



# AN1300: Understanding the Silicon Labs Bluetooth<sup>®</sup> Mesh Sensor Model Demonstration in SDK v2.x

---

The Bluetooth mesh SDK comes with two sample projects that create a wireless network of sensors and sensor clients using Bluetooth mesh technology. The examples assume use of Silicon Labs WSTKs for sensors and sensor clients, and the Silicon Labs Bluetooth Mesh mobile app as provisioner. In this document we discuss the basics of sensor models and describe the related sample applications in the SDK.

## KEY POINTS

- Short introduction to Bluetooth mesh sensor model
- Using the sensor example application
- Sensor example application code walkthrough

## 1. Introduction

This document focuses on explaining the Bluetooth mesh sensor demo, installed as part of the Bluetooth mesh SDK. For the most part the documentation centers on the example application and its usage flow, along with an explanation of key part of the source code. It also includes a brief discussion of some concepts of the sensor model specification that are important for understanding the example.

### 1.1 Sensor Model

The sensor model is Bluetooth mesh's method of interfacing with sensors. For a list of supported sensor types refer to the Bluetooth Mesh Device Properties [specifications](#). This model is made up of sensor states including descriptors, settings, cadence, data, and series columns. This model also defines the messages used for setting and reporting these states between client and server.

**Sensor Descriptors:** The sensor descriptors define the sensor property ID, to indicate the device's sensor type, the positive and negative tolerance of the sensor, the sampling function, the measurement period, and the update interval.

**Sensor Settings:** The sensor settings state controls the parameters of a sensor, such as sensitivity. The sensor setting property ID determines whether the sensor settings can be read and written as well as a raw setting's size and content.

**Sensor cadence:** The sensor cadence state controls how often the sensor data is published. Data can be published either through a trigger or a fast cadence. The trigger can be defined either by the sensor property ID or as a percentage change in the measured value. A fast cadence can be used if the measured value falls within a specified range.

**Sensor Data:** The sensor data state is constructed of a sensor property ID and a raw value. Multiple instances are permitted.

**Sensor Series Column:** Sensor measurements may be organized as arrays, conceptually as columns of data. The sensor series column state is made up of a raw Y value, raw X value, and column width. The sizes and contents of each of these is determined by the sensor property ID.

### 1.2 Sensor Messages

Each state in the sensor model has an acknowledged get message and an unacknowledged status message associated with it. A client requests the status message by sending the get message. In addition, writable sensor states such as the cadence and setting states also have both acknowledged and unacknowledged set messages.

### 1.3 Sensor Server and Client

The defined models are the Sensor Client, Sensor Server, and Sensor Setup Server. In any element where the Sensor Server is present, the Sensor Setup Server must also be present to allow configuration.

## 2. Bluetooth Mesh Sensor Project

This section describes how to create a generic sensor client and server project based on the Bluetooth Mesh SDK v2.x.

Prebuilt demonstration application images are provided for EFR32xG12, xG13 and xG21 parts. EFR32xG22 parts have limited support for Bluetooth Mesh (only LPN is supported). If you wish to modify the application, the following section describes how to create the corresponding project.

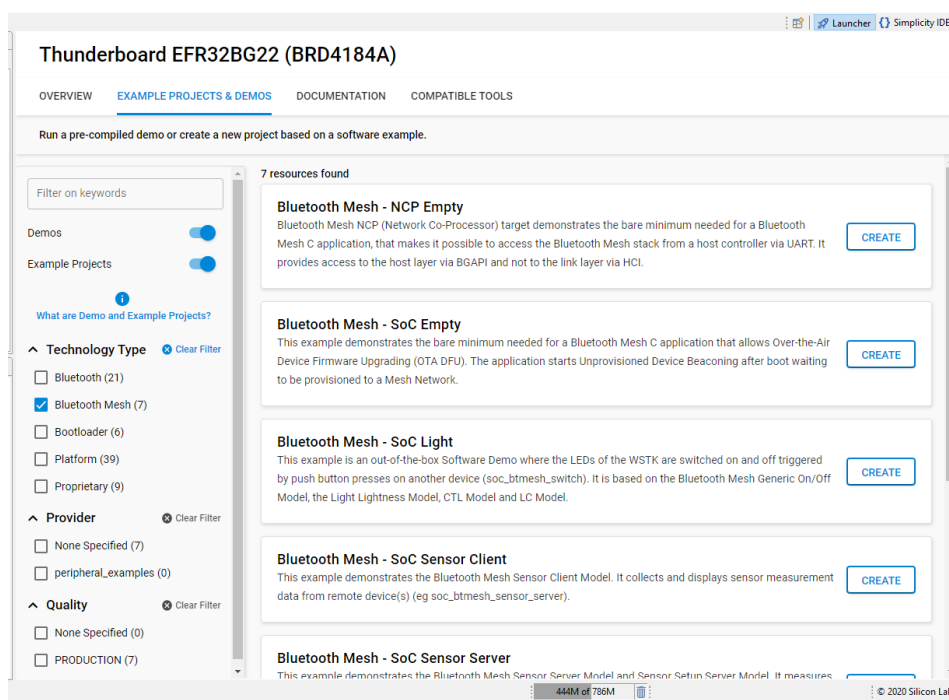
### 2.1 Requirements

- [Simplicity Studio 5](#)
  - Bluetooth Mesh SDK 2.0.0 or later, distributed through Simplicity Studio 5.
  - The pre-built demo binaries and source code are included in the SDK.
  - Simplicity Studio has a network analyzer capable of capturing and decoding Bluetooth mesh packets.
  - The actual code development can be done with Simplicity Studio, IAR EWARM, or command line tools.
  - Used for discovering and provisioning devices
- Silicon Labs EFR32BG Wireless Starter Kits
  - Used for discovering and provisioning devices.

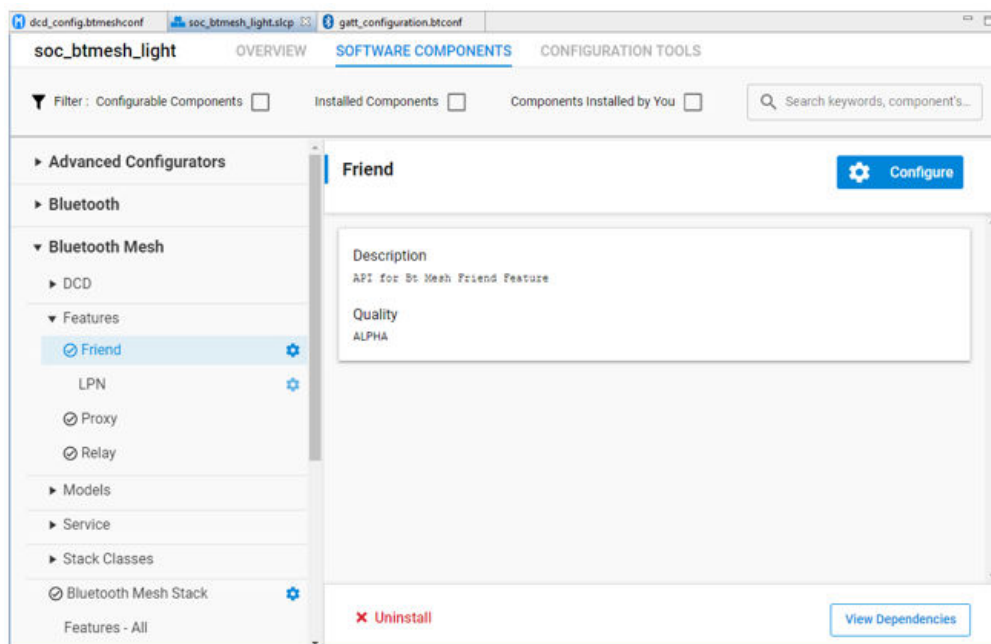
## 2.2 Bluetooth Mesh Sensor Client and Server

Plug a kit to your computer and launch Simplicity Studio 5. The kit should now appear in the Debug Adapter view as well as in the launcher:

Click the **Example Projects & Demos** tab. Under Technology Types, filter on **Bluetooth Mesh**. Next to the **Bluetooth Mesh - SoC Sensor Client** project, click **[Create]**.



Click the **Software Components** tab, and expand the Bluetooth Mesh components group to see the installed features.



Project files autogenerate, with progress reflected in the lower right of the Simplicity IDE. Build the project. Repeat for the **Bluetooth Mesh - SoC Sensor Server** example.

For more information on how to configure a node in the Bluetooth Mesh SDK v2., refer to *UG472: Bluetooth® Mesh Node Configurator User's Guide for SDK v2.x*.

### 3. Bluetooth Mesh Sensor Demonstration

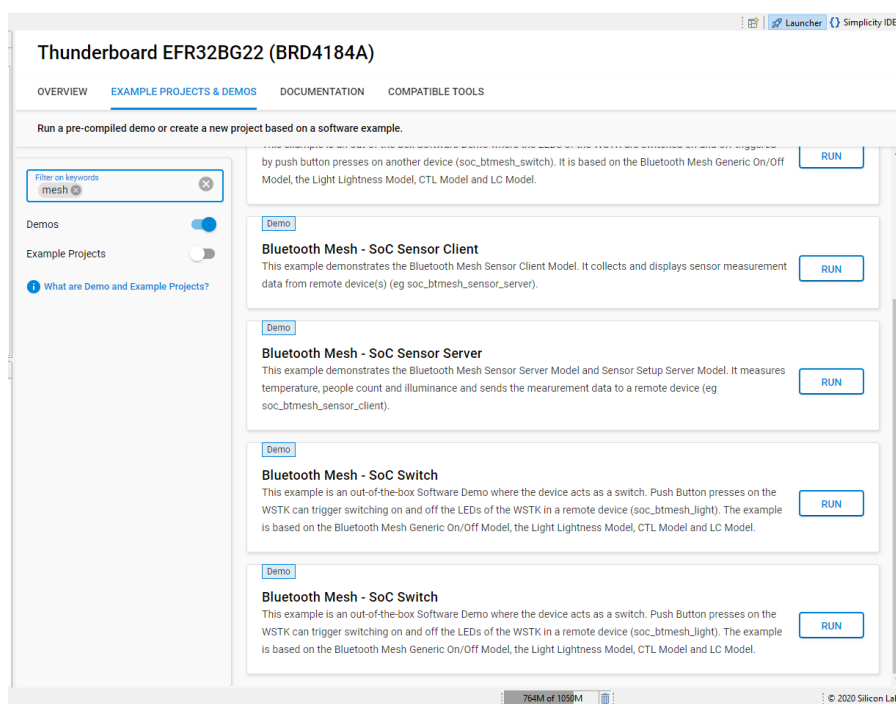
#### 3.1 Requirements

- [Simplicity Studio](#)
  - Bluetooth Mesh SDK 1.5.0 or later, distributed through Simplicity Studio.
  - The pre-built demo binaries and source code are included in the SDK.
  - Simplicity Studio has a network analyzer capable of capturing and decoding Bluetooth mesh packets.
  - The actual code development can be done with Simplicity Studio, IAR EWARM, or command line tools.
- [Silicon Labs Bluetooth Mesh Mobile Application](#)
  - Used for discovering and provisioning devices.
  - Includes network, group and publish-subscribe setup.
  - Allows device configuration for the sensor models.
- For the full experience at least two [Silicon Labs Blue Gecko SoC Wireless Starter Kits](#) are needed.
  - 1 kit is used as the sensor client.
  - 1 kit is used as the sensor server.
  - EFR32BG12, EFR32MG12, EFR32BG13, EFR32MG13 and EFR32xG21 SoCs as well as the BGM13P, and BGM13S
  - Modules support Bluetooth mesh software.

Note that EFR32xG22 SoCs and BGM220 have limited Bluetooth Mesh support (LPN only).

#### 3.2 Load the Demonstrations on the Target

Open Simplicity Studio 5 with a compatible SoC wireless kit plugged to the computer. Click the **Example Projects & Demos** tab. To see only the demos, turn off the Example Projects and, if you have more than one SDK installed, enter 'mesh' in the keyword field. Next to either **Bluetooth Mesh - SoC Sensor Client** or **Bluetooth Mesh - SoC Sensor Server**, click **RUN**.



### 3.3 Mesh Network Implementation

The demonstration implementation process can be divided into four main phases:

1. Unprovisioned mode – After the demo firmware is installed, the device starts in unprovisioned mode.
2. Provisioning – The devices are provisioned to a Bluetooth mesh network and network security is set up.
3. Configuration – The group, publish and subscribe, and application security are configured.
4. Normal operation – The sensor server(s) can be controlled by the client(s).

In the first phase, all the devices are unprovisioned and transmitting unprovisioned beacons. They do not have any network keys or application keys configured, and publish and subscribe settings are not set. In this state the devices are simply waiting for the provisioner to assign them into a Bluetooth mesh network, and configure publish and subscribe settings and mesh models. In this state the devices can be detected by the smartphone application.

In the provisioning phase the provisioner adds sensor servers and clients to the Bluetooth mesh network. A network key is generated and distributed to the nodes and each node is assigned a unicast address.

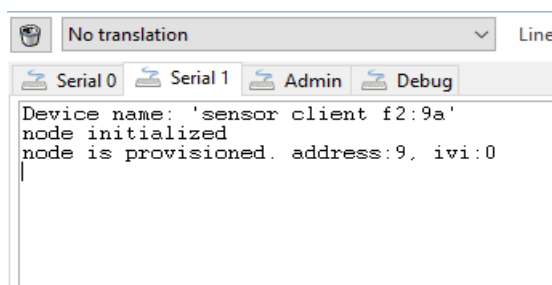
In the configuration phase the provisioner configures groups, publish and subscribe settings, application-level security, and mesh models.

After provisioning and configuration, the Bluetooth mesh network is operational and clients can be used to configure and request data from the sensors.

### 3.4 Running the Example

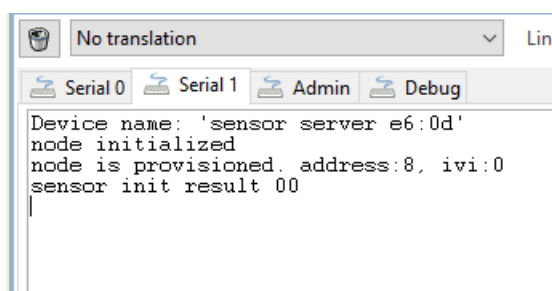
This section assumes you have installed the **BT Mesh – Sensor Client Example** demo binary to one of the WSTKs and the **BT Mesh – Sensor Server Example** to the other, as described in section [3.2 Load the Demonstrations on the Target](#)

- Provision and configure the sample apps using the Silicon Labs Bluetooth Mesh mobile application.
- Open a serial console in Simplicity Studio for each. If you reset the devices this is what you will observe on the console for the client and server:



```

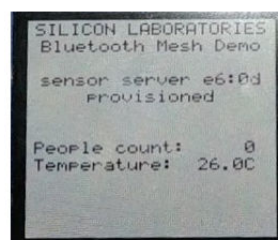
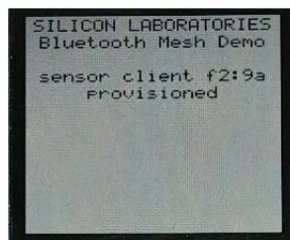
No translation
Serial 0 Serial 1 Admin Debug
Device name: 'sensor client f2:9a'
node initialized
node is provisioned. address:9, ivi:0
    
```



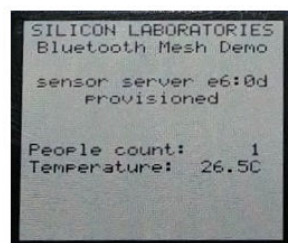
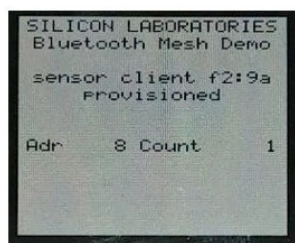
```

No translation
Serial 0 Serial 1 Admin Debug
Device name: 'sensor server e6:0d'
node initialized
node is provisioned. address:8, ivi:0
sensor init result 00
    
```

- And this is what you will see in the WSTK display for the client and server.

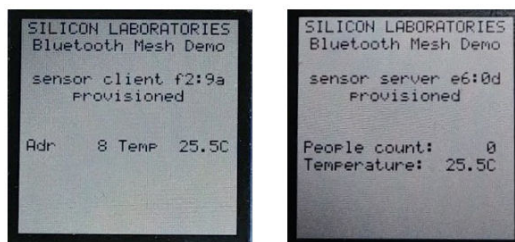


- Press PB0 on the client WSTK to select the people count sensor. Now press PB1 on the client WSTK to register the server. The LCD on the client displays the people count reported by the server.



- Press PB1 on the server to increase the count or PB0 to decrease the count.

- Press PB0 on the client WSTK again to select the temperature sensor. Now the LCD on the client WSTK will display the temperature reported by the server.





## 4. Code Walkthrough

As of BT Mesh SDK v2.x, the code structure of both the stack API and the code examples have been reworked. The Bluetooth Mesh API now abstracts away much of the event handling in generated files and allows the user to focus on application development.

The sections below describe the code in the application source of the examples (app.c)

### 4.1 Unprovisioned Mode, Provisioning, and Configuration

In unprovisioned mode, both examples behave the same way. The unprovisioned device simply starts sending unprovisioned beacons and waits for a provisioner to provision and configure it.

After receiving the system\_boot event (`sl_bt_evt_system_boot_id`), the application checks if a button is pressed in the `handle_boot_event()` routine. If yes, it calls the function `sl_btmesh_initiate_factory_reset()`, which closes connections if any exist and performs a factory reset by erasing NVM3 storage. The factory reset is also done after receiving the mesh\_node\_reset event (`sl_bt_evt_mesh_node_reset_id`). If no button is pressed, then the name of the device is set based on the Bluetooth address, and the function `sl_btmesh_node_init()` is called to initialize the Bluetooth mesh node stack.

The event `sl_btmesh_evt_node_initialized_id` indicates that the Bluetooth mesh node stack initialization is complete. This event also includes information about the node status. The application first checks the provisioning status. If the node is not provisioned (the default state when the device is first powered up after programming) then the application starts unprovisioned beaconing by calling `sl_btmesh_node_start_unprov_beaconing()`.

The API call `sl_btmesh_node_start_unprov_beaconing` takes one parameter (bearer) that selects which bearers are used (PB-ADV, PB-GATT, or both). In this example, both bearers are used. Because the PB-GATT bearer is enabled, the device will begin advertising its provisioning GATT service. This allows the smartphone application to detect unprovisioned nodes. When unprovisioned beaconing has been started the application waits for the provisioner (in this case, the smartphone app) to start provisioning. Start of provisioning is indicated with the event `sl_btmesh_evt_node_provisioning_started_id` (see `sl_btmesh_provisioning_decorator.c`). This is handled in the application code through the `sl_btmesh_on_node_provisioning_started()` callback.

During provisioning, no actions are required from the user application. The configuration of network keys and other operations are handled automatically by the Bluetooth mesh stack. Both the light and the switch applications simply start blinking the two LEDs on the WSTK to indicate that provisioning is in progress. Then they wait for the event `sl_btmesh_evt_node_provisioned_id` (see `sl_btmesh_provisioning_decorator.c` for more details) that indicates provisioning is complete. This is handled in the application code through the `sl_btmesh_on_node_provisioned()` callback.

## 4.2 Sensor Server Example

This section describes basic operation of the **Bluetooth Mesh – SoC Sensor Server**. It is assumed that the node is already provisioned and publish-subscribe settings have been configured by the smartphone app.

The sensor server supports two types of sensors: a People Count sensor and a Present Ambient Temperature sensor. The People Count sensor is simulated by the buttons on the starter kit: PB0 decreases the count and PB1 increases the count. The count is maintained in a 16-bit unsigned integer. The Present Ambient Temperature sensor is a Silicon Labs Si7021. Temperature is reported in units of 0.5 degrees Celsius as a signed 8-bit integer.

Upon receiving the `sl_btmesh_evt_node_initialized_id` event and call to the corresponding callback `handle_node_initialized_event()`, the sensor server node initializes the sensors by calling `sl_btmesh_sensor_server_node_init()`. This function sets the people count to 0 and initializes the temperature sensor hardware. Next, it enables GPIO interrupts for WSTK buttons PB0 and PB1. Otherwise the node starts unprovisioned beaconing and waits for a provisioner. Once provisioned and initialized, the sensor server node simply waits for messages from the client. Events generated by messages from the client are handled in the `sl_btmesh_sensor_server.c`.

Please note that sensor settings and cadence are not supported at this time so the message handlers are stubs.

- Get requests are handled by `handle_sensor_server_get_request()`. The `property_id` is queried. If the value is non-zero and is a supported value then the sensor data for that property is returned. If the `property_id` is zero then all supported sensor data is returned. If the value is non-zero and is an unsupported value, the data length is set to zero to indicate an unsupported property. In all cases, `gecko_cmd_mesh_sensor_server_send_status()` is called to send the status to the client.
- Get Series requests are handled by `handle_sensor_server_get_series_request()`. Neither sensor properties supported in this example include either Series State so only the `property_id` is sent back to the client.
- Get Column requests are handled by `handle_sensor_server_get_column_request()`. Neither of the sensor properties include Column State so the same data is sent back to the client, per the specification.
- Publishing sensor data is handled by `handle_sensor_server_publish_event()` when the publish period expires. Data from both sensors is published.
- The Cadence and Settings States for the properties in this example are not included so their handlers simple return the property ID, per the specification.

### 4.3 Sensor Client Example

This section describes basic operation of the **Bluetooth Mesh – SoC Sensor Client**. It is assumed that the node is already provisioned and publish-subscribe settings have been configured by the smartphone app. The main purpose of the sensor client is to request sensor data from the sensor server. The sensor client supports two types of sensors: a people counting sensor and a temperature sensor.

Upon receiving the `sl_btmesh_evt_mesh_node_initialized_id` event, the sensor client node performs the following actions:

- Initializes the sensor client model by calling `sl_btmesh_cmd_mesh_sensor_client_init()`.
- Handles the buttons on the WSTK through calling `sl_btmesh_button_press_cb()`.
- Requests a sensor descriptor by calling `sl_btmesh_sensor_client_get_descriptor()` with address 0x0000 to publish the message using the publish parameters set by the configuration client.
- Starts a timer to periodically request sensor data.

The WSTK buttons are used as follows:

- PB0 to select the sensor property\_id to interact with.
- PB1 to reset the list of registered devices.

When PB0 is pressed, `sensor_client_change_current_property()` is called to handle the change. This function increments the variable `current_property`, which is used to index the properties array containing a list of the supported properties.

When PB1 is pressed `update_registered_devices()` is called to find and register the devices that support the currently selected property. This is done by calling `sl_btmesh_sensor_client_get_descriptor()` with the selected property as a parameter. A recurring timer is started with a 2000 ms interval. This timer is used to request the sensor data by calling `sl_btmesh_cmd_mesh_sensor_client_get()`.

When a message is received from a sensor server, one of the sensor events are triggered. These events are as follows:

Event	Description	Behavior
<code>sl_btmesh_evt_sensor_client_descriptor_status_id</code>	Indicates that a descriptor status message has been received.	Adds the server to the list of registered devices if it was not previously registered.
<code>sl_btmesh_evt_sensor_client_status_id</code>	Indicates that a sensor status message has been received.	Verifies that the data came from a registered device, determines which type of sensor the data comes from, then saves and displays it.



Smart.  
Connected.  
Energy-Friendly.



**Products**

[www.silabs.com/products](http://www.silabs.com/products)



**Quality**

[www.silabs.com/quality](http://www.silabs.com/quality)



**Support and Community**

[community.silabs.com](http://community.silabs.com)

#### Disclaimer

Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required, or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.

#### Trademark Information

Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, ClockBuilder®, CMEMS®, DSPLL®, EFM®, EFM32®, EFR®, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, Gecko OS, Gecko OS Studio, ISOModem®, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress®, Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.



Silicon Laboratories Inc.  
400 West Cesar Chavez  
Austin, TX 78701  
USA

<http://www.silabs.com>