Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.
AN91184

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 CY8KIT-042-BLE kit.

Contents

1 Introduction.................................................................1
2 PSoC Resources.........................................................2
3 PSoC Creator ..............................................................3
4 Standard Services Versus Custom Services .................4
   4.1 BLE Health Thermometer ........................................4
5 PSoC Creator Project: Health Thermometer .............5
   5.1 Configure the Component..................................5
   5.2 Configure the Firmware ..................................18
5.3 Hardware Configuration.................................23

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

www.cypress.com

Document Number: 001-91184 Rev. *F 1
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](image-url)
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**

---

### BLE

The BLE Component is configured for the Health Thermometer Sensor profile. This profile, device sends temperature information to the remote device at periodic intervals.

---

### ADC

Once the connection is established, a SAR ADC converts reference and thermistor voltage to digital value. The temperature is calculated after removing offset and gain error.

---

### Thermistor Calculator

This component calculates temperature based on the voltage measured using the ADC. It is configured for LUT method with an accuracy of 0.5C.

---

### User Button

This button on BLE Pioneer kit is used for waking up the system from Hibernate mode.

---

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](image)

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
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 6. General Tab

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.
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

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public address</td>
<td>Check Silicon Generated Address</td>
<td>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.</td>
</tr>
<tr>
<td>Device name</td>
<td>MyThermometer</td>
<td>User-defined</td>
</tr>
<tr>
<td>Appearance</td>
<td>Generic Thermometer</td>
<td>The possible values are defined by the Bluetooth SIG. The BLE Component provides an option to select one of the possible values.</td>
</tr>
</tbody>
</table>

**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.
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Check Box</th>
<th>Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Name</td>
<td>Enabled</td>
<td>Complete</td>
<td>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.</td>
</tr>
<tr>
<td>Service UUID</td>
<td>Enabled</td>
<td>N/A</td>
<td>Transmits Service UUIDs ( Universally unique identifier) as a part of the advertisement packet.</td>
</tr>
<tr>
<td>Service UUID &gt; Health Thermometer</td>
<td>Enabled</td>
<td>N/A</td>
<td>Transmits the Health Thermometer Service UUID as a part of the advertisement packet.</td>
</tr>
<tr>
<td>Service UUID &gt; Device Information</td>
<td>Enabled</td>
<td>N/A</td>
<td>Transmits the Device Information Service UUID as a part of the advertisement packet.</td>
</tr>
<tr>
<td>Appearance</td>
<td>Enabled</td>
<td>N/A</td>
<td>Transmits the Appearance value as a part of the advertisement packet.</td>
</tr>
</tbody>
</table>
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 10. GAP Settings – Advertisement Packet**

![GAP Settings](image)

**Figure 11. Pin Configuration**

![Pin Configuration](image)
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.

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

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](image)

**Note:** For accurate temperature measurement, the Temperature (°C) 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](image1)

![Figure 19. Analog Pin Configuration – V_LOW](image2)

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

Application Firmware Flow

- **System Initialization**
  - Reset or Wakeup from Hibernate
  - Initialize ADC
  - Provide Thermistor reference
  - Initialize BLE
  - Register Generic Event Handler

- **System Normal Operation**
  - Process Events
    - Device Connected?
    - WDT event?
    - Measurement interval expired?
  - Measure Temperature
    - Indication Enabled?
    - Send Indication

- **System Low Power Operation**
  - BLESS = Initializing?
    - Y
    - Put BLESS in Deep Sleep mode
    - Get BLESS state
  - Rx/Tx Complete?
    - N
    - Put System in Sleep mode
    - BLESS = Deep Sleep or ECO_ON
    - N
    - Put System in Deep Sleep mode

- **BLE Event Handler**
  - Stack ON?
    - Y
    - Start Advertising
    - N
    - Device Connected?
      - Y
      - Retrieve Connection Handle
      - Initialize WDT
      - Configure wakeup pin
      - Go to Hibernate
      - N
      - Device Disconnected?
        - Y
        - Advertising Timeout?
          - Y
          - Configure wakeup pin
          - Go to Hibernate
          - N
          - Return from Generic event handler
        - N
        - Return from Generic event handler
  - N
  - Error Handling
  - N
  - BLESS Init Success?
    - Y
    - Register Health Thermometer Event Handler
    - - Configure wakeup pin
    - - Go to Hibernate
    - N
  - Y

- **Generic Event Handler**
  - Indication Enabled?
    - Y
    - Start Indication
    - N
    - Indication Disabled?
      - Y
      - Disable Indication
      - N
      - Measurement Interval Write?
        - Y
        - Update Measurement Interval
        - N
        - Return from Generic event handler
      - N
      - Return from Generic event handler
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>This is the main firmware file. It has only one function:</td>
</tr>
<tr>
<td></td>
<td>▪ 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.</td>
</tr>
<tr>
<td>CommonFunctions.c/.h</td>
<td>Implements the common functions that are used for application control. It has the following functions:</td>
</tr>
<tr>
<td></td>
<td>▪ InitializeSystem() – Initializes all the blocks of the system.</td>
</tr>
<tr>
<td></td>
<td>▪ PrepareForDeepSleep – Prepares the system for low-power operation by putting the hardware blocks in the Deep Sleep mode.</td>
</tr>
<tr>
<td></td>
<td>▪ WakeupFromDeepSleep() – Restores the hardware blocks for normal operation.</td>
</tr>
<tr>
<td>Temperature.c/.h</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>▪ 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.</td>
</tr>
<tr>
<td></td>
<td>▪ ProcessTemperature() – This function measures or simulates the temperature value.</td>
</tr>
<tr>
<td>WatchdogTimer.c/.h</td>
<td>It implements the watchdog timer functionality and keeps track of the system time. It has the following functions:</td>
</tr>
<tr>
<td></td>
<td>▪ WatchdogTimer_Start() – Starts the watchdog timer (WDTO) with a 1-s period and an interrupt on match.</td>
</tr>
<tr>
<td></td>
<td>▪ WatchdogTimer_Isr() – The ISR for the WDT; it is used to track the measurement interval. This function is a callback from the watchdog timer.</td>
</tr>
<tr>
<td></td>
<td>▪ WatchdogTimer_Stop() – Stops the watchdog timer (WDTO).</td>
</tr>
<tr>
<td>BLE_HTSS.c/.h</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>▪ GenericEventHandler() – Handles the generic events generated by the BLE stack</td>
</tr>
<tr>
<td></td>
<td>▪ HtssEventHandler() – Handles the events generated for the Health Thermometer Service</td>
</tr>
<tr>
<td></td>
<td>▪ ProcessBLE() – Sends the temperature data as Indication to the GATT Client</td>
</tr>
<tr>
<td></td>
<td>▪ ConvertFloatTemp() – Converts the temperature data from IEEE-754 format to IEEE-11073 format</td>
</tr>
<tr>
<td></td>
<td>▪ EnableBLE() – Starts the BLE Component and registers the event handler functions</td>
</tr>
</tbody>
</table>

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

```c
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

```c
CyBle_HtsRegisterAttrCallback(HealthThermometerEventHandler);
```

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_HTS_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 HtsEventHandler.
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;
    }
    }
}

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_HTS_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 htssIndication flag */
    htssIndication = false;
    break;
}

/* This event is received when measurement interval is updated by 
the central */
case CYBLE_EVT_HTSS_CHAR_WRITE:
{
    /* Retrieve 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 wake-up, 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 °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

```c
#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 °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](image)

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](image)
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](image)

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](image)

### 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](http://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

![Select BLE Dongle Target](image)

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

![Start Scan](image)

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

Figure 31. Discovered Devices

![Discovered Devices](image)
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 “MyThermometer,” 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

   ![Characteristics Declaration Table](image)

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

   ![Attribute Details](image)

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

   Figure 36. Measured Temperature

   ![Primary Service Declaration](image)

   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 °C (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.

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.

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](image)

### 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

<table>
<thead>
<tr>
<th>Application Notes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AN91267 – Getting Started with PSoC 4 BLE</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>AN91162 – Creating a BLE Custom Profile</strong></td>
<td>Describes the methodology for developing a Bluetooth® Low Energy (BLE) application with PSoC 4 BLE or PRoC BLE devices using a custom BLE profile.</td>
</tr>
<tr>
<td><strong>AN92584 – Designing for Low Power and Estimating Battery Life for BLE Applications</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>AN66477 – PSoC® 3, PSoC 4, and PSoC 5LP - Temperature Measurement with a Thermistor</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>AN210781 – Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity</strong></td>
<td>Describes PSoC 6 MCU with BLE Connectivity devices and how to build your first PSoC Creator project</td>
</tr>
<tr>
<td><strong>AN215656 – PSoC 6 MCU: Dual-CPU System Design</strong></td>
<td>Describes the dual-CPU architecture in PSoC 6 MCU, and shows how to build a simple dual-CPU design</td>
</tr>
<tr>
<td><strong>AN219434 – Importing PSoC Creator Code into an IDE for a PSoC 6 MCU Project</strong></td>
<td>Describes how to import the code generated by PSoC Creator into your preferred IDE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PSoC Creator Component Datasheets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pins</strong></td>
</tr>
<tr>
<td><strong>Timer Counter (TCPWM)</strong></td>
</tr>
<tr>
<td><strong>Clock</strong></td>
</tr>
<tr>
<td><strong>Interrupt</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PSoC® 4: PSoC 4200_BLE Family Datasheet Programmable System-on-Chip (PSoC®)</strong></td>
</tr>
<tr>
<td><strong>PSoC 6 MCU: PSoC 63 with BLE Datasheet</strong></td>
</tr>
<tr>
<td><strong>PSoC 6 MCU: PSoC 62 Datasheet</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Development Kit Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CY8CKIT-042-BLE Bluetooth Low Energy Pioneer Kit Guide</strong></td>
</tr>
<tr>
<td><strong>CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit</strong></td>
</tr>
<tr>
<td><strong>CY8CKIT-062-WiFi-BT PSoC 6 WiFi-BT Pioneer Kit</strong></td>
</tr>
<tr>
<td><strong>CY8CPROTO-062-4343W PSoC 6 Wi-Fi BT Prototyping Kit</strong></td>
</tr>
<tr>
<td><strong>CY8CPROTO-063 BLE PSoC 6 BLE Prototyping Kit</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tool Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PSoC Creator</strong></td>
</tr>
</tbody>
</table>
## 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.

<table>
<thead>
<tr>
<th>Code Example</th>
<th>Purpose</th>
<th>GAP role</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLE Find Me</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Device Information Service</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE 4.2 Data Length Security Privacy</td>
<td>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).</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Alert Notification Profile</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Apple Notification Client</td>
<td>This example project demonstrates the BLE Apple Notification Center 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.).</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Battery Level</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Blood Pressure Sensor</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Continuous Glucose Monitoring Sensor</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>Code Example</td>
<td>Purpose</td>
<td>GAP role</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>BLE Cycling Sensor</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Environmental Sensing Profile</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Glucose Meter</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Heart Rate Collector</td>
<td>This example 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.</td>
<td>Central</td>
</tr>
<tr>
<td>BLE Heart Rate Sensor</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE HID Keyboard</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE HID Mouse</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE IPSP Router and Node</td>
<td>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.</td>
<td>Central and Peripheral</td>
</tr>
<tr>
<td>BLE Navigation</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE OTA External Memory Bootloadable and Bootloader</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE OTA Fixed Stack Bootloader and Bootloadable</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE OTA Upgradable Stack HID Keyboard</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Upgradable Stack Example Launcher</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Upgradable Stack Example Stack</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>Code Example</td>
<td>Purpose</td>
<td>GAP role</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>BLE Phone Alert</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Proximity</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Running Speed Cadence</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Temperature Measurement</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Time Sync</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Weight Scale</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>BLE Wireless Power Transmitter and Receiver</td>
<td>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).</td>
<td>Central and Peripheral</td>
</tr>
<tr>
<td>CE11181 - BLE HTTP Proxy Code Examples with PSoC 4 BLE</td>
<td>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.</td>
<td>Central and Peripheral</td>
</tr>
<tr>
<td>CE211245 - Bluetooth Low Energy (BLE) Indoor Positioning</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>CE217613 - Bluetooth Low Energy (BLE) Automation IO</td>
<td>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.</td>
<td>Peripheral</td>
</tr>
</tbody>
</table>
### Document History

**Document Title:** AN91184 – PSoC® 4 BLE – Designing BLE Applications  
**Document Number:** 001-91184

<table>
<thead>
<tr>
<th>Revision</th>
<th>ECN</th>
<th>Orig. of Change</th>
<th>Submission Date</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>****</td>
<td>4635526</td>
<td>PMAD</td>
<td>03/25/2015</td>
<td>New Application Note.</td>
</tr>
<tr>
<td>*A</td>
<td>4767014</td>
<td>UDYG</td>
<td>05/15/2015</td>
<td>Updated to PSoC Creator 3.2 and BLE Component v2.0.</td>
</tr>
<tr>
<td>*B</td>
<td>4784134</td>
<td>UDYG</td>
<td>06/02/2015</td>
<td>Updated images for CySmart PC Tool and CySmart mobile app. Updated template</td>
</tr>
<tr>
<td>*C</td>
<td>4911541</td>
<td>UDYG</td>
<td>09/10/2015</td>
<td>Fixed broken links</td>
</tr>
<tr>
<td>*D</td>
<td>5137953</td>
<td>UDYG</td>
<td>02/15/2016</td>
<td>Updated to PSoC Creator 3.3 SP1 and BLE Component v2.30.</td>
</tr>
<tr>
<td>*E</td>
<td>5834974</td>
<td>AESATMP8</td>
<td>07/27/2017</td>
<td>Updated logo and Copyright.</td>
</tr>
</tbody>
</table>
| *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.

© Cypress Semiconductor Corporation, 2015-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spanion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errors which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products. Cypress, the Cypress logo, Spanion, the Spanion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.