

PSoC® 4 BLE – Designing BLE Applications

Author: Uday Agarwal

Associated Project: Yes

Associated Part Family: CY8C4XX7-BL, CY8C4XX8-BL, CYBL1XX6X, CYBL1XX7X

Software Version: PSoC Creator™ 3.3 SP1

Related Application Notes: [click here](#)

To get the latest version of this application note, please visit
<http://www.cypress.com/go/AN91184>.

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. This application note also applies to the PProC BLE part.

Contents

1	Introduction.....	1	4.2	Configure the Firmware	22
2	PSoC Resources	2	4.3	Hardware Configuration.....	27
2.1	PSoC Creator	2	4.4	Build and Program the Device	28
2.2	PSoC Creator Help	3	5	Application Testing	29
2.3	Code Examples	4	5.1	CySmart Central Emulation Tool	29
2.4	Technical Support.....	5	5.2	CySmart Mobile App.....	33
3	Standard Services Versus Custom Services	6	5.3	Summary	35
3.1	BLE Health Thermometer	6	6	Related Application Notes	35
4	PSoC Creator Project: Health Thermometer	7		Worldwide Sales and Design Support.....	37
4.1	Configure the Component.....	8			

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 a lower bandwidth of 1 Mbps.

Cypress's 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:

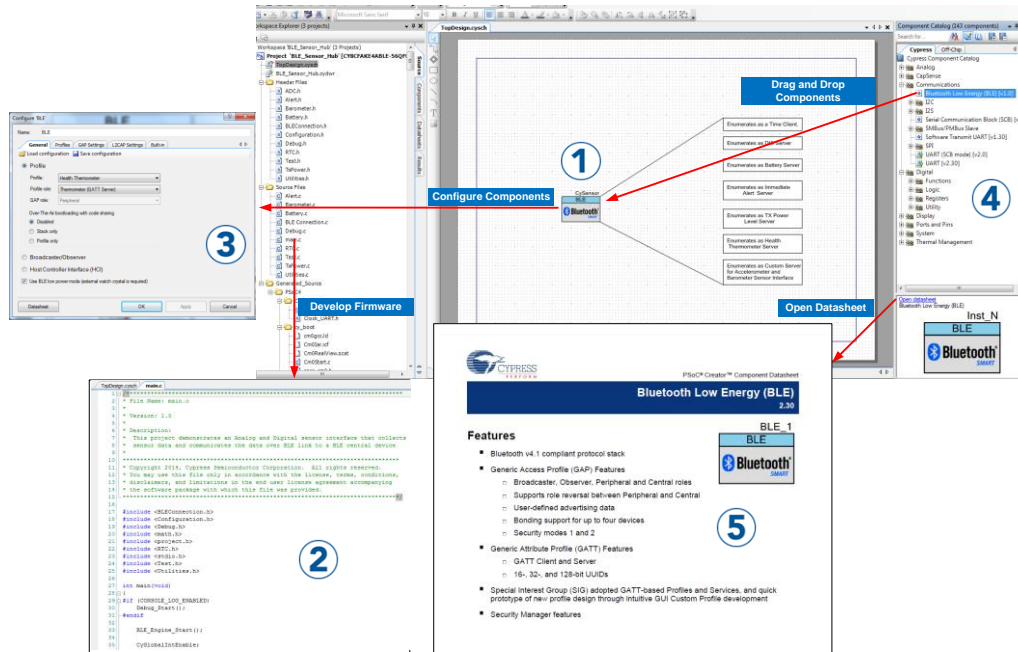
- **Overview:** [PSoC Portfolio](#), [PSoC Roadmap](#)
- **Product Selectors:** [PSoC 1](#), [PSoC 3](#), [PSoC 4](#), or [PSoC 5LP](#). In addition, [PSoC Creator](#) includes a device selection tool.
- **Datasheets** describe and provide electrical specifications for the [PSoC 41XX-BL](#) and [PSoC 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—see [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.

2.1 PSoC Creator

[PSoC Creator](#) is a free Windows-based Integrated Design Environment (IDE). It enables you to design hardware and firmware systems concurrently, based on PSoC 4 BLE and PProC BLE. As [Figure 1](#) shows, with PSoC Creator, you can:

1. Drag and drop Components to build your hardware system design in the main design workspace.
2. Co-design your application firmware with the PSoC hardware.
3. Configure the Components using configuration tools.
4. Explore the library of more than 100 Components.
5. Review the Component datasheets.

Figure 1. PSoC Creator Schematic Entry and Components



2.2 PSoC Creator Help

Visit the [PSoC Creator](#) home page to download and install the latest version of PSoC Creator. Then launch PSoC Creator and navigate to the following items:

- **Quick Start Guide:** Choose **Help > Documentation > Quick Start Guide**. This guide gives you the basics for developing PSoC Creator projects.
- **Simple Component example projects:** Choose **File > Open > Example projects**. These example projects demonstrate how to configure and use PSoC Creator Components.
- **Starter designs:** Choose **File > New > Project > PSoC 4 Starter Designs**. These starter designs demonstrate the unique features of PSoC 4 BLE.
- **System Reference Guide:** Choose **Help > System Reference > System Reference Guide**. This guide lists and describes the system functions provided by PSoC Creator.
- **Component datasheets:** Right-click a Component and select “Open Datasheet.” Visit the [PSoC 4 BLE Component Datasheets](#) page for a list of all PSoC 4 BLE Component datasheets.
- **Document Manager:** PSoC Creator provides a document manager to help you to easily find and review document resources. To open the document manager, choose the menu item **Help > Document Manager**.

2.3 Code Examples

PSoC Creator includes a large number of code example projects. These projects are available from the PSoC Creator Start Page, as [Figure 3](#) shows.

Code examples can speed up your design process by starting you off with a complete design, instead of a blank page. The code examples also show how PSoC Creator Components can be used for various applications. Code examples and datasheets are included, as [Figure 3](#) shows.

In the **Find Code Example** dialog shown in [Figure 3](#), you have several options:

- Filter for examples based on device family, that is, PSoC 4, PSoC 4 BLE, PSoC 4 BLE, and so on; or keyword.
- Select from the menu of examples offered based on the filters. There are more than 20 BLE example projects for you to get started, as shown in [Figure 3](#).
- Review the datasheet for the selection (on the **Documentation** tab)
- Review the code example for the selection. You can copy and paste code from this window to your project, which can help speed up code development.
- Or create a new project (and a new workspace if needed) based on the selection. This can speed up your design process by starting you off with a complete basic design. You can then adapt that design to your application.

Apart from PSoC Creator code examples, you can also find more BLE reference examples on [this](#) GitHub repository.

Figure 2. Code Examples in PSoC Creator

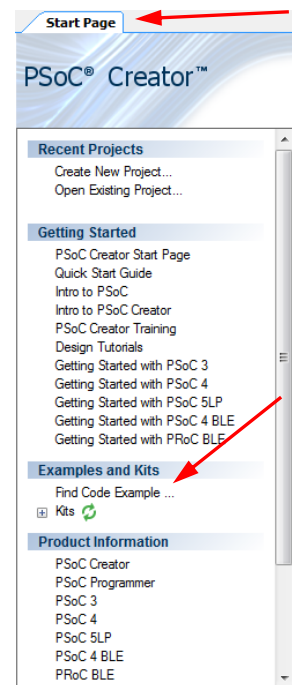
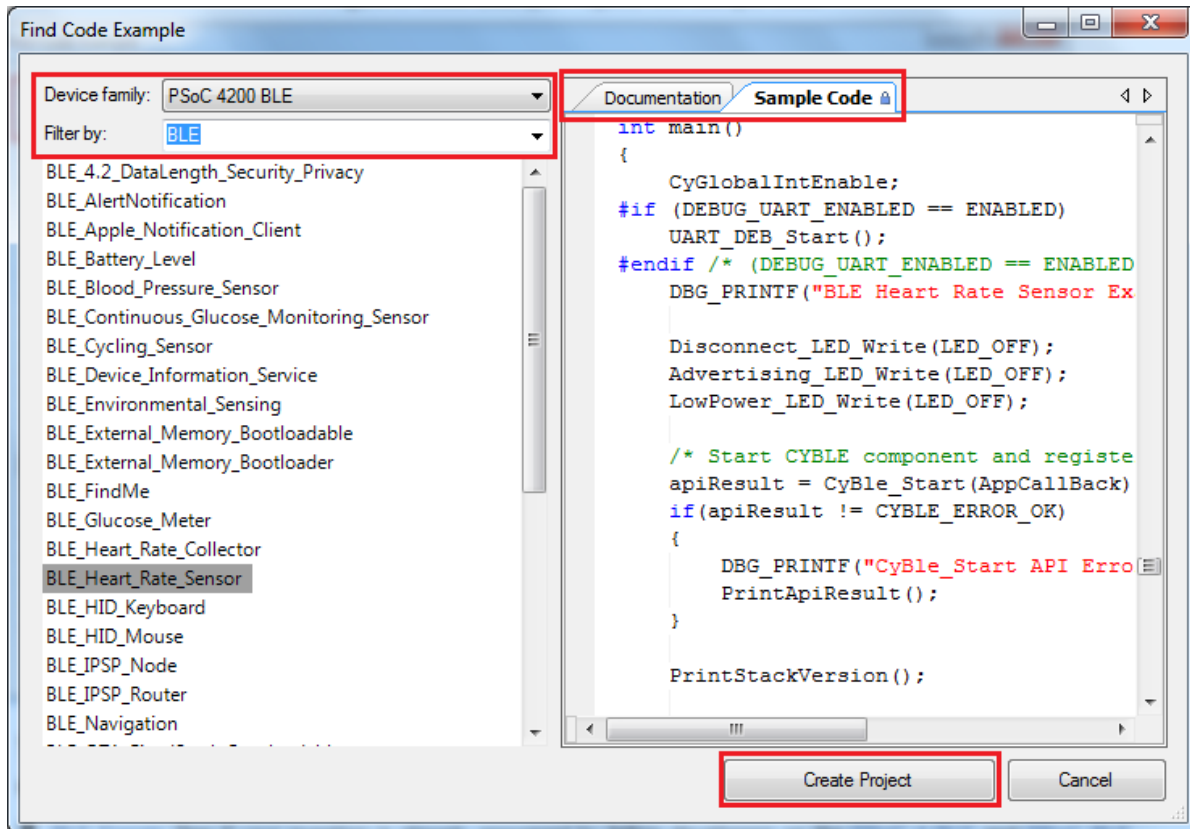


Figure 3. Code Example Projects with Sample Code



2.4 Technical Support

- [Frequently Asked Questions \(FAQs\)](#): Learn more about our BLE ecosystem
- [BLE Forum](#): See if your question is already answered by fellow developers on the [PSoC 4 BLE](#) and [PRoC BLE](#) forums.
- Cypress support: Still no luck? Visit our [support](#) page and create a [technical support case](#) or contact a [local sales representative](#). If you are in the United States, you can talk to our technical support team by calling our toll-free number: +1-800-541-4736. Select option 8 at the prompt.

3 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.

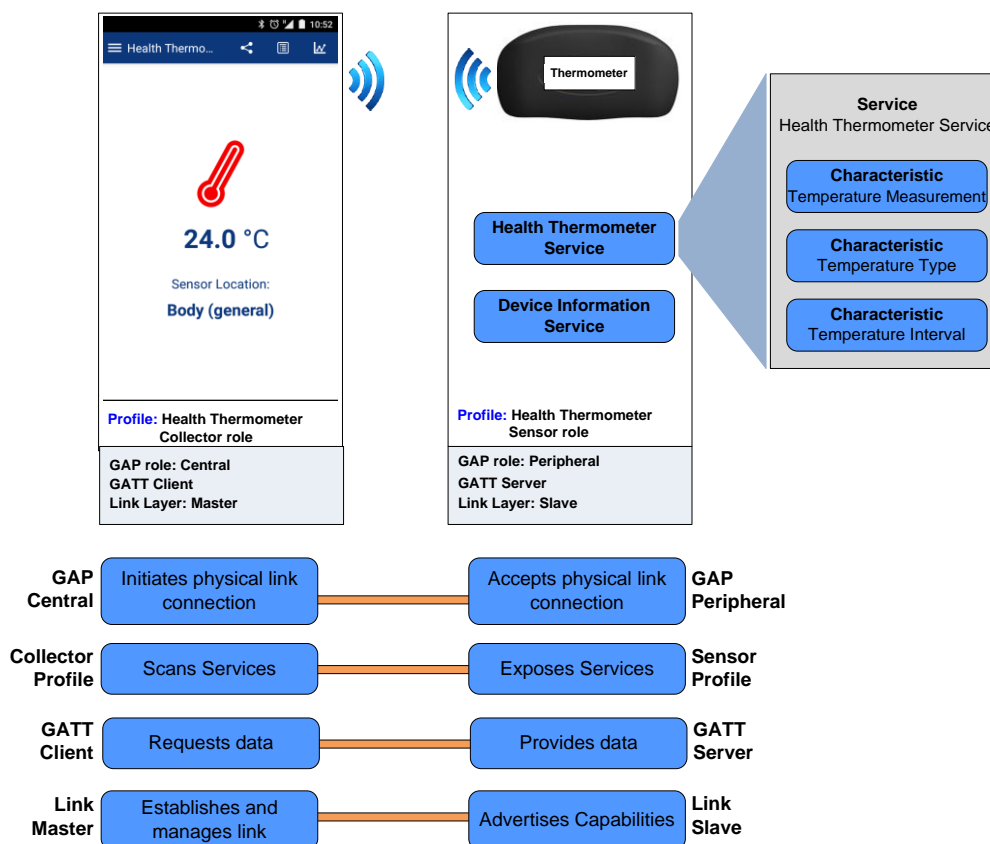
These standard services are defined to address a wide variety of applications. For example, the Heart Rate Service can be configured to report data from a Heart-Rate Sensor in a wristband or a chest-strap monitor. It can also expose the amount of energy expended over a specified interval.

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.

3.1 BLE Health Thermometer

In the BLE Health Thermometer application ([Figure 4](#)), 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 4. BLE System Design



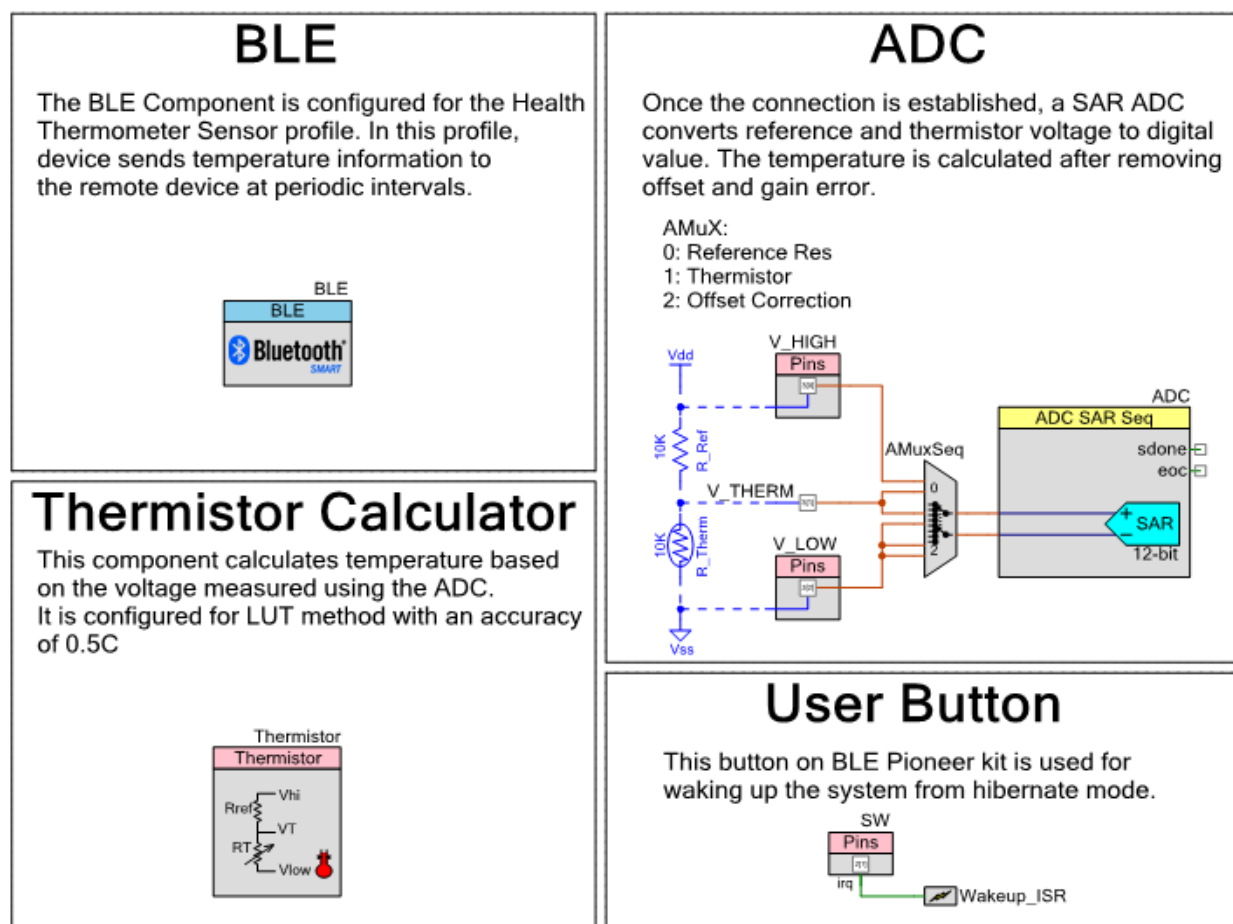
4 PSoC Creator Project: Health Thermometer

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 5 shows the PSoC Creator schematic of the Health Thermometer project.

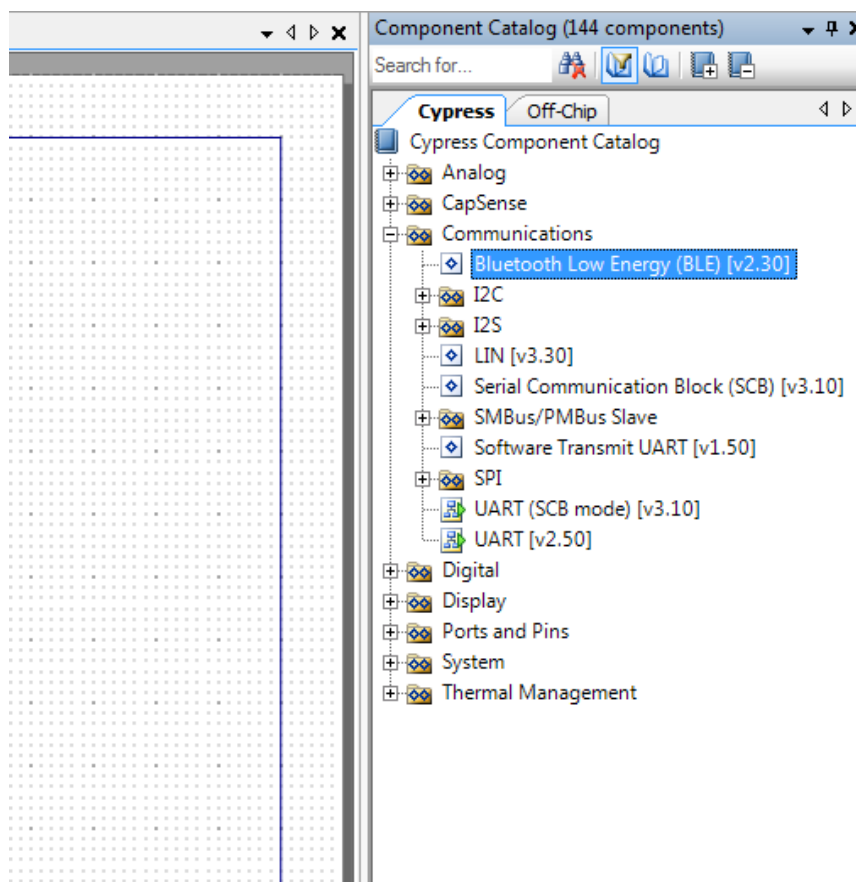
Figure 5. PSoC Creator Schematic



4.1 Configure the Component

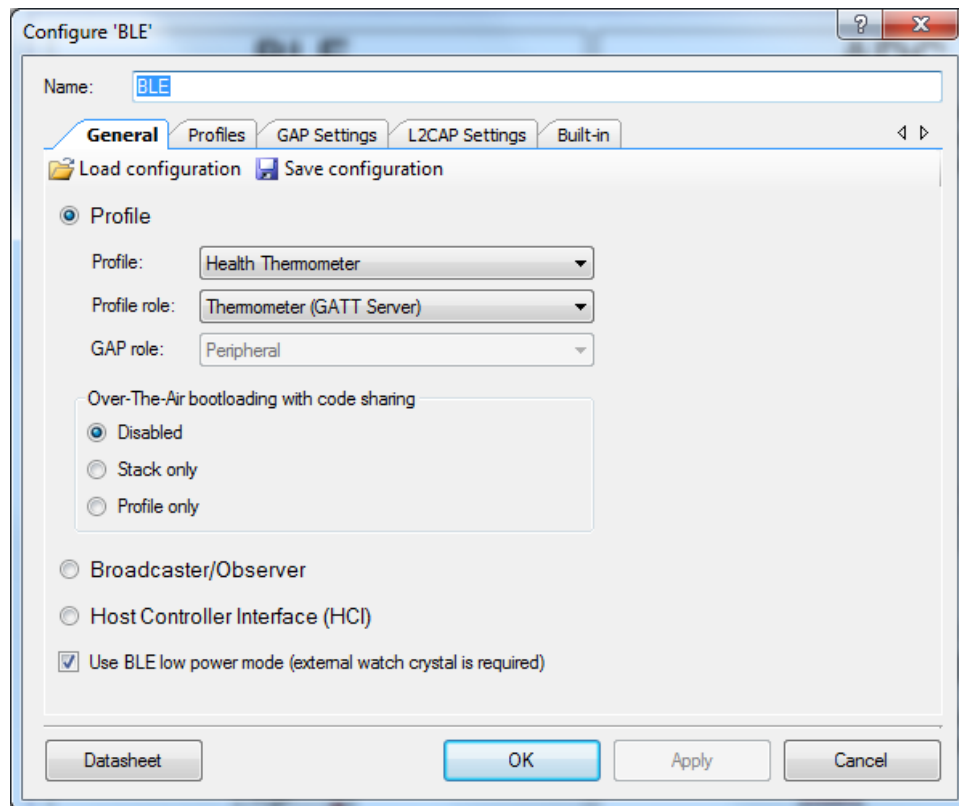
1. Create a new PSoC 4100 BLE / PSoC 4200 BLE Design project. If you are new to PSoC Creator, refer to the [PSoC Creator home page](#).
2. Drag and drop a BLE Component (Component Catalog > Communications) into the TopDesign schematic (refer to [Figure 6](#)).

Figure 6. BLE Component



3. Double-click the BLE Component to configure it. The configuration window appears as shown in [Figure 7](#).

Figure 7. BLE Component – Configuration Window



4. On the **General** tab of the Component Configuration window (refer to [Figure 8](#)), configure the following settings:

Name: BLE

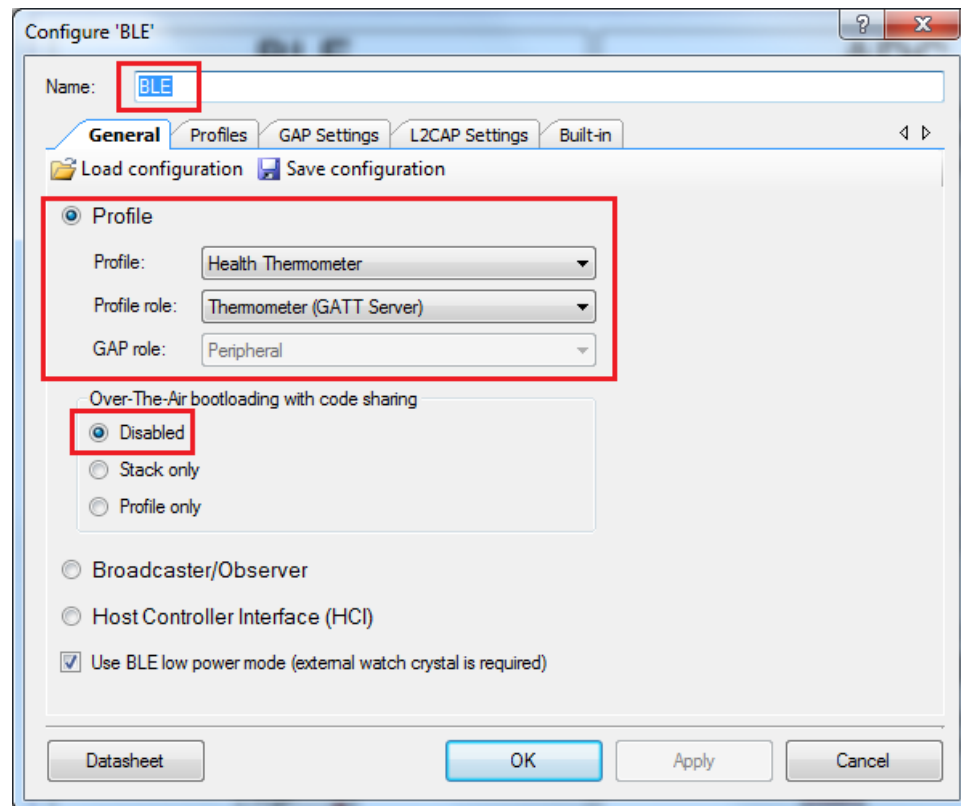
Configuration: Profile Collection

Profile: Health Thermometer

Profile role: Thermometer (GATT Server)

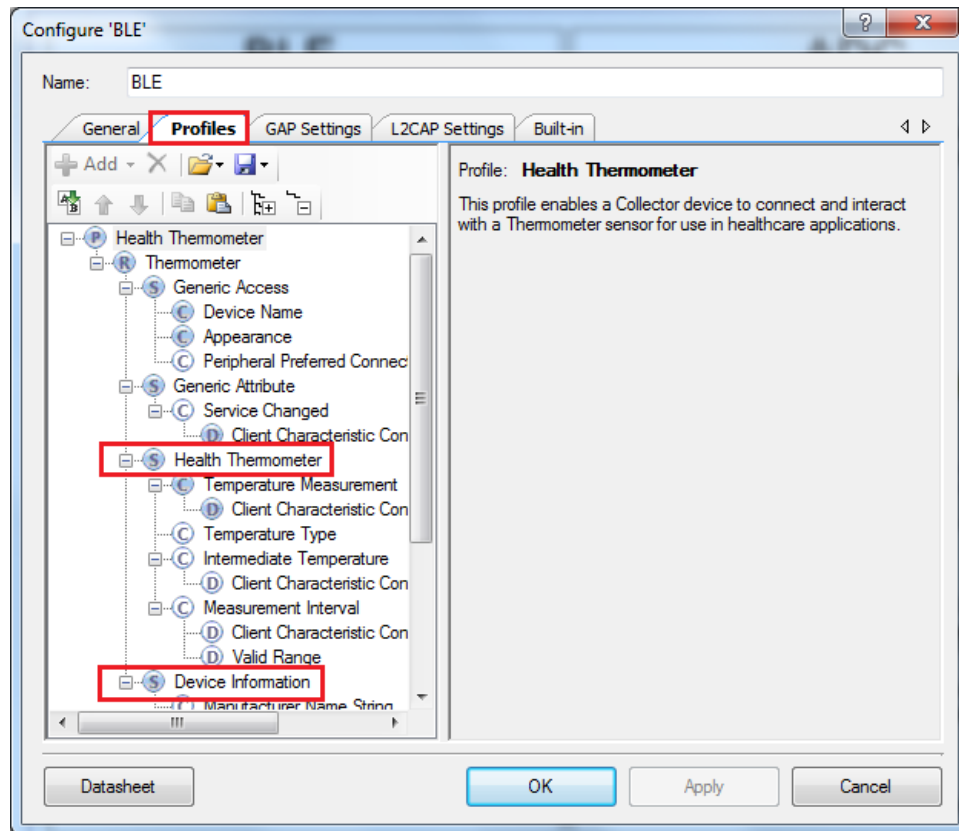
Over-The-Air bootloading with code sharing: Disabled

Figure 8. 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 9](#). To learn more about the Health Thermometer Profile or the Health Thermometer Service, refer to the [Bluetooth Adopted Specifications](#).

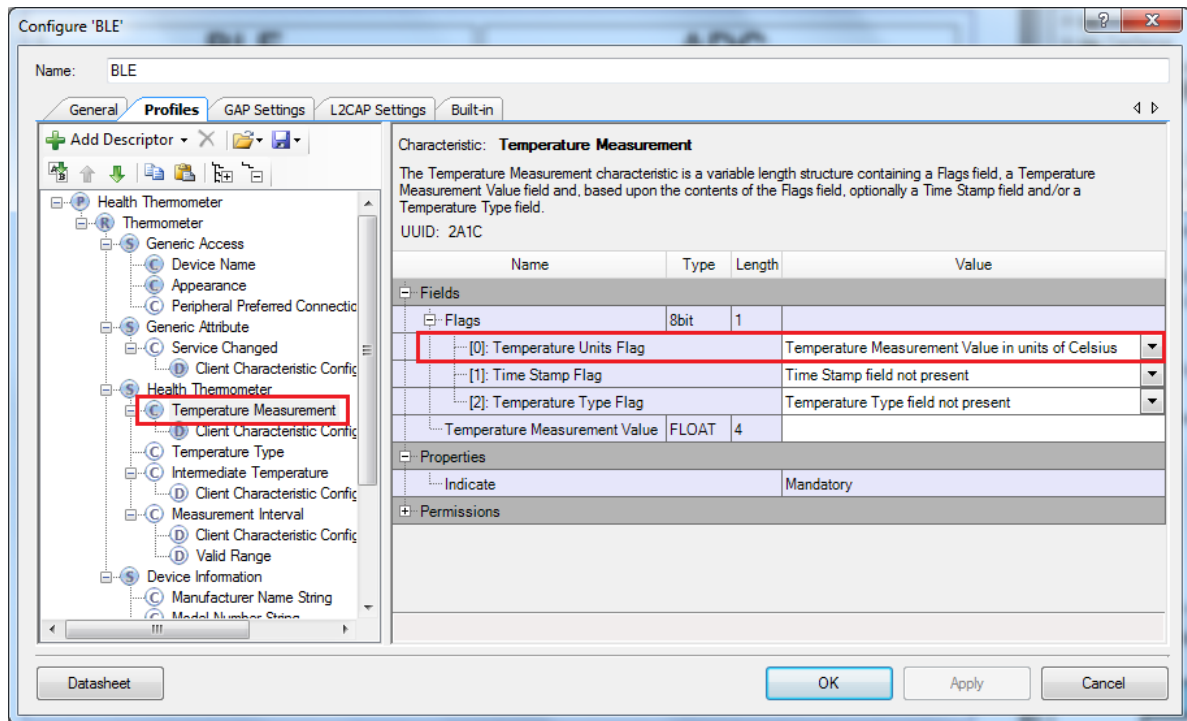
Figure 9. Health Thermometer Profile



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
 - Characteristic: Temperature Type
 - Fields: Temperature Text Description
 - Value: Body (general)

Figure 10 shows the Temperature Measurement Characteristic configuration.

Figure 10. 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

Service	Characteristic	Descriptor	Fields	Value	Remarks
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 3.3	User-defined

Keep the remaining settings at their default values.

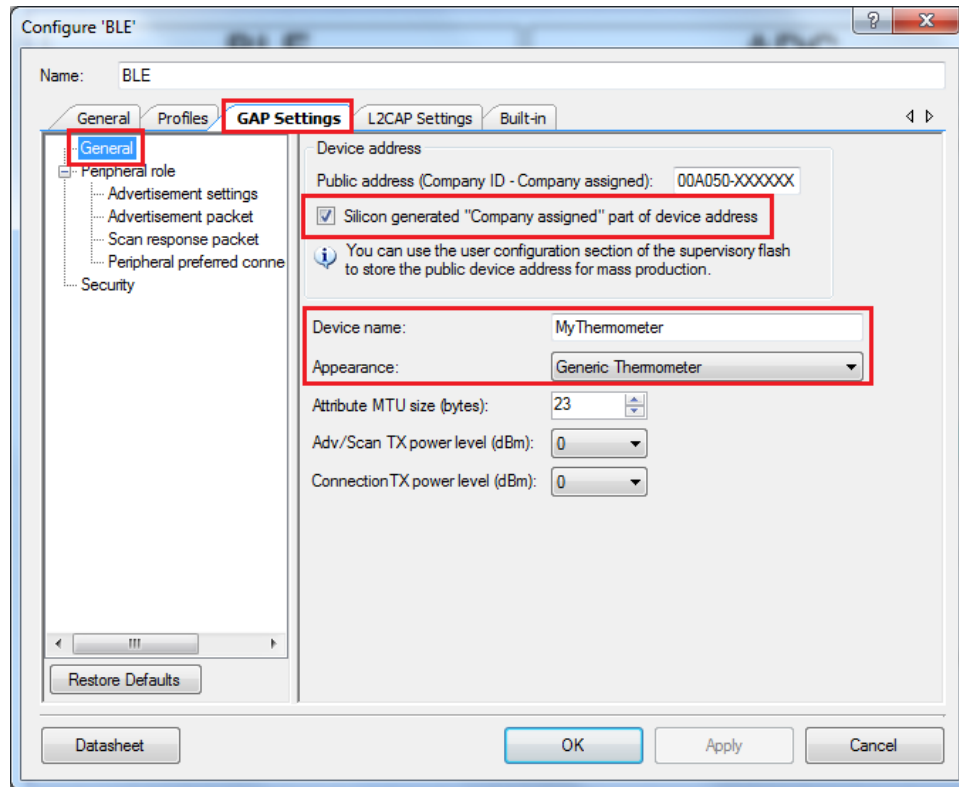
6. Configure the **Bluetooth Device Address (BD_ADDR)**, the **Device name**, and the **Appearance** under **General** settings of the **GAP Settings** tab, per [Table 2](#).

Table 2. General Settings

Name	Value	Remarks	Figure
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.	Figure 11
Device name	MyThermometer	User-defined	Figure 11
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.	Figure 11

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 11. Device Name and Appearance

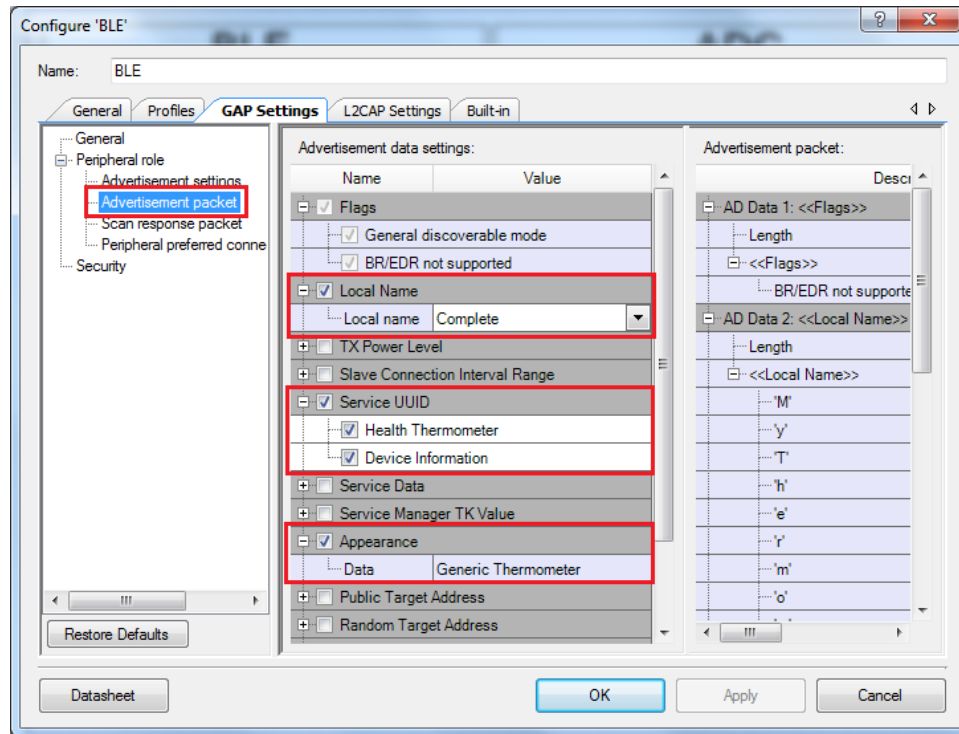


7. The **Advertisement Settings** under **Peripheral Role** will be left at their default values for this project.
8. Configure the **Advertisement packet** settings under **Peripheral role** per [Table 3](#).

Table 3. Advertisement Packet Settings

Name	Check Box	Value	Remarks	Figure
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.	Figure 12
Service UUID	Enabled	N/A	Transmits Service UUIDs (universally unique identifier) as a part of the advertisement packet.	Figure 12
Service UUID > Health Thermometer	Enabled	N/A	Transmits the Health Thermometer Service UUID as a part of the advertisement packet.	Figure 12
Service UUID > Device Information	Enabled	N/A	Transmits the Device Information Service UUID as a part of the advertisement packet.	Figure 12
Appearance	Enabled	N/A	Transmits the Appearance value as a part of the advertisement packet.	Figure 12

Figure 12. 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.

9. Click **Apply** and then click **OK**.
10. Place a Digital Input Pin Component and configure it as shown in Figure 13 and Figure 14.

Figure 13. Pin Configuration

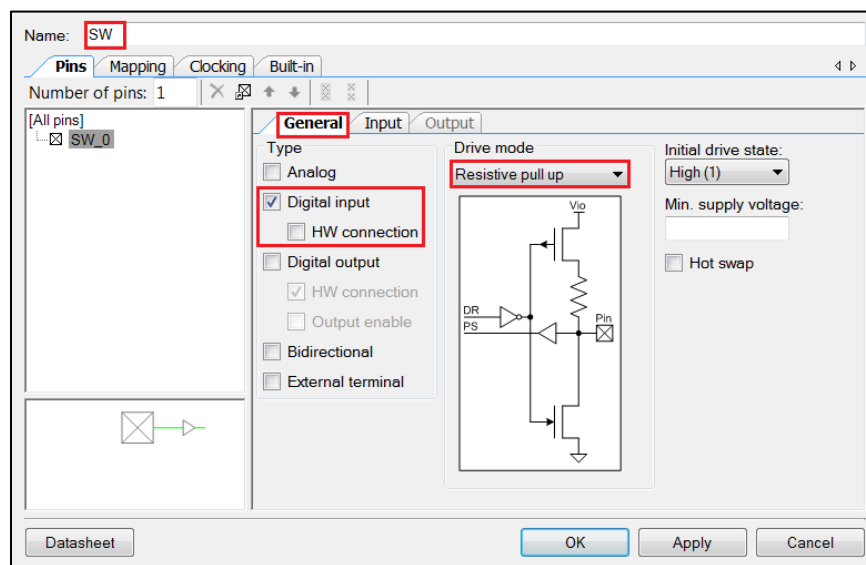
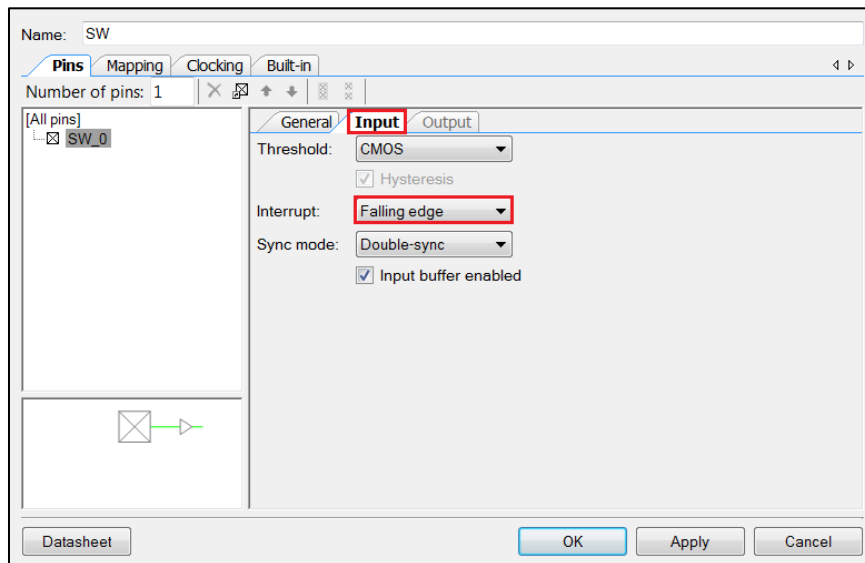


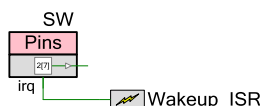
Figure 14. Pin Interrupt Configuration



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**.

11. 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 15. The interrupt Component will be used to record the interrupt signal and trigger the respective function.

Figure 15. SW Pin

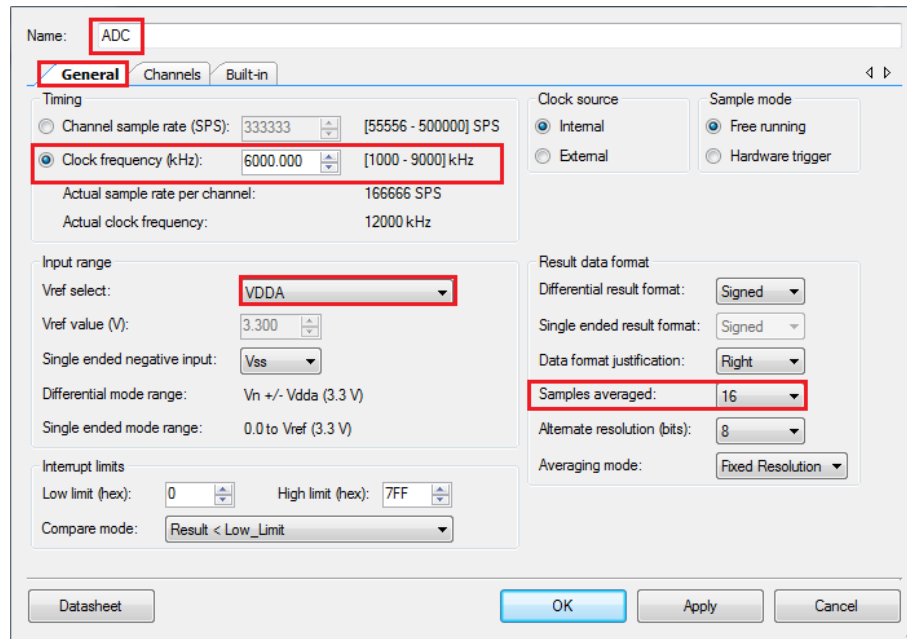


12. To measure temperature, follow steps 13 to 19.
13. Drag and drop the Sequencing SAR ADC Component and configure it per Table 4.

Table 4. SAR ADC Component Configuration

Tab	Name	Check Box	Value	Remarks	Figure
General	Name	N/A	ADC	This name will be used as the prefix to the APIs for the Component.	Figure 16
General	Clock frequency (kHz)	N/A	6000	This application does not require a high accuracy. A faster sample rate helps conserve power by reducing the active time.	Figure 16
General	Vref Select	N/A	VDDA	Input voltage range from 0 to V_{DDA} .	Figure 16
General	Samples Averaged	N/A	16	Sample averaging of 16 helps averaging out any high-frequency noise.	Figure 16
Channels	Sequenced channels	N/A	1	Number of channels to be scanned	Figure 17
Channels	AVG	Checked	N/A	Enable averaging for the corresponding ADC channel	Figure 17

Figure 16. SAR ADC Configuration – General



Name: **ADC**

General Channels Built-in

Timing

☐ Channel sample rate (SPS): 333333 [55556 - 500000] SPS

☒ Clock frequency (kHz): **6000.000** [1000 - 9000] kHz

Actual sample rate per channel: 166666 SPS

Actual clock frequency: 12000 kHz

Clock source: ☒ Internal ☐ External

Sample mode: ☒ Free running ☐ Hardware trigger

Input range

Vref select: **VDDA**

Vref value (V): 3.300

Single ended negative input: Vss

Differential mode range: Vn +/- Vdda (3.3 V)

Single ended mode range: 0.0 to Vref (3.3 V)

Result data format

Differential result format: Signed

Single ended result format: Signed

Data format justification: Right

Samples averaged: 16

Alternate resolution (bits): 8

Averaging mode: Fixed Resolution

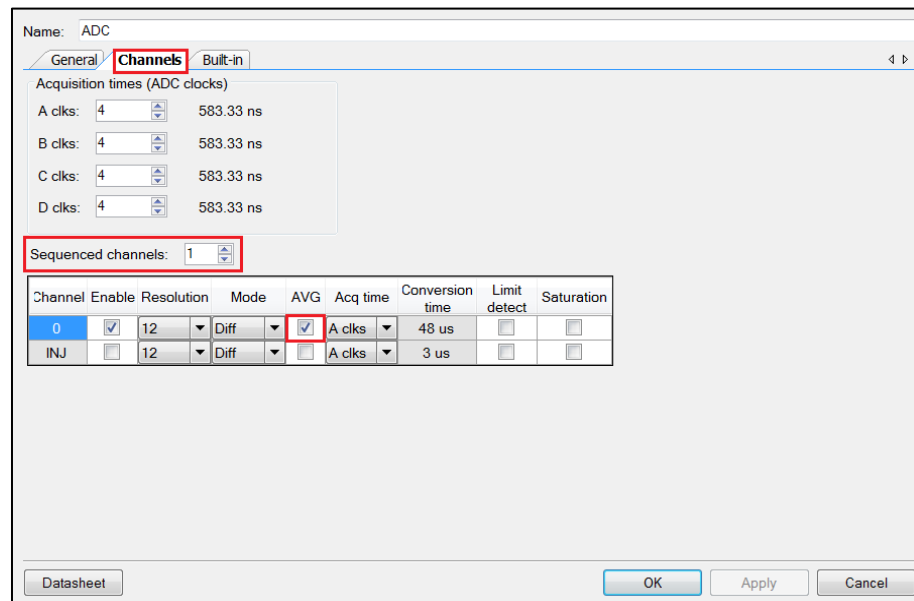
Interrupt limits

Low limit (hex): 0 High limit (hex): 7FF

Compare mode: Result < Low_Limit

Datasheet OK Apply Cancel

Figure 17. SAR ADC Configuration – Channels



Name: ADC

General **Channels** Built-in

Acquisition times (ADC clocks)

A clks: 4 583.33 ns

B clks: 4 583.33 ns

C clks: 4 583.33 ns

D clks: 4 583.33 ns

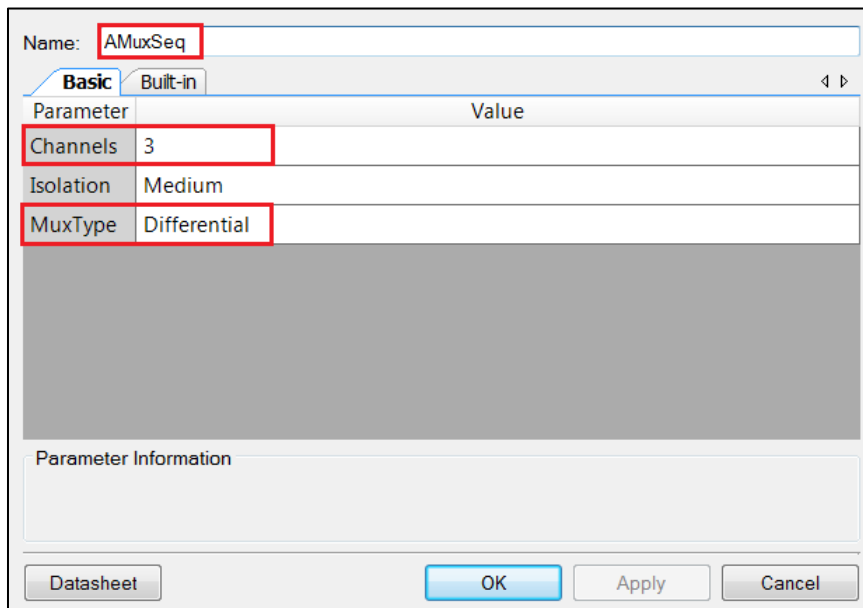
Sequenced channels: **1**

Channel	Enable	Resolution	Mode	AVG	Acq time	Conversion time	Limit detect	Saturation
0	<input checked="" type="checkbox"/>	12	Diff	<input checked="" type="checkbox"/>	A clks	48 us	<input type="checkbox"/>	<input type="checkbox"/>
INJ	<input type="checkbox"/>	12	Diff	<input type="checkbox"/>	A clks	3 us	<input type="checkbox"/>	<input type="checkbox"/>

Datasheet OK Apply Cancel

14. Place an Analog Mux Sequencer Component and configure it per [Figure 18](#). The AMUX Component will be used to mux multiple analog signals to the ADC input.

Figure 18. AMUX Configuration



Name: **AMuxSeq**

Basic Built-in

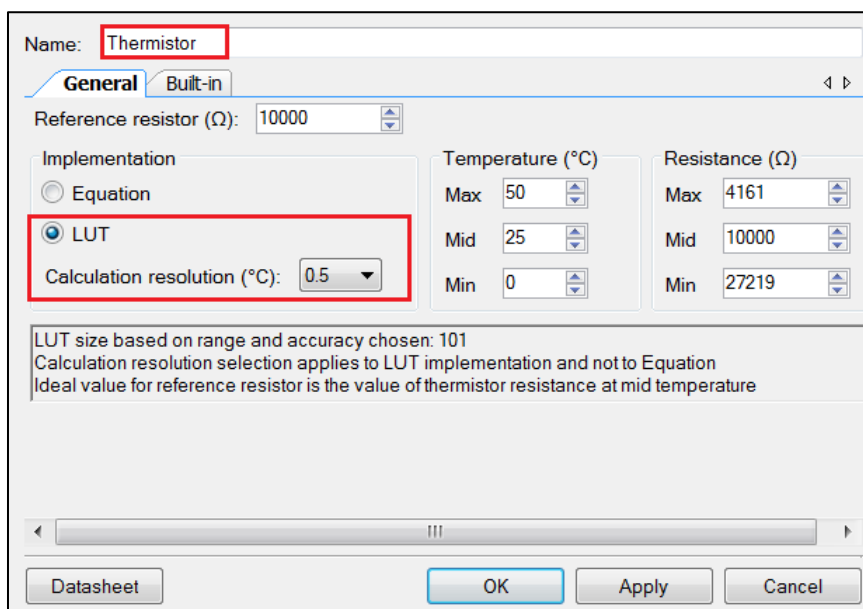
Parameter	Value
Channels	3
Isolation	Medium
MuxType	Differential

Parameter Information

Datasheet OK Apply Cancel

15. Place the Thermistor Calculator Component and configure it per [Figure 19](#). These settings configure this Component for temperature measurement using the thermistor.

Figure 19. Thermistor Calculator Configuration



Name: **Thermistor**

General Built-in

Reference resistor (Ω): 10000

Implementation

☐ Equation

☒ **LUT**

Calculation resolution ($^{\circ}\text{C}$): 0.5

Temperature ($^{\circ}\text{C}$)

Max	50
Mid	25
Min	0

Resistance (Ω)

Max	4161
Mid	10000
Min	27219

LUT size based on range and accuracy chosen: 101
 Calculation resolution selection applies to LUT implementation and not to Equation
 Ideal value for reference resistor is the value of thermistor resistance at mid temperature

Datasheet OK Apply Cancel

Note: For accurate temperature measurement, the Temperature ($^{\circ}\text{C}$) and the Resistance (Ω) column must be updated using the values given in the thermistor datasheet.

16. Place two Analog Pin Components, and configure them per [Figure 20](#) and [Figure 21](#).

Note: These pins are configured as both analog pins and digital outputs. The digital output will allow us to drive either pin to either Vdd or Vss from the firmware.

Figure 20. Analog Pin Configuration – V_HIGH

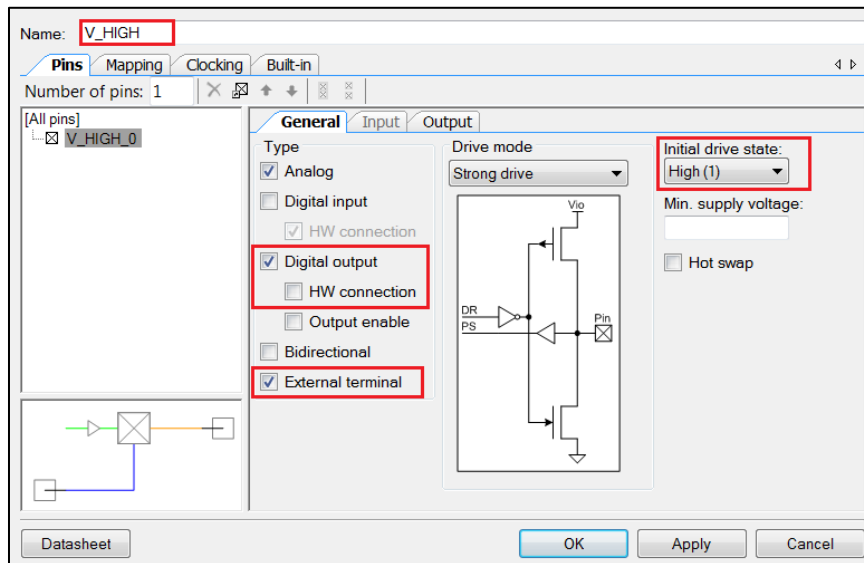
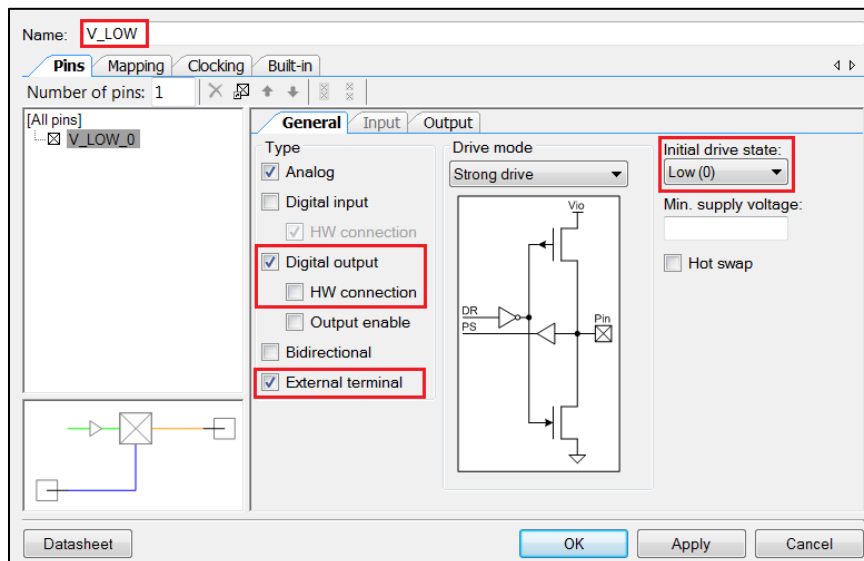


Figure 21. Analog Pin Configuration – V_LOW



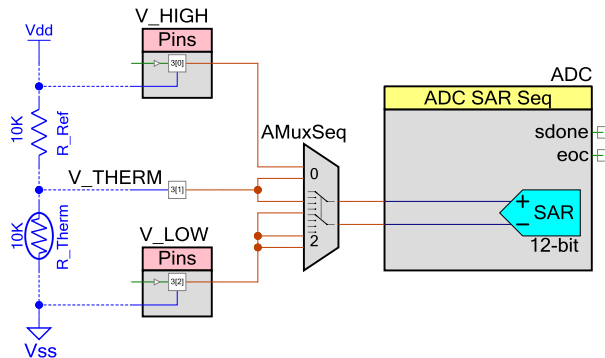
17. Place an Analog Pin Component and name it as “V_THERM.” Enable the **External terminal**.

18. Connect the Analog Mux Sequencer, Sequencing SAR ADC, and Analog Pin Components per the schematic shown in [Figure 22](#).

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.

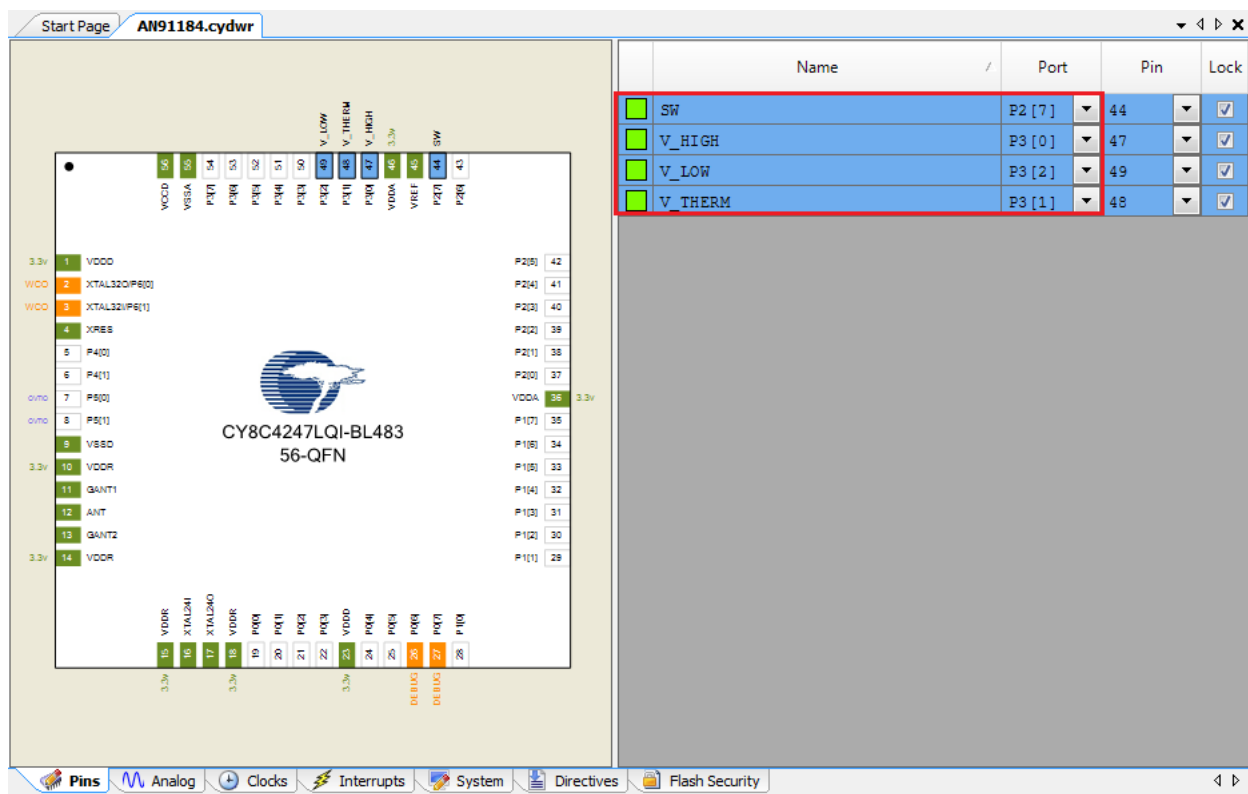
Figure 22. Schematic – Temperature Measurement

AMuX:
0: Reference Res
1: Thermistor
2: Offset Correction



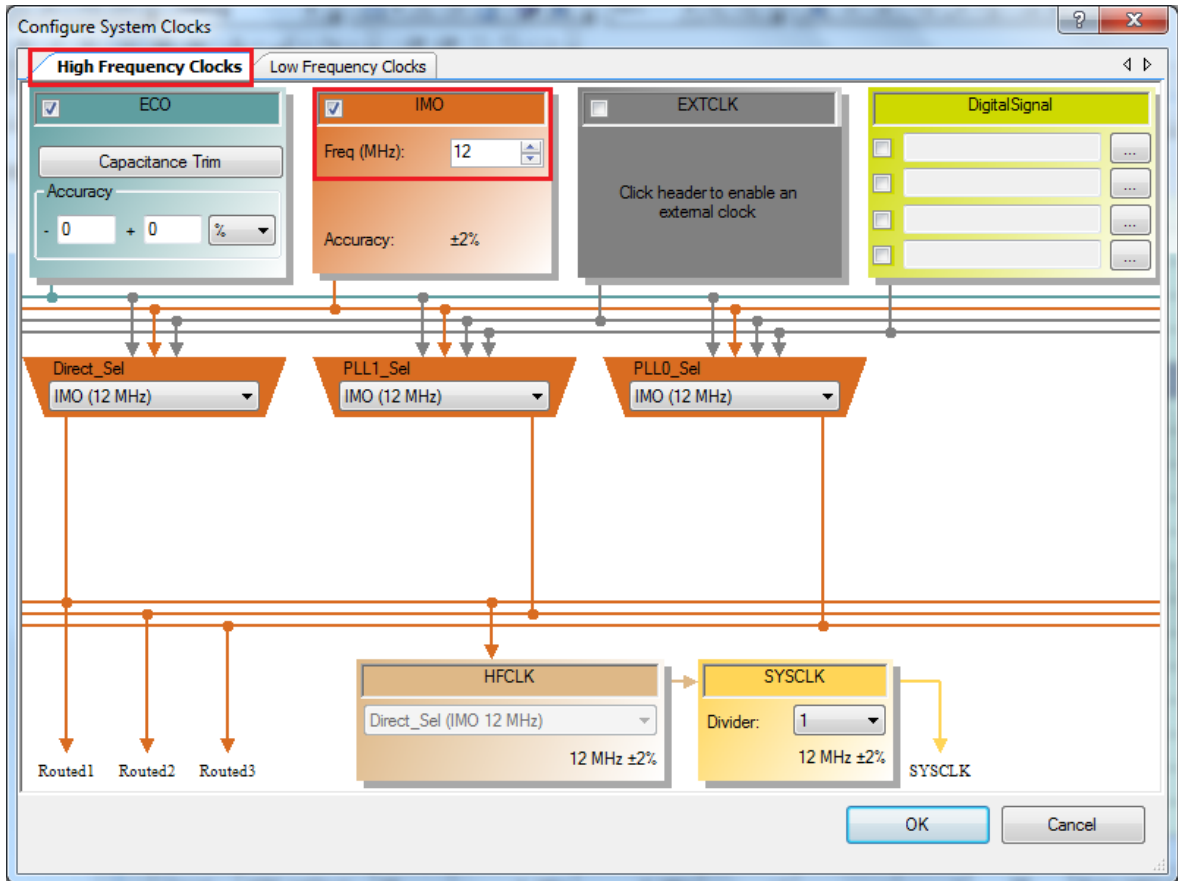
19. In the Pins tab of the design-wide resources window, connect the pins as shown in [Figure 23](#).

Figure 23. Pins Tab of the Design-Wide Resources Window



20. In the Clocks tab of the design-wide resources window, configure the IMO frequency as 12 MHz, as shown in Figure 24.

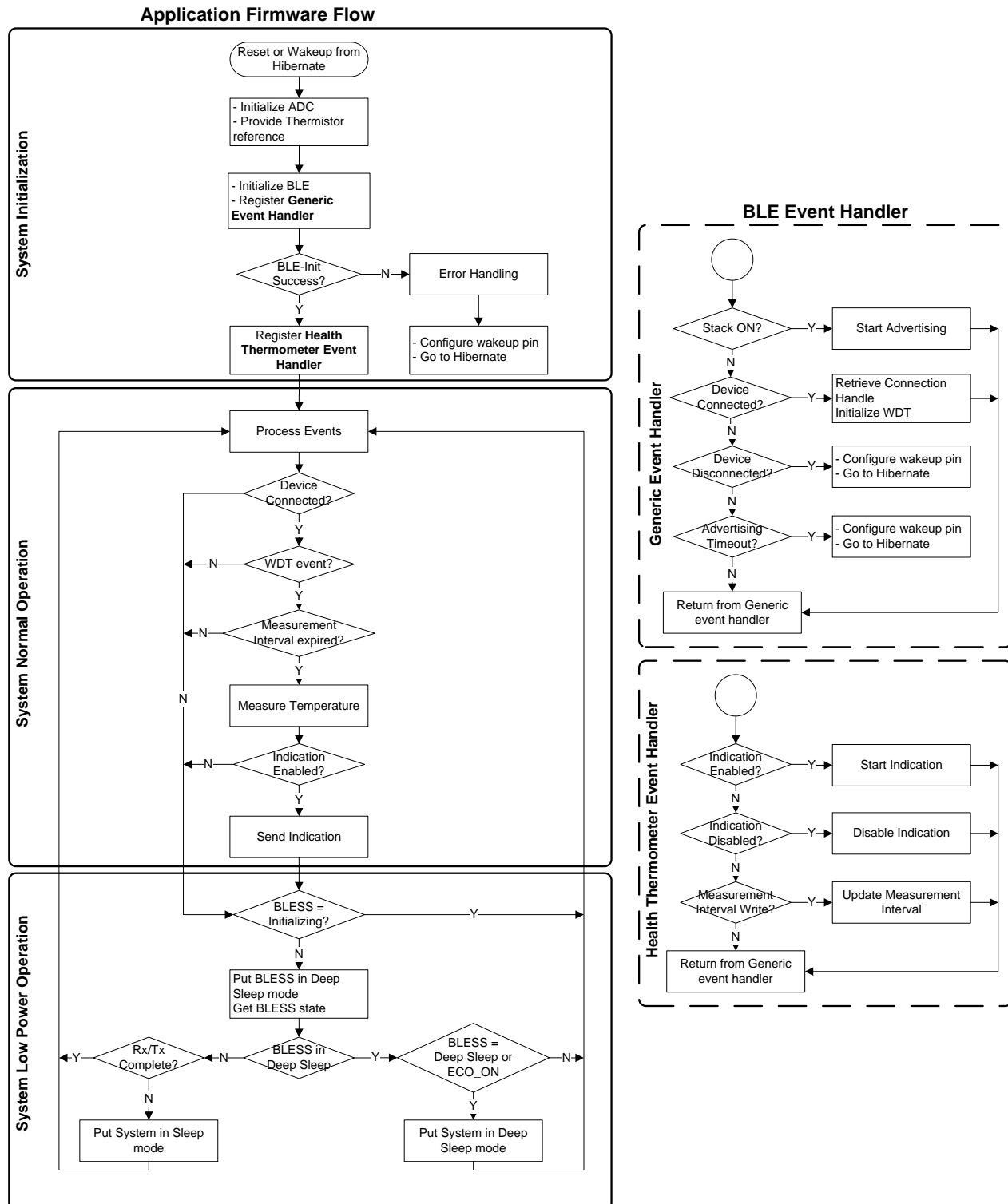
Figure 24. Clock Configuration



4.2 Configure the Firmware

Figure 25 shows the firmware flow for the Health Thermometer application.

Figure 25. 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	<p>This is the main firmware file. It has only one function:</p> <ul style="list-style-type: none"> 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	<p>It implements the common functions that are used for application control. It has the following functions:</p> <ul style="list-style-type: none"> 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. WakeupFromDeepSleep() – Restores the hardware blocks for normal operation.
Temperature.c/.h	<p>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:</p> <ul style="list-style-type: none"> 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	<p>It implements the watchdog timer functionality and keeps track of the system time. It has the following functions:</p> <ul style="list-style-type: none"> 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	<p>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:</p> <ul style="list-style-type: none"> 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.

4.2.1 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 25](#) 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);
```

4.2.2 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

```
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

```

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 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;
    }
}

```

4.2.3 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.

4.2.4 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.

4.2.5 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

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

4.3 Hardware Configuration

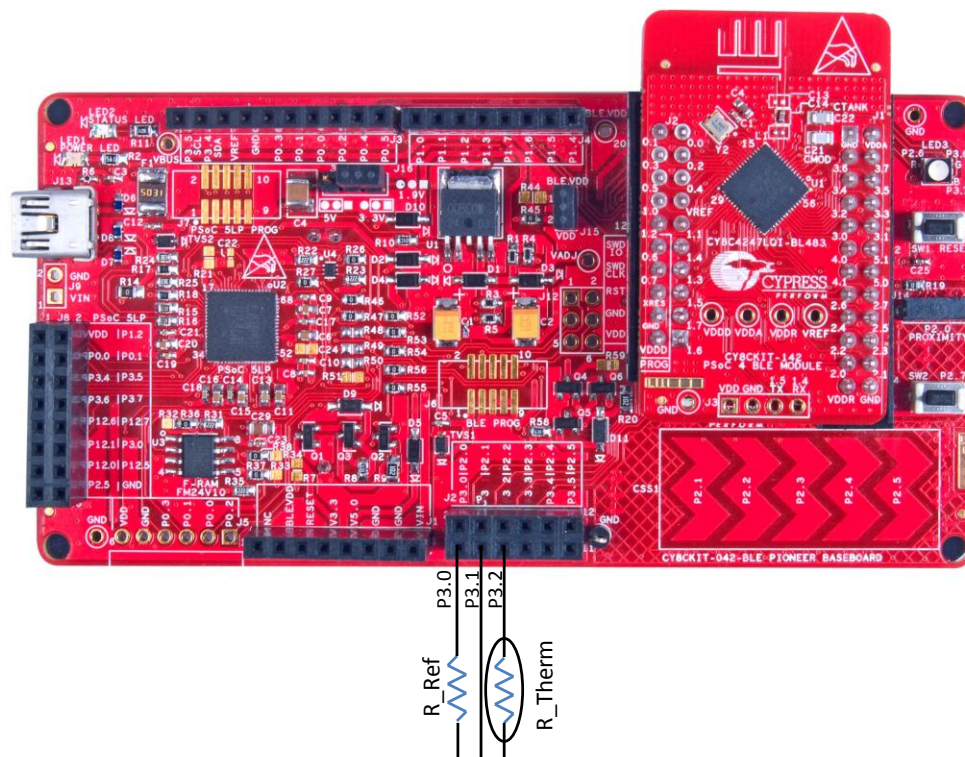
CY8CKIT-042-BLE Bluetooth Low Energy (BLE) Pioneer Kit is the development kit for Cypress PSoC 4 BLE and PSoC 4 BLE devices. Designed for flexibility, this kit offers footprint-compatibility with many third-party Arduino™ shields. The kit includes a provision to populate an extra header to support Digilent® Pmod™ peripheral modules. In addition, the board features a CapSense® slider, an onboard 1-Mb FRAM, 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 26](#).

Figure 26. External Thermistor Connection

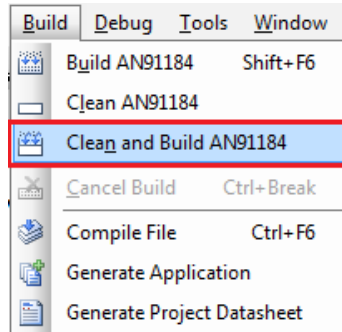


- **Simulated Sensor** – Instead of using a hardware sensor, the firmware simulates the temperature data and increments the temperature by 1 °C per measurement interval (default value is 1 second). Refer to the [Sensor Simulation](#) section for details.

4.4 Build and Program the Device

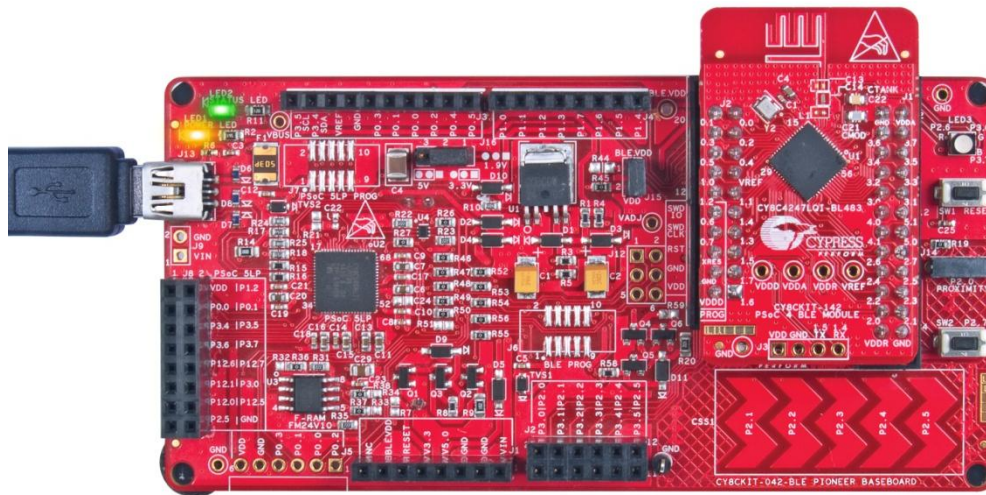
1. Select **Build > Build AN91184** to build and compile the firmware, as shown in Figure 27. The project should build without warnings or errors.

Figure 27. Build Project



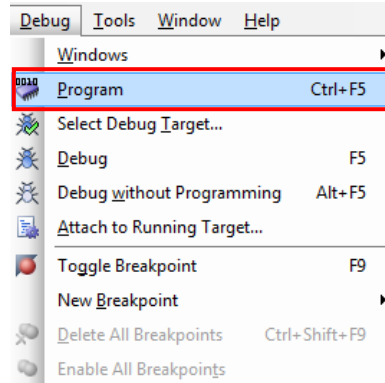
2. Plug the PSoC 4 BLE module (red module) to the BLE Pioneer baseboard, and then connect the kit to your PC using the USB Standard-A to Mini-B cable (see Figure 28). Allow the USB enumeration to complete on the PC.

Figure 28. Connect BLE Pioneer Baseboard to PCB Using a USB Cable



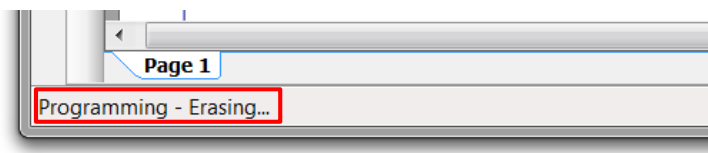
3. Select **Debug > Program**, as shown in Figure 29. If there is only one kit connected to the PC, the programming will start automatically. If multiple kits are present, PSoC Creator will prompt you to choose the kit to be programmed.

Figure 29. 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 30.

Figure 30. Programming Status



5 Application Testing

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

5.1 CySmart Central Emulation Tool

You can use the CySmart Central Emulation Tool along with the CY5670 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.

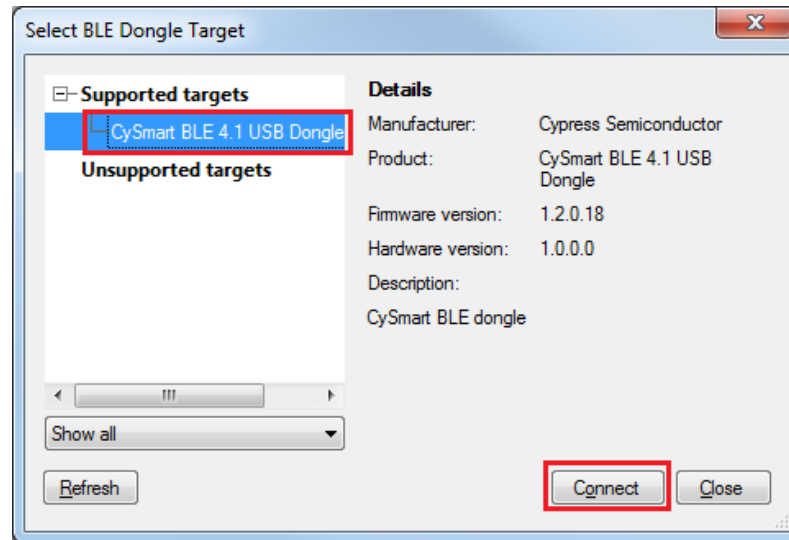
To verify the Health Thermometer application using the CySmart Central Emulation Tool, do the following:

1. Connect the BLE Dongle to the PC and start the CySmart Central Emulation Tool from **Start > All Programs > CySmart 1.2 > CySmart 1.2**.

The CySmart Central Emulation Tool detects the BLE Dongle connected to the USB drive.

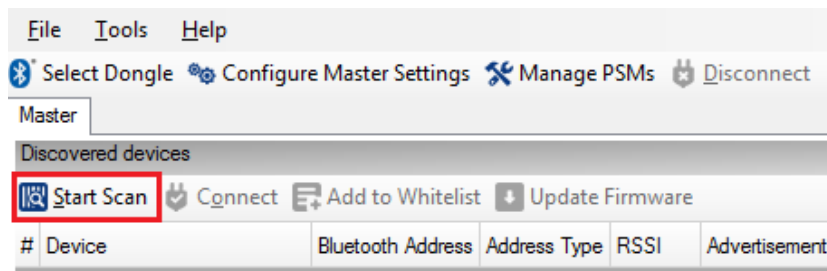
- Click **Connect** to connect to the BLE Dongle, as shown in Figure 31.

Figure 31. Select BLE Dongle Target



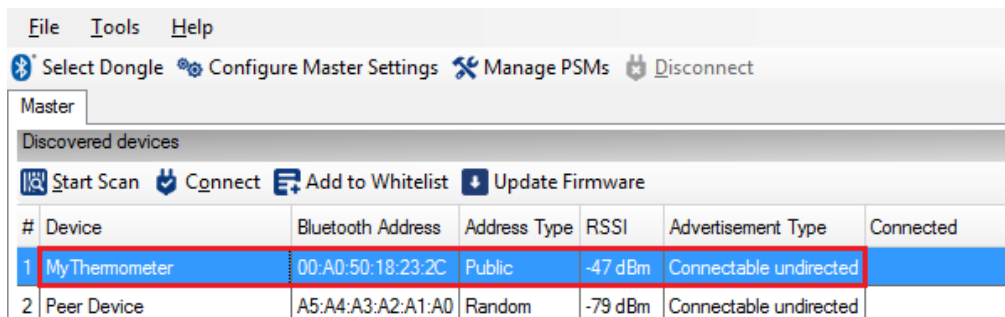
- When the PC is connected with the BLE Dongle, click **Start Scan** to find the BLE devices, as shown in Figure 32.

Figure 32. Start Scan



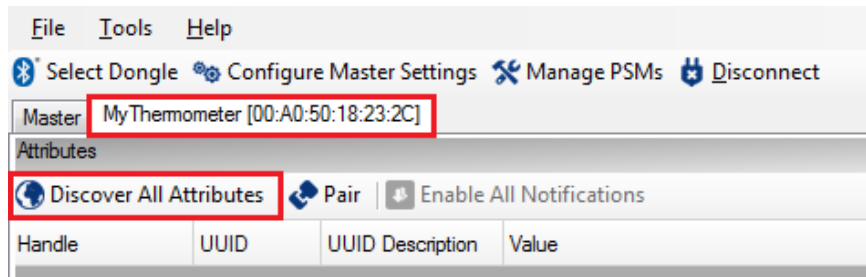
The discovered BLE devices are listed in the CySmart Central Emulation Tool window, as shown in Figure 33.

Figure 33. Discovered Devices



4. Select the device “MyThermometer” by clicking on the name, and then click **Connect**, as shown in [Figure 33](#).
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 34](#).

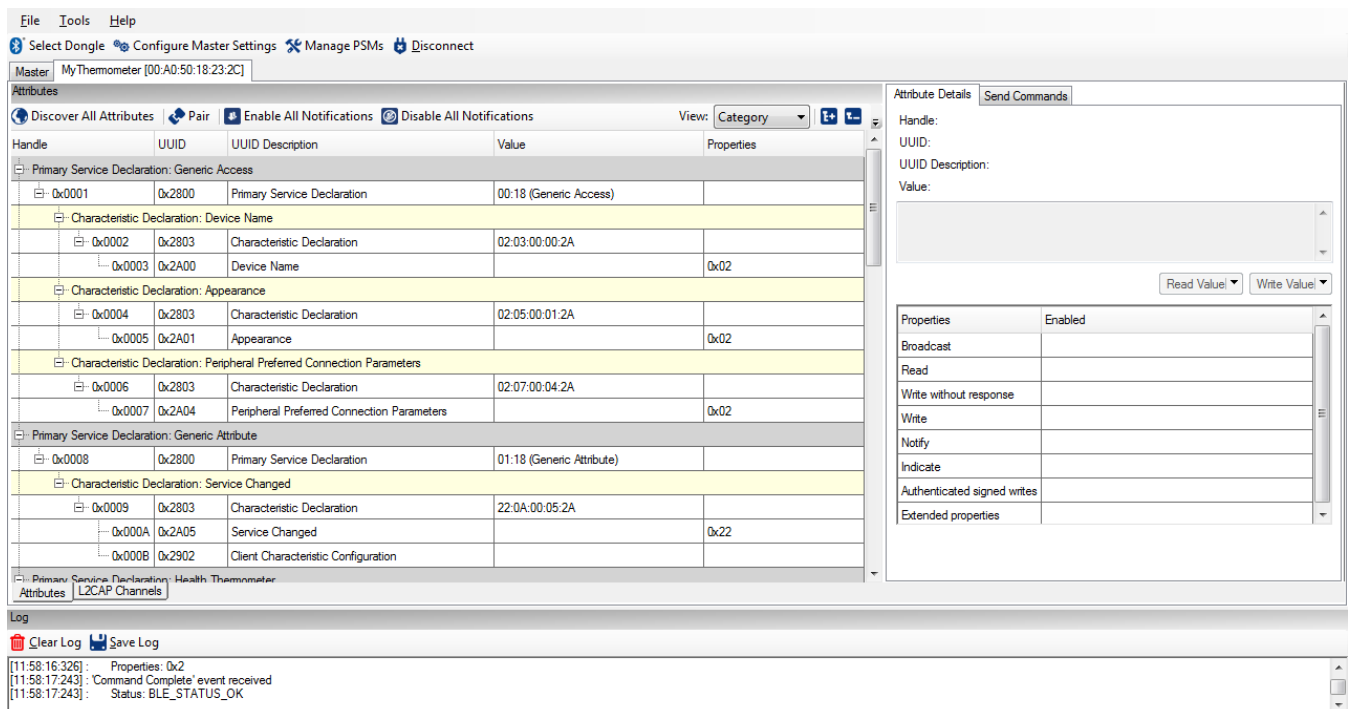
Figure 34. Connect to MyThermometer



5. To discover all the attributes exposed by the Peripheral “MyThermomemeter,” click **Discover All Attributes**, as shown in [Figure 34](#).

All discovered attributes are grouped and displayed as shown in [Figure 35](#).

Figure 35. Discovered Attributes



6. To read the measured temperature, select the **Client Characteristic Configuration** descriptor of the **Temperature Measurement** characteristic, as shown in [Figure 36](#) and [Figure 37](#).

Figure 36. 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				
0x0010	0x2803	Characteristic Declaration	02:11:00:1D:2A	
0x0011	0x2A1D	Temperature Type		0x02

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 37](#).

Note: Refer to [Client Characteristic Configuration Descriptor](#) for more details about bit definitions.

Figure 37. Enable Indications – Step B

Attribute Details
Send Commands

Handle: 0x000F
UUID: 0x2902
UUID Description: Client Characteristic Configuration
Value:

02

Read Value
Write Value

The “Temperature Measurement” attribute is updated with the measured temperature value, as shown in [Figure 38](#).

Figure 38. 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 °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](#).

5.2 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:

1. Open the CySmart app on your device, as shown in [Figure 39](#).

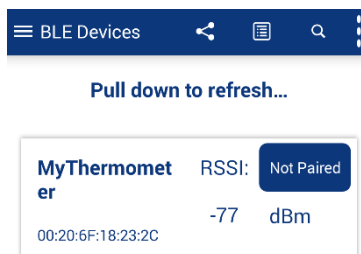
Note: The screenshots are for the CySmart Android app. The look and feel of the CySmart iOS app may differ slightly.

Figure 39. BLE Configuration



2. 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 40](#) shows the BLE devices in vicinity.

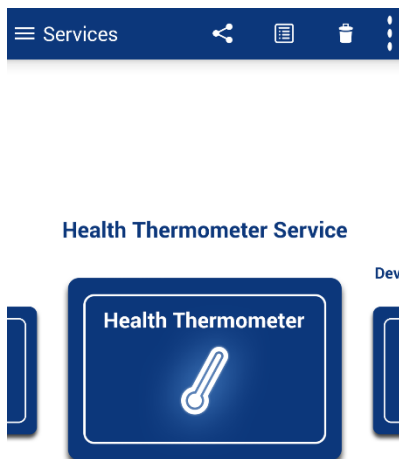
Figure 40. Device List



3. Connect to the device “MyThermometer” by clicking on the device name, as shown in [Figure 40](#).

4. 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 41](#).

Figure 41. Home Page



5. Select the Health Thermometer Service. It reports the current temperature and the Sensor Location, as shown in [Figure 42](#).

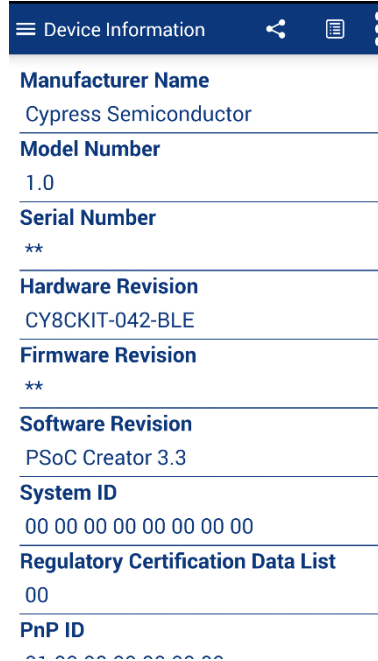
Figure 42. Health Thermometer Service



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

7. Select the “Device Information” Service. It shows the device information configured as a part of the project, as shown in [Figure 43](#).

Figure 43. Device Information Service



Device Information	
Manufacturer Name	Cypress Semiconductor
Model Number	1.0
Serial Number	**
Hardware Revision	CY8CKIT-042-BLE
Firmware Revision	**
Software Revision	PSoC Creator 3.3
System ID	00 00 00 00 00 00 00 00
Regulatory Certification Data List	00
PnP ID	01 00 00 00 00 00 00 00

5.3 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.

6 Related Application Notes

- [AN91267 – Getting Started with PSoC 4 BLE](#)
- [AN91162 – Creating a BLE Custom Profile](#)
- [AN92584 – Designing for Low Power and Estimating Battery Life for BLE Applications](#)
- [AN66477 – PSoC® 3, PSoC 4, and PSoC 5LP - Temperature Measurement with a Thermistor](#)

About the Author

Name: Uday Agarwal

Title: Application Engineer Senior

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.

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

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

[PSoC 1](#) | [PSoC 3](#) | [PSoC 4](#) | [PSoC 5LP](#)

Cypress Developer Community

[Community](#) | [Forums](#) | [Blogs](#) | [Video](#) | [Training](#)

Technical Support

cypress.com/go/support

PSoC is a registered trademark and PSoC Creator is a trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.



Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2015-2016. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.