www.infineon.com

PSoC® 4 BLE – Designing BLE Applications

Author: Uday Agarwal, Ajay Sahu
Associated Project: Yes
Associated Part Family: CY8C4XX7-BL, CY8C4XX8-BL, CYBL1XX6X, CYBL1XX7X
Software Version: PSoC Creator™ 4.2
Related Application Notes: See Related Documents
To get the latest version of this application note, please visit
http://www.cypress.com/go/AN91184.

More code examples? We heard you.
To access an ever-growing list of hundreds of PSoC code examples, please visit our code examples web page. You can also explore the PSoC video library here.

AN91184 shows how to design a Bluetooth® Low Energy (BLE) application based on PSoC 4 BLE, using standard profiles defined by the Bluetooth SIG that are included in the BLE Component in PSoC Creator. It demonstrates how to build an application with the BLE Health Thermometer Profile on the CY8CKIT-042-BLE kit.

Contents

1   Introduction

Bluetooth Low Energy (BLE) is an ultra-low-power wireless standard introduced by the Bluetooth Special Interest Group (SIG) for short-range communication. The BLE physical layer, protocol stack, and profile architecture are designed and optimized to minimize power consumption. Similar to Classic Bluetooth, BLE operates in the 2.4-GHz ISM band but with lower bandwidths ranging from 125 kbps to 2 Mbps.

Cypress' PSoC 4 BLE is a programmable embedded system-on-chip (SoC), integrating BLE along with programmable analog and digital peripheral functions, memory, and an Arm® Cortex®-M0 microcontroller on a single chip.

This application note discusses how to use the PSoC Creator BLE Component to design a BLE Health Thermometer application using the Health Thermometer standard profile, and then validate the application using the CySmart Central Emulation Tool and the CySmart mobile app. The PSoC Creator BLE Component has the standard profiles pre-built; this makes it very easy to use these services in BLE-enabled projects.

This application note assumes that you are familiar with the basics of BLE, PSoC, the PSoC Creator IDE, and temperature measurement using a thermistor. Refer to the following links:

■   AN91267 – Getting Started with PSoC 4 BLE

■   PSoC Creator home page

■   AN66477 – PSoC® 3, PSoC 4, and PSoC 5LP – Temperature Measurement with a Thermistor

2 PSoC Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right PSoC device and quickly and effectively integrate it into your design. For a comprehensive list of resources, see KBA86521, How to Design with PSoC 3, PSoC 4, and PSoC 5LP.

The following is an abbreviated list for PSoC 4 BLE:

- **Overview:** PSoC Portfolio, PSoC Roadmap

- **Product Selectors:** PSoC 1, PSoC 3, PSoC 4, or PSoC 5LP. In addition, PSoC Creator includes a device selection tool.

- **Datasheets** describe and provide electrical specifications for the PSoC 41XX-BL and PSoC 42XX-BL device families.

- **Application Notes and Code Examples** cover a broad range of topics, from basic to advanced level. Many of the application notes include code examples. PSoC Creator provides additional code examples.

- **Technical Reference Manuals (TRMs)** provide detailed descriptions of the architecture and registers in each PSoC 4 BLE device family.

- **CapSense Design Guide:** Learn how to design capacitive touch-sensing applications with the PSoC 4 BLE family of devices.

- **Development Tools**
  - CY8CKIT-042-BLE Bluetooth Low Energy (BLE) Pioneer Kit includes connectors for Arduino™ compatible shields and Digilent® Pmod™ daughter cards.
  - CySmart BLE Host Emulation Tool for Windows, iOS, and Android is an easy-to-use GUI that enables you to test and debug your BLE Peripheral applications. Source code for CySmart mobile apps is also available at Cypress website.

3   PSoC Creator

PSoC Creator is a free Windows-based Integrated Design Environment (IDE). It enables concurrent hardware and firmware design of systems based on PSoC 3, PSoC 4, PSoC 5LP, and PSoC 6. With PSoC Creator, you can:

1. Browse the collection of code examples from the **File** > **Code Example** menu.

2. Explore the library of 100+ Components

3. Drag and drop Components to build your hardware system design in the main design workspace

4. Review Component datasheets

5. Configure Components using configuration tools

6. Codesign your application firmware with the PSoC hardware

Figure 1. PSoC Creator Features
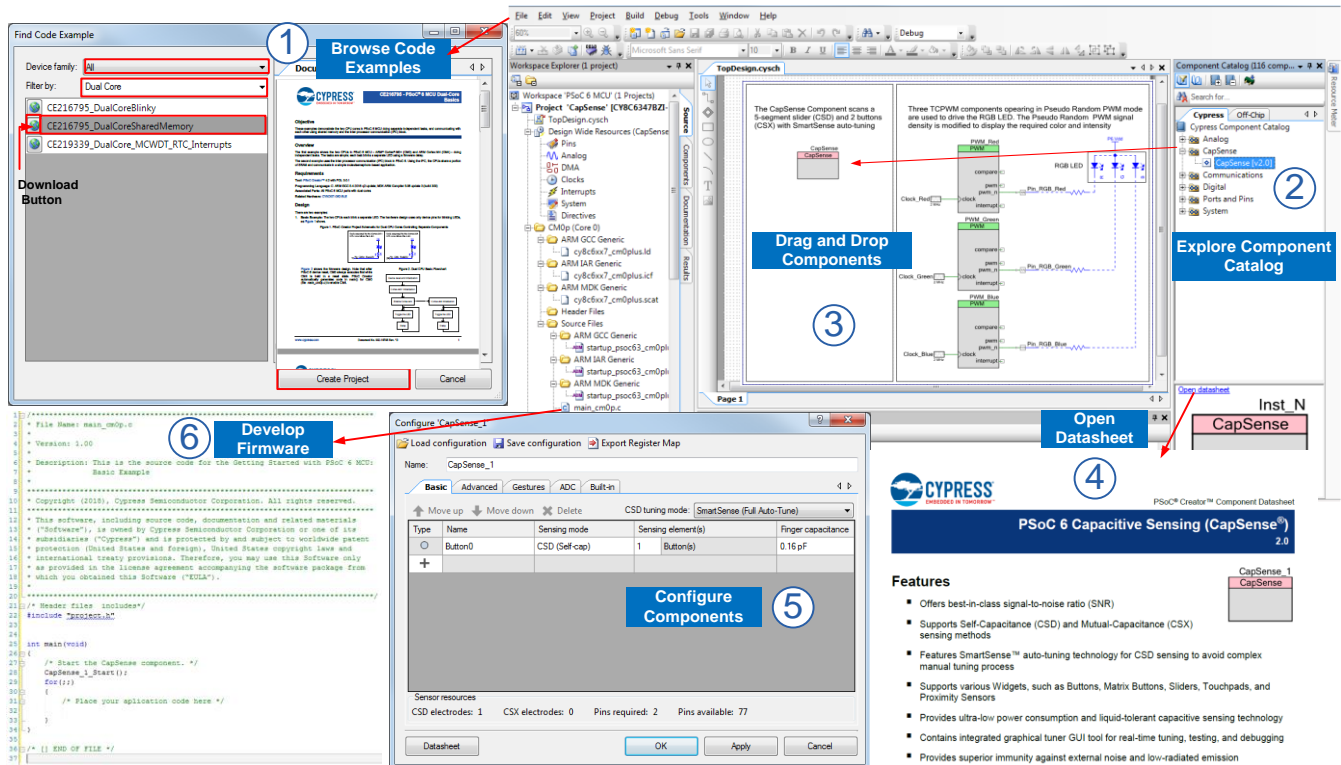
4 Standard Services Versus Custom Services

The Bluetooth SIG defines a set of services that can be configured as either a GATT client or a GATT server. These services are termed Standard Services. Some examples of standard services include: Heart-Rate Service, Health Thermometer Service, Blood Pressure Service, and Immediate Alert Service. Refer to the Bluetooth Developer Portal for the complete list of standard services.

These standard services are defined to address a wide variety of applications. For example, the Heart Rate Service can be configured to report data from a Heart-Rate Sensor in a wristband or a chest-strap monitor. It can also expose the amount of energy expended over a specified interval.

The BLE standard also allows you to create your own services, known as Custom Services. As the name suggests, they are used to define services that are not covered by BLE standard services. These services are equally important as they allow you to deploy BLE devices that can have custom applications.

4.1 BLE Health Thermometer

In the BLE Health Thermometer application (Figure 2), the thermometer device operates as the GAP Peripheral and implements the Health Thermometer Sensor Profile, while the mobile device receiving the data operates as the GAP Central and implements the Health Thermometer Collector Profile. In this example, the Health Thermometer Sensor Profile implements two standard services – the Health Thermometer Service that comprises three characteristics (the Temperature Measurement Characteristic, the Temperature Type Characteristic, and the Measurement Interval Characteristic) and the Device Information Service that comprises nine characteristics, which will be described later in this document.

Figure 2. BLE System Design
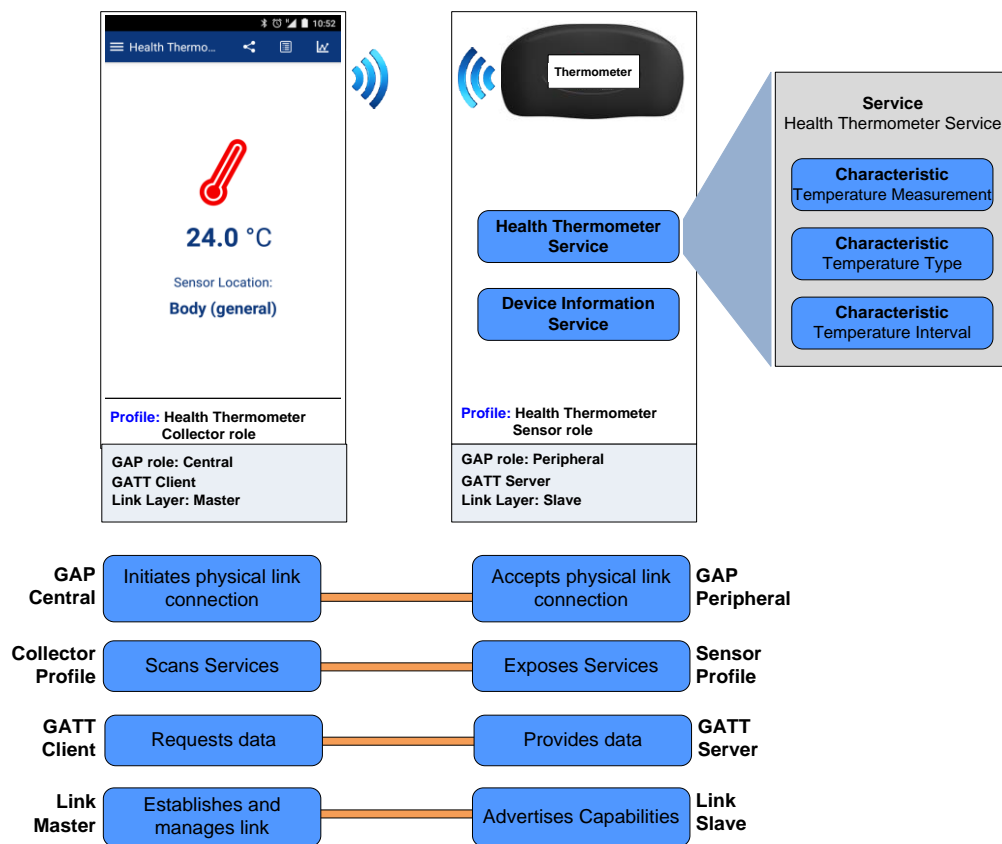
5   PSoC Creator Project: Health Thermometer

In this project, the PSoC 4 BLE device integrates the following:

- A BLE Component that operates as the Peripheral at the GAP layer and as the GATT server at the GATT layer.
- An ADC, which measures the voltage across a thermistor.
- A thermistor calculator, which calculates the temperature using the ADC reading.
- A user button, which wakes up the system from the Hibernate mode.

Figure 3 shows the PSoC Creator schematic of the Health Thermometer project.

Figure 3. PSoC Creator Schematic
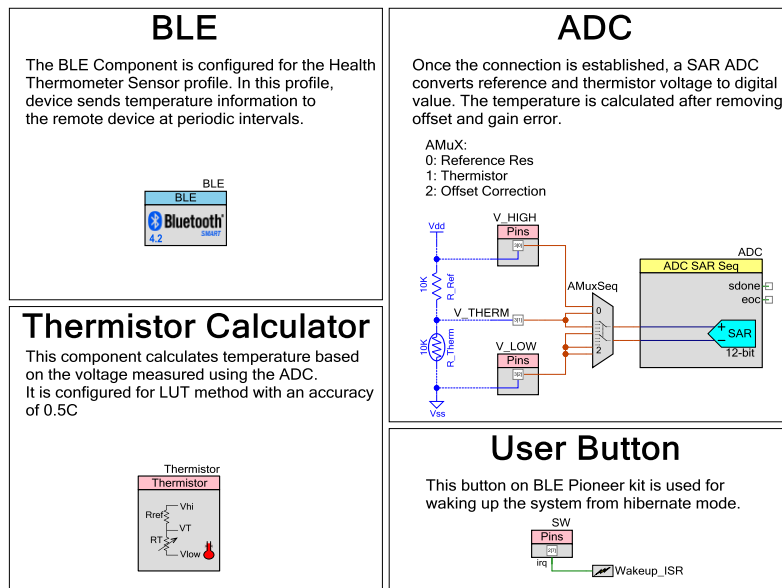
5.1   Configure the Component

1. Create a new PSoC 4100 BLE / PSoC 4200 BLE Design project. If you are new to PSoC Creator, refer to the PSoC Creator home page.

2. Drag and drop a BLE Component (Component Catalog > Communications) into the TopDesign schematic (refer to Figure 4).

Figure 4. BLE Component
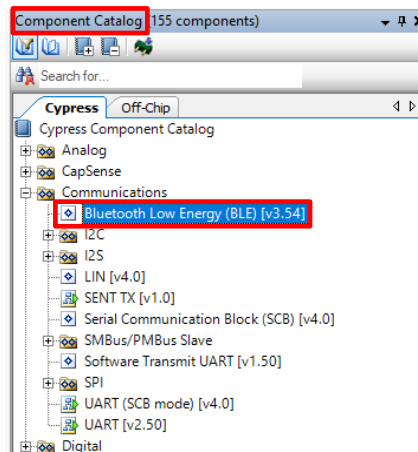
3. Double-click the BLE Component to configure it. The configuration window appears as shown in Figure 5.

Figure 5. BLE Component – Configuration Window
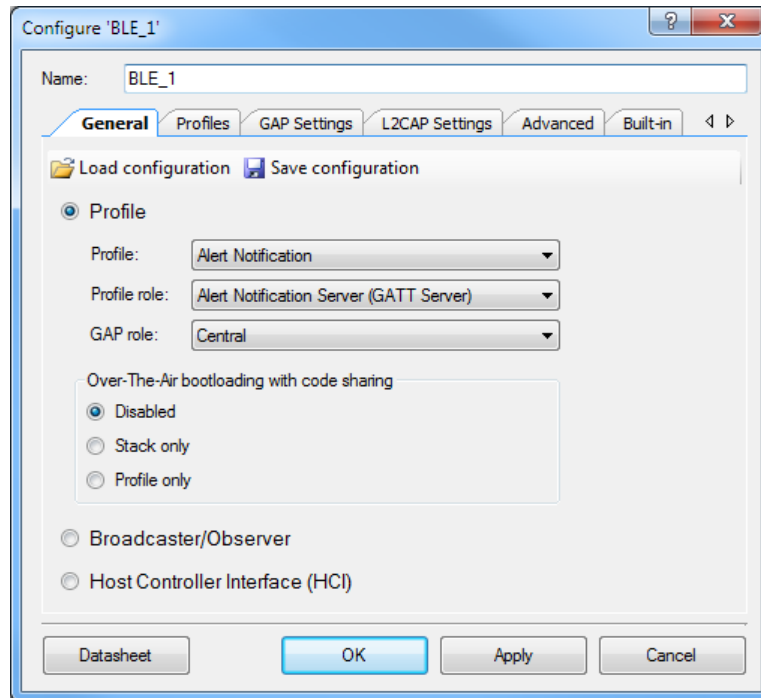

4. On the **General** tab of the Component Configuration window (refer to Figure 6), configure the following settings:

Name: BLE
Configuration: Profile Collection
Profile: Health Thermometer
Profile role: Thermometer (GATT Server)
Over-The-Air bootloading with code sharing: Disabled

Figure 6. General Tab
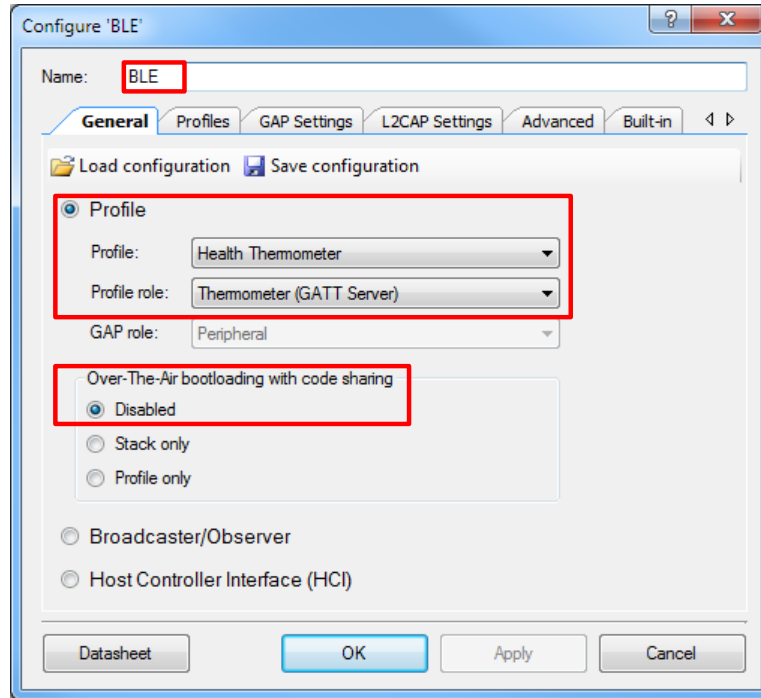
Note: Per the Bluetooth SIG, the Health Thermometer standard profile encapsulates the Health Thermometer Service and the Device Information Service; therefore, these services are added by default, as shown in Figure 7. To learn more about the Health Thermometer Profile or the Health Thermometer Service, refer to the Bluetooth Adopted Specifications.

Figure 7. Health Thermometer Profile
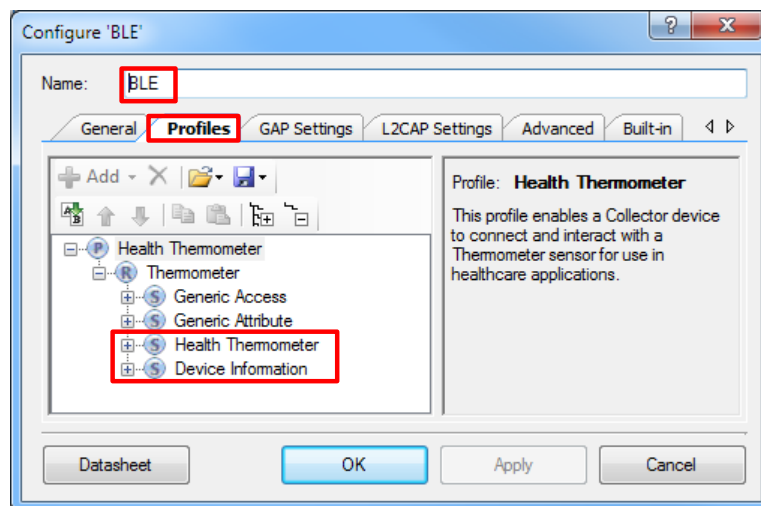
5. Configure the **Profiles** tab with the following settings:

   Service: Health Thermometer
   Characteristic: Temperature Measurement
   Fields: Temperature Units Flag
   Value: Temperature Measurement Value in units of Celsius
   Figure 8 shows the Temperature Measurement Characteristic configuration.

Figure 8. Temperature Measurement Characteristic
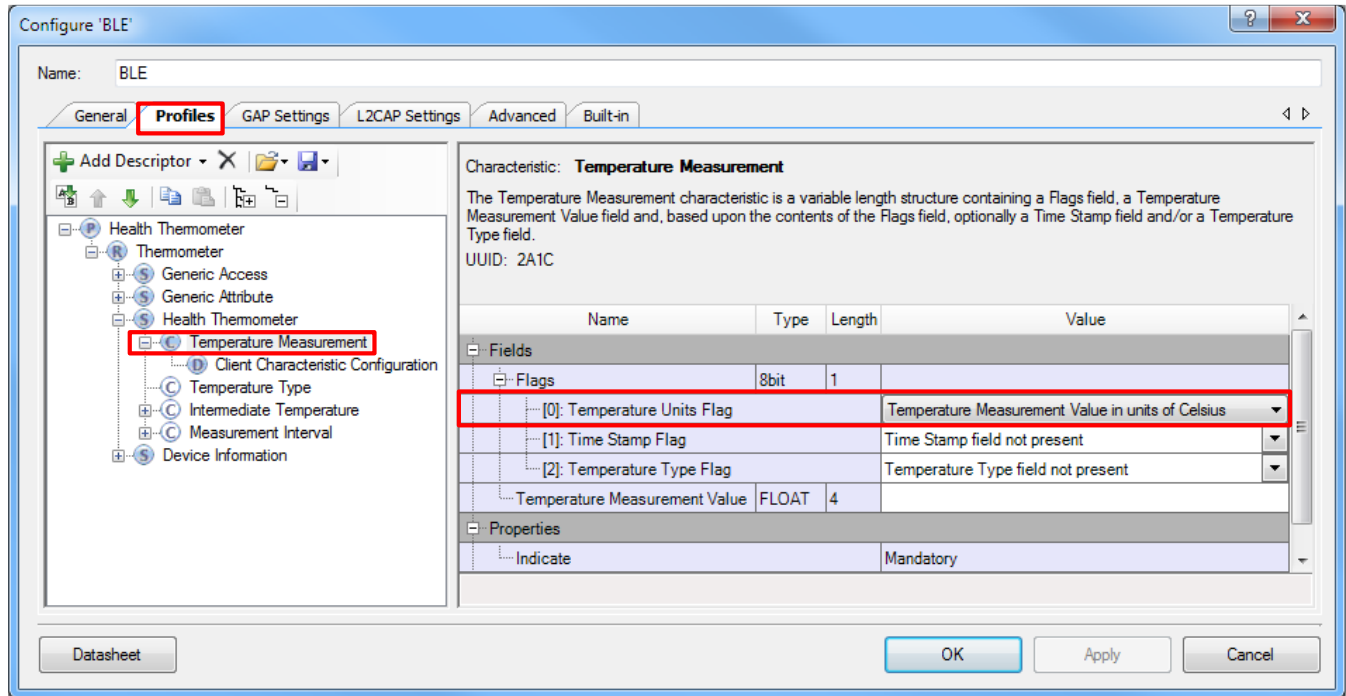
Similarly, update the remaining services and characteristics per Table 1. An "N/A" in the Descriptor column means that the fields and the values refer to the characteristic, but not to the descriptor. For example, the field **Temperature Text Description** belongs to the characteristic **Temperature Type** in the **Health Thermometer** service, but the fields **Lower Inclusive Value** and **Upper Inclusive Value** belong to the descriptor **Valid Range** of the characteristic **Measurement Interval**.

Table 1. Characteristic Configuration

| Service	Characteristic	Descriptor	Fields	Value	Remarks
Health Thermometer	Temperature Measurement	N/A	Temperature Units Flag	Temperature Measurement Value in units of degrees Celsius	The possible values are defined by the Bluetooth SIG. The BLE Component provides an option to select one of the possible values.
	Temperature Type	N/A	Temperature Text Description	Body (general)	The possible values are defined by the Bluetooth SIG. The BLE Component provides an option to select one of the possible values.
	Measurement Interval	N/A	Measurement Interval	1	User-defined Units: seconds, Range: 1-65535
	Measurement Interval	Valid Range	Lower Inclusive Value	1	User-defined Units: seconds, Range: 1-65535
			Upper Inclusive Value	60	User-defined Units: seconds, Range: 1-65535
Device Information	Manufacturer Name String	N/A	Manufacturer Name	Cypress Semiconductor	User-defined
	Model Number String	N/A	Model Number	1.0	User-defined
	Serial Number String	N/A	Serial Number	**	User-defined
	Hardware Revision String	N/A	Hardware Revision	CY8CKIT-042-BLE	User-defined
	Firmware Revision String	N/A	Firmware Revision	**	User-defined
	Software Revision String	N/A	Software Revision	PSoC Creator 4.2	User-defined

Keep the remaining settings at their default values.

6. Configure the **Bluetooth Device Address (BD_ADDR)**, the **Device name**, and the **Appearance** under **General** settings of the **GAP Settings** tab, per Table 2. Figure 9 shows the general GAP settings.

Table 2. General Settings

Name	Value	Remarks
Public address	Check Silicon Generated Address	Use your Company ID and Company-assigned values as the address. If you do not have these details, add the desired address in the field.
Device name	MyThermometer	User-defined
Appearance	Generic Thermometer	The possible values are defined by the Bluetooth SIG. The BLE Component provides an option to select one of the possible values.

Note: Public address refers to the unique 48-bit BD_ADDR that is used to identify the device. It is divided into two parts: Company ID (24 bits) and Company assigned (24 bits). By default, the public address is loaded with the Company ID of Cypress Semiconductor. You should use your 24-bit Company ID assigned by IEEE.

Figure 9. Device Name and Appearance
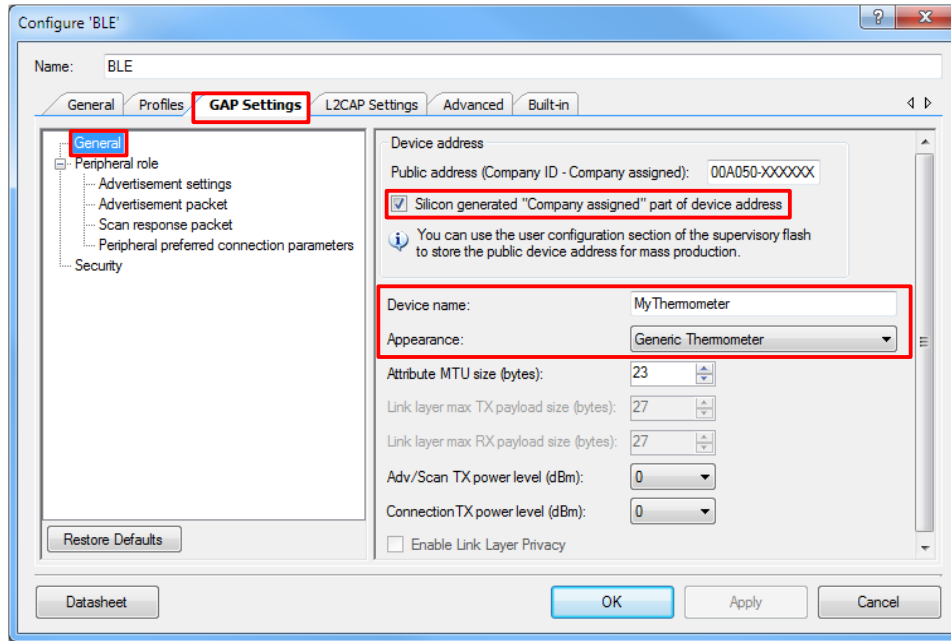
7. The **Advertisement Settings** under **Peripheral Role** will be left at their default values for this project. Configure the **Advertisement packet** settings under **Peripheral role** per Table 3. Figure 10 shows the advertisement packet setting

Table 3. Advertisement Packet Settings

Name	Check Box	Value	Remarks
Local Name	Enabled	Complete	Transmits the complete name as a part of the advertisement packet. You can send the shortened name as well by selecting the number of characters to be sent.
Service UUID	Enabled	N/A	Transmits Service UUIDs (universally unique identifier) as a part of the advertisement packet.
Service UUID > Health Thermometer	Enabled	N/A	Transmits the Health Thermometer Service UUID as a part of the advertisement packet.
Service UUID > Device Information	Enabled	N/A	Transmits the Device Information Service UUID as a part of the advertisement packet.
Appearance	Enabled	N/A	Transmits the Appearance value as a part of the advertisement packet.

Figure 10. GAP Settings – Advertisement Packet
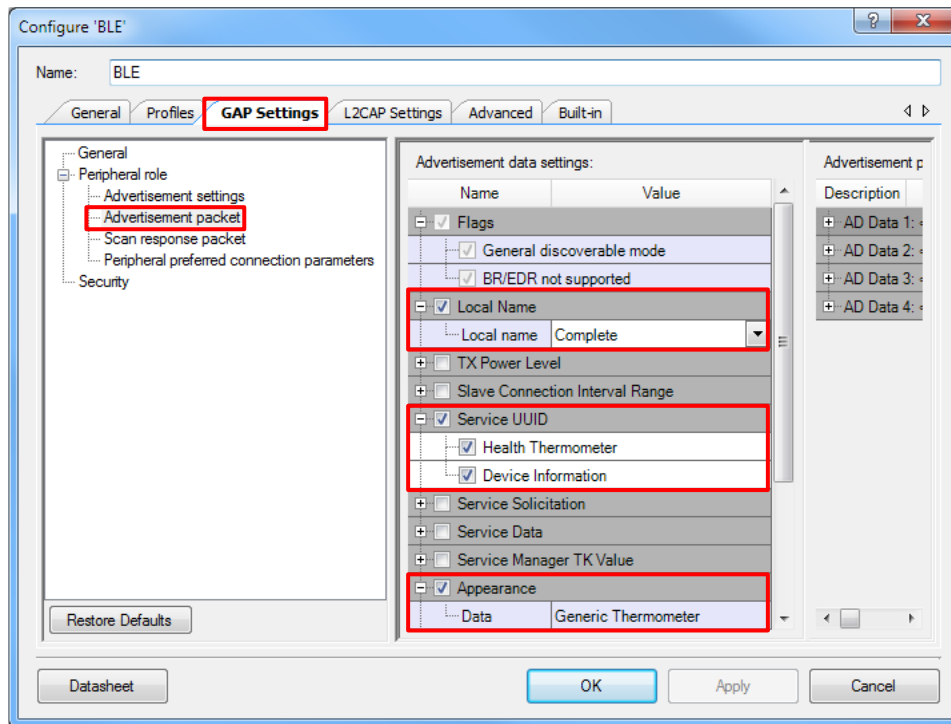
Keep the remaining settings at their default values including the **Scan response packet, Peripheral preferred connection parameters** and **Security** settings. Also, keep the settings in the **L2CAP Settings** tab at their default values.

8. Click **Apply** and then click **OK**.

9. Place a Digital Input Pin Component and configure it as shown in Figure 11 and Figure 12.
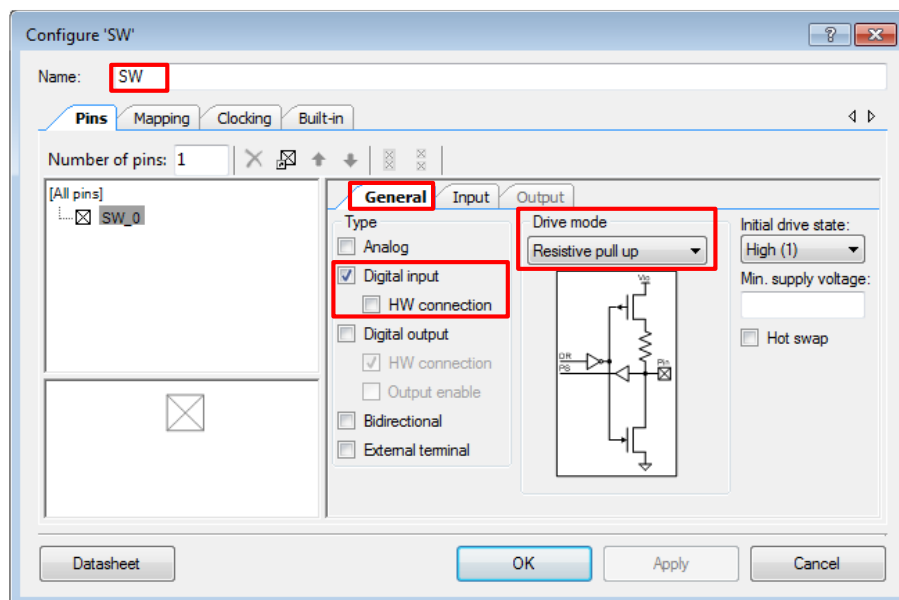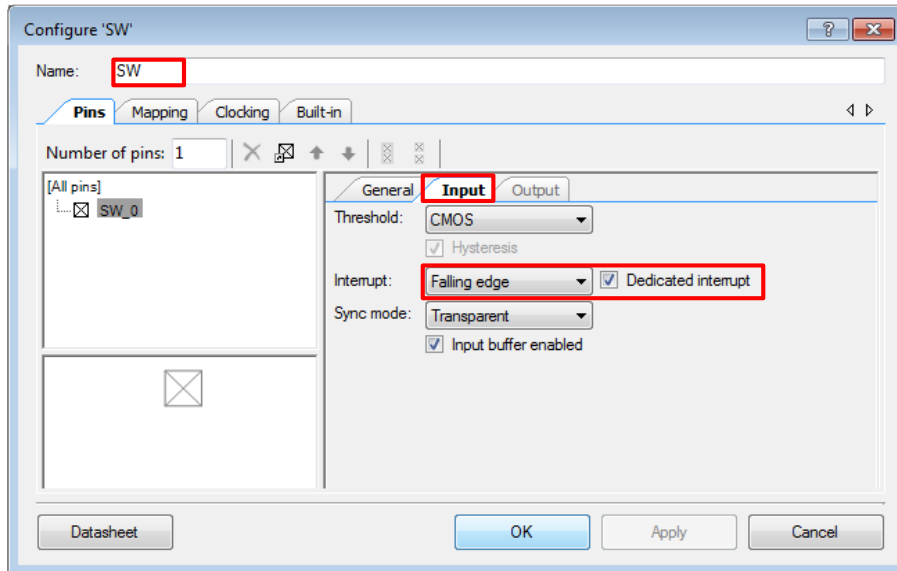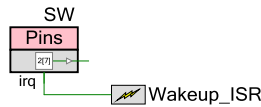
Figure 11. Pin Configuration

Figure 12. Pin Interrupt Configuration

Note: The drive mode for the user switch is selected as **Resistive pull up** to keep the default state of the signal as logic HIGH. When the switch is pressed, the pin is pulled to ground, driving the signal on the pin from logic HIGH to logic LOW. Thus, the interrupt is set for **Falling edge**.

10. Place an Interrupt Component and connect it to the irq pin of the SW Component and rename it as "Wakeup_ISR" as shown in Figure 13. The interrupt Component will be used to record the interrupt signal and trigger the respective function.

Figure 13. SW Pin

11. Drag and drop the Sequencing SAR ADC Component and configure it per Table 4. Figure 14, and Figure 15 show the Sequencing SAR ADC Component configuration settings.

Table 4. SAR ADC Component Configuration

| Tab	Name	Value	Remarks
General	Name	ADC	This name will be used as the prefix to the APIs for the Component.
	Clock frequency (kHz)	6000	This application does not require a high accuracy. A faster sample rate helps conserve power by reducing the active time.
	Vref Select	VDDA	Input voltage range from 0 to V_{DDA}.
	Samples Averaged	16	Sample averaging of 16 helps averaging out any high-frequency noise.
Channels	Sequenced channels	1	Number of channels to be scanned
	AVG	Checked	Enable averaging for the corresponding ADC channel

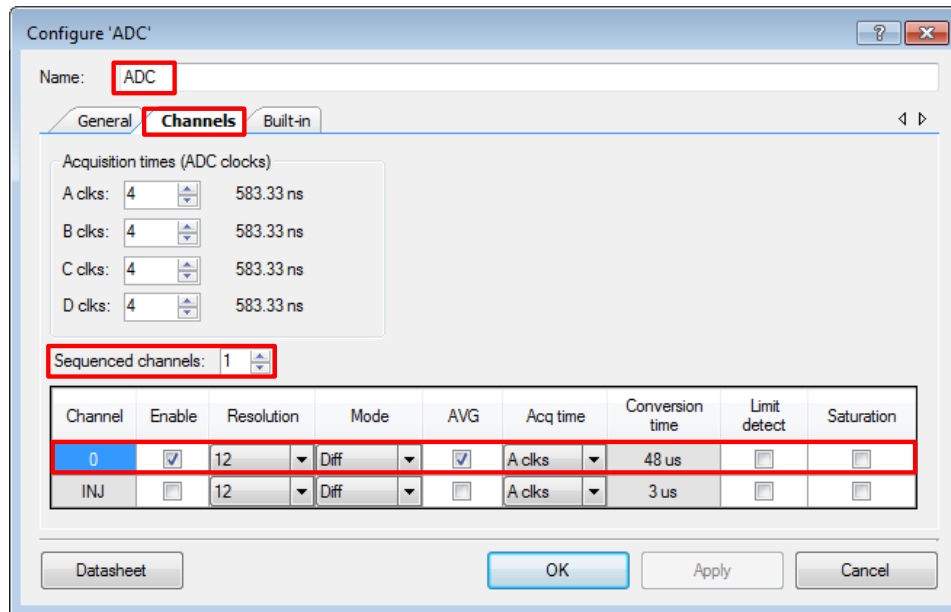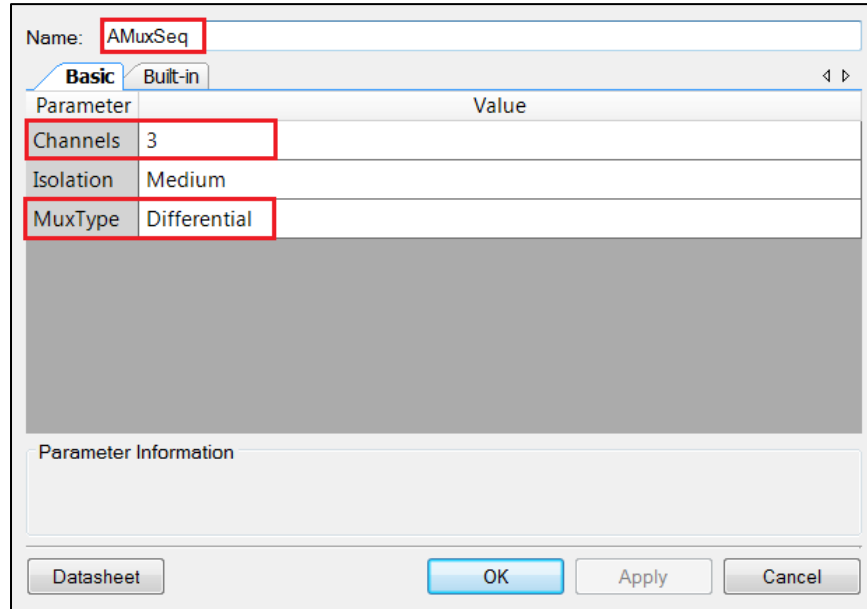Figure 14. SAR ADC Configuration – General
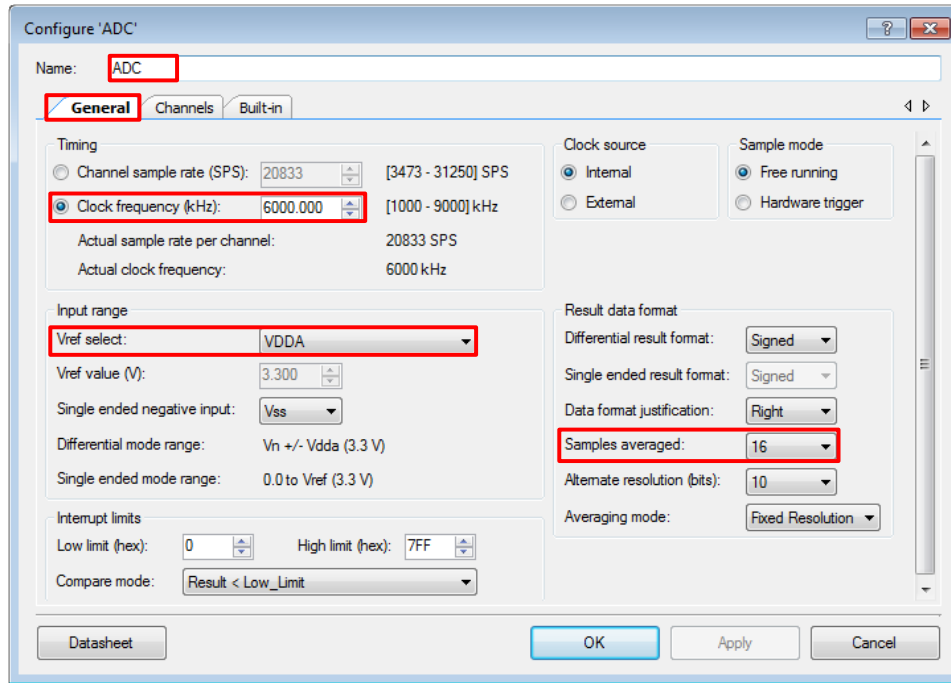
Figure 15. SAR ADC Configuration – Channels

12. To measure temperature, follow steps 11 to 18.

13. Place an Analog Mux Sequencer Component and configure it per Figure 16. The AMUX Component will be used to mux multiple analog signals to the ADC input.
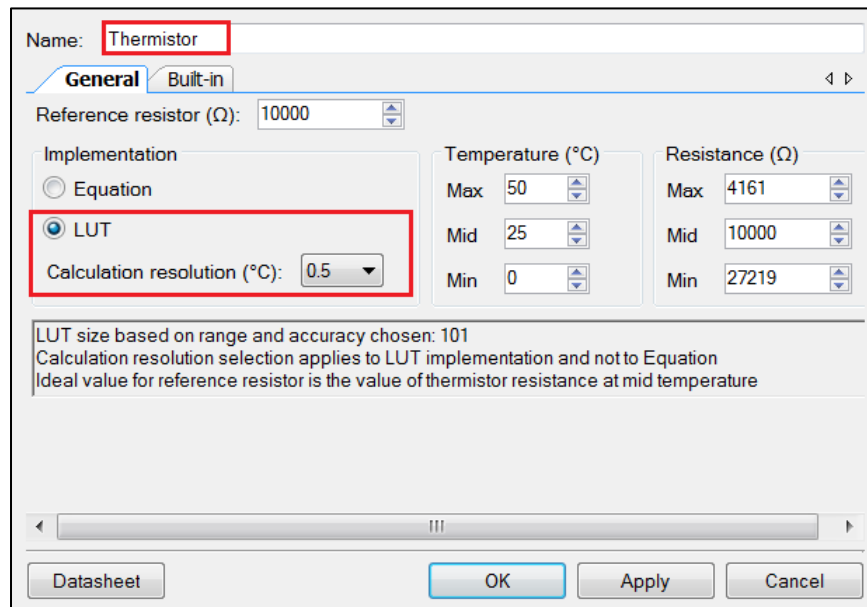
Figure 16. AMUX Configuration

14. Place the Thermistor Calculator Component and configure it per Figure 17. These settings configure this Component for temperature measurement using the thermistor.

Figure 17. Thermistor Calculator Configuration

Note: For accurate temperature measurement, the Temperature (ºC) and the Resistance (Ω) column must be updated using the values given in the thermistor datasheet.

15. Place two Analog Pin Components, and configure them per Figure 18 and Figure 19.

Note: These pins are configured as both analog pins and digital outputs. The digital output will allow us to drive either pin to either V_{DD} or V_{SS} from the firmware.

Figure 18. Analog Pin Configuration – V_HIGH
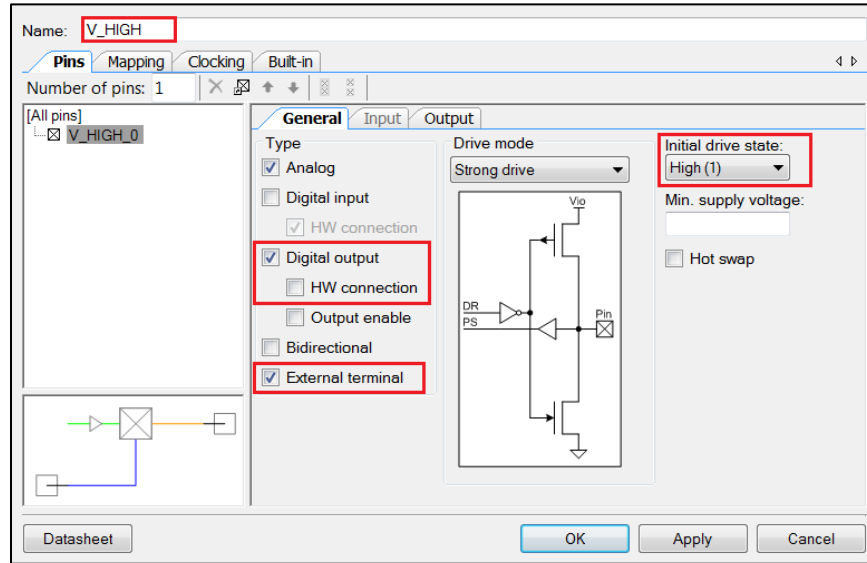
Figure 19. Analog Pin Configuration – V_LOW
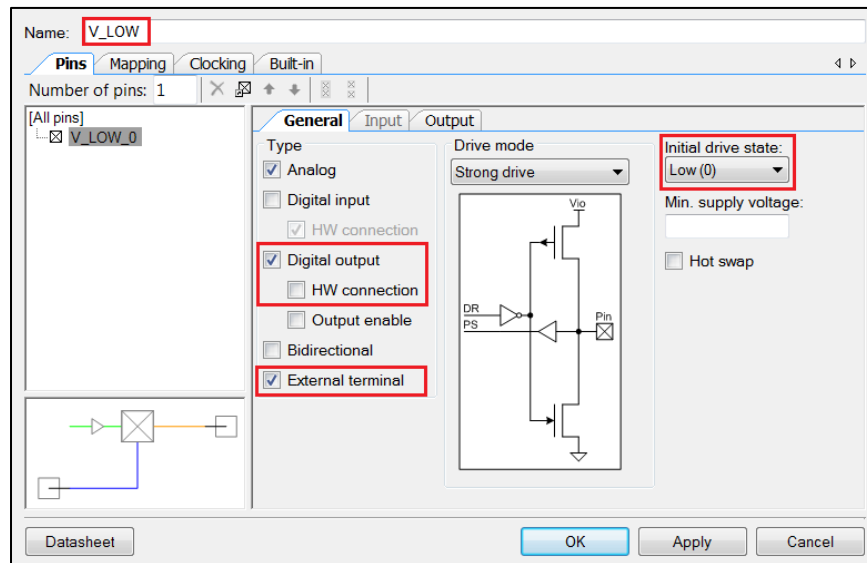
16. Place an Analog Pin Component and name it as "V_THERM." Enable the **External terminal**.

17. Connect the Analog Mux Sequencer, Sequencing SAR ADC, and Analog Pin Components per the schematic shown in Figure 20.

Note: All of the blue components are from the "Off-Chip" catalog. These are used for documentation purposes only but can be used to make the schematic more instructive of what is on the hardware.

Figure 20. Schematic – Temperature Measurement
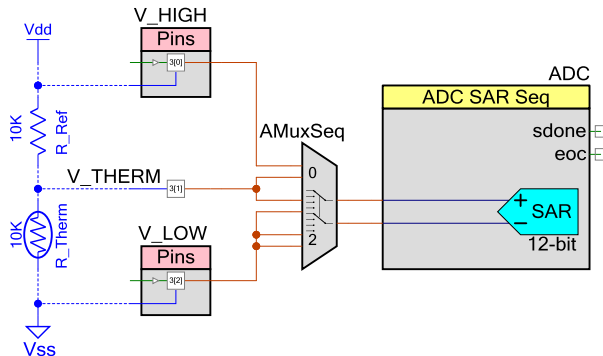
18. In the Pins tab of the design-wide resources window, connect the pins as shown in Figure 21.

Figure 21. Pins Tab of the Design-Wide Resources Window

19. In the Clocks tab of the design-wide resources window, configure the IMO frequency as 12 MHz, as shown in Figure 22.

Figure 22. Clock Configuration
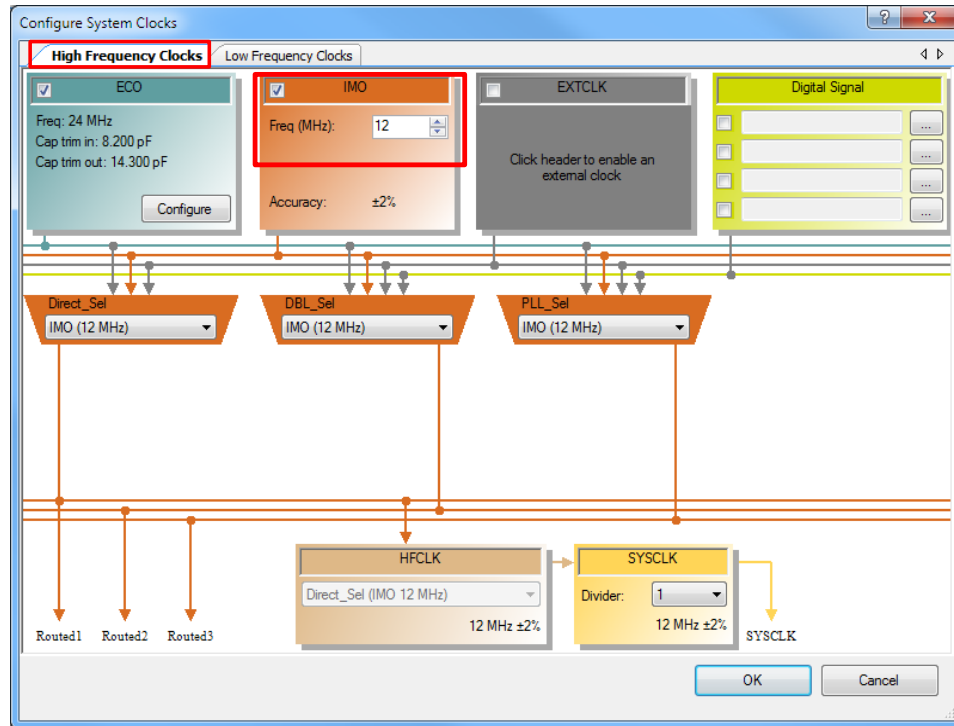
5.2   Configure the Firmware

Figure 23 shows the firmware flow for the Health Thermometer application.

Figure 23. System Flowchart

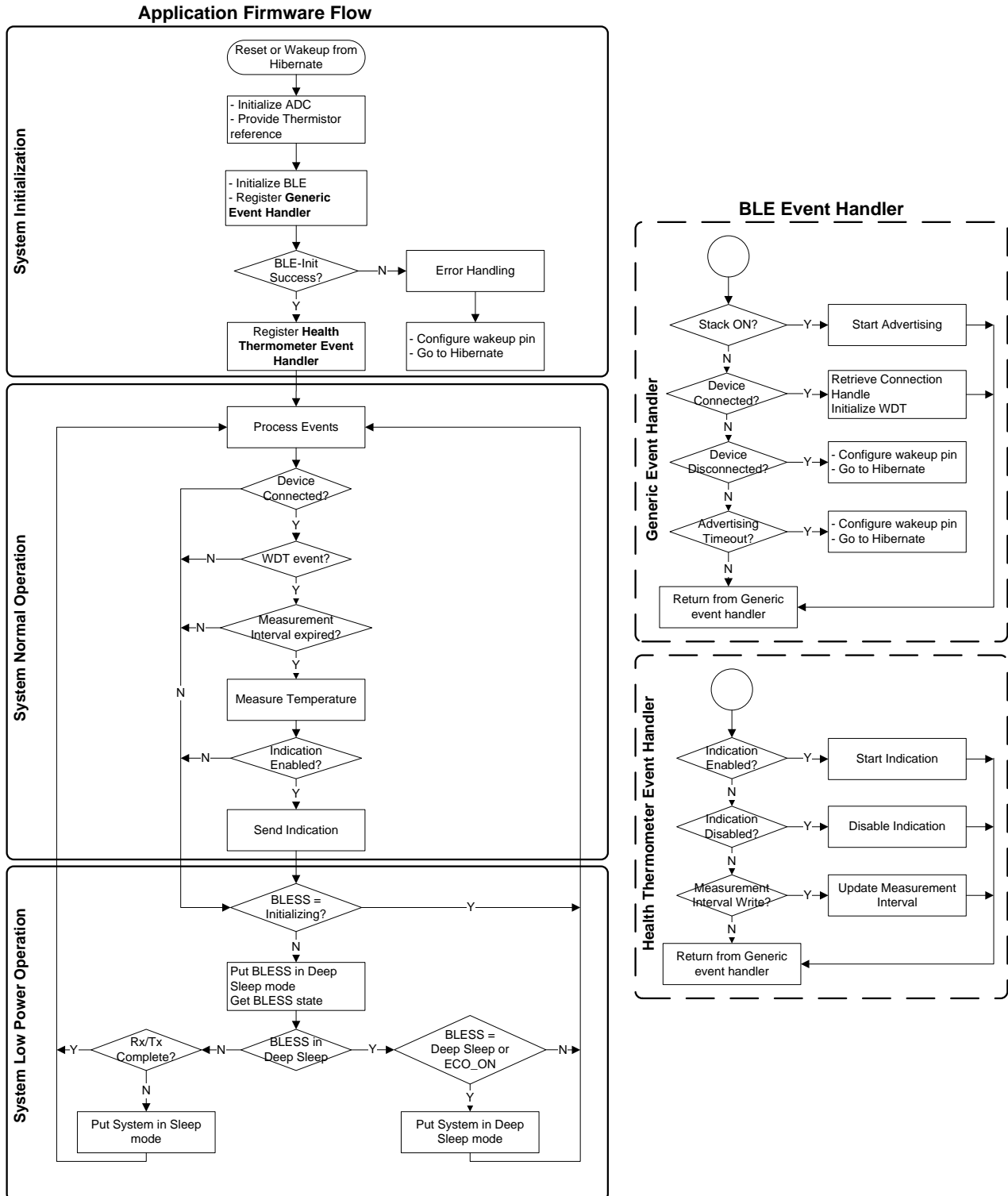**Note:** Source files for the application firmware are in the example project that is included with this application note. You can either include the source files in your own project or you can use the completed example project as-is. There are nine source files for the example project which are listed in Table 5.

Table 5. Example Project Source Files

Name	Description
main.c	This is the main firmware file. It has only one function: ■ `main()` – This function controls the flow of the complete application. Primary tasks performed as a part of this function are initializing the system, application control that includes processing BLE events, application flow control, and low-power implementation.
CommonFunctions.c/.h	Implements the common functions that are used for application control. It has the following functions: ■ `InitializeSystem()` – Initializes all the blocks of the system. ■ `PrepareForDeepSleep` – Prepares the system for low-power operation by putting the hardware blocks in the Deep Sleep mode. ■ `WakupFromDeepSleep()` – Restores the hardware blocks for normal operation.
Temperature.c/.h	It implements the temperature measurement by reading the measured data from the ADC and calculating the temperature using the Thermistor Calculator Component. It also provides an option to the user to simulate the temperature instead of measuring it from the temperature sensor. It has the following functions: ■ `MeasureSensorVoltage()` – This function measures the sensor voltage. This function is not available when the sensor simulation option is selected. For more details, refer to the Sensor Simulation section. ■ `ProcessTemperature()` – This function measures or simulates the temperature value.
WatchdogTimer.c/.h	It implements the watchdog timer functionality and keeps track of the system time. It has the following functions: ■ `WatchdogTimer_Start()` – Starts the watchdog timer (WDT0) with a 1-s period and an interrupt on match. ■ `WatchdogTimer_Isr()` – The ISR for the WDT; it is used to track the measurement interval. This function is a callback from the watchdog timer. ■ `WatchdogTimer_Stop()` – Stops the watchdog timer (WDT0).
BLE_HTSS.c/.h	It handles the BLE-specific functionality of the project. It handles the events generated from the BLE stack, explained in detail in the Event Handler section. It has the following functions: ■ `GenericEventHandler()` – Handles the generic events generated by the BLE stack ■ `HtssEventHandler()` – Handles the events generated for the Health Thermometer Service ■ `ProcessBLE()` – Sends the temperature data as Indication to the GATT Client ■ `ConvertFloatTemp()` – Converts the temperature data from IEEE-754 format to IEEE-11073 format ■ `EnableBLE()` – Starts the BLE Component and registers the event handler functions

The following sections explain the operation of the Health Thermometer application. Note that the complete firmware is not included in this document. Instead, the key concepts are explained in detail. Refer to the included example project for the complete firmware. The Health Thermometer application state machine consists of four states:

■ System Initialization

■ Event Handler
- □ Generic Event Handler
- □ Health Thermometer Event Handler

■ System Normal Operation

■ System Low-Power Operation

These states are discussed in detail in the following sections.

5.2.1 System Initialization

When the device is reset or wakes up from the Hibernate mode, the firmware performs initialization, which includes starting the SAR ADC, enabling global interrupts, starting the opamps, and starting the watchdog timer. After the system is initialized, it initializes the BLE Component, which handles the initialization of the complete BLE subsystem.

Note: As part of the BLE Component initialization, the user code must pass a pointer to the event-handler function that should be called to receive events from the BLE stack. The Generic Event Handler shown in Figure 23 is registered as a part of the BLE initialization. Code 1 shows the code to start the BLE Component and register the Generic Event Handler.

Code 1. BLE Initialization

```
apiResult = CyBle_Start(GenericEventHandler);
```

If the BLE Component initializes successfully, the firmware registers the function that is called to receive the events for the Health Thermometer Service and switches to the normal operation mode. Code 2 shows the snippet for registering the Health Thermometer Service.

Code 2. Health Thermometer Service Event Handler

```
CyBle_HtsRegisterAttrCallback(HealthThermometetEventHandler);
```

5.2.2 Event Handler

In the BLE Component, results of any operation performed on the BLE stack are relayed to the application firmware via a list of events. These events provide the BLE interface status and data. Events can be categorized as follows:

- Common events

  Operations performed at the GAP layer, the GATT layer, and the stack's L2CAP layer generate these events. For example, a CYBLE_EVT_STACK_ON event is received when the BLE stack is initialized and turned ON, a CYBLE_EVT_GAP_DEVICE_CONNECTED event is received when a connection with a remote device is established, and a CYBLE_EVT_GATTS_WRITE_CMD_REQ event is generated when a Write Command is received from the client. For more details on common events, refer to the API documentation of the BLE Component (right-click the BLE Component in PSoC Creator and select **Open API Documentation**).

  The application firmware must include an event handler function to successfully establish and maintain the BLE link. Code 3 shows the implementation of the GenericEventHandler function, where events generated on the initialization of the BLE stack, device connection, disconnection, and timeout are handled.

- Service-specific events

  Service-specific events are generated because of operations performed on the standard services defined by the Bluetooth SIG. For example, a CYBLE_EVT_HTSS_INDICATION_ENABLED event is received by the server when the client writes the client configuration characteristic descriptor to enable the indication for the Temperature Measurement Characteristic. For more details on service-specific events, refer to the API documentation of the BLE Component.

  The BLE Component can route these events to a service-specific event handler. The application firmware should include a service-specific event handler function to handle these events. If a service-specific event handler is not supported, then these events must be handled by the common event handler (GenericEventHandler). Code 4 shows the implementation of the service-specific event handler called HtssEventHandler.

Code 3. Generic Event Handler

```c
void GenericEventHandler(uint32 event, void *eventParam)
{
  switch(event)
  {
    /* This event is received when component is Started */
    case CYBLE_EVT_STACK_ON:
    {
      /* Stop watchdog to reduce power consumption during advertising */
      WatchdogTimer_Stop();
      /* Start Advertisement and enter Discoverable mode*/
      CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
      break;
    }

    /* This event is received when device is disconnected or advertising times out*/
    case CYBLE_EVT_GAP_DEVICE_DISCONNECTED:
    case CYBLE_EVT_TIMEOUT:
    {
      /* Sets the ENABLE_HIBERNATE flag to put system in Hibernate mode */
      SystemFlag |= ENABLE_HIBERNATE;
      break;
    }

    /* This event is received when connection is established */
    case CYBLE_EVT_GATT_CONNECT_IND:
    {
        /* Start watchdog timer with 1s refresh interval */
        /* Note: For this application, wakeup should be 1s because htssInterval
        *  resolution is configured as 1s */
        WatchdogTimer_Start(REFRESH_INTERVAL);
        /* Retrieve BLE connection handle */
        connectionHandle = *(CYBLE_CONN_HANDLE_T *) eventParam;
        break;
    }

    default:
    {
      /* Error handling */
      break;
    }
  }
}
```

Code 4. Health Thermometer Service Event Handler

```c
void HtssEventHandler(uint32 event, void* eventParam)
{
  CYBLE_HTS_CHAR_VALUE_T *interval;
  switch(event)
  {
    /* This event is received when indication are enabled by the central */
    case CYBLE_EVT_HTSS_INDICATION_ENABLED:
    {
      /* Set the htssIndication flag */
      htssIndication = true;
      break;
    }
```

```
        /* This event is received when indication are disabled by the central */
        case CYBLE_EVT_HTSS_INDICATION_DISABLED:
        {
          /* Reset the htssIndiciation flag */
          htssIndication = false;
          break;
        }

        /* This event is received when measurement interval is updated by
         *  the central */
        case CYBLE_EVT_HTSS_CHAR_WRITE:
        {
          /* Retrive interval value */
          interval = ((CYBLE_HTS_CHAR_VALUE_T *)eventParam);
          htssInterval = interval->value->val[1];
          /* Update htssInterval with the updated value */
          htssInterval = (htssInterval << 8) | interval->value->val[0];
          break;
        }

        default:
        {
          /* Error handling */
          break;
        }
      }
    }
```

5.2.3 System Normal Operation

In the system normal operation state, the firmware periodically calls `CyBle_ProcessEvents()` to process BLE stack-related operations and checks if the connection is established.

Note: Any BLE stack-related operation such as receiving or sending data from or to the link layer and event generation to the application layer are performed as a part of the `CyBle_ProcessEvents()` function call. In this application, Code 1 initializes the stack, but the events related to the stack are generated only when the `CyBle_ProcessEvents()` function is called. Similarly, other events related to device connection, disconnection, advertising timeout, and the Health Thermometer Service are generated only when `CyBle_ProcessEvents()`is called.

If the connection is established, the firmware measures the temperature at regular intervals (configured by the Measurement Interval Characteristic of the Health Thermometer Service). After measuring the temperature, if Indications are enabled by the Central device, the firmware sends the temperature data to the BLE Central device as indications.

In a BLE application, the device transmits or receives data only at periodic intervals, also known as advertising intervals or connection intervals, depending on the BLE connection state. Thus, when the system normal operation task is complete, to conserve power, the device enters the system low-power operation mode and wakes up at the next connection/advertisement interval.

5.2.4 System Low-Power Operation

In the system low-power operation state, the device operates in one of the three possible power modes:

- Sleep
  This mode is entered when the CPU is free but the BLE subsystem (BLESS) is active and busy in data transmission or reception. In this scenario, the CPU is put into the Sleep mode while the remaining core, such as clocks and regulator, is kept active for normal BLE operation. To conserve power, the internal main oscillator (IMO) frequency is reduced to 3 MHz; on wakeup, it is switched back to 12 MHz.

- Deep Sleep
  The firmware continuously tries to put the BLESS into the Deep Sleep mode. After the BLESS is successfully put into the Deep Sleep mode, the remaining system also transitions to the Deep Sleep mode.

Note: Transitioning the device into the Deep Sleep mode should happen immediately after the BLESS is put into the Deep Sleep mode. If this cannot be guaranteed, the firmware should disable interrupts (to avoid servicing an ISR) and recheck if the BLESS is still in the Deep Sleep mode or the ECO_ON mode. If the BLESS is in either of these two modes, then the device can safely enter the Deep Sleep mode; if not, the device must wait until the Rx/Tx event is complete.

■ Hibernate

When the device is disconnected or the advertising interval times out, it enters the ultra-low-power mode called "Hibernate". After waking up from this mode, the firmware starts to execute from the beginning of *main.c*, although the RAM contents are retained.

5.2.5 Sensor Simulation

If you do not have a thermistor and a reference resistor to measure the temperature, you may use the temperature simulation mode to test the application. In this mode, the temperature data is simulated and incremented by 1 ^{0}C per measurement interval (default value is 1 second). This can be done in the application code by changing the value of the constant MEASURE_TEMPERATURE_SENSOR from 1 to 0 in the *Temperature.h* file, as shown in Code 5.

Code 5. Simulate Temperature Sensor

```
#define MEASURE_TEMPERATURE_SENSOR (0u)
```

5.3   Hardware Configuration

CY8CKIT-042-BLE Bluetooth Low Energy (BLE) Pioneer Kit is the development kit for Cypress PSoC 4 BLE devices. Designed for flexibility, this kit offers footprint-compatibility with many third-party Arduino™ shields. The kit includes a provision to populate an extra header to support Digilent® Pmod™ peripheral modules. In addition, the board features a CapSense® slider, an onboard 1-Mb F-RAM, an RGB LED, a push-button switch, an integrated USB programmer, a program debug header, and USB-UART/I²C bridges.

1. Place the CY8CKIT-142 PSoC 4 BLE Module (red module) on the BLE Pioneer Baseboard.

   This kit does not have a thermistor, so to test the Health Thermometer application, you can use the BLE Pioneer Kit along with one of the following options:

   ■ External Thermistor – Connect an external 10-kΩ thermistor and a reference resistor to the BLE Pioneer Kit as shown in Figure 24.

Figure 24. External Thermistor Connection
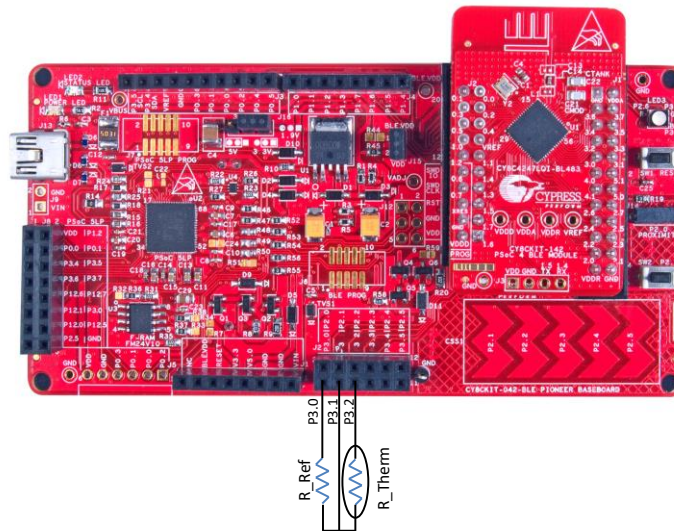
   ■ Simulated Sensor – Instead of using a hardware sensor, the firmware simulates the temperature data and increments the temperature by 1 ^{0}C per measurement interval (default value is 1 second). Refer to the Sensor Simulation section for details.

5.4 Build and Program the Device

1. Select **Build** > **Build AN91184** to build and compile the firmware, as shown in Figure 25. The project should build without warnings or errors.

Figure 25. Build Project
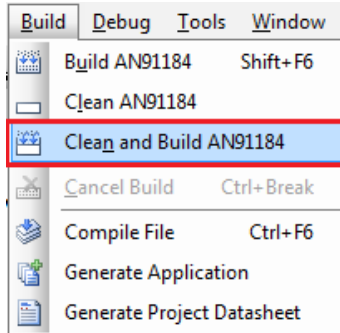
2. Plug the PSoC 4 BLE module (red module) to the BLE Pioneer baseboard, and then connect the kit to your PC using the USB Standard-A to Mini-B cable (see Figure 26). Allow the USB enumeration to complete on the PC.

Figure 26. Connect BLE Pioneer Baseboard to PCB Using a USB Cable
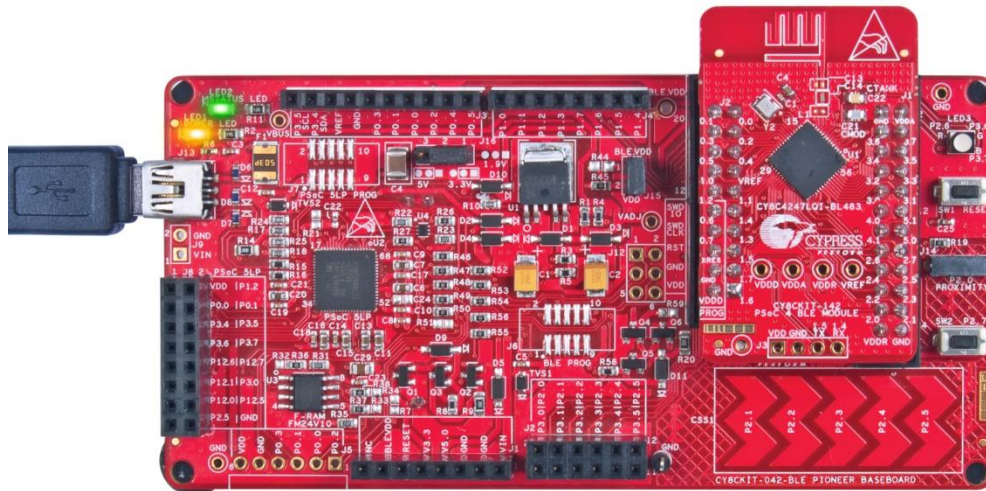
3. Select **Debug** > **Program,** as shown in Figure 27. If there is only one kit connected to the PC, the programming will start automatically. If multiple kits are present, PSoC Creator will prompt you to choose the kit to be programmed.

Figure 27. Programming the Device
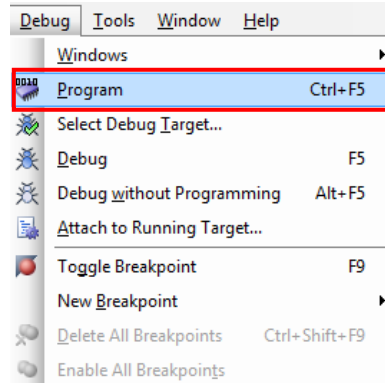
You can view the programming status on the PSoC Creator status bar (lower-left corner of the window), as shown in Figure 28.

Figure 28. Programming Status
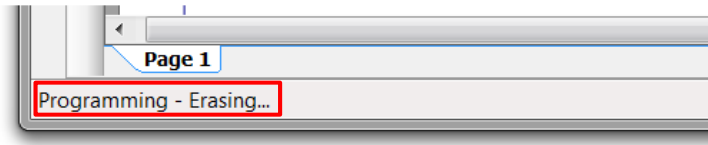
6   Application Testing

The Health Thermometer application can be tested using the CySmart Central Emulation Tool or the CySmart mobile app.

6.1   CySmart Central Emulation Tool

You can use the CySmart Central Emulation Tool along with the CY5670 or CY5677 CySmart USB Dongle (BLE Dongle) to test and verify the operation of a BLE Peripheral device.

Download the latest CySmart Central Emulation Tool from www.cypress.com/cysmart.

To verify the Health Thermometer application using the CySmart Central Emulation Tool, do the following:

1. Connect the BLE Dongle to the PC and start the CySmart Central Emulation Tool from **Start** > **All Programs** > **CySmart 1.3** > **CySmart 1.3.**

   The CySmart Central Emulation Tool detects the BLE Dongle connected to the USB drive.
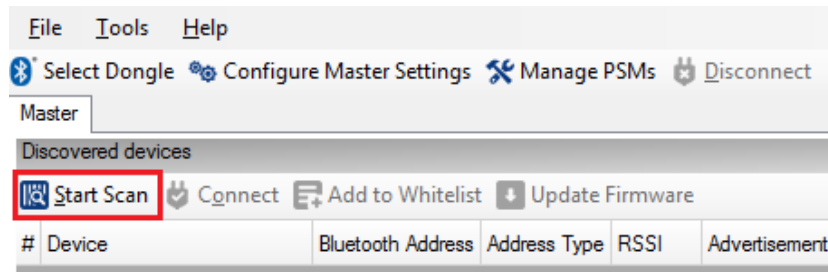
2. Click **Connect** to connect to the BLE Dongle, as shown in Figure 29.

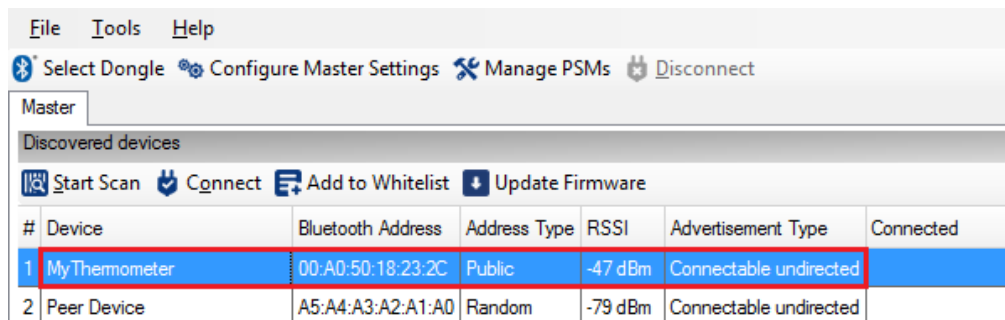Figure 29. Select BLE Dongle Target
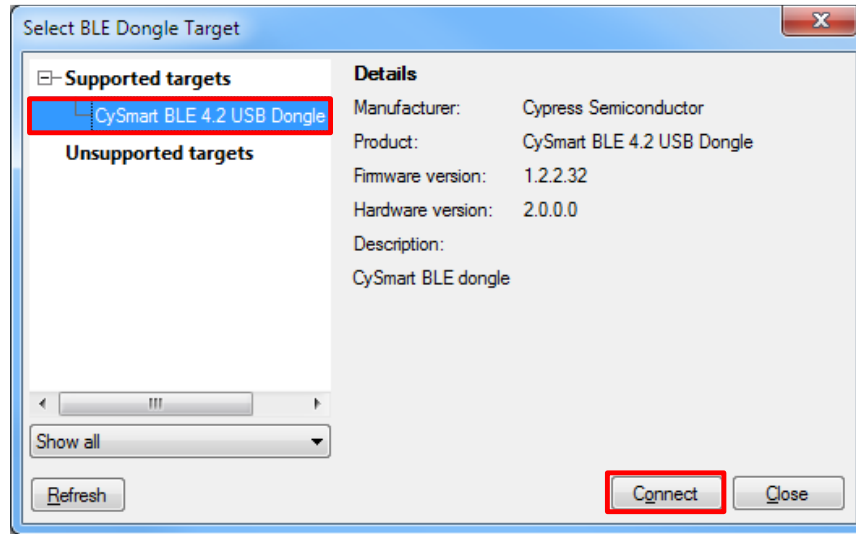
3. When the PC is connected with the BLE Dongle, click **Start Scan** to find the BLE devices, as shown in Figure 30.

Figure 30. Start Scan

The discovered BLE devices are listed in the CySmart Central Emulation Tool window, as shown in Figure 31.

Figure 31. Discovered Devices

4. Select the device "MyThermometer" by clicking on the name, and then click **Connect**, as shown in Figure 31.

When the BLE Dongle is connected to the BLE device, a new tab with the device name and the BD_ADDR is added, as shown in Figure 32.

Figure 32. Connect to MyThermometer
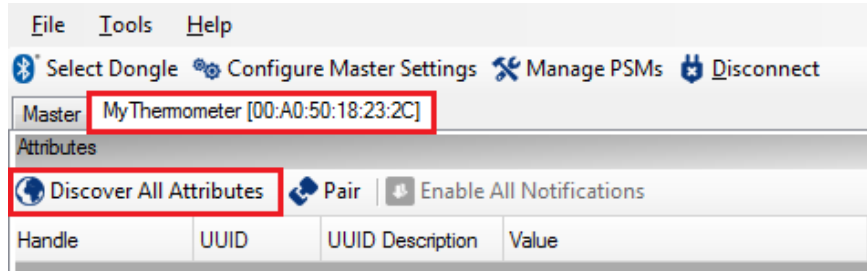
5. To discover all the attributes exposed by the Peripheral "MyThermomemeter," click **Discover All Attributes**, as shown in Figure 32.

All discovered attributes are grouped and displayed as shown in Figure 33.

Figure 33. Discovered Attributes
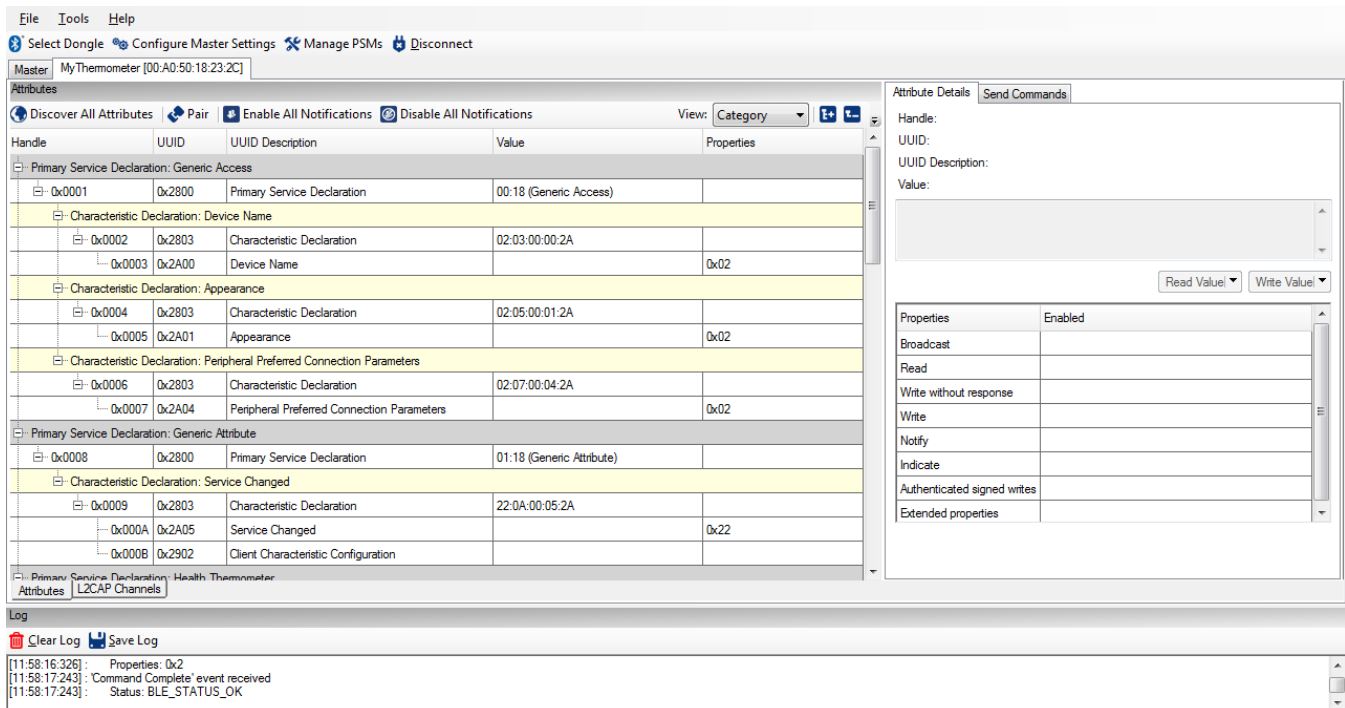
6. To read the measured temperature, select the **Client Characteristic Configuration** descriptor of the **Temperature Measurement** characteristic, as shown in Figure 34 and Figure 35.

Figure 34. Enable Indications – Step A

⊟ Primary Service Declaration: Health Thermometer				
⊟ 0x000C	0x2800	Primary Service Declaration	09:18 (Health Thermometer)	
⊟ Characteristic Declaration: Temperature Measurement				
⊟ 0x000D	0x2803	Characteristic Declaration	20:0E:00:1C:2A	
0x000E	0x2A1C	Temperature Measurement		0x20
0x000F	0x2902	Client Characteristic Configuration		
⊟ Characteristic Declaration: Temperature Type				

7. Write a value of **02** in the **Client Characteristic Configuration** descriptor to enable indications (measured temperature data will be reported by indications), as shown in Figure 35.

Note: Refer to Client Characteristic Configuration Descriptor for more details about bit definitions.

Figure 35. Enable Indications – Step B

The "Temperature Measurement" attribute is updated with the measured temperature value, as shown in Figure 36.

Figure 36. Measured Temperature

⊟ Primary Service Declaration: Health Thermometer				
⊟ 0x000C	0x2800	Primary Service Declaration	09:18 (Health Thermometer)	
⊟ Characteristic Declaration: Temperature Measurement				
⊟ 0x000D	0x2803	Characteristic Declaration	20:0E:00:1C:2A	
0x000E	0x2A1C	Temperature Measurement	00:F0:00:00:FF	0x20
0x000F	0x2902	Client Characteristic Configuration	02:00	
⊟ Characteristic Declaration: Temperature Type				
⊟ 0x0010	0x2803	Characteristic Declaration	02:11:00:1D:2A	
0x0011	0x2A1D	Temperature Type		0x02

The value read back is a 5-byte hex value, where first byte is the Flags configured in the BLE Component and the next 3-byte value is the temperature value represented in the Little Endian format for IEEE-11073 float values. The last byte represents the decimal points – in this case 0xFF represents one decimal point. Thus, the measured temperature value is 24.0 ºC (0xF0 is 240, and one decimal point gives 24.0).

To learn more about the CySmart Central Emulation Tool, refer to CySmart User Guide.

6.2 CySmart Mobile App

Cypress provides a mobile app to validate BLE applications. This app supports various standard and custom profiles. It also provides a user interface to be able to view the GATT database.

You can download the CySmart app from the Apple App Store for iOS devices and through the Google Play Store for Android devices. The source code for these apps is also available at Cypress website.

- Apple App Store: Click here.

- Google Play Store: Click here.

To verify the Health Thermometer application using the CySmart mobile app, follow the steps below:

8. Open the CySmart app on your device, as shown in Figure 37.

    Note: The screenshots are for the CySmart Android app. The look and feel of the CySmart iOS app may differ slightly.

Figure 37. BLE Configuration

9. If Bluetooth is enabled, the mobile device scans for BLE devices and lists them on the screen; otherwise, it prompts the user to enable Bluetooth and then searches for the BLE devices. Figure 38 shows the BLE devices in vicinity.

Figure 38. Device List
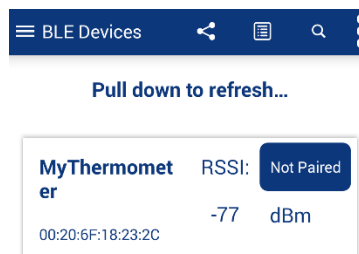
10. Connect to the device "MyThermometer" by clicking on the device name, as shown in Figure 38.

11. Once the connection is established, the app will automatically discover all the attributes and display the discovered services in the carousel format, as shown in Figure 39.

Figure 39. Home Page

12. Select the Health Thermometer Service. It reports the current temperature and the Sensor Location, as shown in Figure 40.
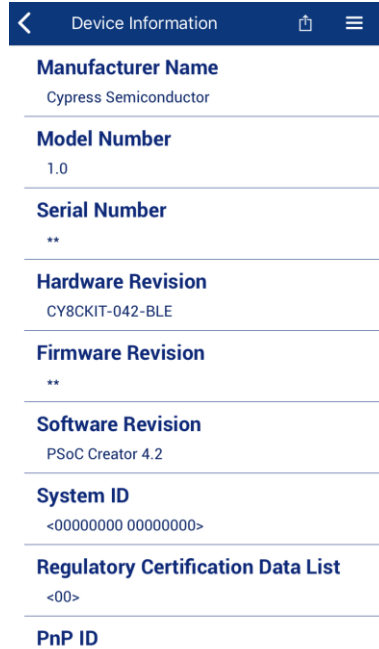
Figure 40. Health Thermometer Service

13. To go back to the home screen of the CySmart app, click on the back button of the screen.

14. Select the "Device Information" Service. It shows the device information configured as a part of the project, as shown in Figure 41.

Figure 41. Device Information Service

6.3 Summary

In this application note, we looked at how to use the PSoC Creator BLE Component to design a BLE Health Thermometer application using the standard BLE profile. We then verified this application using the CySmart Central Emulation Tool and CySmart mobile app provided by Cypress.

7   Related Documents

Application Notes	
AN91267 – Getting Started with PSoC 4 BLE	This application note helps you explore the PSoC 4 BLE architecture and development tools and shows how easily you can create a BLE design using PSoC Creator™, the development tool for PSoC 4 BLE.
AN91162 – Creating a BLE Custom Profile	Describes the methodology for developing a Bluetooth® Low Energy (BLE) application with PSoC 4 BLE or PRoC BLE devices using a custom BLE profile.
AN92584 – Designing for Low Power and Estimating Battery Life for BLE Applications	Describes low-power applications design with PSoC 4 BLE devices. It also guides you on how to compute the current consumption and battery life for a BLE application and provides tips and tricks to minimize the current consumption to increase battery life.
AN66477 – PSoC® 3, PSoC 4, and PSoC 5LP - Temperature Measurement with a Thermistor	Describes how to measure temperature with a thermistor using PSoC® 3, PSoC 4, or PSoC 5LP. This application note describes the PSoC Creator™ Thermistor Calculator Component, which simplifies the math-intensive resistance-to-temperature conversion.
AN210781 – Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity	Describes PSoC 6 MCU with BLE Connectivity devices and how to build your first PSoC Creator project
AN215656 – PSoC 6 MCU: Dual-CPU System Design	Describes the dual-CPU architecture in PSoC 6 MCU, and shows how to build a simple dual-CPU design
AN219434 – Importing PSoC Creator Code into an IDE for a PSoC 6 MCU Project	Describes how to import the code generated by PSoC Creator into your preferred IDE
PSoC Creator Component Datasheets	
Pins	Supports connection of hardware resources to physical pins
Timer Counter (TCPWM)	Supports fixed-function Timer/Counter implementation
Clock	Supports local clock generation
Interrupt	Supports generating interrupts from hardware signals
Device Documentation	
PSoC® 4: PSoC 4200_BLE Family Datasheet Programmable System-on-Chip (PSoC®)	
PSoC 6 MCU: PSoC 63 with BLE Datasheet	
PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual	
PSoC 6 MCU: PSoC 62 Datasheet	
PSoC 6 MCU: PSoC 62 Architecture Technical Reference Manual	
Development Kit Documentation	
CY8CKIT-042-BLE Bluetooth Low Energy Pioneer Kit Guide	
CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit	
CY8CKIT-062-WiFi-BT PSoC 6 WiFi-BT Pioneer Kit	
CY8CPROTO-062-4343W PSoC 6 Wi-Fi BT Prototyping Kit	
CY8CPROTO-063 BLE PSoC 6 BLE Prototyping Kit	
Tool Documentation	
PSoC Creator	Look in the downloads tab for Quick Start and User Guides

Appendix A. Code examples

PSoC Creator's "Find Code Example" browser can be used to find and download code examples related to BLE.
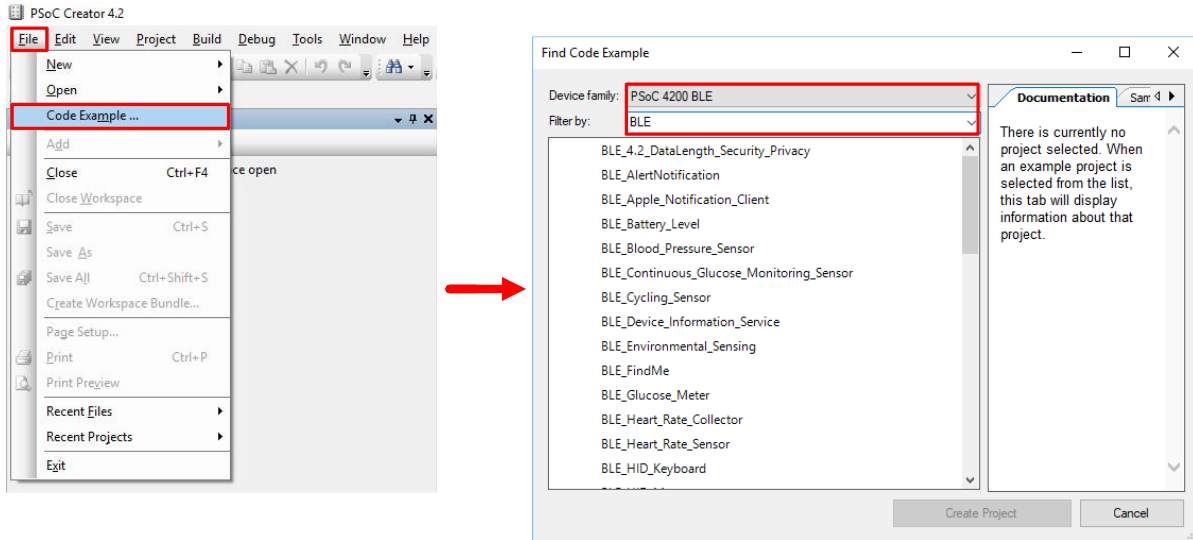
The following table provide summary of the BLE code example that can be found using above.

Code Example	Purpose	GAP role
BLE Find Me	This example project demonstrates the Find Me Profile operation of the BLE Component. The Find Me Target utilizes the Find Me Profile with one instance of Immediate Alert Service to display the alerts if the Client has configured the device for alerting.	Peripheral
BLE Device Information Service	This code example demonstrates how to configure and use BLE Component APIs and an application layer callback. The Device Information Service is used as an example to demonstrate how to configure the BLE service characteristics in the BLE Component.	Peripheral
BLE 4.2 Data Length Security Privacy	This example project demonstrates the BLE 4.2 data length extension, Authenticated LE Secure Connections (SC) pairing with encryption, and link layer Privacy (LL Privacy).	Peripheral
BLE Alert Notification Profile	This example project demonstrates the Alert Notification Client operation of the BLE Component. The Alert Notification Client utilizes the BLE Alert Notification Profile with one instance of Alert Notification Service to receive information about "Email", "Missed call" and "SMS/MMS" alerts from Alert Notification Server.	Peripheral
BLE Apple Notification Client	This example project demonstrates the BLE Apple Notification Client application workflow. The application uses the BLE Apple Notification Center Service in GATT Client mode to communicate with a BLE Apple Notification Center Server (iPhone, iPod, etc.).	Peripheral
BLE Battery Level	This project demonstrates measurements of the battery voltage using PSoC 4 BLE's internal ADC and notifies the BLE central device of any change in the battery voltage using BLE Battery Alert Service.	Peripheral
BLE Blood Pressure Sensor	This example project demonstrates the BLE Blood Pressure Sensor application workflow. The Blood Pressure Sensor application utilizes the BLE Blood Pressure profile to report blood pressure measurement records to the Client.	Peripheral
BLE Continuous Glucose Monitoring Sensor	This example project demonstrates the BLE Continuous Glucose Monitoring Sensor application workflow. The application uses the BLE Continuous Glucose Monitoring Profile to report CGM Measurement records to the Client by the Continuous Glucose Monitoring Service and to manage bonding by the Bond Management Service.	Peripheral

Code Example	Purpose	GAP role
BLE Cycling Sensor	This example demonstrates the Cycling Speed and Cadence Service (CSCS) and Cycling Power Service (CPS). Cycling Speed and Cadence simulates a cycling activity and reports the simulated cycling speed and cadence data to a BLE central device using CSCS. Cycling Power simulates cycling power data and reports the simulated data to a BLE central device using CPS.	Peripheral
BLE Environmental Sensing Profile	This example project demonstrates the Environmental Sensing Profile operation of the BLE Component. The Environmental Sensor utilizes the Environmental Sensing Profile with one instance of Environmental Sensing and Device Information Services to simulate wind speed measuring.	Peripheral
BLE Glucose Meter	This example project demonstrates the BLE Glucose Meter application workflow. The Glucose Meter application uses the BLE Glucose Profile to report glucose measurement records to the client. Also, the Glucose Meter application uses the Battery Service to notify the battery level and the Device Information Service.	Peripheral
BLE Heart Rate Collector	This example project demonstrates the BLE Heart Rate Collector workflow. The project receives Heart Rate data from any BLE enabled Heart Rate Sensor and indicates that data on any terminal software via UART.	Central
BLE Heart Rate Sensor	This example project demonstrates the BLE Heart Rate Sensor workflow. The project simulates Heart Rate data and performs communication with BLE enabled central/client device.	Peripheral
BLE HID Keyboard	This project demonstrates keyboard pressing in the boot and protocol mode. The example also demonstrates handling a suspend event from the central device and entering the low power mode when suspended.	Peripheral
BLE HID Mouse	This project demonstrates the mouse movement and button click HID reports in the boot and protocol mode. The example also demonstrates handling the suspend event from the central device and enters the low power mode when suspended.	Peripheral
BLE IPSP Router and Node	This example project demonstrates the Internet Protocol Support Profile operation of the BLE Component. This example demonstrates how to setup an IPv6 communication infrastructure between two devices over a BLE transport using L2CAP channel. The example consists of two projects: IPSP Router (GAP Central) and IPSP Node (GAP Peripheral). Router sends generated packets with different content to Node in the loop and validates them with the afterwards received data packet. Node simply wraps received data coming from Router, back to the Router.	Central and Peripheral
BLE Navigation	This example project demonstrates the Location and Navigation Pod application workflow. The application uses a BLE Location and Navigation Profile to report location and navigation information to the client.	Peripheral
BLE OTA External Memory Bootloadable and Bootloader	This example project demonstrates an OTA firmware update using the BLE Bootloader Service. By default, this is a regular bootloadable project that contains the BLE component with the Device Information Service. Once the bootloader mode is enabled, this example project is ready for receiving a new image of the bootloadable project and storing it to the external memory.	Peripheral
BLE OTA Fixed Stack Bootloader and Bootloadable	This example project shows how to use the custom linker scripts to share a block of memory between the bootloader and bootloadable projects. It demonstrates how the bootloader can place the API functions so that the bootloadable can also call them. The purpose of the Bootloader project is to replace a bootloadable image on the device with an image sent OTA by the Bluetooth protocol.	Peripheral
BLE OTA Upgradable Stack HID Keyboard	This example project shows how to implement an upgradable Application project (HID keyboard) and upgradable Stack project with the BLE Stack. In addition, the Application project uses the BLE Stack from the Stack project.	Peripheral
BLE Upgradable Stack Example Launcher	This example project shows how to implement an upgradable Application project (HID keyboard) and upgradable Stack project with the BLE Stack. In addition, the Application project uses the BLE Stack from the Stack project.	Peripheral
BLE Upgradable Stack Example Stack	This example project shows how to implement an upgradable Application project (HID keyboard) and upgradable Stack project with the BLE Stack. In addition, the Application project uses the BLE Stack from the Stack project.	Peripheral

Code Example	Purpose	GAP role
BLE Phone Alert	This example project demonstrates the BLE Phone Alert Handler application workflow. The Phone Alert Handler application uses the BLE Phone Alert Status Profile to monitor and control Alert State and Ringer Setting of the Server.	Peripheral
BLE Proximity	This example project demonstrates the Proximity operation of the BLE Component. The Proximity Reporter utilizes the BLE Proximity Profile with one instance of Link Loss Service and one instance of Tx Power Service to display alerts on the device if connection with Client has been lost.	Peripheral
BLE Running Speed Cadence	This example project demonstrates the Running Speed and Cadence Sensor operation of the BLE Component. The device simulates running/walking data measurements and sends it over the BLE Running Speed and Cadence Service.	Peripheral
BLE Temperature Measurement	This example project demonstrates the Health Thermometer Profile operation of the BLE Component. The device simulates thermometer readings and sends it over the BLE Health Thermometer Service. It also measures a battery level value and sends it over the BLE Battery Service.	Peripheral
BLE Time Sync	This example project demonstrates the Time profile operation of the BLE Component. The Time Sync example utilizes the BLE Time Profile (configured for GAP peripheral role as Time Client) with one instance of Current Time Service to demonstrate capability of time synchronization from the external Time Server.	Peripheral
BLE Weight Scale	This example project demonstrates the Weight Scale Profile operation of the BLE Component. The Weight Scale Sensor utilizes one instance of Weight Scale, Body Composition, User Data and Device Information Services to simulate weight measurements for up to four registered users.	Peripheral
BLE Wireless Power Transmitter and Receiver	This example project demonstrates the Wireless Power Transfer Profile operation of the BLE Component. This example demonstrates communication between Power Receiver Unit (PRU) and Power Transmitter Unit (PTU) in the Wireless Power Transfer systems. PTU central device supports time multiplexing connection with up to 8 PRU peripheral devices. PRU simulates power receiver data and reports the simulated data to a PTU using the Wireless Power Transfer Service (WPTS). The example consists of two projects: Wireless Power Transmitter (GATT Client) and Wireless Power Receiver (GATT Server).	Central and Peripheral
CE11181 - BLE HTTP Proxy Code Examples with PSoC 4 BLE	HTTP Proxy Server and HTTP Proxy Client projects are used in a pair to demonstrate the BLE HTTP Proxy Service (HPS) operation. HTTP Proxy Server utilizes one instance of HTTP Proxy Service to simulate HTTP Server on the BLE device. HTTP Proxy Server can also operate with other devices that implement the HTTP Proxy Client Role.	Central and Peripheral
CE211245 - Bluetooth Low Energy (BLE) Indoor Positioning	This example project demonstrates how to create an indoor navigation system using the BLE broadcasting mode that can be configured over GATT connection. This project configures the BLE Pioneer Kit as a time-multiplexed broadcaster and a connectable Indoor Positioning Service (IPS) server.	Peripheral
CE217613 - Bluetooth Low Energy (BLE) Automation IO	BLE example project that demonstrates how use BLE Component's Automation IO profile feature and related APIs. This project configures the BLE Pioneer Kit as an Automation Input Output server (AOIS) with two instances of Digital characteristic, two instances of Analog characteristic and Aggregate characteristic.	Peripheral

Document History

Document Title: AN91184 – PSoC® 4 BLE – Designing BLE Applications

Document Number: 001-91184

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4635526	PMAD	03/25/2015	New Application Note.
*A	4767014	UDYG	05/15/2015	Updated to PSoC Creator 3.2 and BLE Component v2.0.
*B	4784134	UDYG	06/02/2015	Updated images for CySmart PC Tool and CySmart mobile app. Updated template
*C	4911541	UDYG	09/10/2015	Fixed broken links
*D	5137953	UDYG	02/15/2016	Updated to PSoC Creator 3.3 SP1 and BLE Component v2.30.
*E	5834974	AESATMP8	07/27/2017	Updated logo and Copyright.
*F	6164690	AJYA	12/12/2018	Updated to PSoC Creator 4.2 Added list of code examples Updated template

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

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