Objective

This code example demonstrates the implementation of a simple BLE Environmental Sensing Profile and how to send environment sensing parameters like temperature by making use of an on-chip ADC and sending values over a BLE PHY2M link using the CYW20819 Bluetooth SoC. This example uses the ModusToolbox™ integrated development environment (IDE).

Requirements

Tool: ModusToolbox™ IDE 1.1

Programming Language: C

Associated Parts: CYW20819

Related Hardware: CYW920819EVB-02 Evaluation Kit

Overview

This code example demonstrates the BLE Environmental Sensing Service (ESS) specification from the Bluetooth SIG on a CYW20819 device. The application demonstrates the ADC and BLE 5.0 (LE2M PHY) capability of the CYW20819 device. "BLE_EnvironmentSensingTemperature" is the name of the code example that is available in ModusToolbox.

See the "Getting Started" section of ModusToolbox IDE User Guide to learn how to create a starter application from existing examples. Some of the code examples are available within ModusToolbox while others need to be downloaded from GitHub. See the "Getting Started" section of the ModusToolbox IDE User Guide to learn how to download code examples from GitHub. This code example document also describes the hardware connections necessary for the desired behavior of the code example and how to verify the output.

Hardware Setup

This example uses the kit's default configuration. Refer to the kit user guide, if required, to ensure that the kit is configured correctly. The board should be connected to a PC using the micro-B USB cable. Also, make sure that the Thermistor Enable (J14) and Peripheral Enable (J18) jumpers are in the correct positions (shorted). See the "Jumper Setting" section in the kit user guide.

Software Setup

This code example consists of two parts: a BLE Peripheral device (GATT Server) and a BLE Central device (GATT Client). A smartphone or a PC will act as the BLE Central device whereas the programmed kit will act as the BLE Peripheral device. For the Central device, download and install the CySmart app for iOS or Android. You can also use the CySmart Host Emulation Tool desktop application if you have access to the CY5677 CySmart BLE 4.2 USB Dongle. Alternatively, you can use any other mobile app that supports the BLE Environment Sensing Service.

Scan the following QR codes from your mobile phone to download the CySmart app.

iOS   Android

Operation

1. Connect the CYW920819EVB-02 Evaluation Kit to your PC using the provided USB cable.

2. The kit USB interface should enumerate as two serial communication (COM) ports – WICED HCI UART and WICED Peripheral UART. The WICED HCI UART port is used by the ModusToolbox programming interface for downloading the application code; no user configuration is required for this port. The WICED Peripheral UART port is used in this code example for printing the BLE stack event messages and other application-level log messages. Use a serial terminal emulation tool such as PuTTY or Tera Term. Please refer to the kit user guide for more details on driver installation.

3. You can either browse the collection of starter applications during application creation through **File** > **New** > **ModusToolbox IDE Application** or import the code example from the GitHub repository into a new workspace. The CYW920819EVB-02 kit is pre-programmed with this Environment Sensing Profile code example out-of-the-box. For the most recent version of the code example, check the GitHub repository. You can find the Environmental Sensing Profile code example in this path CYW920819EVB-02/apps/demo/env_sensing_temp. If you are not familiar with this import process, see KBA225201.

4. Program the CYW20819 device on the kit. In the project explorer, select the **<App Name>** project. In the Quick Panel, scroll to the **Launches** section and click the **Build + Program** configuration as shown in Figure 1.

Figure 1. Quick Links of ModusToolbox
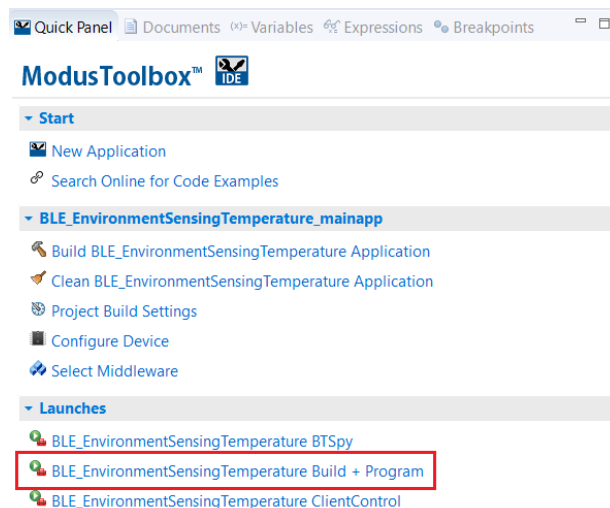
Note: If the download fails, it is possible that a previously loaded application is preventing programming. For example, the application might use a custom baud rate that the download process does not detect. In that case, it may be necessary to put the board in recovery mode, and then try the programming operation again from the IDE. To enter recovery mode, first, press and hold the **Recover** button (SW1), then press and release the **Reset** button (SW2), and then release the **Recover** button (SW1).
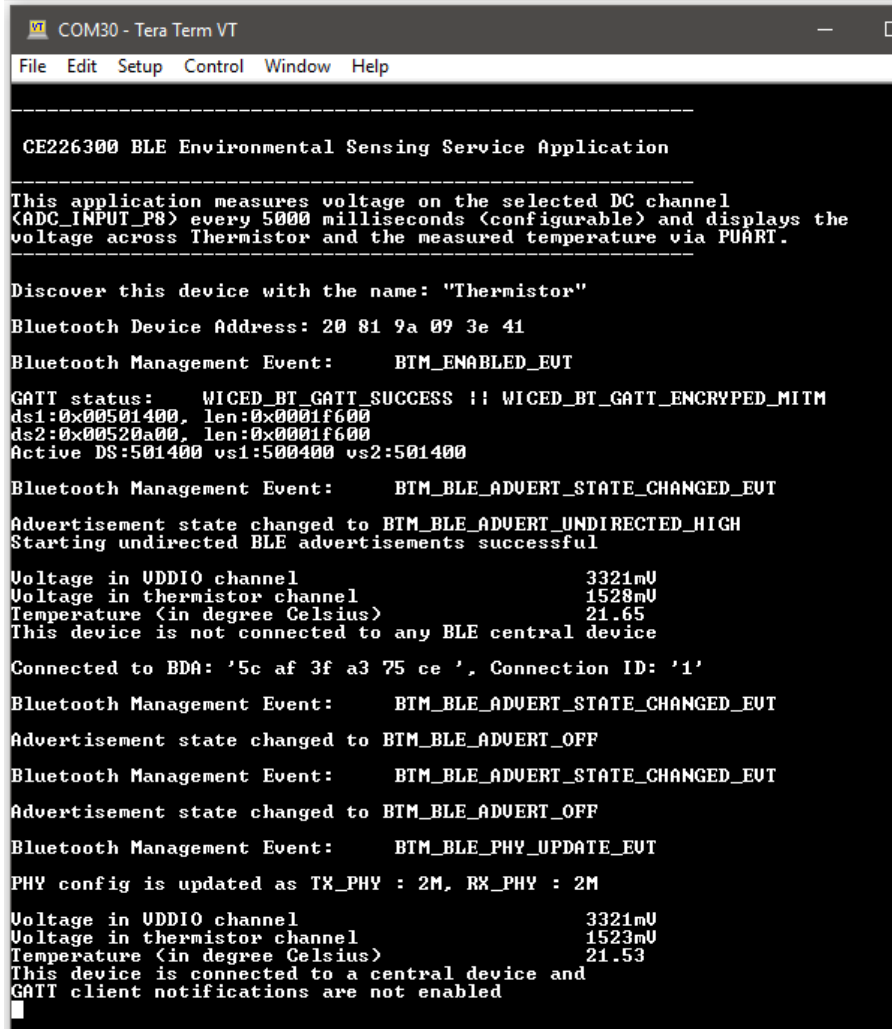
5. Use the PUART serial interface to view the Bluetooth stack and application trace messages:

   a. Use the serial terminal emulation tool and connect to the **PUART** serial port. Configure the terminal emulation tool to access the serial port using settings listed in Table 1.

Table 1. WICED PUART Settings

WICED PUART Serial Port Configuration	Value
Baud rate	115200 bps
Data	8 bits
Parity	None
Stop	1 bit
Flow control	None
Line Feed option	CR+LF or Auto

b. The Bluetooth stack and application trace messages will appear in the terminal window as shown in Figure 2. You may have to reset the device again as you might have missed the initial trace messages after programming.

Figure 2. Log Messages on WICED PUART Serial Port

6. To test using the CySmart mobile app:
    a. Turn ON Bluetooth on your Android or iOS device.
    b. Launch the CySmart app. Figure 3 shows the steps for using the CySmart Android app. The screenshots provided in this document are from the Android app; these will vary slightly from the iOS version.

Figure 3. Testing with the CySmart App on Android
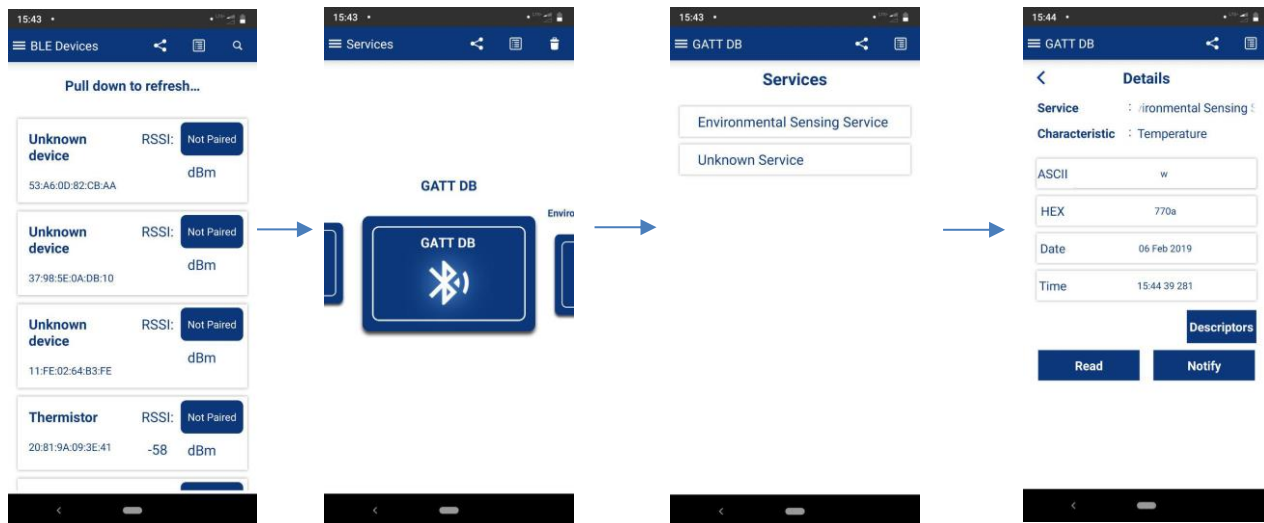
c. Pull (swipe) down on the home screen of the CySmart app to start scanning for BLE Peripherals; your device appears in the CySmart app home screen with the Bluetooth device name as "Thermistor". Select your device to establish a BLE connection. Once the connection is established, the red LED (LED2) turns ON.

d. Select **GATT DB** from the carousel view. The CySmart app displays the available services.

e. Select **Environmental Sensing Service** from the list of available services.

f. The CySmart app displays the temperature characteristic in two representations such as ASCII and hexadecimal values. The same screen also shows the date and time of the last received temperature notification. The bytes in the 2-byte hexadecimal value need to be swapped to interpret the temperature. For example, if the hexadecimal representation in the CySmart app is 0xCE08, then swapping the bytes result in 0x08CE which is the decimal equivalent of 2254. The most significant byte (MSB) represents the integer part of the temperature in Celsius and the least significant byte (LSB) represent the fractional part of the temperature in Celsius (i.e., 22.54 °C).

g. At the bottom of the screen, you can find three buttons, namely,

i. Descriptors (To view additional information about the environmental sensing profile such as valid range, measurement interval, and measurement techniques)

ii. Read (To read the last sampled temperature value once)

iii. Notify (To receive notifications when the server sends an updated temperature value. The code example is configured to send a temperature value every 5 seconds)

7. Upon disconnecting the Central device from the peripheral device, the red LED (LED2) turns OFF and the user will be notified about the remote device's Bluetooth address which has recently disconnected in the PUART log.

Design and Implementation

The code example application runs on the Arm® Cortex®-M4 core of the CYW20819 Bluetooth SoC. This code example is designed to work with CYW920819EVB-02 and developed using the ModusToolbox IDE.

This application demonstrates the ADC and BLE capability of the CYW20819 device. A negative temperature coefficient type of thermistor is used to measure the ambient temperature. The thermistor present on the CYW920819EVB-02 evaluation board is NCU15WF104F60RC from Murata. It is a 100-kΩ thermistor at 25°C with 1% accuracy. The thermistor is connected to GPIO P8 on the CYW20819 device. The ADC is used to measure the voltage across the thermistor channel to determine the temperature. The temperature value is sent over BLE to a Central device, and to the PUART as debug trace messages to a terminal emulation tool. This project demonstrates the following features:

- Functionality of the ADC for temperature measurement.
- BLE Environment Sensing Service (ESS) – GATT Read and Notify functionality.
- Debug Trace messages.
- Connection with one Central device.
- BLE 5.0 Feature - BLE2MPHY if the Central device supports 2M PHY.
- Connection status indication through LED.

The project consists of the following files:

Table 2. BLE_EnvironmentSensing Project Files

File Name	Comments
thermistor_app.c	This file contains the `application_start()` function, which is the entry point of the user code execution and can be considered as the equivalent of the main() function in standard C. The application_start function initializes the Bluetooth stack by calling `wiced_bt_stack_init()`. After the Bluetooth stack is initialized, the users have the control to execute their own application code based on the different Bluetooth events received in the management callback. This management callback acts as a Finite State Machine (FSM) in executing the application code. For the Bluetooth Enabled Event (`BTM_ENABLED_EVT`), the `thermistor_app_init()` function is called. This function initializes the ADC, starts a timer to measure the temperature periodically by calling `seconds_timer_temperature_cb()`, and starts BLE advertisements. This file also holds the function `thermistor_set_advertisement_data()` which sets the advertisement data for the device to be discovered.
thermistor_gatt_handler.c *thermistor_gatt_handler.h*	These files consist of the code to perform GATT read handler, GATT write handler and GATT indication confirm handler. These files hold the code to switch to LE2M PHY if the Central device supports it.
thermistor_temp_db.c *thermistor_temp_db.h*	These files contain the function to map resistance to temperature values of the thermistor using a lookup table (from the thermistor datasheet).
thermistor_util_functions.c *thermistor_util_functions.h*	These files consist of the utility functions that will help make debugging and developing the applications easier by providing more meaningful information. For example, this API provides meaningful strings for Bluetooth events, Bluetooth advertisement modes, GATT status messages and GATT disconnection reasons.
wiced_app_cfg.c	These files contain the runtime Bluetooth stack configuration parameters like device name, advertisement/connection interval, buffer pool configurations, etc.
cycfg_bt.h *cycfg_gatt_db.c* *cycfg_gatt_db.h*	These files reside in the *GeneratedSource* folder under the application folder. They contain the GATT database information generated using the Bluetooth Configurator tool.
cycfg_pins.c *cycfg_pins.h* *cycfg_routing.c* *cycfg_routing.h* *cycfg_notices.h*	These files reside in the *GeneratedSource* folder under the application folder. They contain the pin and peripheral configuration.
readme.txt	This file provides minimal information about the code example such as how to download the application and what to observe from the application, etc.

Flowchart

Figure 4. Flowchart of the BLE Environment Sensing Temperature Application
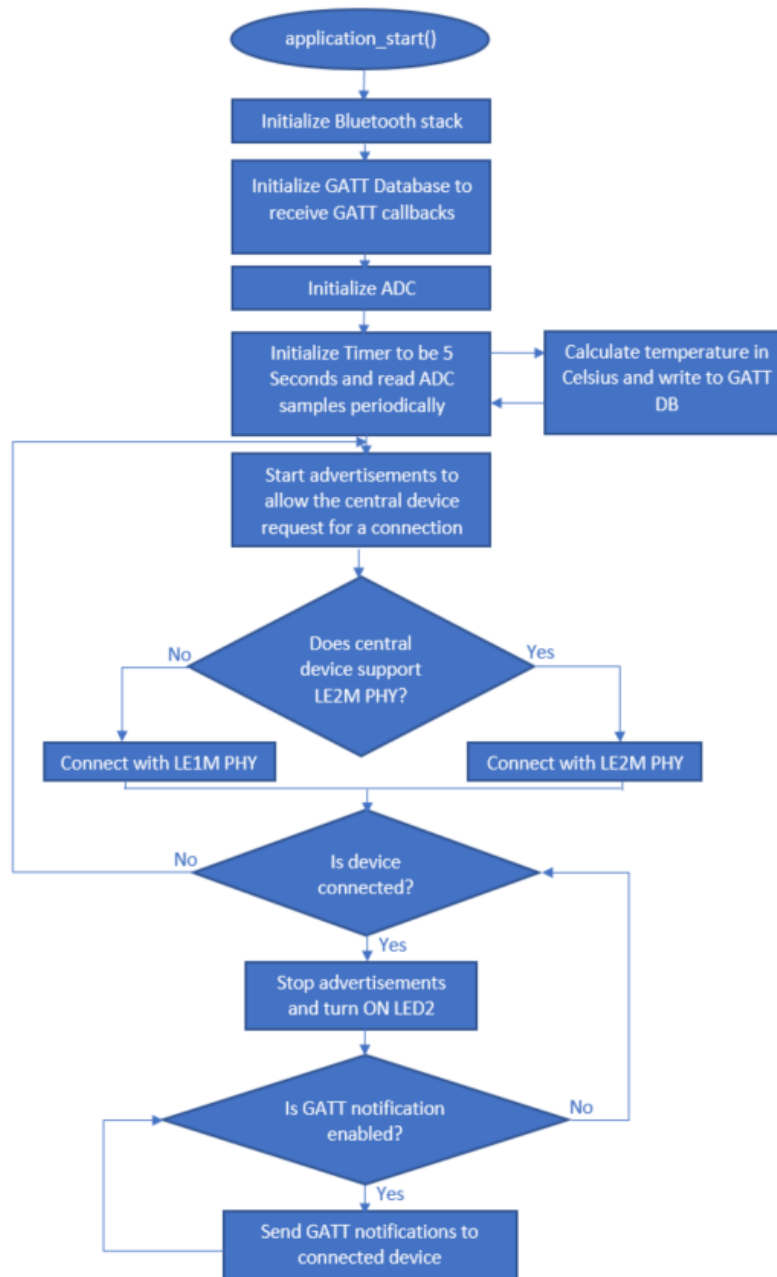
The flowchart as shown in Figure 4 gives an overview of the Environmental Sensing Profile application. The Environmental Sensing Profile in this application consists of one of the Bluetooth SIG-defined service, the Environment Sensing Service (ESS). This project implements only the temperature characteristic from the environmental sensing service. This characteristic supports notification and read operations, which allows the GATT Server to send data to the connected GATT Client device whenever new data is available and a read to be performed from the GATT Client device. The Bluetooth Configurator provided by ModusToolbox makes it easier for the user to design and implement the GATT DB. The Bluetooth Configurator generates files such as *cycfg_gatt_db.c* and *cycfg_gatt_db.h*. All the Environmental Sensing Profile related variables and functions will be generated in these files. When the GATT DB is initialized after the Bluetooth Stack initialization, all the profile related values will

be ready to be advertised to the Central devices. The procedure on how to use the Bluetooth Configurator is discussed later in this document.

The code example utilizes the on-board NCU15WF104F60RC analog temperature sensor (thermistor) to send temperature values over BLE. This is a Negative Temperature Coefficient (NTC) temperature sensor whose resistance increases when ambient temperature increases. The datasheet of NCU15WF104F60RC provides details on the varying characteristics of the temperature sensor.

In the code example, *thermistor_temp_db.c* and *thermistor_temp_db.h* are the files of interest for converting ADC samples to temperature values. CYW20819 does not have a Hardware Floating Point Unit (FPU). The resistance values are multiplied by a factor of 100 to use unsigned integer values instead of floating-point values. The source files hold a mapping table between the thermistor's resistance values and temperature.

The `r_t_look_up()` function in *thermistor_temp_db.c* calculates the temperatures at the previous resistance and the next resistance to calculate the slope. Using this slope value, an accurate temperature is calculated. The temperature value is sent once every 5 seconds to the Central device when connected and GATT notifications are enabled by the Central. To send the temperature values once every 5 seconds, an application timer is used which can be configured as MILLI_SECONDS_PERIODIC_TIMER or SECONDS_PERIODIC_TIMER. When the peripheral device is connected, LED2 will be ON and when it is disconnected, LED2 will be OFF. To turn the LED ON and OFF, generic GPIO functions are used to drive the output pin HIGH or LOW. The LEDs present in CYW920819EVB-02 are active LOW LEDs which means that the LED turns ON when the GPIO is driven LOW.

The *thermistor_gatt_handler.c* and *thermistor_gatt_handler.h* files handle the functionality of GATT callbacks from the Central device. On receiving a connection request, the Bluetooth stack gives a GATT event to the application of `wiced_bt_gatt_evt_t` type. For example, the LED toggle functionality is implemented in the GATT connection callback. Also, set the PHY Rx and Tx connection to BLE2MPHY if the Central device supports it as specified by the Bluetooth 5.0 standard. On a disconnection event, the code resets the CCCD value so that on a reconnect event, notifications will be disabled.

In *wiced_app_cfg.c,* all the runtime Bluetooth stack configuration parameters are defined and will be initialized during Bluetooth stack initialization. Some of the configurations include device name, connection interval, advertisement interval, advertisement channels to use, number of client connections, Maximum Transmission Unit (MTU), etc. You also have the flexibility to configure the buffer pool size, which helps in optimizing the memory and transmission rate depending on the application use case.

The project also contains *thermistor_util_functions.c* and *thermistor_util_functions.h* which provide APIs to see meaningful messages in debug logs in the PUART. Most of the status messages in Bluetooth stack are represented in enumerated values. These functions allow you to view the respective strings instead of the enumerated values.

Resources and Settings

This section explains the ModusToolbox resources and their configuration as used in this code example. Note that all the configuration explained in this section has already been done in the code example. The ModusToolbox IDE stores the configuration settings of the application in the *design.modus* file. This file is used by the graphical configurators, which generate the configuration source files. These files are stored in the application's *GeneratedSource* folder.

Figure 5 shows the Device Configurator settings for this code example. The Device Configurator is used to enable and configure the peripherals and the pins used in the application. To launch the Device Configurator, double-click the *design.modus* file or click on **Configure Device** in the Quick Panel. Any time you make a change in the Device Configurator, click **File** > **Save** to save the updated configuration.

Figure 5. Device Configurator
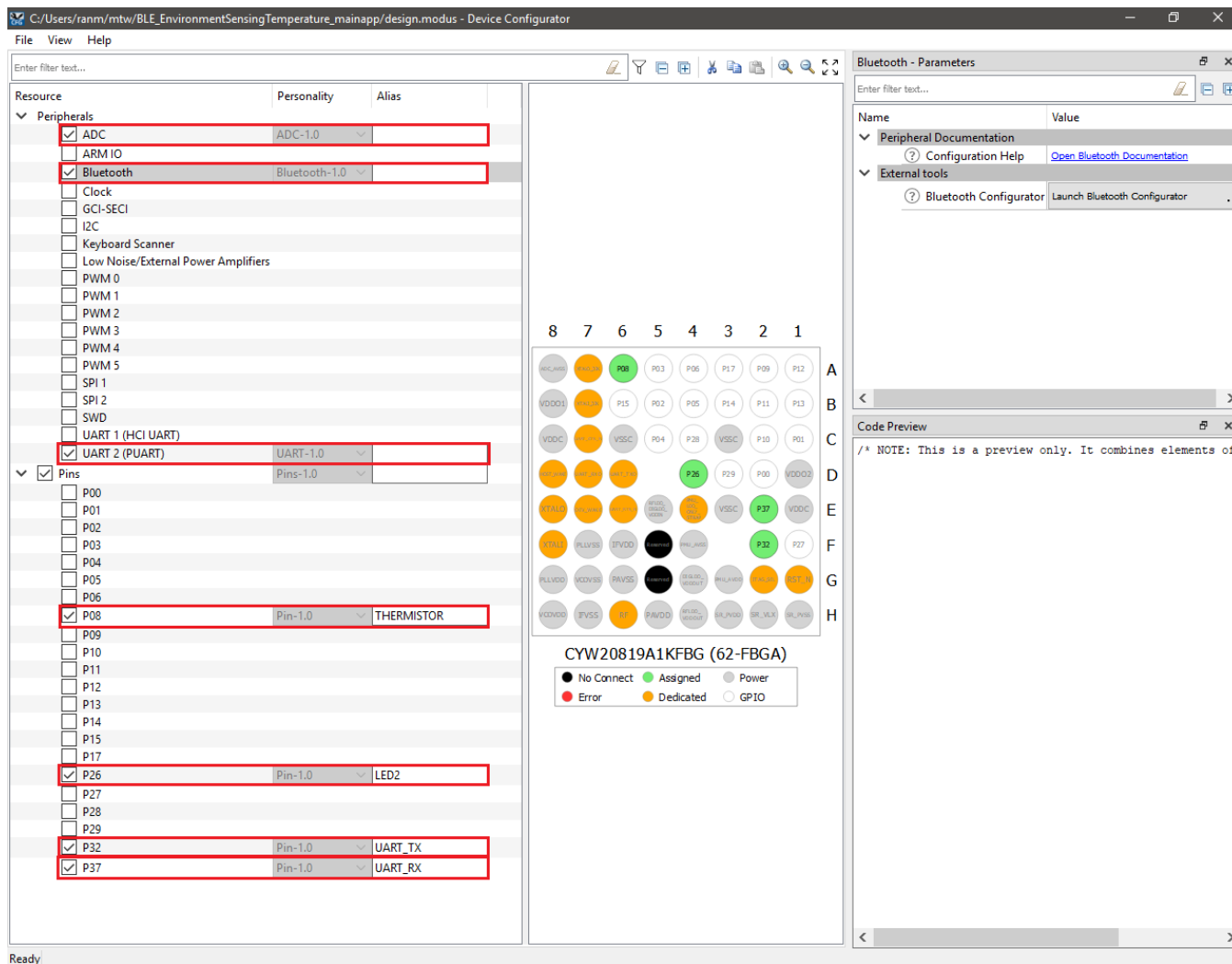
- **Peripherals**: The design uses the ADC, Bluetooth and PUART peripherals which are enabled under the **Peripherals** section as shown in Figure 5. The Peripheral settings of ADC and PUART respectively as shown in Figure 6 and Figure 7.
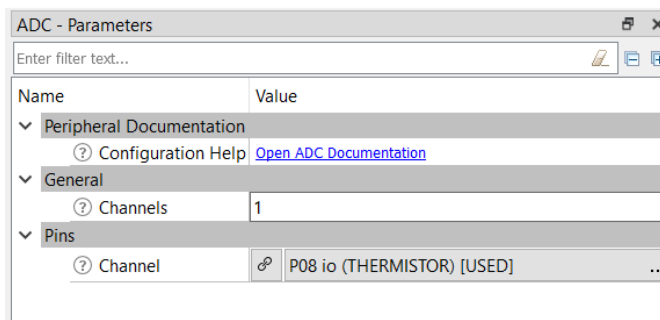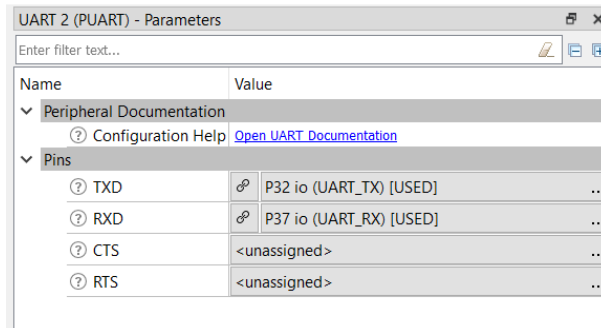
Figure 6. ADC Peripheral Configuration

Figure 7. PUART Peripheral Configuration

- **Bluetooth Configurator**: The Bluetooth Configurator is used for creating and modifying the BLE GATT database. To launch the Bluetooth Configurator from the Device Configurator window, click on the **Bluetooth** resource under the **Peripherals** section, and then click on the **Launch Bluetooth Configurator** shortcut under the **Bluetooth-Parameters** window, as shown in Figure 8.

Figure 8. Launch Bluetooth Configurator
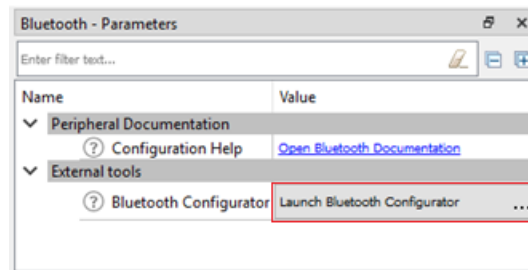
The Bluetooth Configurator for this code example is shown in Figure 9. Note that the Environmental Sensing Service corresponding to the Environmental Sensing Profile has been added as shown in Figure 10. Figure 10 shows how the Environment Sensing profile was added to create the GATT database. Any time you make a change in the Bluetooth Configurator, ensure that you save the changes (click **File** > **Save** or click the **Save** icon).

Figure 9. Bluetooth Configurator
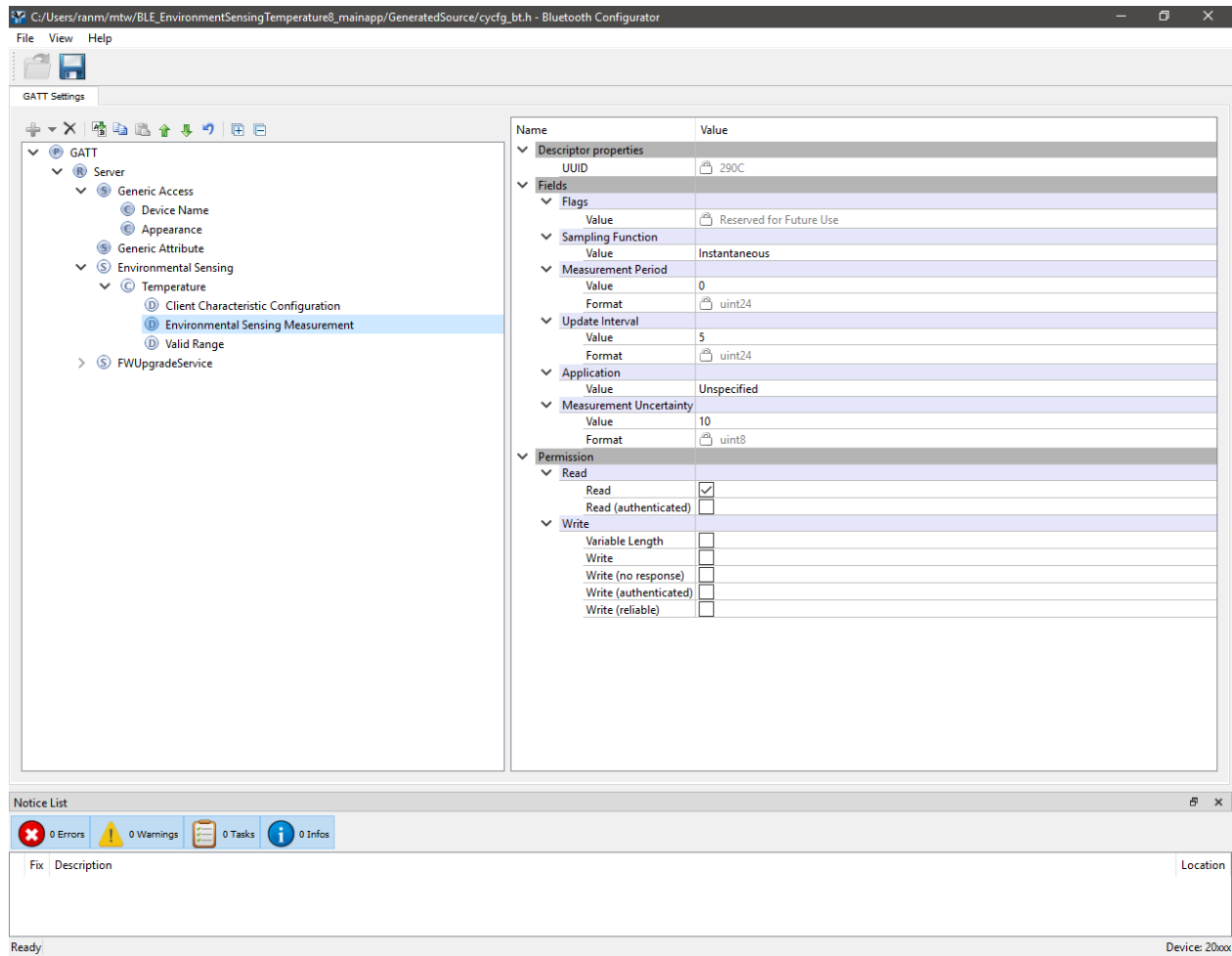
Figure 10. Environment Sensing Profile Bluetooth Configurator
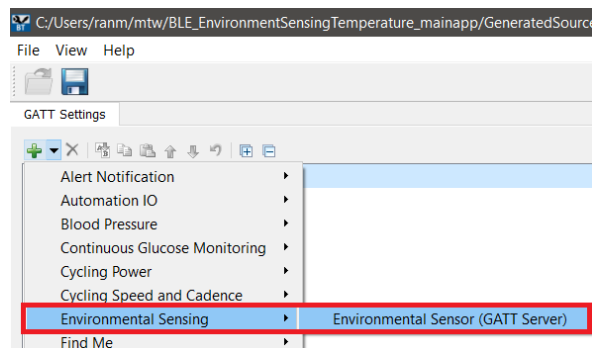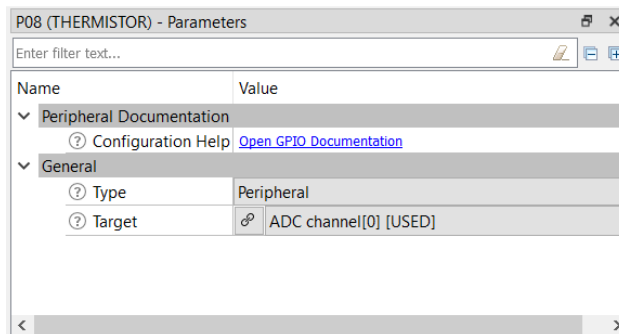
- **Pins**: The pins used in the application are enabled in the **Pins** section of the Device Configurator as shown in Figure 5. Table 3 provides more information on the pins.

Table 3. Pin Mapping Details

Pin	Alias	Purpose	Settings
P08	THERMISTOR	CYW20819's P08 GPIO pin is connected to the thermistor on CYW920819EVB-02. The thermistor gives an analog output, which needs to be sampled and converted to digital value using the on-chip ADC.	See Figure 11
P26	LED2	Mapped to the red LED (LED2) on the kit. LED ON indicates that the device is connected to a Central device.	See Figure 12
P32	PUART_TX	Tx pin of the PUART peripheral.	See Figure 13
P37	PUART_RX	Rx pin of the PUART peripheral.	See Figure 14

Figure 11. Thermistor Pin Settings
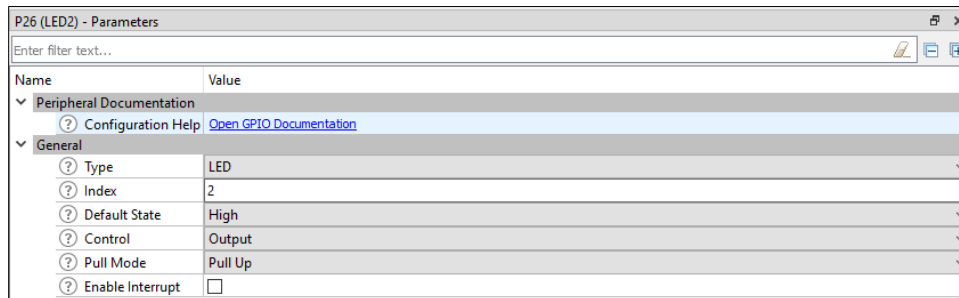
Figure 12. LED2 Pin Settings

Figure 13. PUART Tx Pin Settings

Figure 14. PUART Rx Pin Settings

Related Documents

Application Notes	
AN225684 – Getting Started with CYW20819	Describes CYW20819 Bluetooth SoC and how to build your first BLE application using the device in ModusToolbox IDE
Code Examples	
Visit the Cypress GitHub repository for a comprehensive collection of code examples using ModusToolbox IDE	
Device Documentation	
CYW20819 Device Datasheet	CY5677 CySmart BLE 4.2 USB Dongle	CYW20819 Feature and Peripheral Guide
Development Kits	
CYW920819EVB-02 Evaluation Kit	
Tool Documentation	
ModusToolbox IDE	The Cypress IDE for IoT designers

Document History

Document Title: CE226300 – BLE Environment Sensing Temperature with CYW20819

Document Number: 002-26300

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6490178	RANM	02/20/2019	New code example

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 | Code Examples | Projects | Videos | Blogs | Training| Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.