Optimizing Battery Budget with BG22

This lab procedure walks through the steps to optimize Bluetooth applications for battery-operated devices with the EFR32BG22. In this first part of the lab, a simple beacon will be created to demonstrate some of the low-power features of the EFR32BG22. In the second part of the lab, optimizations will be made to improve the power consumption in connection-based operation.

KEY POINTS

- Create a low-power beacon
- Create a low-power thermometer
- Use EFR Connect mobile app to view device changes

1 Prerequisites

For this lab you will need the following:

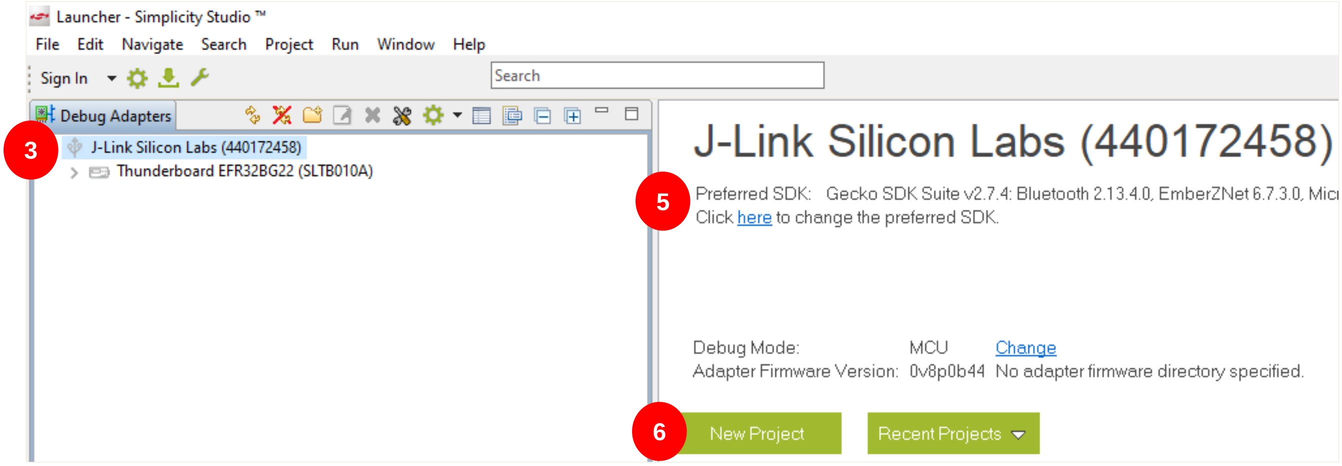
- Complete complementary labs:
  - Lab 1 – Out of the Box Beaconing
- EFR32BG22 Thunderboard (SLTB010A)
- Micro-USB to USB Type-A cable
- Gecko SDK Suite 2.7.4 or later
  - Bluetooth SDK 2.13.4.0 or later
- EFR Connect Mobile App

2 Creating a Low-Power Beacon

2.1 Creating the Project

In order to create a low-power beacon, we will start with the SOC-Empty example project. The SOC-Empty project is a minimal project that should be used as a starting point for custom Bluetooth applications. It implements basic functionality that enables peripheral connectivity and contains a minimal GATT database that can be expanded to fit custom application requirements.

1. Launch Simplicity Studio
2. Connect the Thunberboard BG22 to your PC using a micro-USB cable
3. Once the device is connected to your PC, you should see it listed in the **Debug Adapters** window in Simplicity Studio
4. Select the J-Link for the device to display the associated Demos, Example Projects, and Documentation
5. Be sure to set the preferred SDK to Gecko SDK Suite v 2.7.4 or newer
6. Click the New Project button in the Launcher window

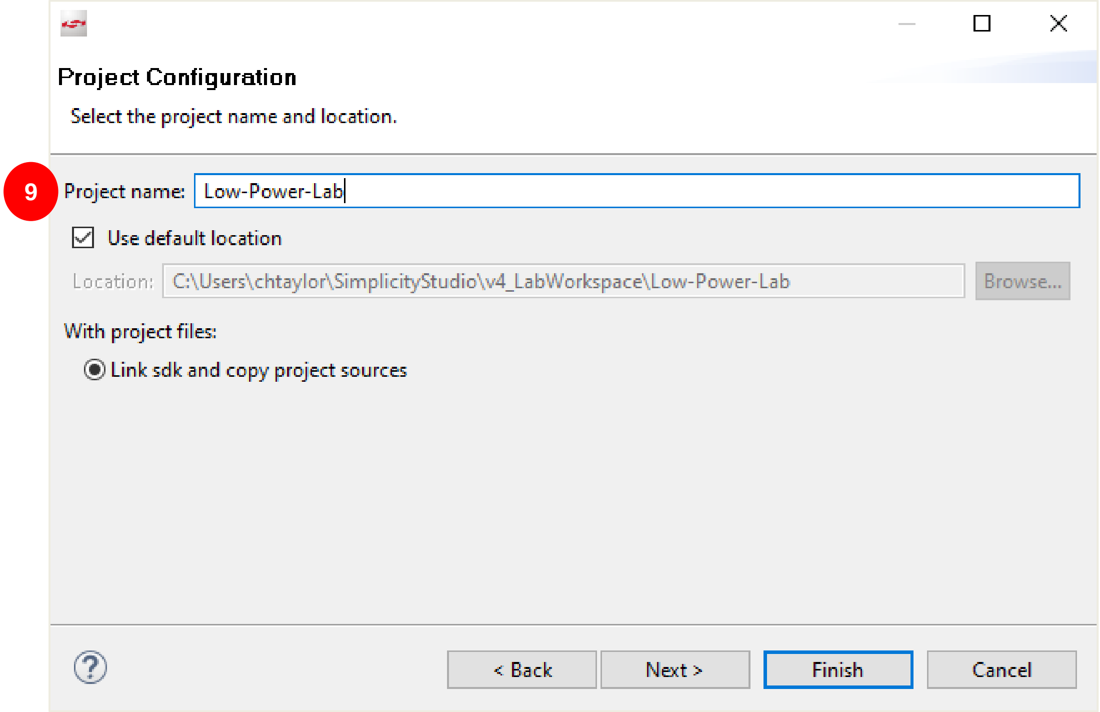


J-Link Silicon Labs (440172458)

Preferred SDK: Gecko SDK Suite v2.7.4: Bluetooth 2.13.4.0, EmberZNet 6.7.3.0, Mic
Click here to change the preferred SDK.

Debug Mode: MCU Change
Adapter Firmware Version: 0v8p0b44 No adapter firmware directory specified.

7. For the application type, select **Bluetooth SDK** and click **Next >**
8. Select the **SOC-Empty** application and click **Next >**
9. Set the project name to **Low-Power-Lab** and click **Next >**



Project Configuration

Select the project name and location.

Project name: Low-Power-Lab

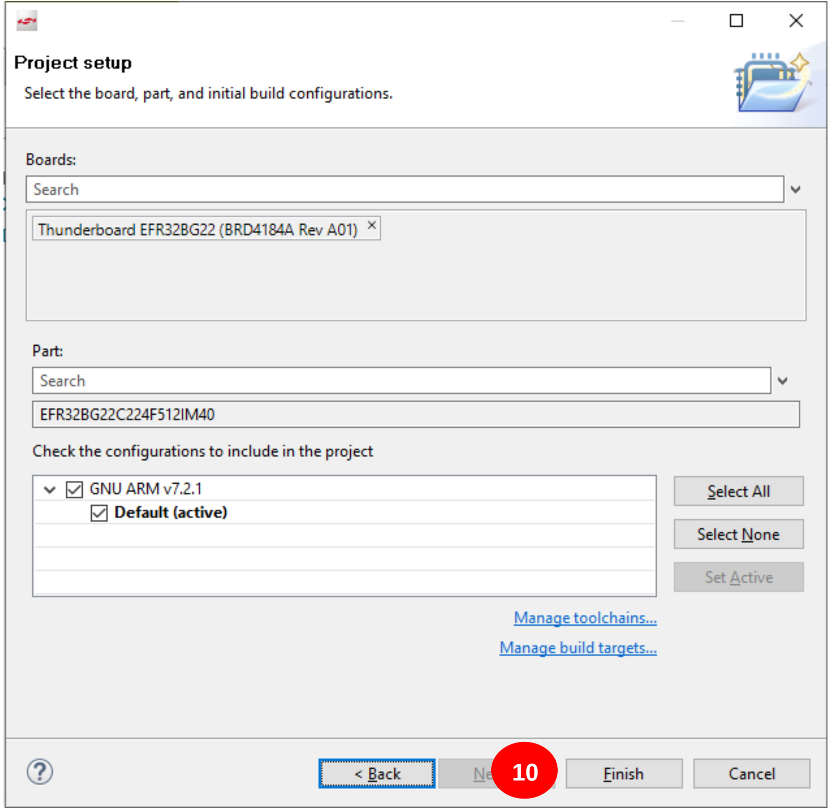
☑ Use default location

Location: C:\Users\chtaylor\SimplicityStudio\v4_LabWorkspace\Low-Power-Lab

With project files:

● Link sdk and copy project sources

10. Verify that the board, part, and build configurations have been configured correctly, then click **Finish**



Project setup

Select the board, part, and initial build configurations.

Boards:

Thunderboard EFR32BG22 (BRD4184A Rev A01)

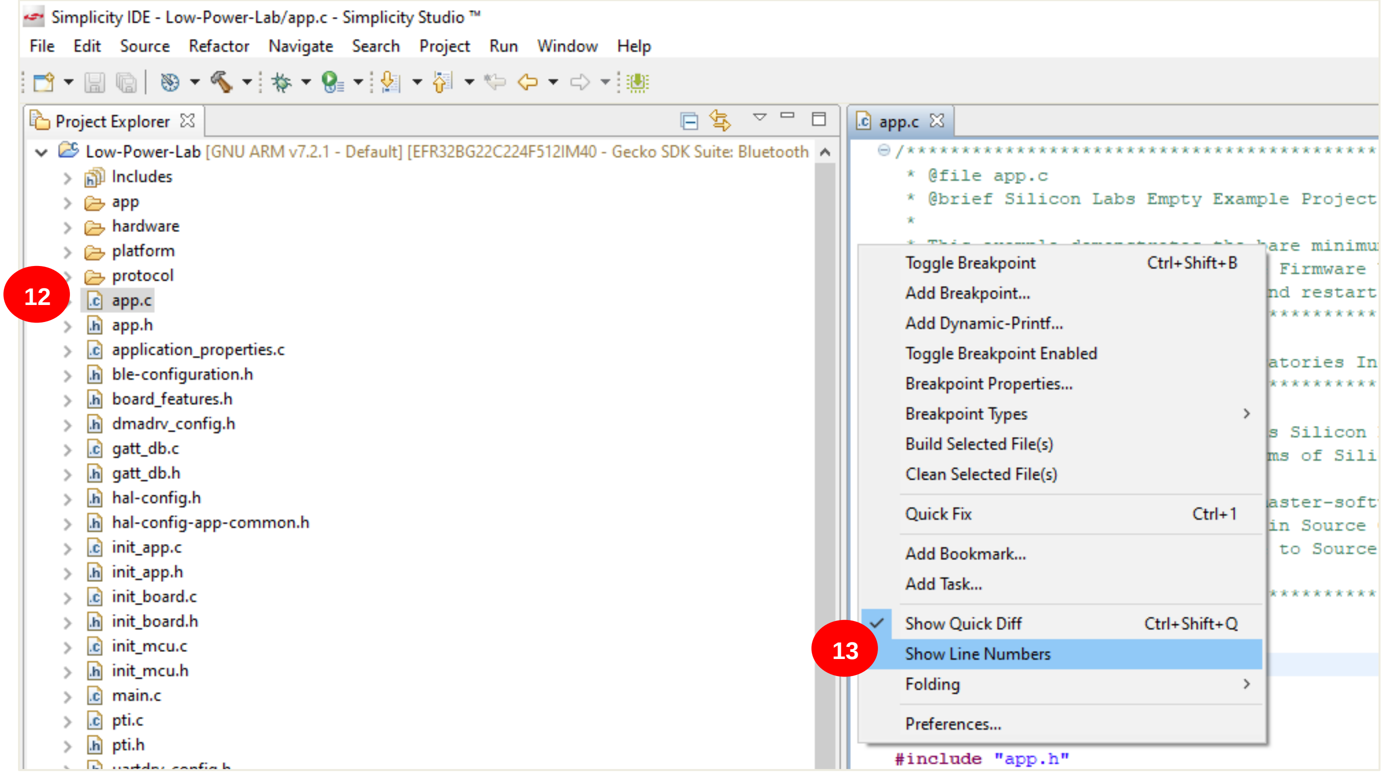
Part:

EFR32BG22C224F512IM40

Check the configurations to include in the project

☑ GNU ARM v7.2.1
  ☑ **Default (active)**

11. Once the project is created, the Simplicity IDE will be launched
12. Open the **app.c** file from the **Project Explorer**
13. Right-click in the left margin of the Editor and select **Show Line Numbers**. This will make it simpler to locate the code that needs to be modified for the lab



Simplicity IDE - Low-Power-Lab/app.c - Simplicity Studio ™

Project Explorer

Low-Power-Lab [GNU ARM v7.2.1 - Default] [EFR32BG22C224F512IM40 - Gecko SDK Suite: Bluetooth

- Includes
- app
- hardware
- platform
- protocol
- app.c
- app.h
- application_properties.c
- ble-configuration.h
- board_features.h
- dmadrv_config.h
- gatt_db.c
- gatt_db.h
- hal-config.h
- hal-config-app-common.h
- init_app.c
- init_app.h
- init_board.c
- init_board.h
- init_mcu.c
- init_mcu.h
- main.c
- pti.c
- pti.h

```
/*********************************************
 * @file app.c
 * @brief Silicon Labs Empty Example Project
 *
```

```
#include "app.h"
```

2.2 Modify the Application

In order to create a Bluetooth beacon and optimize for battery life, modifications will be made to many of the default parameters of the SOC-Empty project. The first change will be to the TX output power. The default TX output power used by the Bluetooth stack for the BG22 is +6dBm. Because this is a low-power beacon application, we will set the TX ouput to 0 dBm. This setting will extend the battery life while still enabling a range of approximately 30 meters or more.

Of the 40 channels in BLE, three channels (37, 38, and 39) are reserved for broadcasting advertising packets that contain information about the broadcasting node. These three channels are known as primary advertising channels. Beacons work by taking advantage of Bluetooth’s ability to broadcast packets with a small amount of customizable embedded data on these advertising channels. By default, the Bluetooth stack advertises on all three of the primary advertising channels. However, the power consumption can be reduced by choosing to advertise on only one of these channels.

The default project uses a 100ms advertisement interval. However, typical battery-powered applications will transmit much less frequently in order to minimize power consumption. For this lab, we will increase the advertisement interval to 1 second.

We will also reduce the advertising payload to only include the device name. This is to reduce the on-time of the radio, rather than sending all of the information in the GATT (device name, manufacturer, OTA service, etc.)

Next, we will set the device to non-connectable. When the device advertises as connectable, it automatically switches to RX mode for some amount of time after every advertisement to listen for an incoming connection request. Since we’re just beaconing, we don’t need to make a connection to a device and thus can lower the power consumption by setting the device to be non-connectable.

The example projects use VCOM to print out debug messages from the application over the UART interface, but there is a small increase to the power consumption by having this enabled. For this lab we will disable the debug messages.

Here are the steps required to make the changes above:

1. Beginning in **app.c**, navigate to line 70 and set the TX output power to 0 dBm using the following command

```
          /* Set tx power to 0dBm */
          gecko_cmd_system_set_tx_power(0);
```

2. Configure the device to advertise on a single advertising channel by modifying the advertising channel map. Add the following in the boot event *prior* to the start advertising command

```
          /* Set adv on channel 37 only */
          gecko_cmd_le_gap_set_advertise_channel_map(0, 1);
```

3. Modify the advertising payload such that it advertises the device name “**BG22**” by adding the following *prior* to the start advertising command. The 1st byte of the array is the length of the element (excluding the length byte itself) and the 2nd byte is the advertising data (AD) type which specifies what data is included in the element. The AD type is defined by the Bluetooth SIG.

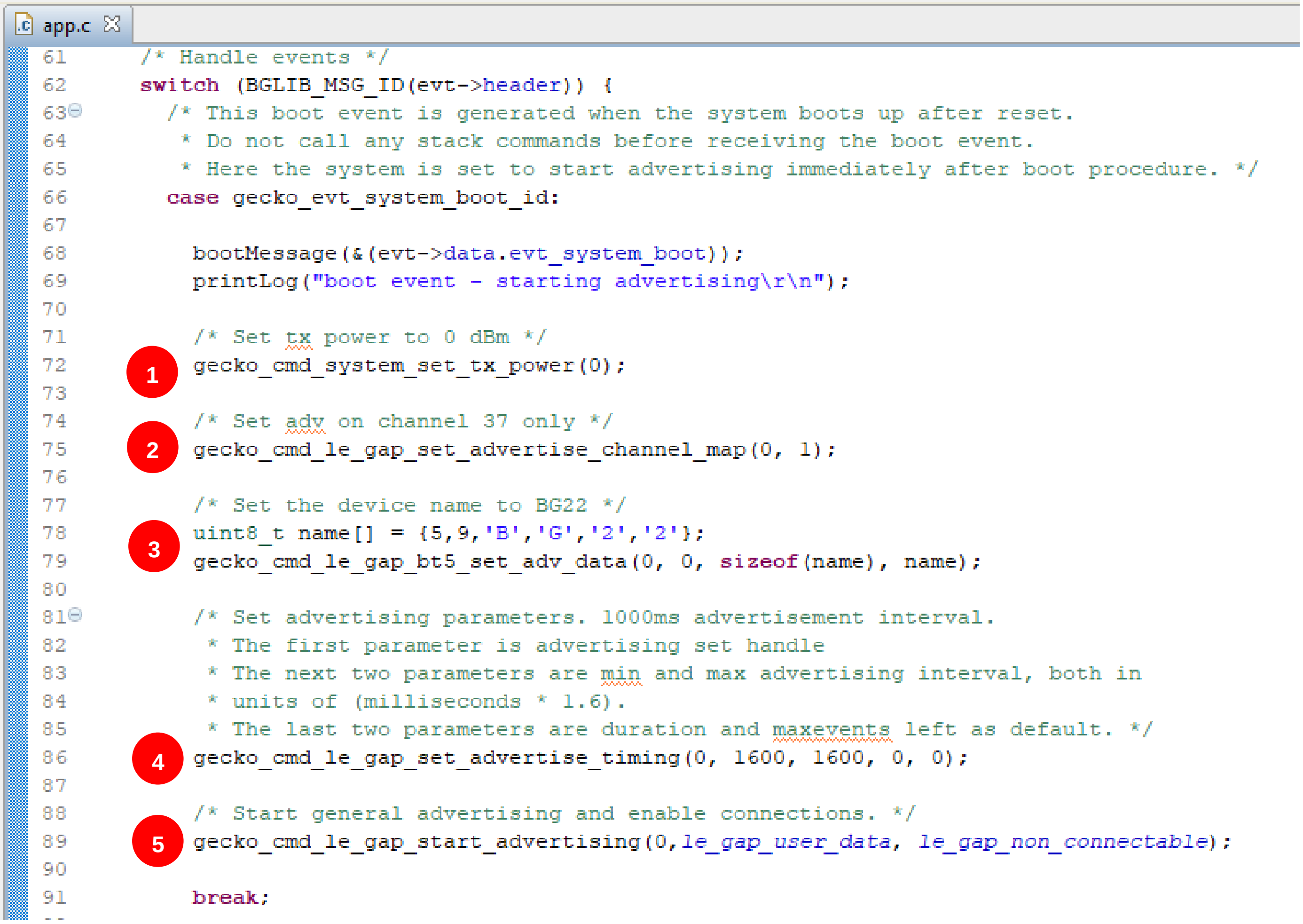
```
          /* Set the device name to BG22 */
          uint8_t name[] = {5,9,'B','G','2','2'};
          gecko_cmd_le_gap_bt5_set_adv_data(0, 0, sizeof(name), name);
```

4. Increase the advertising interval to 1 second by modifying the advertise timing

```
/* Set advertising parameters. 1000ms advertisement interval.
 * The first parameter is advertising set handle
 * The next two parameters are min and max advertising interval, both in
 * units of (milliseconds * 1.6).
 * The last two parameters are duration and maxevents left as default. */

  gecko_cmd_le_gap_set_advertise_timing(0, 1600, 1600, 0, 0);
```

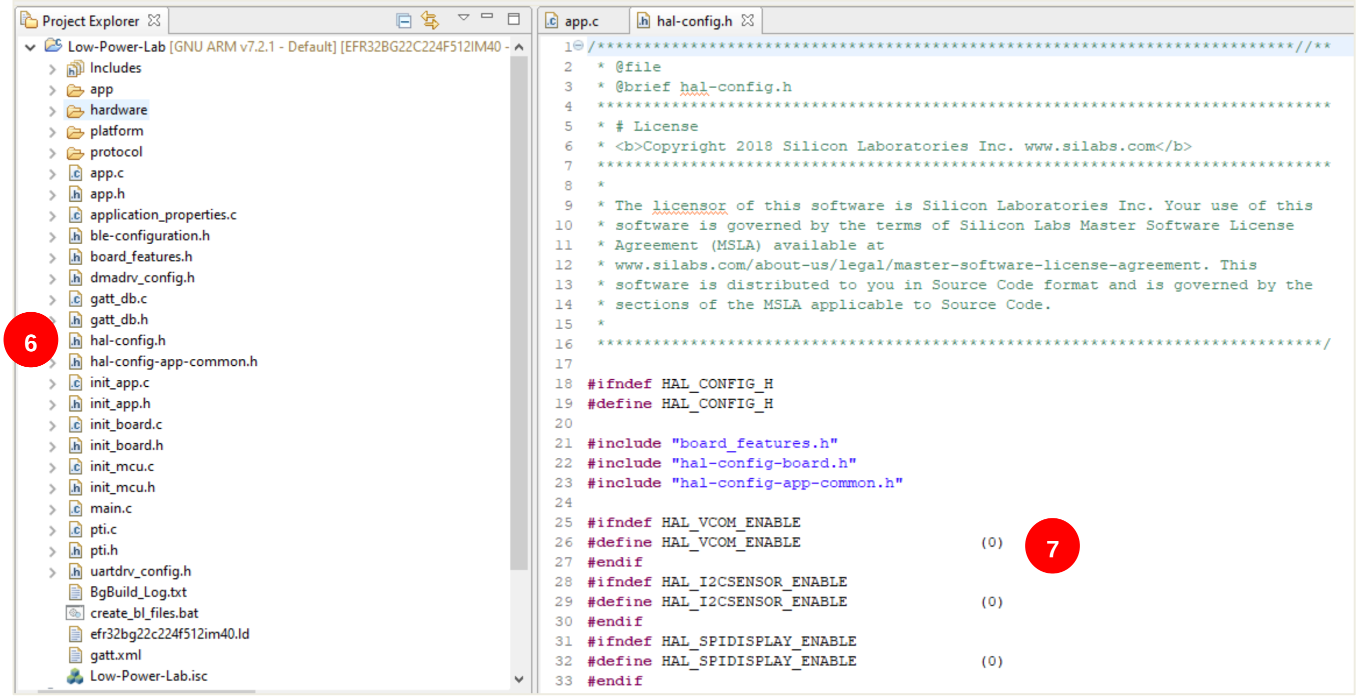
5. Modify the start advertising command to advertise the user data set above and to set the device to be non-connectable

```
/* Start advertising and disable connections. */
gecko_cmd_le_gap_start_advertising(0,le_gap_user_data, le_gap_non_connectable);
```



```
    /* Handle events */
    switch (BGLIB_MSG_ID(evt->header)) {
      /* This boot event is generated when the system boots up after reset.
       * Do not call any stack commands before receiving the boot event.
       * Here the system is set to start advertising immediately after boot procedure. */
      case gecko_evt_system_boot_id:

        bootMessage(&(evt->data.evt_system_boot));
        printLog("boot event - starting advertising\r\n");

        /* Set tx power to 0 dBm */
        gecko_cmd_system_set_tx_power(0);

        /* Set adv on channel 37 only */
        gecko_cmd_le_gap_set_advertise_channel_map(0, 1);

        /* Set the device name to BG22 */
        uint8_t name[] = {5,9,'B','G','2','2'};
        gecko_cmd_le_gap_bt5_set_adv_data(0, 0, sizeof(name), name);

        /* Set advertising parameters. 1000ms advertisement interval.
         * The first parameter is advertising set handle
         * The next two parameters are min and max advertising interval, both in
         * units of (milliseconds * 1.6).
         * The last two parameters are duration and maxevents left as default. */
        gecko_cmd_le_gap_set_advertise_timing(0, 1600, 1600, 0, 0);

        /* Start general advertising and enable connections. */
        gecko_cmd_le_gap_start_advertising(0,le_gap_user_data, le_gap_non_connectable);

        break;
```

6. Open **hal-config.h**
7. Disable the VCOM by setting **HAL_VCOM_ENABLE** to **0**

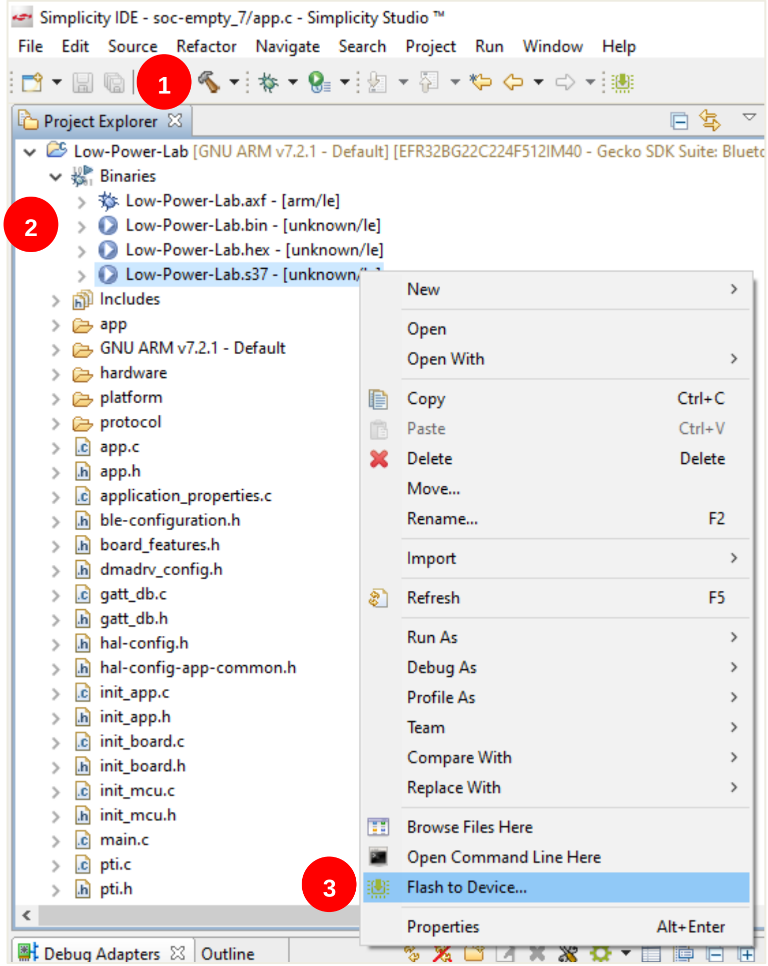


```
/***************************************************************************//**
 * @file
 * @brief hal-config.h
 *******************************************************************************
 * # License
 * <b>Copyright 2018 Silicon Laboratories Inc. www.silabs.com</b>
 *******************************************************************************
 *
 * The licensor of this software is Silicon Laboratories Inc. Your use of this
 * software is governed by the terms of Silicon Labs Master Software License
 * Agreement (MSLA) available at
 * www.silabs.com/about-us/legal/master-software-license-agreement. This
 * software is distributed to you in Source Code format and is governed by the
 * sections of the MSLA applicable to Source Code.
 *
 ******************************************************************************/

#ifndef HAL_CONFIG_H
#define HAL_CONFIG_H

#include "board_features.h"
#include "hal-config-board.h"
#include "hal-config-app-common.h"

#ifndef HAL_VCOM_ENABLE
#define HAL_VCOM_ENABLE                       (0)
#endif
#ifndef HAL_I2CSENSOR_ENABLE
#define HAL_I2CSENSOR_ENABLE                  (0)
#endif
#ifndef HAL_SPIDISPLAY_ENABLE
#define HAL_SPIDISPLAY_ENABLE                 (0)
#endif
```

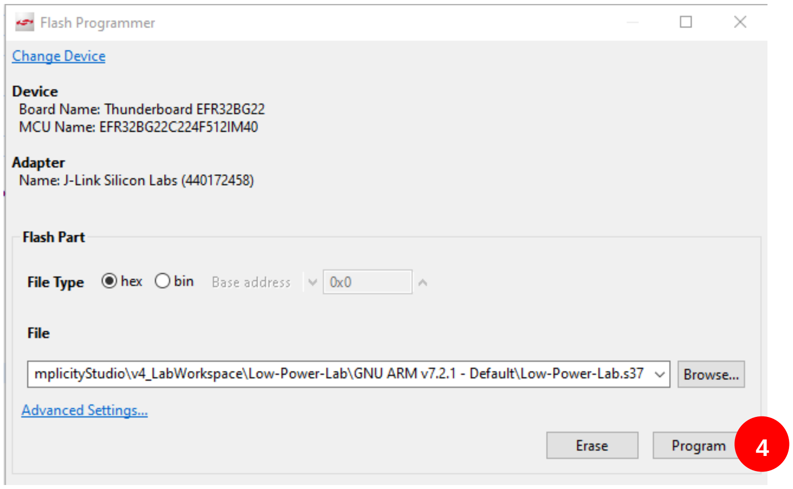
2.3 Build and Flash the Project

Once the changes above have been made, you are now ready to build the project and flash it to your device.

1. With the project selected, click the **Build** icon (🔨) in the toolbar or right-click on the project and select **Build Project**
2. Once the project has built successfully, open the **Binaries** folder
3. Right-click the **Low-Power-Lab.s37** file and select **Flash to Device**



4. When the Flash Programmer launches, click **Program** to flash the device



Device
Board Name: Thunderboard EFR32BG22
MCU Name: EFR32BG22C224F512IM40

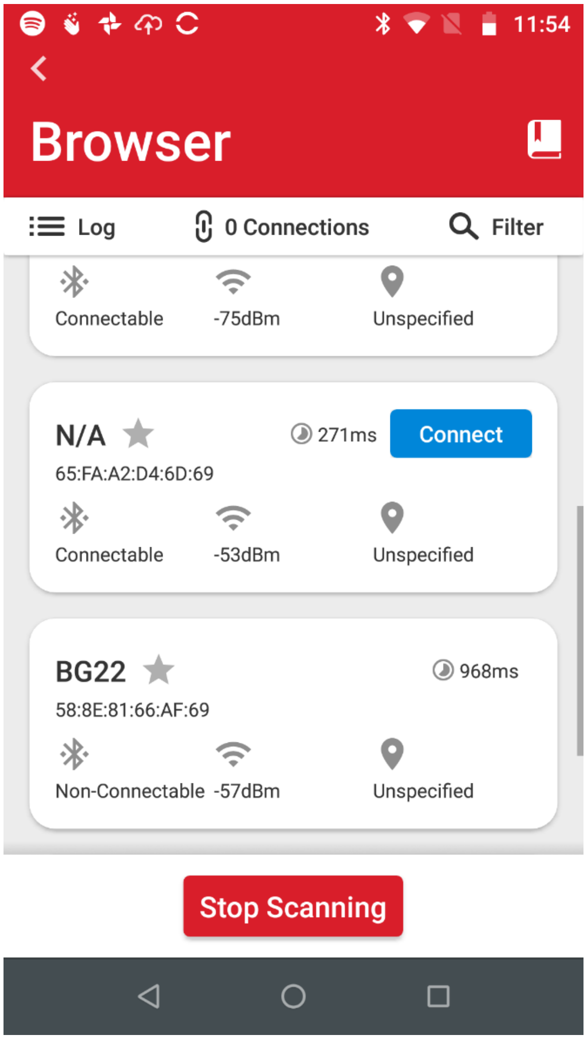
Adapter
Name: J-Link Silicon Labs (440172458)

File Type ● hex ○ bin Base address 0x0

File
mplicityStudio\v4_LabWorkspace\Low-Power-Lab\GNU ARM v7.2.1 - Default\Low-Power-Lab.s37

2.4 Verify the Changes using the EFR Connect App

To verify that your beacon is in fact advertising, launch the **EFR Connect** app on your mobile phone and open the **Browser**, which can be found on the **Develop** tab. You should find your device advertising via the Bluetooth Browser. You will notice that the device is greyed-out because it is non-connectable.



Connectable -75dBm Unspecified

N/A 271ms
65:FA:A2:D4:6D:69
Connectable -53dBm Unspecified

BG22 968ms
58:8E:81:66:AF:69
Non-Connectable -57dBm Unspecified

2.5 Energy Profiler (Hardware Not Included)

For further development beyond what is offered on the Thunderboard BG22, Silicon Labs recommends the EFR32xG22 Wireless Starter Kit (WSTK): https://www.silabs.com/products/development-tools/wireless/efr32xg22-wireless-starter-kit

The WSTK offers additional debug capabilities, including Advanced Energy Monitoring (AEM) hardware which, combined with Simplicty Studio's Energy Profiler tool, enables real-time power consumption measurement for the on-board device or external target devices. For more information about measuring power consumption on EFR32 devices, refer to AN969: https://www.silabs.com/documents/public/application-notes/an969-measuring-power-consumption.pdf

Using a WSTK and the Energy Profiler to measure the current consumption of the default SOC-Empty project in advertising mode, the averaged current consumption is measured to be about 114 µA. When measuring the current consumption of the low-power beacon created above, the average current consumption is measured to be < 3 µA.

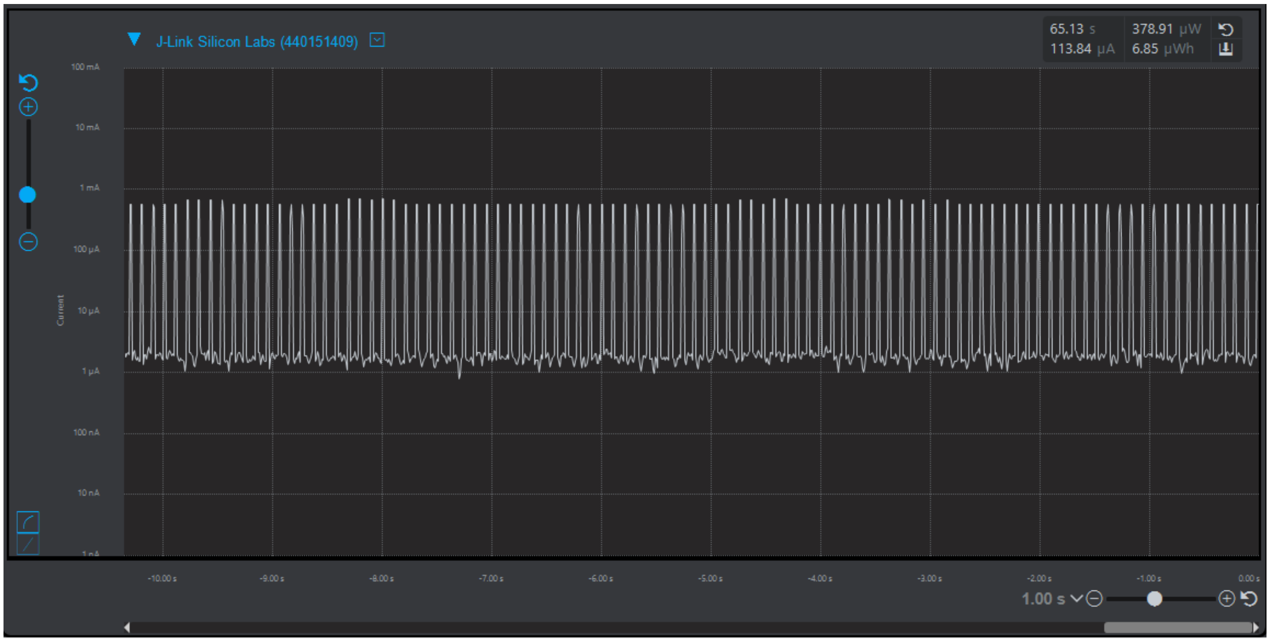


Figure 1: Default SOC-Empty Project

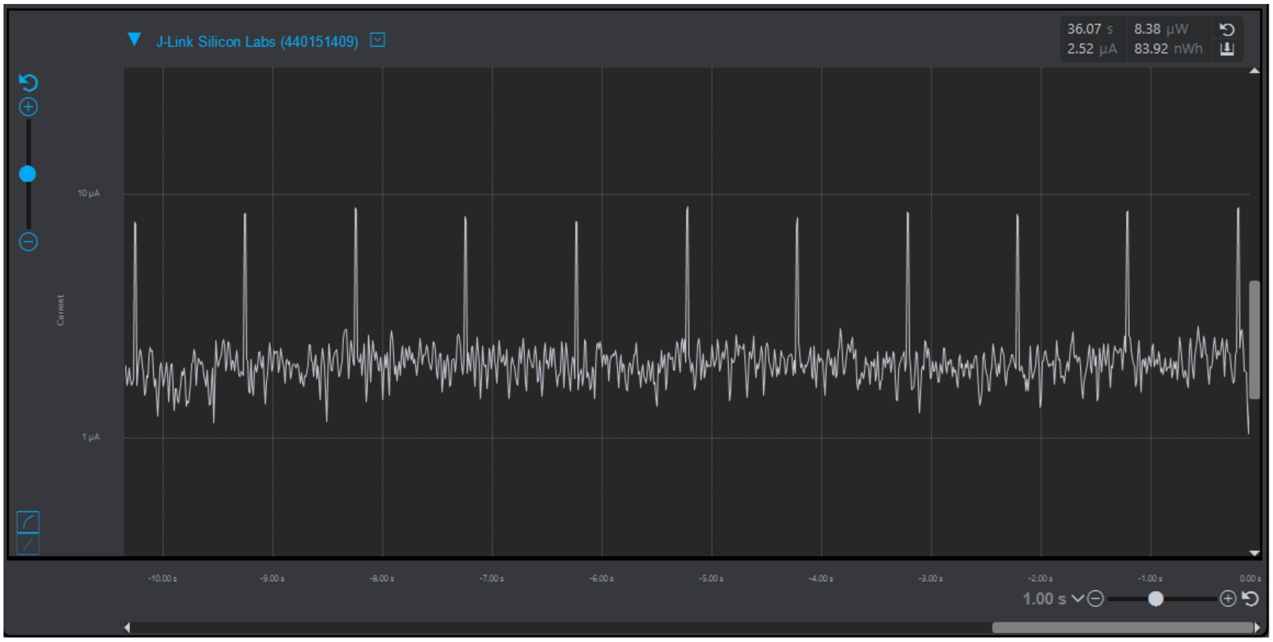


Figure 2: Modified Low-Power Beacon Project

3 Creating a Low-Power Thermometer

In this section of the lab, we will configure the BG22 for connection-based operation and use the built-in temperature sensor to transmit temperature data. Since we will be working with the same project as we did for the low-power beacon, there are a few changes required to allow the device to be connectable.

3.1 Modify the Project to Enable Connections

Beginning from the code created in Section 2 of this lab (above), the following changes should be made to app.c:

1. Reconfigure the device to advertise on all 3 advertising channels by modifying the advertise channel map

```
        /* Set adv on all 3 channels */
        gecko_cmd_le_gap_set_advertise_channel_map(0, 7);
```

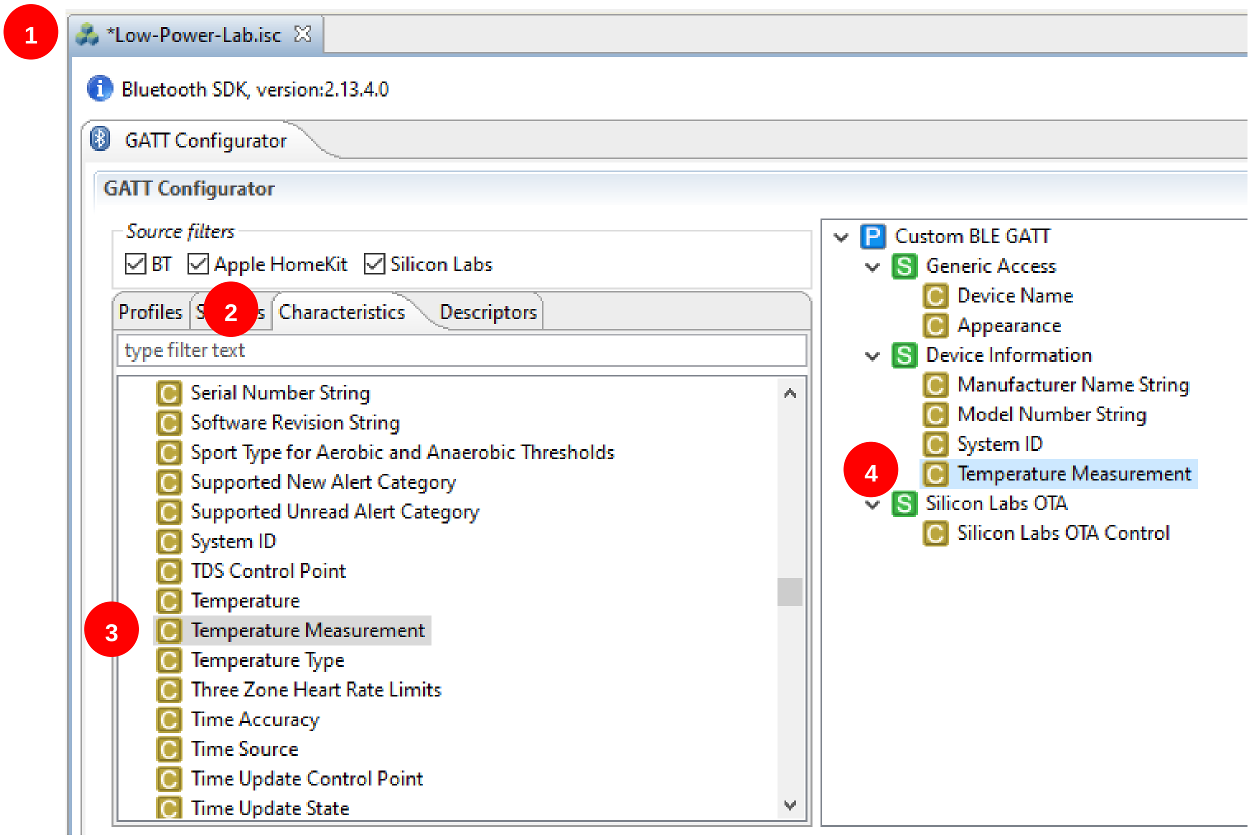
2. Configure the device to advertise as discoverable and connectable by modifying the start advertising command

```
/* Start advertising and enable connections. */
gecko_cmd_le_gap_start_advertising(0,le_gap_general_discoverable,
le_gap_connectable_scannable);
```

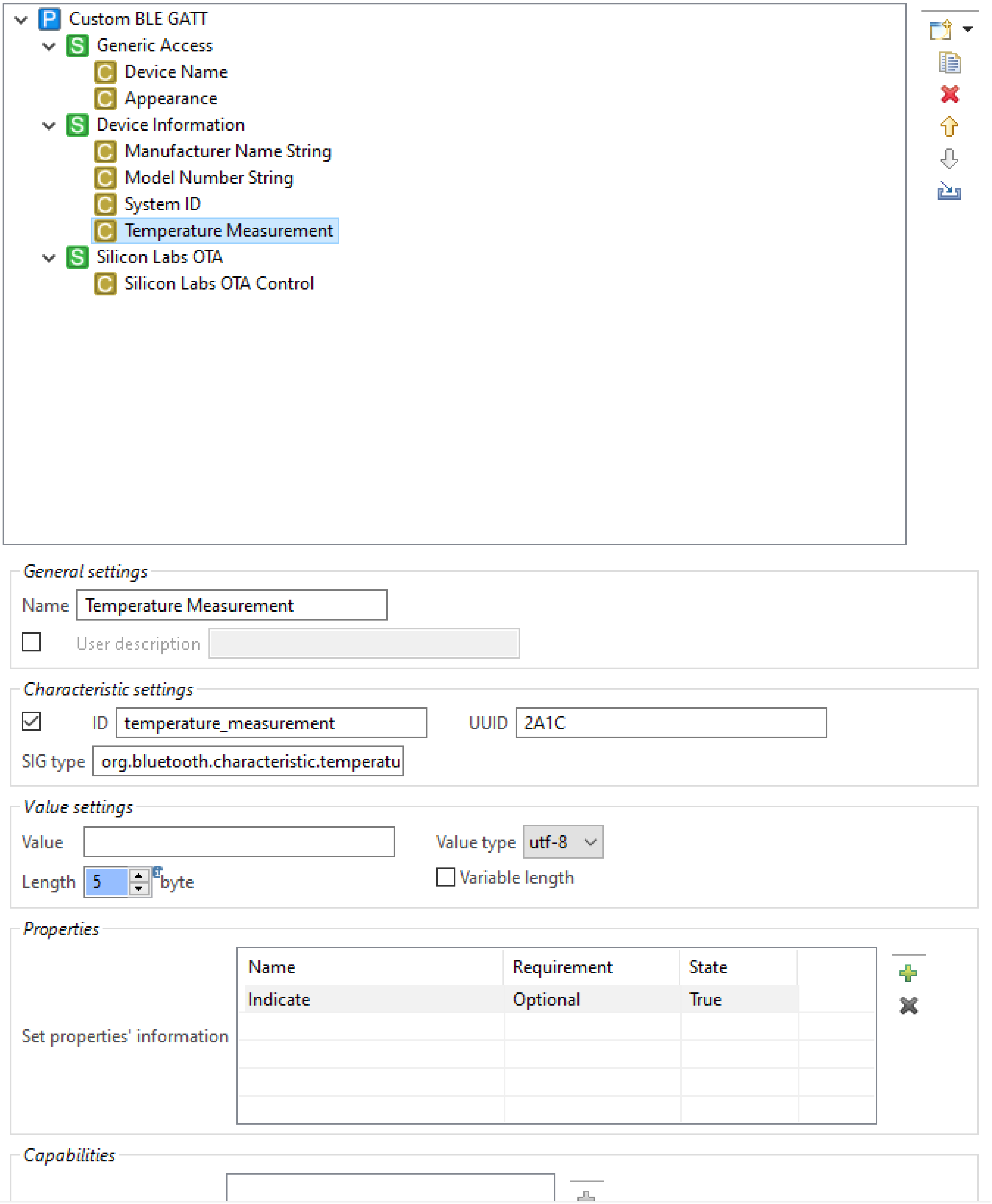
3.2 Adding a Temperature Measurement Characteristic

In this section, we will add a temperature measurement characteristic to the device GATT. The GATT Configurator in Simplicity Studio makes it easy to modify the characteristics within a Bluetooth device.

1. Open the **Low-Power-Lab.isc file**
2. In the GATT Configurator select the **Characteristics** tab
3. Locate the **Temperature Measurement** characteristc
4. Add the characteristic to your GATT by clicking and dragging it over to the **Device Information** service



5. Modify the Temperature Measurement characteristic settings to match the following



- P Custom BLE GATT
  - S Generic Access
    - C Device Name
    - C Appearance
  - S Device Information
    - C Manufacturer Name String
    - C Model Number String
    - C System ID
    - C Temperature Measurement
  - S Silicon Labs OTA
    - C Silicon Labs OTA Control

General settings

Name: Temperature Measurement

□ User description

Characteristic settings

☑ ID: temperature_measurement   UUID: 2A1C

SIG type: org.bluetooth.characteristic.temperatu

Value settings

Value: Value type: utf-8

Length: 5 byte   □ Variable length

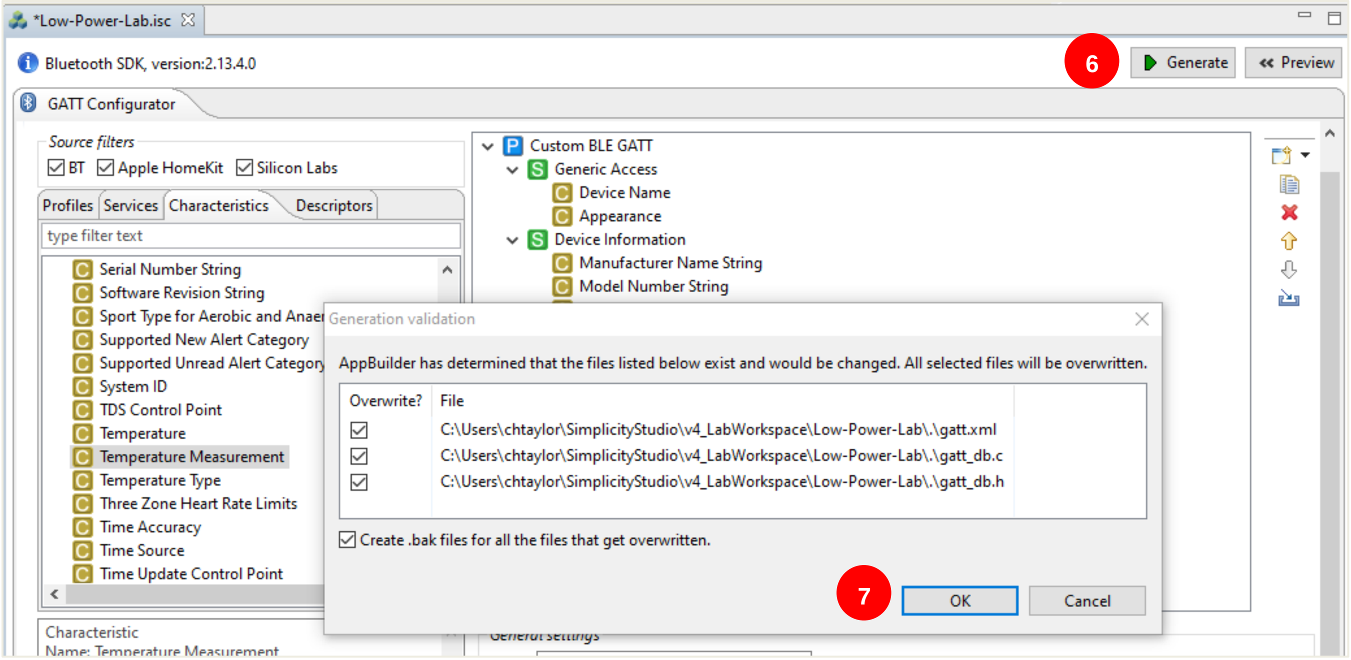
Properties

Set properties' information

Name	Requirement	State
Indicate	Optional	True

Capabilities

6. Click **Generate** in the upper-right corner
7. Click **OK** to acknowledge that the selected GATT files will be overwritten



Generation validation

AppBuilder has determined that the files listed below exist and would be changed. All selected files will be overwritten.

Overwrite?	File
☑	C:\Users\chtaylor\SimplicityStudio\v4_LabWorkspace\Low-Power-Lab\.\gatt.xml
☑	C:\Users\chtaylor\SimplicityStudio\v4_LabWorkspace\Low-Power-Lab\.\gatt_db.c
☑	C:\Users\chtaylor\SimplicityStudio\v4_LabWorkspace\Low-Power-Lab\.\gatt_db.h

☑ Create .bak files for all the files that get overwritten.

3.3 Configuring the Internal Temperature Sensor

1. Open **app.c** and navigate to include files near the top. Beginning at **Line 28**, add the following code:

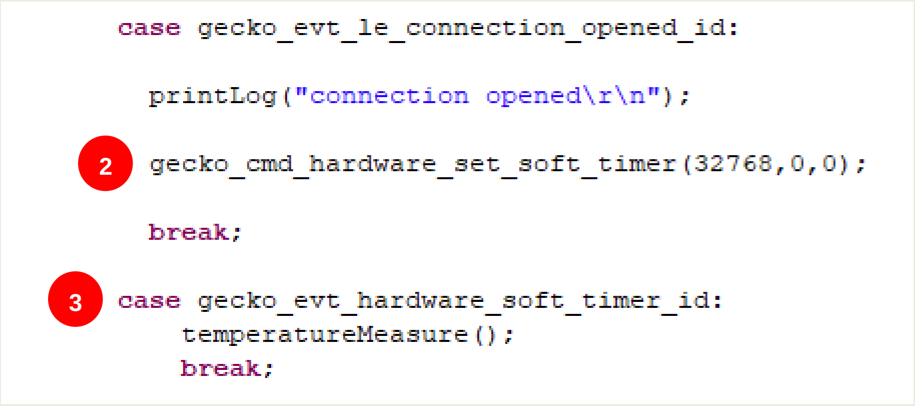
```
#include "em_emu.h"
#include "infrastructure.h"

void temperatureMeasure()
{
  uint8_t htmTempBuffer[5]; /* Stores temp data in the Health Thermometer (HTM) format. */
  uint8_t flags = 0x00;  /* HTM flags set as 0 for Celsius, no time stamp or temp type. */
  uint32_t temperature;    /* Stores temp data read from the sensor in the correct format */
  uint8_t *p = htmTempBuffer; /* Pointer to buffer needed to convert values to bitstream. */

  /* Convert flags to bitstream and append them in HTM temp data buffer (htmTempBuffer) */
  UINT8_TO_BITSTREAM(p, flags);

  /* Convert sensor data to correct temperature format */
  temperature = FLT_TO_UINT32(EMU_TemperatureGet()*10, -1);

  /* Convert temp to bitstream and place it in the HTM temp data buffer (htmTempBuffer) */
  UINT32_TO_BITSTREAM(p, temperature);

  /* Send indication of the temperature in htmTempBuffer to all "listening" clients.
   * This enables the Health Thermometer in the Blue Gecko app to display the temperature.
   *  0xFF as connection ID will send indications to all connections. */
  gecko_cmd_gatt_server_send_characteristic_notification(
    0xFF, gattdb_temperature_measurement, 5, htmTempBuffer);
}
```

2. Navigate to the connection_opened event handler (near line 100) and the following to initialize a software timer with an interval of 1 second

```
    case gecko_evt_le_connection_opened_id:

      printLog("connection opened\r\n");

      gecko_cmd_hardware_set_soft_timer(32768,0,0);

      break;
```

3. Immediately following the connection_opened event handler, add a handler for the software timer and set it to trigger the temperatureMeasure() function every second

```
    case gecko_evt_hardware_soft_timer_id:
      temperatureMeasure();
      break;
```



```
    case gecko_evt_le_connection_opened_id:

      printLog("connection opened\r\n");

      gecko_cmd_hardware_set_soft_timer(32768,0,0);

      break;

    case gecko_evt_hardware_soft_timer_id:
        temperatureMeasure();
        break;
```

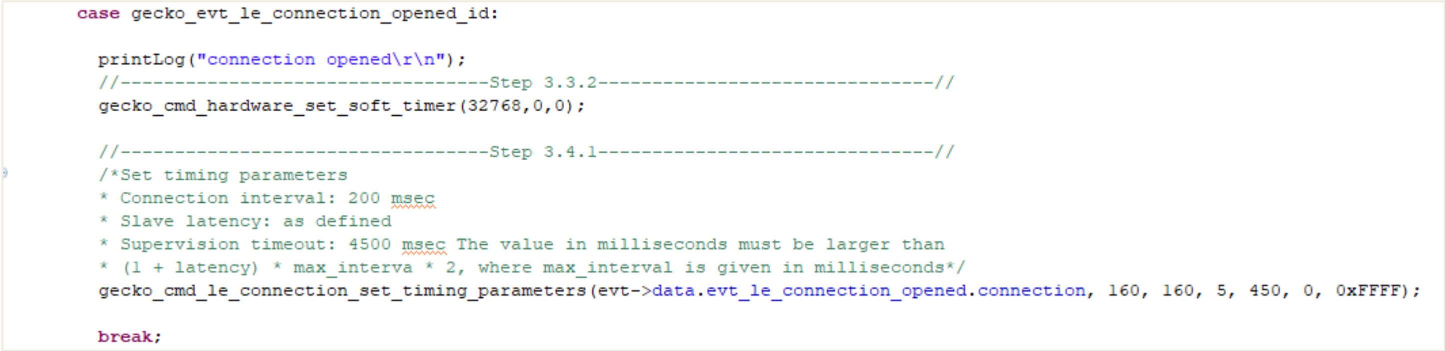
3.4 Power Optimizing Connection Parameters

Because the temperature sensor is configured to measure and update once every second, we can request a longer connection interval and add slave latency to lessen the wake-up frequency and on-time of the radio.

1. In the event **gecko_evt_le_connection_opened_id:**, add the following command to set the timing parameters to increase the connection interval to 200 ms and add a slave latency of 5 intervals:

```
/*Set timing parameters
 * Connection interval: 200 msec
 * Slave latency: as defined
 * Supervision timeout: 4500 msec The value in milliseconds must be larger than
 * (1 + latency) * max_interva * 2, where max_interval is given in milliseconds
*/
gecko_cmd_le_connection_set_timing_parameters(evt->data.evt_le_connection_opened.connection,
160, 160, 5, 450, 0, 0xFFFF);
```

The finished connection_opened event should look like this:

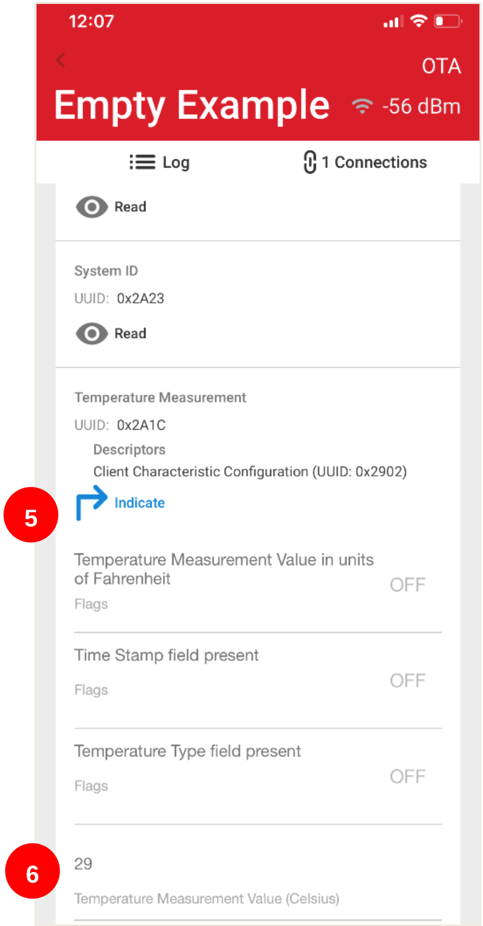


```
    case gecko_evt_le_connection_opened_id:

      printLog("connection opened\r\n");
      //----------------------------------Step 3.3.2-------------------------------//
      gecko_cmd_hardware_set_soft_timer(32768,0,0);

      //----------------------------------Step 3.4.1-------------------------------//
      /*Set timing parameters
      * Connection interval: 200 msec
      * Slave latency: as defined
      * Supervision timeout: 4500 msec The value in milliseconds must be larger than
      * (1 + latency) * max_interva * 2, where max_interval is given in milliseconds*/
      gecko_cmd_le_connection_set_timing_parameters(evt->data.evt_le_connection_opened.connection, 160, 160, 5, 450, 0, 0xFFFF);

      break;
```

2. With the project selected, click the **Build** icon (🔨) in the toolbar or right-click on the project and select **Build Project**
3. Once the project has built successfully, open the **Binaries** folder
4. Right-click the **Low-Power-Lab.s37** file and select **Flash to Device**

3.5 View the Temperature Readings in EFR Connect

1. Open the **EFR Connect** app on your mobile device
2. Select the **Develop** tab and open the **Browser**
3. Locate your device and **Connect** (the default device name is ‘Empty Example)
4. Open the **Device Information Service**
5. Click **Indicate** on the **Temperature Measurement** characteristic (otherwise the measurements will not be displayed)
6. View the temperature in the Temperature Measurement Value, rounded to the nearest degree Celsius



Empty Example -56 dBm

System ID
UUID: 0x2A23

Temperature Measurement
UUID: 0x2A1C
Descriptors
Client Characteristic Configuration (UUID: 0x2902)

Temperature Measurement Value in units of Fahrenheit OFF
Flags

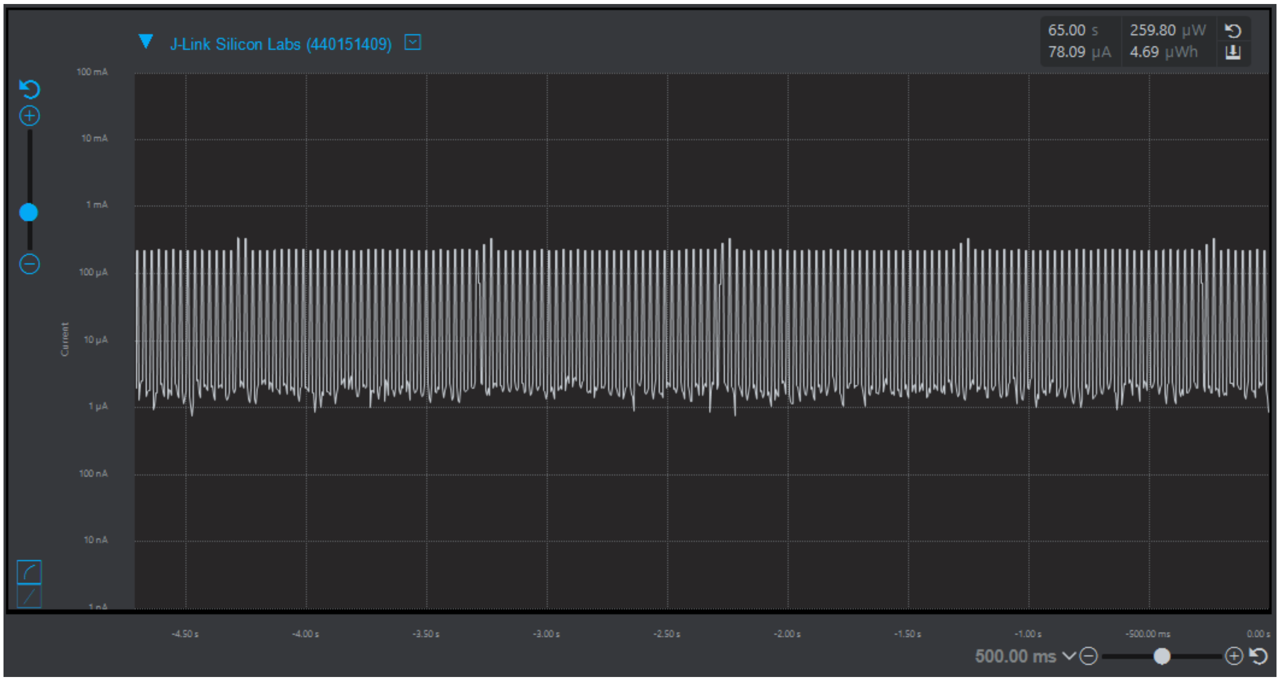
Time Stamp field present OFF
Flags

Temperature Type field present OFF
Flags

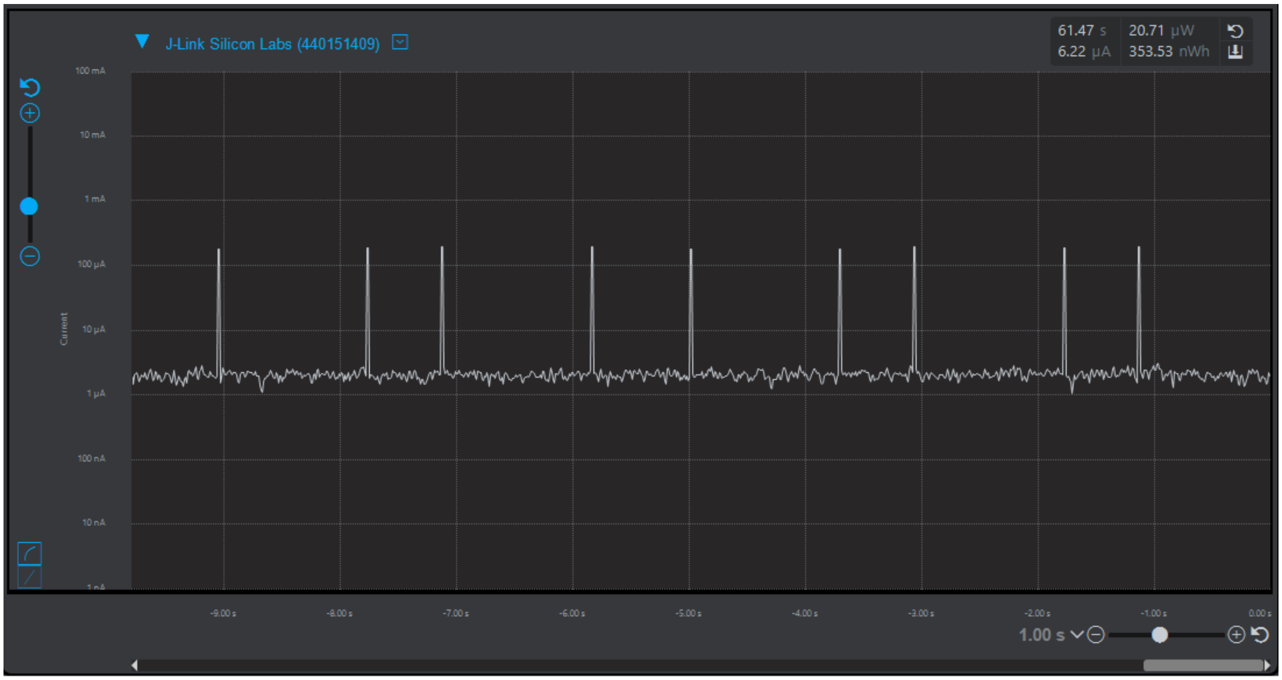
29
Temperature Measurement Value (Celsius)

3.6 Energy Profiler (Hardware Not Included)

With the thermometer characteristic configured and using the default connection parameters, the averaged current consumption of the application is measured to be about 78uA.



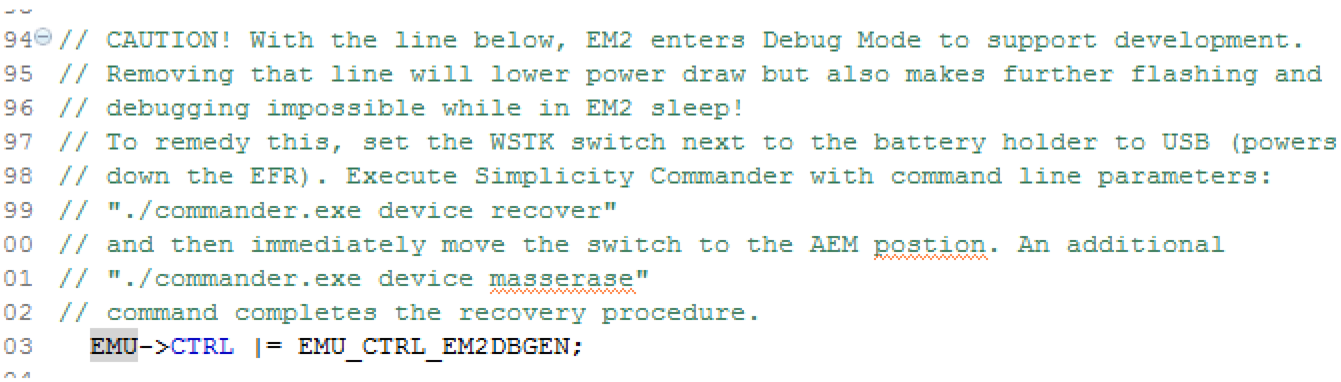
After changing the connection interval and adding slave latency, the averaged current consumption is measured to be < 6.5 µA.



4 Backup

4.1 Disable EM2 Debug Mode

For this demonstration, we left debug mode enabled in EM2 (deep sleep). However, in a production application this would likely be disabled. If you were to remove this line from init_mcu, you would get a further reduction of more than .5uA. Note though that disabling debug mode in EM2 would then require you to recover the development board per the instructions below. For development, modifying this setting is not recommended.



```
// CAUTION! With the line below, EM2 enters Debug Mode to support development.
// Removing that line will lower power draw but also makes further flashing and
// debugging impossible while in EM2 sleep!
// To remedy this, set the WSTK switch next to the battery holder to USB (powers
// down the EFR). Execute Simplicity Commander with command line parameters:
// "./commander.exe device recover"
// and then immediately move the switch to the AEM postion. An additional
// "./commander.exe device masserase"
// command completes the recovery procedure.
  EMU->CTRL |= EMU_CTRL_EM2DBGEN;
```

5 Appendix

5.1 Low Power Beacon

```
/***************************************************************************//**
 * @file app.c
 * @brief Silicon Labs Empty Example Project
 *
 * This example demonstrates the bare minimum needed for a Blue Gecko C application
 * that allows Over-the-Air Device Firmware Upgrading (OTA DFU). The application
 * starts advertising after boot and restarts advertising after a connection is closed.
 *******************************************************************************
 * # License
 * <b>Copyright 2018 Silicon Laboratories Inc. www.silabs.com</b>
 *******************************************************************************
 *
 * The licensor of this software is Silicon Laboratories Inc. Your use of this
 * software is governed by the terms of Silicon Labs Master Software License
 * Agreement (MSLA) available at
 * www.silabs.com/about-us/legal/master-software-license-agreement. This
 * software is distributed to you in Source Code format and is governed by the
 * sections of the MSLA applicable to Source Code.
 *
 ******************************************************************************/

/* Bluetooth stack headers */
#include "bg_types.h"
#include "native_gecko.h"
#include "gatt_db.h"

#include "app.h"

/* Print boot message */
static void bootMessage(struct gecko_msg_system_boot_evt_t *bootevt);

/* Flag for indicating DFU Reset must be performed */
static uint8_t boot_to_dfu = 0;

/* Main application */
void appMain(gecko_configuration_t *pconfig)
{
#if DISABLE_SLEEP > 0
  pconfig->sleep.flags = 0;
#endif

  /* Initialize debug prints. Note: debug prints are off by default. See DEBUG_LEVEL in
app.h */
  initLog();

  /* Initialize stack */
  gecko_init(pconfig);

  while (1) {
    /* Event pointer for handling events */
    struct gecko_cmd_packet* evt;

    /* if there are no events pending then the next call to gecko_wait_event() may cause
```

```
     * device go to deep sleep. Make sure that debug prints are flushed before going to
sleep */
    if (!gecko_event_pending()) {
      flushLog();
    }

    /* Check for stack event. This is a blocking event listener. If you want non-blocking
please see UG136. */
    evt = gecko_wait_event();

    /* Handle events */
    switch (BGLIB_MSG_ID(evt->header)) {
      /* This boot event is generated when the system boots up after reset.
       * Do not call any stack commands before receiving the boot event.
       * Here the system is set to start advertising immediately after boot procedure. */
      case gecko_evt_system_boot_id:

        bootMessage(&(evt->data.evt_system_boot));
        printLog("boot event - starting advertising\r\n");

        /* Set tx power to 0 dBm */
        gecko_cmd_system_set_tx_power(0);

        /* Set adv on channel 37 only */
        gecko_cmd_le_gap_set_advertise_channel_map(0, 1);

        /* Set the device name to BG22 */
        uint8_t name[] = {5,9,'B','G','2','2'};
        gecko_cmd_le_gap_bt5_set_adv_data(0, 0, sizeof(name), name);

        /* Set advertising parameters. 1000ms advertisement interval.
         * The first parameter is advertising set handle
         * The next two parameters are min and max advertising interval, both in
         * units of (milliseconds * 1.6).
         * The last two parameters are duration and maxevents left as default. */
        gecko_cmd_le_gap_set_advertise_timing(0, 1600, 1600, 0, 0);

        /* Start advertising and disable connections. */
        gecko_cmd_le_gap_start_advertising(0,le_gap_user_data, le_gap_non_connectable);

        break;

      case gecko_evt_le_connection_opened_id:

        printLog("connection opened\r\n");

        break;

      case gecko_evt_le_connection_closed_id:

        printLog("connection closed, reason: 0x%2.2x\r\n", evt->data.evt_le_connection_closed.reason);

        /* Check if need to boot to OTA DFU mode */
        if (boot_to_dfu) {
          /* Enter to OTA DFU mode */
          gecko_cmd_system_reset(2);
        } else {
          /* Restart advertising after client has disconnected */
          gecko_cmd_le_gap_start_advertising(0, le_gap_general_discoverable, le_gap_con-
nectable_scannable);
        }
        break;

      /* Events related to OTA upgrading
         ----------------------------------------------------------------------------- */

      /* Check if the user-type OTA Control Characteristic was written.
       * If ota_control was written, boot the device into Device Firmware Upgrade (DFU)
mode. */
      case gecko_evt_gatt_server_user_write_request_id:

        if (evt->data.evt_gatt_server_user_write_request.characteristic ==
gattdb_ota_control) {
          /* Set flag to enter to OTA mode */
          boot_to_dfu = 1;
          /* Send response to Write Request */
          gecko_cmd_gatt_server_send_user_write_response(
            evt->data.evt_gatt_server_user_write_request.connection,
            gattdb_ota_control,
            bg_err_success);

          /* Close connection to enter to DFU OTA mode */
          gecko_cmd_le_connection_close(evt->data.evt_gatt_server_user_write_request.con-
nection);
        }
        break;

      /* Add additional event handlers as your application requires */

      default:
        break;
    }
  }
}

/* Print stack version and local Bluetooth address as boot message */
static void bootMessage(struct gecko_msg_system_boot_evt_t *bootevt)
{
#if DEBUG_LEVEL
  bd_addr local_addr;
  int i;

  printLog("stack version: %u.%u.%u\r\n", bootevt->major, bootevt->minor, bootevt-
>patch);
  local_addr = gecko_cmd_system_get_bt_address()->address;

  printLog("local BT device address: ");
  for (i = 0; i < 5; i++) {
    printLog("%2.2x:", local_addr.addr[5 - i]);
  }
  printLog("%2.2x\r\n", local_addr.addr[0]);
#endif
}
```

5.2 Low Power Thermometer

```
/***************************************************************************//**
 * @file app.c
 * @brief Silicon Labs Empty Example Project
 *
 * This example demonstrates the bare minimum needed for a Blue Gecko C application
 * that allows Over-the-Air Device Firmware Upgrading (OTA DFU). The application
 * starts advertising after boot and restarts advertising after a connection is closed.
 *******************************************************************************
 * # License
 * <b>Copyright 2018 Silicon Laboratories Inc. www.silabs.com</b>
 *******************************************************************************
 *
 * The licensor of this software is Silicon Laboratories Inc. Your use of this
 * software is governed by the terms of Silicon Labs Master Software License
 * Agreement (MSLA) available at
 * www.silabs.com/about-us/legal/master-software-license-agreement. This
 * software is distributed to you in Source Code format and is governed by the
 * sections of the MSLA applicable to Source Code.
 *
 ******************************************************************************/

/* Bluetooth stack headers */
#include "bg_types.h"
#include "native_gecko.h"
#include "gatt_db.h"

#include "app.h"

#include "em_emu.h"
#include "infrastructure.h"

void temperatureMeasure()
{
  uint8_t htmTempBuffer[5]; /* Stores temp data in the Health Thermometer (HTM) format.
*/
  uint8_t flags = 0x00; /* HTM flags set as 0 for Celsius, no time stamp or temp type. */
  uint32_t temperature;   /* Stores temp data read from the sensor in the correct format
*/
  uint8_t *p = htmTempBuffer; /* Pointer to buffer needed to convert values to bitstream.
*/

  /* Convert flags to bitstream and append them in HTM temp data buffer (htmTempBuffer)
*/
  UINT8_TO_BITSTREAM(p, flags);

  /* Convert sensor data to correct temperature format */
  temperature = FLT_TO_UINT32(EMU_TemperatureGet()*10, -1);

  /* Convert temp to bitstream and place it in the HTM temp data buffer (htmTempBuffer)
*/
  UINT32_TO_BITSTREAM(p, temperature);

  /* Send indication of the temperature in htmTempBuffer to all "listening" clients.
   * This enables the Health Thermometer in the Blue Gecko app to display the tempera-
ture.
   *  0xFF as connection ID will send indications to all connections. */
  gecko_cmd_gatt_server_send_characteristic_notification(
    0xFF, gattdb_temperature_measurement, 5, htmTempBuffer);
}

/* Print boot message */
static void bootMessage(struct gecko_msg_system_boot_evt_t *bootevt);

/* Flag for indicating DFU Reset must be performed */
static uint8_t boot_to_dfu = 0;

/* Main application */
void appMain(gecko_configuration_t *pconfig)
{
#if DISABLE_SLEEP > 0
  pconfig->sleep.flags = 0;
#endif

  /* Initialize debug prints. Note: debug prints are off by default. See DEBUG_LEVEL in
app.h */
  initLog();

  /* Initialize stack */
  gecko_init(pconfig);

  while (1) {
    /* Event pointer for handling events */
    struct gecko_cmd_packet* evt;

    /* if there are no events pending then the next call to gecko_wait_event() may cause
     * device go to deep sleep. Make sure that debug prints are flushed before going to
sleep */
    if (!gecko_event_pending()) {
      flushLog();
    }

    /* Check for stack event. This is a blocking event listener. If you want non-blocking
please see UG136. */
    evt = gecko_wait_event();

    /* Handle events */
    switch (BGLIB_MSG_ID(evt->header)) {
      /* This boot event is generated when the system boots up after reset.
       * Do not call any stack commands before receiving the boot event.
       * Here the system is set to start advertising immediately after boot procedure. */
      case gecko_evt_system_boot_id:

        bootMessage(&(evt->data.evt_system_boot));
        printLog("boot event - starting advertising\r\n");

        /* Set tx power to 0 dBm */
        gecko_cmd_system_set_tx_power(0);

        /* Set adv on channel 37 only */
        //gecko_cmd_le_gap_set_advertise_channel_map(0, 1);

        /* Set the device name to BG22 */
        uint8_t name[] = {5,9,'B','G','2','2'};
        gecko_cmd_le_gap_bt5_set_adv_data(0, 0, sizeof(name), name);

        /* Set advertising parameters. 1000ms advertisement interval.
     * The first parameter is advertising set handle
     * The next two parameters are min and max advertising interval, both in
     * units of (milliseconds * 1.6).
     * The last two parameters are duration and maxevents left as default. */
    gecko_cmd_le_gap_set_advertise_timing(0, 1600, 1600, 0, 0);

    /* Start general advertising and enable connections. */
    gecko_cmd_le_gap_start_advertising(0,le_gap_general_discoverable, le_gap_connect-
able_scannable);

    break;

   case gecko_evt_le_connection_opened_id:

    printLog("connection opened\r\n");

    gecko_cmd_hardware_set_soft_timer(32768,0,0);
    gecko_cmd_le_connection_set_timing_parameters(evt->data.evt_le_connec-
tion_opened.connection, 160, 160, 5, 450, 0, 0xFFFF);

    break;

   case gecko_evt_hardware_soft_timer_id:

    temperatureMeasure();

   break;

   case gecko_evt_le_connection_closed_id:

    printLog("connection closed, reason: 0x%2.2x\r\n", evt->data.evt_le_connec-
tion_closed.reason);

    /* Check if need to boot to OTA DFU mode */
    if (boot_to_dfu) {
      /* Enter to OTA DFU mode */
      gecko_cmd_system_reset(2);
    } else {
      /* Restart advertising after client has disconnected */
      gecko_cmd_le_gap_start_advertising(0, le_gap_general_discoverable, le_gap_con-
nectable_scannable);
    }
    break;

   /* Events related to OTA upgrading
     ----------------------------------------------------------------------------- */

   /* Check if the user-type OTA Control Characteristic was written.
    * If ota_control was written, boot the device into Device Firmware Upgrade (DFU)
mode. */
   case gecko_evt_gatt_server_user_write_request_id:

    if (evt->data.evt_gatt_server_user_write_request.characteristic ==
gattdb_ota_control) {
      /* Set flag to enter to OTA mode */
      boot_to_dfu = 1;
      /* Send response to Write Request */
      gecko_cmd_gatt_server_send_user_write_response(
        evt->data.evt_gatt_server_user_write_request.connection,
      gattdb_ota_control,
      bg_err_success);

     /* Close connection to enter to DFU OTA mode */
     gecko_cmd_le_connection_close(evt->data.evt_gatt_server_user_write_request.connection);
    }
    break;

   /* Add additional event handlers as your application requires */

   default:
    break;
  }
 }
}

/* Print stack version and local Bluetooth address as boot message */
static void bootMessage(struct gecko_msg_system_boot_evt_t *bootevt)
{
#if DEBUG_LEVEL
 bd_addr local_addr;
 int i;

 printLog("stack version: %u.%u.%u\r\n", bootevt->major, bootevt->minor, bootevt->patch);
 local_addr = gecko_cmd_system_get_bt_address()->address;

 printLog("local BT device address: ");
 for (i = 0; i < 5; i++) {
  printLog("%2.2x:", local_addr.addr[5 - i]);
 }
 printLog("%2.2x\r\n", local_addr.addr[0]);
#endif

}
```