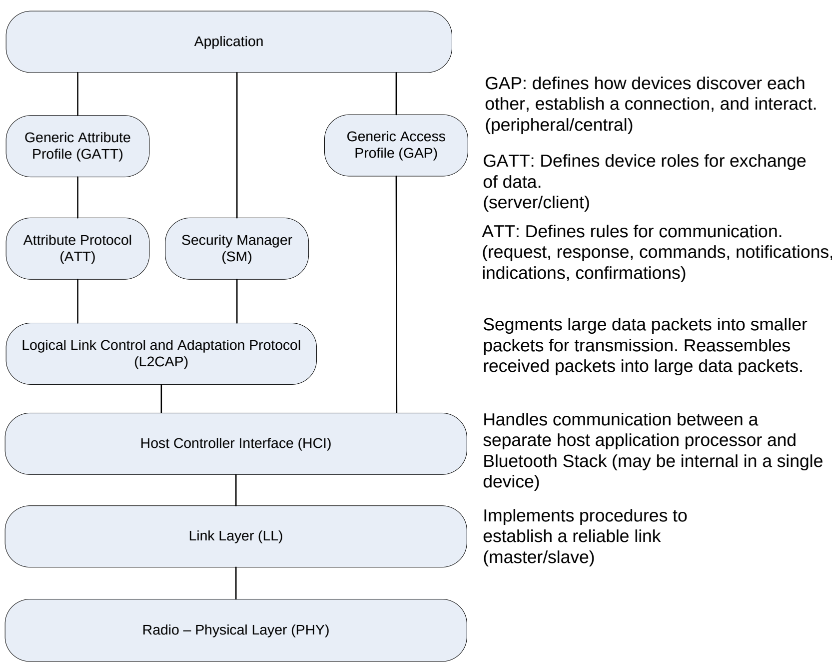
Convention	Usage	Example
`Courier New`	Displays code and text commands	`CY_ISR_PROTO(MyISR);` `make build`
Italics	Displays file names and paths	*sourcefile.hex*
[**bracketed, bold**]	Displays keyboard commands in procedures	[**Enter**] or [**Ctrl**] [**C**]
Menu > Selection	Represents menu paths	**File > New Project > Clone**
Bold	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the **Debugger** icon, and then click **Next**.

10.1 Arm debug port

Hardware debugging on Bluetooth® applications can done using an OpenOCD supported probe just like on any other PSoC™ 6 application. However, if your device has an active Bluetooth® LE connection it will drop the connection when the CPU is halted in the debugger (after the connection timeout). Therefore, for Bluetooth® LE applications it is more common to use WICED HCI and an application like BTSpy for debugging rather than a hardware debugger. Unlike traditional hardware debugging, WICED HCI provides debug messages while the Bluetooth® connection stays active. That's what the rest of this chapter covers.

10.2 Host Controller Interface (HCI)

In many complicated systems, hierarchy is used to manage the complexity and Bluetooth® is one of them. The Bluetooth® stack is called a stack because it is a set of blocks that have well defined interfaces. Here is a simple picture of the software system that we have been using. You have been writing code in the block called "Application." You have made API calls and gotten events from the "Attribute Protocol" and you implemented the "Generic Attribute Profile" by building the GATT Database. Moreover, you advertised using GAP and you Paired and Bonded by using the Security Manager.

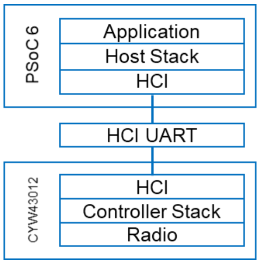


The next block to talk about is the "Host Controller Interface." In many Bluetooth® systems, the radio is a separate chip from the one that runs the upper levels of the stack and the application. The radio chip takes the name of controller because it contains the radio and radio controller while the chip running the upper level of the stack and the application is called the host because it hosts the application. The interface between the host and controller is therefore called the Host Controller Interface, or HCI for short.

By standardizing the HCI, it allows big application processors (like those exiting in PCs and cellphones) to interface with Bluetooth®.

Sometimes host and controller are integrated into one chip but HCI persists even though both sides may be physically on the same chip. In that case, the HCI layer is essentially just a pass-through. Other times, the

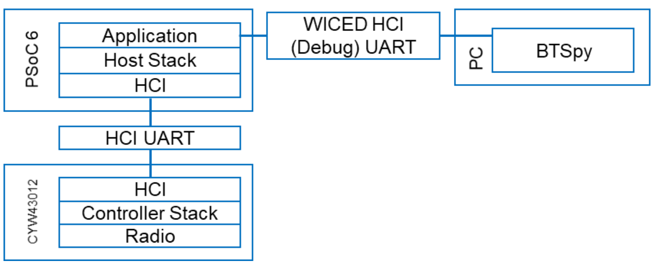
Bluetooth® stack can is split into separate host and controller parts. For example, the PSoC™ 6 and CYW43012 combo radio that we have been using is a 2-chip solution that looks like this:



In this case, a dedicated UART interface is used to send HCI messages back and forth between the PSoC™ 6 and CYW43012.

10.3 WICED HCI

The HCI concept was extended by the Infineon software team to provide a means of mirroring the HCI traffic between the devices to allow debugging of the Bluetooth® interface while it is running. This interface is called "WICED HCI." Normally you send WICED HCI messages to the debug UART interface that connects between the PSoC™ 6 and a PC. The result is that from a PC you can view your standard debug print messages along with all HCI messages between the host and controller devices.



10.3.1 Message Format

WICED HCI is a packet-based format for PC applications to interact with the application running on the host. The packets have 3 standard fields plus an optional number of additional bytes for payload. The packet looks like this:

- 0x19 – the initial byte to indicate a WICED HCI packet
- 2-byte little endian Opcode consisting of a 1-byte Control Group (a.k.a. Group Code) and a 1-byte Sub-Command (a.k.a. Command Code)
- A 2-byte little endian length of the additional bytes
- An optional number of additional bytes for payload

The Control Group is one of a predefined list of categories of transactions including `Device=0x00`, `BLE=0x01`, `GATT=0x02`, etc. These groups are defined in *mtb_shared/btstack/<version>/wiced_include/hci_control_api.h*. Each Control Group has one or more optional Sub-Commands. For instance, the Device Control Group has Sub-Commands `Reset=0x01`, `Trace Enable=0x02`, etc.

The Control Group plus the Sub-Command together is called a "Command" or an "Opcode" and is a 16-bit number. For example, Device Reset = 0x0001. In the actual data packet, the Opcode is represented little endian. For example, the packet for a Device Reset = 19 01 00 00 00.

10.4 WICED HCI debug firmware architecture

In order to enable WICED HCI debug traces, there are just three things that need to be done in the application: (1) enable the WICED HCI logging feature in the stack; (2) include the appropriate header file; and (3) setup the debug UART to send messages in the WICED HCI format.

10.4.1 Enable logging

The first step is to enable BTSpy logging from the stack. This is done by setting a define in the application's *Makefile*:

```
DEFINES=ENABLE_BT_SPY_LOG
```

10.4.2 Include WICED HCI header file

The following file must be included to get access to the WICED HCI functions:

```
#include "cybt_debug_uart.h"
```

10.4.3 Setup debug UART

When using WICED HCI, the debug UART needs to be initialized using the function `cybt_debug_uart_init`. This can be done in initialization immediately after calling `cybt_platform_config_init`. The first argument is a pointer to a configuration structure of type `cybt_debug_uart_config_t` where you specify the pins, baud rate, and flow control setting. The second argument is an optional callback function in case you want to do your own processing when a debug message is received.

You can add the new code conditionally so that the debug UART uses the standard Retarget-IO functionality if BTSpy logging is not enabled in the *Makefile.*

```
#ifdef ENABLE_BT_SPY_LOG
      cybt_debug_uart_config_t config =
      {
            .uart_tx_pin = CYBSP_DEBUG_UART_TX,
            .uart_rx_pin = CYBSP_DEBUG_UART_RX,
            .uart_rts_pin = CYBSP_DEBUG_UART_RTS,
            .uart_cts_pin = CYBSP_DEBUG_UART_CTS,
            .baud_rate = 3000000,
            .flow_control = true,
      };
      cybt_debug_uart_init(&config, NULL);
#else
      cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
                          CY_RETARGET_IO_BAUDRATE);

#endif
```

Note: The BTSpy application supports baud rates of 115200 or 3000000 but the higher value is recommended.

Optionally, you can call the function `cybt_platform_set_trace_level` to change which blocks and which severity of messages are printed (e.g. info, warning, error). The available trace IDs and levels can be found in *mtb_shared/bluetooth-freertos/<version>/platform/include/cybt_platform_trace.h*. The default values are `CYBT_TRACE_ID_STACK` and `CYBT_TRACE_LEVEL_DEBUG`.

If you are going to call `cybt_platform_set_trace_level`, you must include an additional header file:

```
#include "cybt_platform_trace.h"
```

10.5 BTSpy Application

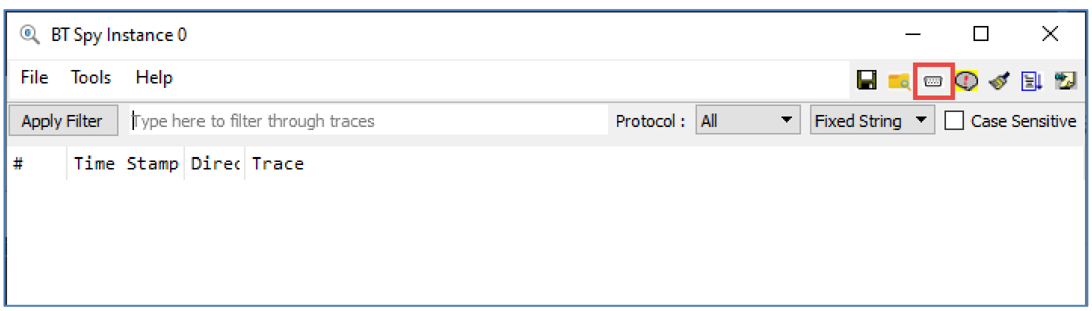
The BTSpy application is available in a Bluetooth® utilities library on GitHub. To get the library, open a terminal (e.g. `modus-shell` on Windows) and execute this command from the directory where you want the library to be downloaded:

```
git clone https://github.com/Infineon/btsdk-utils.git
```

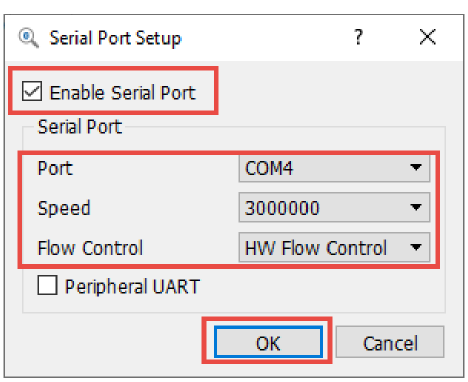
Once it is cloned, go to the directory *btstk-utils/BTSpy*. You will see subdirectories for Linux64, OSX, and Windows versions of the tool. The screen shots below are from the Windows version.

In the *Windows* directory, the executable is *BTSPy.exe*. It can be run directly from the terminal (`./BTSpy.exe`) or by double-clicking on it from Windows explorer.

Once it starts up, you will see a window like this:



The first thing to do is to setup the UART connection. Click the **Serial Port Setup** button (see the red box in the figure above) or use the menu item **Tools > Serial Port Setup**. Check the box to **Enable Serial Port**, select the correct port for your kit, set the baud rate, and enable hardware flow control. Click **OK** when you are done.



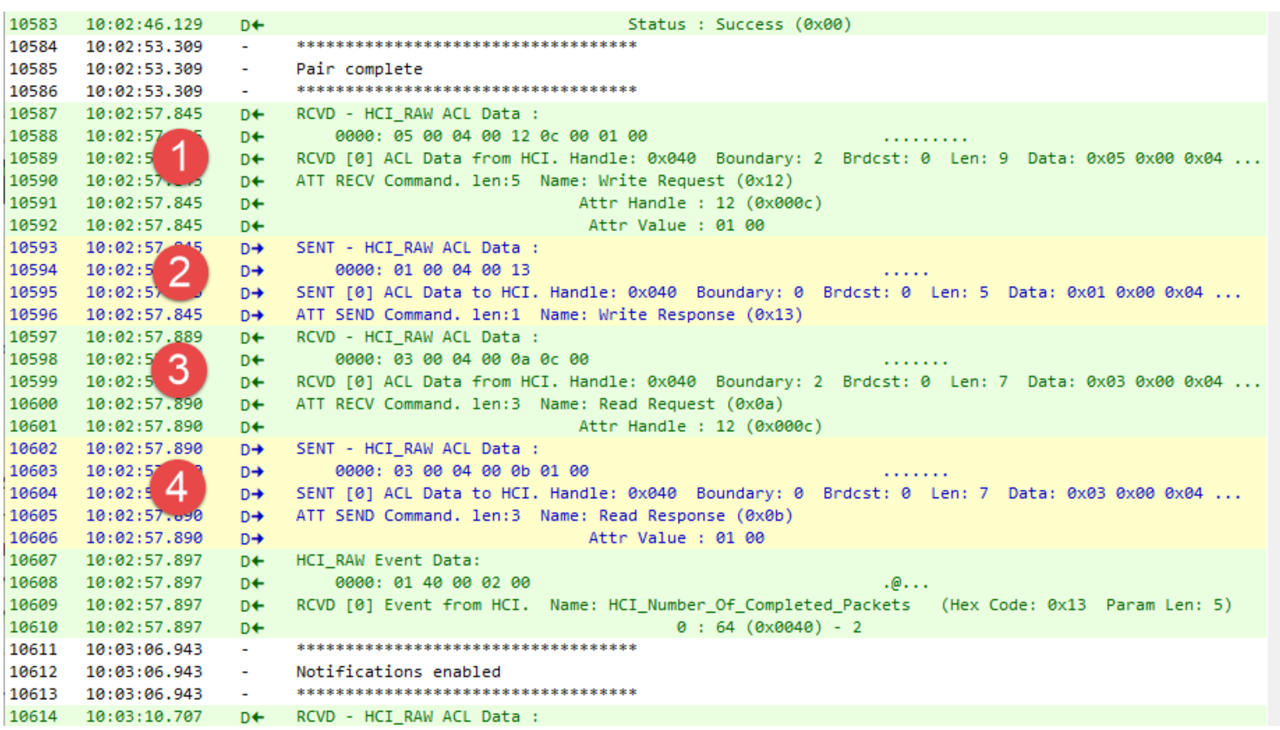
Note: To disconnect from the kit, you must open the serial port setup window again, uncheck the Enable Serial Port button, and click OK.

As messages are received from the kit, you will see green messages for incoming Bluetooth® traffic, yellow for outgoing Bluetooth® traffic, and white for non-HCI messages.

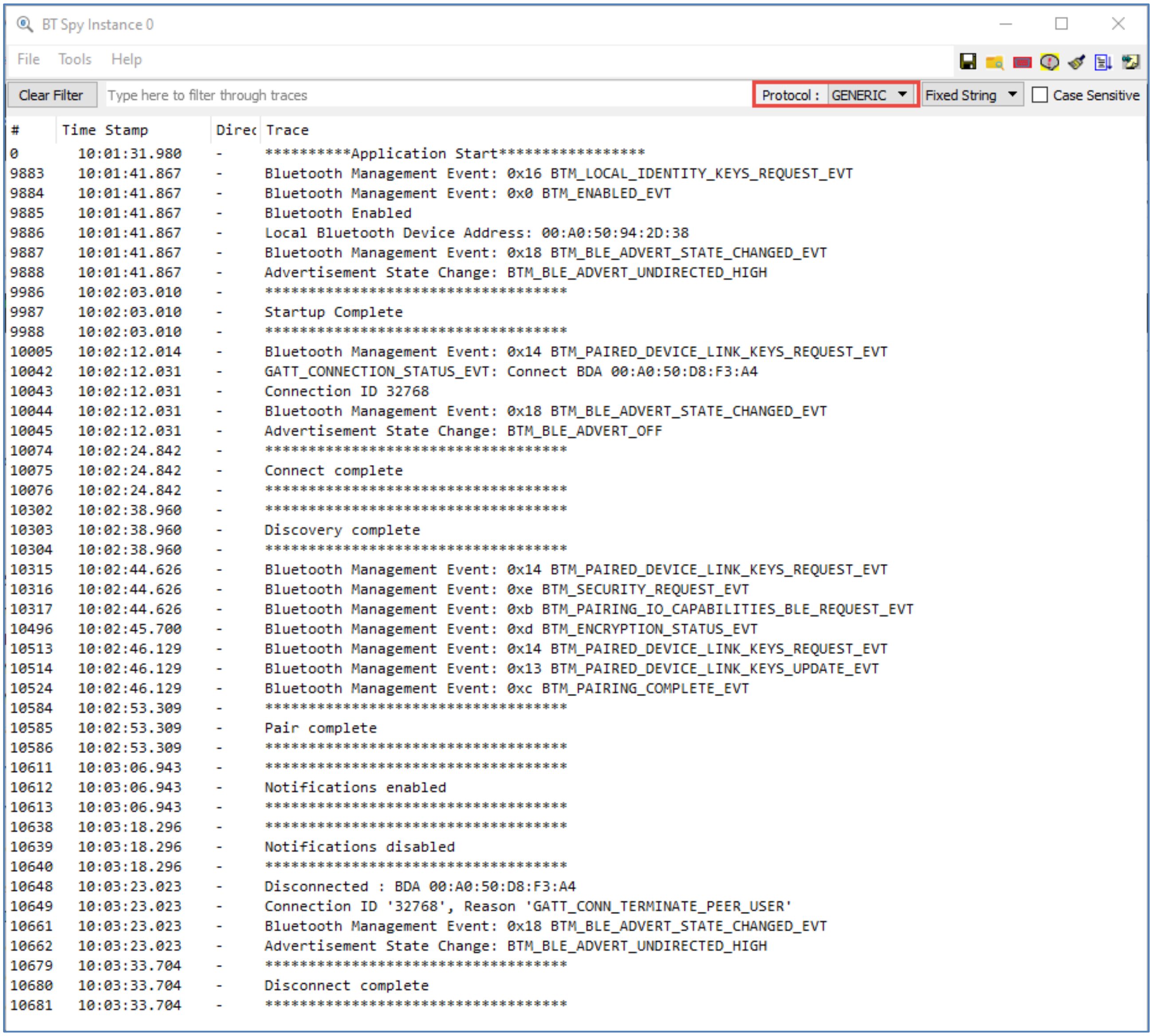
A few other controls of interest are:

Command	Icon	Use
Clear Trace		Clear the log history
Note		Allows you to enter a note that shows up in the log
File Logging Options		Setup and start/stop logging messages to a file
Open Logged File		Open a previous log file in the tool
Protocol	Protocol : All	Allows you to filter out messages of certain types (ALL, HCI, GENERIC, etc.)

The following figure shows an example of a log that was created for a Bluetooth® peripheral that allows pairing and has notifications. The lines shown with starts above/below were added using the Note function (the stars are added automatically by the tool). As you can see, after pairing is complete, the user enables notifications from the central. To do this, the central sends a GATT write request (1) to handle 0x000C (which is the handle for the CCCD) and a value of 01 00 (which is the code to enable notifications). The peripheral responds to the write request (2). Next, the central sends a read request (3) to the same handle and the peripheral sends back a read response (4) with the new value of the CCCD of 01 00. This allows you to see exactly what is being sent back and forth across the Bluetooth® link which is extremely helpful during debugging.



If you change the protocol filter from ALL to GENERIC, you will see just the messages that are printed from the application along with the notes that were added manually.



10.6 Exercises

Exercise 1: Add WICED HCI traces to pairing application and use BTSpy to test

In this exercise, you will start from the Bluetooth® pairing exercise from a previous chapter and you will add in WICED HCI trace capability. You will then use BTSpy to look at the application messages and HCI messages.

- [ ] 1. Create a new application for the BSP you are using.

  On the application template page, use the **Browse** button to start from the completed application for the *ch05_ex01_pair* exercise. If you did not complete that exercise, the solution can be found in *Projects/key_ ch05_ex01_pair*.
  Name the new application *ch10_ex01_btspy*.
- [ ] 2. Open the Bluetooth® configurator and on the GAP settings tab change the device name to `<inits>_btspy`.
- [ ] 3. Follow the instructions in section 10.4 to enable WICED HCI logging.
- [ ] 4. Program your application to the kit.
- [ ] 5. If you have a UART terminal open, close it.
- [ ] 6. If you don't already have it, download the *btsdk-utils* repo from GitHub as described in section 10.5.
- [ ] 7. Run the BTSpy application. Configure the serial port for your kit and enable it.
- [ ] 8. Clear the log and then reset the kit to observe the messages.
- [ ] 9. When startup has completed (advertising has started) enter a note that says "Startup Complete."
- [ ] 10. Open AIROC™ Connect and connect to your device.
- [ ] 11. Once the connection has been made, enter a note in BTSpy that says "Connect complete."
- [ ] 12. Click the GATT DB widget followed by the Unknown Service and then the Characteristic that has Read & Notify properties. This is the Counter Characteristic that counts button presses.
- [ ] 13. Accept the pairing request on the mobile device. Once it completes, enter a note in BTSpy that says "Pairing Complete."
- [ ] 14. Click the **Notify** button in AIROC™ Connect. Once it completes, enter a note in BTSpy that says "Notifications Enabled."
- [ ] 15. Scroll through the BTSpy log messages to see what is going on during each operation. You will see the notes that you added to be able to tell when each operation starts/stops.
- [ ] 16. Use the **Protocol** dropdown to select **GENERIC**. Now you will see just the informational messages from your application along with the notes that you added.
- [ ] 17. Disconnect and close AIROC™ Connect. Close BTSpy.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Published by
Infineon Technologies AG
81726 Munich, Germany