AN223400

Getting Started with EZ-BT WICED Modules

Author: David Solda

Associated Project: Yes

Associated Part Family: CYBT-XXXXXX-0X

Software Version: WICED® Studio™ SDK v6.2

Related Application Notes: For a complete list of the application notes, click here.

AN23400 introduces you to Cypress' EZ-BT™ WICED® family of Bluetooth modules. EZ-BT modules are fully qualified and certified Bluetooth solutions, supporting Bluetooth Basic Rate, Enhanced Data Rate, and Bluetooth Low Energy (BLE) standards. These modules provide a complete Bluetooth solution, integrating a Bluetooth radio system, crystal, antenna, and passive components required for Bluetooth operation. This application note helps you explore the EZ-BT Module architecture and development tools and shows you how to create your first project from existing sample application projects within WICED Studio SDK. This application note also guides you to more resources to accelerate in-depth learning about EZ-BT WICED solutions.

Contents

1 Introduction

Bluetooth is a wireless standard that operates in the 2.4-GHz ISM band, originally created as an RS-232 cable replacement technology. Bluetooth technology and applications have evolved significantly since the initial Bluetooth v1.0 specification release in 1999.

- 1999~2004: Bluetooth applications were focused on short-range wireless cable replacement (e.g., printers, PC mice, and keyboards).
- 2004~2011: Bluetooth Special Interest Group (SIG) adopts Bluetooth Core Specification 2.0, introducing Enhanced Data Rate, capable of transmitting up to 3 Mbits/s, opening up one of the most dominant applications for Bluetooth technologies: audio streaming. The first Bluetooth-enabled stereo headset was released in the same year. From 2004 to 2011, the number of Bluetooth-enabled products and mobile/ smartphones continued to expand within these application segments.
- 2011~2017: Bluetooth SIG adopts Core Specification v4.0, which introduced Bluetooth Low Energy (BLE), also known as Bluetooth Smart. The introduction of Bluetooth Smart provided a lower-power alternative to the prior Bluetooth Classic (BR and EDR) technologies, opening a wide array of applications in what is now known as the IoT (Internet of Things). Bluetooth Low Energy has continued to evolve since its introduction, adding increased security, data rates, and improved privacy for users.
- 2017: Mesh networking capability is added to Bluetooth, enabling products to increase the scale and range for Bluetooth applications.

Cypress' Bluetooth portfolio consists of Bluetooth Low Energy (BLE)-only and dual-mode Bluetooth solutions that support Bluetooth Classic (i.e., Basic Rate (BR) and Enhanced Data Rate (EDR)) as well as BLE. This application note will focus on the Cypress dual-mode Bluetooth module portfolio, which includes Bluetooth 5.0 qualified, BR + EDR + BLE devices that integrate Bluetooth standard profiles and protocols for embedded applications.

WICED (pronounced “wik-id”) is Cypress’ IoT platform that enables rapid development and deployment of connected IoT products. Wireless Internet Connectivity for Embedded Devices (WICED) in conjunction with EZ-BT modules provides a great feature set to simplify development and release of Bluetooth-enabled products by eliminating the complexity of wireless RF hardware design, allowing customers to focus on their IoT product development.

The WICED Studio™ SDK is pre-integrated, pre-tested, and continuously updated, containing:

- WICED APIs and drivers to make wireless connectivity easy and flexible
- Proven production-ready stacks (e.g., networking, security)

Cypress’ EZ-BT modules accelerate your time-to-market, by providing:

- Proven RF solutions, reducing hardware development and debugging
- Fully qualified and certified solutions, decreasing costs and cycle times associated with RF standards
- Multiple module options to maximize your system design flexibility

EZ-BT WICED Modules enable quick time-to-market by eliminating time-consuming and costly RF hardware development, certification, and qualification processes, offering an effective alternative to completing a Bluetooth system design from the ground up.

The EZ-BT WICED Module family provides fully integrated, qualified, and certified Bluetooth systems that integrate a 24-MHz crystal oscillator, passive components, on-board or on-chip memory, on-board chip or trace antennas, and the WICED Bluetooth silicon device, which includes the Bluetooth radio, analog-to-digital converter inputs, PWM control, serial communication protocols (I^2C, SPI, UART), audio interfaces (I^2S, PCM, PDM), and an Arm® Cortex®-M3 or M4 microcontroller.

EZ-BT WICED Modules provide a cost-effective solution for sensor-based IoT solutions, while still achieving world-class RF performance by utilizing the latest WICED Bluetooth silicon devices.

2 More Information

This section provides a list of learning resources for the EZ-BT Module family that can help you to get started and develop complete applications with your EZ-BT Module. All documentation can be accessed at the Cypress Bluetooth Module website.

2.1 EZ-BT WICED Module Datasheets

EZ-BT WICED Module datasheets list the features, pinouts, device-level specifications, and fixed-function peripheral electrical specifications of EZ-BT WICED Modules. Click on the **Documentation** tab of the Cypress Bluetooth Module website to access all datasheets for EZ-BT WICED Modules.

2.2 EZ-BT WICED Evaluation Boards

Each EZ-BT WICED Module offers a low-cost Arduino-compatible evaluation board, providing an easy-to-use vehicle to develop and evaluate EZ-BT WICED Modules without requiring custom hardware design. Each evaluation board is capable of interfacing to Arduino-compatible shields. Click on the **Kits** tab on the Cypress Bluetooth Module website to view and purchase Cypress EZ-BT evaluation boards.

2.3 Silicon Device Datasheet

Cypress WICED Bluetooth datasheets lists the features, pinouts, device-level specifications, and fixed-function peripheral electrical specifications of all Cypress WICED Bluetooth devices. Datasheets for applicable WICED Bluetooth devices discussed in this application note can be found at the following links:

- CYW20706 Bluetooth SoC for Embedded Wireless Devices
- CYW20707 Bluetooth SoC for Embedded Wireless Devices
- CYW20719 Enhanced Low Power, BR/EDR/BLE Bluetooth 5.0 SOC

2.4 Cypress WICED Bluetooth Community

Whether you're a customer, partner, or a developer interested in the latest Cypress innovations, the Cypress WICED Bluetooth Community offers you a place to learn, share and engage with both Cypress experts and other embedded engineers around the world.

2.5 Application Notes

Application notes assist you with understanding specific features of your device for designing your Bluetooth application.

2.6 Technical Support

If you have any questions, our technical support team is happy to assist you. You can create a support request by visiting Cypress Technical Support.

If you are in the United States, you can talk to our technical support team by calling our toll-free number: +1-800-541-4736. You can also use the following support resources if you need quick assistance.

- Self-help
- Local sales office locations

3 EZ-BT WICED Module Overview

This application note introduces the reader specifically to EZ-BT WICED Module solutions and how to get started.

EZ-BT WICED Modules offer fully integrated and fully certified Bluetooth solutions allowing rapid development and deployment of your Bluetooth-enabled product. This section provides an overview of EZ-BT WICED Modules available today. For detailed information on each module referenced in this section, see Appendix B: EZ-BT Module Product Details.

All EZ-BT WICED Modules ship with all required components to achieve full Bluetooth functionality, including:

- PCB substrate
- Cypress WICED Bluetooth IC
  Refer to the Module datasheet for references and links to the datasheet of the silicon used in each module.
- Crystal oscillators: 24-MHz external crystal oscillator
- EZ-BT WICED Modules utilize an integrated low-power oscillator on the silicon device and do not contain a physical 32-kHz external crystal oscillator. Each module provides a connection option for an external 32-kHz input if required.
- On-module or on-chip flash memory
- Chip or Trace antenna
- Passive components (resistor, capacitor, inductor)
- RF Shield, unless otherwise noted

3.1 EZ-BT WICED Module Family Features

Table 1 summarizes the features and capabilities that EZ-BT WICED Modules offer.

Table 1. EZ-BT WICED Module Features and Capabilities

Features	Details
Bluetooth Subsystem	Bluetooth BR/EDR + BLE radio and link-layer hardware
CPU	Arm Cortex-M3 or Arm Cortex-M4 32-bit processor
Flash Memory	512 KB or 1 MB (module-dependent)
SRAM	Up to 512 KB (module-dependent)
ROM	Up to 2 MB ROM, containing Bluetooth stack and specific Bluetooth profiles
GPIOs	Up to 17 (module-dependent)
CapSense®	None (See Getting Started with EZ-BLE Creator Modules for modules that include this functionality)
CapSense® Gestures	
ADC	10-bit ENoB for DC measurement 12-bit ENoB for Audio measurement
Opamps	None (See Getting Started with EZ-BLE Creator Modules for modules that include this functionality)
Comparators	
Current DACs	
Power Supply Range	1.62 V to 3.6 V (module-dependent)
Low-Power Modes	HIDOFF mode as low as 400 nA typical (CYW20719-based modules) Deep Sleep mode as low as 61µA with RAM retention (CYW20719-based modules)
Serial Communication	I^2C, SPI, Peripheral-UART (application interface), HCI-UART (programming)
Audio Interface	I^2S, PCM, PDM (module-dependent)
Pulse-Width Modulator (PWM)	Up to 6
Universal Digital Blocks (UDBs)	None (See Getting Started with EZ-BLE Creator Modules for modules that include this functionality)
Clocks	32-kHz LPCLK (Low-power clock)
Power Supply Monitoring	Power-on reset (POR)
Integrated Crystal Oscillators	24-MHz integrated on module 32-kHz connection available (optional)
Antenna Type	Trace or Chip Antenna (module dependent)
Certifications	FCC, ISED, MIC, CE, unless otherwise noted in the datasheet Each EZ-BT WICED Module has a Cypress Knowledge Base Article, which contains the regulatory testing reports and certificates for all countries the module is certified against. See the **More Information** section of the module datasheet for links to this information or refer to the Certifications tab available online at www.cypress.com/cypress_bluetooth_modules.

3.2 EZ-BT WICED Module Low-Power Modes

EZ-BT WICED Modules support a variety of low-power modes, depending on the specific module selected. For details on the current consumption and supported power modes for each module solution, see the module datasheet. The power modes shown below are supported on the CYBT-423028-02 module:

- **Active mode:** Normal operating mode in which all peripherals are available, and the CPU is active.
- **Pause mode:** In this mode, the CPU is in "Wait for Interrupt" (WFI) and the HCLK, which is the high-frequency clock derived from the main crystal oscillator, is running at a lower clock speed. Other clocks are active and the state of the entire chip is retained. Pause mode is chosen when other low-power modes are not possible.
- **PMU Sleep mode:** In this mode, the CPU is in WFI and the HCLK is not running. The Power Management Unit (PMU) determines whether other clocks can be turned OFF and does so accordingly. The state of the entire chip is retained, the internal LDOs run at a lower voltage (voltage is managed by the PMU), and SRAM is retained.
- **Power Down Sleep (PDS) mode:** This mode is an extension of the PMU Sleep wherein most of the peripherals such as UART and SPI are turned OFF. The entire memory is retained, and on wakeup the execution resumes from where it paused.
- **Shut Down Sleep (SDS):** Everything is turned OFF except the I/O Power Domain, Real Time Clock (RTC), and LPO (Low-Power Oscillator). The device can come out of this mode either due to BT activity or by an external interrupt. Before going into this mode, the application can store up to 256 bytes of data into "Always On RAM" (AON). When the device comes out of this mode, the data from AON is retained. After waking from SDS, the application will start from the beginning (warm boot) and will restore its state based on information stored in AON. In the SDS mode, a single BT task with no data activity, such as an ACL connection, BLE connection, or BLE advertisement can be performed. The following guidelines and restrictions in usage apply to SDS mode.
  - The device address type should only be public.
  - The connection interval should be greater than 100 ms.
  - If the device is doing connectable advertisement in SDS mode, two connection requests need to be sent to the device to connect to it: the first one to wake the device and the second one to make an actual connection. If the device is already connected, there is no need to send any second connection request.
  - The pairing process cannot be done in SDS mode.
  - High-duty-cycle advertisement with a low interval will not allow the device to enter SDS. The device enters SDS mode immediately when the device is configured for low-duty-cycle advertisements without any advertisement time out (time out is set for infinity). If the low-duty-cycle advertisement is configured for a finite time, the device will enter SDS only after this time expires (i.e., when the advertisement stops).
  - SDS is allowed only if the next activity is more than 100 ms away; otherwise PDS mode is used as frequent wakeup from SDS will consume more power than staying in PDS mode.
  - To get a GPIO interrupt callback upon wake up, the interrupt callback needs to be registered in the application.
- **HIDOFF (Timed-Wake) mode:** The device can enter this mode asynchronously, that is, the application can force the device into this mode at any time. I/O Power Domain, RTC, and LPO are the only active blocks. A timer that runs off the LPO is used to wake the device up after a predetermined fixed time. The mode can also be woken up via an external interrupt.
- **HIDOFF (External Interrupt-Waked) mode:** This mode is similar to Timed-Wake, but in HID-off mode, even the LPO and RTC are turned OFF. So, the only wakeup source is an external interrupt.

Table 2. EZ-BT Module MPN Features and Capabilities

Marketing Part Number	Silicon Device	Active	Pause	PMU Sleep	PDS	SDS	Timed Wake and HID-OFF
CYBT-343026-01	CYW20706	✓		✓	✓		✓
CYBT-353027-02	CYW20707	✓		✓	✓		✓
CYBT-423028-02	CYW20719	✓	✓	✓	✓	✓	
CYBT-413034-02	CYW20719	✓	✓	✓	✓	✓	
CYBT-483039-02	CYW20719	✓	✓	✓	✓	✓	

3.3 EZ-BT WICED Part Number Overview

Each device within the EZ-BT WICED Module family has a unique Marketing Part Number (MPN) used for ordering. The MPN format is shown in Figure 1.

Figure 1. EZ-BT Module Marketing Part Numbering Format

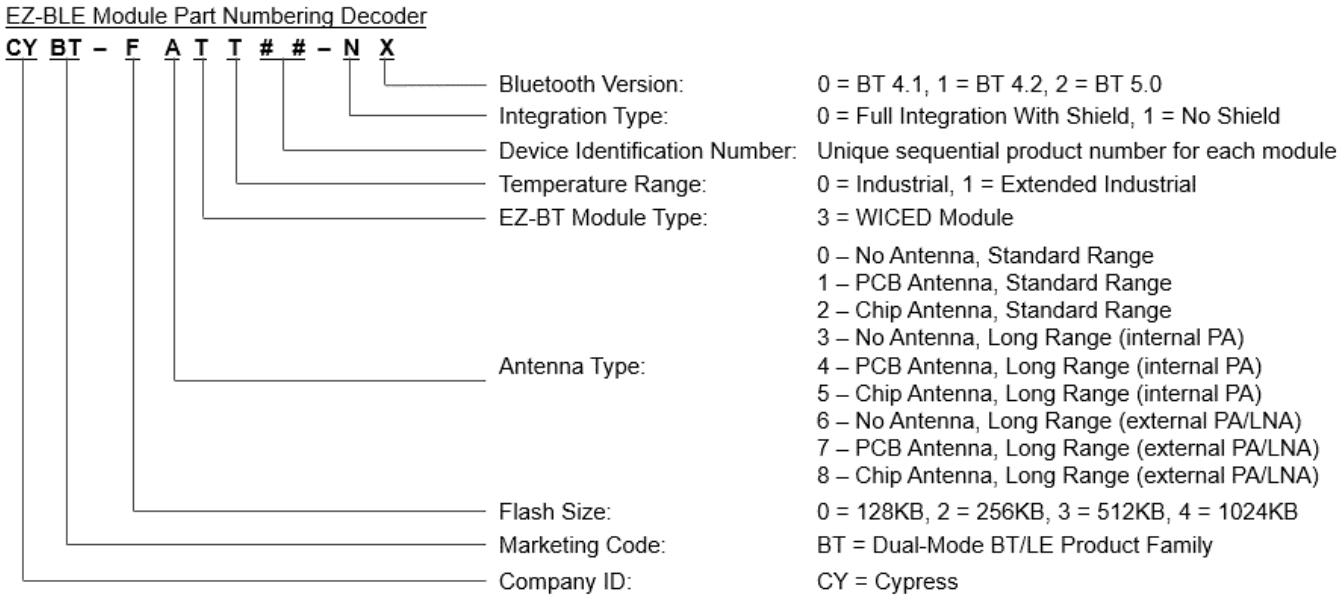


Table 3 summarizes the features and capabilities of each specific EZ-BT WICED Module MPN available from Cypress. Click on the specific part number for more detailed information on the device or refer to Appendix B: EZ-BT Module Product Details. Table 3 details all modules that are sampling or in production. For the latest list of modules available, see the Cypress Bluetooth Module website.

Table 3. EZ-BT Module MPN Features and Capabilities

Marketing Part Number	Silicon Device	Module Size (mm)	Range[1]	Regulatory Certification	BLE Standard	SIG Mesh Qualified	HomeKit Capable	Antenna Type	Package	GPIOs (Maximum)	Serial Flash (KB)	SRAM (KB)	I²S/PCM	PDM	PWMs	ADC	CapSense
CYBT-343026-01	CYW20706	14.52 x 19.2 x 1.95	225	Yes	5.0	Yes	Yes	Trace	24-SMT	14	512	352	Yes	No	4	Yes	No
CYBT-353027-02	CYW20707	9.00 x 9.00 x 1.75	150	Yes	5.0	Yes	Yes	Chip	19-SMT	8	512	352	Yes	No	0	Yes	No
CYBT-423028-02	CYW20719	11.00 x 11.00 x 1.70	75	Yes	5.0	Yes	Yes	Chip	28-SMT	17	1024	512	Yes	Yes	6	Yes	No
CYBT-413034-02	CYW20719	12.00 x 16.30 x 1.70	75	Yes	5.0	Yes	Yes	Trace	30-SMT	17	1024	512	Yes	Yes	6	Yes	No
CYBT-483039-02	CYW20719	12.75 x 18.59 x 1.80	1000[2]	Yes	5.0	Yes	Yes	Chip	34-SMT	15	1024	512	Yes	Yes	6	Yes	No

[1] Measured in meters and is specified as Full Line-of-Sight (LoS) in a Noise-Free environment.

[2] 1 km range is certified for US/FCC use only. European and Japan certifications are limited to 10 dBm (600 meters maximum).

4 Development Tools

Cypress supports EZ-BT WICED Modules with high-quality, integrated software tools. These include the following software:

1. WICED Studio SDK and IDE
2. CySmart™ PC application
3. CySmart Android app
4. CySmart iOS app

4.1 WICED Studio SDK and IDE

The WICED Studio SDK provides an Eclipse-based IDE and complete software library for developing on EZ-BT modules. This tool enables a simple build and download process as well as debugging capabilities on supported development kits. It also includes the graphical WICED Bluetooth Designer tool for quickly defining new BT/BLE designs and custom GATT database structures.

For Cypress module solutions, the WICED Studio SDK is **only** applicable to EZ-BT WICED Modules and not for EZ-BLE WICED modules or EZ-BLE Creator modules.

The WICED Studio SDK includes the following:

- Bluetooth 5.0-compliant software stack supporting both Bluetooth BR/EDR, Bluetooth Low Energy, and SIG Mesh
- Generic profile-level API
- Drivers to access onboard peripherals including UART, SPI, I^2C, ADC, PWM, etc.
- Reference applications for devices with profiles defined by the Bluetooth SIG
- WICED Studio API documentation and related documents
- Utilities to support development in Windows, Mac OS X, and Linux environments

The WICED Studio SDK runs on 32- and 64-bit versions of Microsoft Windows, Mac OS X, and Linux. The SDK is distributed as both a standalone 7-zip file suitable for all operating systems, and a bundle with the WICED Integrated Development Environment as an executable installer for Windows and Mac operating systems. The development computer requires a single USB port to connect to the Cypress EZ-BT evaluation boards.

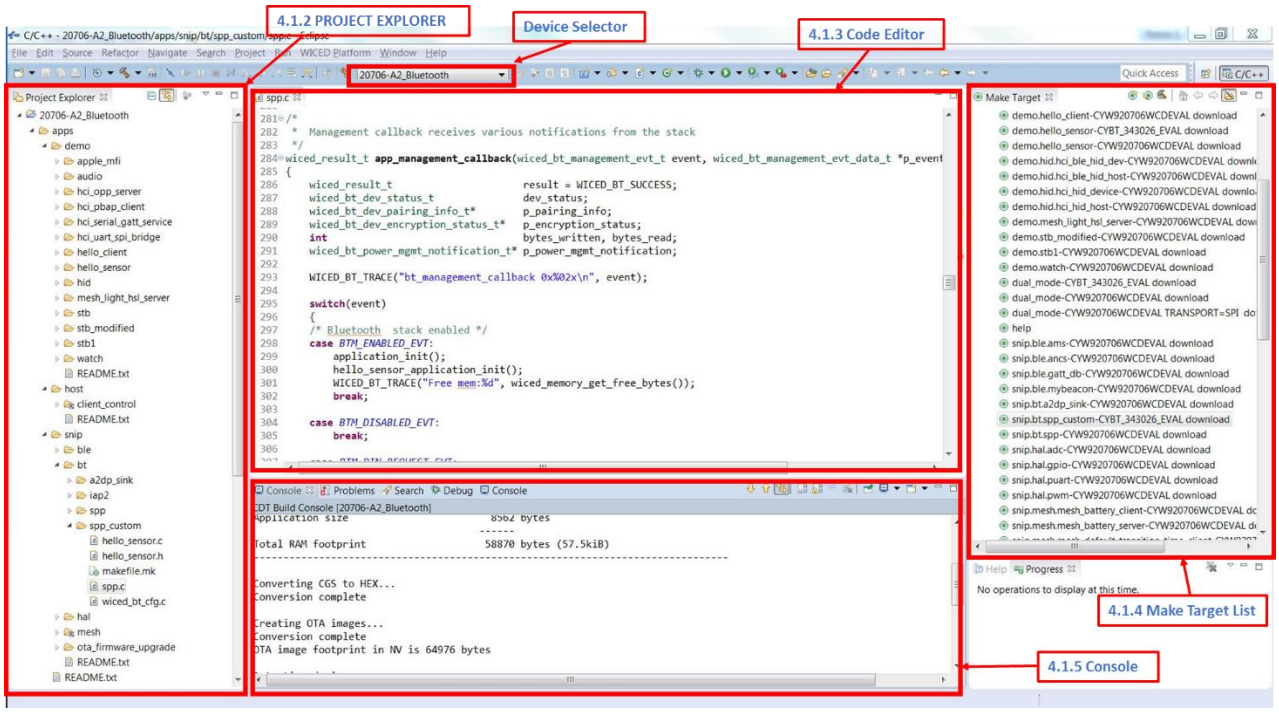
Note that a 32-bit version of Java is required to run the Eclipse-based IDE.

4.1.1 WICED Studio IDE Overview

The WICED Studio SDK comes with an Eclipse-based IDE that provides a comprehensive environment for creating, building, programming, and debugging WICED Bluetooth applications. Figure 2 below shows the default layout with various sections of the IDE.

Building and downloading WICED Studio projects requires a slightly different procedure from that some IDEs and toolchains use, so it is a good idea to familiarize yourself with the application early.

Figure 2. WICED Studio IDE Layout



4.1.2 Project Explorer

This pane in the IDE provides access to all the source files for the projects active in the current workspace. The WICED Studio SDK comes with a significant set of example projects inside the */apps* subfolder where the SDK is installed. These projects are visible in the Project Explorer view by default. New projects can be created under */apps* subfolder.

The standard workspace root folder is the SDK installation root folder, containing the following items:

- */apps* folder with all example projects and any created projects
- */build* folder with build output files (created when using “Make” targets as intended)
- */doc* folder with various SDK-related HTML and PDF reference materials
- */drivers* folder with FTDI USB-to-UART bridge device drivers for Microsoft Windows
- /libraries folder with source and header files for specific functionalities
- */include* folder with header files supporting various chipset hardware features
- */platforms* folder with board-specific toolchain and build definition files
- */wiced_tools with easy-to-use tools for various functionalities including collecting traces, build and download firmware etc.*
- */Wiced-BT* folder with header and source files for many application-visible APIs
- Top-level Makefile template and make scripts for building all WICED Studio projects
- Changelog, license, version, and general README text files
- Once you get started, the */apps* folder containing your projects will be the most relevant location. However, the documentation and header/source files provide a lot of helpful reference information during development.

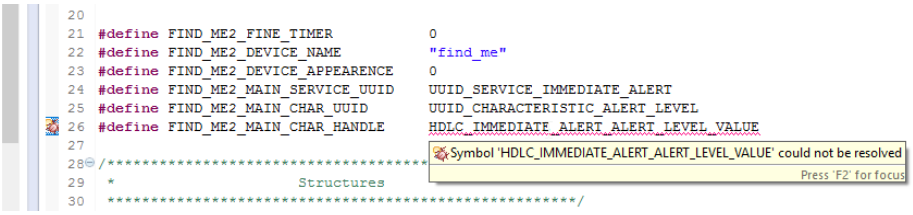
4.1.3 Code Editor

This pane allows editing all source code present in any project in the workspace. Open-source files are arranged by tabs for easy navigation. The Eclipse IDE foundation provides comprehensive syntax highlighting features, code completion, and other helpful functionality.

4.1.3.1 Eliminating False Code Analysis Errors

The Arm-GCC toolchain that WICED Studio uses to compile source files is not directly accessible to the Eclipse editor's built-in code analysis tools. Instead, the IDE uses a different compiler for live analysis, and this may result in identified errors that are not actually errors. For example, you may encounter this error while following the example project instructions contained in this guide:

Figure 3. Incorrect Code Analysis Error Identification

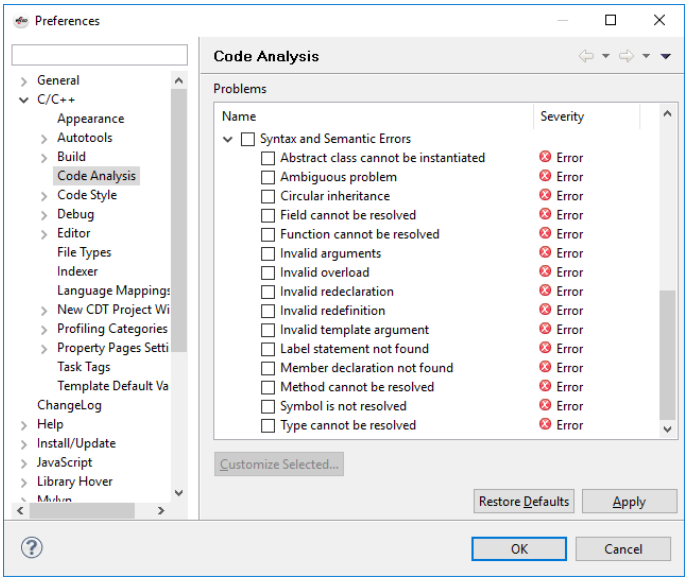


The best test for syntax errors in your code is the compile process, as any real warnings or errors will be included in the build output. However, you can selectively disable the code analysis features that trigger these errors by following these simple steps:

1. Click the **Window** menu, then "**Preferences**" item.
2. Expand **C/C++** and select the **Code Analysis** entry.
3. Uncheck the **Syntax and Semantic Errors** box (may need to scroll down in the "Problems" section to see this).
4. Click **OK** to save changes.

Figure 4 shows what the final Preferences subsection looks like after disabling Code Analysis syntax errors as described above.

Figure 4. Disabling Syntax and Semantic Analysis



4.1.3.2 Improving Search Results

Because the workspace includes multiple example projects and many SDK resources, global code searches often return more information than you need. To mitigate this, you can configure narrower search parameters to allow searching a single project at a time by defining working sets:

1. Click the **Search** menu, then "Search…" item (or press **Ctrl** + **H**).
2. Click the **Customize...** button and disable all items except "File Search", and then click **OK**.
3. Click the **Choose…** button next to the "Working set" selection field.

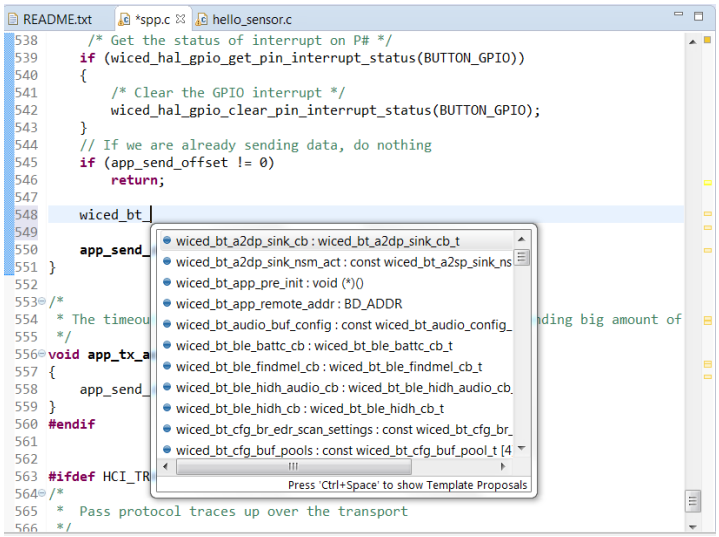
4. Click the **New…** button to define a new working set, and then choose “C/C++” and click **Next**.
5. Expand **20706-A2_Bluetooth** > **Apps** and select your project folder.
6. Enter a working set name (e.g., the same name as your project).
7. Click **Finish** to complete the working set definition.
8. Click the **Selected Working Sets** option and enable only the new set, then click **OK**.
9. Change the **Scope** setting to “Working set” if it does not change automatically.
10. Search as required with these settings to obtain results only within your project.

You can still perform global searches simply by changing the scope back to “**Workspace**” at any time, or by highlighting any text in the code editor and pressing **Ctrl** + **Alt** + **G**.

4.1.3.3 Taking Advantage of Code Completion

The WICED Studio SDK provides numerous APIs to use the features available on supported target chipsets; it can be challenging to keep the names and parameters straight. To help with this, use the **Ctrl** + **Space** shortcut key after typing the first few letters or prefix of a function name; Eclipse will pop up a quick list of potential completed names, as shown in the Figure 5.

Figure 5. Code Completion Example



Try code completion with any of the following API method prefixes (not a comprehensive list):

- `wiced_bt` – Core Bluetooth application functions, callbacks, and some connection management
- `wiced_bt_app` – For timer-related functions and others
- `wiced_hal_gpio–` General-purpose I/O (GPIO) features
- `wiced_hal` – ADC, GPIO, PUART, SFLASH, Watchdog, NVRAM, etc.

4.1.4 Make Target List

This pane contains individual build targets for example projects that come pre-installed with the WICED Studio SDK, as well as new make targets that you create for new projects. Each build target provides a unique combination of the following items:

- Path excluding root Apps folder (e.g., “snip.bt.”)
- Project name (e.g., “spp_custom”)
- Target platform (e.g., “CYBT_434026_EVAL”)
- Operational arguments (e.g., “download”, “UART=COM5”, and others)

Double-clicking on a make target will trigger the build process for that target. You can also use the **F9** keyboard shortcut in the Eclipse IDE to rebuild the last selected make target.

The path and project name are separated by a dot (“.”). The project name and target platform are separated by a dash (“–”), while the name/target and all subsequent operational arguments are separated by spaces. Possible arguments are described in the output from the “help” target, which you can build at any time to see details. The output from this target is reproduced here for quick reference:

```
Usage: make <target> [TRANSPORT=SPI] [FREQ=24Mhz|40Mhz] [download] [recover] [DEBUG=1|0] [VERBOSE=1] [UART=yyyy] [JOBS=x]
[PLATFORM_NV=EEPROM|SFLASH] [BT_DEVICE_ADDRESS=zzzzzzzzzzzz|random] [OTA_FW_UPGRADE=1] [B49=1] [LEGACY_BOARD=1]

  <target>
    One each of the following mandatory [and optional] components separated by '-'
      * Application (Apps in sub-directories are referenced by subdir.appname)
      * Hardware Platform (CYBT_343026_EVAL CYW920706WCDEVAL)
      * [BASE location] (BASErom BASEram BASEflash)
      * [SPAR location] (SPARrom SPARram SPARflash)
      * [Toolchain] (RealView Wiced CodeSourcery)

  [TRANSPORT]
    HCI transport interface (default is UART)

  [FREQ]
    Crystal frequency on the reference board (24Mhz/40Mhz)

  [download]
    Download firmware image to target platform

  [build]
    Builds the firmware and OTA images.

  [recover]
    Recover a corrupted target platform

  [library]
    Make a library of the app objects instead of fully linking it.

  [DEBUG=1|0]
    Enable or disable debug code in application. When DEBUG=1, watchdog will be disabled,
    sleep will be disabled and the app may optionally wait in a while(1) for the debugger
    to connect

  [VERBOSE=1]
    Shows the commands as they are being executed

  [JOBS=x]
    Sets the maximum number of parallel build threads (default=4)

  [UART=yyyy]
    Use the uart specified here instead of trying to detect the Wiced-BT device.
    This is useful when you are working on multiple BT devices simultaneously.

  [PLATFORM_NV=EEPROM|SFLASH]
    The default non-volatile storage. Default is EEPROM.

  [BT_DEVICE_ADDRESS=zzzzzzzzzzzz|random]
    Use the 48-bit Bluetooth address specified here instead of the default setting from
    platform/*/*.btp file. The special string 'random' (without the quotes) will generate
    a random Bluetooth device address on every download.

  [OTA_FW_UPGRADE]
    Applications which want to upgrade firmware via OTA, have to be downloaded through installer using this option.

  [B49=1]
    Set B49=1 if using the B49 platform board
```

Any single project may have one or more defined make targets. For instance, one target might perform only the compile step, while another performs both compile and download, and another may explicitly provide the programming COM port to avoid the serial port detection step if the port is known.

Here are some examples of make targets that come with the SDK:

- snip.bt.spp_custom-CYBT_343026_EVAL download
- snip.hal.pwm-CYBT_343026_EVAL download
- demo.hello_sensor-CYBT_343026_EVAL download

The platforms shipped with the SDK are used with the CYW920706WCDEVAL evaluation products that are built around the CYW20706 chipsets. However, for EZ-BT WICED Module evaluation, we will be working with the CYBT-343026-EVAL board instead, which requires a different platform definition. Section 7.3 (My First EZ-BT WICED Module Design) provides instructions on where to obtain and how to install this platform, or you can refer to the Cypress Knowledge Base Article (KBA) that is referenced in each EZ-BT module datasheet. The CYBT-343026-01 module platform file can be found in KBA221025 on the Cypress Community.

4.1.5 Console

This pane provides access to compiler output, which is helpful for status updates and critical for error analysis. The same area of the IDE window also allows a quick look at search results after performing a search, and enabled code analysis warnings or errors, and tasks identified by “TODO” comments in source files.

4.2 CySmart PC Application

CySmart Host Emulation Tool is a Windows application that emulates a BLE Central device using the CY5670 or CY5677 USB dongle. It provides a platform for you to test your EZ-BT WICED Module BLE Peripheral implementation over GATT or L2CAP connection-oriented channels by allowing you to discover and configure BLE services, characteristics, and attributes on your Peripheral.

The CySmart PC application provides only BLE functions; it cannot communicate over any non-low-energy Bluetooth protocols (RFCOMM, SCO, etc.) that are supported on EZ-BT WICED Modules.

Operations that you can perform with CySmart Host Emulation Tool include, but are not limited to:

- Scan BLE Peripherals to discover available devices to which you can connect
- Discover available BLE attributes including services and characteristics on the connected Peripheral device
- Perform read and write operations on characteristic values and descriptors
- Receive characteristic notifications and indications from the connected Peripheral device
- Establish a bond with the connected Peripheral device using BLE Security Manager procedures
- Establish a BLE L2CAP connection-oriented session with the Peripheral device and exchange data per the Bluetooth 4.2 specification

Figure 6 and Figure 7 show the user interface of CySmart Host Emulation Tool. For more information on how to set up and use this tool, see the CySmart user guide from the **Help** menu.

Figure 6. CySmart Host Emulation Tool Master Device Tab

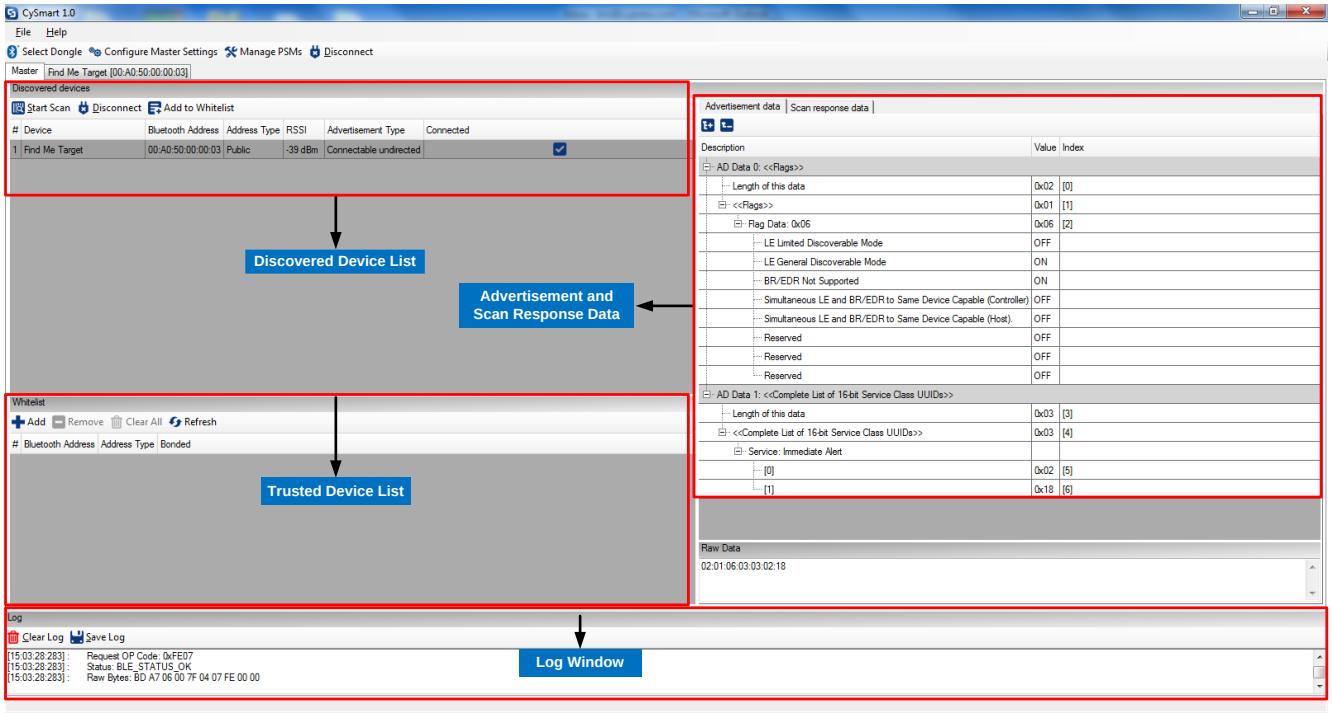
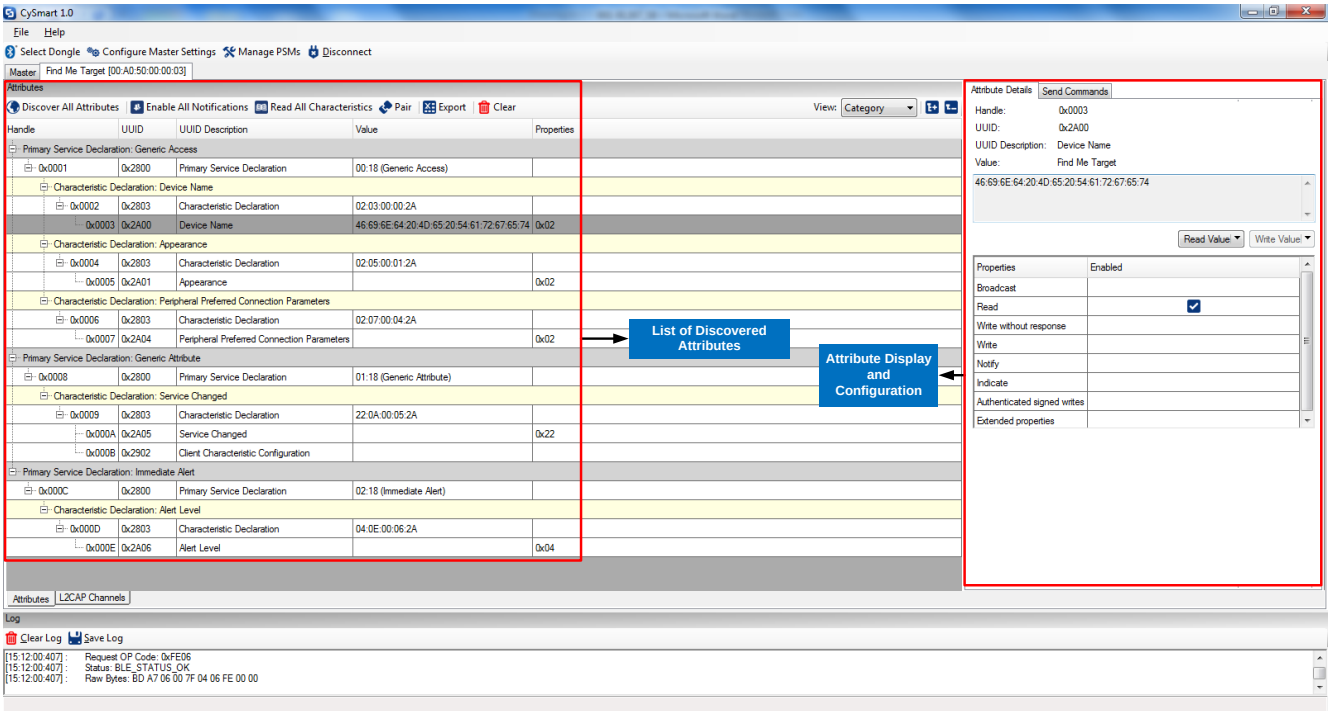


Figure 7. CySmart Host Emulation Tool Peripheral Device Attributes Tab



4.3 CySmart Mobile App

In addition to the PC tool, you can download the CySmart mobile app for iOS or Android from the respective app stores. This app uses the iOS Core Bluetooth framework and the Android built-in platform framework for BLE respectively to configure your BLE-enabled smartphone as a Central device that can scan and connect to Peripheral devices.

The CySmart mobile app provides only BLE functions; it cannot communicate over any non-low-energy Bluetooth protocols (RFCOMM, SCO, etc.) that are supported on EZ-BT WICED Modules.

The mobile app supports SIG-adopted BLE standard profiles through an intuitive GUI and abstracts the underlying BLE service and characteristic details. Figure 8 and Figure 9 show an example of CySmart app screenshots for a Heart Rate Profile user interface.

Figure 8. CySmart iOS App Heart Rate Profile Example

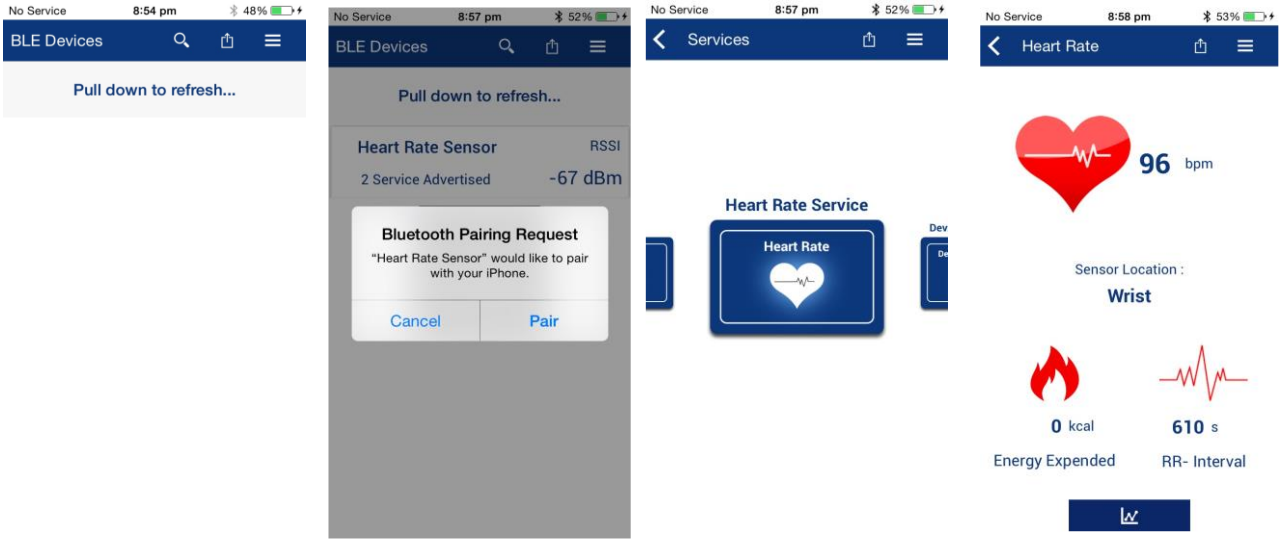
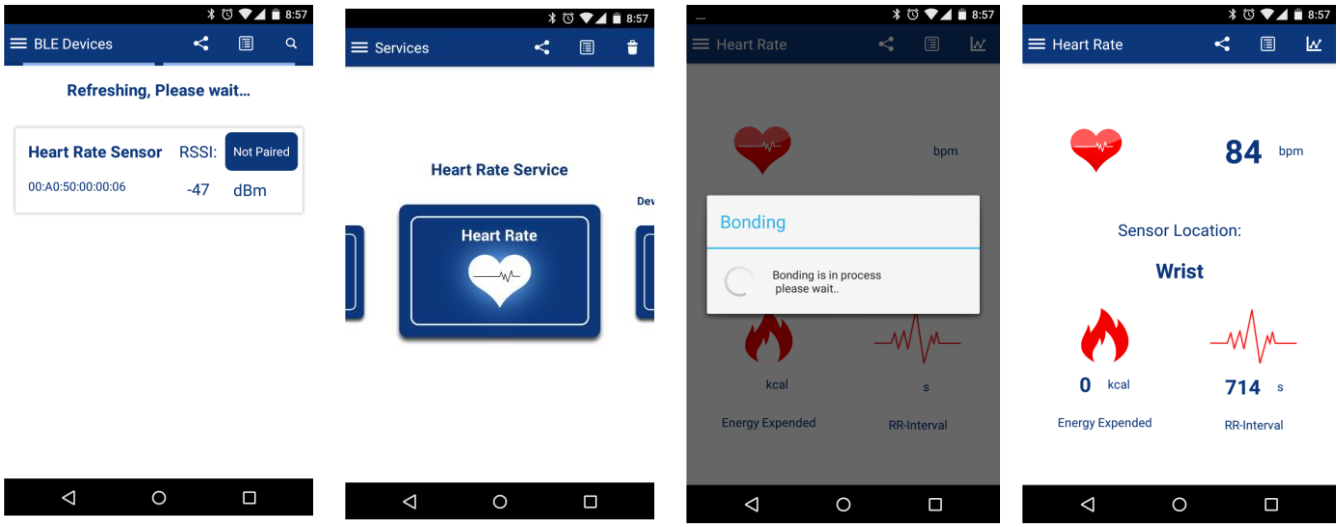


Figure 9. CySmart Android App Heart Rate Profile Example



5 Development Kits and Evaluation Boards

Cypress provides easy-to-use evaluation boards to help you develop your EZ-BT WICED Module design.

5.1 EZ-BT WICED Module Evaluation Boards

Each EZ-BT WICED Module provides an evaluation board that can be used to develop and test the performance of the Cypress EZ-BT WICED Module. EZ-BT WICED evaluation boards are Arduino-compatible baseboards, designed to work as stand-alone evaluation vehicles, or in conjunction with Arduino-compatible shields.

EZ-BT WICED evaluation boards allow you to evaluate Cypress EZ-BT Modules without having to design custom hardware to mount the Cypress EZ-BT Module.

Table 4 lists available EZ-BT WICED Modules and their corresponding evaluation board part numbers. Click on your evaluation board for additional information.

Table 4. EZ-BT Modules and Corresponding Evaluation Board Part Numbers

EZ-BT WICED Module Part Number	EZ-BT WICED Evaluation Board Part Number
CYBT-343026-01	CYBT-343026-EVAL
CYBT-353027-02	CYBT-353027-EVAL
CYBT-423028-02	CYBT-423028-EVAL
CYBT-413034-02	CYBT-413034-EVAL
CYBT-483039-02	CYBT-483039-EVAL

Each EZ-BT WICED evaluation board contains the following components:

- Cypress EZ-BT Module – soldered directly to the evaluation board
- PCB substrate
- Arduino-compatible baseboard headers
- USB-to-UART Bridge
- USB connection (for WICED Studio SDK PC interface and programming)
- Two-pin header connector for current consumption measurement
- Configuration headers for setting the required power supply level
- Power supply jumper for current consumption measurement
- Reset and switch
- User-defined switch element
- User-defined LED
- Inductors (for power supply noise reduction) – refer to your EZ-BT WICED Module datasheet for recommended external components)

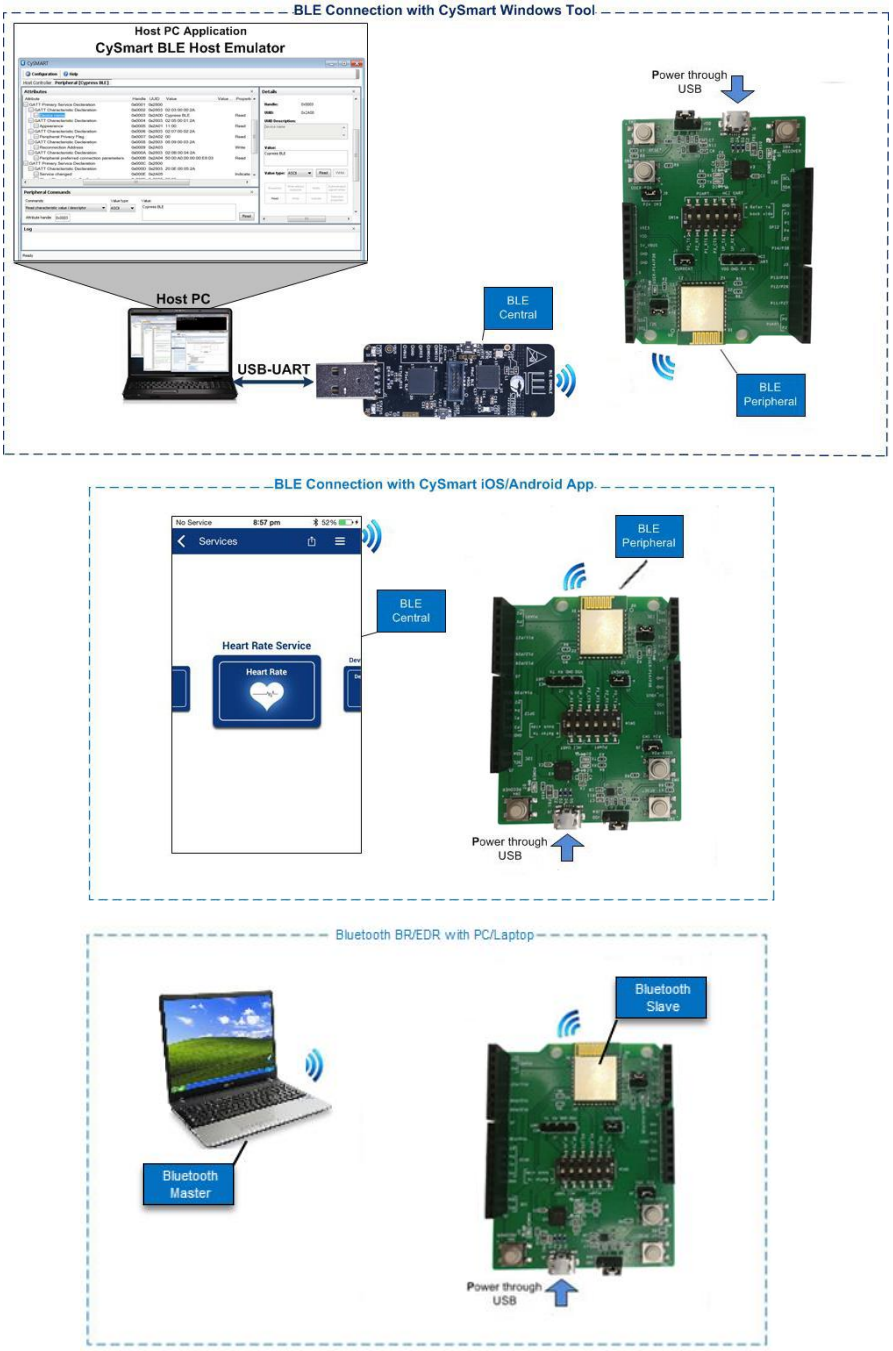
EZ-BT WICED evaluation boards are designed to simulate the placement and connection of the EZ-BT Modules in a final application. All host-side layout pattern recommendations (as shown in each specific module's datasheet) are followed for each evaluation board.

See Appendix C: EZ-BT WICED Evaluation Board Details for details on the connections available for each EZ-BT WICED evaluation board.

6 EZ-BT WICED Module Development Setup

Figure 10 shows the hardware and software required for evaluating Bluetooth accessory design. The EZ-BT WICED evaluation board is a self-contained Bluetooth device that can communicate with either a CySmart iOS/Android app or the CySmart Host Emulation Tool that acts as a Central device for BLE; and with a standard Bluetooth BR/EDR host as a master or slave. CySmart Host Emulation Tool requires a BLE dongle (black board in Figure 10) for operation. The dongle is included in the CY5677 kit.

Figure 10. BT/BLE Functional Setup with EZ-BT WICED Evaluation Board



The My First EZ-BT WICED Module Design section will walk you through a step-by-step configuration and programming of the EZ-BT WICED Module by creating a simple Bluetooth BR/EDR + BLE application.

7 My First EZ-BT WICED Module Design

This section gives you a step-by-step process for building a simple design with the CYBT-343026-EVAL kit using existing sample applications which are part of the WICED Studio installation. Although this example project focuses on CYBT-343026-EVAL, methods and process described here can be used for any CYBT WICED modules.

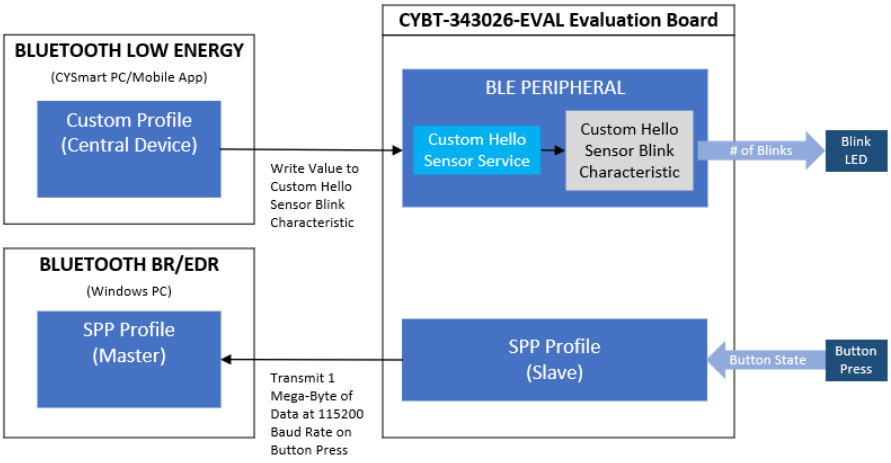
7.1 About the Design

This design implements the Serial Port Profile (SPP) profile for Bluetooth BR/EDR in slave mode, and a custom profile for BLE in Peripheral mode.

Bluetooth SPP data is transferred from the CYBT-343026-EVAL board to a host that supports the Bluetooth BR/EDR SPP profile. In this example project, we explain how to view the data that is transferred over SPP on a PC COM port using Tera Term (or equivalent). This implementation demonstrates bulk data transfer using the Bluetooth SPP profile.

The BLE Peripheral with custom profile can communicate with any standard BLE host. Write requests received on a custom characteristic (characteristic handle: 0x2D) from the BLE host will command the CYBT-343026-EVAL board to blink an onboard LED. The number of LED blinks depends on the byte value received for this custom characteristic.

Figure 11. My First EZ-BT WICED Module Design



7.2 Prerequisites

Before you get started with the implementation, ensure that you have the following software and hardware available:

- WICED Studio SDK v6.2.1 or later
- CySmart Host Emulation Tool or CySmart iOS/Android app
- PC which supports SPP profile for Bluetooth BR/EDR
- Tera Term application (or equivalent) running on the PC
- CYBT-343026-EVAL EZ-BT WICED Evaluation Board
- CYBT-343026-EVAL Platform files (KBA221025)

You can create your first EZ-BT WICED Module design in four steps:

1. Create a project which includes the SPP profile of Bluetooth BR/EDR and a custom profile of Bluetooth Low Energy. Merge SPP and Hello Sensor sample application projects. These example projects are contained in the WICED Studio SDK. Jump to Section 7.3.
2. Understand the flow of the WICED Studio application project and make necessary modifications based on the application requirements. Jump to Section 7.4.
3. Program the EZ-BT WICED Module on the Evaluation Kit. Jump to Section 7.5.
4. Test your design using the CySmart Host Emulation Tool or mobile application and PC, which supports Bluetooth BR/EDR and has Tera Term (or equivalent) application. Jump to Section 7.6.

7.3 Part 1: Merge SPP and Hello Sensor Application Samples

This section takes you step-by-step through the process of merging the Bluetooth BR/EDR and BLE sample application projects. In this example project, we will use the SPP (Bluetooth BR/EDR) and hello_sensor (BLE) sample application projects.

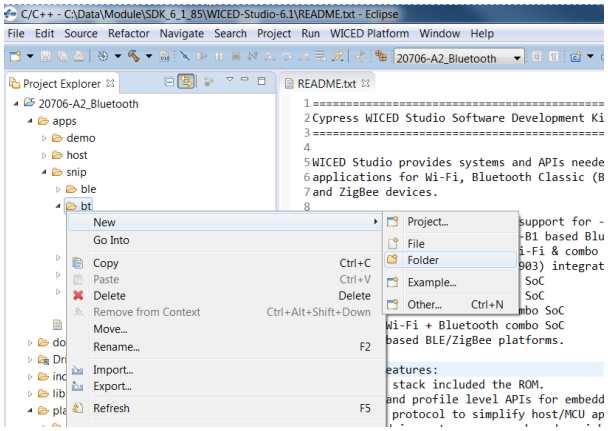
The merging process includes the following steps: 1) creating a new project, 2) copying appropriate files in to the new project folder, and 3) making necessary modifications. You can skip this section if you simply wish to try the example project provided with this application note without going through the code development process. Detailed steps to complete the merging process are shown below.

7.3.1 Creating a New Project

1. Create a new folder, and name it as "spp_custom" under the *apps/snip/bt* folder in Project Explorer (see Figure 12). In this example, the project folder is created at the following location:

   ...\WICED-Studio-6.1\20706-A2_Bluetooth\apps\snip\bt\spp_custom

Figure 12. Creating a New Project Folder



The next two steps involve copying files from the existing sample application projects, spp and hello_sensor, which are available as part of the WICED SDK installation.

2. Copy all files (*spp.c, makefile.mk, wiced_bt_cfg.c*) from the *spp* sample application project folder located at *…\WICED-Studio-6.1\20706-A2_Bluetooth\apps\snip\bt\spp* and paste them in the *spp_custom* folder created in Step 1.
3. Copy two files, *hello_sensor.c* and *hello_sensor.h* from the location *…\WICED-Studio-6.1\20706-A2_Bluetooth\apps\demo\hello_sensor* and paste them into the *spp_custom* folder created in Step 1.

7.3.2 Add 'hello_sensor' Project Files to the make file

1. Open *makefile.mk* and make the following changes:
   - Add *hello_sensor.c* application source file:

Code 1. *makefile.mk*: Add Application Source Files

```
APP_SRC += hello_sensor.c
```

2. Open *spp.c* and make the following changes:
   - Include the *hello_sensor.h* header file.

Code 2. *spp.c*: Add Header File

```
#include "hello_sensor.h"
```

7.3.3 Add BLE-related Initialization and Other Callback Functions in the spp.c File

3. As we are merging two projects, initialization of both SPP and Hello Sensor projects should occur in one call back function, `app_management_callback()`, of *spp.c*. Add the `hello_sensor_application_init()` function in the `app_management_callback()` function.

Code 3. *spp.c*: Add `hello_sensor_appplication_init` Funtion

```
    case BTM_ENABLED_EVT:
        application_init();
        hello_sensor_application_init();
        WICED_BT_TRACE("Free mem:%d", wiced_memory_get_free_bytes());
        break;
```

4. Add the following BLE stack events in the `app_management_callback()` function for BLE operation:
Because both BLE and Bluetooth BR/EDR events are consolidated in one callback function, it is required to define additional events for BLE that are missing in spp Bluetooth BR/EDR project file.

Note: Declare `wiced_bt_ble_advert_mode_t *p_mode` at the beginning of `app_management_callback()`.

- `BTM_SECURITY_REQUEST_EVT`: To grant access to BLE device pairing.
- `BTM_BLE_ADVERT_STATE_CHANGED_EVT`: To start BLE advertisement if advertisement is stopped after a timeout.
- `BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT`: To configure I/O capabilities of the BLE device which are exchanged during the pairing process with the remote Central device.

Code 4. *spp.c*: Add `wiced_bt_ble_security_grant` API

```
case BTM_SECURITY_REQUEST_EVT:
        wiced_bt_ble_security_grant( p_event_data->security_request.bd_addr, WICED_BT_SUCCESS );
    break;

case BTM_BLE_ADVERT_STATE_CHANGED_EVT:
        p_mode = &p_event_data->ble_advert_state_changed;
        WICED_BT_TRACE( "Advertisement State Change: %d\n", *p_mode);
        if ( *p_mode == BTM_BLE_ADVERT_OFF )
        {
            hello_sensor_advertisement_stopped();
        }
    break;

case BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT:
        p_event_data->pairing_io_capabilities_ble_request.local_io_cap   =
                                                                BTM_IO_CAPABILITIES_NONE;
        p_event_data->pairing_io_capabilities_ble_request.oob_data    = BTM_OOB_NONE;
        p_event_data->pairing_io_capabilities_ble_request.auth_req    = BTM_LE_AUTH_REQ_SC_BOND;
        p_event_data->pairing_io_capabilities_ble_request.max_key_size  = 0x10;
        p_event_data->pairing_io_capabilities_ble_request.init_keys     =
                                    BTM_LE_KEY_PENC|BTM_LE_KEY_PID|BTM_LE_KEY_PCSRK|BTM_LE_KEY_LENC;
        p_event_data->pairing_io_capabilities_ble_request.resp_keys     =
                                    BTM_LE_KEY_PENC|BTM_LE_KEY_PID|BTM_LE_KEY_PCSRK|BTM_LE_KEY_LENC;
    break;
```

5. Add the hello_sensor_encryption_changed function in BTM_ENCRYPTION_STATUS_EVT.

Code 5. *spp.c*: Add `hello_sensor_encryption_changed` API

```
case BTM_ENCRYPTION_STATUS_EVT:
        p_encryption_status = &p_event_data->encryption_status;
        WICED_BT_TRACE("Encryption Status Event: bd (%B) res %d\n", p_encryption_status->bd_addr,
                                                                p_encryption_status->result);
     hello_sensor_encryption_changed( p_encryption_status->result, p_encryption_status->bd_addr );
   break;
```

7.3.4 Modify 'hello_sensor' Files to Remove Duplication and Allow Access from spp.c

1. Open *hello_sensor.h* and add the function shown in Code 6:

Code 6. *hello_sensor.h*: Add hello_sensor Function Declarations

```
void hello_sensor_application_init(void);
void hello_sensor_encryption_changed( wiced_result_t result, uint8_t* bd_addr );
```

2. Open *hello_sensor.c* and make the following changes:

The merged project should have a single APPLICATION_START() function, and should register only one common Bluetooth event callback function for all BT BR/EDR and BLE events from the stack. To accomplish this requirement, make the following changes:

- Comment out the `hello_sensor_management_cback` function definition:

Code 7. *hello_sensor.c*: Comment `hello_sensor_management_cback` Function Definition

```
//static wiced_result_t  hello_sensor_management_cback( wiced_bt_management_evt_t event,
                                                wiced_bt_management_evt_data_t *p_event_data );
```

- Comment out the `hello_sensor_encryption_changed` function definition. This function was added into the *hello_sensor.h* file in Code 6. This function is commented out here because it is declared as a static function, which would not be accessible in *spp.c*.

Code 8. *hello_sensor.c*: Comment `hello_sensor_encryption_changed` Function Definition

```
//static void hello_sensor_encryption_changed( wiced_result_t result, uint8_t* bd_addr );
```

- Comment out the `hello_sensor_application_init` function definition. This function was added into the *hello_sensor.h* file in Code 6. This function is commented here because it is declared as a static function, which would not be accessible in *spp.c*.

Code 9. *hello_sensor.c*: Comment `hello_sensor_application_init` Function Definition

```
//static void hello_sensor_application_init( void );
```

- Comment out the complete `APPLICATION_START()` function.

Code 10: *hello_sensor.c*: Comment `application_start()` Function

```
//APPLICATION_START( )
//{
//    wiced_transport_init( &transport_cfg );
//
//#ifdef WICED_BT_TRACE_ENABLE
//    // Set the debug uart as WICED_ROUTE_DEBUG_NONE to get rid of prints
//    // wiced_set_debug_uart(WICED_ROUTE_DEBUG_NONE);
//
//    // Set to PUART to see traces on peripheral uart(puart)
//    wiced_set_debug_uart( WICED_ROUTE_DEBUG_TO_PUART );
//    wiced_hal_puart_select_uart_pads( WICED_PUART_RXD, WICED_PUART_TXD, 0, 0);
//
//    // Set to HCI to see traces on HCI uart - default if no call to wiced_set_debug_uart()
//    // wiced_set_debug_uart( WICED_ROUTE_DEBUG_TO_HCI_UART );
//
//    // Use WICED_ROUTE_DEBUG_TO_WICED_UART to send formatted debug strings over the WICED
//    // HCI debug interface to be parsed by ClientControl/BtSpy.
//    // wiced_set_debug_uart(WICED_ROUTE_DEBUG_TO_WICED_UART);
//#endif
//
//    WICED_BT_TRACE( "Hello Sensor Start\n" );
//
//    // Register call back and configuration with stack
//    wiced_bt_stack_init( hello_sensor_management_cback ,
//                    &wiced_bt_cfg_settings, wiced_bt_cfg_buf_pools );
//}
```

- Go to `hello_sensor_application_init()` and make the following changes:
  - Comment out `wiced_bt_app_init();`

Code 11. *hello_sensor.c*: Comment `wiced_bt_app_init` Function

```
//    wiced_bt_app_init();
```

- Comment the GPIO configuration shown in Code 12. CYBT-343026-EVAL has one user button, and it is configured in *spp.c* for a specific functionality. If there is a need for an additional button to execute other functionality, you should configure another available GPIO and verify the functionality by manually toggling the state of the GPIO using a blue wire.

Code 12. *hello_sensor.c*: Comment GPIO Functions Related to Button Activation

```
//    wiced_hal_gpio_configure_pin( WICED_GPIO_BUTTON, WICED_GPIO_BUTTON_SETTINGS(
                                    GPIO_EN_INT_RISING_EDGE ), WICED_GPIO_BUTTON_DEFAULT_STATE );

//    wiced_hal_gpio_register_pin_for_interrupt( WICED_GPIO_BUTTON, hello_sensor_interrupt_handler,
                                                                                    NULL );
```

- Comment out the complete `hello_sensor_management_cback()` function. *spp.c* has an `app_manangement_callback()` registered during the initialization for all BT BR/EDR and BLE events; therefore, there is no need for `hello_sensor_management_cback()`. Code 13 shows the commented out `hello_sensor_management_cback` function.

Code 13. *hello_sensor.c*: Comment `hello_sensor_management_cback` Function

```
//wiced_result_t hello_sensor_management_cback( wiced_bt_management_evt_t event, wiced_bt_management_evt_data_t *p_event_data
)
//{
//    wiced_result_t                    result = WICED_BT_SUCCESS;
//    wiced_bt_dev_encryption_status_t *p_status;
//    wiced_bt_dev_ble_pairing_info_t  *p_info;
//    wiced_bt_ble_advert_mode_t       *p_mode;
//    uint8_t                          *p_keys;
//
//    WICED_BT_TRACE("hello_sensor_management_cback: %x\n", event );
//
//    switch( event )
//    {
//    /* Bluetooth  stack enabled */
//    case BTM_ENABLED_EVT:
//        hello_sensor_application_init();
//        break;
//
//    case BTM_DISABLED_EVT:
//        break;
//
//    case BTM_USER_CONFIRMATION_REQUEST_EVT:
//        WICED_BT_TRACE("numeric_value: %d \n", p_event_data->user_confirmation_request.numeric_value);
//        wiced_bt_dev_confirm_req_reply( WICED_BT_SUCCESS , p_event_data->user_confirmation_request.bd_addr);
//        break;
//
//    case BTM_PASSKEY_NOTIFICATION_EVT:
//        WICED_BT_TRACE("PassKey Notification. BDA %B, Key %d \n", p_event_data->user_passkey_notification.bd_addr,
//                                          p_event_data->user_passkey_notification.passkey );
//        wiced_bt_dev_confirm_req_reply(WICED_BT_SUCCESS, p_event_data->user_passkey_notification.bd_addr );
//        break;
//
//    case BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT:
//        p_event_data->pairing_io_capabilities_ble_request.local_io_cap   = BTM_IO_CAPABILITIES_NONE;
//        p_event_data->pairing_io_capabilities_ble_request.oob_data   = BTM_OOB_NONE;
//        p_event_data->pairing_io_capabilities_ble_request.auth_req   = BTM_LE_AUTH_REQ_SC_BOND;
//        p_event_data->pairing_io_capabilities_ble_request.max_key_size  = 0x10;
//        p_event_data->pairing_io_capabilities_ble_request.init_keys     =
//                                    BTM_LE_KEY_PENC|BTM_LE_KEY_PID|BTM_LE_KEY_PCSRK|BTM_LE_KEY_LENC;
//        p_event_data->pairing_io_capabilities_ble_request.resp_keys     =
//                                    BTM_LE_KEY_PENC|BTM_LE_KEY_PID|BTM_LE_KEY_PCSRK|BTM_LE_KEY_LENC;
//        break;
//
//    case BTM_PAIRING_COMPLETE_EVT:
//        p_info =  &p_event_data->pairing_complete.pairing_complete_info.ble;
//        WICED_BT_TRACE( "Pairing Complete: %d",p_info->reason);
//        hello_sensor_smp_bond_result( p_info->reason );
//        break;
//
//    case BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT:
//            /* save keys to NVRAM */
//            p_keys = (uint8_t*)&p_event_data->paired_device_link_keys_update;
//            wiced_hal_write_nvram ( HELLO_SENSOR_PAIRED_KEYS_VS_ID, sizeof( wiced_bt_device_link_keys_t ), p_keys ,&result
);
//            WICED_BT_TRACE("keys save to NVRAM %B result: %d \n", p_keys, result);
//        break;
//
```

```
//     case  BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT:
//            /* read keys from NVRAM */
//            p_keys = (uint8_t *)&p_event_data->paired_device_link_keys_request;
//            wiced_hal_read_nvram( HELLO_SENSOR_PAIRED_KEYS_VS_ID, sizeof(wiced_bt_device_link_keys_t), p_keys, &result );
//            WICED_BT_TRACE("keys read from NVRAM %B result: %d \n", p_keys, result);
//        break;
//
//    case BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT:
//            /* save keys to NVRAM */
//            p_keys = (uint8_t*)&p_event_data->local_identity_keys_update;
//          wiced_hal_write_nvram ( HELLO_SENSOR_LOCAL_KEYS_VS_ID, sizeof( wiced_bt_local_identity_keys_t ), p_keys ,&result);
//            WICED_BT_TRACE("local keys save to NVRAM result: %d \n", result);
//        break;
//
//
//    case BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT:
//            /* read keys from NVRAM */
//            p_keys = (uint8_t *)&p_event_data->local_identity_keys_request;
//            wiced_hal_read_nvram( HELLO_SENSOR_LOCAL_KEYS_VS_ID, sizeof(wiced_bt_local_identity_keys_t), p_keys, &result );
//            WICED_BT_TRACE("local keys read from NVRAM result: %d \n",  result);
//        break;
//
//        case BTM_ENCRYPTION_STATUS_EVT:
//            p_status = &p_event_data->encryption_status;
//            WICED_BT_TRACE( "Encryption Status Event: bd ( %B ) res %d", p_status->bd_addr, p_status->result);
//            hello_sensor_encryption_changed( p_status->result, p_status->bd_addr );
//        break;
//
//        case BTM_SECURITY_REQUEST_EVT:
//            wiced_bt_ble_security_grant( p_event_data->security_request.bd_addr, WICED_BT_SUCCESS );
//        break;
//
//    case BTM_BLE_ADVERT_STATE_CHANGED_EVT:
//        p_mode = &p_event_data->ble_advert_state_changed;
//        WICED_BT_TRACE( "Advertisement State Change: %d\n", *p_mode);
//        if ( *p_mode == BTM_BLE_ADVERT_OFF )
//        {
//            hello_sensor_advertisement_stopped();
//        }
//        break;
//
//    default:
//        break;
//    }
//
//    return result;
//}
```

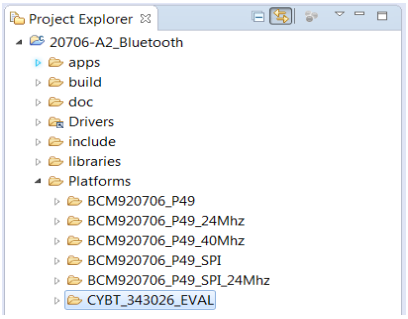
7.3.5 Create Platform Files, Build, and Download

1. If you have not installed the WICED SDK platform files for the CYBT-343026-EVAL board at this point, you should do so now. The platform files define the functional GPIO assignments, NVRAM type, and other low-level characteristics that control the firmware image download process for a specific target device. This custom platform file definition is required to properly use the CYBT-343026-EVAL board. You can find the reference and link to the EZ-BT platform file in the respective EZ-BT module datasheet. After you download the platform file, install it and do the following:

   CYBT-343026-01 Platform File Location: Visit community.cypress.com/docs/DOC-13750 and download the *CYBLE_343026_EVAL Platform files.zip* file. Follow the instructions described in the KBA; they are presented below for quick reference:

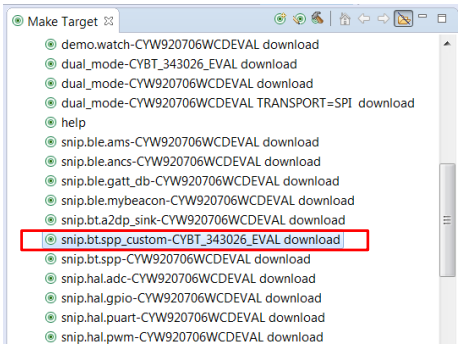
   - Download *CYBT_343026_EVAL.zip* attached with this KBA.
   - Save and extract the downloaded file.
   - Navigate to the folder *CYBT-343026-EVAL > 002-21025* and extract *CYBT_343026_EVAL_PLATFORM_FILES.zip*.
   - Navigate to the folder *CYBT-343026-EVAL > 002- 21025 > CYBT_343026_EVAL_PLATFORM_FILES > CYBT_343026_EVAL_PLATFORM_FILES*.
   - Select and copy the folder *CYBT_343026_EVAL* under *CYBT_343026_EVAL_PLATFORM_FILES*. This folder contains the necessary platform files.
   - Paste the folder in WICED Studio 6.2.1 or later at the following path: …\ *WICED-Studio-6.2.1\20706-A2_Bluetooth\Platforms*. Figure 13 shows the location of the folder in WICED Studio 6.2.1.

Figure 13. Location of CYBT-343026-EVAL Platform Folder



2. The Make Target list is located at the top-right location of the WICED Studio IDE. Right-click on any of the existing make targets and click “new” to create a new make target for your project. If you used the “spp_custom” name for the project as described in this guide, the make target should be named as `snip.bt.spp_custom-CYBT_343026_EVAL download`, as shown in Figure 14. The Make Target list provides the mechanism for compiling and downloading firmware to the target device. The build system supports some custom arguments to assist with troubleshooting or special cases. These arguments are discussed in Section 4.1.4 (Make Target List).

Figure 14. *Download* Make Target for spp_custom Project



3. To confirm that the modified code compiles successfully, double-click the `snip.bt.spp_custom-CYBT_343026_EVAL download` make target and observe the output from the compile process. If everything is working normally, the project will build successfully and show a memory usage summary, as shown in Figure 15. However, it will not download because no suitable target device is connected and ready to flash, as shown in Figure 16.

Figure 15. Successful Build and Memory Footprint

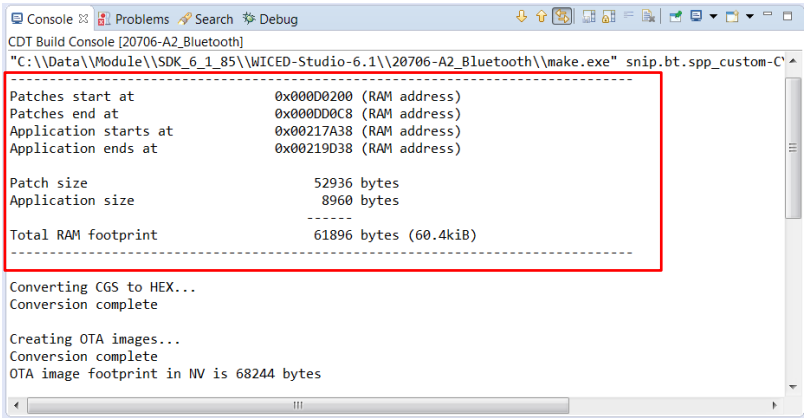
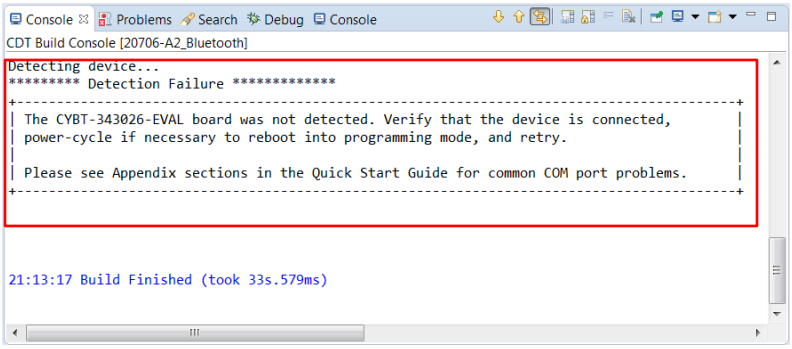


Figure 16. Expected Download Failure After Compiling



The WICED Studio IDE may show some symbol resolution errors in the main code window. Often, these symbols would have been resolved correctly, but the IDE is misidentifying issues in the code due to differences between the code analysis toolchain and the one used for the final build process. For instructions on how to disable these errors, see Section 4.1.3.1 (Eliminating False Code Analysis Errors).

7.4 Part 2: Understanding the Flow of the WICED Project

EZ-BT WICED Module contains ROM (Read Only Memory), RAM (Random Access Memory), and Flash memory. The ROM area of the module contains the full Bluetooth stack. The RAM area is used for applying patches to the ROM-stored stack as well as for running applications. Flash memory in the device is a nonvolatile memory, which stores the application. For devices without internal flash, a host device is required to load the application program into RAM.

The application focuses on application-specific functionality while the stack deals with the low-level details. For example, the stack transparently handles a GATT Client read request or discovery. Also, the application provides various configuration details for the stack such as advertisement content and interval or output power during transmission.

The following main steps are required to develop an application for an EZ-BT WICED Module:

- Define the data to be exchanged between the Client/Master and Server/Slave and prepare a database for BLE and Bluetooth BR/EDR. (In this example, this is already accomplished during the merging process described in Section 7.2 Part 1)
- Determine whether additional devices such as a UART-connected MCU or an SPI-connected peripheral sensor will be included in the solution. The UART, SPI or GPIO configurations of the application depend on the connected Peripheral devices. In this example project, a button and a LED are configured as part of the application.
- Adjust the application configuration to provide the parameters required by the application. Common changes include advertisement parameters, and device name. In this example project, BLE peripheral advertises with the name “Hello” and Bluetooth BR/EDR slave shows up as “spp_test” during inquiry process.
- Define and code the functions for the Bluetooth Stack callbacks required by the application. The application typically requires notifications from the stack when certain Bluetooth events occur such as connection establishment, disconnection, GATT write operations, or bonding.

Three main firmware blocks are required for designing Bluetooth applications using the WICED Studio SDK:

1. System initialization
2. Bluetooth stack event handlers
3. Bluetooth service-specific event handlers

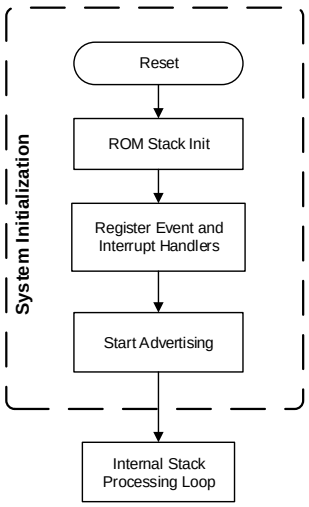
The following sections discuss these blocks with respect to the design that was configured as part of the merging process in Section 7.2 Part 1. Most of the functions involved achieve the required functionality. However, some of the functions need to have additional code added to achieve the specific behavior that your application requires.

Unlike other platforms, the WICED Studio SDK Bluetooth stack does not require or provide an application-level “main loop” function that repeats forever. Instead, the main loop is handled internally along with low-power transitions, and all application behavior must be fully event-driven based on interrupts triggered by timers, wireless (Bluetooth) activity, or wired (GPIO, UART, etc.) activity.

7.4.1 System Initialization

When the EZ-BT WICED Module starts up, it enters the `APPLICATION_START()` function. The `BTM_ENABLED_EVT` is the first stack-generated event after power ON or reset; it indicates that Bluetooth stack initialization is complete. The initialization functions `application_init` and `hello_sensor_applicaiton_init` functions are executed in the `BTM_ENABLED_EVT` event. These initialization functions call appropriate APIs to initialize the Service Discovery Protocol (SDP) database for Bluetooth BR/EDR, initialize the GATT database for BLE, set the advertisement data, set the inquiry response data and register appropriate callbacks, initialize the timers, and configure the button and LED based on the application requirements. Figure 17 shows the flowchart for system initialization.

Figure 17. System Initialization Flowchart



Before looking at the application initialization routines, look at *wiced_bt_cfg.c*. The configuration structure in this file allows simple control over advertisement and scan parameters for BLE and inquiry/page scan interval for Bluetooth BR/EDR, and some of the profile-specific behavior. In many cases, the SDK provides APIs that allow you to trigger behavior with custom settings rather than the values that are set here. However, you can often use these configuration values as-is and avoid further complexity in your code.

The following Bluetooth configuration parameters can be modified based on application requirements by modifying values in *wiced_bt_cfg.c* shown in Code 14.

- Advertisement and scan settings for BLE
- Choice of advertisement channels to be used for BLE
- Scan and page interval settings for Bluetooth BR/EDR
- Bluetooth Device name
- MTU size, etc.

The “spp_custom” example described in this guide does not require any modifications to this structure except BT_LOCAL_NAME. You can choose to retain the default name as well; for the spp sample application, it is “spp”.

Code 14. wiced_bt_cfg: Project Configuration Structure

```
/* Null-Terminated Local Device Name */
uint8_t BT_LOCAL_NAME[] = { 's','p','p',' ','t','e','s','t','\0' };
const uint16_t BT_LOCAL_NAME_CAPACITY = sizeof(BT_LOCAL_NAME);

const wiced_bt_cfg_settings_t wiced_bt_cfg_settings =
{
    (uint8_t*)BT_LOCAL_NAME,        /**< Local device name (NULL terminated) */
    {0x00, 0x00, 0x00},             /**< Local device class */
    BTM_SEC_IN_AUTHENTICATE | BTM_SEC_OUT_AUTHENTICATE | BTM_SEC_ENCRYPT,   /**< Security requirements mask
(BTM_SEC_NONE, or combination of BTM_SEC_IN_AUTHENTICATE, BTM_SEC_OUT_AUTHENTICATE, BTM_SEC_ENCRYPT */
    3,
    /* BR/EDR Scan Configuration */
    {
```

```
    BTM_SCAN_TYPE_STANDARD,    /**< Inquiry Scan Type (BTM_SCAN_TYPE_STANDARD or
                    BTM_SCAN_TYPE_INTERLACED) */
    WICED_BT_CFG_DEFAULT_INQUIRY_SCAN_INTERVAL,                    /**< Inquiry Scan Interval (0 to
                     use default) */
    WICED_BT_CFG_DEFAULT_INQUIRY_SCAN_WINDOW,                      /**< Inquiry Scan Window (0 to
                                                                        use default) */

    BTM_SCAN_TYPE_STANDARD,  /**< Page Scan Type (BTM_SCAN_TYPE_STANDARD or BTM_SCAN_TYPE_INTERLACED) */
    WICED_BT_CFG_DEFAULT_PAGE_SCAN_INTERVAL,              /**< Page Scan Interval (0 to use default) */
    WICED_BT_CFG_DEFAULT_PAGE_SCAN_WINDOW,                 /**< Page Scan Window (0 to use default) */
},

/* BLE scan settings  */
{
    BTM_BLE_SCAN_MODE_ACTIVE,                                  /**< BLE scan mode (
                    BTM_BLE_SCAN_MODE_PASSIVE, BTM_BLE_SCAN_MODE_ACTIVE, or BTM_BLE_SCAN_MODE_NONE ) */

    /* Advertisement scan configuration */
    96,                 /**< High duty scan interval */
    48,                 /**< High duty scan window */
    30,                 /**< High duty scan duration in seconds ( 0 for infinite ) */

    2048,                /**< Low duty scan interval  */
    48,                 /**< Low duty scan window */
    30,                 /**< Low duty scan duration in seconds ( 0 for infinite ) */

    /* Connection scan configuration */
    96,            /**< High duty cycle connection scan interval */
    48,            /**< High duty cycle connection scan window */
    30,            /**< High duty cycle connection duration in seconds ( 0 for infinite ) */

    2048,           /**< Low duty cycle connection scan interval */
    48,             /**< Low duty cycle connection scan window */
    30,             /**< Low duty cycle connection duration in seconds ( 0 for infinite ) */

    /* Connection configuration */
    WICED_BT_CFG_DEFAULT_CONN_MIN_INTERVAL,                    /**< Minimum connection interval */
    WICED_BT_CFG_DEFAULT_CONN_MAX_INTERVAL,                    /**< Maximum connection interval */
    WICED_BT_CFG_DEFAULT_CONN_LATENCY,                         /**< Connection latency */
    WICED_BT_CFG_DEFAULT_CONN_SUPERVISION_TIMEOUT,          /**< Connection link supervision timeout */
},

/* BLE advertisement settings */
{
    BTM_BLE_ADVERT_CHNL_37 |                                   /**< Advertising channel map ( mask of
                            BTM_BLE_ADVERT_CHNL_37, BTM_BLE_ADVERT_CHNL_38, BTM_BLE_ADVERT_CHNL_39 ) */
    BTM_BLE_ADVERT_CHNL_38 |
    BTM_BLE_ADVERT_CHNL_39,

    WICED_BT_CFG_DEFAULT_HIGH_DUTY_ADV_MIN_INTERVAL,           /**< High duty undirected connectable
                                                                    minimum advertising interval */
    WICED_BT_CFG_DEFAULT_HIGH_DUTY_ADV_MAX_INTERVAL,           /**< High duty undirected connectable
                                                                  maximum advertising interval */
    30,                 /**< High duty undirected connectable advertising duration in seconds ( 0 for
                                                                      infinite ) */

    WICED_BT_CFG_DEFAULT_LOW_DUTY_ADV_MIN_INTERVAL,            /**< Low duty undirected connectable
                                                                     minimum advertising interval */
    WICED_BT_CFG_DEFAULT_LOW_DUTY_ADV_MAX_INTERVAL,            /**< Low duty undirected connectable
                                                                   maximum advertising interval */
    60,                 /**< Low duty undirected connectable advertising duration in seconds ( 0 for
                                                                        infinite ) */

    WICED_BT_CFG_DEFAULT_HIGH_DUTY_DIRECTED_ADV_MIN_INTERVAL,  /**< High duty directed connectable
                                                                  minimum advertising interval */
    WICED_BT_CFG_DEFAULT_HIGH_DUTY_DIRECTED_ADV_MAX_INTERVAL,  /**< High duty directed connectable
                                                                  maximum advertising interval */

    WICED_BT_CFG_DEFAULT_LOW_DUTY_DIRECTED_ADV_MIN_INTERVAL,   /**< Low duty directed connectable
                                                                  minimum advertising interval */
    WICED_BT_CFG_DEFAULT_LOW_DUTY_DIRECTED_ADV_MAX_INTERVAL,   /**< Low duty directed connectable
                                                                  maximum advertising interval */
    30,                 /**< Low duty directed connectable advertising duration in seconds ( 0 for
                                                                    infinite ) */

    WICED_BT_CFG_DEFAULT_HIGH_DUTY_NONCONN_ADV_MIN_INTERVAL,   /**< High duty non-connectable minimum
                                                                  advertising interval */
```

```
        WICED_BT_CFG_DEFAULT_HIGH_DUTY_NONCONN_ADV_MAX_INTERVAL,     /**< High duty non-connectable maximum
                                                                       advertising interval */
        30,                 /**< High duty non-connectable advertising duration in seconds ( 0 for infinite
                                                                                            ) */

        WICED_BT_CFG_DEFAULT_LOW_DUTY_NONCONN_ADV_MIN_INTERVAL,      /**< Low duty non-connectable minimum
                                                                      advertising interval */
        WICED_BT_CFG_DEFAULT_LOW_DUTY_NONCONN_ADV_MAX_INTERVAL,      /**< Low duty non-connectable maximum
                                                                      advertising interval */
        0               /**< Low duty non-connectable advertising duration in seconds ( 0 for infinite ) */
    },

    /* GATT Configuration */
    {
        APPEARANCE_GENERIC_COMPUTER,   /**< GATT appearance ( see gatt_appearance_e ) */
        3,          /**< Client config: maximum number of servers that local client can connect to  */
        1,          /**< Server config: maximum number of remote clients connections allowed by the local */
        512         /**< Client config: maximum number of bytes that local client can receive over LE link  */
    },

    /* RFCOMM Configuration */
    {
        2,                  /**< Maximum Number of simultaneous RFCOMM ports */
        2,                  /**< Maximum Number of simultaneous RFCOMM connections */
    },

    /* Application managed l2cap protocol configuration */
    {
        0,                  /**< Maximum number of application-managed l2cap links (BR/EDR and LE) */
                            /* BR EDR l2cap configuration */
        0,                  /**< Maximum number of application-managed BR/EDR PSMs */
        0,                  /**< Maximum number of application-managed BR/EDR channels  */
                            /* LE L2cap connection-oriented channels configuration */
        0,                  /**< Maximum number of application-managed LE PSMs */
        0,                  /**< Maximum number of application-managed LE channels */
    },

    /* Audio/Video Distribution configuration */
    {
        0,                  /**< Maximum simultaneous audio/video links */
    },

    /* Audio/Video Remote Control configuration */
    {
        0,                  /**< Mask of local roles supported (AVRC_CONN_INITIATOR|AVRC_CONN_ACCEPTOR) */
        0,                  /**< Maximum simultaneous remote control links */
    },
    5,          /**< LE Address Resolution DB settings - effective only for pre 4.2 controller*/
    517,        /**< Maximum MTU size for GATT connections, should be between 23 and (max_attr_len + 5 )*/
    12
};
```

The first visible entry point in the “spp_custom” example application you have created here is the `APPLICATION_START()` function, as shown in Code 15:

Code 15. `APPLICATION_START`: Stack initialization is ROM-Driven Initialization

```
APPLICATION_START()
{
    wiced_result_t result;

#if defined WICED_BT_TRACE_ENABLE || defined HCI_TRACE_OVER_TRANSPORT
    wiced_transport_init(&transport_cfg);

    // create special pool for sending data to the MCU
    host_trans_pool = wiced_transport_create_buffer_pool(TRANS_UART_BUFFER_SIZE, TRANS_MAX_BUFFERS);

    // Set the debug uart as WICED_ROUTE_DEBUG_NONE to get rid of prints
    // wiced_set_debug_uart(WICED_ROUTE_DEBUG_NONE);

    // Set to PUART to see traces on peripheral uart(puart)
    wiced_set_debug_uart( WICED_ROUTE_DEBUG_TO_PUART );
    wiced_hal_puart_select_uart_pads( WICED_PUART_RXD, WICED_PUART_TXD, 0, 0);

    // Set to HCI to see traces on HCI uart - default if no call to wiced_set_debug_uart()
    // wiced_set_debug_uart( WICED_ROUTE_DEBUG_TO_HCI_UART );
```

```
    // Use WICED_ROUTE_DEBUG_TO_WICED_UART to send formatted debug strings over the WICED
    // HCI debug interface to be parsed by ClientControl/BtSpy.
    // Note: WICED HCI must be configured to use this - see wiced_trasnport_init(), must
    // be called with wiced_transport_cfg_t.wiced_tranport_data_handler_t callback present
    // wiced_set_debug_uart(WICED_ROUTE_DEBUG_TO_WICED_UART);
#endif

    WICED_BT_TRACE("APP Start\n");

    /* Initialize Stack and Register Management Callback */
    // Register call back and configuration with stack
    wiced_bt_stack_init(app_management_callback, &wiced_bt_cfg_settings, wiced_bt_cfg_buf_pools);
}
```

The `APPLICATION_START()` function does the following

- Transport method is initialized by calling `wiced_transport_init()`; in this case UART
- Bluetooth stack is initialized by calling `wiced_bt_stack_init`.
- As part of the `wiced_bt_stack_init` function, Bluetooth configuration settings wiced_bt_cfg_settings are passed to the stack and `app_management_callback` function is registered to receive both BLE and Bluetooth BR/EDR stack events.

Note that this function already exists and does not need to be added to your project source file. The `wiced_bt_stack_init` function is declared in the *wiced_bt_stack.h* standard SDK library include file, actual implementation of this function is in ROM.

Either PUART (WICED_ROUTE_DEBUG_TO_PUART) or HCI UART (WICED_ROUTE_DEBUG_TO_HCI_UART) can be configured to receive debug traces based on the application requirement. PUART is enabled by default in this example application.

The *spp.c, hello_sensor.c/h*, and *wiced_bt_cfg.c* source files contain the following key functions for BR/EDR and BLE:

Table 5. Key BR/EDR and BLE Functions Used in My First EZ-BT WICED Module Design

Common functions for Bluetooth BR/EDR and BLE (*spp_custom.c*)	
application_init	
app_management_callback	
Bluetooth BR/EDR functions (*spp_custom.c*)	**BLE functions**
app_write_eir	hello_sensor_application_init
app_timeout	hello_sensor_set_advertisement_data
spp_connection_up_callback	hello_sensor_advertisement_stopped
spp_connection_down_callback	hello_sensor_timeout
spp_rx_data_callback	hello_sensor_fine_timeout
bt_write_nvram	hello_sensor_smp_bond_result
bt_read_nvram	hello_sensor_encryption_changed
app_send_data	hello_sensor_conn_idle_timeout
app_interrupt_handler	hello_sensor_send_message
app_tx_ack_timeout	hello_sensor_gatts_conn_status_cb
app_trace_callback	hello_sensor_gatts_req_cb
	hello_sensor_gatts_callback

In addition to the API shown in Table 5, the *spp.c*, *hello_sensor.c*/h, and *wiced_bt_cfg.c* source files contain the following databases:

- app_sdp_db[] defines SDP data base for SPP profile of Bluetooth BR/EDR
- hello_sensor_gatt_database[] defines GATT database for a custom profile of BLE

The next function to look at is `application_init`; it runs after the ROM-driven Bluetooth stack initialization is succesful. (the `BTM_ENABLED_EVT` event is generated by the stack). This function sets up the application-specific behavior, initializes the SPP library, initializes the SDP database for Bluetooth BR/EDR, initializes the GATT database, registers application callbacks to occur when various link layer, GAP, or GATT events occur within the stack, initializes timers, and configures the button and LED based on the application requirements as shown in Code 16.

Code 16. `application_init`: Application-Driven Initialization

```
 void application_init(void)
{
    wiced_bt_gatt_status_t gatt_status;
    wiced_result_t         result;

    /* Initialize wiced app */
    wiced_bt_app_init();

#if SEND_DATA_ON_INTERRUPT
    /* Configure the button available on the platform */
    wiced_hal_gpio_configure_pin( WICED_GPIO_BUTTON, WICED_GPIO_BUTTON_SETTINGS( GPIO_EN_INT_RISING_EDGE ),
WICED_GPIO_BUTTON_DEFAULT_STATE );
    wiced_hal_gpio_register_pin_for_interrupt(WICED_GPIO_BUTTON, app_interrupt_handler, NULL);

    // init timer that we will use for the rx data flow control.
    wiced_init_timer(&app_tx_timer, app_tx_ack_timeout, 0, WICED_MILLI_SECONDS_TIMER);
#endif

    app_write_eir();

    // Initialize RFCOMM.  We will not be using application buffer pool and will rely on the
    // stack pools configured in the wiced_bt_cfg.c
    wiced_bt_rfcomm_init(MAX_TX_BUFFER, 1);

    // Initialize SPP library
    wiced_bt_spp_startup(&spp_reg);

#ifdef HCI_TRACE_OVER_TRANSPORT
    // There is a virtual HCI interface between upper layers of the stack and
    // the controller portion of the chip with lower layers of the BT stack.
    // Register with the stack to receive all HCI commands, events and data.
    wiced_bt_dev_register_hci_trace(app_trace_callback);
#endif
    /* create SDP records */
    wiced_bt_sdp_db_init((uint8_t *)app_sdp_db, sizeof(app_sdp_db));

    // This application will always configure device connectable and discoverable
    wiced_bt_dev_set_discoverability(BTM_GENERAL_DISCOVERABLE, 0x0012, 0x0800);
    wiced_bt_dev_set_connectability(BTM_CONNECTABLE, 0x0012, 0x0800);

#if SEND_DATA_ON_TIMEOUT
    /* Starting the app timers, seconds timer and the ms timer  */
    wiced_bt_app_start_timer(1, 0, app_timeout, NULL);
#endif

    hello_sensor_application_init();
}


void hello_sensor_application_init( void )
{
    wiced_bt_gatt_status_t gatt_status;
    wiced_result_t         result;

    /* Initialize wiced app */
//    wiced_bt_app_init();
    wiced_bt_app_led_init();

    /* Configure buttons available on the platform (pin should be configured before registering interrupt
                handler ) */
//    wiced_hal_gpio_configure_pin( WICED_GPIO_BUTTON, WICED_GPIO_BUTTON_SETTINGS( GPIO_EN_INT_RISING_EDGE
//                ), WICED_GPIO_BUTTON_DEFAULT_STATE );
//    wiced_hal_gpio_register_pin_for_interrupt( WICED_GPIO_BUTTON, hello_sensor_interrupt_handler, NULL );

    /* Register with stack to receive GATT callback */
    gatt_status = wiced_bt_gatt_register( hello_sensor_gatts_callback );

    WICED_BT_TRACE( "wiced_bt_gatt_register: %d\n", gatt_status );
```

```
    /*  Tell stack to use our GATT database */
    gatt_status =  wiced_bt_gatt_db_init( hello_sensor_gatt_database, sizeof(hello_sensor_gatt_database) );

    WICED_BT_TRACE("wiced_bt_gatt_db_init %d\n", gatt_status);

    /* Allow peer to pair */
    wiced_bt_set_pairable_mode(WICED_TRUE, 0);

#ifdef ENABLE_HCI_TRACE
    /* Register callback for receiving hci traces */
    wiced_bt_dev_register_hci_trace( hello_sensor_hci_trace_cback );
#endif

    /* Starting the app timers , seconds timer and the ms timer  */
    wiced_bt_app_start_timer( HELLO_SENSOR_APP_TIMEOUT_IN_SECONDS, HELLO_SENSOR_APP_FINE_TIMEOUT_IN_MS,
        hello_sensor_timeout, hello_sensor_fine_timeout );

    /* Enable privacy to advertise with RPA */
 //   wiced_bt_ble_enable_privacy ( WICED_TRUE );

    /* Load the address resolution DB with the keys stored in the NVRAM */
 //   hello_sensor_load_keys_for_address_resolution();

    /* Set the advertising params and make the device discoverable */
    hello_sensor_set_advertisement_data();

    result =  wiced_bt_start_advertisements( BTM_BLE_ADVERT_UNDIRECTED_HIGH, 0, NULL );

    WICED_BT_TRACE( "wiced_bt_start_advertisements %d\n", result );

    /*
     * Set flag_stay_connected to remain connected after all messages are sent
     * Reset flag to 0, to disconnect
     */
    hello_sensor_state.flag_stay_connected = 1;
}
```

Take note of how the `application_init()` initialization function ends, which occurs right before the stack gets execution control again.

For BLE operation, Code 16 sets the advertisement data and begins fast undirected and connectable advertisements. A remote device is able to scan the peripheral device and issue a connect request, which will trigger the callback `hello_sensor_gatts_callback` (the `hello_sensor_gatts_callback` event handler registered in Code 16) and allow further application-specific behavior for BLE operation.

For Bluetooth BR/EDR operation, the code sets the inquiry response data and makes the device discoverable. A remote host device performs inquiry and paging process, and issues a connect request to the Bluetooth device which triggers `spp_connection_up_callback`, and allows further application-specific behavior for Bluetooth BR/EDR operation.

7.4.2 Bluetooth Stack Event Handlers

A single Bluetooth event handler function `app_management_callback` is registered to receive stack callback events for both Bluetooth BR/EDR and BLE connections. The Bluetooth stack initialization complete event is triggered by the `BTM_ENABLED_EVT` event. Application features are initialized, and necessary callbacks are registered in `BTM_ENABLED_EVT`. Security/ pairing capabilities are exchanged and link keys are stored after appropriate events are triggered by the stack in `app_management_callback`.

Code 17. `App_manangement_callback`: Bluetooth Stack Event Handler

```
wiced_result_t app_management_callback(wiced_bt_management_evt_t event, wiced_bt_management_evt_data_t
*p_event_data)
{
    wiced_result_t                      result = WICED_BT_SUCCESS;
    wiced_bt_dev_status_t               dev_status;
    wiced_bt_dev_pairing_info_t*        p_pairing_info;
    wiced_bt_dev_encryption_status_t*   p_encryption_status;
    int                                 bytes_written, bytes_read;
    wiced_bt_power_mgmt_notification_t* p_power_mgmt_notification;
    wiced_bt_ble_advert_mode_t         *p_mode;
```

```
WICED_BT_TRACE("bt_management_callback 0x%02x\n", event);

switch(event)
{
/* Bluetooth  stack enabled */
case BTM_ENABLED_EVT:
    application_init();
    hello_sensor_application_init();
    WICED_BT_TRACE("Free mem:%d", wiced_memory_get_free_bytes());
    break;

case BTM_DISABLED_EVT:
    break;

case BTM_PIN_REQUEST_EVT:
    WICED_BT_TRACE("remote address= %B\n", p_event_data->pin_request.bd_addr);
    wiced_bt_dev_pin_code_reply(*p_event_data->pin_request.bd_addr,result/*WICED_BT_SUCCESS*/,4,
                    &pincode[0]);
    break;

case BTM_USER_CONFIRMATION_REQUEST_EVT:
    /* This application always confirms peer's attempt to pair */
    wiced_bt_dev_confirm_req_reply (WICED_BT_SUCCESS, p_event_data->user_confirmation_request.bd_addr);
    break;

case BTM_PAIRING_IO_CAPABILITIES_BR_EDR_REQUEST_EVT:
    /* This application supports only Just Works pairing */
    WICED_BT_TRACE("BTM_PAIRING_IO_CAPABILITIES_REQUEST_EVT bda %B\n", p_event_data-
                                        >pairing_io_capabilities_br_edr_request.bd_addr);
    p_event_data->pairing_io_capabilities_br_edr_request.local_io_cap   = BTM_IO_CAPABILITIES_NONE;
    p_event_data->pairing_io_capabilities_br_edr_request.auth_req       =
                                                BTM_AUTH_SINGLE_PROFILE_GENERAL_BONDING_NO;
    break;

case BTM_PAIRING_COMPLETE_EVT:
    p_pairing_info = &p_event_data->pairing_complete.pairing_complete_info;
    WICED_BT_TRACE("Pairing Complete: %d\n", p_pairing_info->br_edr.status);
    result = WICED_BT_USE_DEFAULT_SECURITY;
    break;

case BTM_ENCRYPTION_STATUS_EVT:
    p_encryption_status = &p_event_data->encryption_status;
    WICED_BT_TRACE("Encryption Status Event: bd (%B) res %d\n", p_encryption_status->bd_addr,
                                                                p_encryption_status->result);
    hello_sensor_encryption_changed( p_encryption_status->result, p_encryption_status->bd_addr );
    break;

case BTM_SECURITY_REQUEST_EVT:
    wiced_bt_ble_security_grant( p_event_data->security_request.bd_addr, WICED_BT_SUCCESS );
    WICED_BT_TRACE( "BTM_SECURITY_REQUEST_EVT: bd ( %B ) ", p_event_data->security_request.bd_addr);
break;


case BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT:
    /* This application supports a single paired host, we can save keys under the same NVRAM ID
          overwriting previous pairing if any */
    bt_write_nvram(SPP_NVRAM_ID, sizeof(wiced_bt_device_link_keys_t), &p_event_data-
          >paired_device_link_keys_update);
    break;

case  BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT:
    /* read existing key from the NVRAM  */
    if (bt_read_nvram(SPP_NVRAM_ID, &p_event_data->paired_device_link_keys_request,
            sizeof(wiced_bt_device_link_keys_t)) != 0)
    {
        result = WICED_BT_SUCCESS;
    }
    else
    {
        result = WICED_BT_ERROR;
        WICED_BT_TRACE("Key retrieval failure\n");
    }
    break;

case BTM_POWER_MANAGEMENT_STATUS_EVT:
    p_power_mgmt_notification = &p_event_data->power_mgmt_notification;
```

```
        WICED_BT_TRACE("Power mgmt status event: bd (%B) status:%d hci_status:%d\n",
             p_power_mgmt_notification->bd_addr, \
               p_power_mgmt_notification->status, p_power_mgmt_notification->hci_status);
        break;

    case BTM_BLE_ADVERT_STATE_CHANGED_EVT:
        p_mode = &p_event_data->ble_advert_state_changed;
        WICED_BT_TRACE( "Advertisement State Change: %d\n", *p_mode);
        if ( *p_mode == BTM_BLE_ADVERT_OFF )
        {
            hello_sensor_advertisement_stopped();
        }
        break;

    case BTM_SECURITY_REQUEST_EVT:
        wiced_bt_ble_security_grant( p_event_data->security_request.bd_addr, WICED_BT_SUCCESS );
    break;

    case BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT:
        p_event_data->pairing_io_capabilities_ble_request.local_io_cap   = BTM_IO_CAPABILITIES_NONE;
        p_event_data->pairing_io_capabilities_ble_request.oob_data    = BTM_OOB_NONE;
        p_event_data->pairing_io_capabilities_ble_request.auth_req    = BTM_LE_AUTH_REQ_SC_BOND;
        p_event_data->pairing_io_capabilities_ble_request.max_key_size  = 0x10;
        p_event_data->pairing_io_capabilities_ble_request.init_keys     =
                                    BTM_LE_KEY_PENC|BTM_LE_KEY_PID|BTM_LE_KEY_PCSRK|BTM_LE_KEY_LENC;
        p_event_data->pairing_io_capabilities_ble_request.resp_keys     =
                                    BTM_LE_KEY_PENC|BTM_LE_KEY_PID|BTM_LE_KEY_PCSRK|BTM_LE_KEY_LENC;
        break;

    default:
        result = WICED_BT_USE_DEFAULT_SECURITY;
        break;
    }
    return result;
}
```

7.4.3 BLE Functions

Advertisement data can be modified based on application requirements by adding/removing necessary elements in the `wiced_bt_ble_advert_elem_t` structure as shown in Code 18. In this example, BLE device advertisement name is set to “Hello”.

Code 18. `hello_sensor_set_advertisement_data`: BLE Advertisement Data

```
void hello_sensor_set_advertisement_data(void)
{
    wiced_bt_ble_advert_elem_t adv_elem[4];
    uint8_t num_elem = 0;
    uint8_t flag = BTM_BLE_GENERAL_DISCOVERABLE_FLAG | BTM_BLE_BREDR_NOT_SUPPORTED;
    /*
     * hci_control_le_local_name - Advertising Complete Name
     * Note : Max Length is 8 bytes for the Advertisement Name, rest of 21 bytes are used for
     *  BTM_BLE_ADVERT_TYPE_FLAG,BTM_BLE_ADVERT_TYPE_128SRV_COMPLETE
     */
    uint8_t  hci_control_le_local_name[]      =   "Hello" ; //Alternate way to declare {'h', 'e', 'l', 'l',
                                                            'o', 0x00, 0x00};
    uint8_t hello_service_uuid[LEN_UUID_128] = { UUID_HELLO_SERVICE };

    adv_elem[num_elem].advert_type  = BTM_BLE_ADVERT_TYPE_FLAG;
    adv_elem[num_elem].len          = sizeof(uint8_t);
    adv_elem[num_elem].p_data       = &flag;
    num_elem++;

    adv_elem[num_elem].advert_type  = BTM_BLE_ADVERT_TYPE_128SRV_COMPLETE;
    adv_elem[num_elem].len          = sizeof(hello_service_uuid);
    adv_elem[num_elem].p_data       = ( uint8_t* )hello_service_uuid;
    num_elem++;

    adv_elem[num_elem].advert_type  = BTM_BLE_ADVERT_TYPE_NAME_COMPLETE;
    adv_elem[num_elem].len          = strlen(hci_control_le_local_name);
    adv_elem[num_elem].p_data       = ( uint8_t* )hci_control_le_local_name;
    num_elem++;

    wiced_bt_ble_set_raw_advertisement_data(num_elem, adv_elem);
}
```

Next, check the `hello_sensor_advertisement_stopped` funtion, which is called in the Bluetooth stack event `BTM_BLE_ADVERT_STATE_CHANGED_EVT`. The Bluetooth stack triggers this event automatically after the BLE advertising period timeout elapses. The default advertisement timeouts for fast ('high') and slow ('low') rates are defined in the `wiced_bt_cfg_settings` structure near the top of *wiced_bt_cfg.c* file, and are 30 and 300 seconds respectively. The code in this event handler ensures that advertisements always resume automatically whenever either mode times out.

Code 19. `hello_sensor_advertisement_stopped`: Advertisement Timeout Event Handler

```
void hello_sensor_advertisement_stopped( void )
{
    wiced_result_t result;

    if ( hello_sensor_state.flag_stay_connected && !hello_sensor_state.conn_id )
    {
        result =  wiced_bt_start_advertisements( BTM_BLE_ADVERT_UNDIRECTED_LOW, 0, NULL );
        WICED_BT_TRACE( "wiced_bt_start_advertisements: %d\n", result );
    }
    else
    {
        WICED_BT_TRACE( "ADV stop\n");
    }
}
```

After this funtion, there are two functions which concern bonding and encryption. First is the `hello_sensor_smp_bond_result` funtion shown in Code 20, which occurs after the remote peer successfully bonds with the local device. This function stores the bonded BD (Bluetooth Device) address of remote device in the hello_sensor_`hostinfo` structure, then writes that structure into the non-volatile memory for later retrieval.

Code 20. `hello_sensor_smp_bond_result`: Bonding Event Handler

```
void hello_sensor_smp_bond_result( uint8_t result )
{
    wiced_result_t status;
    uint8_t written_byte = 0;
    WICED_BT_TRACE( "hello_sensor, bond result: %d\n", result );

    /* Bonding success */
    if ( result == WICED_BT_SUCCESS )
    {
        /* Pack the data to be stored into the hostinfo structure */
        memcpy( hello_sensor_hostinfo.bdaddr, hello_sensor_state.remote_addr, sizeof( BD_ADDR ) );

        /* Write to NVRAM */
        written_byte = wiced_hal_write_nvram( HELLO_SENSOR_VS_ID, sizeof(hello_sensor_hostinfo),
(uint8_t*)&hello_sensor_hostinfo, &status );
        WICED_BT_TRACE("NVRAM write: %d\n", written_byte);
    }
}
```

Another security-related function is the `hello_sensor_encryption_changed` funtion, which is called when the stack encryption state change event, `BTM_ENCRYPTION_STATUS_EVT`, is triggered. This typically occurs during the bonding process after the link is successfully encrypted. While it is not strictly necessary to catch this event, the existing hello_sensor sample application has a function to handle encryption change actions. In this case, outstanding messages are pushed to the stack. An idle timer is started in this funtion to disconnect the link after the specified timeout.

Code 21. `hello_sensor_encryption_changed`: Encryption State Change

```
void hello_sensor_encryption_changed( wiced_result_t result, uint8_t* bd_addr )
{
    WICED_BT_TRACE( "encryp change bd ( %B ) res: %d ", hello_sensor_hostinfo.bdaddr,  result);

    /* Connection has been encrypted meaning that we have correct/paired device
     * restore values in the database
     */
    wiced_hal_read_nvram( HELLO_SENSOR_VS_ID, sizeof(hello_sensor_hostinfo),
(uint8_t*)&hello_sensor_hostinfo, &result );

    // If there are outstanding messages that we could not send out because
    // connection was not up and/or encrypted, send them now.  If we are sending
```

```
    // indications, we can send only one and need to wait for ack.
    while ( ( hello_sensor_state.num_to_write != 0 ) && !hello_sensor_state.flag_indication_sent )
    {
        hello_sensor_state.num_to_write--;
        hello_sensor_send_message();
    }

    // If configured to disconnect after delivering data, start idle timeout
    // to do disconnection
    if ( ( !hello_sensor_state.flag_stay_connected ) && !hello_sensor_state.flag_indication_sent  )
    {
        wiced_bt_app_start_conn_idle_timer( HELLO_SENSOR_CONN_IDLE_TIMEOUT_IN_SECONDS,
hello_sensor_conn_idle_timeout );
    }
}
```

After security-related events, the `hello_sensor_gatts_req_cb` function shown in Code 22 is called on receiving `GATT_ATTRIBUTE_REQUEST_EVT` event from the connected remote client. `hello_sensor_gatts_req_cb` processes GATT read, GATT write, GATT write execute, MTU request, and indication confirmation actions.

Code 22. `hello_sensor_gatts_req_cb`: Top-Level GATT Function

```
wiced_bt_gatt_status_t hello_sensor_gatts_req_cb( wiced_bt_gatt_attribute_request_t *p_data )
{
    wiced_result_t result = WICED_BT_GATT_INVALID_PDU;

    WICED_BT_TRACE( "hello_sensor_gatts_req_cb. conn %d, type %d\n", p_data->conn_id, p_data->request_type );

    switch ( p_data->request_type )
    {
    case GATTS_REQ_TYPE_READ:
        result = hello_sensor_gatts_req_read_handler( p_data->conn_id, &(p_data->data.read_req) );
        break;

    case GATTS_REQ_TYPE_WRITE:
        result = hello_sensor_gatts_req_write_handler( p_data->conn_id, &(p_data->data.write_req) );
        break;

    case GATTS_REQ_TYPE_WRITE_EXEC:
        result = hello_sensor_gatts_req_write_exec_handler( p_data->conn_id, p_data->data.exec_write );
        break;

    case GATTS_REQ_TYPE_MTU:
        result = hello_sensor_gatts_req_mtu_handler( p_data->conn_id, p_data->data.mtu );
        break;

    case GATTS_REQ_TYPE_CONF:
        result = hello_sensor_gatts_req_conf_handler( p_data->conn_id, p_data->data.handle );
        break;

   default:
        break;
    }
}
```

For this application, `hello_sensor_gatts_req_read_handler`, `hello_sensor_gatts_req_write_handler`, and `hello_sensor_gatts_req_conf_handler` are used to process read, write, and indication confirmation requests.

7.4.3.1 GATT Read Request

The Client can issue a read request to retreive data from any charactersitic which has read property enabled in the GATT database. The `hello_sensor_gatts_req_read_handler` function processes the read request command from the connected remote Client device. In this example, this funtion sends dummy battery level data when the remote Client issues a read request to the battery level charactersitc handle(0x62), `HANDLE_HSENS_BATTERY_SERVICE_CHAR_LEVEL_VAL`. Valid characteritic handler verification is done at the beginning of the funtion; a negative response is sent for read requests wtih invalid characteristic handles. Reqeusted data is packaged into an appropriate form and passed for further processing to the stack.

Code 23. BLE GATT Read Request

```
wiced_bt_gatt_status_t hello_sensor_gatts_req_read_handler( uint16_t conn_id, wiced_bt_gatt_read_t *
p_read_data )
{
    attribute_t *puAttribute;
    int          attr_len_to_copy;

    if ( ( puAttribute = hello_sensor_get_attribute(p_read_data->handle) ) == NULL)
    {
        WICED_BT_TRACE("read_hndlr attr not found hdl:%x\n", p_read_data->handle );
        return WICED_BT_GATT_INVALID_HANDLE;
    }

    /* Dummy battery value read increment */
    if( p_read_data->handle == HANDLE_HSENS_BATTERY_SERVICE_CHAR_LEVEL_VAL)
    {
        if ( hello_sensor_state.battery_level++ > 5)
        {
            hello_sensor_state.battery_level = 0;
        }
    }


    attr_len_to_copy = puAttribute->attr_len;

    WICED_BT_TRACE("read_hndlr conn_id:%d hdl:%x offset:%d len:%d\n", conn_id, p_read_data->handle,
p_read_data->offset, attr_len_to_copy );

    if ( p_read_data->offset >= puAttribute->attr_len )
    {
        attr_len_to_copy = 0;
    }

    if ( attr_len_to_copy != 0 )
    {
        uint8_t *from;
        int      to_copy = attr_len_to_copy - p_read_data->offset;


        if ( to_copy > *p_read_data->p_val_len )
        {
            to_copy = *p_read_data->p_val_len;
        }

        from = ((uint8_t *)puAttribute->p_attr) + p_read_data->offset;
        *p_read_data->p_val_len = to_copy;

        memcpy( p_read_data->p_val, from, to_copy);
    }

    return WICED_BT_GATT_SUCCESS;
}
```

7.4.3.2 GATT Write Request

The Client can issue a write request to update the value on any characteristic which has the write property enabled in the GATT database. The `hello_sensor_gatts_req_write_handler` function processes the GATT write request sent by the connected remote Client device.

- This funtion receives the Client Characterstic Configuration information on the characteristic handle (0x2B) `HANDLE_HSENS_SERVICE_CHAR_CFG_DESC`, and stores it in the nonvoltatile memory. The Client configuration data determines the usage of notifications or indications for data transfer from a Peripheral to Central in this example project.

- This function also receives data on a custom characteristic handle `HANDLE_HSENS_SERVICE_CHAR_BLINK_VAL` from the connected remote Client. *A* byte value received on this handle determines number of LED blinks on the evaluation board.

Code 24. BLE GATT Write Request

```
wiced_bt_gatt_status_t hello_sensor_gatts_req_write_handler( uint16_t conn_id, wiced_bt_gatt_write_t *
p_data )
{
    wiced_bt_gatt_status_t result    = WICED_BT_GATT_SUCCESS;
    uint8_t               *p_attr   = p_data->p_val;
    uint8_t               nv_update = WICED_FALSE;

    WICED_BT_TRACE("write_handler: conn_id:%d hdl:0x%x prep:%d offset:%d len:%d\n ", conn_id, p_data-
>handle, p_data->is_prep, p_data->offset, p_data->val_len );

    switch ( p_data->handle )
    {
    /* By writing into Characteristic Client Configuration descriptor
     * peer can enable or disable notification or indication */
    case HANDLE_HSENS_SERVICE_CHAR_CFG_DESC:
        if ( p_data->val_len != 2 )
        {
            return WICED_BT_GATT_INVALID_ATTR_LEN;
        }
        hello_sensor_hostinfo.characteristic_client_configuration = p_attr[0] | ( p_attr[1] << 8 );
        nv_update = WICED_TRUE;
        break;

    case HANDLE_HSENS_SERVICE_CHAR_BLINK_VAL:
        if ( p_data->val_len != 1 )
        {
            return WICED_BT_GATT_INVALID_ATTR_LEN;
        }
        hello_sensor_hostinfo.number_of_blinks = p_attr[0];
        if ( hello_sensor_hostinfo.number_of_blinks != 0 )
        {
            WICED_BT_TRACE( "hello_sensor_write_handler:num blinks: %d\n",
hello_sensor_hostinfo.number_of_blinks );
            wiced_bt_app_hal_led_blink( WICED_PLATFORM_LED_1, 250, 250,
hello_sensor_hostinfo.number_of_blinks );
            nv_update = WICED_TRUE;
        }
        break;

    default:
        result = WICED_BT_GATT_INVALID_HANDLE;
        break;
    }

    if ( nv_update )
    {
        wiced_result_t rc;
        int bytes_written = wiced_hal_write_nvram( HELLO_SENSOR_VS_ID, sizeof(hello_sensor_hostinfo),
(uint8_t*)&hello_sensor_hostinfo, &rc );
        WICED_BT_TRACE("NVRAM write:%d rc:%d", bytes_written, rc);
    }

    return result;
}
```

7.4.3.3 Indication Confirmation

If the Client chooses indications instead of notifications, which is determined by the Client configuration data received as part of the GATT write request from the Client shown in Code 24, the Peripheral should wait for confirmation before sending the next indication.

Code 25. BLE Indication Confirmation

```
wiced_bt_gatt_status_t hello_sensor_gatts_req_conf_handler( uint16_t conn_id, uint16_t handle )
{
    WICED_BT_TRACE( "hello_sensor_indication_cfm, conn %d hdl %d\n", conn_id, handle );

    if ( !hello_sensor_state.flag_indication_sent )
    {
        WICED_BT_TRACE("Hello: Wrong Confirmation!");
        return WICED_BT_GATT_SUCCESS;
    }
```

```
    hello_sensor_state.flag_indication_sent = 0;

    /* We might need to send more indications */
    if ( hello_sensor_state.num_to_write )
    {
        hello_sensor_state.num_to_write--;
        hello_sensor_send_message();
    }
    /* if we sent all messages, start connection idle timer to disconnect */
    if ( !hello_sensor_state.flag_stay_connected && !hello_sensor_state.flag_indication_sent )
    {
       wiced_bt_app_start_conn_idle_timer( HELLO_SENSOR_CONN_IDLE_TIMEOUT_IN_SECONDS,
                                                                    hello_sensor_conn_idle_timeout );
    }

    return WICED_BT_GATT_SUCCESS;

}
```

Two other callbacks remain in `hello_sensor_gatts_req_cb`, which are not absolutely necessary for this example. Place holders for the rest of two funtions are provided in this file, `hello_sensor_gatts_req_write_exec_handler` and `hello_sensor_gatts_req_mtu_handler`.

The next funtion to look at is `hello_sensor_gatts_conn_status_cb`. This callback funtion is registered in the `hello_sensor_application_init` during the intialization phase. The Bluetooth stack triggers this callback funtion on BLE connection or disconnection.

Code 26. `hello_sensor_gatts_callback`

```
wiced_bt_gatt_status_t hello_sensor_gatts_callback( wiced_bt_gatt_evt_t event, wiced_bt_gatt_event_data_t
*p_data)
{
    wiced_bt_gatt_status_t result = WICED_BT_GATT_INVALID_PDU;

    switch(event)
    {
    case GATT_CONNECTION_STATUS_EVT:
        result = hello_sensor_gatts_conn_status_cb( &p_data->connection_status );
        break;

    case GATT_ATTRIBUTE_REQUEST_EVT:
        result = hello_sensor_gatts_req_cb( &p_data->attribute_request );
        break;

    default:
        break;
    }

    return result;
}
```

7.4.3.4 Connection Up

In the connection up funtion `hello_sensor_gatts_connection_up`, the BD Address of the remote Client and connection ID are copied into the local RAM and also saved in NVRAM. Advertisement is stopped as the Peripheral device is now connected with the remote Central device.

Code 27. `hello_sensor_gatts_connection_up`

```
wiced_bt_gatt_status_t hello_sensor_gatts_connection_up( wiced_bt_gatt_connection_status_t *p_status )
{
    wiced_result_t result;
    uint8_t        bytes_written = 0;

    WICED_BT_TRACE( "hello_sensor_conn_up %B id:%d\n:", p_status->bd_addr, p_status->conn_id);

    /* Update the connection handler.  Save address of the connected device. */
    hello_sensor_state.conn_id = p_status->conn_id;
    memcpy(hello_sensor_state.remote_addr, p_status->bd_addr, sizeof(BD_ADDR));

    /* Stop advertising */
    result =  wiced_bt_start_advertisements( BTM_BLE_ADVERT_OFF, 0, NULL );
```

```
    WICED_BT_TRACE( "Stopping Advertisements%d\n", result );

    /* Stop idle timer */
    wiced_bt_app_stop_conn_idle_timer();

    /* Updating the bd address in the  host info in NVRAM  */
    memcpy( hello_sensor_hostinfo.bdaddr, p_status->bd_addr, sizeof( BD_ADDR ) );

    hello_sensor_hostinfo.characteristic_client_configuration = 0;
    hello_sensor_hostinfo.number_of_blinks                    = 0;

    /* Save the  host info in NVRAM  */
    bytes_written = wiced_hal_write_nvram( HELLO_SENSOR_VS_ID, sizeof(hello_sensor_hostinfo),
(uint8_t*)&hello_sensor_hostinfo, &result );
    WICED_BT_TRACE("NVRAM write %d\n", bytes_written);

    return WICED_BT_GATT_SUCCESS;
}
```

7.4.3.5 Connection Down

In the connection down funtion, `hello_sensor_gatts_connection_down`, the BD Address of the remote client is erased from NVRAM, the connection ID is reset to '0', and a low-duty advertisement is started.

Code 28. `hello_sensor_gatts_connection_down`

```
wiced_bt_gatt_status_t hello_sensor_gatts_connection_down( wiced_bt_gatt_connection_status_t *p_status )
{
    wiced_result_t result;

    WICED_BT_TRACE( "connection_down %B conn_id:%d reason:%d\n", hello_sensor_state.remote_addr, p_status-
>conn_id, p_status->reason );

    /* Resetting the device info */
    memset( hello_sensor_state.remote_addr, 0, 6 );
    hello_sensor_state.conn_id = 0;

    /*
     * If we are configured to stay connected, disconnection was
     * caused by the peer, start low advertisements, so that peer
     * can connect when it wakes up
     */
    if ( hello_sensor_state.flag_stay_connected )
    {
        result =  wiced_bt_start_advertisements( BTM_BLE_ADVERT_UNDIRECTED_LOW, 0, NULL );
        WICED_BT_TRACE( "wiced_bt_start_advertisements %d\n", result );
    }
    return WICED_BT_SUCCESS;
}
```

Custom application-specific features can be implemented in timeout funtions according to requirements,

- `hello_sensor_timeout` – seconds timer
- `hello_sensor_fine_timeout` – millisecond timer

  In this example project, button interrupt is not enabled for sending BLE notifications/indications and the privacy feature is disabled; therefore, the following funtions are not used:

- `hello_sensor_send_message`
- `hello_sensor_interrupt_handler`
- `hello_sensor_load_keys_for_address_resolution`

7.4.4 Bluetooth Functions

Callback functions for specific Bluetooth BR/EDR events, connection up, connection down, and received data are registered during the SPP library initialization inside the `application_init()` function (see Code 16). The `wiced_bt_spp_reg_t` structure with appropriate callbacks and `wiced_bt_spp_startup()` are shown in Code 29.

Code 29. *spp.c*: Structure with Details of Callbacks Needed for SPP Application and Function to Initialize SPP Library

```
wiced_bt_spp_reg_t spp_reg =
{
    SPP_RFCOMM_SCN,                     /* RFCOMM service channel number for SPP connection */
    MAX_TX_BUFFER,                      /* RFCOMM MTU for SPP connection */
    spp_connection_up_callback,         /* SPP connection established */
    NULL,                               /* SPP connection establishment failed, not used because this app
                                           never initiates connection */
    NULL,                               /* SPP service not found, not used because this app never initiates
                                           connection */
    spp_connection_down_callback,       /* SPP connection disconnected */
    spp_rx_data_callback,               /* Data packet received */
};
```

```
 // Initialize SPP library
 wiced_bt_spp_startup(&spp_reg);
```

The Inquiry response data for Bluetooth BR/EDR can be set in the `app_write_eir function`. In this example, the device name and UUID of the SPP service are published as part of the inquiry response.

Code 30. `App_write_eir`: Inquiry Response Data

```
void app_write_eir(void)
{
    uint8_t *pBuf;
    uint8_t *p;
    uint8_t length;
    uint16_t eir_length;

    pBuf = (uint8_t *)wiced_bt_get_buffer(WICED_EIR_BUF_MAX_SIZE);
    WICED_BT_TRACE("hci_control_write_eir %x\n", pBuf);

    if (!pBuf)
    {
        WICED_BT_TRACE("app_write_eir %x\n", pBuf);
        return;
    }

    p = pBuf;

    length = strlen((char *)wiced_bt_cfg_settings.device_name);

    *p++ = length + 1;
    *p++ = BT_EIR_COMPLETE_LOCAL_NAME_TYPE;        // EIR type full name
    memcpy(p, wiced_bt_cfg_settings.device_name, length);
    p += length;

    *p++ = 2 + 1;                                  // Length of 16 bit services
    *p++ = BT_EIR_COMPLETE_16BITS_UUID_TYPE;       // 0x03 EIR type full list of 16 bit service UUIDs
    *p++ = UUID_SERVCLASS_SERIAL_PORT & 0xff;
    *p++ = (UUID_SERVCLASS_SERIAL_PORT >> 8) & 0xff;

    *p++ = 0;                                      // end of EIR Data is 0

    eir_length = (uint16_t) (p - pBuf);

    // print EIR data
    wiced_bt_trace_array("EIR :", pBuf, MIN(p-pBuf, 100));
    wiced_bt_dev_write_eir(pBuf, eir_length);

    return;
}
```

The next function to look for are connection up and connection down callback functions triggered by the Bluetooth stack on successful connection or disconnection from the remote Bluetooth BR/EDR master. The Bluetooth connection handle is saved by the connection up callback; it is reset to '0' by the connection down callback as shown in Code 30.

Code 31. Connection Up and Connection Down Call Back Functions

```
/*
 * SPP connection up callback
 */
void spp_connection_up_callback(uint16_t handle, uint8_t* bda)
{
    WICED_BT_TRACE("%s handle:%d address:%B\n", __FUNCTION__, handle, bda);
    spp_handle = handle;
}
/*
 * SPP connection down callback
 */
void spp_connection_down_callback(uint16_t handle)
{
    WICED_BT_TRACE("%s handle:%d\n", __FUNCTION__, handle);
    spp_handle = 0;
}
```

The next function to look at is `spp_rx_data_callback`. The Bluetooth stack triggers this callback function when data is received from the master Bluetooth BR/EDR device over the SPP profile. The received data is sent back to the host if the `LOOPBACK_DATA` macro is set to '1'. This macro is disabled in this example; so uncomment `LOOPBACK_DATA` based on application requirement.

Code 32. `spp_rx_data_callback`: Callback for Received Data over SPP Profile

```
wiced_bool_t spp_rx_data_callback(uint16_t handle, uint8_t* p_data, uint32_t data_len)
{
    int i;
//    wiced_bt_buffer_statistics_t buffer_stats[4];

//    wiced_bt_get_buffer_usage (buffer_stats, sizeof(buffer_stats));

//    WICED_BT_TRACE("0:%d/%d 1:%d/%d 2:%d/%d 3:%d/%d\n", buffer_stats[0].current_allocated_count,
buffer_stats[0].max_allocated_count,
//                   buffer_stats[1].current_allocated_count, buffer_stats[1].max_allocated_count,
//                   buffer_stats[2].current_allocated_count, buffer_stats[2].max_allocated_count,
//                   buffer_stats[3].current_allocated_count, buffer_stats[3].max_allocated_count);

//    wiced_result_t wiced_bt_get_buffer_usage (&buffer_stats, sizeof(buffer_stats));

    WICED_BT_TRACE("%s handle:%d len:%d %02x-%02x\n", __FUNCTION__, handle, data_len, p_data[0],
p_data[data_len - 1]);

#if LOOPBACK_DATA
    wiced_bt_spp_send_session_data(handle, p_data, data_len);
#endif
    return WICED_TRUE;
}
```

The next functions to look at are read and write NVRAM. Pairing keys are stored in the NVRAM of the Peripheral/slave Bluetooth device after successful pairing. NVRAM write and read operations are performed to store pairing keys and to retrieve them when the Bluetooth stack triggers the `BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT` and `BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT` events respectively.

Code 33. NVRAM Access Function

```
int bt_write_nvram(int nvram_id, int data_len, void *p_data)
{
    wiced_result_t  result;
    int             bytes_written = wiced_hal_write_nvram(nvram_id, data_len, (uint8_t*)p_data, &result);

    WICED_BT_TRACE("NVRAM ID:%d written :%d bytes result:%d\n", nvram_id, bytes_written, result);
    return (bytes_written);
}



int bt_read_nvram(int nvram_id, void *p_data, int data_len)
{
    uint16_t        read_bytes = 0;
    wiced_result_t  result;

    if (data_len >= sizeof(wiced_bt_device_link_keys_t))
```

```
    {
        read_bytes = wiced_hal_read_nvram(nvram_id, sizeof(wiced_bt_device_link_keys_t), p_data, &result);
        WICED_BT_TRACE("NVRAM ID:%d read out of %d bytes:%d result:%d\n", nvram_id,
sizeof(wiced_bt_device_link_keys_t), read_bytes, result);
    }
    return (read_bytes);
}
```

In this example, data is transferred to the host over the SPP profile on a button interrupt or timeout. The `SEND_DATA_ON_INTERRUPT` macro is enabled in this example. The `app_interrupt_handler` function is the registered callback function for Button interrupt; the data transmit process is initiated in this callback function. Every button press triggers a 1 MB of known data transmission to the remote Host when the slave device is in connected state. In case if 1 MB data transfer was interrupted due to interference or any other reason, a timer callback is registered with a timeout to begin the transfer of the remaining data over the SPP profile.

Code 34. `app_interrupt_handler`: Send Data on Button Interrupt

```
#if SEND_DATA_ON_INTERRUPT
void app_send_data(void)
{
    int i;

    while ((spp_handle != 0) && wiced_bt_spp_can_send_more_data() && (app_send_offset !=
APP_TOTAL_DATA_TO_SEND))
    {
        int bytes_to_send = app_send_offset + SPP_MAX_PAYLOAD < APP_TOTAL_DATA_TO_SEND ? SPP_MAX_PAYLOAD :
APP_TOTAL_DATA_TO_SEND - app_send_offset;
        for (i = 0; i < bytes_to_send; i++)
        {
            app_send_buffer[i] = app_send_offset + i;
        }
        wiced_bt_spp_send_session_data(spp_handle, app_send_buffer, bytes_to_send);
        app_send_offset += bytes_to_send;
    }
    // Check if we were able to send everything
    if (app_send_offset < APP_TOTAL_DATA_TO_SEND)
    {
        wiced_start_timer(&app_tx_timer, 100);
    }
    else
    {
        app_send_offset = 0;
    }
}

void app_interrupt_handler(void *data, uint8_t port_pin)
{
    int i;

    WICED_BT_TRACE("gpio_interrupt_handler pin:%d send_offset:%d\n", port_pin, app_send_offset);

     /* Get the status of interrupt on P# */
    if (wiced_hal_gpio_get_pin_interrupt_status(BUTTON_GPIO))
    {
        /* Clear the GPIO interrupt */
        wiced_hal_gpio_clear_pin_interrupt_status(BUTTON_GPIO);
    }
    // If we are already sending data, do nothing
    if (app_send_offset != 0)
        return;

    app_send_data();
}

void app_tx_ack_timeout(uint32_t param)
{
    app_send_data();
}
#endif
```

7.4.5 Low-Power Implementation and Eliminating Leakage Current

- To configure sleep mode, a call to `wiced_sleep_config` should be made after receiving the Bluetooth stack event `BTM_ENABLED_EVT` as shown in Code 35.

Code 35. *spp.c*: Configure Sleep Mode

```
    case BTM_ENABLED_EVT:
        application_init();
        wiced_sleep_config(TRUE, null, null);
        WICED_BT_TRACE("Free mem:%d", wiced_memory_get_free_bytes());
        break;
```

- Disable either or both seconds timer and fine timer based on application requirements in `wiced_bt_app_start_timer`. This will ensure that the module is not waking up from sleep at regular intervals.
- Disable traces in *makefile.mk* if the application does not require UART prints over a COM port.

Code 36. *makefile.mk*: Disable Trace

```
#C_FLAGS += -DWICED_BT_TRACE_ENABLE
```

- It is recommended to do one of the following if the application does not require UART communication:
  - Comment out `wiced_transport_init( &transport_cfg );` or
  - Configure switch elements of HCI_UART (SW4) and PUART(SW5) to OFF position. Modify the state of HCI_UART(SW4) switches to ON position when programming CYBT-343026 EVAL.

7.5 Part 3: Program the Module

This section details the steps required to program the EZ-BT WICED Evaluation Board. The CYBT-343026-01 module on the CYBT-343026-EVAL evaluation board includes a UART-based bootloader in the onboard chipset ROM, and therefore does not require an external programmer of any kind.

Note: The source project for this design is available on this application note's webpage.

7.5.1 Host UART Interface Selection and Preparation

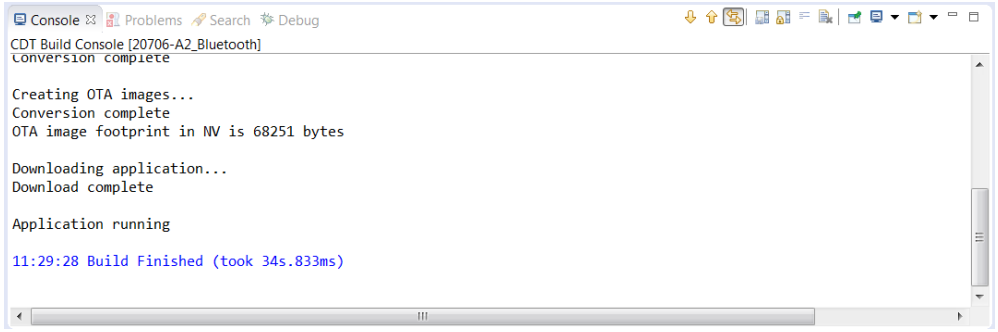
Host access to the HCI UART interface required for programming is available using either the built-in USB-to-UART Bridge or the 6-pin J1 header on the evaluation board. The HCI UART switch (four-position DIP switch) controls whether the USB-to-UART Bridge is connected to the HCI UART or pins are isolated from making connections with the USB-UART IC on the board. The PUART switch (four-position DIP switch) controls whether the USB-to-UART Bridge is connected to Peripheral UART or pins are isolated from making connections with USB-UART IC on the board.

7.5.2 Compiling and Downloading into the Module

Now that you have completed the code updates and prepared the programming interface, all that remains is to compile and flash the firmware image into the target device. To do this, simply double-click on the make target that you defined previously (`snip.bt.spp_custom-CYBT_343026_EVAL download`).

After a brief compile process, the console output in WICED Studio should indicate success, as shown in Figure 18.

Figure 18. Programming the Firmware



If you do not see this success message, ensure that you have correctly connected and configured the programming interface from the host as described in the previous sections, and try downloading again. If it continues to fail, refer to the following recovery steps.

7.5.3 Performing a Recovery Procedure and (Re-)Programming Your Module

In some cases, the normal firmware download procedure does not succeed even though all connections and switches are correct. This may happen as a result of SFLASH corruption due to incorrect application design or power loss during a normal firmware download process. If this happens, you may need to recover the device. Do the following to reset CYBT-343026-EVAL board to the factory default state.

Note: This will erase any user application in the memory and reset the board to the default state.

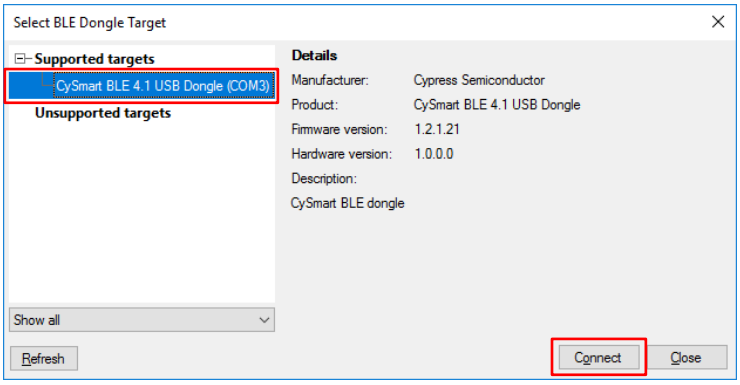
1. Ensure that HCI UART is connected to the module (via direct connection) or evaluation board (via configuration switches). This step is required only if you want to reprogram your module after the following recovery procedure has completed.
2. Press and hold the RECOVER button (RECOVER switch is defined in each EZ-BT evaluation board).
3. Press and hold the RESET button (RESET switch is defined in each EZ-BT evaluation board) for 1 second.
4. Release the RESET button.
5. Release the RECOVER button.
6. Re-program the board using the WICED SDK and your make target.

7.6 Part 4: Test Your Design

This section describes how to test your BLE design using the CySmart mobile apps and PC tool. The setup for testing your design using the EZ-BT WICED evaluation board is shown in Figure 19.

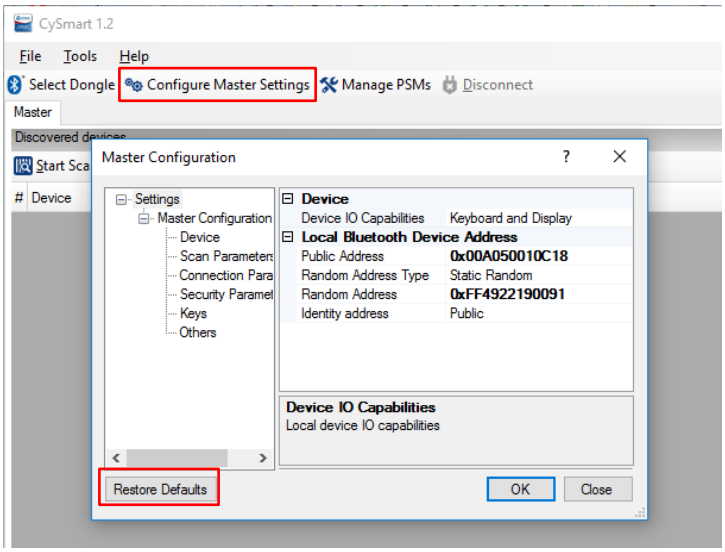
1. Turn ON Bluetooth on your Windows PC.
2. Launch the CySmart application.
3. Connect the BLE dongle to your Windows machine. Wait for the driver installation to complete.
4. Launch the CySmart Host Emulation Tool; it automatically detects the BLE Dongle. Click **Refresh** if the BLE Dongle does not appear in the **Select BLE Dongle Target** pop-up window. Click **Connect,** as shown in Figure 19.

Figure 19. CySmart BLE Dongle Selection



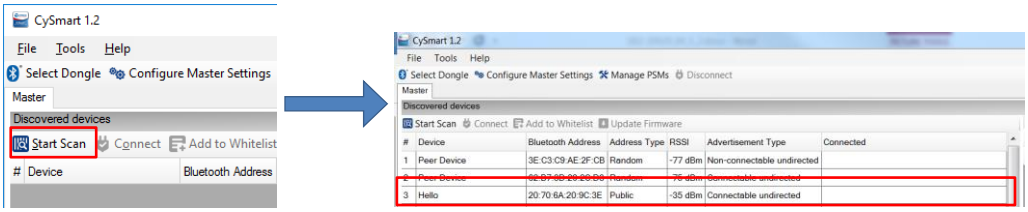
5. Select **Configure Master Settings** and restore the values to the default settings, as shown in Figure 20.

Figure 20. CySmart Master Settings Configuration



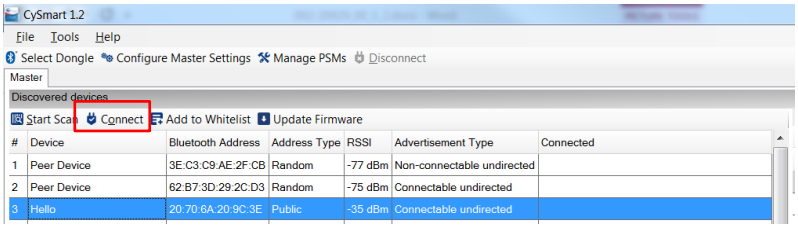
6. Press the reset switch on the EZ-BT WICED Evaluation board to start BLE advertisements from your design.
7. On the CySmart Host Emulation Tool, click **Start Scan**. Your device name should appear in the Discovered devices list, as shown in Figure 21.

Figure 21. CySmart Device Discovery



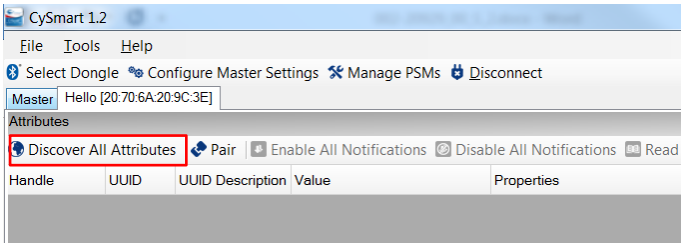
8. Select your device and click **Connect** to establish a BLE connection between the CySmart Host Emulation Tool and your device, as shown in Figure 22.

Figure 22. CySmart Device Connection



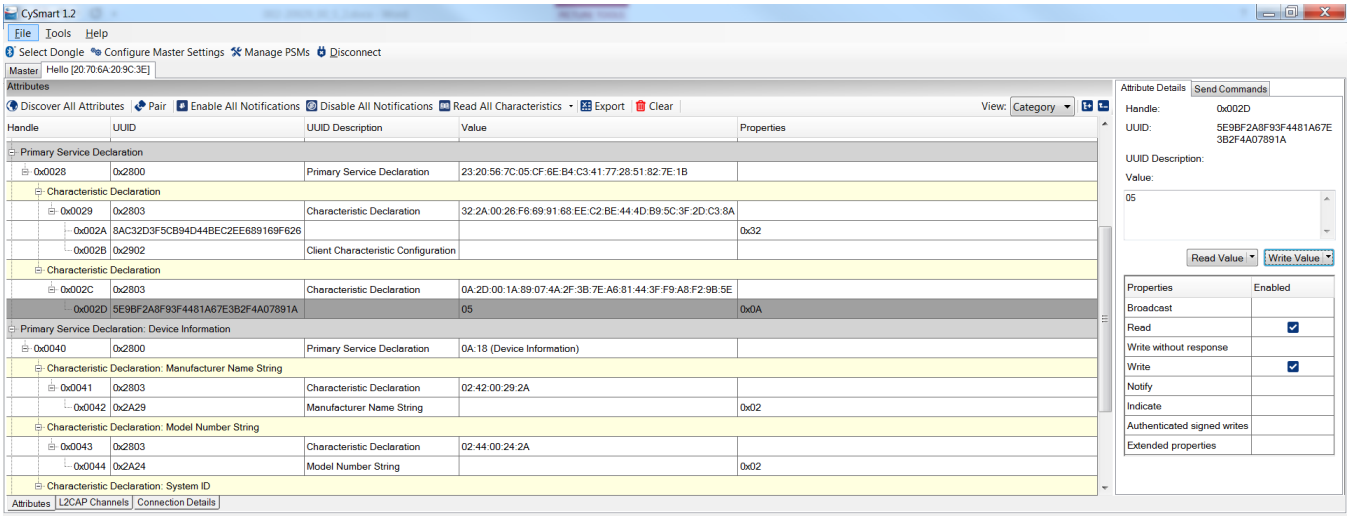
9. Once connected, click **Discover All Attributes** on your design from the CySmart Host Emulation Tool, as shown in Figure 23.

Figure 23. CySmart Attribute Discovery



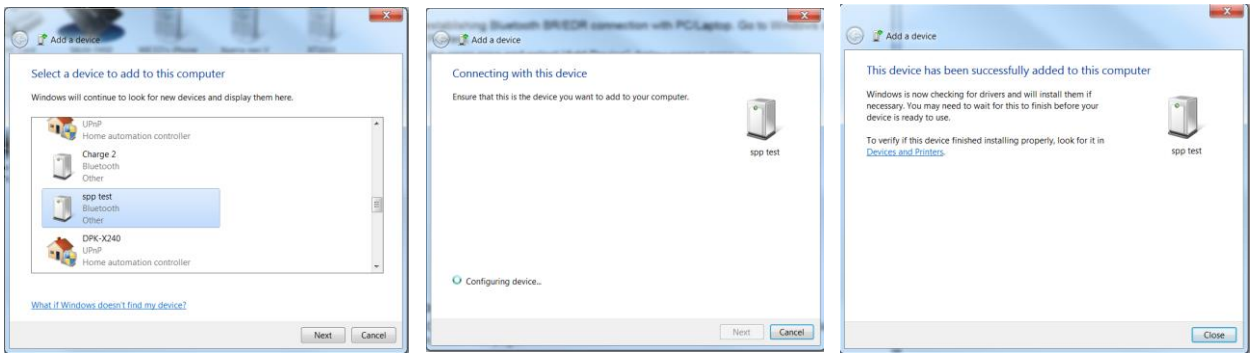
10. Write a value between 1 to 255 to the Characteristic handle 0x2D which is part of the custom service, as shown in Figure 24. Observe that the number of LED blinks on CYBT-343026 EVAL is equal to the byte value sent over this characteristic.

Figure 24. Testing with CySmart Host Emulation Tool



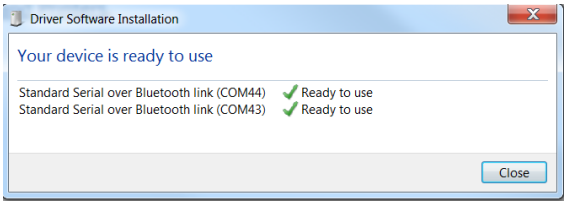
11. Now, let's establish a Bluetooth BR/EDR connection with the PC. Go to Windows **Control Panel** and open **Devices and Printers**.
12. Right-click in white space and select **Add Device**. Locate your device in the list of scanned devices. Select **spp_test** for this example and click **Next as** shown in Figure 25. Observe that Bluetooth tries to configure the device and successfully add the device to the PC.

Figure 25. Add Device Series of Events



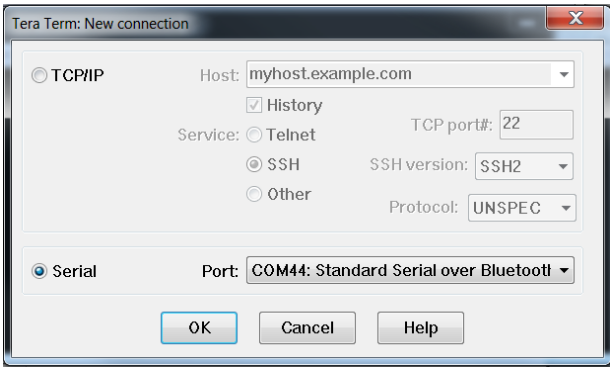
13. Driver software installation starts automatically; observe that COM port installation is in progress and gets successfully completed as shown in Figure 26.

Figure 26. COM Port Enumeration Successful



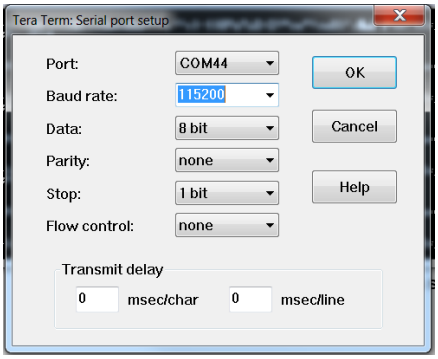
14. Choose Tera Term or equivalent and select COM 44 or equivalent COM port as per your device enumeration in the step above.

Figure 27. COM Port Selection



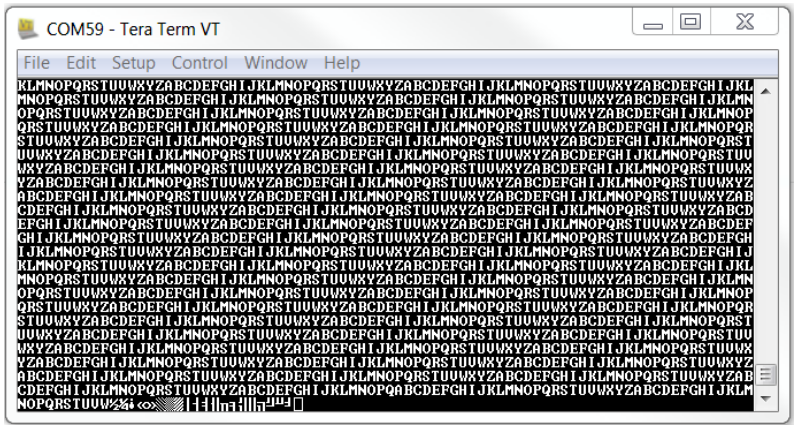
15. Set baud rate to 115200 as shown in Figure 28.

Figure 28. Set Baud Rate



16. Press the user button on CYBT-343026 EVAL and observe that a continuous stream of 1 MB of data is sent to the COM port over the SPP profile of Bluetooth BR/EDR link as shown in Figure 29.

Figure 29. ASCII Data Over Bluetooth BR/EDR Link



17. Both Bluetooth BR/EDR and BLE connections can be maintained simultaneously. BLE write requests can be sent while data transfer is in progress for Bluetooth BR/EDR. During Step 16, try to perform Step 10 and observe simultaneous Bluetooth BR/EDR data transfer and LED blink operation.

7.7 UART Debug Trace

Debug traces in the application project can be transmitted over HCI UART or PUART. The following configuration details help to enable or disable and route the trace data to the UART port of your choice.

- Every sample application has a local *makefile.mk* file. The following declaration enables traces for the sample application.

Code 37. *makefile.mk*: Enable Trace

```
C_FLAGS += -DWICED_BT_TRACE_ENABLE
```

- Next step is to populate the `wiced_transport_cfg_t` data structure, whose definition can be found in the *wiced_transport.h* file, which is part of the WICED SDK installation.

Code 38. *wiced_transport.h*: `wiced_transport_cfg_t` Structure

```
typedef PACKED struct
{
    wiced_transport_type_t              type;                   /**< Wiced transport type. */
    wiced_transport_interface_cfg_t     cfg;          /**< Wiced transport interface config. */
    wiced_transport_rx_buff_pool_cfg_t  rx_buff_pool_cfg; /**< Wiced rx buffer pool config. */
    wiced_transport_status_handler_t    p_status_handler; /**< Wiced transport status
                                                               handler.*/
    wiced_tranport_data_handler_t       p_data_handler;  /**< Wiced transport receive data
                                                               handler. */
    wiced_transport_tx_complete_t       p_tx_complete_cback;  /**< Wiced transport tx complete
                                                               callback. */
}wiced_transport_cfg_t;
```

An example of the transport configuration is shown below; here, UART is chosen as the transport type with a baud rate of 115200.

Code 39. *spp.c*: `wiced_transport_cfg_t` Structure Initialization

```
const wiced_transport_cfg_t transport_cfg =
{
    WICED_TRANSPORT_UART,
    { WICED_TRANSPORT_UART_HCI_MODE, 115200 },
    { TRANS_UART_BUFFER_SIZE, 1},
    NULL,
    NULL,
    NULL
};
```

- Initialize the transport and select the UART communication pins to be used as shown in the following code. In this example, debug traces are configured to be sent over PUART at a baud rate of 115200 bps.

Code 40. *spp.c*: Configuring UART and Baud Rate

```
wiced_transport_init(&transport_cfg);

    // Set to PUART to see traces on peripheral PUART
    wiced_set_debug_uart( WICED_ROUTE_DEBUG_TO_PUART );
    wiced_hal_puart_select_uart_pads(WICED_P33, WICED_P31, 0, 0);


    // Set to HCI to see traces on HCI uart - default if no call to wiced_set_debug_uart()
   // wiced_set_debug_uart( WICED_ROUTE_DEBUG_TO_HCI_UART );
```

7.8 Design Source

The functional WICED Studio SDK project for the BLE example design described in this application note is distributed on this application note's web page.

8 Module Placement and Enclosure Considerations

EZ-BT WICED Modules are designed to be soldered to a host PCB to provide seamless Bluetooth connectivity. To maximize the RF performance of the final product, care needs to be taken on the placement of the module and antenna. This section describes in detail the recommended placement of the module on a host board to ensure optimal RF performance. This section also details the effect of metallic or nonmetallic enclosure and metal obstructions near the module.

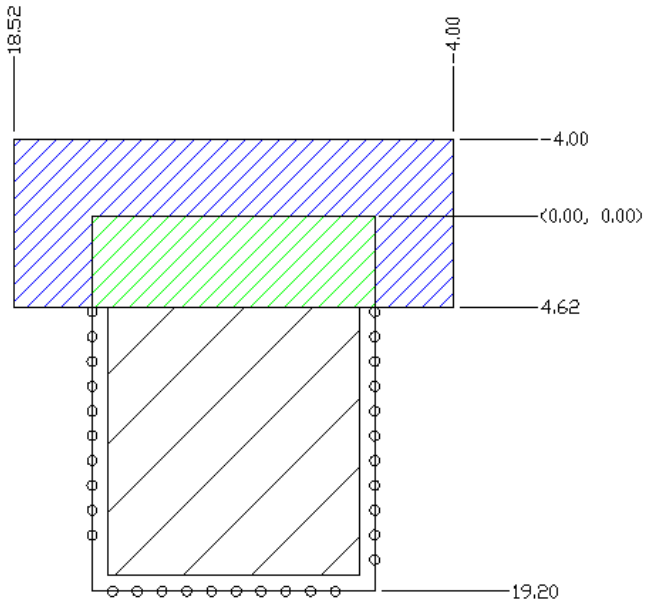
8.1 Antenna Ground Clearance

A monopole antenna requires that no ground plane is present below the antenna. The ground plane below it will not allow the field to propagate. This is defined as the Ground Clearance requirement. However, after some distance, a ground must be present for a monopole antenna. Defining this region is a very significant step for any antenna design. The Ground Clearance region defines the bandwidth and efficiency of the antenna.

Each specific EZ-BT WICED Module marketing part number specifies the Ground Clearance used for the design of the module, and offers recommended additional ground keep-out area to maximize the RF performance. The following examples reference the CYBT-343026-00 module specifically. For details on other modules, see the specific module datasheet.

The following example references a PCB trace antenna implementation (shown in the green hatched area), but the same rules and properties apply for chip antennas used on other Cypress EZ-BT Modules. The specific PCB trace antenna shown in Figure 30 requires a Ground Clearance area of 4.62 mm × 14.52 mm. To maximize the RF performance, an additional 4 mm of ground clearance is recommended. This is denoted in the blue hatched area. This additional ground clearance is not required but may improve the RF performance if implemented.

Figure 30. Antenna Clearance



In Figure 30, the PCB trace antenna is placed at the edge of the module. The green area in Figure 30 does not have any ground on any layer. The module placement in a host board needs to ensure that no traces or ground layers of the host board comes within this region. Any ground plane below a monopole antenna degrades the radiation and adversely affects the RF efficiency.

8.2 Module Placement in a Host System

The EZ-BT Module is soldered to a host board and a clearance must be provided for the antenna where no routing or ground is allowed on any layer. Placing the module at the edge of the host board is recommended because it provides the best RF performance and simplifies the requirement of not routing signal or ground traces under the antenna Ground Clearance region. Figure 31 shows four placement options on a host board, with option 1 being the most efficient.

Figure 31. Module Placement in a Host Board

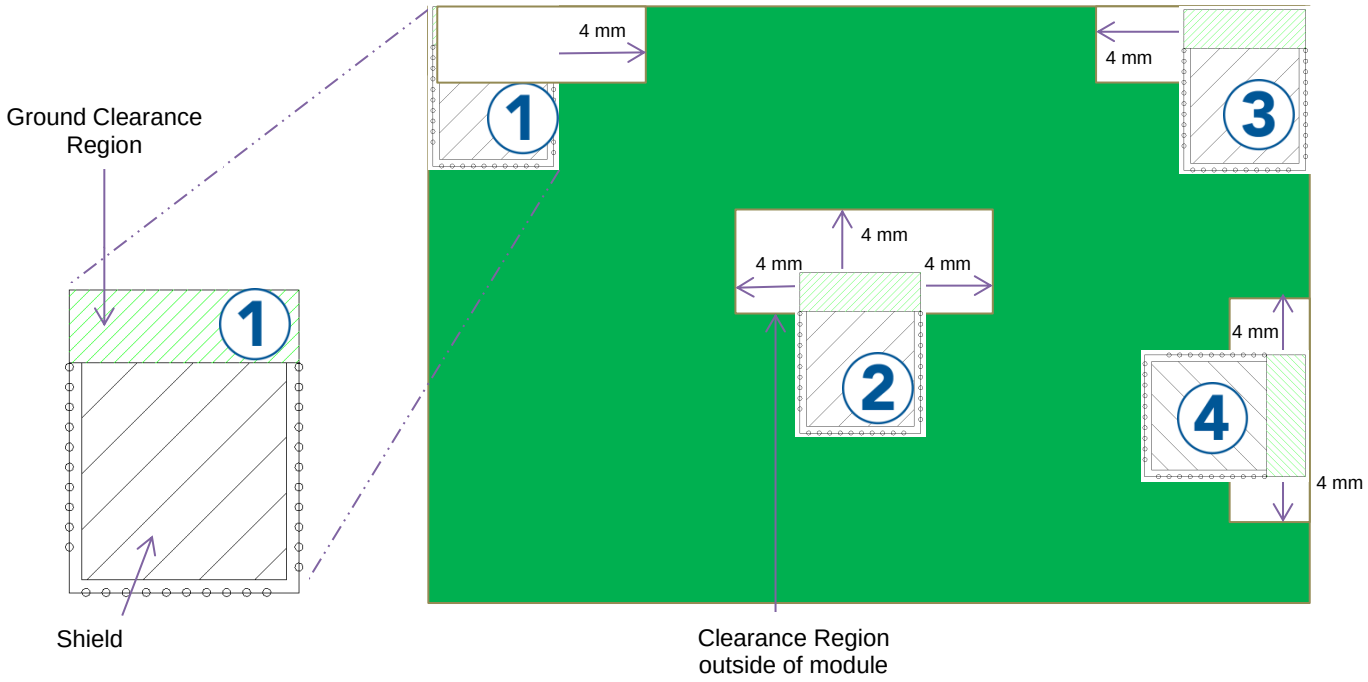


Figure 31 shows an example of four positions of the module in a host board, ‘1’, ‘2’, ‘3’, and ‘4’. The white area shown around the module is the additional clearance area. For the antenna in question, it is recommended to provide a clearance area of 4 mm in each direction. For details on the recommended clearance area for your EZ-BT Module, see the specific module datasheet.

As can be seen in Figure 31, when placing the module at the edge (placement options ‘1’ or ‘3’) of the host board, the additional clearance area is only required facing inwards towards the center of the main board. In all cases, there must be no possibility of signal or ground traces to be beneath the antenna Ground Clearance region. Conversely, if the module is placed in the middle (placement option ‘2’) of the host board, the clearance area must be provided in order to achieve an optimal RF performance.

Placement option ‘1’ or ‘3’ are the best options shown in Figure 31, because it removes the need to reroute signal or ground traces away from the Ground Clearance region of the module (because no GPIO are located at the top left or right corner of the module). Furthermore, it minimizes additional clearance area if optimal RF performance is required, because the antenna faces outward with the antenna exposed to open space.

In placement option ‘4’, although the module is placed at the edge of the host board, the antenna is not exposed to the maximum amount of free space.

Placement option ‘2’ not only wastes PCB real estate, but also provides diminished RF performance compared to position ‘1’ and ‘3’.

8.3 Enclosure Effects on Antenna Performance

Antennas used in consumer products are sensitive to the PCB RF ground size, the product's plastic casing, and metallic enclosures. This section describes the effect of each of these environmental factors on RF performance.

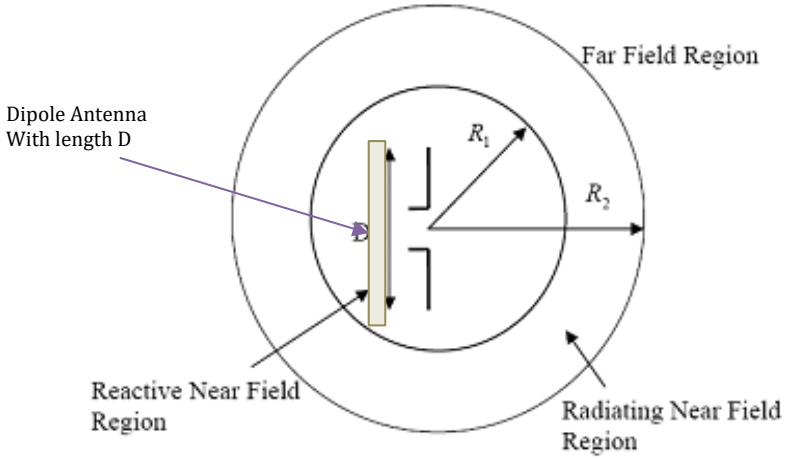
8.3.1 Antenna Near-Field and Far-Field

Every antenna contains two regions surrounding it: the near-field and the far-field.

The near-field is the region where the radiated field has not yet formed. In this region, the electric and magnetic fields are not orthogonal to each other. This region is very close to the antenna. The near-field region has two regions: the *reactive near-field region* and the *radiating near-field region*. The transition to a far-field region happens in the radiating near-field region.

The radiation field is formed after the transition to the far-field region. In this region, the relative angular variation of the field does not depend on the distance. This means that if you plot the angular radiation field at a distance from the antenna in the far-field region, their shapes remain the same. Only with distance, the field strength decreases. However, the shape of the radiation pattern remains the same with respect to the angular variation. This region is called the far-field region. An object in the far field does not affect the radiation pattern much. However, any obstruction in the near-field can completely change the radiation pattern. If the obstruction is metal, the effect on the radiation pattern is much more pronounced. Figure 32 shows the regions for a dipole antenna.

Figure 32. Near and Far Field



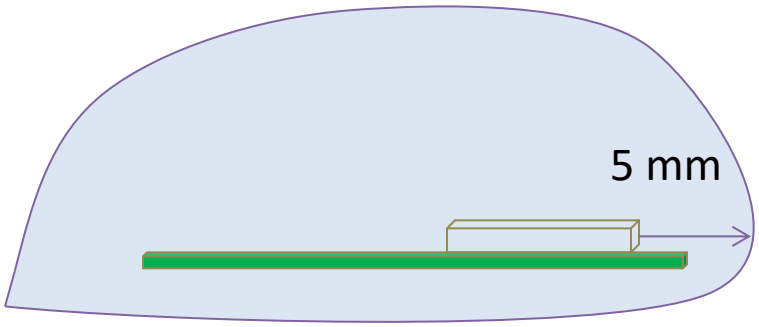
For a module based on a 2.4-GHz chip antenna, the near-field extends up to 4 mm.

8.3.2 Effect of Nonmetallic Enclosure

Any plastic enclosure changes the resonating frequency of the antenna. The antenna can be modeled as an LC resonator whose resonant frequency decreases when either L (inductance) or C (capacitance) increases. A larger RF ground plane and plastic casing increase the effective capacitance and thus reduce the resonant frequency. See the application note AN91445 for more details on the effect of an enclosure.

Figure 33 details a module antenna in a plastic enclosure. The clearance from the antenna to the plastic enclosure can be as little as 2 mm. However, clearance of this amount can affect the tuning of the antenna. This can be resolved by retuning the antenna; however, for a module solution, it is not recommended to attempt retuning of the antenna. To minimize effects on the module antenna, it is recommended to have a minimum clearance of 5 mm.

Figure 33. EZ-BT WICED Module Inside of a Plastic Mouse Enclosure

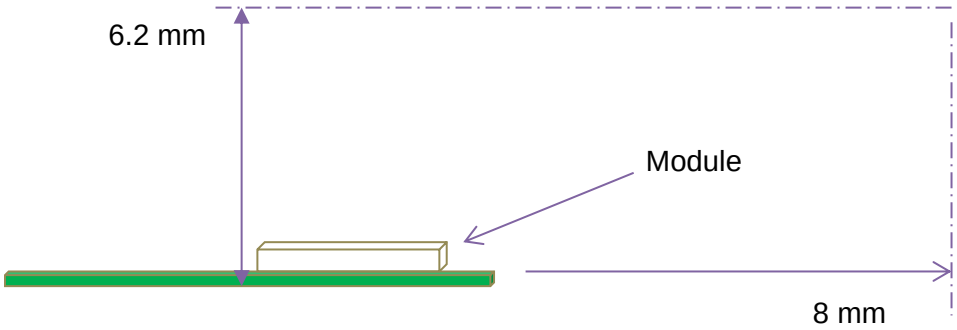


8.3.3 Effect of Metallic Objects

An antenna is sensitive to the presence of metallic objects in its vicinity. A metallic object shorts the electric field and thus changes the radiation field. Depending on the size of the obstruction, electromagnetic waves go through different diffraction patterns or may be completely shielded by the metallic object.

Metallic objects in the near-field can have a drastic impact on the radiation pattern. The thickness of the CYBT-343026-00 module is 2.25 mm (including the shield) and the near field of this module extends up to 4 mm from the antenna. Therefore, it is recommended that any metallic obstruction be at least 6.2 mm away from the PCB plane to avoid negative effects to the RF performance. Cypress recommends an 8-mm gap from the module PCB plane to any metallic enclosure. Figure 34 details the required clearance from the EZ-BT WICED Module to small metal obstructions.

Figure 34. Clearance from Small Metal Obstructions



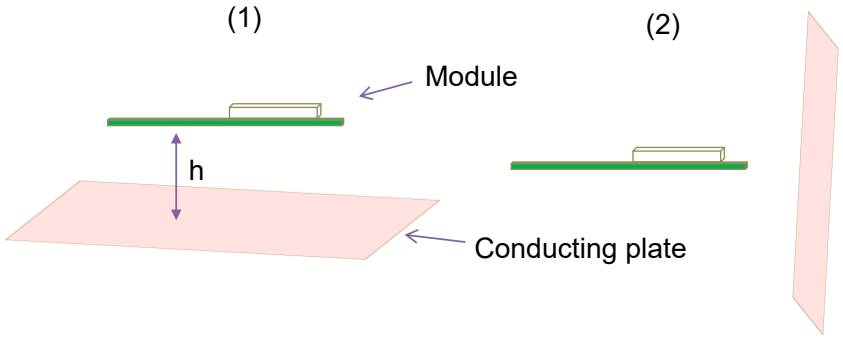
8.3.4 Recommendations for Placement over a Large Metal Plane

The other effect of metal is the formation of an image antenna. The best practice in this case is to orient the metal orthogonal to the antenna to ensure minimum effects. If the length or width of the plane approaches the size of the module, it is considered a large metal object near the antenna. Figure 35 details two placement options for this scenario. Of these two placement options, option “1” should be avoided.

It is recommended to not have any large metallic objects parallel to the antenna. This has a drastic effect because the image antenna is of opposite polarity. The interference caused by such an antenna is destructive to RF radiation.

If it is not possible to avoid a large metallic object running parallel to the module plane, you should maintain a distance (h) of at least 30 mm. This will ensure that the interference caused by the image antenna will not be completely destructive. The radiation will be strongly directional below the 30-mm distance; the efficiency will dramatically drop at a distance (h) below 8 mm. At a distance (h) of around 2 mm, the radiation efficiency can go below 20%.

Figure 35. Clearance from a Large Metal Plane



8.4 Guidelines for Enclosures and Ground Plane

Use the following best practices with respect to enclosure design and ground planes:

- Ensure that there is no component, mounting screw, or ground plane near the tip or the length of the antenna located on the EZ-BT WICED Module.
- Ensure that no battery cable, microphone cable, or trace crosses the antenna trace on the PCB.
- Ensure that the antenna is not completely covered by a metallic enclosure. If the product has a metallic casing or shield, the casing should not cover the antenna. No metal is allowed in the antenna near the field.
- Ensure that paint on plastic enclosures is nonmetallic near the antenna.
- Ensure that the orientation of the antenna is in-line with the final product orientation (if possible) so that radiation is maximized in the required direction. The polarization and position of the receive antenna should be taken into account so that the module can be oriented to maximize the radiation.
- Ensure that there is no ground directly below the antenna Ground Clearance region of the module.

9 Manufacturing with EZ-BT WICED Modules

EZ-BT WICED Modules are intended to be used with traditional Surface Mount Technology (SMT) manufacturing lines and are compatible with industry-standard reflow profiles for Pb-free solders.

9.1 SMT Manufacturing Pick-and-Place

The modules should be picked up from the topside of the module using industry-standard pick-and-place machinery and nozzles. The ideal location for picking up the module is on the shield area of the module. For the optimal location for your EZ-BT WICED Module, see the module's datasheet.

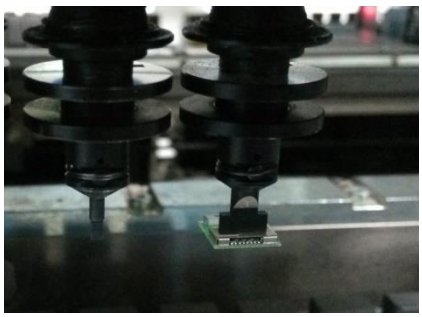
Each module MPN has a unique center-of-mass detailed in each product's datasheet. This center-of-mass is the area that represents the optimal location to pick up the unit with the nozzle. Using the center-of-mass guidelines for the pick-and-place location minimizes SMT line disturbances caused by units releasing prematurely from the nozzle.

Figure 36 shows an image of a nozzle used by Cypress for manufacturing the CYBT-343026-EVAL Evaluation Board product. See the center-of-mass dimensions in each module's datasheet to select an appropriate nozzle for your manufacturing line equipment.

Figure 36. Nozzle Used by Cypress for Evaluation Board Production

Figure 37 shows an image of a Cypress EZ-BT Module being picked up at the center-of-mass by the nozzle referenced above.

Figure 37. Image of Nozzle Used by Cypress for Evaluation Board Production



9.2 Manufacturing Solder Reflow

EZ-BT WICED Modules are compatible with industry-standard reflow profiles for Pb-free solder. Table 6 details the solder reflow specifications for all modules.

Table 6. EZ-BT WICED Module Solder Reflow Specification

Module Package	Maximum Peak Temperature	Time at Maximum Temperature
All Packages	260 °C	30 seconds

10 Summary

This application note explores the EZ-BT WICED Module solutions, architectures, development tools, host board placement and orientation, and production manufacturing. EZ-BT WICED Modules are fully integrated BLE solutions that allow rapid development and production release for customer applications. The core of the EZ-BT WICED Modules is the Cypress WICED Bluetooth Smart Ready silicon devices, integrating the Bluetooth radio, digital peripheral functions, memory, and an Arm Cortex-M3 or M4 microcontroller. The Cypress EZ-BT Module family provides multiple module options to service the needs of any customer application.

11 Related Application Notes

- AN91445 – Antenna Design Guide
- AN96841 – Getting Started With EZ-BT Creator Modules

About the Authors

Name:	David Solda, Santhosh Vojjala
Title:	Senior Business Unit Director, Principle Applications Engineer
Background:	David Solda has a BS in Computer/Electrical Engineering, a BS in Mathematics, and an MBA from Santa Clara University, California.

Appendix A. Cypress Terms of Art

This section lists the most commonly used terms that you might hear while working with Cypress's WICED family of devices.

EZ-BT™ WICED Module (EZ-BT WICED) - EZ-BT WICED Modules are fully integrated, fully certified, Bluetooth Smart or Bluetooth Low Energy (BLE) module designed for ease-of-use and reducing time-to-market. It contains Cypress's WICED BLE chip, one crystal, PCB trace antenna, shield and passive components. EZ-BT WICED Module provides a simple and low-cost way to add a microcontroller and Bluetooth Smart connectivity to any system.

EZ-BT™ WICED Modules (EZ-BT WICED) - EZ-BT WICED Modules are fully integrated, fully certified, Bluetooth Smart Ready (Bluetooth Basic Rate, Enhanced Data Rate, and Bluetooth Low-Energy) modules designed for ease-of-use and reducing time-to-market. They contain Cypress' WICED dual-mode chip, one crystal, PCB trace antenna, shield and passive components. EZ-BT WICED Modules provide a simple and low-cost way to add a microcontroller and Bluetooth Smart Ready connectivity to any system.

WICED Studio SDK – Cypress' WICED (Wireless Connectivity for Embedded Devices) is a full-featured platform with proven Software Development Kits (SDKs) and turnkey hardware solutions from partners to readily enable Wi-Fi and Bluetooth connectivity in system design.

Appendix B. EZ-BT WICED Module Product Details

The information contained for each module part number includes the following:

- Physical image for each EZ-BT WICED Module marketing part number
- Pinout and functionality for each EZ-BT WICED Module marketing part number
- Recommended host PCB layout footprint for each EZ-BT WICED Module marketing part number
- Recommended additional clearance area for each EZ-BT WICED Module marketing part number

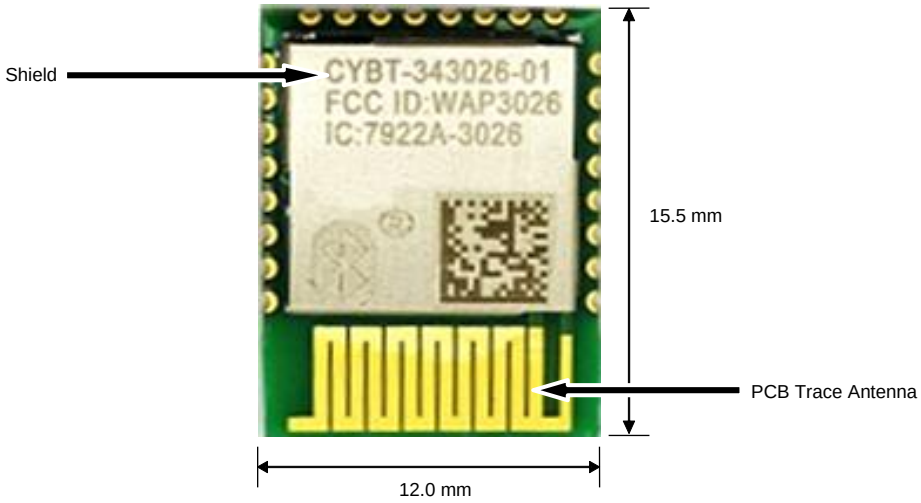
To jump to your specific EZ-BT WICED Module, click the marketing part number in the below list:

- CYBT-343026-01
- CYBT-353027-02
- CYBT-423028-02
- CYBT-413034-02
- CYBT-483039-02

B.1 CYBT-343026-01

CYBT-343026-01 is a Bluetooth 5.0-qualified dual-mode (BR/EDR + BLE) module containing 512 KB of on-module serial flash. Figure 38 shows a physical picture of the CYBT-343026-01 EZ-BT WICED Module.

Figure 38. CYBT-343026-01 Module Top View

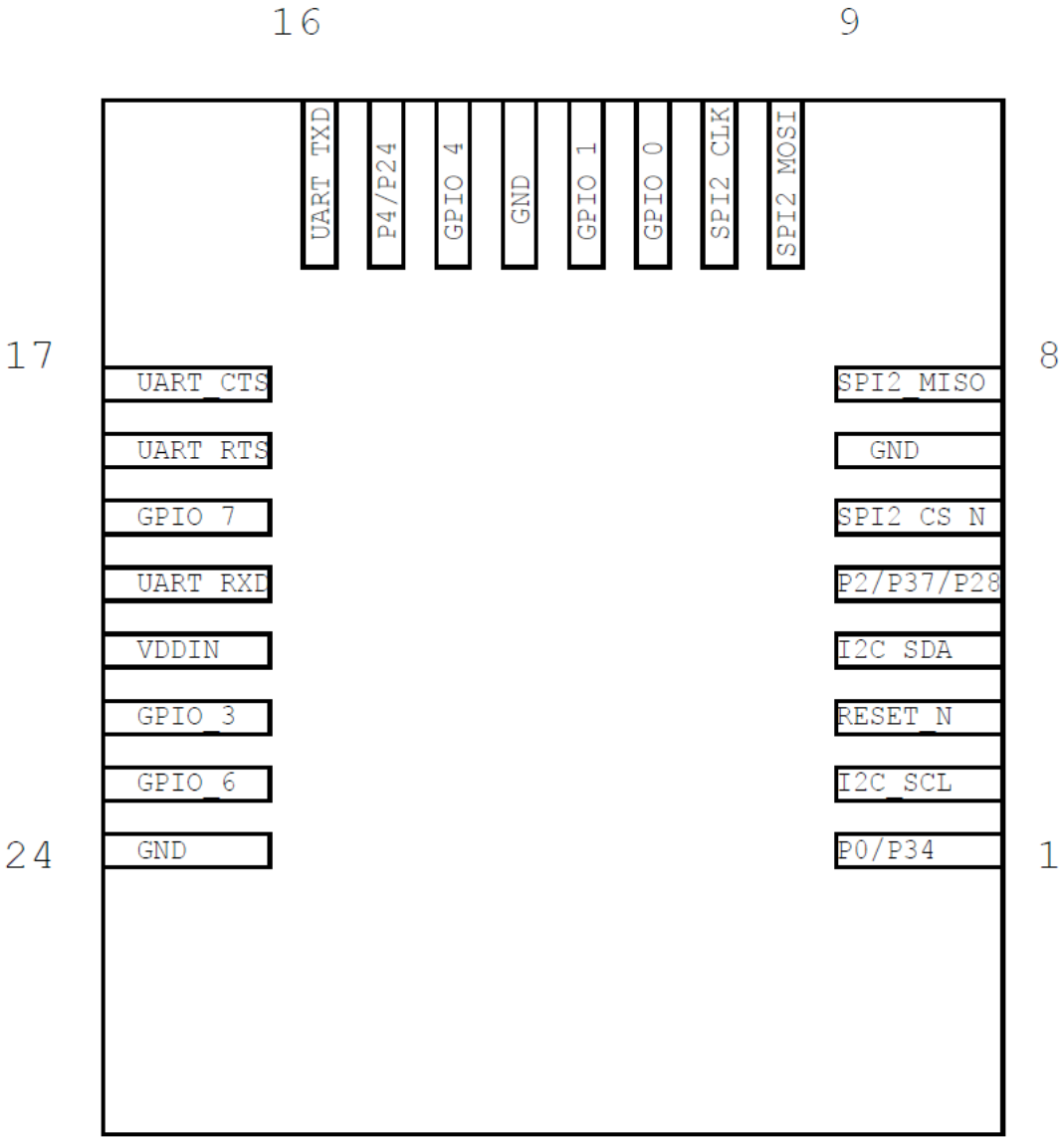


For more details on this module's dimensions, external component connections, and module placement recommendations, see the CYBT-343026-01 datasheet.

B.1.1 Pinout and Functionality

The CYBLE-343026-01 module is designed to mount as a component on an end-product PCB. Only a portion of the available connections of the CYW20706 WICED Bluetooth silicon device are exposed on the CYBT-343026-01 module in order to minimize the module footprint size. The CYBT-343026-01 module contains 24 connections on the bottom side of the module. Figure 39 details the bottom side connections available on the CYBT-343026-01 module.

Figure 39. CYBT-343026-01 Module Bottom View (Seen from Bottom)



A list of the available I/Os and supported functionality for each I/O of CYBT-343026-01 is shown in Table 7.

Table 7. CYBT-343026-01 Module Available Connections and Functionality

Pad	Pad Name	Silicon Port Pin Name	Functionality							
			UART[3]	SPI[4, 5]	I2C	ADC	Coex	CLK/ XTAL	GPIO	OTHER
1	P0/P34	PCM_Sync/I2S_ WS/P0/P34	PUART_TX/P0 PUART_RX/P34	SPI1_MOSI/P0 (master/slave)		IN29/P0 IN5/P34			Yes	PCM_Sync I2S_WS
2	I2C_SCL	I2S_DO/ PCM_Out/P3/P2 9/P35	PUART_CTS/ P3 or P35	SPI1_CLK/P3 (master/slave)	SCL SDA/P35	IN4/P35 IN10/P29			Yes (P3/P29/ P35)	I2S_DO PCM_Out PWM3 (P29)
3	XRES	RESET_N	External Reset (Active LOW)							
4	I2C_SDA	PCM_IN/ I2S_DI/P12			SDA	IN23/P12			Yes (P12)	PCN_IN I2S_DI
5	P2/P37/P28	PCM_CLK/I2S_ CLK/P2/P28/P37	PUART_RX/P2	SPI1_CS/P2 (slave) SPI1_MOSI/P2 (master) SPI1_MISO/P37 (slave)	SCL/P37	IN11/P28 IN2/P37		ACLK1 /P37	Yes	PWM2 (P28) I2S_CLK PCM_CLK
6	SPI2_CS_N	N/A	No Connect (Used for on-module memory SPI interface for CYBT-343026-01)							
7	GND	GND	Ground							
8	SPI2_MISO	N/A	No Connect (Used for on-module memory SPI interface for CYBT-343026-01)							
9	SPI2_MOSI	N/A	No Connect (Used for on-module memory SPI interface for CYBT-343026-01)							
10	SPI2_CLK	N/A	No Connect (Used for on-module memory SPI interface for CYBT-343026-01)							
11	GPIO_0	BT_GPIO_0/ P36/P38		SPI1_CLK/P36 SPI1_MOSI/P38 (master/slave)		IN3/P36 IN1/P38		ACLK0 /P36	Yes (DevWake)	
12	GPIO_1	BT_GPIO_1/ P25/P32	PUART_RX/P25 PUART_TX/P32	SPI1_MISO/P25 (master/slave) SPI1_CS/P32 (slave)		IN7/P32		ACLK0 /P32	Yes (HostWake)	
13	GND	GND	No Connect (Used for on-module memory SPI interface for CYBT-343026-01)							
14	GPIO_4	BT_GPIO_4/P6/ P31/LPO_IN	PUART_RTS/P6 PUART_TX/P31	SPI1_CS/P6 (slave)		IN8/P31			Yes	Ext LPO In
15	P4/P24	BT_CLK_REQ[6]/ P4/P24	PUART_RX/P4 PUART_TX/P24	SPI1_MOSI/P4 (master/slave) SPI1_CLK/P24 (master/slave)					Yes (CLK_REQ)	

[3] Peripheral UART operates as a default 2-wire UART interface (PUART_TX and PUART_RX). Flow control connections (RTS and CTS) are provided, but UART flow control is not supported in hardware. Flow control operation requires logic to be completed in the application code in order for flow control on PUART to be functional. HCI UART supports flow control in Hardware; no additional logic is required.

[4] CYBT-343026-01 contains a single SPI (SPI1) peripheral supporting both master or slave configurations. SPI2 is used for on-module serial memory interface.

[5] In Master mode, any available GPIO can be configured as SPI1_CS. This function is not explicitly shown in the table above.

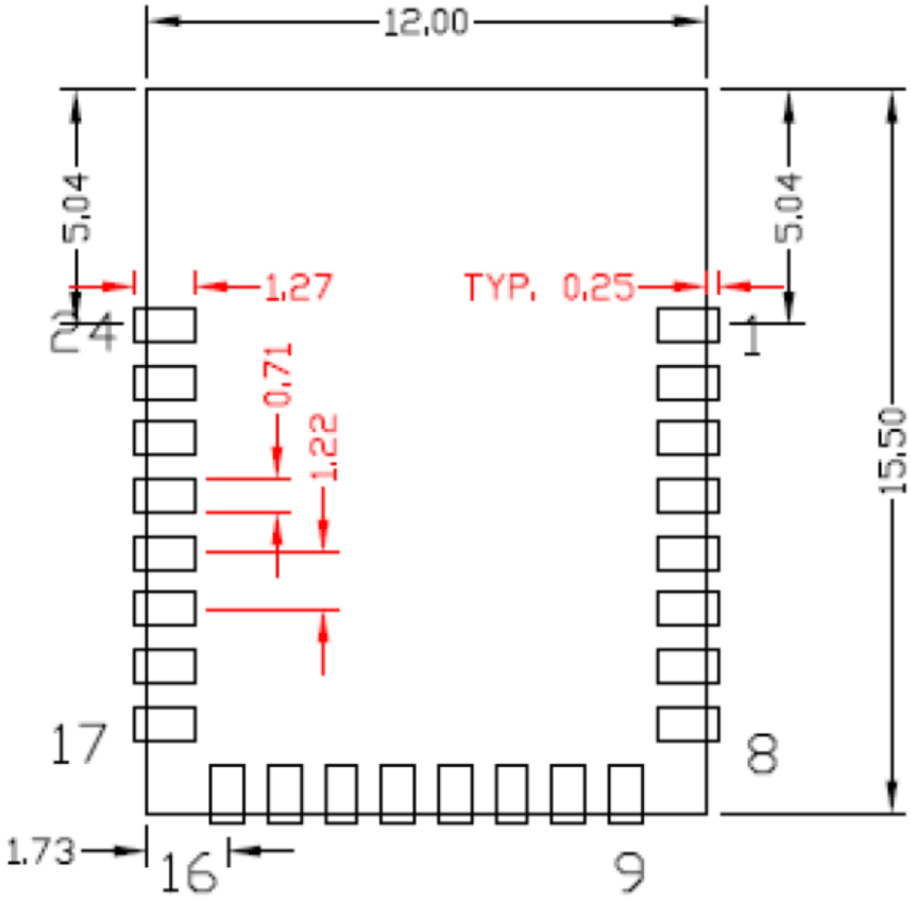
[6] Pad 15 of the CYBT-343026-01 module is configured to have BT_CLK_REQ set as the default. To use P4 or P24 as GPIO or other functions, this pad must be explicitly configured to operate as a GPIO (and not BT_CLK_REQ). To do this, enter the following declaration in the applications code to configure the port pin (recommended towards the end of the Bluetooth Application Initialization routine): `*((volatile uint32_t*)(0x003201b8)) = 0x7000;`

Pad	Pad Name	Silicon Port Pin Name	Functionality							
			UART[3]	SPI[4, 5]	I2C	ADC	Coex	CLK/ XTAL	GPIO	OTHER
16	UART_TXD	BT_UART_TXD	HCI UART Transmit Data							
17	UART_CTS	BT_UART_CTS	HCI UART Clear to Send Input							
18	UART_RTS	BT_UART_RTS	HCI UART Request to Send Output							
19	GPIO_7	BT_GPIO_7/ P30	PUART_RTS/P30			IN9/P30	Yes (GCI_SECI_OUT)		Yes	
20	UART_RXD	BT_UART_RXD	HCI UART Receive Data							
21	VDDIN	VDDIN	VDDIN (2.3 V ~ 3.6 V)							
22	GPIO_3	BT_GPIO_3/P27 /P33	PUART_RX/P33	SPI1_MOSI/P27 (master/slave) SPI1_MOSI/P33 (slave)		IN6/P33		ACLK1 /P33	Yes	PWM1 (P27)
23	GPIO_6	BT_GPIO_6/P11 /P26		SPI1_CS/P26 (slave)		IN24/P11	Yes (GCI_SECI_IN)		Yes	PWM0 (P26)
24	GND	GND	No Connect (Used for on-module memory SPI interface for CYBT-343026-01)							

B.1.2 Host Recommended PCB Layout

To assist in the host PCB layout design for CYBT-343026-01, Cypress provides three host PCB landing pattern reference drawings in Figure 40, Figure 41, and in Figure 42, and Table 8. Figure 40 provides a dimensioned view of the host PCB layout. Figure 41 provides the location to the center edge of each solder pad relative to the origin of the module (upper right PCB outline). Figure 42 and Table 8 provides the location to each solder pad center location for the host PCB layout. Dimensions shown are in mm unless otherwise stated.

Figure 40. Host Board Required PCB Layout Pattern (Dimensioned View)



Note: Pad length shown includes overhang of the host pad beyond the module pad outline. The minimum recommended pad length on the host PCB is 1.27 mm.

Figure 41. Host Board Required PCB Layout Pattern: To Pad Center Edge Relative to Origin

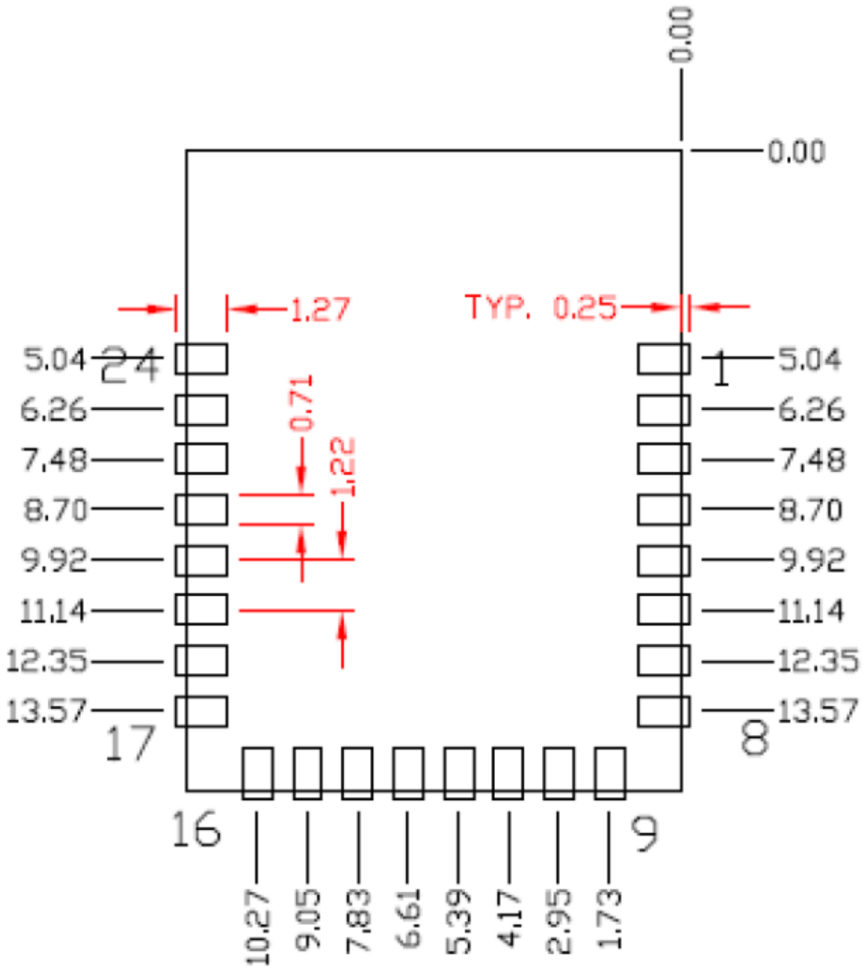


Figure 42. Host Board Required PCB Layout Pattern To Pad Center Relative to Origin

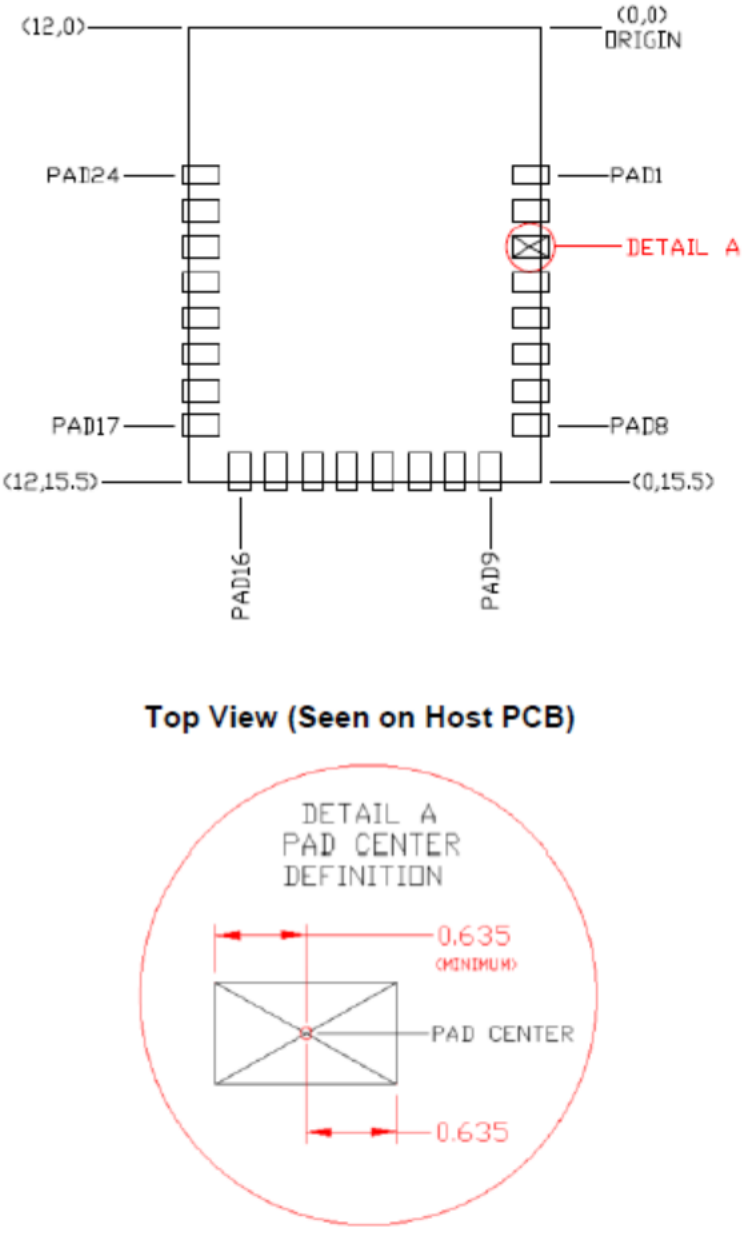
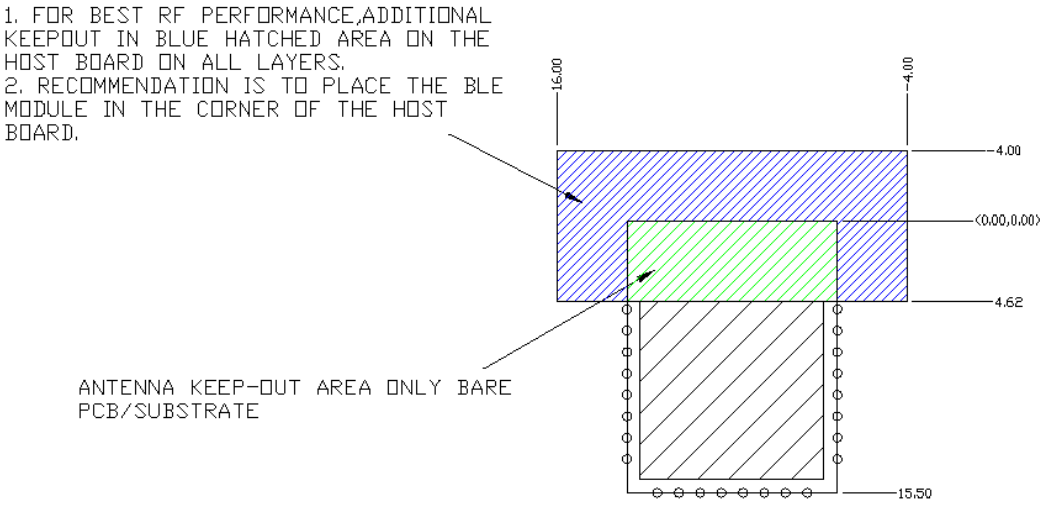


Table 8. Location to Pad Center from Origin (dimensions in mm and mils)

Solder Pad (Center of Pad)	Location (X,Y) from Origin (mm)	Location (X,Y) from Origin (mils)
1	(0.38, 5.04)	(14.96, 198.42)
2	(0.38, 6.26)	(14.96, 246.46)
3	(0.38, 7.48)	(14.96, 294.49)
4	(0.38, 8.70)	(14.96, 342.52)
5	(0.38, 9.92)	(14.96, 390.55)
6	(0.38, 11.14)	(14.96, 438.58)
7	(0.38, 12.35)	(14.96, 486.22)
8	(0.38, 13.57)	(14.96, 534.25)
9	(1.73, 15.11)	(68.11, 594.88)
10	(2.95, 15.11)	(116.14, 594.88)
11	(4.17, 15.11)	(164.17, 594.88)
12	(5.39, 15.11)	(212.20, 594.88)
13	(6.61, 15.11)	(260.24, 594.88)
14	(7.83, 15.11)	(308.27, 594.88)
15	(9.05, 15.11)	(356.30, 594.88)
16	(10.27, 15.11)	(404.33, 594.88)
17	(11.62, 13.57)	(457.48, 534.25)
18	(11.62, 12.35)	(457.48, 486.22)
19	(11.62, 11.14)	(457.48, 438.58)
20	(11.62, 9.92)	(457.48, 390.55)
21	(11.62, 8.70)	(457.48, 342.52)
22	(11.62, 7.48)	(457.48, 294.49)
23	(11.62, 6.26)	(457.48, 246.46)
24	(11.62, 5.04)	(457.48, 198.42)

Figure 43 below details additional host board keep out area to achieve optimal RF performance with the CYBT-343026-01 module (denoted in blue hatched area).

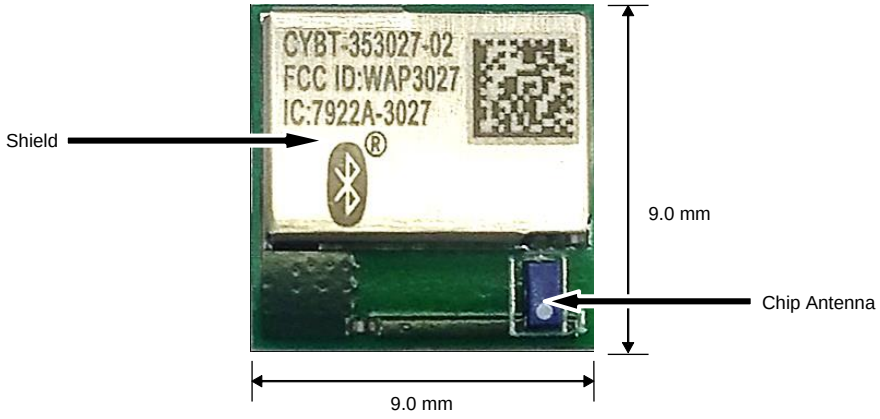
Figure 43. Host Board Additional Keep Out Area for Optimal RF Performance



B.2 CYBT-353027-02

CYBT-353027-02 is an ultra-small form factor, Bluetooth 5.0-qualified dual-mode (BR/EDR + BLE) module containing 512 KB of on-module serial flash. Figure 38 shows a physical picture of the CYBT-353027-02 EZ-BT WICED Module.

Figure 44. CYBT-353027-02 Module Top View



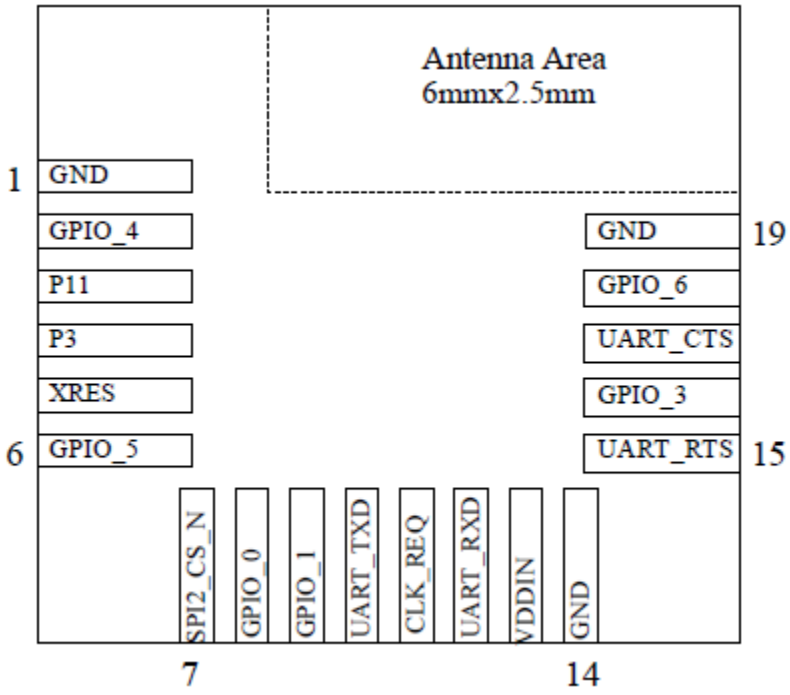
Because of its small form-factor, CYBT-353027-02 is ideal for space-constrained applications that require a BR/EDR or BLE wireless connection. CYBT-353027-02 supports several serial communication interfaces (I2C, SPI, 2-wire UART, I2S, PCM) and provides up to eight GPIOs. CYBT-353027-02 does not contain support for PWM resources.

For more details on this module's dimensions, external component connections, and module placement recommendations, see the CYBT-353027-02 datasheet.

B.2.1 Pinout and Functionality

The CYBLE-353027-02 module is designed to mount as a component on an end-product PCB. Only a portion of the available connections of the CYW20707 WICED Bluetooth silicon device are exposed on the CYBT-353027-02 module in order to minimize the module footprint size. The CYBT-353027-02 module contains 19 connections on the bottom side of the module. Figure 39 details the bottom side connections available on the CYBT-353027-02 module.

Figure 45. CYBT-353027-02 Module Bottom View (Seen from Bottom)



A list of the available I/Os and supported functionality for each I/O of CYBT-353027-02 is shown in Table 9.

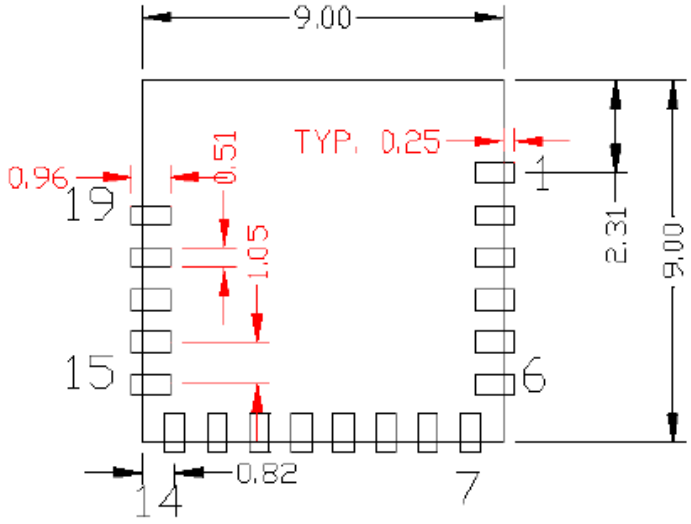
Table 9. CYBT-353027-02 Module Available Connections and Functionality

Pad	Pad Name	Silicon Port Pin Name	Functionality							
			UART	SPI	I2C	ADC	Coex	CLK/XTAL	GPIO	OTHER
1	GND	GND	Ground							
2	GPIO_4	GPIO_4/P1/ I2S_CLK/ PCM_CLK		SPI1_MISO/P1 (master)		IN28/P1			Yes (P1)	PCM_CLK I2S_CLK
3	P11	P11/I2S_WS/ PCM_SYNC				IN24/P11			Yes (P11)	PCM_Sync I2S_WS
4	P3	P3/I2S_DI/ PCM_IN		SPI1_CLK (master)	SDA				Yes (P3)	PCM_DI I2S_DI
5	XRES	RST_N	External Reset (Active LOW)							
6	GPIO_5	BT_GPIO_5/ P8/P33	PUART_RX/P33			IN27/P8 IN6/P33	GCI_SECI_OUT	ACK1/P33	Yes (P8/P33)	
7	SPI2_CS_N	SPI2_CSN		SPI2_CS_N						
8	GPIO_0	BT_GPIO_0							Yes (DevWake)	
9	GPIO_1	BT_GPIO_1							Yes (HostWake)	
10	UART_TXD	BT_UART_TXD	HCI UART Transmit Data							
11	CLK_REQ	BT_CLK_REQ	Used for shared-clock applications							
12	UART_RXD	BT_UART_RXD	HCI UART Receive Data							
13	VDDIN	VDDO	VDDIN (2.3 V ~ 3.6 V)							
14	GND	GND	Ground							
15	UART_RTS	BT_UART_RTS_N	HCI UART Request to Send Output							
16	GPIO_3	BT_GPIO_3/P0	PUART_TX/P0	SPI1_MOSI/P0 (master)		IN29/P0			Yes (P0)	
17	UART_CTS	BT_UART_CTS	HCI UART Clear to Send Input							
18	GPIO_6	BT_GPIO_6/P9/ I2S_DO/ PCM_OUT				IN26/P9			Yes (P9)	I2S_DO PCM_OUT
19	GND	GND	Ground							

B.2.2 Host Recommended PCB Layout

To assist in the host PCB layout design for CYBT-353027-02, Cypress provides three host PCB landing pattern reference drawings in Figure 40, Figure 41, and in Figure 42, and Table 8. Figure 40 provides a dimensioned view of the host PCB layout. Figure 41 provides the location to the center edge of each solder pad relative to the origin of the module (upper right PCB outline). Figure 42 and Table 8 provides the location to each solder pad center location for the host PCB layout. Dimensions shown are in mm unless otherwise stated.

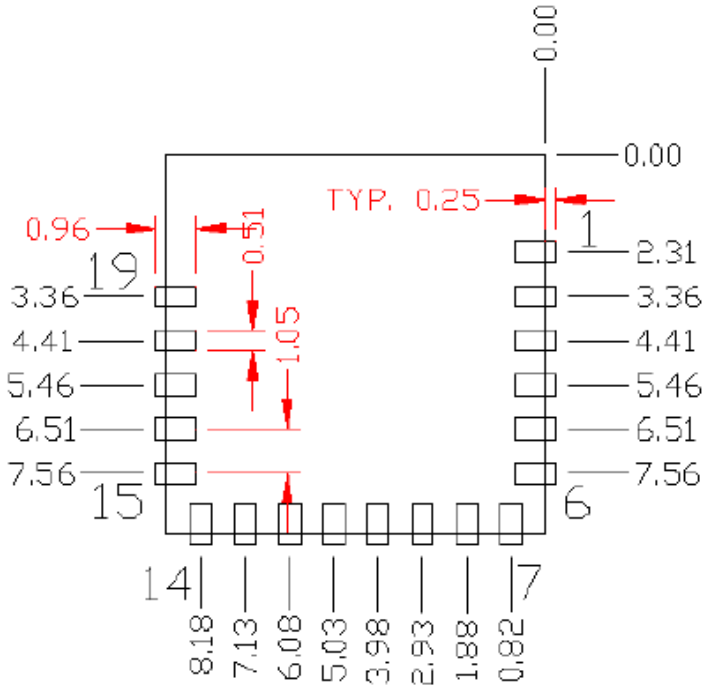
Figure 46. Host Board Required PCB Layout Pattern (Dimensioned View)



Top View (Seen on Host PCB)

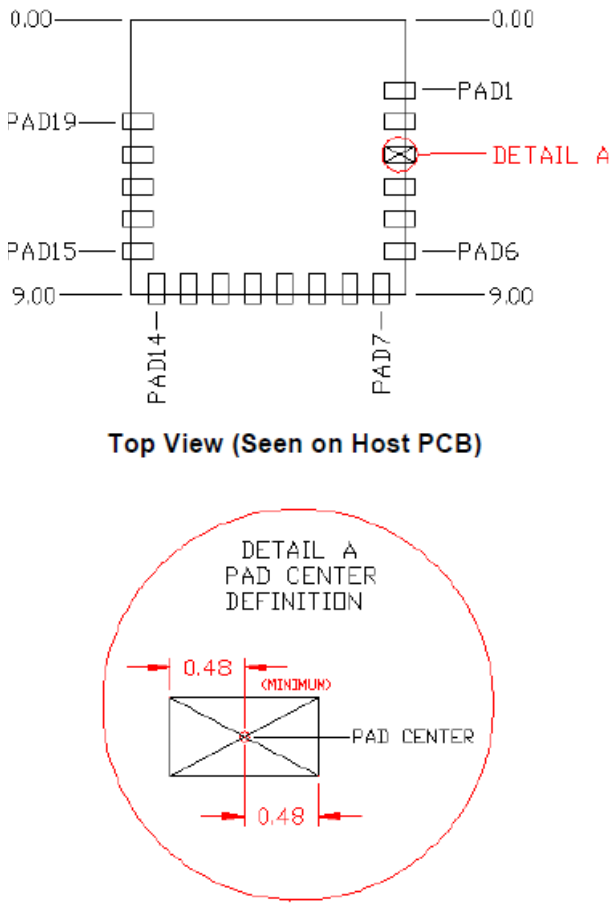
Note: Pad length shown includes overhang of the host pad beyond the module pad outline. The minimum recommended pad length on the host PCB is 0.96 mm.

Figure 47. Host Board Required PCB Layout Pattern: To Pad Center Edge Relative to Origin



Top View (Seen on Host PCB)

Figure 48. Host Board Required PCB Layout Pattern To Pad Center Relative to Origin



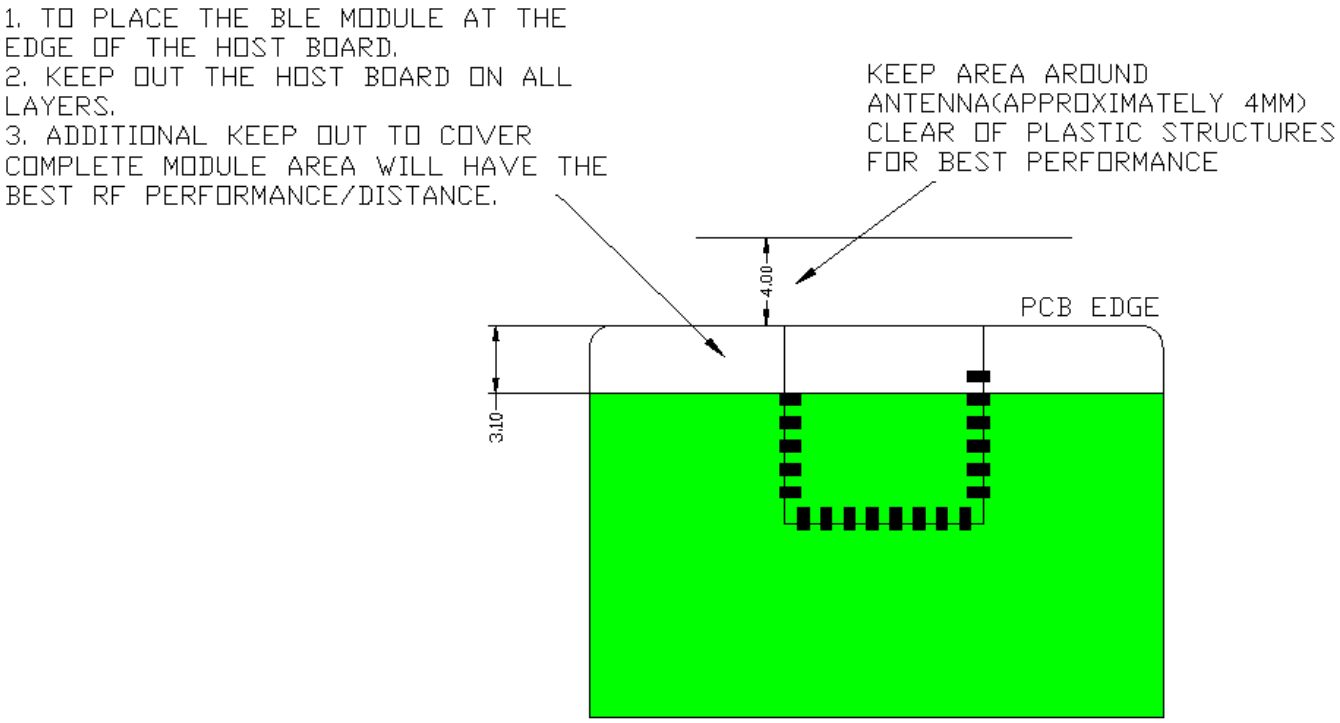
Top View (Seen on Host PCB)

Table 10. Location to Pad Center from Origin (dimensions in mm and mils)

Solder Pad (Center of Pad)	Location (X,Y) from Origin (mm)	Location (X,Y) from Origin (mils)
1	(0.23, 2.31)	(9.06, 90.94)
2	(0.23, 3.36)	(9.06, 132.28)
3	(0.23, 4.41)	(9.06, 173.62)
4	(0.23, 5.46)	(9.06, 214.96)
5	(0.23, 6.51)	(9.06, 256.30)
6	(0.23, 7.56)	(9.06, 297.64)
7	(0.82, 8.77)	(32.28, 345.27)
8	(1.88, 8.77)	(74.02, 345.27)
9	(2.93, 8.77)	(115.35, 345.27)
10	(3.98, 8.77)	(156.69, 345.27)
11	(5.03, 8.77)	(198.03, 345.27)
12	(6.08, 8.77)	(239.37, 345.27)
13	(7.13, 8.77)	(280.71, 345.27)
14	(8.18, 8.77)	(322.05, 345.27)
15	(8.77, 7.56)	(345.27, 297.64)
16	(8.77, 6.51)	(345.27, 256.30)
17	(8.77, 5.46)	(345.27, 214.96)
18	(8.77, 4.41)	(345.27, 173.62)
19	(8.77, 3.36)	(345.27, 132.28)

Figure 49 shows additional host board keep out area to achieve optimal RF performance with the CYBT-353027-02 module.

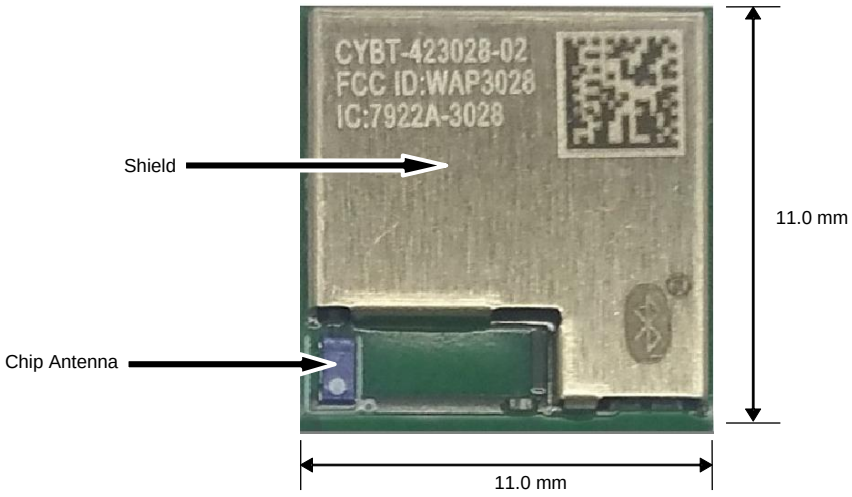
Figure 49. Host Board Additional Keep Out Area for Optimal RF Performance



B.3 CYBT-423028-02

CYBT-423028-02 is a CYW20719-based Bluetooth 5.0 dual-mode (BR/EDR + BLE) module containing 1 MB of on-chip flash and supporting new BLE features such as 2 Mbps data rate, and SIG Mesh. Figure 50 shows a physical picture of the CYBT-423028-02 EZ-BT WICED Module.

Figure 50. CYBT-423028-02 Module Top View



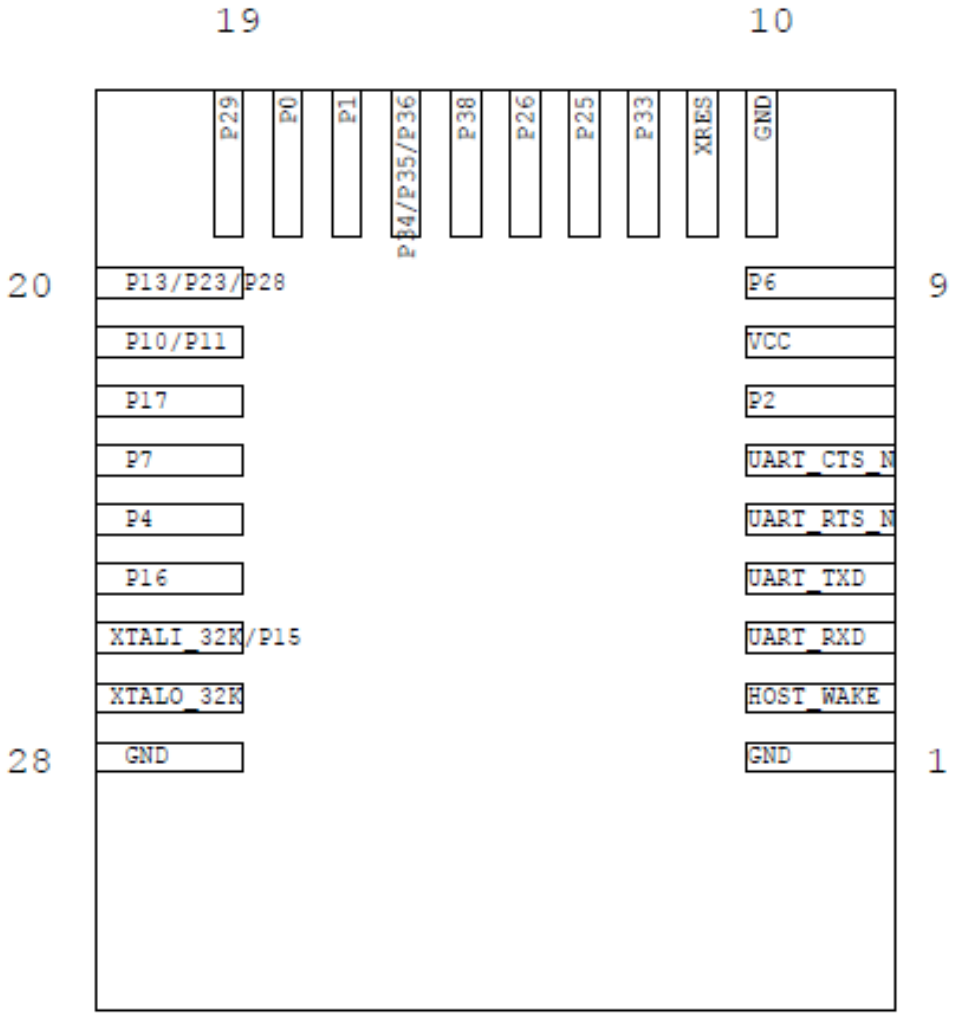
Because of its small form-factor, CYBT-423028-02 is ideal for space-constrained applications that require a large memory footprint, increased processing power, and low-power Bluetooth communication. CYBT-423028-02 supports a number serial communication interfaces (I2C, SPI, 2-wire UART, I2S, PCM) and peripheral functions (ADC, PWM, Timer, Counter), and provides up to 17 GPIOs.

For more details on this module's dimensions, external component connections, and module placement recommendations, see the CYBT-423028-02 datasheet.

B.3.1 Pinout and Functionality

The CYBLE-423028-02 module is designed to mount as a component on an end-product PCB. Only a portion of the available connections of the CYW20719 WICED Bluetooth silicon device are exposed on the CYBT-423028-02 module in order to minimize the module footprint size. The CYBT-423028-02 module contains 28 connections on the bottom side of the module. Figure 51 details the bottom side connections available on the CYBT-423028-02 module.

Figure 51. CYBT-423028-02 Module Bottom View (Seen from Bottom)



A list of the available I/Os and supported functionality for each I/O of CYBT-423028-02 is shown in Table 11.

Table 11. CYBT-423028-02 Module Available Connections and Functionality

Solder Pad	Pad Name	Silicon Port Pin Name	Functionality			
			XTALI/O	ADC	GPIO	SuperMux[7] Capable
1	GND	GND	Ground			
2	HOST_WAKE	BT_HOST_WAKE	A signal from the CYBT-423028-02 module to the host indicating that the Bluetooth device requires attention.			
3	UART_RXD	BT_UART_RXD	HCI UART Receive Data			
4	UART_TXD	BT_UART_TXD	HCI UART Transmit Data			
5	UART_RTS	BT_UART_RTS_N	HCI UART Request to Send Output			
6	UART_CTS	BT_UART_CTS	HCI UART Clear to Send Input			
7	P2	P2			Yes	Yes
8	VCC	VDDIO	Power Supply Input (1.71V ~ 3.63V)			
9	P6	P6			Yes	Yes
10	GND	GND	Ground			
11	XRES	RST_N	External Reset (Active LOW)			
12	P33	P33		IN6	Yes	Yes
13	P25	P25			Yes	Yes
14	P26	P26			Yes	Yes
15	P38	P38		IN1	Yes	Yes
16	P34/P35/P36	P34 P35 P36		IN5 (P34) IN4 (P35) IN3 (P36)	Yes (P34/P35/P36)	Yes
17	P1	P1		IN28	Yes	Yes
18	P0	P0		IN29	Yes	Yes
19	P29	P29		IN10	Yes	Yes
20	P13/P23/P28	P13 P23 P28		IN22 (P13) IN12 (P23) IN11 (P28)	Yes (P13/P23/P28)	Yes
21	P10/P11	P10 P11		IN25 (P10) IN14 (P11)	Yes (P10/P11)	Yes
22	P17	P17		IN18	Yes	Yes
23	P7	P7			Yes	
24	P4	P4			Yes	
25	P16	P16		IN19	Yes	
26	XTALI_32K/P15[8]	XTALI_32K P15	External Oscillator Input (32 kHz)	IN20 (P15)	Yes (P15)	Yes (P15)
27	XTALO_32K	XTALO_32K	External Oscillator Output (32 kHz)			
28	GND	GND	Ground			

[7] The CYBT-423028-02 can configure GPIO connections to any Input/Output function described in Table 12.

[8] P15 should not be driven HIGH externally while the part is held in reset (it can be floating or driven LOW). Failure to do so may cause some current to flow through P15 until the device comes out of reset.

Table 12 details the available functions that can be configured on SuperMux capable GPIOs in Table 11.

Table 12. GPIO SuperMux Input and Output Functions

Function	Input or Output	Function Type	GPIOs Required	Function Connection Description
SWD	Input	Serial Communication and Debug	2	SWDCK, Serial Wire Debugger Clock
	Input/Output			SWDIO, Serial Wire Debugger I/O
SPI 1	Input/Output	Serial Communication (Master or Slave)	4 ~ 8	SPI 1 Clock
				SPI 1 Chip Select
				SPI 1 MOSI
				SPI 1 MISO
				SPI 1 I/O 2 (Quad SPI)
				SPI 1 I/O 3 (Quad SPI)
				SPI 1 Interrupt
	Output			SPI 1 DCX (DBI-C DCX 8-bit mode)
SPI 2	Input/Output	Serial Communication (Master or Slave)	4 ~ 8	SPI 2 Clock
				SPI 2 Chip Select
				SPI 2 MOSI
				SPI 2 MISO
				SPI 2 I/O 2 (Quad SPI)
				SPI 2 I/O 3 (Quad SPI)
				SPI 2 Interrupt
	Output			SPI 2 DCX (DBI-C DCX 8-bit mode)
PUART	Input	Serial Communication Input	4	Peripheral UART RX
				Peripheral UART CTS[9]
	Output	Serial Communication Output		Peripheral UART TX
				Peripheral UART RTS[12]
I^2C	Input/Output	Serial Communication (Master or Slave)	2	I2C Clock
				I2C Data
I^2C 2	Input/Output	Serial Communication (Master or Slave)	2	I2C 2 Clock
				I2C 2 Data
PCM In	Input	Audio Input Communication	3	PCM Input
				PCM Clock
				PCM Sync
PCM Out	Output	Audio Output Communication	3	PCM Output
				PCM Clock
				PCM Sync
I^2S In	Input	Audio Input Communication	3	I2S DI, Data Input
				I2S WS, Word Select
				I2S Clock
I^2S Out	Output	Audio Output Communication	3	I2S DO, Data Output
				I2S WS, Word Select
				I2S Clock

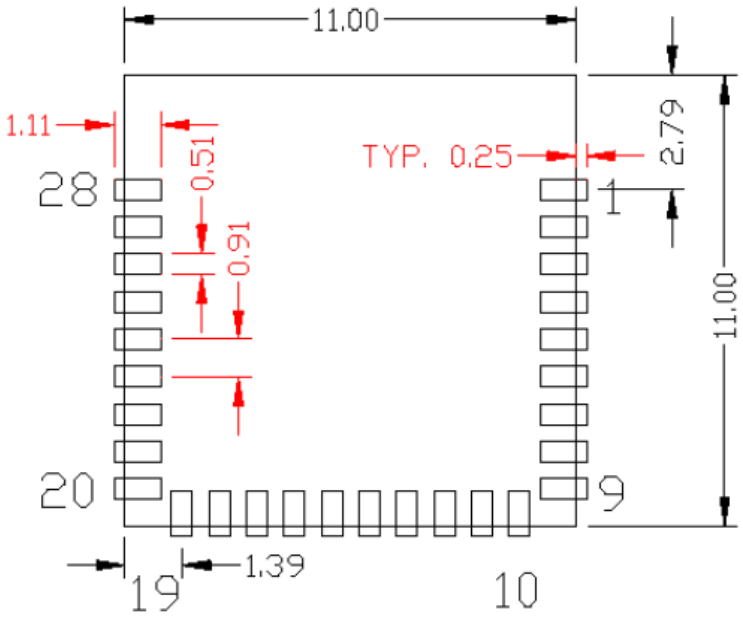
[9] Flow control connections for the peripheral UART (PUART) are not supported in hardware. Hardware flow control is only supported on the HCI UART connection. To enable flow control on the PUART interface, the flow control logic must be developed in the WICED application.

Function	Input or Output	Function Type	GPIOs Required	Function Connection Description
PDM	Input	Microphone	1 ~ 2	PDM Input Channel 1
				PDM Input Channel 2
PWM	Output	Pulse Width Modulator	1 ~ 6	PWM Channel 0
				PWM Channel 1
				PWM Channel 2
				PWM Channel 3
				PWM Channel 4
				PWM Channel 5
ACLK	Output	Auxiliary Clock	1 ~ 2	Auxiliary Clock 0 (ACLK0)
				Auxiliary Clock 1 (ACLK1)
HIDOFF	Output	HID-OFF Indicator	1	HID-OFF Indicator to host

B.3.2 Host Recommended PCB Layout

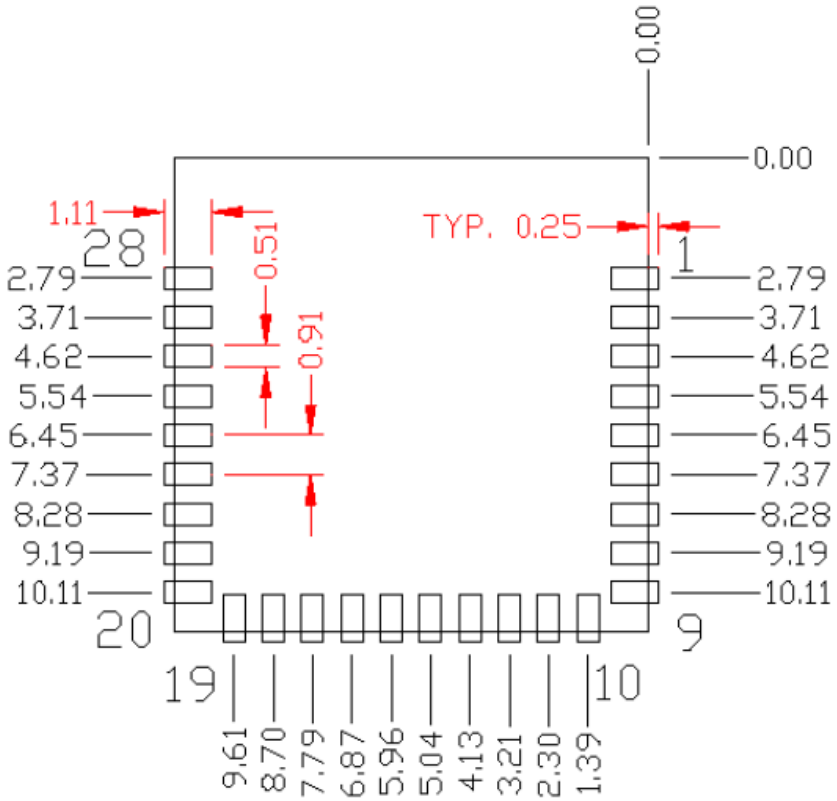
To assist in the host PCB layout design for CYBT-423028-02, Cypress provides three host PCB landing pattern reference drawings in Figure 52, Figure 53, and in Figure 54 and Table 13. Figure 52 provides a dimensioned view of the host PCB layout. Figure 53 provides the location to the center edge of each solder pad relative to the origin of the module (upper right PCB outline). Figure 54 and Table 13 provides the location to each solder pad center location for the host PCB layout. Dimensions shown are in mm unless otherwise stated.

Figure 52. Host Board Required PCB Layout Pattern (Dimensioned View)



Note: Pad length shown includes overhang of the host pad beyond the module pad outline. The minimum recommended pad length on the host PCB is 1.11 mm.

Figure 53. Host Board Required PCB Layout Pattern: To Pad Center Edge Relative to Origin



Top View (Seen on Host PCB)

Figure 54. Host Board Required PCB Layout Pattern To Pad Center Relative to Origin

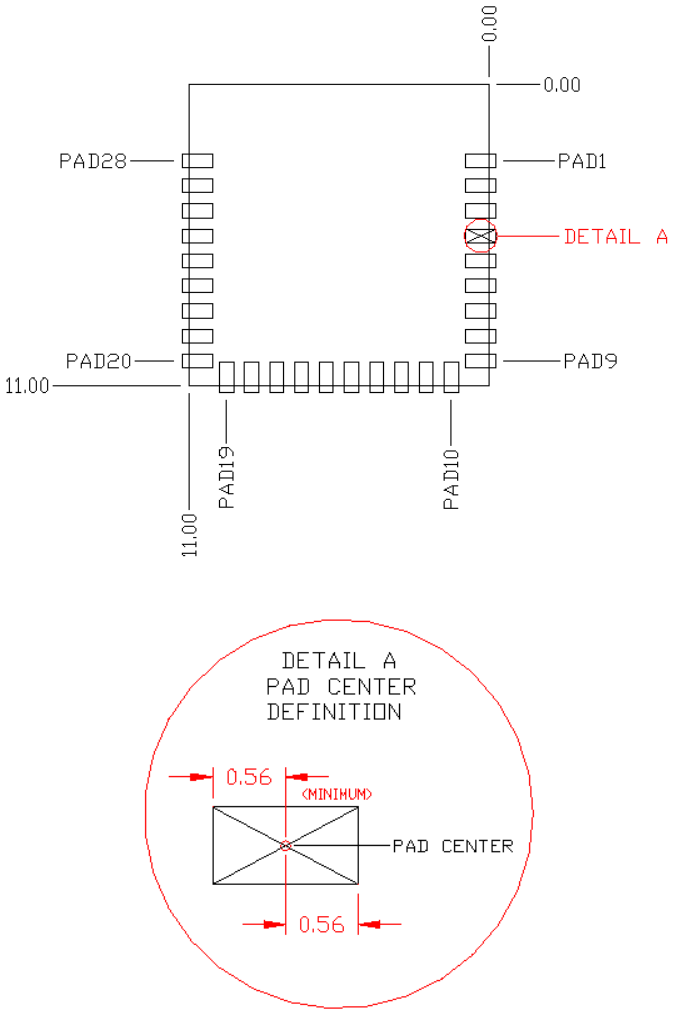
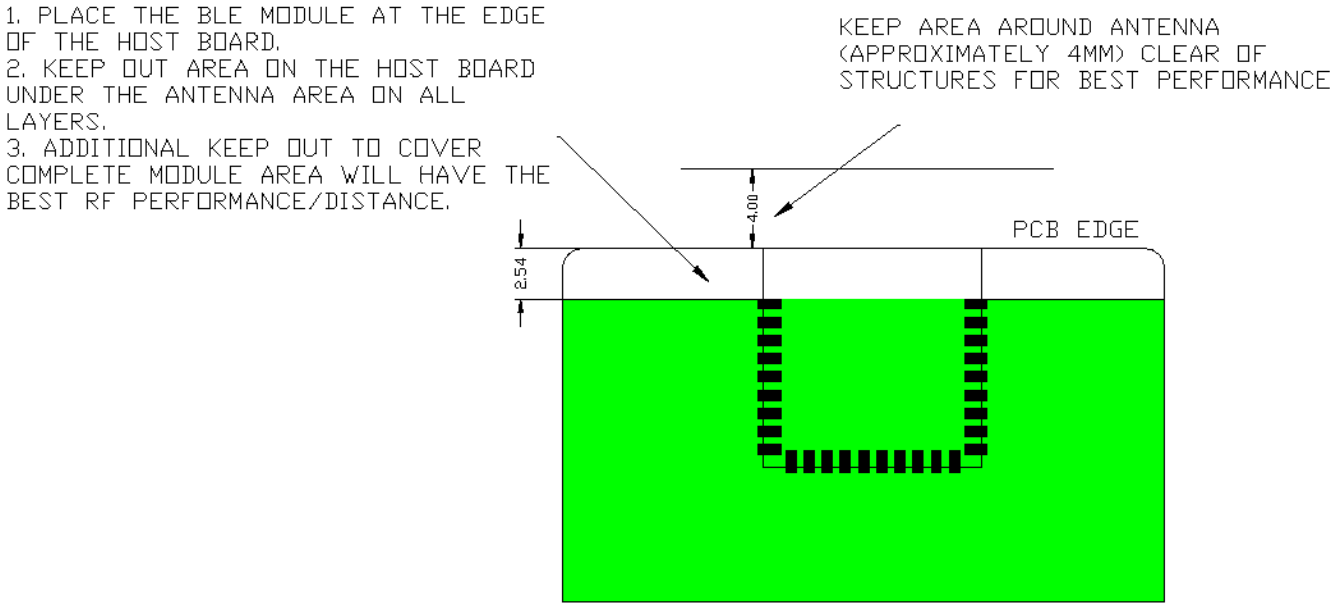


Table 13. Location to Pad Center from Origin (dimensions in mm and mils)

Solder Pad (Center of Pad)	Location (X,Y) from Origin (mm)	Location (X,Y) from Origin (mils)
1	(0.31, 2.79)	(12.20, 109.84)
2	(0.31, 3.71)	(12.20, 146.06)
3	(0.31, 4.62)	(12.20, 181.89)
4	(0.31, 5.54)	(12.20, 218.11)
5	(0.31, 6.45)	(12.20, 253.94)
6	(0.31, 7.37)	(12.20, 290.16)
7	(0.31, 8.28)	(12.20, 325.98)
8	(0.31, 9.19)	(12.20, 361.81)
9	(0.31, 10.11)	(12.20, 398.03)
10	(1.39, 10.69)	(54.72, 420.87)
11	(2.30, 10.69)	(90.55, 420.87)
12	(3.21, 10.69)	(126.38, 420.87)
13	(4.13, 10.69)	(162.60, 420.87)
14	(5.04, 10.69)	(198.42, 420.87)
15	(5.96, 10.69)	(234.65, 420.87)
16	(6.87, 10.69)	(270.47, 420.87)
17	(7.79, 10.69)	(306.69, 420.87)
18	(8.70, 10.69)	(342.52, 420.87)
19	(9.61, 10.69)	(378.35, 420.87)
20	(10.69, 10.11)	(420.87, 398.03)
21	(10.69, 9.19)	(420.87, 361.81)
22	(10.69, 8.28)	(420.87, 325.98)
23	(10.69, 7.37)	(420.87, 290.16)
24	(10.69, 6.45)	(420.87, 253.94)
25	(10.69, 5.54)	(420.87, 218.11)
26	(10.69, 4.62)	(420.87, 181.89)
27	(10.69, 3.71)	(420.87, 146.06)
28	(10.69, 2.79)	(420.87, 109.84)

Figure 55 details additional host board keep out area to achieve optimal RF performance with the CYBT-423028-02 module.

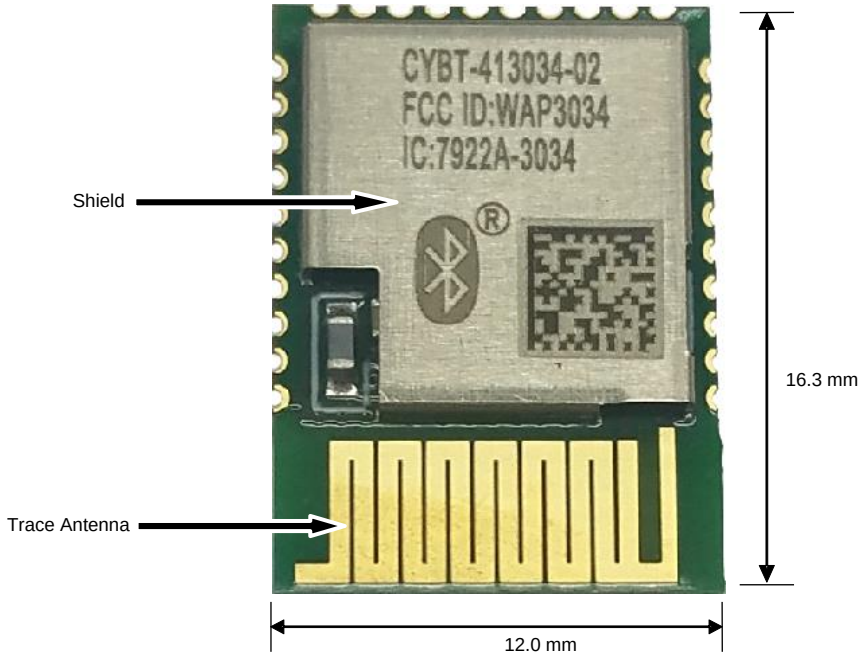
Figure 55. Host Board Additional Keep Out Area for Optimal RF Performance



B.4 CYBT-413034-02

CYBT-413034-02 is a CYW20719-based Bluetooth 5.0 dual-mode (BR/EDR + BLE) module containing 1 MB of on-chip flash and supporting new BLE features such as 2 Mbps data rate, and SIG Mesh. Figure 56 shows a physical picture of the CYBT-413034-02 EZ-BT WICED Module.

Figure 56. CYBT-413034-02 Module Top View



CYBT-413034-02 is ideal for applications which require a large memory footprint, increased processing power, and low-power Bluetooth communication. CYBT-413034-02 supports all of the same functionality that CYBT-423028-02 supports but is designed in a more economical module footprint to save cost for customers that do not require the smallest footprint. CYBT-413034-02 provides up to 17 GPIOs.

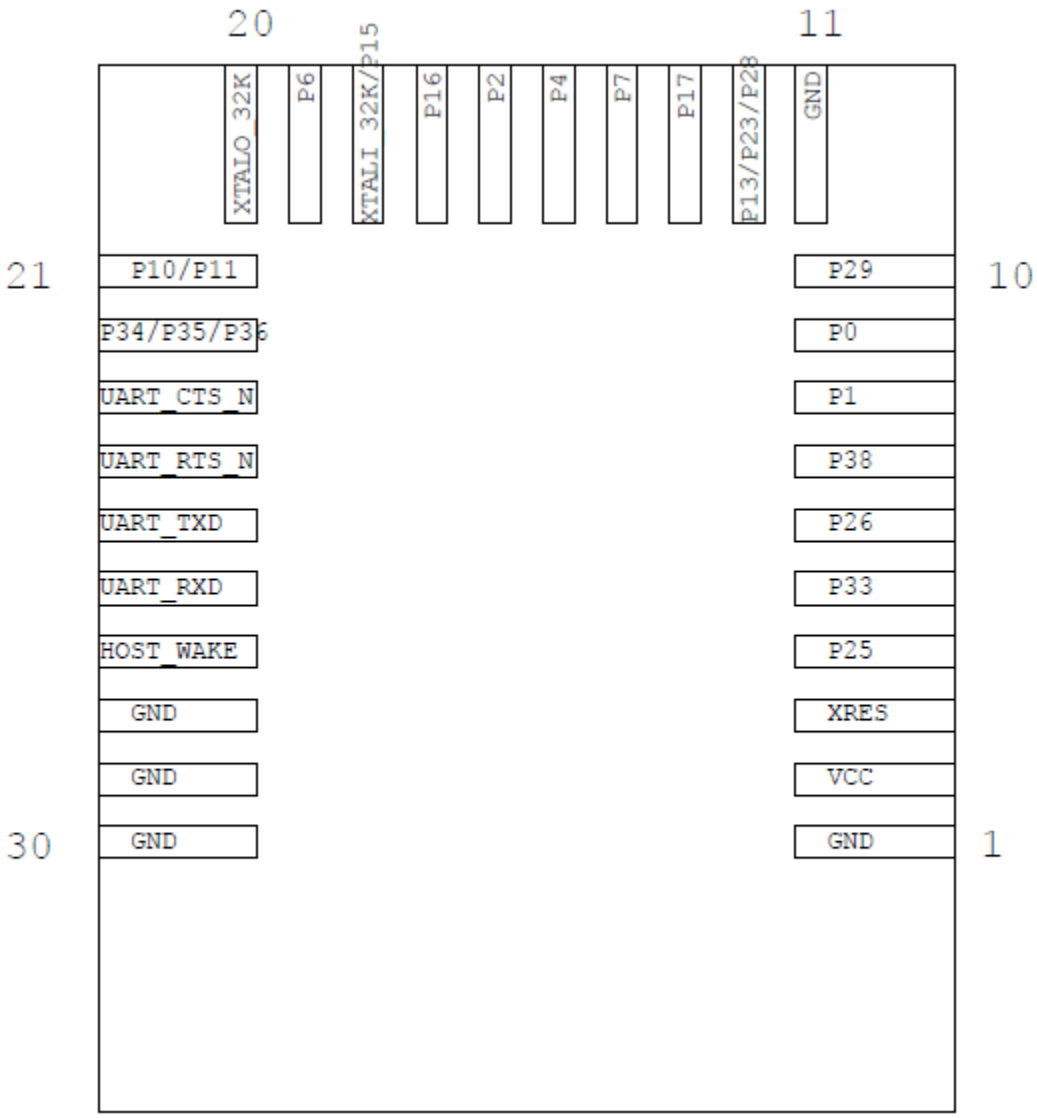
For more details on this module's dimensions, external component connections, and module placement recommendations, see the CYBT-413034-02 datasheet.

B.4.1 Pinout and Functionality

The CYBLE-413034-02 module is designed to mount as a component on an end-product PCB. CYBT-413034-02 is designed with castellated GPIO pads for ease of manual soldering and prototyping.

Only a portion of the available connections of the CYW20719 WICED Bluetooth silicon device are exposed on the CYBT-413034-02 module in order to minimize the module footprint size. The CYBT-413034-02 module contains 30 connections on the castellated pads on the periphery of the module. Figure 57 details the solder pad connections available on the CYBT-413034-02 module.

Figure 57. CYBT-413034-02 Module Bottom View (Seen from Bottom)



A list of the available I/Os and supported functionality for each I/O of CYBT-413034-02 is shown in Table 14.

Table 14. CYBT-413034-02 Module Available Connections and Functionality

Solder Pad	Pad Name	Silicon Port Pin Name	Functionality			
			XTALI/O	ADC	GPIO	SuperMux[10] Capable
1	GND	GND	Ground			
2	VDD	VDDIO	Power Supply Input (1.76V ~ 3.63V)			
3	XRES	RST_N	External Reset (Active LOW)			
4	P25	P25			Yes	Yes
5	P33	P33		IN6	Yes	Yes
6	P26	P26			Yes	Yes
7	P38	P38		IN1	Yes	Yes
8	P1	P1		IN28	Yes	Yes
9	P0	P0		IN29	Yes	Yes
10	P29	P29		IN10	Yes	Yes
11	GND	GND	Ground			
12	P13/P23/P28	P13 P23 P28		IN22 (P13) IN12 (P23) IN11 (P28)	Yes (P13/P23/P28)	Yes
13	P17	P17		IN18	Yes	Yes
14	P7	P7			Yes	
15	P4	P4			Yes	
16	P2	P2			Yes	Yes
17	P16	P16		IN19	Yes	
18	XTALI_32K/P15[11]	XTALI_32K P15	External Oscillator Input (32 kHz)	IN20 (P15)	Yes (P15)	Yes (P15)
19	P6	P6			Yes	Yes
20	XTALO_32K	XTALO_32K	External Oscillator Output (32 kHz)			
21	P10/P11	P10 P11		IN25 (P10) IN14 (P11)	Yes (P10/P11)	Yes
22	P34/P35/P36	P34 P35 P36		IN5 (P34) IN4 (P35) IN3 (P36)	Yes (P34/P35/P36)	Yes
23	UART_CTS	BT_UART_CTS	HCI UART Clear to Send Input			
24	UART_RTS	BT_UART_RTS_N	HCI UART Request to Send Output			
25	UART_TXD	BT_UART_TXD	HCI UART Transmit Data			
26	UART_RXD	BT_UART_RXD	HCI UART Receive Data			
27	HOST_WAKE	BT_HOST_WAKE	A signal from the CYBT-413034-02 module to the host indicating that the Bluetooth device requires attention.			
28	GND	GND	Ground			
29	GND	GND	Ground			
30	GND	GND	Ground			

Table 12 on Page 74 details the available functions that can be configured on SuperMux capable GPIOs shown above.

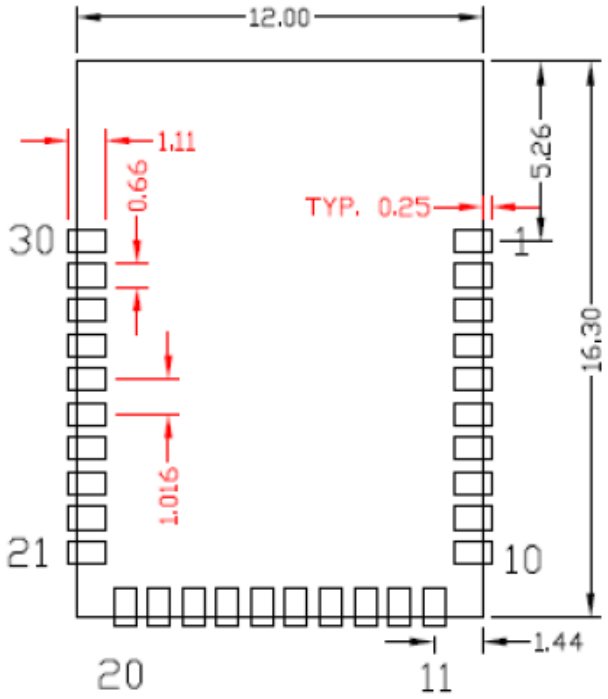
[10] CYBT-423028-02 can configure GPIO connections to any Input/Output function described in Table 12.

[11] P15 should not be driven HIGH externally while the part is held in reset (it can be floating or driven LOW). Failure to do so may cause some current to flow through P15 until the device comes out of reset.

B.4.2 Host Recommended PCB Layout

To assist in the host PCB layout design for CYBT-413034-02, Cypress provides three host PCB landing pattern reference drawings in Figure 58, Figure 59, and in Figure 60 and Table 15. Figure 58 provides a dimensioned view of the host PCB layout. Figure 59 provides the location to the center edge of each solder pad relative to the origin of the module (upper right PCB outline). Figure 60 and Table 15 provides the location to each solder pad center location for the host PCB layout. Dimensions shown are in mm unless otherwise stated.

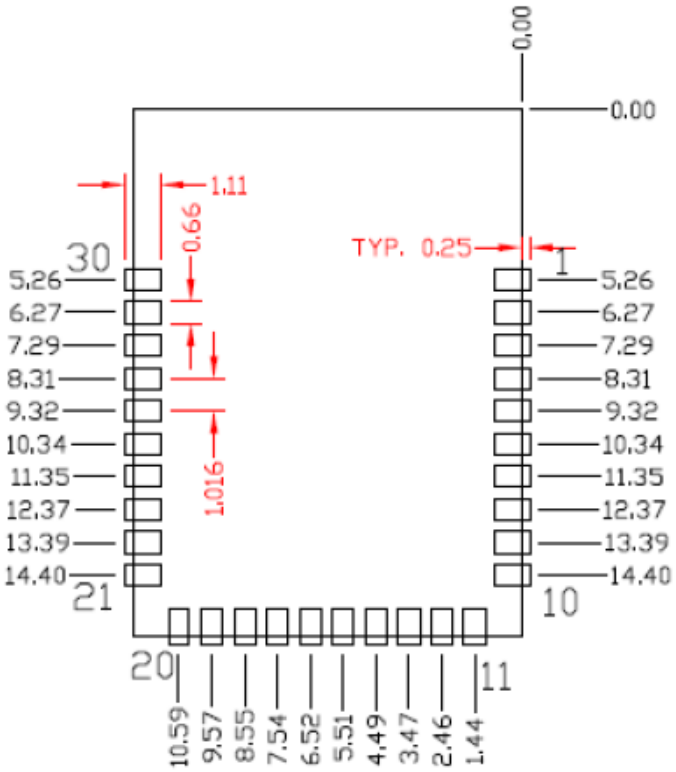
Figure 58. Host Board Required PCB Layout Pattern (Dimensioned View)



Top View (Seen on Host PCB)

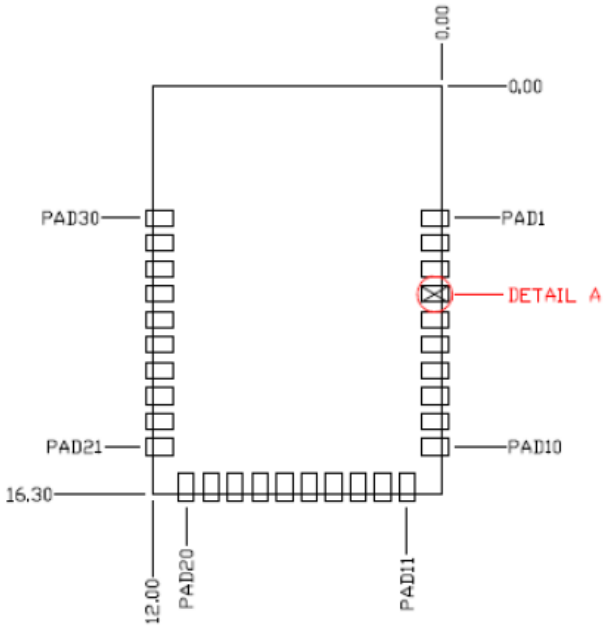
Note: Pad length shown includes overhang of the host pad beyond the module pad outline. The minimum recommended pad length on the host PCB is 1.11 mm.

Figure 59. Host Board Required PCB Layout Pattern: To Pad Center Edge Relative to Origin



Top View (Seen on Host PCB)

Figure 60. Host Board Required PCB Layout Pattern To Pad Center Relative to Origin



Top View (Seen on Host PCB)

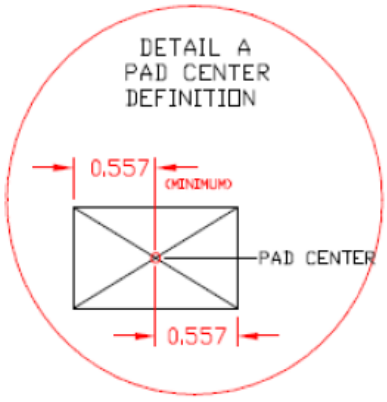
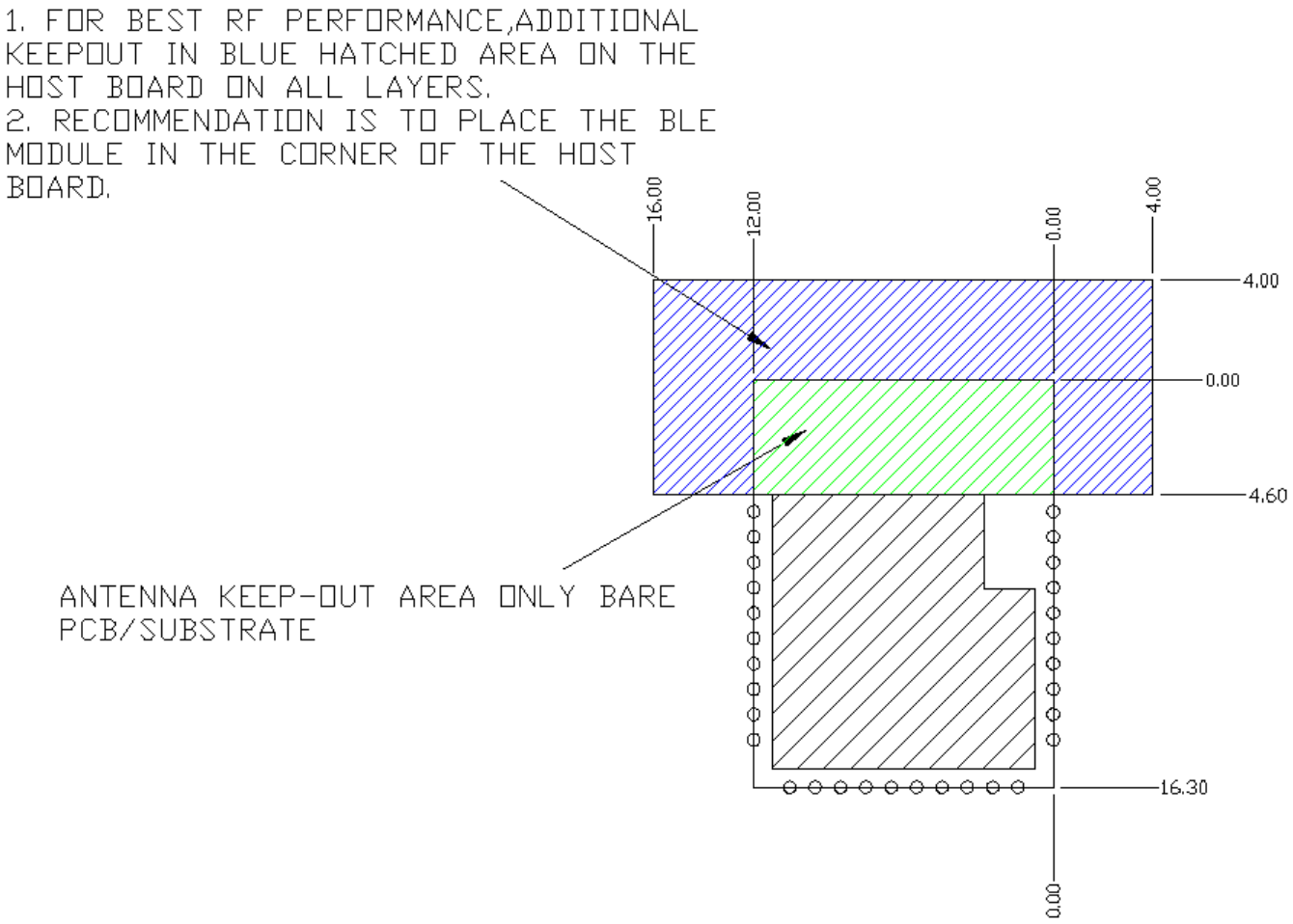


Table 15. Location to Pad Center from Origin (dimensions in mm and mils)

Solder Pad (Center of Pad)	Location (X,Y) from Origin (mm)	Location (X,Y) from Origin (mils)
1	(0.31, 5.26)	(12.20, 207.09)
2	(0.31, 6.27)	(12.20, 246.85)
3	(0.31, 7.29)	(12.20, 287.01)
4	(0.31, 8.31)	(12.20, 327.16)
5	(0.31, 9.32)	(12.20, 366.93)
6	(0.31, 10.34)	(12.20, 407.09)
7	(0.31, 11.35)	(12.20, 446.85)
8	(0.31, 12.37)	(12.20, 487.01)
9	(0.31, 13.39)	(12.20, 527.16)
10	(0.31, 14.40)	(12.20, 566.93)
11	(1.44, 15.99)	(56.69, 629.53)
12	(2.46, 15.99)	(96.85, 629.53)
13	(3.47, 15.99)	(136.61, 629.53)
14	(4.49, 15.99)	(176.77, 629.53)
15	(5.51, 15.99)	(216.93, 629.53)
16	(6.52, 15.99)	(256.69, 629.53)
17	(7.54, 15.99)	(296.85, 629.53)
18	(8.55, 15.99)	(336.61, 629.53)
19	(9.57, 15.99)	(376.77, 629.53)
20	(10.59, 15.99)	(416.93, 629.53)
21	(11.69, 14.40)	(460.24, 566.93)
22	(11.69, 13.39)	(460.24, 527.16)
23	(11.69, 12.37)	(460.24, 487.01)
24	(11.69, 11.35)	(460.24, 446.85)
25	(11.69, 10.34)	(460.24, 407.09)
26	(11.69, 9.32)	(460.24, 366.93)
27	(11.69, 8.31)	(460.24, 327.16)
28	(11.69, 7.29)	(460.24, 287.01)
29	(11.69, 6.27)	(460.24, 246.85)
30	(11.69, 5.26)	(460.24, 207.09)

Figure 61 details additional host board keep out area to achieve optimal RF performance with the CYBT-413034-02 module.

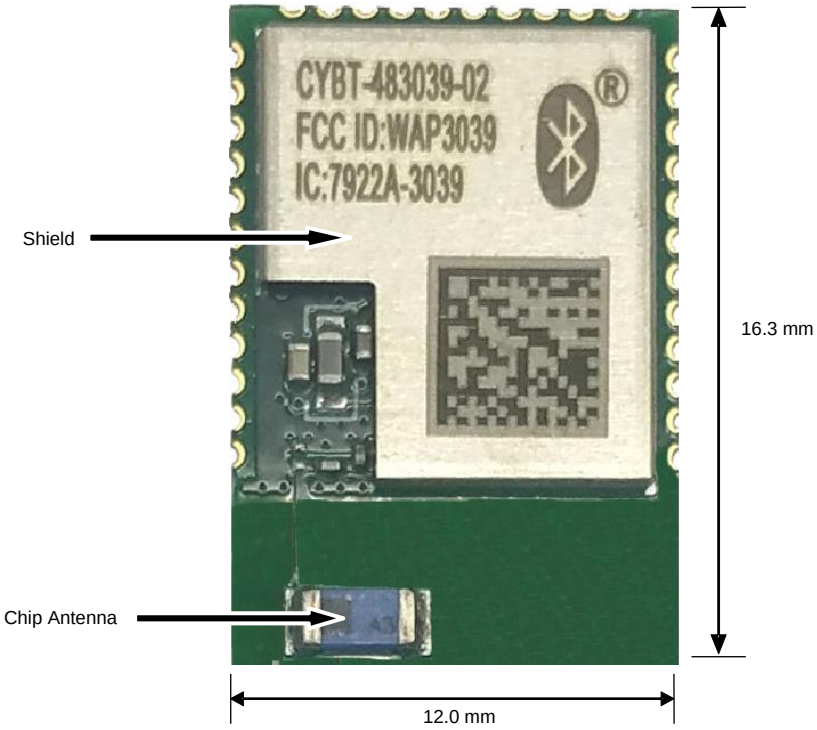
Figure 61. Host Board Additional Keep Out Area for Optimal RF Performance



B.5 CYBT-483039-02

CYBT-483039-02 is a CYW20719-based Bluetooth 5.0 dual-mode (BR/EDR + BLE) module containing 1 MB of on-chip flash, supports new BLE features such as 2 Mbps data rate and SIG Mesh, and boasts the longest range in the Cypress Bluetooth portfolio with over 1 km of connection-based range. Figure 62 shows a physical picture of the CYBT-483039-02 EZ-BT WICED Module.

Figure 62. CYBT-483039-02 Module Top View



CYBT-483039-02 is a specialized module, ideal for customers desiring maximum connection range for Bluetooth Low Energy solutions. CYBT-483029-02 can be used to communicate over long distances (>1 km), or to provide adequate range in challenging environments. If maximizing the range of your application is not a priority in your design, then it is recommended to look at the CYBT-423028-02 or CYBT-413034-02 modules instead. CYBT-483039-02 provides up to 15 GPIOs.

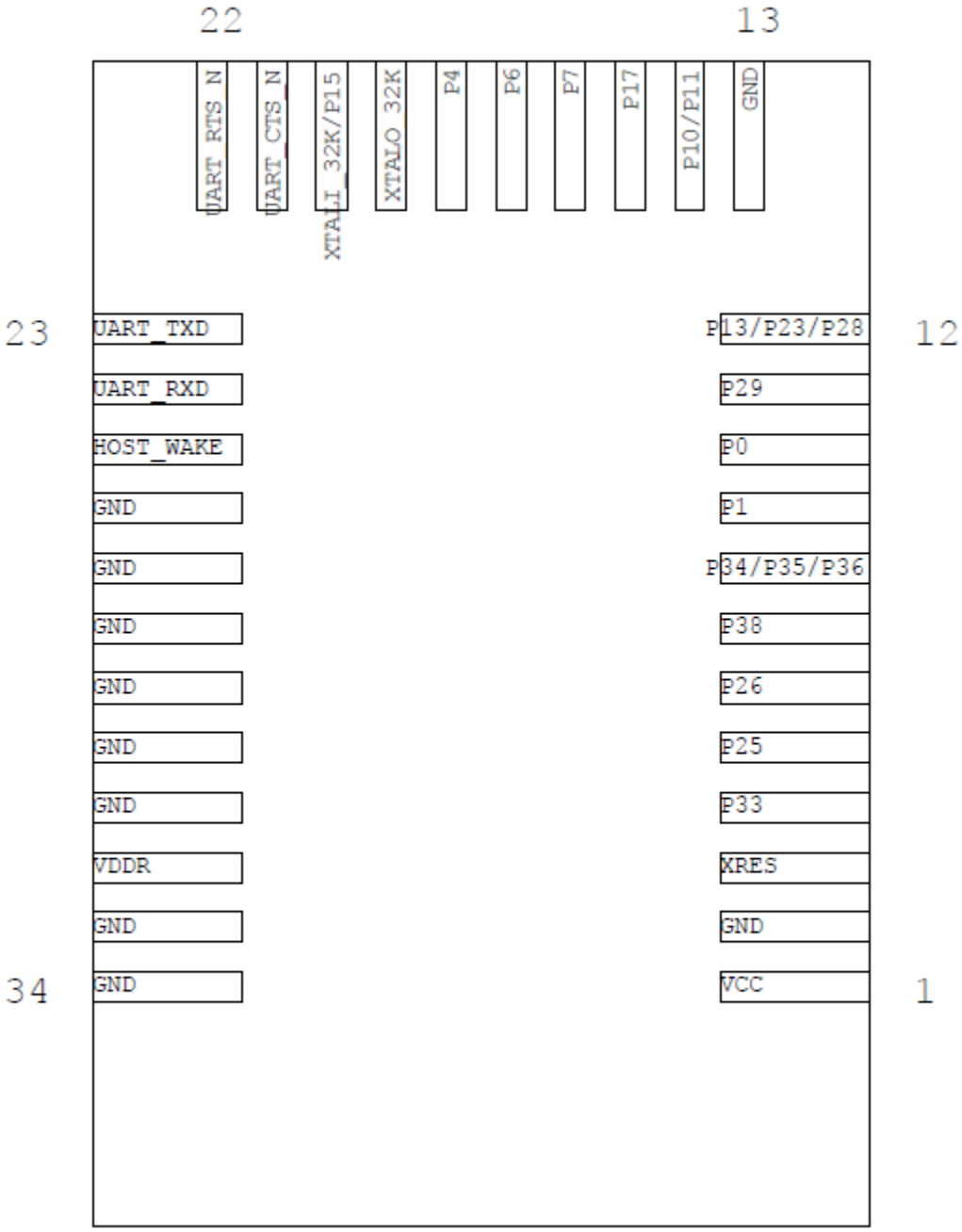
For more details on this module's dimensions, external component connections, and module placement recommendations, see the CYBT-483039-02 datasheet.

B.5.1 Pinout and Functionality

CYBLE-483039-02 module is designed to mount as a component on an end-product PCB. CYBT-483039-02 is designed with castellated GPIO pads for ease of manual soldering and prototyping.

Only a portion of the available connections of the CYW20719 WICED Bluetooth silicon device are exposed on the CYBT-483039-02 module in order to minimize the module footprint size. The CYBT-483039-02 module contains 34 connections on the castellated pads on the periphery of the module. Figure 63 details the solder pad connections available on the CYBT-483039-02 module.

Figure 63. CYBT-483039-02 Module Bottom View (Seen from Bottom)



A list of the available I/Os and supported functionality for each I/O of CYBT-483039-02 is shown in Table 16.

Table 16. CYBT-483039-02 Module Available Connections and Functionality

Solder Pad	Pad Name	Silicon Port Pin Name	Functionality			
			XTALI/O	ADC	GPIO	SuperMux[12] Capable
1	VDD	VDDIO	Power Supply Input (2.0V ~ 3.63V)			
2	GND	GND	Ground			
3	XRES	RST_N	External Reset (Active LOW)			
4	P33	P33		IN6	Yes	Yes
5	P25	P25			Yes	Yes
6	P26	P26			Yes	Yes
7	P38	P38		IN1	Yes	Yes
8	P34/P35/P36	P34 P35 P36		IN5 (P34) IN4 (P35) IN3 (P36)	Yes (P34/P35/P36)	Yes
9	P1	P1		IN28	Yes	Yes
10	P0	P0		IN29	Yes	Yes
11	P29	P29		IN10	Yes	Yes
12	P13/P23/P28	P13 P23 P28		IN22 (P13) IN12 (P23) IN11 (P28)	Yes (P13/P23/P28)	Yes
13	GND	GND	Ground			
14	P10/P11	P10 P11		IN25 (P10) IN14 (P11)	Yes (P10/P11)	Yes
15	P17	P17		IN18	Yes	Yes
16	P7	P7			Yes	
17	P6	P6			Yes	Yes
18	P4	P4			Yes	
19	XTALO_32K	XTALO_32K	External Oscillator Output (32 kHz)			
20	XTALI_32K/P15[13]	XTALI_32K P15	External Oscillator Input (32 kHz)	IN20 (P15)	Yes (P15)	Yes (P15)
21	UART_CTS	BT_UART_CTS	HCI UART Clear to Send Input			
22	UART_RTS	BT_UART_RTS_N	HCI UART Request to Send Output			
23	UART_TXD	BT_UART_TXD	HCI UART Transmit Data			
24	UART_RXD	BT_UART_RXD	HCI UART Receive Data			
25	HOST_WAKE	BT_HOST_WAKE	A signal from the CYBT-413034-02 module to the host indicating that the Bluetooth device requires attention.			
26	GND	GND	Ground			
27	GND	GND	Ground			

[12] CYBT-423028-02 can configure GPIO connections to any Input/Output function described in Table 12.

[13] P15 should not be driven HIGH externally while the part is held in reset (it can be floating or driven LOW). Failure to do so may cause some current to flow through P15 until the device comes out of reset.

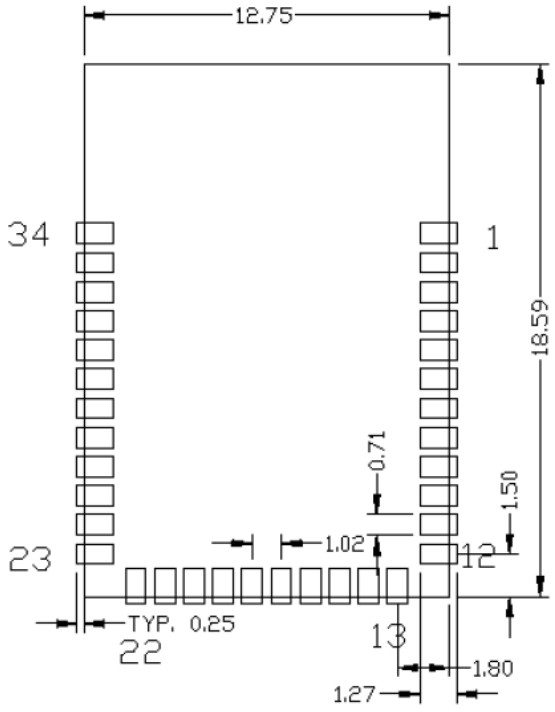
Solder Pad	Pad Name	Silicon Port Pin Name	Functionality			
			XTALI/O	ADC	GPIO	SuperMux[12] Capable
28	GND	GND	Ground			
29	GND	GND	Ground			
30	GND	GND	Ground			
31	GND	GND	Ground			
32	VDDPA	N/A	PA/LNA Power Supply Voltage (2.0 ~ 3.6V)			
33	GND	GND	Ground			
34	GND	GND	Ground			

Table 12 on Page 74 details the available functions that can be configured on SuperMux capable GPIOs shown above.

B.5.2 Host Recommended PCB Layout

To assist in the host PCB layout design for CYBT-483039-02, Cypress provides three host PCB landing pattern reference drawings in Figure 64, Figure 65, and in Figure 66 and Table 17. Figure 64 provides a dimensioned view of the host PCB layout. Figure 65 provides the location to the center edge of each solder pad relative to the origin of the module (upper right PCB outline). Figure 66 and Table 17 provides the location to each solder pad center location for the host PCB layout. Dimensions shown are in mm unless otherwise stated.

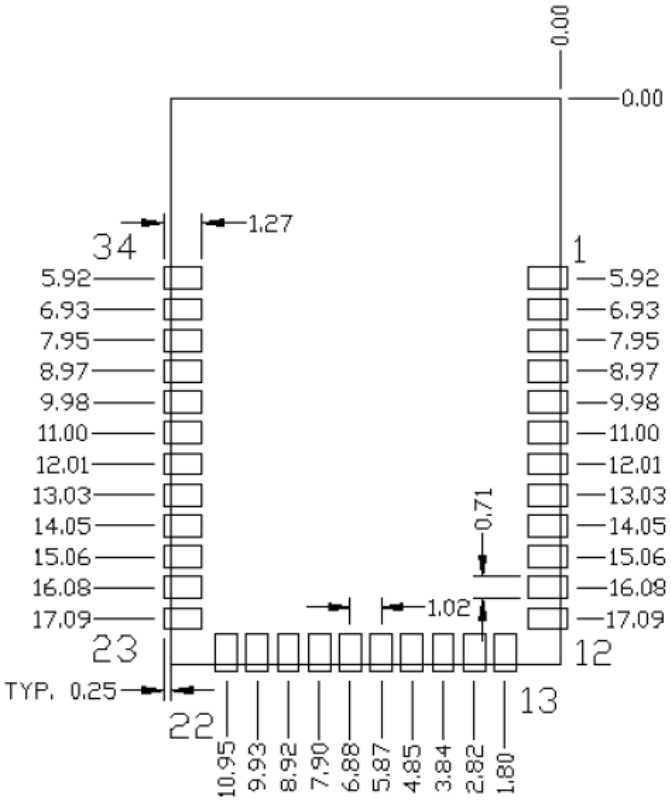
Figure 64. Host Board Required PCB Layout Pattern (Dimensioned View)



Top View (Seen on Host PCB)

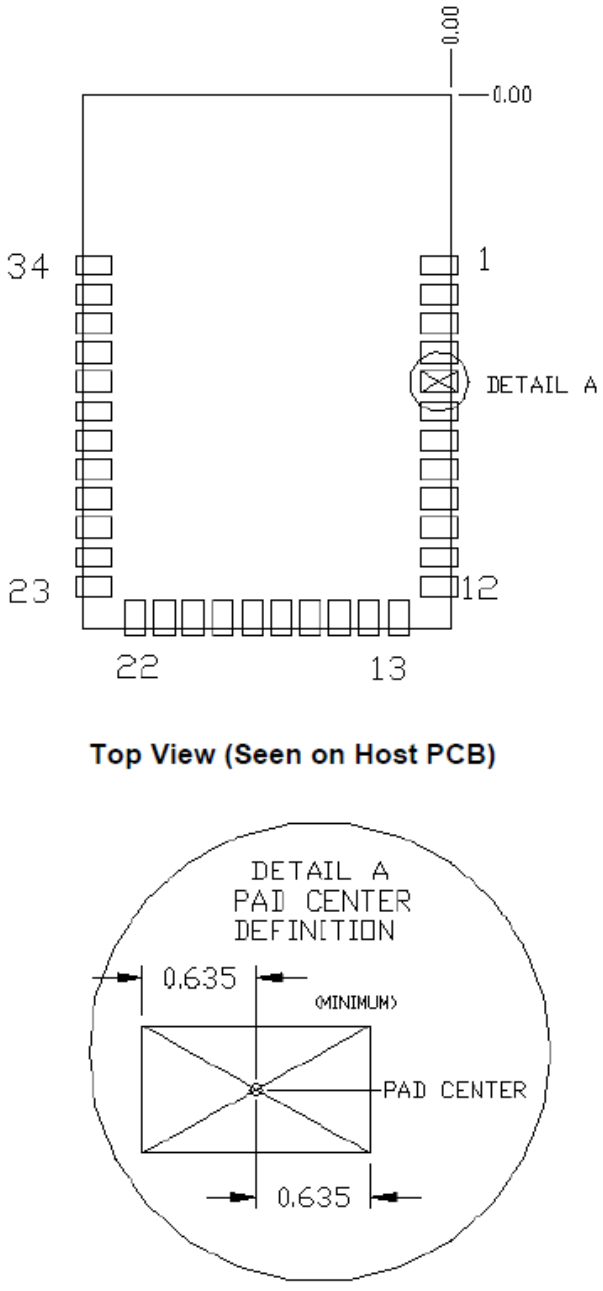
Note: Pad length shown includes overhang of the host pad beyond the module pad outline. The minimum recommended pad length on the host PCB is 1.27 mm.

Figure 65. Host Board Required PCB Layout Pattern: To Pad Center Edge Relative to Origin



Top View (Seen on Host PCB)

Figure 66. Host Board Required PCB Layout Pattern To Pad Center Relative to Origin



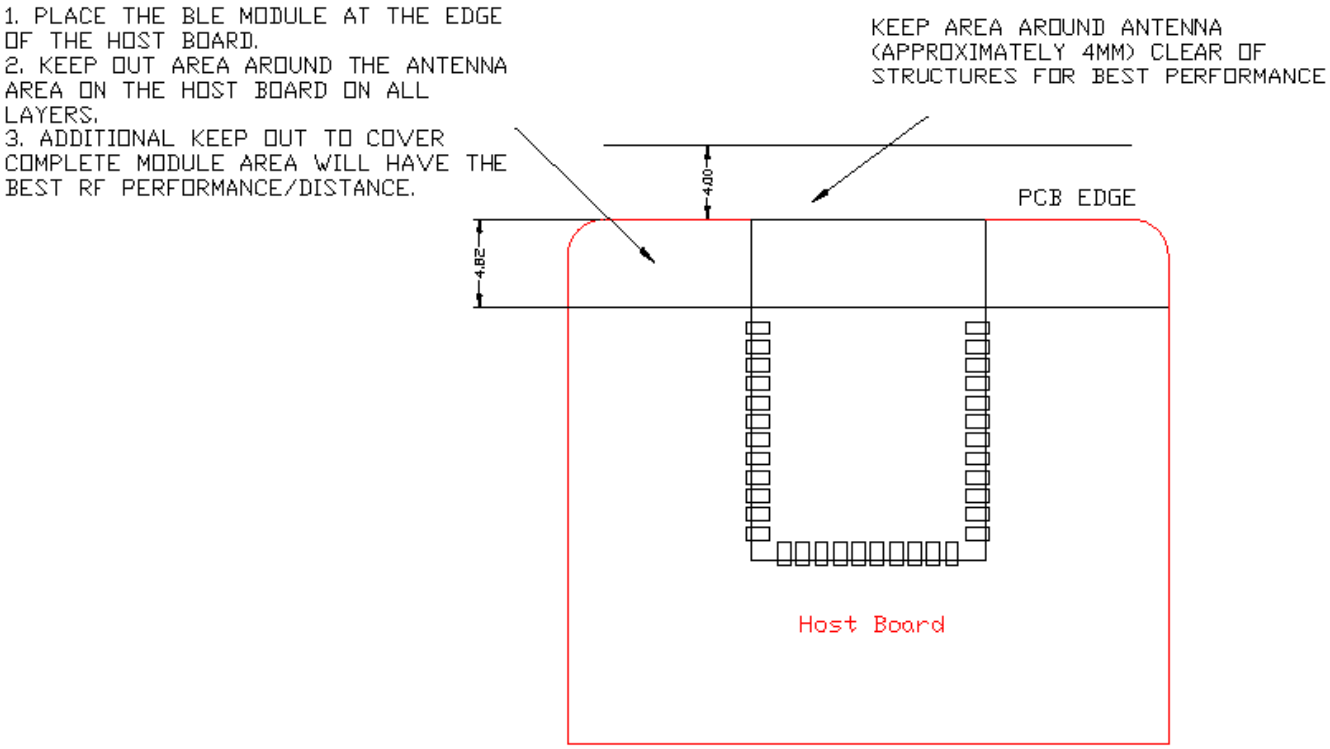
Top View (Seen on Host PCB)

Table 17. Location to Pad Center from Origin (dimensions in mm and mils)

Solder Pad (Center of Pad)	Location (X,Y) from Origin (mm)	Location (X,Y) from Origin (mils)
1	(0.38, 5.92)	(14.96, 233.07)
2	(0.38, 6.93)	(14.96, 272.83)
3	(0.38, 7.95)	(14.96, 312.99)
4	(0.38, 8.97)	(14.96, 353.15)
5	(0.38, 9.98)	(14.96, 392.91)
6	(0.38, 11.00)	(14.96, 433.07)
7	(0.38, 12.01)	(14.96, 472.83)
8	(0.38, 13.03)	(14.96, 512.99)
9	(0.38, 14.05)	(14.96, 553.15)
10	(0.38, 15.06)	(14.96, 592.91)
11	(0.38, 16.08)	(14.96, 633.07)
12	(0.38, 17.09)	(14.96, 672.83)
13	(1.80, 18.21)	(70.87, 716.93)
14	(2.82, 18.21)	(111.02, 716.93)
15	(3.84, 18.21)	(151.18, 716.93)
16	(4.85, 18.21)	(190.94, 716.93)
17	(5.87, 18.21)	(231.10, 716.93)
18	(6.88, 18.21)	(270.87, 716.93)
19	(7.90, 18.21)	(311.02, 716.93)
20	(8.92, 18.21)	(351.18, 716.93)
21	(9.93, 18.21)	(390.94, 716.93)
22	(10.95, 18.21)	(431.10, 716.93)
23	(12.37, 17.09)	(487.01, 672.83)
24	(12.37, 16.08)	(487.01, 633.07)
25	(12.37, 15.06)	(487.01, 592.91)
26	(12.37, 14.05)	(487.01, 553.15)
27	(12.37, 13.03)	(487.01, 512.99)
28	(12.37, 12.01)	(487.01, 472.83)
29	(12.37, 11.00)	(487.01, 433.07)
30	(12.37, 9.98)	(487.01, 392.91)
31	(12.37, 8.97)	(487.01, 353.15)
32	(12.37, 7.95)	(487.01, 312.99)
33	(12.37, 6.93)	(487.01, 272.83)
34	(12.37, 5.92)	(487.01, 233.07)

Figure 67 details additional host board keep out area to achieve optimal RF performance with the CYBT-483039-02 module.

Figure 67. Host Board Additional Keep Out Area for Optimal RF Performance



Appendix C. EZ-BT WICED Evaluation Boards

Appendix C provides detailed information on each EZ-BT WICED Evaluation Boards. The information contained for each subsection below includes the following:

- Physical image for each EZ-BT WICED Evaluation marketing part number
- What's included on the specific EZ-BT WICED Evaluation board

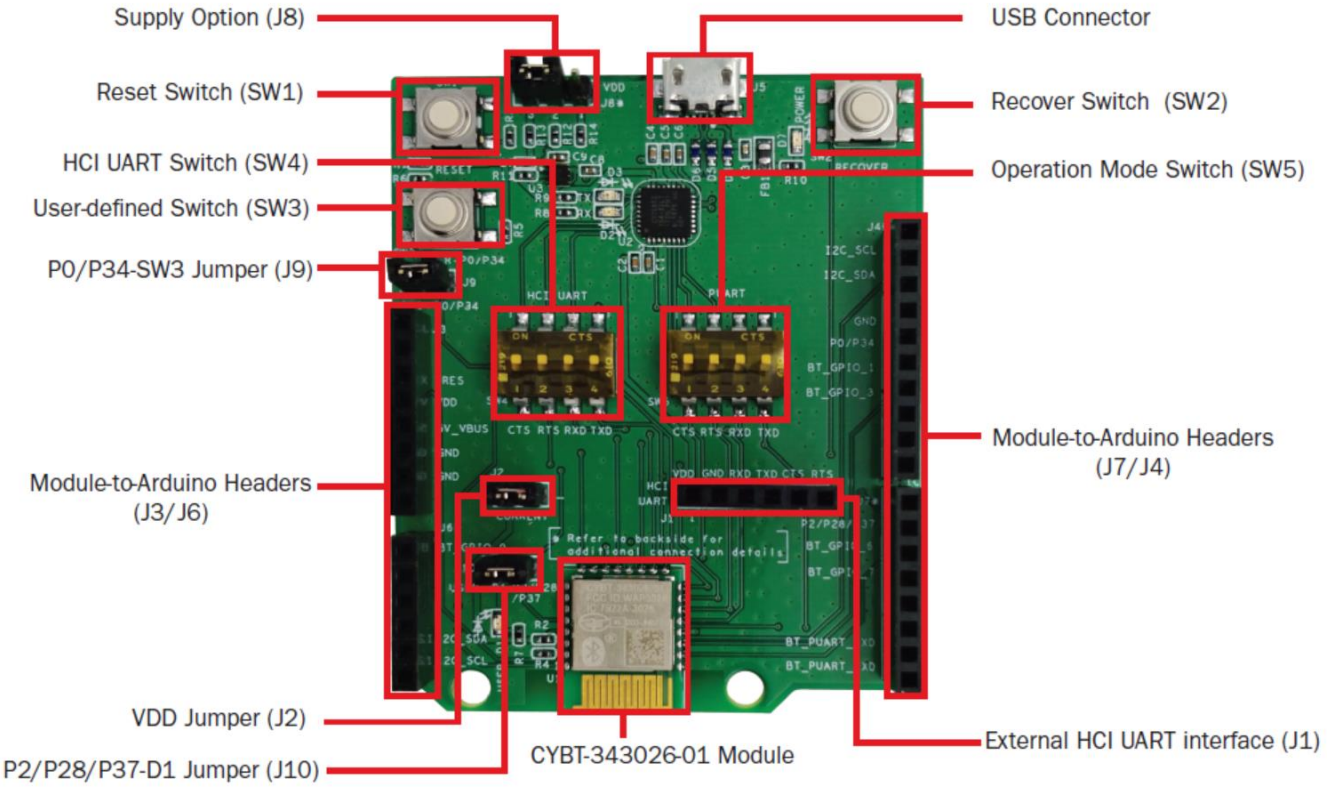
To jump to your specific EZ-BT WICED Evaluation board, click the marketing part number in the below list:

- CYBT-343026-EVAL
- CYBT-353027-EVAL
- CYBT-423028-EVAL
- CYBT-413034-EVAL
- CYBT-483039-EVAL

C.1 CYBT-343026-EVAL

CYBT-343026-EVAL is the evaluation board for the CYBT-343026-01 WICED module. Figure 68 shows the CYBT-343026-EVAL board and calls out the main components and connections available on the board.

Figure 68. CYBT-343026-EVAL Evaluation Board



Note: Connections not called out on J3, J4, J6, and J7 Arduino-compatible headers are NC (No Connect), where no physical connection is present between the CYBT-343026-01 EZ-BT WICED Module and associated headers.

CYBT-343026-EVAL includes the following elements:

C.1.1 Active Devices

- EZ-BT WICED Module: The EZ-BT WICED Module is mounted to the evaluation board as shown in Figure 68.
- USB-to-UART bridge device: A USB-to-UART bridge device is provided on the evaluation board to translate USB communication to UART communication (and vice-versa). For example, when HCI UART communication is configured to “On” on SW4, the USB traffic from the Host PC is connected to the HCI UART connections of the CYBT-343026-01 module.

C.1.2 Connectors and Headers

- USB receptacle: The USB connection on the CYBT-343026-EVAL board provides power to the evaluation board and also provides communication to the board via USB, which is translated to UART communication and routed to either the HCI UART or PUART connection on the EZ-BT WICED Module depending on the configuration of SW4 and SW5 on CYBT-343026-EVAL.
- Power Supply Option Header (J8): J8 allows for configuration of three different power supplies to the EZ-BT WICED Module. All power is sourced from the USB connection, as mentioned above, but the power supply directly input to the CYBT-343026-01 module can be configured to either 2.3 V, 3.3 V, or 3.6 V using the provided three-pin header. The power supply input is configured by shorting two neighboring header positions on J8 or leaving the header open. Table 18 details the available power supply options and the associated header position connections required.

Table 18. J8 Header Power Supply Connection Options

J8 Jumper Configuration	VDD Voltage Level
Short 1 & 2	3.6 V Supply
Short 2 & 3	3.3 V Supply
No Jumper (all headers open)	2.3 V Supply
All Other Configurations	Not Allowed

- Arduino-compatible base headers – headers J3/J4/J6/J7: The CYBT-343026-EVAL provides Arduino-compatible base headers that can be used for Arduino shield connections. The associated signals that are routed out to these headers are noted on the PCB silk screen on both the top and bottom side of the evaluation board.
- HCI UART direct connection header (J1): The J1 header provides all HCI UART communication lines to the user. This allows for connection to the EZ-BT WICED Module without having to connect through USB. This can be used to connect a host controller evaluation board directly to the EZ-BT WICED Module HCI UART connection. HCI UART connections are not brought out to the Arduino-compatible headers. The J1 direct HCI UART connection bypasses the SW4 (HCI UART to USB-to-UART bridge to Host PC) configuration. It is recommended that SW4 elements 1 ~ 4 are placed in the OFF position if the J1 direct HCI UART connection is used.
- Power consumption measurement header (J2): J2 is provided to allow for easy power consumption measurement reading with a multimeter or current measurement probe. Refer to KBA223800 for information related to low-power implementations for the CYBT-343026-01 module as well as power consumption measurement methods using the J2 header.
- User Element Disconnection Headers (J9 and J10): J9 (USER SW3) and J10 (USER D1 - LED) are provided to allow for disconnection of these elements from the CYBT-343026-01 module. In both cases, the GPIO that is routed to these USER elements is also routed to the Arduino-compatible headers. Disconnecting the elements (SW3 and/or D1) may be required if you plan to use either of these GPIOs through the Arduino-compatible headers.

C.1.3 UART Configuration Switches and LEDs

- SW4 (HCI UART configuration switch): SW4 controls the connection of USB data traffic to the HCI UART connections of the module. SW4 is a four-element switch (each element corresponding to the 4-wire UART signals – TX, RX, CTS, RTS); used to connect HCI UART signals on CYBT-343026-EVAL to the host PC USB interface. To connect HCI UART signals from CYBT-343026-01 to the host PC, simply put each element of SW4 in the “On” position. To disconnect HCI UART communication from the CYBT-343026-01 module and the host PC, simply set each of the SW4 elements to the Off position. There is no High-Z configuration available on the SW4 elements; only On or Off positions. Flow control is supported in hardware on the HCI UART connection.
- SW5 (Peripheral UART (PUART) configuration switch): SW5 controls the connection of the PUART connections of the CYBT-343026-01 module to the host PC connection (via the USB/UART Bridge). SW5 is a four-element switch; to connect PUART signals on CYBT-343026-EVAL to the host PC, simply put each element of SW4 in the “On” position. To disconnect PUART communication from the CYBT-343026-02 module and the host PC, simply set each of the SW5 elements to the Off position. There is no High-Z configuration available on the SW5 elements; only On or Off positions.

  Note: PUART on the CYBT-343026-01 module does not support flow control in HW. Flow control logic is required to be developed in the WICED SDK if this is required. In any case, if flow control is developed and enabled, only the GPIOs that are noted as supporting RTS and/or CTS can be used for this functionality.

  Note: PUART connections from the CYBT-343026-01 module to Arduino headers are not gated by SW5. These connections are always present.

Notes on HCI UART and PUART configurations:

- It is acceptable to have SW4 and SW5 elements all set to the “On” position at the same time (i.e., HCI UART and PUART both connected to the CYBT-343026-01 module). PUART and HCI UART connections are handled by two different hardware UART blocks, and the connection to the CYBT-343026-01 module from the USB-to-UART bridge enumerates as two different COM ports on the host PC (not a shared connection/bus).
- Both HCI UART and PUART configurations can be used for UART communication within your application.
- HCI UART (Positions 1 ~ 4 of SW4 = **ON**) is the UART configuration generally used for interfacing to the WICED Studio SDK, with the goal of downloading a compiled image to the EZ-BT WICED Module. The HCI UART connection can also be used for general UART communication in Application mode (executing a downloaded

image in SFLASH). If 4-wire UART with flow control is required, it is recommended to use HCI UART as the UART connection for your application.

- PUART (Positions 1 ~ 4 of SW5 = **ON**) is the configuration to use when you need to have the PUART connected to the host PC. This PUART interface to the host PC can be used for terminal communication (print statements).

  Note: PUART communication signals routed to Arduino headers are not gated by SW5 and can be used to communicate with any peripheral device over UART.

  Note: The PUART interface cannot be used to download a new binary image to the CYBT-343026-01 module.

C.1.4 Additional Switches and LEDs

- SW1 (RESET switch): SW1 is a tactile switch that is connected directly to the XRES connection of the CYBT-343026-01 module. Activating this switch will reset the EZ-BT WICED Module. This switch is also used in the process of (re-)programming the CYBT-343026-01 module on the evaluation board.
- SW2 (RECOVER switch): SW2 is provided to recover a module that has the standard programming bootloader and/or application image corrupted or erased. Recovery mode is activated by holding this button down while pressing and releasing SW1 (RESET). Recovery mode is used prior to programming the module with a new binary image.
- SW3 (USER switch): SW3 is provided as an element that the user can configure as required. The SW3 element is connected to P0/P34 on the CYBT-343026-01 module (solder pad 1). Header J9 is provided on the CYBT-343026-EVAL board to disconnect P0/P34 from the USER switch to allow this connection to be used on the Arduino Headers without SW3 connected.
- D1 USER LED is provided for user-configured behavior as required. The D1 LED is connected to the module P2/P28/P37 connection (solder pad 5). Header J10 is provided on the CYBT-343026-EVAL board to disconnect P2/P28/P37 from the USER LED to allow this connection to be used on the Arduino Headers without SW3 connected.
- D2 and D3 LEDs are provided to display programming or USB activity while in progress.
- D7 LED is provided to show that power is provided from the host PC.

C.1.5 Programming or Reprogramming the CYBT-343026-EVAL Board

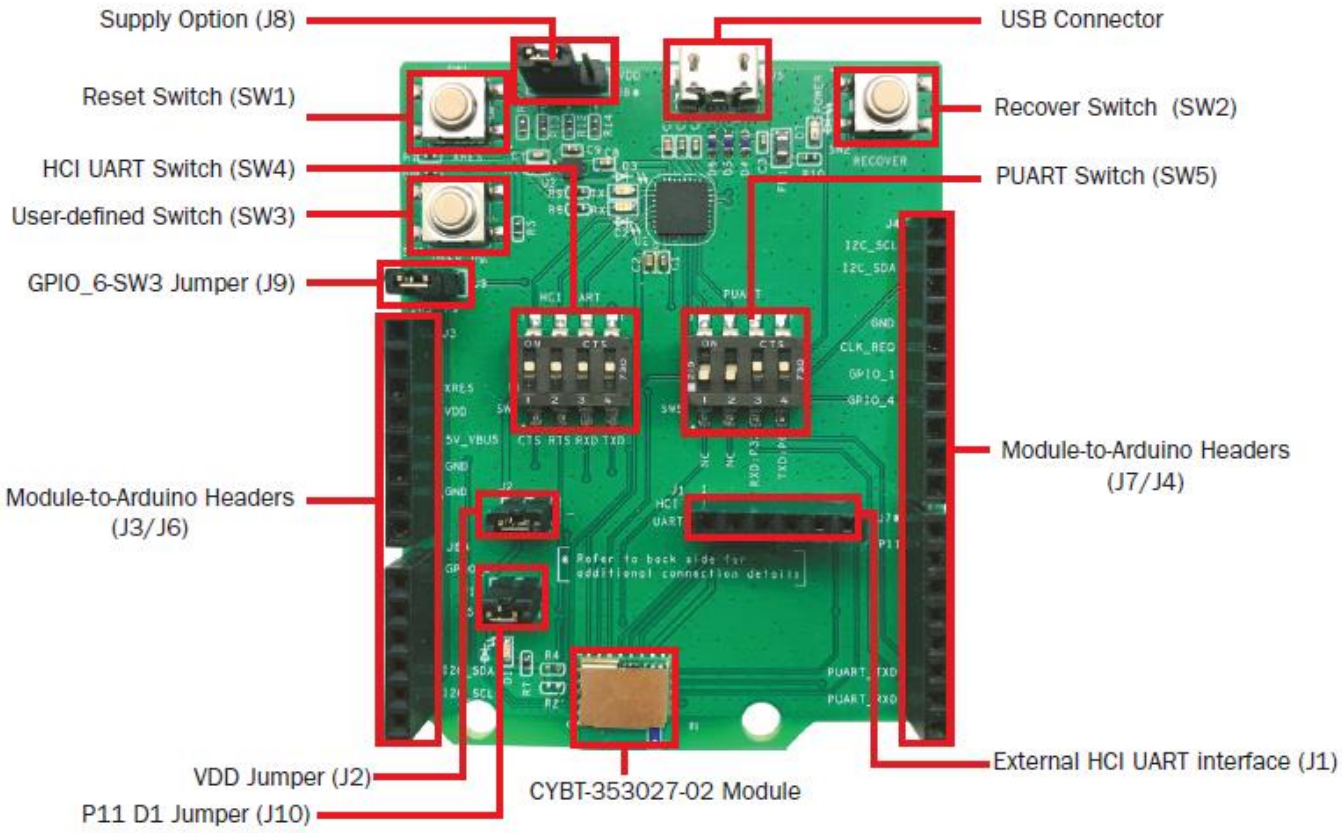
Refer to Section 7.5.3 for general information on how to program or reprogram EZ-BT WICED Modules. You may also refer to KBA223428 for online information.

To program an EZ-BT WICED module in your own manufacturing line, you must ensure that both the Recover and Reset connections on the EZ-BT module are exposed in order to put the module into programming/recovery mode.

C.2 CYBT-353027-EVAL

CYBT-353027-EVAL is the evaluation board for the CYBT-353027-02 WICED module. Figure 69 shows the CYBT-353027-EVAL board and calls out the main components and connections available on the board.

Figure 69. CYBT-353027-EVAL Evaluation Board



Note: Connections not called out on J3, J4, J6, and J7 Arduino compatible headers are NC (No Connect), where no physical connection is present between the CYBT-353027-02 EZ-BT WICED Module and the associated headers.

The CYBT-353027-EVAL includes the following elements:

C.2.1 Active Devices

- EZ-BT WICED Module: The EZ-BT WICED Module is mounted to the evaluation board as shown in Figure 69.
- USB-to-UART bridge device: A USB-to-UART bridge device is provided on the evaluation board to translate USB communication to UART communication (and vice-versa). For example, when HCI UART communication is configured to “On” on SW4, the USB traffic from the Host PC is connected to the HCI UART connections of the CYBT-353027-02 module.

C.2.2 Connectors and Headers

- USB receptacle: The USB connection on the CYBT-353027-EVAL board provides power to the evaluation board and also provides communication to the board via USB, which is translated to UART communication and routed to either the HCI UART or PUART connection on the EZ-BT WICED Module depending on the configuration of SW4 and SW5 on CYBT-353027-EVAL.
- Power Supply Option Header (J8): J8 allows for configuration of three different power supplies to the EZ-BT WICED Module. All power is sourced from the USB connection, as mentioned above, but the power supply directly input to the CYBT-353027-02 module can be configured to either 2.3 V, 3.3 V, or 3.6 V using the provided three-pin header. The power supply input is configured by shorting two neighboring header positions on J8 or leaving the

header open. Table 19 details the available power supply options and the associated header position connections required.

Table 19. J8 Header Power Supply Connection Options

J8 Jumper Configuration	VDD Voltage Level
Short 1 & 2	3.6 V Supply
Short 2 & 3	3.3 V Supply
No Jumper (all headers open)	2.3 V Supply
All Other Configurations	Not Allowed

- Arduino-compatible base headers – headers J3/J4/J6/J7: CYBT-353027-EVAL provides Arduino-compatible base headers that can be used for Arduino shield connections. Associated signals that are routed out to these headers are noted on the PCB silk screen on both the top and bottom side of the evaluation board.
- HCI UART direct connection header (J1): The J1 header provides all HCI UART communication lines to the user. This allows for connection to the EZ-BT WICED Module without having to connect through USB. This can be used to connect a host controller evaluation board directly to the EZ-BT WICED Module HCI UART connection. HCI UART connections are not brought out to the Arduino-compatible headers. The J1 direct HCI UART connection bypasses the SW4 (HCI UART to USB-to-UART bridge to Host PC) configuration. It is recommended that SW4 elements 1 ~ 4 are placed in the OFF position if the J1 direct HCI UART connection is used.
- Power consumption measurement header (J2): J2 is provided to allow for easy power consumption measurement reading with a multimeter or current measurement probe. Refer to KBA223800 for information related to low-power implementations for the CYBT-353027-02 module as well as power consumption measurement methods using the J2 header.
- User Element Disconnection Headers (J9 and J10): J9 (USER SW3) and J10 (USER D1 - LED) are provided to allow for disconnection of these elements from the CYBT-353027-02 module. In both cases, the GPIO that is routed to these USER elements are also routed to the Arduino-compatible headers. Disconnecting the elements (SW3 and/or D1) may be required if you plan to use either of these GPIOs through the Arduino-compatible headers.

C.2.3 Switches and LEDs

- SW4 (HCI UART configuration switch): SW4 controls the connection of USB data traffic to the HCI UART connections of the module. SW4 is a four-element switch (each element corresponding to the 4-wire UART signals – TX, RX, CTS, RTS); used to connect the HCI UART signals on CYBT-353027-EVAL to the host PC USB interface. To connect HCI UART signals from CYBT-353027-02 to the host PC, simply put each element of SW4 in the “On” position. To disconnect HCI UART communication from the CYBT-353027-02 module and the host PC, simply set each of the SW4 elements to the Off position. There is no High-Z configuration available on the SW4 elements; only On or Off positions. Flow control is supported in hardware on the HCI UART connection.
- SW5 (Peripheral UART (PUART) configuration switch): SW5 controls the connection of the PUART signals of the CYBT-353027-02 module to the host PC connection (via the USB/UART Bridge). SW5 is a four-element switch; to connect PUART signals on CYBT-353027-EVAL to the host PC, simply put each element of SW4 in the “On” position. To disconnect PUART communication from the CYBT-353027-02 module and the host PC, simply set each of the SW5 elements to the Off position. There is no High-Z configuration available on the SW5 elements; only On or Off positions. PUART of the CYBT-353027-02 module supports only 2-wire UART interface. Flow control is not supported on PUART of the CYBT-353027-02 module.

  Note: PUART connections from the CYBT-353027-02 module to the Arduino headers are not gated by SW5. These connections are always present.

Notes on HCI UART and PUART configurations:

- It is acceptable to have SW4 and SW5 elements all set to the “On” position at the same time (i.e., HCI UART and PUART both connected to the CYBT-353027-02 module). The PUART and HCI UART connections are handled by two different hardware UART blocks, and the connection to the CYBT-353027-02 module from the USB-to-UART bridge enumerates as two different COM ports on the host PC (not a shared connection/bus).
- Both HCI UART and PUART configurations can be used for UART communication within your application.
- HCI UART (Positions 1 ~ 4 of SW4 = **ON**) is the UART configuration generally used for interfacing to the WICED Studio SDK, with the goal of downloading a compiled image to the EZ-BT WICED Module. The HCI UART connection can also be used for general UART communication in Application mode (executing a downloaded

image in SFLASH). If 4-wire UART with flow control is required, it is recommended to use HCI UART as the UART connection for your application.

- PUART (Positions 1 ~ 4 of SW5 = **ON**) is the configuration to use when desiring to have the PUART connected to the host PC. This PUART interface to the host PC can be used for terminal communication (print statements).

  Note: PUART communication signals routed to Arduino headers are not gated by SW5 and can be used to communicate with any peripheral device over UART.

  Note: The PUART interface cannot be used to download a new binary image to the CYBT-353027-02 module.

C.2.4 Additional Switches and LEDs

- SW1 (RESET switch): SW1 is a tactile switch that is connected directly to the XRES connection of the CYBT-353027-02 module. Activating this switch will reset the EZ-BT WICED Module. This switch is also used in the process of (re-)programming the CYBT-353027-02 module on the evaluation board.
- SW2 (RECOVER switch): SW2 is provided to recover a module that has the standard programming bootloader and/or application image corrupted or erased. Recovery mode is activated by holding this button down while pressing and releasing SW1 (RESET). Recovery mode is used prior to programming the module with a new binary image.
- SW3 (USER switch): SW3 is provided as an element that the user can configure as required. The SW3 element is connected to GPIO_6 on the CYBT-353027-02 module (solder pad 18). Header J9 is provided on the CYBT-353027-EVAL board to disconnect GPIO_6 from the USER switch to allow this connection to be used on the Arduino Headers without SW3 connected.
- D1 USER LED is provided for user-configured behavior as required. The D1 LED is connected to the module P11 connection (solder pad 5). Header J10 is provided on the CYBT-353027-EVAL board to disconnect P11 from the USER LED to allow this connection to be used on the Arduino Headers without D1 connected. Note that the CYBT-353027-02 module does not support PWM hardware blocks, so functionality of the D1 LED is limited to driving the LED using a GPIO signal (either on, off, or a firmware-based timer PWM representation).
- D2 and D3 LEDs are provided to display programming or USB activity while in progress.
- D7 LED is provided to show that power is provided from the host PC.

C.2.5 Programming or Reprogramming the CYBT-353027-EVAL Board

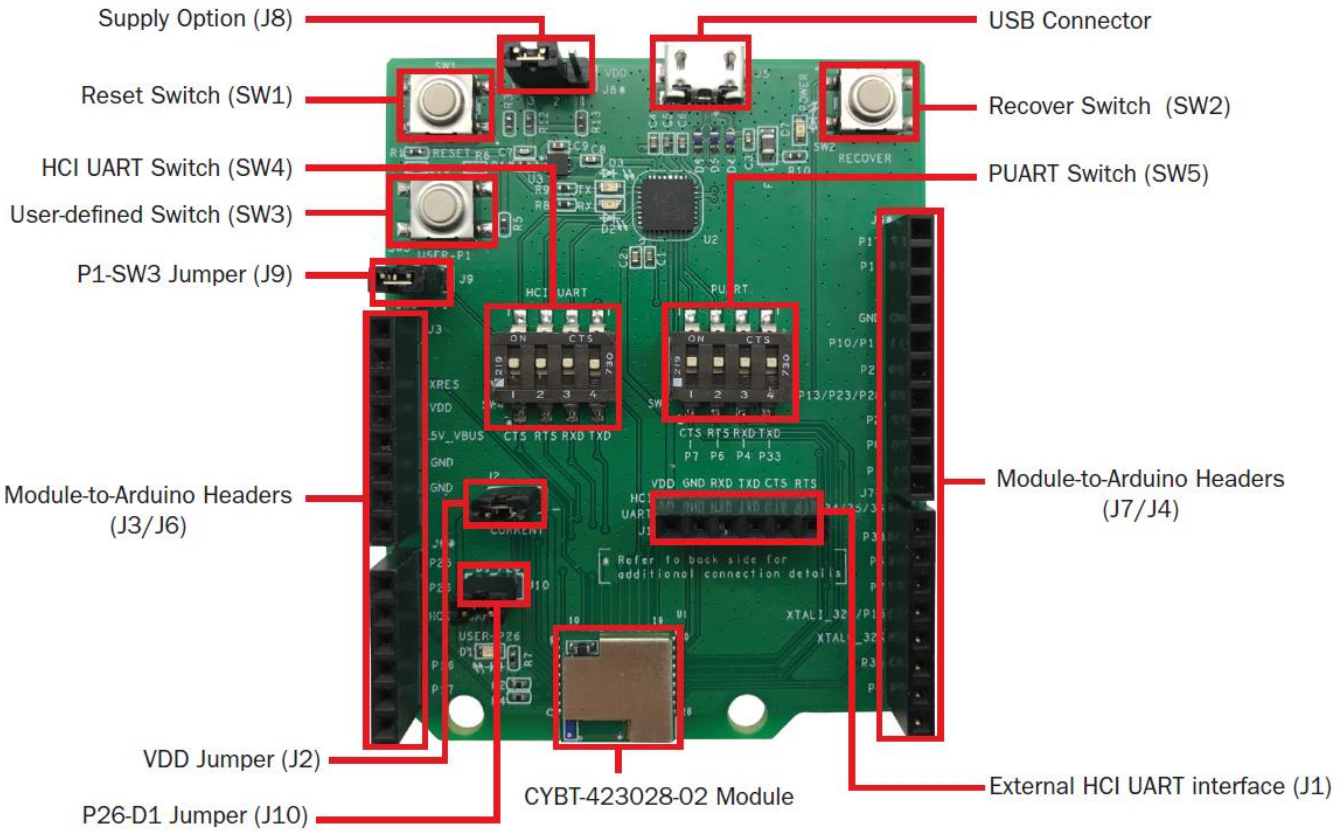
Refer to Section 7.5.3 for general information on how to program or reprogram EZ-BT WICED Modules. You may also refer to KBA223428 for online information.

To program an EZ-BT WICED module in your own manufacturing line, you must ensure that both the Recover and Reset connections on the EZ-BT module are exposed in order to put the module into programming/recovery mode.

C.3 CYBT-423028-EVAL

CYBT-423028-EVAL is the evaluation board for the CYBT-423028-02 WICED module. Figure 70 shows the CYBT-423028-EVAL board and calls out the main components and connections available on the board.

Figure 70. CYBT-423028-EVAL Evaluation Board



Note: Connections not called out on J3, J4, J6, and J7 Arduino compatible headers are NC (No Connect), where no physical connection is present between the CYBT-423028-02 EZ-BT WICED Module and the associated headers.

The CYBT-423028-EVAL includes the following elements:

C.3.1 Active Devices

- EZ-BT WICED Module: The EZ-BT WICED Module is mounted to the evaluation board as shown in Figure 70.
- USB-to-UART bridge device: A USB-to-UART bridge device is provided on the evaluation board to translate USB communication to UART communication (and vice-versa). For example, when HCI UART communication is configured to “On” on SW4, the USB traffic from the host PC is connected to the HCI UART connections of the CYBT-423028-02 module.

C.3.2 Connectors and Headers

- USB receptacle: The USB connection on the CYBT-423028-EVAL board provides power to the evaluation board and also provides communication to the board via USB, which is translated to UART communication and routed to either the HCI UART or PUART connection on the EZ-BT WICED Module depending on the configuration of SW4 and SW5 on CYBT-423028-EVAL.
- Power Supply Option Header (J8): J8 allows for configuration of two different power supplies to the EZ-BT WICED Module. All power is sourced from the USB connection, as mentioned above, but the power supply directly inputted to the CYBT-423028-02 module can be configured to either 1.8 V or 3.3 V using the provided three-pin header.

The power supply input is configured by shorting two neighboring header positions on J8. Table 20 details the available power supply options and the associated header position connections required.

Table 20. J8 Header Power Supply Connection Options

J8 Jumper Configuration	VDD Voltage Level
Short 1 & 2	3.3 V Supply
Short 2 & 3	1.8 V Supply
All Other Configurations	Not Allowed

- Arduino-compatible base headers – headers J3/J4/J6/J7: CYBT-423028-EVAL provides Arduino-compatible base headers that can be used for Arduino shield connections. Associated signals that are routed out to these headers are noted on the PCB silk screen on both the top and bottom side of the evaluation board.
- HCI UART direct connection header (J1): The J1 header provides all HCI UART communication lines to the user. This allows for connection to the EZ-BT WICED Module without having to connect through USB. This can be used to connect a host controller evaluation board directly to the EZ-BT WICED Module HCI UART connection. HCI UART connections are not brought out to the Arduino-compatible headers. The J1 direct HCI UART connection bypasses the SW4 (HCI UART to USB-to-UART bridge to Host PC) configuration. It is recommended that SW4 elements 1 ~ 4 are placed in the OFF position if the J1 direct HCI UART connection is used.
- Power consumption measurement header (J2): J2 is provided to allow for easy power consumption measurement reading with a multimeter or current measurement probe.
- User Element Disconnection Headers (J9 and J10): J9 (USER SW3) and J10 (USER D1 - LED) are provided to allow for disconnection of these elements from the CYBT-423028-02 module. In both cases, the GPIO that is routed to these USER elements is also routed to the Arduino-compatible headers. Disconnecting the elements (SW3 and/or D1) may be required if you plan to use either of these GPIOs through Arduino-compatible headers.

C.3.3 Switches and LEDs

- SW4 (HCI UART configuration switch): SW4 controls the connection of the USB data traffic to the HCI UART connections of the module. SW4 is a four-element switch (each element corresponding to the 4-wire UART signals – TX, RX, CTS, RTS); used to connect the HCI UART signals on CYBT-423028-EVAL to the host PC USB interface. To connect HCI UART signals from CYBT-423028-02 to the host PC, simply put each element of SW4 in the “On” position. To disconnect HCI UART communication from the CYBT-423028-02 module and the host PC, simply set each of the SW4 elements to the Off position. There is no High-Z configuration available on the SW4 elements; only On or Off positions. Flow control is supported in hardware on the HCI UART connection.
- SW5 (Peripheral UART (PUART) configuration switch): SW5 controls the connection of the PUART signals of the CYBT-423028-02 module to the host PC connection (via the USB/UART Bridge). SW5 is a four-element switch; to connect PUART signals on CYBT-423028-EVAL to the host PC, simply put each element of SW4 in the “On” position. To disconnect PUART communication from the CYBT-423028-02 module and the host PC, simply set each of the SW5 elements to the Off position. There is no High-Z configuration available on the SW5 elements; only On or Off positions.

  Note: PUART on the CYBT-423028-02 module does not support flow control in hardware. Flow control logic is required to be developed in the WICED SDK if this is required. In any case, if flow control is developed and enabled, only the GPIOs that are noted as supporting RTS and/or CTS can be used for this functionality.

  Note: PUART connections from the CYBT-423028-02 module to the Arduino headers are not gated by SW5. These connections are always present.

Notes on HCI UART and PUART configurations:

- It is acceptable to have SW4 and SW5 elements all set to the “On” position at the same time (i.e., HCI UART and PUART both connected to the CYBT-423028-02 module). PUART and HCI UART connections are handled by two different hardware UART blocks, and the connection to the CYBT-423028-02 module from the USB-to-UART bridge enumerates as two different COM ports on the host PC (not a shared connection/bus).
- Both HCI UART and PUART configurations can be used for UART communication within your application.
- HCI UART (Positions 1 ~ 4 of SW4 = **ON**) is the UART configuration generally used for interfacing to the WICED Studio SDK, with the goal of downloading a compiled image to the EZ-BT WICED Module. The HCI UART connection can also be used for general UART communication in Application mode (executing a downloaded

image from on-chip flash). If 4-wire UART with flow control is required, it is recommended to use HCI UART as the UART connection for your application.

- PUART (Positions 1 ~ 4 of SW5 = **ON**) is the configuration to use when desiring to have the PUART connected to the host PC. This PUART interface to the host PC can be used for terminal communication (print statements).

  Note: PUART communication signals routed to the Arduino headers are not gated by SW5 and can be used to communicate with any peripheral device over UART.

  Note: The PUART interface cannot be used to download a new binary image to the CYBT-423028-02 module.

C.3.4 Additional Switches and LEDs

- SW1 (RESET switch): SW1 is a tactile switch that is connected directly to the XRES connection of the CYBT-423028-02 module. Activating this switch will reset the EZ-BT WICED Module. This switch is also used in the process of (re-)programming the CYBT-423028-02 module on the evaluation board.
- SW2 (RECOVER switch): SW2 is provided to recover a module that has the standard programming bootloader and/or application image corrupted or erased. Recovery mode is activated by holding this button down while pressing and releasing SW1 (RESET). Recovery mode is used prior to programming the module with a new binary image.
- SW3 (USER switch): SW3 is provided as an element that the user can configure as required. The SW3 element is connected to P1 on the CYBT-423028-02 module (solder pad 17). Header J9 is provided on the CYBT-423028-EVAL board to disconnect P1 from the USER switch to allow this connection to be used on the Arduino Headers without SW3 connected.
- D1 USER LED is provided for user-configured behavior as required. The D1 LED is connected to the module P26 connection (solder pad 14). Header J10 is provided on the CYBT-423028-EVAL board to disconnect P26 from the USER LED to allow this connection to be used on the Arduino Headers without D1 connected.
- D2 and D3 LEDs are provided to display programming or USB activity while in progress.
- D7 LED is provided to show that power is provided from the host PC.

C.3.5 Programming or Reprogramming the CYBT-423028-EVAL Board

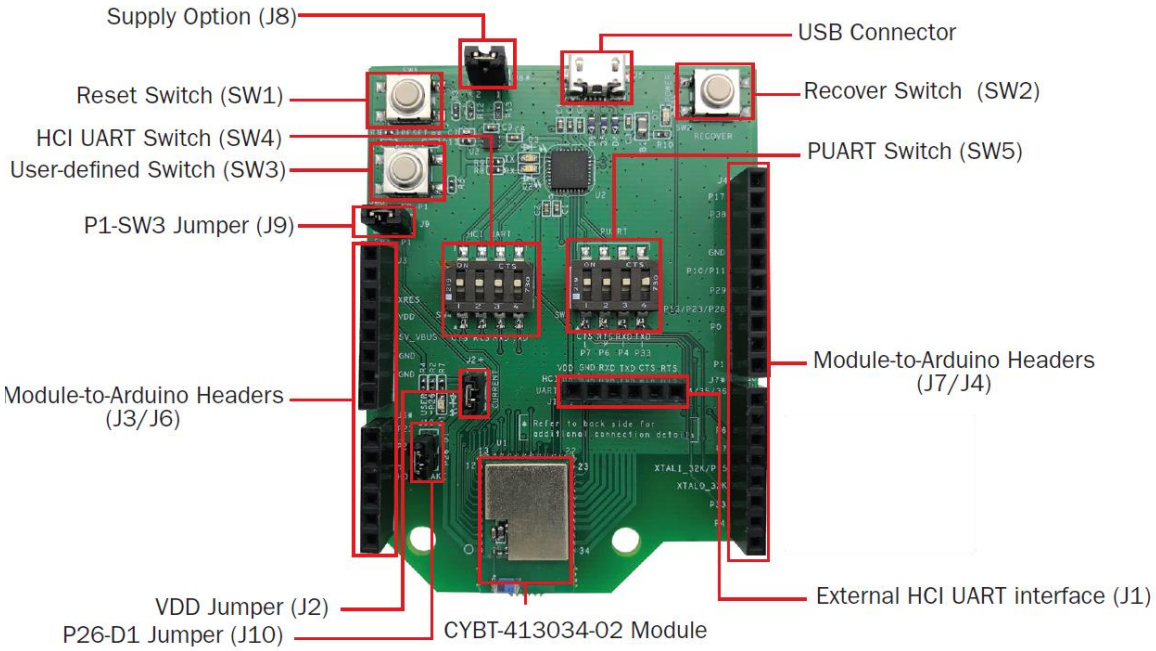
Refer to Section 7.5.3 for general information on how to program or reprogram EZ-BT WICED Modules. You may also refer to KBA223428 for online information.

To program an EZ-BT WICED module in your own manufacturing line, you must ensure that both the Recover and Reset connections on the EZ-BT module are exposed in order to put the module into programming/recovery mode.

C.4 CYBT-413034-EVAL

The CYBT-413034-EVAL is the evaluation board for the CYBT-413034-02 WICED module. Figure 71 shows the CYBT-413034-EVAL board and calls out the main components and connections available on the board.

Figure 71. CYBT-413034-EVAL Evaluation Board



Note: Connections not called out on J3, J4, J6, and J7 Arduino compatible headers are NC (No Connect), where no physical connection is present between the CYBT-413034-02 EZ-BT WICED Module and the associated headers.

CYBT-413034-EVAL includes the following elements:

C.4.1 Active Devices

- EZ-BT WICED Module: The EZ-BT WICED Module is mounted to the evaluation board as shown in Figure 71.
- USB-to-UART bridge device: A USB-to-UART bridge device is provided on the evaluation board to translate USB communication to UART communication (and vice-versa). For example, when HCI UART communication is configured to “On” on SW4, the USB traffic from the Host PC is connected to the HCI UART connections of the CYBT-413034-02 module.

C.4.2 Connectors and Headers

- USB receptacle: The USB connection on the CYBT-413034-EVAL board provides power to the evaluation board and also provides communication to the board via USB, which is translated to UART communication and routed to either the HCI UART or PUART connection on the EZ-BT WICED Module depending on the configuration of SW4 and SW5 on CYBT-413034-EVAL.
- Power Supply Option Header (J8): J8 allows for configuration of two different power supplies to the EZ-BT WICED Module. All power is sourced from the USB connection, as mentioned above, but the power supply directly inputted to the CYBT-413034-02 module can be configured to either 1.8 V or 3.3 V using the provided two-pin header. The power supply input is configured by shorting the header positions or leaving them open on J8. Table 21 details the available power supply options and the associated header position connections required.

Table 21. J8 Header Power Supply Connection Options

J8 Jumper Configuration	VDD Voltage Level
Short 1 & 2	3.3 V Supply
Leave 1 & 2 Open	1.8 V Supply

- Arduino-compatible base headers – headers J3/J4/J6/J7: The CYBT-413034-EVAL provides Arduino-compatible base headers that can be used for Arduino shield connections. The associated signals that are routed out to these headers are noted on the PCB silk screen on both the top and bottom side of the evaluation board.
- HCI UART direct connection header (J1): The J1 header provides all HCI UART communication lines to the user. This allows for connection to the EZ-BT WICED Module without having to connect through USB. This can be used to connect a host controller evaluation board directly to the EZ-BT WICED Module HCI UART connection. HCI UART connections are not brought out to the Arduino-compatible headers. The J1 direct HCI UART connection bypasses the SW4 (HCI UART to USB-to-UART bridge to Host PC) configuration. It is recommended that SW4 elements 1 ~ 4 are placed in the OFF position if the J1 direct HCI UART connection is used.
- Power consumption measurement header (J2): J2 is provided to allow for easy power consumption measurement reading with a multimeter or current measurement probe.
- User Element Disconnection Headers (J9 and J10): J9 (USER SW3) and J10 (USER D1 - LED) are provided to allow for disconnection of these elements from the CYBT-413034-02 module. In both cases, the GPIO that is routed to these USER elements is also routed to the Arduino-compatible headers. Disconnecting the elements (SW3 and/or D1) may be required if you plan to use either of these GPIOs through the Arduino-compatible headers.

C.4.3 Switches and LEDs

- SW4 (HCI UART configuration switch): SW4 controls the connection of USB data traffic to the HCI UART connections of the module. SW4 is a four-element switch (each element corresponding to the 4-wire UART signals – TX, RX, CTS, RTS); used to connect the HCI UART signals on CYBT-413034-EVAL to the host PC USB interface. To connect HCI UART signals from CYBT-413034-02 to the host PC, simply put each element of SW4 in the “On” position. To disconnect HCI UART communication from the CYBT-413034-02 module and the host PC, simply set each of the SW4 elements to the Off position. There is no High-Z configuration available on the SW4 elements; only On or Off positions. Flow control is supported in Hardware on the HCI UART connection.
- SW5 (Peripheral UART (PUART) configuration switch): SW5 controls the connection of the PUART signals of the CYBT-413034-02 module to the Host PC connection (via the USB/UART Bridge). SW5 is a four-element switch; to connect PUART signals on CYBT-413034-EVAL to the host PC, simply put each element of SW4 in the “On” position. To disconnect PUART communication from the CYBT-413034-02 module and the host PC, simply set each of the SW5 elements to the Off position. There is no High-Z configuration available on the SW5 elements; only On or Off positions.

  Note: PUART on the CYBT-413034-02 module does not support flow control in hardware. Flow control logic is required to be developed in the WICED SDK if this is required. In any case, if flow control is developed and enabled, only the GPIOs that are noted as supporting RTS and/or CTS can be used for this functionality.

  Note: PUART connections from the CYBT-413034-02 module to the Arduino headers are not gated by SW5. These connections are always present.

Notes on HCI UART and PUART configurations:

- It is acceptable to have SW4 and SW5 elements all set to the “On” position at the same time (i.e., HCI UART and PUART both connected to the CYBT-413034-02 module). PUART and HCI UART connections are handled by two different hardware UART blocks, and the connection to the CYBT-413034-02 module from the USB-to-UART bridge enumerates as two different COM ports on the host PC (not a shared connection/bus).
- Both HCI UART and PUART configurations can be used for UART communication within your application.
- HCI UART (Positions 1 ~ 4 of SW4 = **ON**) is the UART configuration generally used for interfacing to the WICED Studio SDK, with the goal of downloading a compiled image to the EZ-BT WICED Module. The HCI UART connection can also be used for general UART communication in Application mode (executing a downloaded image from on-chip flash). If 4-wire UART with flow control is required, it is recommended to use HCI UART as the UART connection for your application.
- PUART (Positions 1 ~ 4 of SW5 = **ON**) is the configuration to use when desiring to have the PUART connected to the host PC. This PUART interface to the host PC can be used for terminal communication (print statements).

  Note: PUART communication signals routed to the Arduino headers are not gated by SW5 and can be used to communicate with any peripheral device over UART.

  Note: The PUART interface cannot be used to download a new binary image to the CYBT-413034-02 module.

C.4.4 Additional Switches and LEDs

- SW1 (RESET switch): SW1 is a tactile switch that is connected directly to the XRES connection of the CYBT-413034-02 module. Activating this switch will reset the EZ-BT WICED Module. This switch is also used in the process of (re-)programming the CYBT-413034-02 module on the evaluation board.
- SW2 (RECOVER switch): SW2 is provided to recover a module that has the standard programming bootloader and/or application image corrupted or erased. Recovery mode is activated by holding this button down while pressing and releasing SW1 (RESET). Recovery mode is used prior to programming the module with a new binary image.
- SW3 (USER switch): SW3 is provided as an element that the user can configure as required. The SW3 element is connected to P1 on the CYBT-413034-02 module (solder pad 8). Header J9 is provided on the CYBT-413034-EVAL board to disconnect P1 from the USER switch to allow this connection to be used on the Arduino Headers without SW3 connected.
- D1 USER LED is provided for user-configured behavior as required. The D1 LED is connected to the module P26 connection (solder pad 6). Header J10 is provided on the CYBT-413034-EVAL board to disconnect P26 from the USER LED to allow this connection to be used on the Arduino Headers without D1 connected.
- D2 and D3 LEDs are provided to display programming or USB activity while in progress.
- D7 LED is provided to show that power is provided from the host PC.

C.4.5 Programming or Reprogramming the CYBT-413034-EVAL Board

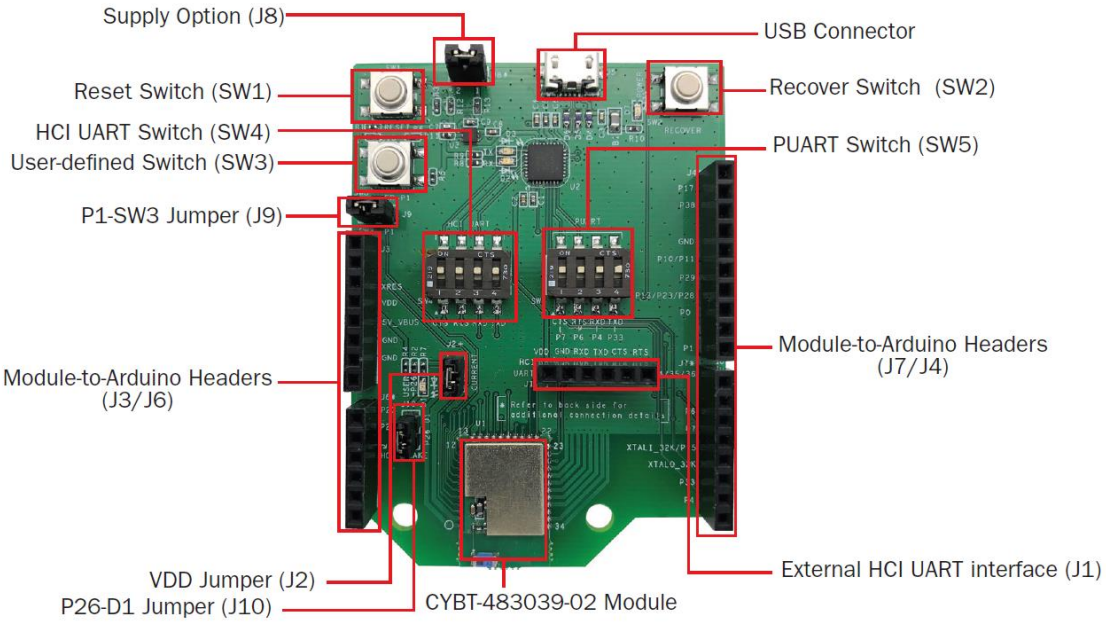
Refer to Section 7.5.3 for general information on how to program or reprogram EZ-BT WICED Modules. You may also refer to KBA223428 for online information.

To program an EZ-BT WICED module in your own manufacturing line, you must ensure that both the Recover and Reset connections on the EZ-BT module are exposed in order to put the module into programming/recovery mode.

C.5 CYBT-483039-EVAL

CYBT-483039-EVAL is the evaluation board for the CYBT-483039-02 WICED module. Figure 72 shows the CYBT-483039-EVAL board and calls out the main components and connections available on the board.

Figure 72. CYBT-483039-EVAL Evaluation Board



Note: Connections not called out on J3, J4, J6, and J7 Arduino compatible headers are NC (No Connect), where no physical connection is present between the CYBT-483039-02 EZ-BT WICED Module and the associated headers.

The CYBT-483039-EVAL includes the following elements:

C.5.1 Active Devices

- EZ-BT WICED Module: The EZ-BT WICED Module is mounted to the evaluation board as shown in Figure 72.
- USB-to-UART bridge device: A USB-to-UART bridge device is provided on the evaluation board to translate USB communication to UART communication (and vice-versa). For example, when HCI UART communication is configured to “On” on SW4, the USB traffic from the Host PC is connected to the HCI UART connections of the CYBT-483039-02 module.

C.5.2 Connectors and Headers

- USB receptacle: The USB connection on the CYBT-483039-EVAL board provides power to the evaluation board and also provides communication to the board via USB, which is translated to UART communication and routed to either the HCI UART or PUART connection on the EZ-BT WICED Module depending on the configuration of SW4 and SW5 on CYBT-483039-EVAL.
- Power Supply Option Header (J8): J8 allows for configuration of two different power supplies to the EZ-BT WICED Module. All power is sourced from the USB connection, as mentioned above, but the power supply directly inputted to the CYBT-483039-02 module can be configured to either 2.0 V or 3.3 V using the provided two-pin header. The power supply input is configured by shorting the header positions or leaving them open on J8. Table 22 details the available power supply options and the associated header position connections required.

Table 22. J8 Header Power Supply Connection Options

J8 Jumper Configuration	VDD Voltage Level
Short 1 & 2	3.3 V Supply
Leave 1 & 2 Open	2.0 V Supply

- Arduino-compatible base headers – headers J3/J4/J6/J7: The CYBT-483039-EVAL provides Arduino-compatible base headers that can be used for Arduino shield connections. Associated signals that are routed out to these headers are noted on the PCB silk screen on both the top and bottom side of the evaluation board.
- HCI UART direct connection header (J1): The J1 header provides all HCI UART communication lines to the user. This allows for connection to the EZ-BT WICED Module without having to connect through USB. This can be used to connect a host controller evaluation board directly to the EZ-BT WICED Module HCI UART connection. HCI UART connections are not brought out to the Arduino-compatible headers. The J1 direct HCI UART connection bypasses the SW4 (HCI UART to USB-to-UART bridge to Host PC) configuration. It is recommended that SW4 elements 1 ~ 4 are placed in the OFF position if the J1 direct HCI UART connection is used.
- Power consumption measurement header (J2): J2 is provided to allow for easy power consumption measurement reading with a multi-meter or current measurement probe.
- User Element Disconnection Headers (J9 and J10): J9 (USER SW3) and J10 (USER D1 - LED) are provided to allow for disconnection of these elements from the CYBT-483039-02 module. In both cases, the GPIO that is routed to these USER elements is also routed to the Arduino-compatible headers. Disconnecting the elements (SW3 and/or D1) may be required if you plan to use either of these GPIOs through the Arduino-compatible headers.

C.5.3 Switches and LEDs

- SW4 (HCI UART configuration switch): SW4 controls the connection of USB data traffic to the HCI UART connections of the module. SW4 is a four-element switch (each element corresponding to the 4-wire UART signals – TX, RX, CTS, RTS); used to connect the HCI UART signals on CYBT-483039-EVAL to the host PC USB interface. To connect HCI UART signals from CYBT-483039-02 to the host PC, simply put each element of SW4 in the “On” position. To disconnect HCI UART communication from the CYBT-483039-02 module and the host PC, simply set each of the SW4 elements to the Off position. There is no High-Z configuration available on the SW4 elements; only On or Off positions. Flow control is supported in Hardware on the HCI UART connection.
- SW5 (Peripheral UART (PUART) configuration switch): SW5 controls the connection of the PUART signals of the CYBT-483039-02 module to the Host PC connection (via the USB/UART Bridge). SW5 is a four-element switch; to connect PUART signals on CYBT-483039-EVAL to the host PC, simply put each element of SW4 in the “On” position. To disconnect PUART communication from the CYBT-483039-02 module and the host PC, simply set each of the SW5 elements to the Off position. There is no High-Z configuration available on the SW5 elements; only On or Off positions.

  Note: PUART on the CYBT-483039-02 module does not support flow control in hardware. Flow control logic is required to be developed in the WICED SDK if this is required. In any case, if flow control is developed and enabled, only the GPIOs that are noted as supporting RTS and/or CTS can be used for this functionality.

  Note: PUART connections from the CYBT-483039-02 module to the Arduino headers are not gated by SW5. These connections are always present.

Notes on HCI UART and PUART configurations:

- It is acceptable to have SW4 and SW5 elements all set to the “On” position at the same time (i.e., HCI UART and PUART both connected to the CYBT-483039-02 module). The PUART and HCI UART connections are handled by two different hardware UART blocks, and the connection to the CYBT-483039-02 module from the USB-to-UART bridge enumerates as two different COM ports on the host PC (not a shared connection/bus).
- Both HCI UART and PUART configurations can be used for UART communication within your application.
- HCI UART (Positions 1 ~ 4 of SW4 = **ON**) is the UART configuration generally used for interfacing to the WICED Studio SDK, with the goal of downloading a compiled image to the EZ-BT WICED Module. The HCI UART connection can also be used for general UART communication in Application mode (executing a downloaded image from on-chip flash). If 4-wire UART with flow control is required, it is recommended to use HCI UART as the UART connection for your application.
- PUART (Positions 1 ~ 4 of SW5 = **ON**) is the configuration to use when desiring to have the PUART connected to the host PC. This PUART interface to the host PC can be used for terminal communication (print statements).

  Note: PUART communication signals routed to the Arduino headers are not gated by SW5 and can be used to communicate with any peripheral device over UART.

  Note: The PUART interface cannot be used to download a new binary image to the CYBT-483039-02 module.

C.5.4 Additional Switches and LEDs

- SW1 (RESET switch): SW1 is a tactile switch that is connected directly to the XRES connection of the CYBT-483039-02 module. Activating this switch will reset the EZ-BT WICED Module. This switch is also used in the process of (re-)programming the CYBT-483039-02 module on the evaluation board.
- SW2 (RECOVER switch): SW2 is provided to recover a module that has the standard programming bootloader and/or application image corrupted or erased. Recovery mode is activated by holding this button down while pressing and releasing SW1 (RESET). Recovery mode is used prior to programming the module with a new binary image.
- SW3 (USER switch): SW3 is provided as an element that the user can configure as required. The SW3 element is connected to P1 on the CYBT-483039-02 module (solder pad 9). Header J9 is provided on the CYBT-483039-EVAL board to disconnect P1 from the USER switch to allow this connection to be used on the Arduino Headers without SW3 connected.
- D1 USER LED is provided for user-configured behavior as required. The D1 LED is connected to the module P26 connection (solder pad 6). Header J10 is provided on the CYBT-483039-EVAL board to disconnect P26 from the USER LED to allow this connection to be used on the Arduino Headers without D1 connected.
- D2 and D3 LEDs are provided to display programming or USB activity while in progress.
- D7 LED is provided to show that power is provided from the host PC.

C.5.5 Programming or Reprogramming the CYBT-483039-EVAL Board

Refer to Section 7.5.3 for general information on how to program or reprogram EZ-BT WICED Modules. You may also refer to KBA223428 for online information.

To program an EZ-BT WICED module in your own manufacturing line, you must ensure that both the Recover and Reset connections on the EZ-BT module are exposed in order to put the module into programming/recovery mode.

Appendix D. Code Examples

The WICED Studio SDK comes with a set of sample application projects that demonstrate a variety of Bluetooth BR/EDR, BLE and peripheral functionality. These projects are available in the Project Explorer inside the IDE, as Figure 73 shows.

Example projects can speed up your design process by starting you off with a reference design, instead of a blank page. Code examples include comments describing their functionality and basic demonstration instructions near the top of the main **.c** source file for each project.

Each example also has a dedicated make target available to provide an easy way to build and download onto an EZ-BT Module. These make targets should be renamed before use in order to provide compatibility with the CYBT-343026-EVAL board. Simply change the "CYW920706WCDEVAL" portion of the target name to "CYBT_343026_EVAL" instead. For example:

Before: `demo.hello_sensor-CYW920706WCDEVAL download`

After: `demo.hello_sensor-CYBT_343026_EVAL download`

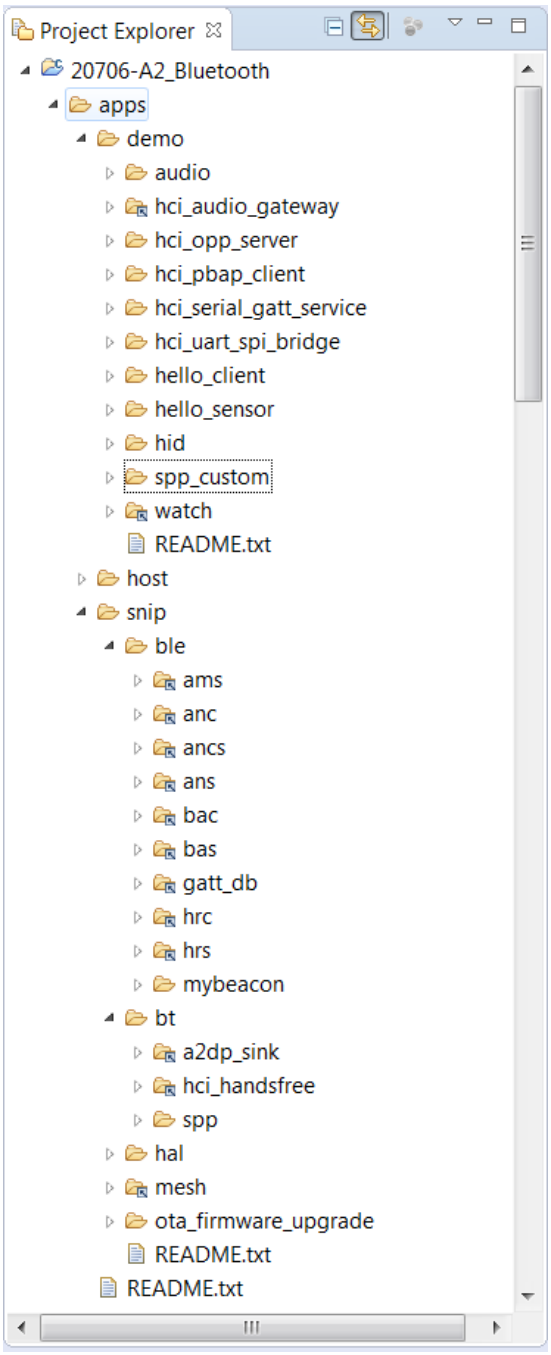
Some examples assume peripheral devices that are not present or are routed differently on the CYBT-343026-EVAL board compared to CYW920706WCDEVAL evaluation board. The table below defines all built-in peripherals and their routed pin connections.

Table 23. CYBT-343026-EVAL Pin Assignments

Pin	Function	Macro
P24	Button (active HIGH)	WICED_P00
P14	LED (active LOW)	WICED_P28
P2	PUART RXD	WICED_P04
P0	PUART TXD	WICED_P31
P1	PUART RTS	WICED_P30
P3	PUART CTS	WICED_P03

These pins are defined in the custom platform files (refer KBA221025 for CYBT-343026-EVAL), accessible via compiler macros to improve code portability. However, some projects may also use pin numbers directly. As you explore the examples that come with the SDK, be sure to double-check any instances of GPIO pin usage if you encounter compiler errors or missing peripheral functionality.

Figure 73. Code Examples in WICED Studio



Appendix E. Example Project *spp.c*

```
#include "sparcommon.h"
#include "wiced.h"
#include "wiced_gki.h"
#include "wiced_bt_dev.h"
#include "wiced_bt_sdp.h"
#include "wiced_bt_ble.h"
#include "wiced_bt_uuid.h"
#include "wiced_hal_nvram.h"
#include "wiced_bt_app_hal_common.h"
#include "wiced_bt_trace.h"
#include "wiced_bt_cfg.h"
#include "wiced_bt_spp.h"
#include "hello_sensor.h"

#include "wiced_timer.h"
#include "wiced_transport.h"
#include "wiced_hal_platform.h"
#include "wiced_memory.h"

#define HCI_TRACE_OVER_TRANSPORT            1   // If defined HCI traces are send over transport/WICED HCI interface
#define SEND_DATA_ON_INTERRUPT              1   // If defined application button causes 1Meg of data to be sent
#define SEND_DATA_ON_TIMEOUT                1   // If defined application sends 4 bytes of data every second
//#define LOOPBACK_DATA                     1   // If defined application loops back received data

#define WICED_EIR_BUF_MAX_SIZE              264
#define SPP_NVRAM_ID                        0x50

/* Max TX packet to be sent over SPP */
#define MAX_TX_BUFFER                       1017
#define TRANS_MAX_BUFFERS                   10
#define TRANS_UART_BUFFER_SIZE              1024
#define SPP_MAX_PAYLOAD                     1007

#if SEND_DATA_ON_INTERRUPT
#include "wiced_hal_gpio.h"
#include "wiced_hal_platform.h"


#define APP_TOTAL_DATA_TO_SEND             1000000
#define BUTTON_GPIO                         WICED_P30
int     app_send_offset = 0;
uint8_t app_send_buffer[SPP_MAX_PAYLOAD];
#endif

#ifdef SEND_DATA_ON_TIMEOUT
void app_timeout(uint32_t count);
#endif

/******************************************************************************
**  Structures
******************************************************************************/

#define SPP_RFCOMM_SCN                  2

static void         spp_connection_up_callback(uint16_t handle, uint8_t* bda);
static void         spp_connection_down_callback(uint16_t handle);
static wiced_bool_t spp_rx_data_callback(uint16_t handle, uint8_t* p_data, uint32_t data_len);

wiced_bt_spp_reg_t spp_reg =
{
    SPP_RFCOMM_SCN,                     /* RFCOMM service channel number for SPP connection */
    MAX_TX_BUFFER,                      /* RFCOMM MTU for SPP connection */
    spp_connection_up_callback,         /* SPP connection established */
    NULL,                               /* SPP connection establishment failed, not used because this app never
initiates connection */
    NULL,                               /* SPP service not found, not used because this app never initiates
connection */
    spp_connection_down_callback,       /* SPP connection disconnected */
    spp_rx_data_callback,               /* Data packet received */
};

wiced_transport_buffer_pool_t*  host_trans_pool;
uint16_t                        spp_handle;
wiced_timer_t                   app_tx_timer;

const uint8_t app_sdp_db[] = // Define SDP database
```

```
{
    SDP_ATTR_SEQUENCE_2(142),
    SDP_ATTR_SEQUENCE_1(69),                                                // 2 bytes
        SDP_ATTR_RECORD_HANDLE(0x10003),                                    // 8 bytes
        SDP_ATTR_CLASS_ID(UUID_SERVCLASS_SERIAL_PORT),                      // 8
        SDP_ATTR_RFCOMM_PROTOCOL_DESC_LIST( SPP_RFCOMM_SCN ),               // 17 bytes
        SDP_ATTR_BROWSE_LIST,                                               // 8
        SDP_ATTR_PROFILE_DESC_LIST(UUID_SERVCLASS_SERIAL_PORT, 0x0102),     // 13 byte
        SDP_ATTR_SERVICE_NAME(10),                                          // 15
        'S', 'P', 'P', ' ', 'S', 'E', 'R', 'V', 'E', 'R',

    // Device ID service
    SDP_ATTR_SEQUENCE_1(69),                                                // 2 bytes, length of the record
        SDP_ATTR_RECORD_HANDLE(0x10002),                                    // 8 byte
        SDP_ATTR_CLASS_ID(UUID_SERVCLASS_PNP_INFORMATION),                  // 8
        SDP_ATTR_PROTOCOL_DESC_LIST(1),                                     // 18
        SDP_ATTR_UINT2(ATTR_ID_SPECIFICATION_ID, 0x103),                    // 6
        SDP_ATTR_UINT2(ATTR_ID_VENDOR_ID, 0x0f),                            // 6
        SDP_ATTR_UINT2(ATTR_ID_PRODUCT_ID, 0x0401),                         // 6
        SDP_ATTR_UINT2(ATTR_ID_PRODUCT_VERSION, 0x0001),                    // 6
        SDP_ATTR_BOOLEAN(ATTR_ID_PRIMARY_RECORD, 0x01),                     // 5
        SDP_ATTR_UINT2(ATTR_ID_VENDOR_ID_SOURCE, DI_VENDOR_ID_SOURCE_BTSIG) // 6
};

// Length of the SDP database
const uint16_t app_sdp_db_len = sizeof(app_sdp_db);

uint8_t pincode[4] = { 0x30, 0x30, 0x30, 0x30 };

extern const wiced_bt_cfg_settings_t wiced_bt_cfg_settings;
extern const wiced_bt_cfg_buf_pool_t wiced_bt_cfg_buf_pools[WICED_BT_CFG_NUM_BUF_POOLS];

#if defined WICED_BT_TRACE_ENABLE || defined HCI_TRACE_OVER_TRANSPORT
const wiced_transport_cfg_t transport_cfg =
{
    WICED_TRANSPORT_UART,
    { WICED_TRANSPORT_UART_HCI_MODE, HCI_UART_DEFAULT_BAUD },
    { TRANS_UART_BUFFER_SIZE, 1},
    NULL,
    NULL,
    NULL
};
#endif

/******************************************************************
 * Function Prototypes
 ******************************************************************/
static wiced_bt_dev_status_t app_management_callback (wiced_bt_management_evt_t event, wiced_bt_management_evt_data_t
*p_event_data);
static void                  app_write_eir(void);
static int                   app_write_nvram(int nvram_id, int data_len, void *p_data);
static int                   app_read_nvram(int nvram_id, void *p_data, int data_len);

#if SEND_DATA_ON_INTERRUPT
static void                  app_tx_ack_timeout(uint32_t param);
static void                  app_interrupt_handler(void *data, uint8_t port_pin);
#endif
#ifdef HCI_TRACE_OVER_TRANSPORT
static void                  app_trace_callback(wiced_bt_hci_trace_type_t type, uint16_t length, uint8_t* p_data);
#endif

//extern wiced_bt_rfcomm_result_t wiced_bt_rfcomm_init(uint32_t buffer_size, uint32_t buffer_cnt);

/******************************************************************
 * Function Definitions
 ******************************************************************/

/*
 * Entry point to the application. Set device configuration and start BT
 * stack initialization.  The actual application initialization will happen
 * when stack reports that BT device is ready
 */
APPLICATION_START()
{
    wiced_result_t result;

#if defined WICED_BT_TRACE_ENABLE || defined HCI_TRACE_OVER_TRANSPORT
    wiced_transport_init(&transport_cfg);
```

```
    // create special pool for sending data to the MCU
    host_trans_pool = wiced_transport_create_buffer_pool(TRANS_UART_BUFFER_SIZE, TRANS_MAX_BUFFERS);

    // Set the debug uart as WICED_ROUTE_DEBUG_NONE to get rid of prints
    // wiced_set_debug_uart(WICED_ROUTE_DEBUG_NONE);

    // Set to PUART to see traces on peripheral uart(puart)
    wiced_set_debug_uart( WICED_ROUTE_DEBUG_TO_PUART );
    wiced_hal_puart_select_uart_pads( WICED_PUART_RXD, WICED_PUART_TXD, 0, 0);

    // Set to HCI to see traces on HCI uart - default if no call to wiced_set_debug_uart()
    // wiced_set_debug_uart( WICED_ROUTE_DEBUG_TO_HCI_UART );

    // Use WICED_ROUTE_DEBUG_TO_WICED_UART to send formatted debug strings over the WICED
    // HCI debug interface to be parsed by ClientControl/BtSpy.
    // Note: WICED HCI must be configured to use this - see wiced_trasnport_init(), must
    // be called with wiced_transport_cfg_t.wiced_tranport_data_handler_t callback present
    // wiced_set_debug_uart(WICED_ROUTE_DEBUG_TO_WICED_UART);
#endif

    WICED_BT_TRACE("APP Start\n");

    /* Initialize Stack and Register Management Callback */
    // Register call back and configuration with stack
    wiced_bt_stack_init(app_management_callback, &wiced_bt_cfg_settings, wiced_bt_cfg_buf_pools);
}

/*
 * AMS application initialization is executed after BT stack initialization is completed.
 */
void application_init(void)
{
    wiced_bt_gatt_status_t gatt_status;
    wiced_result_t         result;

    /* Initialize wiced app */
    wiced_bt_app_init();

#if SEND_DATA_ON_INTERRUPT
    /* Configure the button available on the platform */
    wiced_hal_gpio_configure_pin( WICED_GPIO_BUTTON, WICED_GPIO_BUTTON_SETTINGS( GPIO_EN_INT_RISING_EDGE ),
WICED_GPIO_BUTTON_DEFAULT_STATE );
    wiced_hal_gpio_register_pin_for_interrupt(WICED_GPIO_BUTTON, app_interrupt_handler, NULL);

    // init timer that we will use for the rx data flow control.
    wiced_init_timer(&app_tx_timer, app_tx_ack_timeout, 0, WICED_MILLI_SECONDS_TIMER);
#endif

    app_write_eir();

    // Initialize RFCOMM.  We will not be using application buffer pool and will rely on the
    // stack pools configured in the wiced_bt_cfg.c
    wiced_bt_rfcomm_init(MAX_TX_BUFFER, 1);

    // Initialize SPP library
    wiced_bt_spp_startup(&spp_reg);

#ifdef HCI_TRACE_OVER_TRANSPORT
    // There is a virtual HCI interface between upper layers of the stack and
    // the controller portion of the chip with lower layers of the BT stack.
    // Register with the stack to receive all HCI commands, events and data.
    wiced_bt_dev_register_hci_trace(app_trace_callback);
#endif
    /* create SDP records */
    wiced_bt_sdp_db_init((uint8_t *)app_sdp_db, sizeof(app_sdp_db));

    // This application will always configure device connectable and discoverable
    wiced_bt_dev_set_discoverability(BTM_GENERAL_DISCOVERABLE, 0x0012, 0x0800);
    wiced_bt_dev_set_connectability(BTM_CONNECTABLE, 0x0012, 0x0800);

#if SEND_DATA_ON_TIMEOUT
    /* Starting the app timers, seconds timer and the ms timer  */
    wiced_bt_app_start_timer(1, 0, app_timeout, NULL);
#endif
}

/*
 *  Management callback receives various notifications from the stack
 */
```

```
wiced_result_t app_management_callback(wiced_bt_management_evt_t event, wiced_bt_management_evt_data_t *p_event_data)
{
    wiced_result_t                      result = WICED_BT_SUCCESS;
    wiced_bt_dev_status_t               dev_status;
    wiced_bt_dev_pairing_info_t*        p_pairing_info;
    wiced_bt_dev_encryption_status_t*   p_encryption_status;
    int                                 bytes_written, bytes_read;
    wiced_bt_power_mgmt_notification_t* p_power_mgmt_notification;

    WICED_BT_TRACE("bt_management_callback 0x%02x\n", event);

    switch(event)
    {
    /* Bluetooth  stack enabled */
    case BTM_ENABLED_EVT:
        application_init();
        hello_sensor_application_init();
        WICED_BT_TRACE("Free mem:%d", wiced_memory_get_free_bytes());
        break;

    case BTM_DISABLED_EVT:
        break;

    case BTM_PIN_REQUEST_EVT:
        WICED_BT_TRACE("remote address= %B\n", p_event_data->pin_request.bd_addr);
        wiced_bt_dev_pin_code_reply(*p_event_data->pin_request.bd_addr,result/*WICED_BT_SUCCESS*/,4, &pincode[0]);
        break;

    case BTM_USER_CONFIRMATION_REQUEST_EVT:
        /* This application always confirms peer's attempt to pair */
        wiced_bt_dev_confirm_req_reply (WICED_BT_SUCCESS, p_event_data->user_confirmation_request.bd_addr);
        break;

    case BTM_PAIRING_IO_CAPABILITIES_BR_EDR_REQUEST_EVT:
        /* This application supports only Just Works pairing */
        WICED_BT_TRACE("BTM_PAIRING_IO_CAPABILITIES_REQUEST_EVT bda %B\n", p_event_data-
>pairing_io_capabilities_br_edr_request.bd_addr);
        p_event_data->pairing_io_capabilities_br_edr_request.local_io_cap    = BTM_IO_CAPABILITIES_NONE;
        p_event_data->pairing_io_capabilities_br_edr_request.auth_req        =
BTM_AUTH_SINGLE_PROFILE_GENERAL_BONDING_NO;
        break;

    case BTM_PAIRING_COMPLETE_EVT:
        p_pairing_info = &p_event_data->pairing_complete.pairing_complete_info;
        WICED_BT_TRACE("Pairing Complete: %d\n", p_pairing_info->br_edr.status);
        result = WICED_BT_USE_DEFAULT_SECURITY;
        break;

    case BTM_ENCRYPTION_STATUS_EVT:
        p_encryption_status = &p_event_data->encryption_status;
        WICED_BT_TRACE("Encryption Status Event: bd (%B) res %d\n", p_encryption_status->bd_addr,
p_encryption_status->result);
        break;

    case BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT:
        /* This application supports a single paired host, we can save keys under the same NVRAM ID overwriting
previous pairing if any */
        bt_write_nvram(SPP_NVRAM_ID, sizeof(wiced_bt_device_link_keys_t), &p_event_data-
>paired_device_link_keys_update);
        break;

    case  BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT:
        /* read existing key from the NVRAM  */
        if (bt_read_nvram(SPP_NVRAM_ID, &p_event_data->paired_device_link_keys_request,
sizeof(wiced_bt_device_link_keys_t)) != 0)
        {
            result = WICED_BT_SUCCESS;
        }
        else
        {
            result = WICED_BT_ERROR;
            WICED_BT_TRACE("Key retrieval failure\n");
        }
        break;

    case BTM_POWER_MANAGEMENT_STATUS_EVT:
        p_power_mgmt_notification = &p_event_data->power_mgmt_notification;
        WICED_BT_TRACE("Power mgmt status event: bd (%B) status:%d hci_status:%d\n", p_power_mgmt_notification-
>bd_addr, \
```

```
                p_power_mgmt_notification->status, p_power_mgmt_notification->hci_status);
        break;

    default:
        result = WICED_BT_USE_DEFAULT_SECURITY;
        break;
    }
    return result;
}


/*
 *  Prepare extended inquiry response data.  Current version publishes device name and 16bit
 *  SPP service.
 */
void app_write_eir(void)
{
    uint8_t *pBuf;
    uint8_t *p;
    uint8_t length;
    uint16_t eir_length;

    pBuf = (uint8_t *)wiced_bt_get_buffer(WICED_EIR_BUF_MAX_SIZE);
    WICED_BT_TRACE("hci_control_write_eir %x\n", pBuf);

    if (!pBuf)
    {
        WICED_BT_TRACE("app_write_eir %x\n", pBuf);
        return;
    }

    p = pBuf;

    length = strlen((char *)wiced_bt_cfg_settings.device_name);

    *p++ = length + 1;
    *p++ = BT_EIR_COMPLETE_LOCAL_NAME_TYPE;        // EIR type full name
    memcpy(p, wiced_bt_cfg_settings.device_name, length);
    p += length;

    *p++ = 2 + 1;                                  // Length of 16 bit services
    *p++ = BT_EIR_COMPLETE_16BITS_UUID_TYPE;       // 0x03 EIR type full list of 16 bit service UUIDs
    *p++ = UUID_SERVCLASS_SERIAL_PORT & 0xff;
    *p++ = (UUID_SERVCLASS_SERIAL_PORT >> 8) & 0xff;

    *p++ = 0;                                      // end of EIR Data is 0

    eir_length = (uint16_t) (p - pBuf);

    // print EIR data
    wiced_bt_trace_array("EIR :", pBuf, MIN(p-pBuf, 100));
    wiced_bt_dev_write_eir(pBuf, eir_length);

    return;
}

/*
 * The function invoked on timeout of app seconds timer.
 */
#if SEND_DATA_ON_TIMEOUT
void app_timeout(uint32_t count)
{
    static uint32_t timer_count = 0;
    timer_count++;
    WICED_BT_TRACE("app_timeout: %d\n", timer_count);
    if (spp_handle != 0)
    {
        wiced_bt_spp_send_session_data(spp_handle, (uint8_t *)&timer_count, sizeof(uint32_t));
    }
}
#endif

/*
 * SPP connection up callback
 */
void spp_connection_up_callback(uint16_t handle, uint8_t* bda)
{
    WICED_BT_TRACE("%s handle:%d address:%B\n", __FUNCTION__, handle, bda);
    spp_handle = handle;
```

```
}

/*
 * SPP connection down callback
 */
void spp_connection_down_callback(uint16_t handle)
{
    WICED_BT_TRACE("%s handle:%d\n", __FUNCTION__, handle);
    spp_handle = 0;
}

/*
 * Process data received over EA session.  Return TRUE if we were able to allocate buffer to
 * deliver to the host.
 */
wiced_bool_t spp_rx_data_callback(uint16_t handle, uint8_t* p_data, uint32_t data_len)
{
    int i;
//    wiced_bt_buffer_statistics_t buffer_stats[4];

//    wiced_bt_get_buffer_usage (buffer_stats, sizeof(buffer_stats));

//    WICED_BT_TRACE("0:%d/%d 1:%d/%d 2:%d/%d 3:%d/%d\n", buffer_stats[0].current_allocated_count,
buffer_stats[0].max_allocated_count,
//                   buffer_stats[1].current_allocated_count, buffer_stats[1].max_allocated_count,
//                   buffer_stats[2].current_allocated_count, buffer_stats[2].max_allocated_count,
//                   buffer_stats[3].current_allocated_count, buffer_stats[3].max_allocated_count);

//    wiced_result_t wiced_bt_get_buffer_usage (&buffer_stats, sizeof(buffer_stats));

    WICED_BT_TRACE("%s handle:%d len:%d %02x-%02x\n", __FUNCTION__, handle, data_len, p_data[0], p_data[data_len -
1]);

#if LOOPBACK_DATA
    wiced_bt_spp_send_session_data(handle, p_data, data_len);
#endif
    return WICED_TRUE;
}

/*
 * Write NVRAM function is called to store information in the NVRAM.
 */
int bt_write_nvram(int nvram_id, int data_len, void *p_data)
{
    wiced_result_t  result;
    int             bytes_written = wiced_hal_write_nvram(nvram_id, data_len, (uint8_t*)p_data, &result);

    WICED_BT_TRACE("NVRAM ID:%d written :%d bytes result:%d\n", nvram_id, bytes_written, result);
    return (bytes_written);
}

/*
 * Read data from the NVRAM and return in the passed buffer
 */
int bt_read_nvram(int nvram_id, void *p_data, int data_len)
{
    uint16_t        read_bytes = 0;
    wiced_result_t  result;

    if (data_len >= sizeof(wiced_bt_device_link_keys_t))
    {
        read_bytes = wiced_hal_read_nvram(nvram_id, sizeof(wiced_bt_device_link_keys_t), p_data, &result);
        WICED_BT_TRACE("NVRAM ID:%d read out of %d bytes:%d result:%d\n", nvram_id,
sizeof(wiced_bt_device_link_keys_t), read_bytes, result);
    }
    return (read_bytes);
}

#if SEND_DATA_ON_INTERRUPT
/*
 * Test function which sends as much data as possible.
 */
void app_send_data(void)
{
    int i;
    int increment = 0;

    while ((spp_handle != 0) && wiced_bt_spp_can_send_more_data() && (app_send_offset != APP_TOTAL_DATA_TO_SEND))
    {
```

```
        int bytes_to_send = app_send_offset + SPP_MAX_PAYLOAD < APP_TOTAL_DATA_TO_SEND ? SPP_MAX_PAYLOAD :
APP_TOTAL_DATA_TO_SEND - app_send_offset;
        for (i = 0; i < bytes_to_send; i++)
        {
//              app_send_buffer[i] = app_send_offset + i;
                app_send_buffer[i] = 65 + increment;
                increment++;
                if (increment == 26)
                {
                    increment = 0;
                }
        }
        wiced_bt_spp_send_session_data(spp_handle, app_send_buffer, bytes_to_send);
        app_send_offset += bytes_to_send;
    }
    // Check if we were able to send everything
    if (app_send_offset < APP_TOTAL_DATA_TO_SEND)
    {
        wiced_start_timer(&app_tx_timer, 100);
    }
    else
    {
        app_send_offset = 0;
    }
}

/*
 * Test function which start sending data.
 */
void app_interrupt_handler(void *data, uint8_t port_pin)
{
    int i;

    WICED_BT_TRACE("gpio_interrupt_handler pin:%d send_offset:%d\n", port_pin, app_send_offset);

     /* Get the status of interrupt on P# */
    if (wiced_hal_gpio_get_pin_interrupt_status(BUTTON_GPIO))
    {
        /* Clear the GPIO interrupt */
        wiced_hal_gpio_clear_pin_interrupt_status(BUTTON_GPIO);
    }
    // If we are already sending data, do nothing
    if (app_send_offset != 0)
        return;

    app_send_data();
}

/*
 * The timeout function is periodically called while we are sending big amount of data
 */
void app_tx_ack_timeout(uint32_t param)
{
    app_send_data();
}
#endif


#ifdef HCI_TRACE_OVER_TRANSPORT
/*
 *  Pass protocol traces up over the transport
 */
void app_trace_callback(wiced_bt_hci_trace_type_t type, uint16_t length, uint8_t* p_data)
{
    wiced_transport_send_hci_trace(host_trans_pool, type, length, p_data);
}
#endif
```

Appendix F. Makefile Customization

The WICED Studio SDK build system uses a hierarchical Makefile structure when building each project. The root Makefile is found in the main SDK installation folder as *\WICED-Studio-SDK\Makefile*, and each project contains its own specific *makefile.mk* file along with the source files in its dedicated directory. To customize the build process for a specific project, always edit the project's own makefile.mk content instead of modifying the top-level file.

The most common changes that you may need to make are as follows:

1. **Adding new .c source files to be compiled and linked:**

   Splitting the source code into multiple files can greatly improve organization and maintainability as the project grows. To add more files to the build process beyond the initial set that is created from the WICED Bluetooth Designer tool, use the `APP_SRC` keyword:

   ```
   APP_SRC  = spp.c
   APP_SRC += hello_sensor.c
   APP_SRC += wiced_bt_cfg.c
   ```

   You can include as many extra files as you need. Note that the very first file should use the direct assignment operator ("="), while all subsequent files should use the append operator ("+=").

2. **Adding extra include folders into the search path:**

   Some projects require the use of additional libraries that assume particular 'include' folders are in the compiler's include search path. To avoid having to rewrite source files with explicit include paths throughout, use the `INCS` keyword along with the `$(DIR)` variable to denote the project's root folder:

   ```
   INCS += $(DIR)/library1/include
   INCS += $(DIR)/library2/include
   INCS += $(DIR)/some/other/include
   ```

   You can add as many extra include search folders as you need. Note that all additional folders should use the append operator ("+=") since the SDK's top-level Makefile assigns some folders already. Using the direct assignment operator ("=") will wipe out these default folders and break the compile process.

3. **Applying pre-built optional patches that are part of the WICED SDK:**

   Since the WICED Studio Bluetooth LE stack is part of the chipset ROM inside the module, updates and fixes to low-level functionality require the use of precompiled patches, which are loaded and applied during the boot process. These patches must be included especially during the compile process so that they are part of the final firmware binary image. To specify patches for this purpose, use the `APP_PATCHES_AND_LIBS` keyword:

   ```
   APP_PATCHES_AND_LIBS += FM25Q04_sflash.a
   ```

Appendix G. Bluetooth Qualification and Regulatory Certification References

Table 24 details the references for Bluetooth Qualification and Listing as well as Regulatory test reports and certificates for each EZ-BT module. These references can be found by visiting www.cypress.com/cypress_bluetooth_modules or by clicking on the hyperlinks in the table below.

Table 24. Bluetooth QDID and Regulatory Test Report and Certificate References

EZ-BT Module Part Number	Knowledge Base Article Containing Regulatory Reports and Certificates	Module QDID and Link to Bluetooth SIG Listing Page
CYBT-343026-01	KBA220396	D035378
CYBT-353027-02	KBA223952	D039123
CYBT-423028-02	KBA223751	D039125
CYBT-413034-02	TBD	D040142
CYBT-483039-02	TBD	D040143

Document History

Document Title: AN223400 – Getting Started with EZ-BT WICED Modules

Document Number: 002-23400

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6352193	DSO	12/06/2018	New application note

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709